ITEA-Project:	COMPACT
ITEA Reference Number:	16018
Funding Organizations:	Austrian Research Promotion Agency FFG (project number 863828 and 864769)
	Finnish funding agency for innovation Business Finland (Diary Number 3098/31/2017)
	German ministry of education and research (BMBF) (reference number: 01IS17028)
Project Duration:	01.09.2017 until 31.12.2020
Task:	T1.4 Technical project baseline and ITEA living roadmap
	T5.1 Demonstrators

Deliverable

D1.5 Final COMPACT contribution to ITEA Living Roadmap

D5.3 State-of-the-Art Update

Deliverable Type:	Report
Dissemination Level:	Public
Due Date:	31.12.2020
Date of Creation:	15.12.2020
Involved Partners:	ABIX GmbH (ABI) Eberhard Karls Universität Tübingen (EKUT) FZI Forschungszentrum Informatik (FZI) Infineon Technologies AG (IFX) Kasper & Oswald GmbH (KAOS) Noiseless Imaging Oy (NI) OFFIS (OFF) Robert Bosch GmbH (RB) SparxSystems CE (SSCE) Tampere University (TAU) Technische Universität München (TUM) Universität Paderborn (UPB) Visy Oy (VIS)
Deliverable Lead Partner	Universität Paderborn (UPB)

Content

1 Introduction

The Internet-of-things (IoT) has the potential to improve our lives dramatically. The backbone of industry automation, smarter homes, higher energy efficiency, better health care, assistance of elderly people and more flexibility in working environments are only some areas that can be imagined today and realized tomorrow. The tremendous impact of IoT on our industrial environments and our private life is a key reason to consider IoT research and developments as important pillars in the European Horizon 2020. Impact to our private life are, for instance, in home automation via so-called IoT edge devices like smart light bulbs which are expected to come almost at the same costs as non IoT enabled devices in near future. Another rapidly evolving market is in industry automation (Industrial IoT), which is expected to grow dramatically for the next decade pushed by several initiatives like the German Industry 4.0 strategy.

IoT devices with sensors and actuators need electronics to connect that world of “things” with the digital world of the Internet. Yet software runs the IoT electronic device hardware. Since IoT devices need to be smart, cheap and capable to run with extremely small amounts of energy – known as ultra-thin IoT nodes – IoT software must also be ultra-thin with extremely small memory footprints and ultra-low energy consumption. At the same time, software must provide smart functions including real-time computing capabilities, connectivity, security, safety, and remote update mechanisms. Recent publications support the claim of ultra-thin – and low cost – IoT devices. So, for instance, Walmart calls for sub-1$ IoT sensors [80] and ARM TechCON targets at sub-50cent IoT SoCs [81].

These constraints put a high pressure on IoT software development. Due to the very limited resources provided by IoT nodes, today’s commonly used design approach to trade-off development time with software efficiency is not competitive any longer. Therefore, an industry-wide effort in the course of the COMPACT project provided novel solutions for the application-specific and customer-oriented realization of ultra-thin IoT nodes with focus on software generation for IoT devices with ultra-small memory footprints and ultra-low power consumption. To obtain this goal, COMPACT created innovations to automate the software development and configuration flow for ultra-constrained IoT nodes. The automation methodology followed the OMG notion of model-driven architecture (MDA) and applies it to the development of IoT device software. Due to the main principles of MDA, COMPACT followed a scalable approach using carefully designed meta-models and generators for auto generating the required software as its key concept.

The next chapter provides the current state-of-the-art in areas which were related to the research and development activities of the COMPACT project.

2 State-of-the-Art Analysis

2.1 Model-Driven Design

The Object Management Group (OMG), which adopted the UML as a standard, has developed a meta-modelling architecture to define the UML. This meta-modelling architecture enables the extension of the UML with so-called UML profiles. The standard UML profile for Modelling and Analysis of Real-Time and Embedded systems (MARTE) extends the UML with key notations (non-functional properties such as time and resources) for real-time computing. MARTE extends UML with the capability to model the hardware and software layers (middleware) and interconnections that compose an execution platform. Platform components can be described at the same level of abstraction as the application, and they may thus also contain timing information along with structural and behavioural aspects. The allocation model describes the mapping of application functions onto execution platform resources. This enables the optimisation of the software with regard to scheduling. However, for optimisation of the embedded software, a more general specification of the hardware base line is required, and system level approaches are still insufficiently supported.

The Systems Modelling Language (SysML) on the other hand extends UML to offer support for modelling hardware (mechanical and electrical). The SysML extends the UML, to model hardware aspects such as parallelism, critical regions, or requirement modelling. The language is often used in the domain of systems engineering. It widens the UML focus towards software and enables the modelling of the complete system. It uses blocks to model the parts of the system, instead of classes as part of the software. On basis of the system view, SysML provides an interesting part in modelling the information basis for the generation of ultra-thin software. However, it still lacks aspects such as variability or a detailed conditional architectural specification.

Other UML profiles exist such as EAST-ADL2 or UPDM. All the UML profiles focus on dedicated system or software aspect. In the effort to create a comprehensive data basis for the generation of ultra-thin software for IoT devices these approaches have to be unified, and the required parts of each approach identified respectively.

State-of-the-art technology available in a powerful modelling tool like Enterprise Architect by SparxSystems (an OMG member) allows the implementation of a new modelling profile, approach and best practices that adheres to desired standards while at the same time extending capabilities beyond the current confines of UML to address IoT-specific requirements and effectively merge the heretofore segregated evolution of software, hardware and firmware.

When UML comes with code generation, it is typically based on the principles of the Model-Driven Architecture (MDA), which has provided significant advances in the design and analysis of embedded software throughout the last years [72]. By providing a domain-specific profile or meta-model, instead of the source code, MDA offers the benefit to support different generation processes and, therefore, to easily adjust to different scenarios or hardware base lines. The influence of the underlying hardware baseline is crucial for the quality of the design and the fulfilment of requirements in a model-based design process for embedded software, especially if a resource efficient ultra-thin design is targeted. Currently, Model-Driven Architecture (MDA) is applied to different, vertical domains based on different meta-models, UML profiles, languages, middleware and operating systems.

There has been various work and projects to bring model-driven architecture (MDA) principles to the IoT domain. One MDA tool is described in [30]. It offers a graphical domain-specific language for design entry and modelling and code generation facilities for IoT standards. It combines the different domain views on IoT communication, things and IDs as well as the processing of vast amounts of data. Another work in [31] describes a model-driven IoT flow focusing mainly on communication. They propose to specify functionality with IoT-specific DSL formats and map the specification on an automata-based platform independent meta-model. Code generators generate the platform-dependent code. The MDA tool proposed in [32] uses state transition diagrams as programming model for design and modelling IoT applications in a platform-independent way.

Code generators produce the platform dependent code. For commercial solutions, code generation tools and the MDE design flow from MATLAB/Simulink is widely adopted in industry. [73] introduces a SysML profile that addresses heterogeneity within a system of IoT devices and enables model-driven development of IoT applications. The profile is based on the IoT domain reference model introduced by the IoT-A project [74]. [75] introduces a model-based design process using the BIP component language [76] for the analysis of REST-based web services that integrate IoT devices. However, all do not fit the specific needs of very resource-constrained ultra-thin hardware platforms and the generation of efficient application software for such platforms.

Within the IoT domain, also variability of software and hardware platforms needs to be considered. A large number of variability modeling languages exists for different domains. Approaches like the Common Variability Language (CVL) [77], or feature diagrams, which have been introduced as part of the Feature-Oriented Domain Analysis (FODA) method [78], use graphical notations. Textual variability languages, such as Clafer [79], the INDENICA Variability Modeling Language (IVML) [80], or the Textual Variability Specification Language (VSL) [81] promise improved scalability with respect to model size and complexity over purely graphical variability modeling approaches.

Besides UML based approaches, other modelling approaches exist used by the industry. IP-XACT [67] is Accellera Systems Initiative's specification for documenting Intellectual Property (IPs) hardware designs. It enables highly automated design creation and configuration in a tool independent and machine-readable manner. Especially in the automotive domain the structural view at the hardware baseline level is needed to allow the automated import of IP-XACT components as well as the automated generation of virtual prototypes including multicore processors, peripheral devices, and reused IP blocks. While this is a detailed specification of the interfaces of IP and structural view of an IP-based design, it only contains very abstract inclusion of software aspects. Kactus2 [68] is an Open Source EDA tool for creating and editing IP-XACT designs. Kactus2 strictly follows the IP-XACT standard and provides RTL synthesis for targeting FPGA and ASIC designs.

ThingML [69] has been applied as a modelling language for embedded and distributed devices. ThingML focuses on the Internet of Things [8] domain and targets resource constrained embedded systems such as low power sensor and microcontroller based devices. There exists a model-driven software engineering tool-chain for ThingML, and code generators for various devices. Another widely adopted modelling language for embedded devices is the Architecture Analysis and Design Language (AADL) [27]. AADL targets at distributed computing systems with real-time requirements. It offers a device class suitable to describe IoT devices, yet it focuses mainly on their interaction not on the implementation. Thus, it also does not offer the capabilities required for ultra-thin code generation. Another model-based language focusing on data-flow oriented design is RVC-CAL defined in ISO Standard 23001-4:2011 [28]. RVC-CAL is a high-level data flow model-based language that can be considered similar to MATLAB/SIMULINK in the sense that it needs to be compiled/synthesized to C or Verilog for implementations. An open source compiler "Orcc" exists [29] and offers code generation to a variety of platforms. Code generation approaches from RVC-CAL to resource constrained devices do exist: a very recent work [70] generates multi-threaded C code that is not dependent on any external libraries, whereas [71] generates LLVM bytecode, however intended for a particular "TTA" (Transport Triggered Architecture) type processor architecture.

This summarizes the main model-based systems engineering (MBSE) tools and trends. During the COMPACT project period, the whole MBSE domain developed further, but the key fields stayed the same. Many of the actual trends confirm the design decisions taken w.r.t. the IoT-PML. Exemplary the further development of the SysML v2 should be mentioned here. Even if the initial request for proposal (RFP) was already launched in 2014, parallel to the development of the IoT-PML, further concretization of the SysML v2 was done. One of the important improvements of the SysML should be the improvement of the interoperability. One aspect that should be achieved is the „single source of truth" for system design. A similar goal for the design of IoT devices was achieved by the IoT-PML that captures both the IoT node hardware, software stack, functional and non-functional requirements in a single model and provide therefore a single source of information for different code generators. Another similarity between the further development of SysML and IoT-PML is the emancipation against the UML. SysML v2 should contain an independent meta model, compatible with UML. The IoT-PML of the COMPACT project followed a similar development. While the reference implementation is based on the UML and implemented as UML profile, the concept of the IoT-PML is independent and can be implemented with an independent meta model.

2.2 IoT Software Compilation and Optimization

A major driving force for improvements in code size and performance are the two major compilers GCC and LLVM that can target embedded devices. Many of their advances can be applied orthogonally to the methods developed in COMPACT at no additional integration cost.

LLVM has released a few major versions since COMPACT has started [100]. However, most of the changes are internal and do not affect users significantly. A notable change is that the RISC-V target is no longer considered experimental and is therefore included in official distributions and installation packages. With this change, the developers signal that the target has reached a certain stability and reliability. The developers have also decided that preliminary extensions can be implemented in LLVM before ratification and have already added support for the current drafts of the bit-manipulation and vector extensions. Furthermore, LLVM has added a new parallelizable JIT Compiler API named “ORCv2”. They also introduced mitigation techniques against the speculative execution vulnerabilities “Spectre”.

The GNU Compiler Collection (GCC) gained a few major releases as well, but nothing of particular interest for COMPACT [101]. The changes improve language support, diagnostics, compile time and code generation.

In terms of code generation, a new C-like domain specific language for device drivers was proposed, that achieves a reduction of memory consumption, run time and development effort through register layout optimization [102].

2.3 Ultra-Low Power Software

There exist various code compression techniques to reduce the instruction memory footprint of embedded processors. The basic idea is to store a compressed version of the machine instructions in memory and decompress them during instruction fetch. In asymmetric compression methods, the code is compressed in software and decompressed by hardware. There is no need to change the CPU or compiler, but an additional HW decompression block is required that may penalize performance. The block may either be located before or behind the instruction cache, which then either holds decompressed original instructions or the compressed instructions. With this, [13] achieved 74% compression ratio for MIPS cores using Huffman codes. Using V2FCC, [14] reached 68-84% compression ratio for the VLIW processor TMS320C6x. [15] proposed dictionary-based code-compression. The basic idea is to look for the most used instruction words and encode these into the shortest code words. Decompression is relatively simple but the dictionary must be counted as additional hardware overhead. [16] describes bitmask-based code compression. It additionally exploits that symbols next to each other in the dictionary are often similar by defining a bitmask with position and difference. The field is still actively studied, e.g. [17] proposed separated dictionaries recently to improve performance and power use. For commercial solutions, e.g., for IBM’s CodePACK tool a compression ratio above 60% was reported for PowerPC [18]. Some compression techniques can also directly be implemented in the compiler and do not require hardware modifications [63]. If the hardware can be tailored to a specific piece of software, it is also possible to adjust the ISA in order to reduce redundancies [64]. Some tools have been developed to explore the code compression design space [65, 66].

There exist many ways to reduce the memory footprint by writing efficient source code and applying source-code-level best practices. These may include stringent use of *const* declaration on constants and look-up tables, careful allocation of stack memory, compressed data structures, or use of unions and bit fields. Another way is to generate the source code based on a configuration. An example is Infineon’s DAVE Tool that can generate the SW for the XMC controller family.

The arrival of artificial intelligence (AI) applications to IoT devices has brought considerable challenges to IoT device memory needs. Convolutional Neural Networks (CNNs) present the cutting-edge paradigm in AI technologies, but require by default huge amounts of memory to operate. To this end, many papers are addressing memory footprint reduction for CNN computation with CPUs or accelerators in the recent years. One of the most promising approaches is the binarization of CNNs, which reduces the memory need of a CNN to 1/30 of the original [86]. Another very specific approach utilizing generation for obtaining optimized results is described in [83]. Here, an optimized mapping of quadratic programs for embedded is described.

Another aspect is addressed in [82]. Here, the benefit of heterogeneous multi-core architectures for smaller memory footprints is addressed. To generalize these statements, digital blocks can be used to reduce memory footprint as well.

In addition to code compactness, care should be given for efficient data handling. Raw data should be processed as early as possible, computing efficiency permitting, to reduce need to transmit or store redundant information. The information could also be truncated to remove unnecessary detail, e.g., noise, above spec accuracy. When performing calculations, efficient algorithms should be used with emphasis on keeping the working set in either local scratch memory or in a write-back configured data cache, if present. This recommendation is to avoid spending energy on hitting the system memory bus.

Many small processors include data processing DSP or SIMD extensions, e.g., Xtensa HIFI extensions, ARM Cortex-M4/M33/M7 DSP extensions, or the upcoming RISC-V V extension. These usually are more optimized for data processing than regular computing engines. Using these does often require extra programming effort due to restrictions on data types or the need to invoke the intrinsics directly, because compilers lack the support to use these automatically.

Unless the data processing is time-critical, it would be prudent to reduce the clock frequency and voltage. Though this would increase the compute time, the overall energy per calculation would decrease. However, using this technique does require some hardware support.

One well known aspect of saving energy while executing embedded software with a real time schedule is to use timing slacks to either reduce the execution speed as well as the supply voltage (dynamic voltage frequency scaling) or to temporarily disable system components while not needed (dynamic power management) [99]. Another aspect to be regarded is the memory subsystem. One promising approach to improve the energy demand for embedded systems in the IoT domain even further is to optimize the memory layout so mostly the lower-power memory units a system is providing are utilized. An approach with scheduling in mind is evaluated in [98], the authors combine multicore real time application scheduling with an efficient use of local and shared memories. A few existing approaches are based on systems using scratchpad memories (SPM) [91,92,93,94,97] but a few other papers also take memory systems without scratchpads into account for optimization [95,96]. The approach in [91] considers the dynamic copy of instruction data into (SPM) while execution, while [92] focuses on an optimized mapping of data for pipelined streaming applications. In [3] the authors present a mapping scheme for code and data, which apparently performs comparable to hardware caches. An efficient mapping of data to a hybrid SPM composed of SRAM and non-volatile memory to minimize energy demand is examined in [97]. A few other approaches don't use SPM's, in [94] the authors evaluate an efficient mapping of code from flash to RAM memory. A two-dimensional problem is solved by [5] by finding a mapping of code and data to multiple banks of different size while optimizing the amount and sizes of memory banks.

2.4 IoT Security

In most IoT scenarios, ultra-thin IoT devices communicate with each other and/or some form of infrastructure, e.g., servers over the Internet. In this context, IoT security means, ultimately, the protection of data against untrusted parties. Commonly, there are three forms of data that need to be protected: data in motion, data in use, and data at rest.

While protecting data in use is an important goal on its own, achieving meaningful results on IoT devices is difficult without special processors and memories. This type of hardware is only available in very high-end solutions and thus unlikely to be used in a typical IoT device. Similarly, protecting data at rest in an IoT device has its challenges: protecting sensitive data stored in a ROM against invasive attacks requires special memories, which again adds cost. To target the largest class of IoT devices – those, which do not have extra security hardware – the focus is on protecting data in motion, i.e., data that is transmitted between IoT devices or between an IoT device and a server. To this end, two principal modes of attack are assumed: A would-be attacker intercepts or eavesdrops on data as it is transmitted to read or modify it. Alternatively, the attacker may use public communication interfaces of IoT devices or servers to try to gain knowledge of sensitive data.

Protecting data in motion against the above-mentioned attacker requires strong cryptography to assert integrity, confidentiality, and authenticity of data (or a subset of these attributes). Protecting interfaces requires ensuring that all program code that handles interactions with the public (i.e., the Internet) is free from logic and programming errors which could be exploited. While achieving bug-free software is still considered impossible, exposed code can be significantly hardened by manual/semi-automatic code reviews. Another class of attack on interfaces that needs to be prevented are so-called side-channel attacks. Consistent with the attacker model outlined above, we consider only such channels that can potentially be executed via the Internet, e.g., timing side-channels. These attacks aim to extract cryptographic secrets to break encrypted communication; the best countermeasure is to make sure that all cryptographic code is timing invariant.

Protecting data in motion over the Internet is commonly achieved via TLS (Transport Layer Security) [33]. However, for the IoT scenario (where computational power is low) alternatives have been developed such as Datagram TLS (DTLS) [34], HIP Diet EXchange (DEX) [35], and minimal IKEv2 [36]. All three of them propose the usage of public key cryptography for key agreement and entity authentication. Only DTLS optionally defines a symmetric key based key agreement scheme. On the other hand, techniques from WSNs (Wireless Sensor Networks) are adapted to the context of IoT. As an example, a very low cost symmetric key based solution has been proposed in [37], [38] to secure a home automation system. The solution is based on the SPINS scheme [39] and ZigBee symmetric key agreement protocol [40].

Protecting public interfaces against the exploitation of software problems has a relatively long history, with most the research targeting PC-based systems. Commercial tools such as Klocwork [41] or Coverty [42] automatically validate that program code (such as C, C++) conforms to a given standard, e.g., MISRA-C. These tools furthermore greatly simplify the manual analysis of code.

The protection of cryptographic code against timing side-channels came into the spotlight after an academic attack on the RSA algorithm [43]. Research has been conducted with the goal of preventing such attacks; the idea was to write code such that the timing behaviour is independent of any sensitive data (e.g., key material). Instead of programming standard algorithms in a specific way—thus artificially slowing down already complex algorithms—a new set of cryptographic algorithms was developed which is inherently resistant to timing attacks [44]. Work is now ongoing to incorporate these algorithms into standards such as TLS and DTLS. Finally, to ensure that code carefully written in C (or C++) is not transformed into non-constant code by the compiler, recent work has focused on statistically measuring the timing behaviour of machine code [45].

2.5 IoT Operating Systems

There exists a range of Operating Systems (OS), usually, with real-time capabilities (RTOS), for the embedded and IoT domain. These Operating Systems are developed for resource constraint devices and usually provide options for feature customizations. For example, *TinyOS* is an open source BSD licensed OS for low power wireless embedded systems such as sensor node-type IoT devices [19]. *eCos* is an embedded operating system supporting a large number of target architectures and platforms [24]. It is highly configurable to enable its adaptation for a particular application. For example, it can be configured to support the POSIX thread API or to enable/disable support for task pre-emption in the scheduler. Over 200 such, partially inter-dependent, options are available [25]. A commercial RTOS solution is *VxWorks* from Intel Windriver [22]. VxWorks is a modular OS that can be configured for embedded target devices. The open source TinyOS, the embedded operating system eCos as well as the commercial RTOS solution is VxWorks experienced less activity during the project period.

Zephyr OS is an open source RTOS, managed by the Linux foundation. It supports multiple architectures and targets connected resource-constrained IoT devices [20]. Zephyr OS showed significant development progress during the project period. Not only has it consistently been extended to support newly released SoCs and peripheral devices, it also increasingly transfers concepts formerly only found in full-fledged operating systems to the embedded domain. These include support for Memory Protection and Memory Management Units (MPU/MMU), device tree based platform configuration, separation between kernel and user space, and support for symmetric multiprocessing (SMP). However, the availability and usability of these features varies strongly from platform to platform. *RIOT OS* is another open source project. The RIOT OS runs on several platforms including embedded devices and PCs. Its greatest advantage is an easy-to-use API. It targets power efficiency and has low resource demand [21]. RIOT OS was applied in some COMPACT demonstrators. Both operating systems are maintained by an active open-source community and are interesting solutions.

A commercial solution focusing on safety-certified, real-time applications, but more on high-performance platforms, is the *Nucleus RTOS* [23]. During the project period, e.g., 64-bit support for multicore system-on-chips was introduced. Another commercial solution is the *Maestro* from HIPPEROS, which provide a tight integration into the Xilinx SDSoC tools. Most recently, ERIKA Enterprise became popular in the automotive domain. ERIKA Enterprise is an open-source OSEK/VDX (AUTOSAR) hard real time OS with 1 - 4KB flash footprint and multi-core and stack sharing support. The OS ERIKA Enterprise switch during the project period to *ERIKA v3* [85], which is a complete rewrite of the kernel code base compared to ERIKA v2. The IoT-focused *Apache Mynewt* operating system is especially notable for its open Bluetooth 5 software stack implementation, called NimBLE [88]. However, NimBLE has also been integrated into other operating systems, such as RIOT OS. *HIPPEROS Tiny* is a minimalist version of the HIPPEROS RTOS family designed specifically for IoT devices (Class 1 or Class 2) [26]. Finally, *FreeRTOS* is a real-time operating system for microcontrollers. FreeRTOS experienced an active development though out the project duration. Especially with developments around the RISC-V instruction set, FreeRTOS provides ports in this direction.

2.6 GPU Code Generation and Optimization

Graphics Processing Units (GPUs) are the prevailing programmable accelerator concept for speeding up AI applications on IoT devices as well as in mobile computing. NVidia has a strong hold of desktop and server GPU hardware for AI related computing. This is mainly due to the advanced level of software interfaces it provides for the user. Specifically, the CuDNN middleware (and the CUDA compiler architecture) is designed for speeding up computationally intensive AI computations. For example, CuDNN offers NVidia hardware optimized implementations of convolution and matrix multiplication, which form the majority (>90%) of computing time used by the device. Outside the NVidia ecosystem, the OpenCL language is the only competitor. The benefit of OpenCL is its larger range of available platforms and vendors, including embedded, desktop and cloud enabled hardware.

Current research on automatic code optimization for GPUs is largely focused on the OpenCL programming model [56]. However, due to its generality and device type independence, the common platform model it presents poses some limitations in exposing the particular features of each GPU platform, resulting in multiple vendor specific extensions or several versions of the same program, costing extra working hours of programmers [57] [60]. Automatic code optimization is therefore essential to lower the cost of using GPU platforms for general computing.

Research on GPU code optimization focuses on two aspects that significantly differ between platforms: parallelism granularity and the memory model [58]. Jääskeläinen et al. [57] propose an optimizing OpenCL kernel compiler. The compiler, which works at the LLVM IR level, starts by extracting data parallelism information from the OpenCL kernels in a device independent phase. This information is then compiled in a target dependent manner which supports several types of fine grained parallel resources, such as SIMD extensions, SIMD data paths and static multi-use. Shen et al. [58] focus on code transformations for OpenCL performance portability, which, despite being aimed at CPU target platforms, are also of interest when targeting different types of GPU architectures. The proposed transformations, which can be applied in an automated way by a source to source compiler, include tiling for increased cache-locality, selection between implicit and explicit vectorization, adaptation of the memory access patterns depending on the support for coalesced memory accesses and work-group size selection for optimal trade-off between scheduling overhead and flexibility. Daga et al. [59] identify a series of target specific optimizations that can be implemented at compile time which include: mitigation of divergent executing, selective loop unrolling and automatic vectorization. Divergent execution paths affect different architectures differently, depending on the total number of cores handled by each single scheduler. One proposed method to avoid divergence is to use kernel splitting, separating different branches into different kernels. This solution is useful when the conditional value can be determined before to the kernel launch, in which case the required branch can be automatically selected by the scheduler. Another proposed optimization is loop unrolling that accounts for the target memory architecture. Here the authors propose that in some cases, it might be beneficial to leave memory accesses vectorized while unrolling only the computational elements. Finally, the use of vector types for computation, as opposed to scalar types, can also lead to significant performance gains in platforms sporting VLIW architecture, both by increasing the computational unit usage and lowering the dynamic instruction count. All these transformations, either at source level or intermediate level can be introduced in an optimizing kernel compiler which is aware of different hardware architectures and transforms the parallel regions accordingly, such as pocl [57]. However, combining different optimizations must be done with care, as not all of them are orthogonal [59], as, for example, the increase of register pressure might lead to a reduced total number of simultaneous threads. Finally, because of its general nature and the problems faced when supporting very different GPU architectures, optimizations designed for OpenCL code can be applied to other GPU programming models, such as NVIDIA's CUDA.

The inconvenience that programming of (embedded) GPUs is done by both the CUDA and the OpenCL language is alleviated by Halide [87], a high-level functional language that has a compiler that can produce executables for both CUDA and OpenCL enabled devices. Another benefit of Halide is that the language decouples the functional description of algorithms from the scheduling of the algorithms' operations, which makes Halide algorithm descriptions very portable, ranging from regular CPUs to DSPs and GPUs.

For computationally heavy AI applications the go-to accelerator is Graphics Processing Units (GPUs) due to their availability and high performance. NVidia hardware together its CUDA compiler architecture is still one of the most popular and highest performing options for an AI implementation. In 2017 NVidia released their Deep Learning Accelerator (NVDLA) hardware and software architecture as open source [88] making AI implementations more accessible to the public.

The only contender for NVidia's ecosystem is the OpenCL language which by design is intended for a wider range of hardware devices. Most of the recent GPU code optimization effort is done using OpenCL. In 2020 the Khronos OpenCL Working Group released version 3.0 of the OpenCL specifications which included new extensions that will improve the support of embedded processors. Compiler support in LLVM for OpenCL 3.0 is still a work-in-progress.

Halide is a high-level functional language that can be targeted for different GPU APIs including CUDA and OpenCL i.e. the same Halide code can be compiled for either platform. Image processing has been one of the key application areas for Halide from the start and AI applications are a natural extension of that. Using Halide for deep learning has already been proposed for example by Li et al. in [89] which shows Halide implementation to even exceed optimized CUDA program in performance while having lower coding effort.

2.7 IoT Analysis

The COMPACT analysis framework mainly applies Virtual Prototyping platforms. They provide the means to model, simulate, analyse, and verify heterogeneous mixed software/hardware system models. In the context of Electronic System Level (ESL) design, the notion of Virtual Prototyping mainly refers to the execution of target compiled software binary on hardware models before the final hardware is available.

Through the last years, virtual prototyping platforms based on Just-in-Time compilation became quite popular as they provide a significantly faster execution speed than cycle-accurate simulators. Popular commercial virtual prototyping environments are the Cadence Virtual System Platform [48], Mentor Graphics Vista [49], Synopsys Virtualizer [50], and Intel Simics [51]. QEMU [52], OVP (IMPERAS) [53], and ETISS [103] are freely available platforms with a high stability and speed. The latter mainly support the execution and debugging of binary software without Timing and power analysis support. The COMPACT project has implemented a time annotation extension of QEMU (QEMU Timing Analyzer) with aiT (Absint) as a static timing analyzer front-end.

The different commercial and open source platforms vary in the supported Instruction Set Architecture and hardware modelling language. The different open source platforms come with different license models.

Besides the execution of target binary code in a simulator, approaches exist that allow timing simulations by execution of timing-annotated software directly on the simulation host, often referred to as source level simulation. This approach is also capable for virtual prototyping and offers even faster simulations than binary level simulators with only little less accuracy [61, 62].

For analysis of ultra-thin software, other open source and commercial solutions already exist, such as Valgrind, mtrace or many others. Not all can handle arbitrary embedded devices, as they have usually no understanding of the underlying hardware.

2.8 Model-Aware Debugging

Several approaches towards model-level debugging have been proposed and some few commercial CASE tools exist for debugging at the model level. These can be categorized into host and target debugging. Host debugging refers to model-level debugging in a simulated environment while target debugging describes debugging on a target system. Target debugging can further be categorized into debugging with gdb facilities and debugging with a proprietary trace communication.

Some host based debugging approaches are based on model-checking [104] and simulation [105, 107, 109, 110]. Matlab/Simulink is one of the commonly used commercial tools for model driven software development in the area of embedded system design. It provides debugging at the model-level via simulation, as well as debugging the generated code via Software-in-the-Loop simulation [109]. Matlab/Simulink also supports real time testing (Rapid Prototyping and Hardware-in-the-Loop) [110]. Another approach discussed in [105] is the Rational Rhapsody. Rhapsody generates code with instrumentation from the model and uses its animation feature to validate the model by tracing and simulating the executable model.

In [107] a target based debugging approach based on generating and instrumenting code for different abstraction levels of an embedded system is described. At run-time trace data are collected and mapped back to the model. Debugging is then performed at model-level by visualizing the input data. Although this approach provides a mapping of run-time data from the platform to the model level, the code instrumentation increases the size of the code and affects the execution time in a production code. This approach (as in the Rhapsody approach) is in-efficient for small target systems due to limited resources. The target based debugging approach in [106] focuses on generating code with model-to-code traceability tags for State-chart models. During debugging, program slicing techniques are used on the generated code to identify a reduced program (Slice) responsible for an unwanted behavior. The program slice is then related to State-chart using the traceability tags. However, this approach is limited as program slicing will not work where there are faults due to errors of omission, for example missing variable initialization etc.

In the above approaches, debugging of an application on target system with both source-level and model-level debugging is not fully supported. [111, 112] addresses the combination of source-level and model-level debugging of an application running on a target platform using a de-facto gdb-like debugger.

3 References

[1] Gartner, “Gartner Says the Processing, Sensing and Communications Semiconductor Device Portion of the IoT Is Set for Rapid Growth,” Gartner, 03 11 2014. [Online]. Available: http://www.gartner.com/newsroom/id/2895917. [Accessed 23 10 2016].

[2] BI Intelligence Estimates, “The importance of SW in IoT,” [Online]. Available: http://blogs-images.forbes.com/louiscolumbus/files/2015/12/software-BI.jpg. [Accessed 25 10 2016].

[3] J. Gubbi, R. Buyya, S. Marusic and M. Palaniswami, “Internet of Things (IoT): A vision, architectural elements, and future directions,” Future Generation Computer Systems, vol. 29, no. 07, p. 1645–1660, 09 2013.

[4] A. Botta, W. de Donato, V. Persico and A. Pescape, “On the Integration of Cloud Computing and Internet of Things,” in FiCloud, 2014.

[5] W. He, Y. Gongjun and D. X. Li, “Developing Vehicular Data Cloud Services in the IoT Environment,” IEEE Trans. Ind. Informatics, vol. 10, no. 2, p. 1587–1595, 05 2014.

[6] G. Girardin, A. Bonnabel and E. Mounier, “Sensors & Technologies for the Internet of Things. Business & Market Trends 2014 – 2024,” Yole Development, May 2014.

[7] Y. Développement, “2014 TOP 20 MEMS Players Ranking,” 2014.

[8] European Commission, “The Internet of Things,” European Commission, 14 09 2016. [Online]. Available: https://ec.europa.eu/digital-single-market/en/internet-things. [Accessed 23 10 2016].

[9] S. K. Debray, W. Evans, R. Muth and a. B. D. Sutter, “Compiler techniques for code compaction,” ACM Transactions on Programming Languages and Systems (TOPLAS), 2000.

[10] B. D. Bus, B. D. Sutter, L. V. Put, D. Chanet and a. K. D. Bosschere, “Link-time optimization of ARM binaries,” Conference on Languages, compilers, and tools for embedded systems (LCTES), 2004.

[11] C. Lattner and V. Adve, “LLVM: A compilation framework for lifelong program analysis & transformation,” International Symposium on Code Generation and Optimization (CGO), 2004.

[12] T. Glek and J. Hubicka, “Optimizing real-world applications with GCC Link Time Optimization,” GCC Dev. Summit, 2010.

[13] A. Wolfe and A. Chanin, “Executing compressed programs on an embedded RISC architecture,” Proceedings of the 25th annual international symposium on Microarchitecture (MICRO 25), 1992.

[14] Y. Xie, W. Wolf and H. Lekatsas, “Code compression for embedded VLIW processors using variable-to-fixed coding,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, May 2006.

[15] C. Lefurgy, P. Bird, I.-C. Chen and T. Mudge, “Improving code density using compression techniques,” Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture (MICRO 30), 1997.

[16] S.-W. Seong and P. Mishra, “A bitmask-based code compression technique for embedded systems,” Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design (ICCAD '06), 2006.

[17] W. Wang and C.-H. Lin, “Code Compression for Embedded Systems Using Separated Dictionaries,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2016.

[18] A. Orpaz and S. Weiss, “A study of CodePack: optimizing embedded code space,” Proceedings of the tenth international symposium on Hardware/software codesign (CODES '02), 2002.

[19] TinyOS, 2016. [Online]. Available: http://webs.cs.berkeley.edu/tos/.

[20] Zephyr OS, “Zephyr Project Website,” 2016. [Online]. Available: https://www.zephyrproject.org.

[21] RIOT OS, “RiotOS Project Website,” 2016. [Online]. Available: http://www.riot-os.org.

[22] Intel Windriver, “VxWORKS RTOS,” 2016. [Online]. Available: https://windriver.com/products/vxworks/#VxWorks.

[23] Mentor, “Nucleus RTOS,” 2016. [Online]. Available: https://www.mentor.com/embedded-software/nucleus/.

[24] Ecos OS, “Ecos OS project page,” 2016. [Online]. Available: http://ecos.sourceware.org.

[25] Ecos OS Options, “Ecos Os Options,” 2016. [Online]. Available: http://ecos.sourceware.org/fom-serv/ecos/cache/23.html.

[26] HIPPEROS, “HIPPEROS,” 2016. [Online]. Available: http://www.hipperos.com/.

[27] AADL, “Architecture Analysis and Design Language,” 2016. [Online]. Available: http://www.aadl.info.

[28] ISO, Information Technology -MPEG Systems Technologies - Part 4: Codec Configuration Representation, Std. 23001-4:2011, 2011.

[29] H. Yviquel, A. Lorence, K. Jerbi, G. Cocherel, A. Sanchez and M. Raulet, “Orcc: Multimedia development made easy,” in Proc. 21st ACM Int. Conf. Multimedia, 2013.

[30] Pramudianto, F., I. R. I. and a. M. Jarke, “Model Driven Development for Internet of Things Application Prototyping,” The 25th International Conference on Software Engineering and Knowledge Engineering (SEKE), 2013.

[31] T. Riedel, D. Yordanov, N. Fantana, M. Scholz and C. Decker, “A Model Driven Internet of Thing,” Networked Sensing Systems (INSS), 2010.

[32] C. Prehofer, “From the Internet of Things to Trusted Apps for Things,” IEEE International Conference on and IEEE Cyber; Physical and Social Computing, 2013.

[33] IETF networking working group, “The Transport Layer Security (TLS) Protocol Version 1.2”. https://tools.ietf.org/html/rfc5246, accessed: 6.2.2018.

[34] IETF networking working group, “Datagram Transport Layer Security”. https://tools.ietf.org/html/rfc4347, accessed: 6.2.2018.

[35] R. Moskowitz, “HIP Diet EXchange (DEX),” draft-moskowitz-hip-dex-00 (WiP); IETF, 2012.

[36] T. Kivinen, “Minimal IKEv2,” draft-kivinen-ipsecme-ikev2-minimal-01(WiP); IETF, 2012.

[37] R. Smeets, K. Aerts, N. Mentens, D. Singelée, A. Braeken, L. Segers, A. Touhafi, K. Steenhaut and D. Niccolo, “A cryptographic key management architecture for dynamic 6LowPan networks,” Proc. of the 9th ICAI, 2014.

[38] R. Smeets, K. Aerts, N. Mentens, D. Singelée, A. Braeken, L. Segers, A. Touhafi and K. Steenhaut, “Efficient key pre-distribution for 6LoWPAN,” Proc. of the 5th International Conference on Applications and Technologies in Information Security (ATIS), 2014.

[39] A. Perrig, R. Szewczyk, J. Tygar, V. Wen and D. Culler, “Spins: Security protocols for sensor networks,” Wireless Networks, 2002.

[40] E. Yuksel, H. Nielson and F. Nielson, “ZigBee-2007 Security Essentials,” Proc. Of the 13th NordSec, 2008.

[41] Klocwork, “Klocwork static code analysis”. https://www.klocwork.com/products-services/klocwork/static-code-analysis, accessed: 6.2.2018.

[42] Synopsys, “Static application security testing”. https://www.synopsys.com/software-integrity/security-testing/static-analysis-sast.html, accessed: 6.2.2018.

[43] P. Kocher, “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”, Advances in Cryptology, Proceedings of Crypto, 1996.

[44] D J. Bernstein, T. Lange and P. Schwabe, "The security impact of a new cryptographic library", Proceedings of LatinCrypt, 2012.

[45] O. Reparaz, J. Balasch and I. Verbauwhede, “Dude, is my code constant time?”, Proceedings of DATE, 2017.

[46] K. S. a. R. Poovendran, “A Survey on Mix Networks and Their Secure Applications,” Proceedings of the IEEE, pp. 2142-2181, 2006.

[47] S. Mauw, J. H. S. Verschuren and E. P. Vink, “A Formalization of Anonymity and Onion Routing,” Computer Security – Esorics, 2004.

[48] Cadence, “Virtual Prototyping. www.cadence.com/products/sd/virtual_system,” 2016.

[49] Mentor, “Virtual prototyping. www.mentor.com/esl/ vista/virtualprototyping,” 2016.

[50] Synopsys, “Virtual Prototyping”. www.synopsys.com/Prototyping/VirtualPrototyping, 2016.

[51] Intel, “Simics. www.virtutech.com,” 2016.

[52] QEMU, “Homepage. www.qemu.org,” 2016.

[53] Imperas, “OVP. www.ovpworld.org,” 2016.

[54] J. Boutellier, J. Ersfolk, J. Lilius, M. Mattavelli, G. Roquier and O. Silven, “Actor merging for dataflow process networks.,” IEEE Transactions on Signal Processing, vol. 63, no. 10, pp. 2496-2508, 2015.

[55] J. Manyika, M. Chui, P. Bisson, J. Woetzel, R. Dobbs, J. Bughin and D. Aharon, “Unlocking the potential of the Internet of Things,” McKinsey Global Institute, 06 2016. [Online]. Available: http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/the-internet-of-things-the-value-of-digitizing-the-physical-world. [Accessed 24 10 2016].

[56] Khronos OpenCL Working Group. "OpenCL Specification Version 2.2,” May 2017.

[57] Jääskeläinen, Pekka, et al. "pocl: A performance-portable OpenCL implementation." *International Journal of Parallel Programming* 43.5 (2015): 752-785.

[58] Shen, Jie, et al. "Performance traps in OpenCL for CPUs." *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*. IEEE, 2013.

[59] Daga, Mayank, Thomas Scogland, and Wu-chun Feng. "Architecture-aware mapping and optimization on a 1600-core gpu." *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*. IEEE, 2011.

[60] Fang, Jianbin, et al. "Grover: looking for performance improvement by disabling local memory usage in OpenCL kernels." *Parallel Processing (ICPP), 2014 43rd International Conference on*. IEEE, 2014.

[61] Stattelmann, S., Bringmann, O., Rosenstiel, W. (2011). Fast and accurate source-level simulation of software timing considering complex code optimizations. *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, (S. 486-491).

[62] Bringmann, O., Ecker, W., Gerstlauer, A., Goyal, A., Mueller-Gritschneder, D., Sasidharan, P., & Singh, S. (2015). The Next Generation of Virtual Prototyping: Ultra-fast Yet Accurate Simulation of HW/SW Systems. *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, (S. 1698-1707).

[63] Cooper, Keith D., and Nathaniel McIntosh. "Enhanced code compression for embedded RISC processors." ACM SIGPLAN Notices 34.5 (1999): 139-149.

[64] Larin, Sergei Y., and Thomas M. Conte. "Compiler-driven cached code compression schemes for embedded ILP processors." Microarchitecture, 1999. MICRO-32. Proceedings. 32nd Annual International Symposium on. IEEE, 1999.

[65] Menon, Sreejith K., and Priti Shankar. "A code compression advisory tool for embedded processors." Proceedings of the 2005 ACM symposium on Applied computing. ACM, 2005.

[66] Menon, Sreejith K. "Studying the code compression design space–A synthesis approach." Journal of Systems Architecture 60.2 (2014): 179-193.

[67] IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating and Reusing IP within Tool Flows, IEEE Std 1685-2014.

[68] A. Kamppi, E. Pekkarinen, J. Virtanen, J.-M. Määttä, J. Järvinen, L. Matilainen, M. Teuho and T. D. Hämäläinen, „Kactus2: A graphical EDA tool built on the IP-XACT standard", The Journal of Open Source Software, 2017.

[69] F. Fleurey and B. Morin, "ThingML: A Generative Approach to Engineer Heterogeneous and Distributed Systems," 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), Gothenburg, 2017, pp. 185-188.

[70] I. Hautala, J. Boutellier, O. Silvén, „Towards efficient execution of RVC-CAL dataflow programs on multicore platforms", Springer Journal of Signal Processing Systems, 2018, to appear.

[71] H. Yviquel, A. Sanchez, P. Jääskeläinen, J. Takala, M. Raulet, E. Casseau, „Embedded multi-core systems dedicated to dynamic dataflow programs", Journal of Signal Processing Systems 80 (1), 121-136

[72] Object Management Group. UML. [Online] [Cited: 02 09, 2018.] http://www.omg.org/technology/readingroom/UML.htm

[73] Costa, Pires and Delicato. *Modeling IoT Applications with SysML4IoT.* 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2016.

[74] IoT-A Architectural Reference Model. [Online] [Cited: 02 09, 2018.] http://open-platforms.eu/standard_protocol/iot-a-architectural-reference-model

[75] Lekidis, et al. Model-Based Design of IoT Systems with the BIP Component Framework. *Software Practice and Expericen.* January 2018

[76] Basu, et al. Rigorous Component-Based System Design Using the BIP Framework. *IEEE Software.* 2011, Vol. 28, 3.

[75] Object Management Group. Common Variability Language. [Online] [Cited: 02 09, 2018.] http://www.omgwiki.org/variability/doku.php

[76] Kang, et al. *Feature-Oriented Domain Analysis (FODA) Feasibility Study.* s.l. : Carnegie Mellon University Pittsburgh, 1990.

[77] Bąk, et al. Clafer: Unifying Class and Feature Modeling. *Software & Systems Modeling.* 2016, Bd. 15, 3.

[78] Eichelberger, et al. *Integrated Variability Modeling Language: Language Specification.* 2015.

[79] Abele, et al. The CVM Framework - A Prototype Tool for Compositional Variability Management. *Fourth International Workshop on Variability Modelling of Software-Intensive Systems.* 2010.

[80] R. Merritt. Walmart Calls for Sub-$1 IoT Sensor. EET Online 12/4/2017. https://www.eetimes.com/document.asp?doc_id=1332669

[81] R. Merritt. IoT May Need Sub-50-Cent SoCs. EET Online 10/27/2017. https://www.eetimes.com/document.asp?doc_id=1332517

[82] J. Kathuria. Understanding IoT requirements 101, part 1. Embedded Computing, April 4th, 2017. http://www.embedded-computing.com/embedded-computing-design/understanding-iot-requirements-101-part-1

[83] G. Banjac, B. Stellato, N. Moehle, P. Goulart, A. Bemporad, and S. Boyd. Embedded Code Generation Using the OSQP Solver, IEEE Conference on Decision and Control, December 2017.

[84] T. Johnson, M. Amini, X. D. Li, A Framework for Scalable and Incremental Link-Time Optimization, IEEE/ACM International Symposium on Code Generation and Optimization (CGO), February 2017.

[85] Evidence Embedding Technology. ERIKA3. Online 26/02/2018. http://www.erika-enterprise.com

[86] Courbariaux, Matthieu, et al. "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1." *arXiv preprint arXiv:1602.02830*(2016).

[87] Ragan-Kelley, Jonathan, et al. "Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines." *ACM SIGPLAN Notices* 48.6 (2013): 519-530.

[88] Apache Mynewt, “Apache Mynewt Project Website,” 2020. [Online]. Available: https://mynewt.apache.org/

[89] NVidia, “NVDLA Open Source Project documentation”, 2020. [Online]. Available: http://nvdla.org/

[90] Tzu-Mao Li, Michaël Gharbi, Andrew Adams, Frédo Durand, Jonathan Ragan-Kelley. “Differentiable Programming for Image Processing and Deep Learning in Halide”. ACM Transactions on Graphics 37(4) (Proceedings of ACM SIGGRAPH 2018)

[91] A. Janapsatya, A. Ignjatovic, and S. Parameswaran, “A Novel Instruction Scratchpad Memory Optimization Method based on Concomitance Metric,” p. 6, 2006, doi: 10.1145/1118299.1118443.

[92] V. Suhendra, C. Raghavan, and T. Mitra, “Integrated scratchpad memory optimization and task scheduling for MPSoC architectures,” in *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems - CASES ’06*, Seoul, Korea, 2006, p. 401, doi: 10.1145/1176760.1176809.

[93] S. Udayakumaran, A. Dominguez, and R. Barua, “Dynamic allocation for scratch-pad memory using compile-time decisions,” *ACM Trans. Embed. Comput. Syst.*, vol. 5, no. 2, pp. 472–511, May 2006, doi: 10.1145/1151074.1151085.

[94] J. Hu, C. J. Xue, Q. Zhuge, W.-C. Tseng, and E. H.-M. Sha, “Data Allocation Optimization for Hybrid Scratch Pad Memory With SRAM and Nonvolatile Memory,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 6, pp. 1094–1102, Jun. 2013, doi: 10.1109/TVLSI.2012.2202700.

[95] J. Pallister, K. Eder, and S. J. Hollis, “Optimizing the flash-RAM Energy Trade-off in Deeply Embedded Systems,” in *Proceedings of the 13th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, Washington, DC, USA, 2015, pp. 115–124, Accessed: Jan. 21, 2019. [Online]. Available: http://dl.acm.org/citation.cfm?id=2738600.2738615.

[96] M. Strobel, M. Eggenberger, and M. Radetzki, “Low power memory allocation and mapping for area-constrained systems-on-chips,” *J Embedded Systems*, vol. 2017, no. 1, Jul. 2016, doi: 10.1186/s13639-016-0039-5.

[97] Y. Li, J. Zhan, W. Jiang, and J. Yu, “Energy optimization of branch-aware data variable allocation on hybrid SRAM+NVM SPM for CPS,” in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, Limassol Cyprus, Apr. 2019, pp. 236–241, doi: 10.1145/3297280.3297305.

[98] C. Fu, G. Calinescu, K. Wang, M. Li, and C. J. Xue, “Energy-Aware Real-Time Task Scheduling on Local/Shared Memory Systems,” in 2016 IEEE Real-Time Systems Symposium (RTSS), Porto, Portugal, Nov. 2016, pp. 269–278, doi: 10.1109/RTSS.2016.034.

[99] M. E. T. Gerards, J. L. Hurink, and P. K. F. Hölzenspies, “A survey of offline algorithms for energy minimization under deadline constraints,” Journal of Scheduling, vol. 19, no. 1, pp. 3–19, Feb. 2016, doi: 10.1007/s10951-015-0463-8.

[100] LLVM, “LLVM Website”, 2020. [Online]. Available: https://llvm.org

[101] GCC, „GCC Website“, 2020. [Online]. Available: https://gcc.gnu.org

[102] Stahl, Rafael, Daniel Mueller-Gritschneder, and Ulf Schlichtmann. "Driver generation for IoT nodes with optimization of the hardware/software interface." IEEE Embedded Systems Letters 12.2 (2019): 66-69.

[103] Daniel Mueller-Gritschneder; Martin Dittrich; Marc Greim; Keerthikumara Devarajegowda; Wolfgang Ecker; Ulf Schlichtmann. The Extendable Translating Instruction Set Simulator (ETISS) Interlinked with an MDA Framework for Fast RISC Prototyping 2017 International Symposium on Rapid System Prototyping (RSP), 2017.

[104] María del Mar Gallardo, Pedro Merino, and Ernesto Pimentel. 2002. Debugging UML Designs with Model Checking. *Journal of Object Technology* 1, 2 (July 2002), 101–117. https://doi.org/10.5381/jot.2002.1.2.a1

[105] Eran Gery, David Harel, and Eldad Palachi. 2002. Rhapsody: A Complete Life-Cycle Model-Based Development System. In Integrated Formal Methods, Michael Butler, Luigia Petre, and Kaisa Sere (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–10.

[106] Liang Guo and Abhik Roychoudhury. 2008. Debugging Statecharts Via Model-Code Traceability. In Leveraging Applications of Formal Methods, Verification and Validation, Tiziana Margaria and Bernhard Steffen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 292–306.

[107] W. Haberl, M. Herrmannsdoerfer, J. Birke, and U. Baumgarten. 2010. Model-Level Debugging of Embedded Real-Time Systems. In 10th IEEE International Conference on Computer and Information Technology. 1887–1894. https://doi.org/10.1109/CIT.2010.323

[108] MathWorks. 2018. Stateflow Documentation – MathWorks Deutschland. Retrieved October 11, 2018 from https://de.mathworks.com/help/stateflow/

[109] MathWorks. 2018. Test and Debug Simulations – Matlab & Simulink - MathWorks Deutschland. Retrieved October 16, 2018 from https://de.mathworks.com/help/simulink/test-and-debug-simulations.html

[110] MathWorks. 2018. Test Models in Real Time – Matlab & Simulink - MathWorks Deutschland. Retrieved October 16, 2018 from https://de.mathworks.com/help/sltest/ug/test-models-in-real-time-and-assess-results.html

[111] Bewoayia Kebianyor, Philipp Ittershagen, and Kim Grüttner. 2019. Towards Stateflow Model-Aware Debugging using Model-to-Source Tags with LLDB. 2nd International Workshop on Embedded Software for Industrial IoT (ESIIT) at DATE'19

[112] Bewoayia Kebianyor, Philipp Ittershagen, and Kim Grüttner. 2019. Towards Stateflow Model Aware Debugging with LLDB. In Proceedings of the Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO '19). Association for Computing Machinery, New York, NY, USA, Article 1, 1–8. DOI: https://doi.org/10.1145/3300189.3300190

4 Appendix A - Table of Acronyms

Acronym	Definition
AADL	Architecture Analysis and Design Language
ARM	Advanced RISC Machine
ASIC	Application Specific Integrated Circuit
BIP	Behaviour, Interaction, Priority
BSD	Berkeley Software Distribution
COMPACT	Cost-efficient Smart System Software Synthesis
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
CVL	Common Variability Language
DSP	Digital Signal Proccessor
DTLS	Datagram Transport Layer Security
EAST-ADL	Electronics Architecture and Software Technology - Architecture Description Language
EDA	Electronic Design Automation
ESL	Electronic System Level
FODA	Feature-Oriented Domain Analysis
FPGA	Field-programmable Gate Array
GPU	Graphics Processing Unit
IoT	Internet of Things
IoT-PML	Internet of Things Platform Modelling Language
IPO	Inter-procedural Optimization
ISA	Instruction Set Architecture
IR	Intermediate Representation
IVML	INDENICA Variability Modeling Language
LIPO	Light-weight Inter-procedural Optimization
LLVM	Low Level Virtual Machine
LTO	Link Time Optimization
MDA	Model Driven Architecture
MDE	Model Driven Engineering
MIPS	Mega Instructions Per Second
OMG	Object Management Group
OVP	Open Virtual Platforms
PGO	Profile Guided Optimizations
POSIX	Portable Operating System Interface
QEMU	Quick Emulator
RAM	Random Access Memory
REST	Representational State Transfer
RISC	Reduces Instruction Set Computer
ROM	Read Only Memory
RSA	Rivest–Shamir–Adleman
RTOS	Real Time Operating System
RVC-CAL	Reconfigurable Video Coding – Cal Actor Language
SoC	System on Chip
SIMD	Single Instruction, Multiple Data
SysML	Systems Modelling Language
TLS	Transport Layer Security
TTA	Transport Triggered Architecture
UML	Unified Modeling Language
UPDM	Unified Profile for DoDAF/MODAF
VLIW	Very Long Instruction Word
VSL	Variability Specification Language
V2FCC	Variable-to-fixed Code Compression