Framework for an Embedded Operating System Adaptation

Abstract—**Each year, manufacturers develop better and more effective processors. As a reaction to this continuous development, developers of software have to adapt their software to those new processors. At least, code of an operating system has to be changed to enable execution of other user applications. Most of modern embedded operating systems have a kernel architecture. It means that those operating systems manage: I/O devices of a processor, memory of a system, and scheduling of tasks. When the operating system has to be used on a new processor architecture, mentioned blocks have to be modified. Mostly affected part of an operating system is its platform dependent layer. In this layer, the code connected to the processor architecture has to be changed. In our work we proposed a framework for an operating system management which helps to reduce complexity of an adaptation process. In this paper we present concept of this framework.**

Keywords-**Adaptation of Embedded Operating systems, Formal description of Processors, Source Code Generation, Processor Structure**

I. Introduction

With growing number of new processor architectures grows need for methodology which allows fast and effective operating system adaptation or porting. Future embedded systems will have multi-core architecture, many-core architecture [1] or mixed architectures combining multi-core processors into clusters. New types of architectures will introduce new types of operating systems which will be self-adaptive [2]. New operating systems running in a heterogeneous environment will need a database of existing processor ports, device modules and processing cores. Needed modules and platform ports will be loaded to processor program memories during system initialization or will be accessed online at a run-time. New hardware architectures will be easily extendable even during system run-time so operating system will have to be loaded to new added processors from a database. To create such database developer needs to implement missing device modules and processing core modules and platform ports for an operating system. The task is not easy because the number of existing processor is growing quickly.

In this paper we propose a framework which will be supporting process of adaptation of kernel embedded operating systems. The framework consists of tools which helps to describe processor [3], create platform port (platform dependent code) [4], describe operating system modules which manages devices and processing cores of a processor, and implement those modules. The process of an operating system adaptation is described step by step in this paper.

II. Related Work

Adaptation of operating systems is mostly realized per one operating system. Mostly it is a need to port specific operating system to architecture which is in some way special like is adaptation of OSE operating system to many-core architecture Tilepro64 [5]. Mapping of operating system scheduler was done on mentioned many-core architecture.

Well known operating systems such is FreeRTOS [6] or more other operating systems [7], [8] provides to user adaptation manuals. In those manuals is summarized which parts of an operating system have to be re-implemented during adaptation to a new processor.

Many-core systems change standard concept of processor as a system with many devices and some processing cores. The number of cores will grow and the new architectures will use intelligent devices connected to a network together with cores [1]. This highly scalable architecture calls for adaptivnes of operating systems and change of standard architecture of operating systems into distributed architecture.

III. Process of OS Adaptation

Adaptation framework is designed to help developer of an embedded operating system to produce a platform dependent code of an operating system for chosen processor and to develop needed modules of an operating system which will manage existing processor devices and cores.

The framework is connected with adaptation process which is described in a Figure 1. The adaptation process is suitable for kernel embedded operating systems which can be divided into platform dependent and platform independent layer as are MOS [9], FreeRTOS [6] and many more. As an input of this process developer needs:

- **Processor datasheet** where can be found information about processor devices and processing cores.
- **Processing cores datasheets** where can be found information about processing core of the processor.
- **Processor description file** which contains information about communication interface of every device and processing core in the processor [3].

Processor formal description file defined in [3] contains information about every device and core which is in the processor. This description helps to developer and also to computer easily identify communication interfaces of devices and processing cores. This communication interface is central part of an operating system platform dependent code. More information about the formal description can be found in a section III-C.

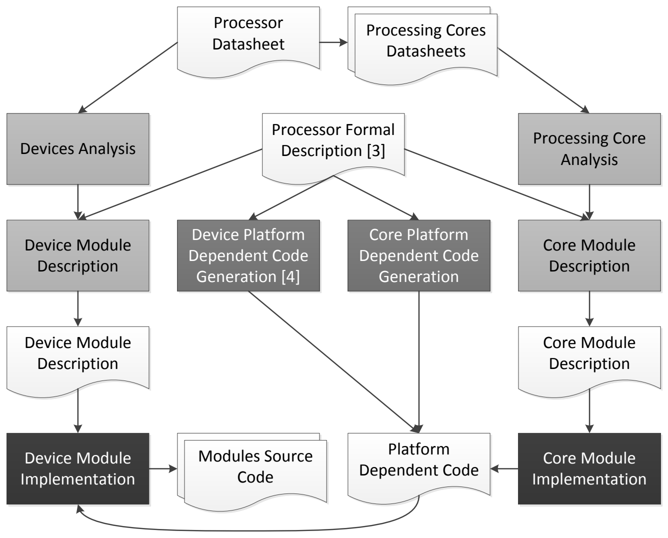
Processor Datasheet
Processing Cores Datasheets
Devices Analysis
Processor Formal Description [3]
Processing Core Analysis
Device Module Description
Device Platform Dependent Code Generation [4]
Core Platform Dependent Code Generation
Core Module Description
Device Module Description
Core Module Description
Device Module Implementation
Modules Source Code
Platform Dependent Code
Core Module Implementation

Fig. 1. Process of an embedded operating system adaptation to a new processor.

According to Figure 1. the adaptation process can be divided into two separate workflows. A workflow on the left side of the figure is focused on processor devices. A workflow on the right side of the figure is focused on processing cores.

A. Device Workflow

The device workflow consists of device analysis phase, device module description phase, platform dependent code generation phase and device module implementation phase.

Device workflow begins with the device analysis phase where a functionality of each device in the processor is analyzed. In the analysis phase, developer of an operating system processes information about devices from a processor datasheet. Developer searches for information such as device initialization, possible working modes, device timing diagrams and behavior, and sources of interrupts.

In the device module description phase developer produces a description of a device module. In this phase, a description of a device communication interface is used from the processor formal description file. Developer maps the interface to a module description and creates a connection between the module and the processor device. Result of this phase is module description file, which allows full or partial code generation depending on complexity of device and created module description. More information about the description file can be found in the section III-E.

Parallel phase to the device description phase is phase of device platform dependent code generation. In this phase a platform dependent code for a device communication interface is generated. The resulting code based on a device formal description creates an interface layer between a processor and a device module. More information about generation of a platform dependent code can be found in the section III-D.

Last phase of the device workflow is a phase in which description of a module is implemented and/or generated to chosen programming language. In this phase a device module is implemented and linked with the platform dependent code.

B. Core Workflow

Similarly as is in the device workflow the core workflow consist of analysis phase, module description phase, platform dependent code generation phase and module implementation phase.

In the analysis phase, developer has to analyze processing core architecture. Main tasks of the analysis is to analyze interrupt processing, process of a processing core initialization, possible working modes of processing core, and how a task switch routine can be implemented. Some information about processing core is written in the processor datasheet but full description can be mostly found in a separate datasheet of the processing core.

In the processing core module description phase a description of a core is produced. The resulting file describes the process of the processing core initialization, interrupt handling and task switch. More information can be found in the section III-F.

Based on processing core description form the processor formal description a platform dependent code of a core communication interface is generated. More information about the formal description can be found in the section III-D.

In an implementation phase of a processing core module, developer implements code which allows initialization of processing core and handles interrupt subsystem and task switch. The nature of the processing core module determines its placement into a platform dependent layer of an operating system because the code is mostly written in assembly language compatible with a processor and therefore is platform dependent.

C. Processor Formal Description

The processor formal description is a way how can be processor described in a form which is readable by computer. This description was designed for platform dependent code generation. Main idea is that the processor can be decomposed into devices and processing cores. Devices collect and send data to processing cores. Cores process data under defined program which is represented by instructions. The formal description describes communication interfaces of devices and cores [3].

Structure of the formal description file is shown in the Figure 2. It shows that each device of a processor consists of a set of *interrupt signals* and a set of *registers*. Each register can be then decomposed into a set of *parts*. Each of them can come by more *options*. In some situations, one part of a device register can depend on another part if there exist connection which can limit access to the dependent part.

The description of a processor cores is work in progress but in short a core can be decomposed into a set of general purpose registers and status registers which represent state space of a processing core [3].

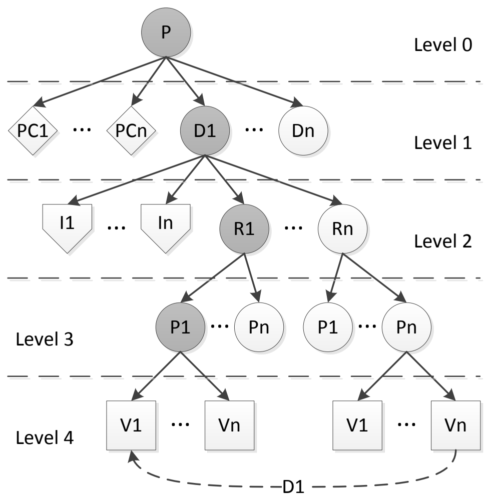

Fig. 2. Structure of the processor formal description file. Where P is processor, PCi is Processing core, Di is Device, Ri is Device register, Ii is interrupt signal, Pi is Register part, Vi is Part Value and Di is a dependence between parts [4].

Main advantage of the formal description is that this description is not connected to any programming language so the developer of an operating system can select a programming language which he prefers during implementation not during design phase. Another advantage of the formal description is that information about devices and cores can be processed by code generator. Generated code is then compatible with operating system structures so the adaptation of operating system is faster and more effective.

Disadvantage of processor formal description is that it has to be prepared by developer during analysis of the processor. But we believe that this file can be prepared by processor manufacturer together with processor datasheet in the future. Even now most of processor manufacturers provide definition files where a processor platform dependent code can be found but the code structure varies from manufacturer to manufacturer so the processor formal description will standardize those files.

D. Generation of Platform Dependent Code

Generator of code processes information obtained from the processor formal description file of a processor. The generator generates pieces of platform dependent code which interface processor with operating system structures.

The process of code generation can be divided into two phases. In the first phase that is called *definition phase*, definitions for every device, register, part and option are generated [3]. Those definitions link every mentioned element with an identifier which helps to understand communication interface of a device. In the second phase that is called *function phase*, interface functions that manages writes and reads to/from registers and parts are generated. Generated function realizes simple operation which uses definitions to address needed register or part of a device without any loops or conditions.

Simplicity of the processor formal description allows to implement generator which produces code in a programming language preferred by developer of an operating system and in a structure and nomenclature which is compatible with an operating system structure [3].

E. Description of Device Modules

Device module of operating system manages mapped device of the processor. The description can be divided into 3 parts which are modeled independently.

- Device Control,
- Device Access Management,
- Device Interrupt.

Processor device is often a complex structure which can operate in more modes and under many influences. The control description declares how a device will be initiated at system start up or how the device operation will be altered during system execution. It is also used for device status monitoring. Device control can consist of simpler control flows which can manage device start up, device termination or power management.

The device access management controls whether a device is used by task or not. The complexity of access management can vary from simple solutions controlling whether device is running up to complicated solutions with waiting task stacks and task priority ordering. The design of manager is up to developer of operating system.

Most of processor devices can produce interrupt signal which informs processor about specific situation in which a device is. The description of interrupt describes how each interrupt source of a device will be processed. An interrupt handler is mostly represented by simple condition, which searches for source of interrupt in a device and calls predefined routine that is executed in a user mode.

F. Description of Core Modules

Similarly as device modules a processing core modules are described in a way which allows simpler implementation of defined processing core functionality. The description can be divided into 3 separate parts:

- Core initialization,
- Core Task switch,
- Core Interrupt.

During system start up, a processing core has to be initialized. This initialization takes care on set up of operating modes of core and preparation of operating system start up.

Main task of operating system is planning and switching of tasks. Task planning does not depend on chosen processor as much as task switching. To enable task switch, a procedure which stores old task context and loads new task context have to be implemented. Task switch can occur after event such as interrupt, task create or after defined period of a time. The complexity of a task switch depends on an instruction set architecture of the processing core and it can vary from one instruction which realizes whole process in a one atomic step to a complex set of instructions which can run through several operating modes of the processing core. Especially processors for embedded application which have RISC architecture have often complicated task switch routine. The complexity of routine also depends on an operating system architecture. An operating system which runs in a kernel and a user mode uses the processor operation modes to separate a user space form a kernel space which have good security reasons but this can result into more complicated task switch procedures.

Similarly like task switch the interrupt process is described to enable implementation of interrupt processing routine of processing core. This routine mostly reads an interrupt source and searches for an interrupt subroutine which will be called. The task switch is often one of the called routines.

IV. OS Adaptation Framework

Concept of OS adaptation framework is based on operating system adaptation process. This framework will support the process by set of services and databases which help manage operating system adaptation. In a Figure 3. is presented concept of the framework. The framework services are:

- processor formal description describing,
- processor formal description validating,
- platform dependent code generating,
- device module description modeling,
- processing core description modeling,
- device module to communication interface mapping,
- core module to communication interface mapping,
- device or core module validating,
- code implementation and/or generation, and
- selection of platform dependent code and modules code.

Beside framework's services the framework uses 3 databases:

- database of processor formal description,
- database of operating system modules, and
- database of operating system source codes.

1) Description of a Processor: The framework allows preparing processor formal description files (PFD) from datasheets. Prepared PFD is then validated and sent to a database. Stored description can be then used by generator to produce platform dependent code of operating system where from description of the processor can be generated platform dependent layer of an operating system. Descriptions are also used for mapping of operating system modules to communication interfaces of devices and processing cores.

Inserted description of processor will be decomposed into devices and cores. Each identified device will be inserted to a database under version which will guarantee that device is not presented in the database as a duplicity (We assume that each device and core will have unique identification).

2) Description of a Module: Since the formal description of a processor covers only a communication interface of a device or processing core there is still need for development of an OS module which manages this device or processing core. The formal description will be used for mapping of module to communication interface. Resulting model of an operating system module will be validated and then inserted to a database of operating system modules. Each module will have unique identification. The documentation to a model will be compulsory part of the design.

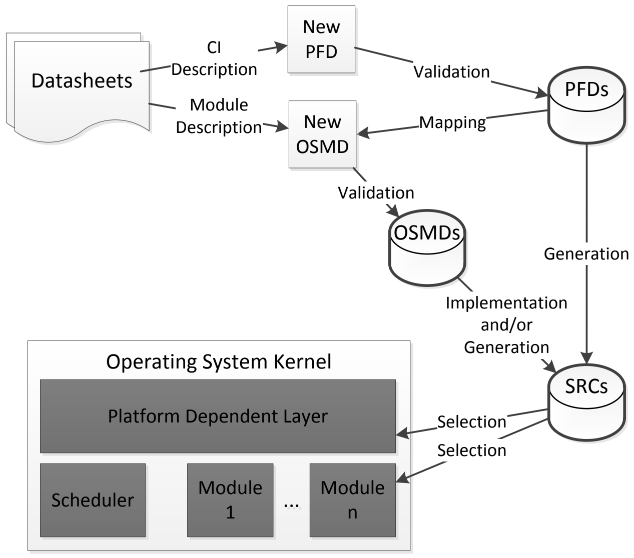

Fig. 3. Set of adaptation framework services, where CI is an communication interface of a device or processing core, PDF is processor description file, OSMD is operating system module description, PDFs is a database of processor formal descriptions, OSMDs is a database of operating system modules and SRCs is database of implemented and/or generated source codes.

3) Module Code Generation/Implementation: From module description developer will have to implement operating system module. Some parts of module can be generated automatically as a skeleton of the module which will help to developer during implementation. Implemented module is then stored in a database of operating system source codes together with unique version, linkage to parent module description and compulsory documentation.

4) Selection of Operating System Parts: As a part of a system design, developer of embedded system will have access to database of operating system source codes. From this database developer will choose platform dependent layer and select compatible device and processing core modules for chosen processor. He can also add/describe/implement missing modules or other parts.

A. Database of Descriptions

Full formal description files of processors can be found in a database of descriptions. Those files will be also decomposed into separate devices and processing cores. The problem can rise when same device exists in a more processors so this situation has to be solved by device unique identification. A protection from processor description duplication has to be created which does not allows upload again existing processor description. If existing processor was revised by manufacturer there will be possibility to revise formal description too.

The database can be used also during creation of new description files where developer can search existing devices and processing cores in database and include them into new description, which will reduce duplicity and description time.

B. Database of Modules

This database will store descriptions of operating system device and processing core modules. As in database of descriptions this database will use unique identification and versioning. Descriptions from database can be used for describing similar device modules as working examples. Existing modules then can be reused in module description which will reduce description time.

C. Database of Sources

In the database of sources will be stored unique versions and ports of operating system source code. Developer will be able to select platform dependent code for selected processor and he will be able to select module source codes based on description of a module, because there will be presented more versions of the module.

V. Conclusions

A concept of adaptation framework for embedded operating systems was presented in this paper. The framework will provide services for developer of embedded operating system which helps during adaptation of an operating system to new processors. The adaptation time will be shorter and adaptation complexity simpler. Until now, a formal description of processor and generator of platform dependent code was developed. Generator generates platform dependent code in programming language C. Next step in the work is design of a module description tool which allows describing operating system modules.

Acknowledgment

References

[1] P. Ranganathan, "From microprocessors to nanostores: Rethinking data-centric systems," *Computer*, vol. 44, no. 1, pp. 39–48, Jan 2011.

[2] M. Seltzer and C. Small, "Self-monitoring and self-adapting operating systems," in *Operating Systems, 1997., The Sixth Workshop on Hot Topics in*, May 1997, pp. 124–129.

[3] M. Vojtko and T. Krajčovič, "Adaptability of an Embedded Operating System: a Formal Description of a Processor," in *10th International Joint Conferences on Computer, Information, Systems Sciences, and Engineering*, Dec. 2014, p. 4, in press. [Online]. Available: http://www2.fiit.stuba.sk/%7evojtko/VojtkoAoEOS.pdf

[4] M. Vojtko and T. Krajčovič, "Adaptability of an Embedded Operating System: a Generator of a Platform Dependent Code," in *Digital System Design (DSD), 2015 18th Euromicro Conference on*, Aug 2015, p. 5, in review. [Online]. Available: http://www2.fiit.stuba.sk/%7evojtko/VojtkoAoEOS2.pdf

[5] V. Avula, "Adapting operating systems to embedded manycores: Scheduling and inter-process communication," Master's thesis, Uppsala universitet, 2014.

[6] *The FreeRTOS Project*, 2011, http://www.freertos.org/.

[7] L. Barello, *AvrX Real Time Kernel*, 2007, http://www.barello.net/avrx/.

[8] *TinyOS*, 2011, http://www.tinyos.net.

[9] M. Vojtko and T. Krajčovič, "Prototype of modular operating system for embedded applications," in *Applied Electronics (AE), 2013 International Conference on*, Sept 2013, p. 4.