E²R: Hardware Abstraction Layer for Configurable Execution Modules

Hendrik Seidel, Marcus Bronzel, Technische Universität Dresden (seidel@ifn.et.tu-dresden.de)
Syed Naveen Altaf A, Institute for Infocomm Research, Singapore (naveen@i2r.a-star.edu.sg)
Bernd Steinke, Nokia Research Center, Bochum, Germany (bernd.steinke@nokia.com)
Mirsad Halimic, Panasonic MDL, Uxbridge, Middlesex, UK (mirsad.halimic@panasonic-pmdc.co.uk)
Laurent Alimi, Thales, France (laurent.alimi@fr.thalesgroup.com)

***Abstract*—The dynamic and platform independent reconfiguration of reconfigurable devices in an E²R environment is based on a reconfiguration strategy which is distributed across different layers of abstraction. This paper introduces an architecture of the hardware abstraction layer for configurable execution modules, which perform the reconfiguration on a fine grain level and carry out the tasks associated with a particular radio access technology.**

***Index Terms*—E²R, Physical Layer, Hardware Abstraction Layer, Configurable Execution Modules, Logical Device Driver, Physical Device Driver, Service Interface**

I. Introduction

FUNCTIONAL elements in an End-to-End Reconfigurable (E²R) [1] environment have been identified and described in [2]. In order to provide partial or seamless dynamic reconfiguration of E²R equipment, a layered approach is followed which distributes the process of reconfiguration across several layers. Functional elements are described and modelled on different levels of abstraction which hide the implementation details to higher layers, thus enabling a dynamic reconfiguration process on different layers.

After presentation of the general layered architecture of E²R devices which allows for a platform independent hierarchical reconfiguration, a Hardware Abstraction Layer (HAL) is introduced, which hides the details of the underlying Configurable Execution Modules (CEM) to the Configuration Control Module (CCM) in the higher layer. The HAL architecture and its methods to access various classes of CEMs are discussed in detail, followed by an outline of an operating environment for programmable CEMs, which is needed to carry out real time tasks with modem functionality for a particular Radio Access Technology (RAT). Different configuration processes for CEMs are explained with particular emphasis on the communication fabric, which interconnects the CEMs. The role of configuration data from the CCM and a suitable configuration language is covered in [3].

Related work with a different application background was done by the OMG within the Software-Based Communication Task Force [4].

II. General Architecture of E²R-Devices

Wireless terminals and basestations or access points in an E²R-environment provide an abstract configuration interface to the signal processing components. This interface is supplied by the CCM, which is implemented in an Instruction Set Architecture (ISA) device. An overlaying CMM is able to access and modify the implemented RAT in the device through this interface. The interface is realized with the CCM Service Interface which is hiding implementation details of the underlying hardware. The access is unified, hence, platform and vendor independent.

The CEMs in an E²R device execute the functional chain of a particular RAT. Similar to the CCM, which hides implementation details from higher layers using a service interface [5], CEMs also provide a service interface to hide their implementation details from the CCM. This CEM Service Interface (CEM_Service_IF) is located inside the CEM HAL. The major part of a CEM concerns the hardware to execute RAT functions; the hardware abstraction layer of a CEM is therefore just the necessary overhead to unify the access to CEM hardware. The abstraction layer of the CEMs allows two different views on CEMs:

- ***Functional view:*** The overlaying CCM is able to configure the implemented functions on the CEM. How to configure these functions is generalized and independent from the individual CEM implementation.
- ***Configuration view:*** A hierarchical configuration of the implemented CEM is possible. Starting from a configuration of parameters available on all realizations of the specific class of CEM to the configuration of the individual implementation of the particular CEM. Configuration can simply add or remove implemented functions or adjust their functionality.

As shown in Figure 1, CEMs are subdivided into two classes:

The first class of Processing-CEMs (P-CEM) carries out the data processing functions for a given RAT implementation. These P-CEMs perform computational tasks.

The second class of Communication-CEMs (C-CEM) is responsible for connecting the P-CEMs. These C-CEMs mange the complete interconnect for processing and control data.

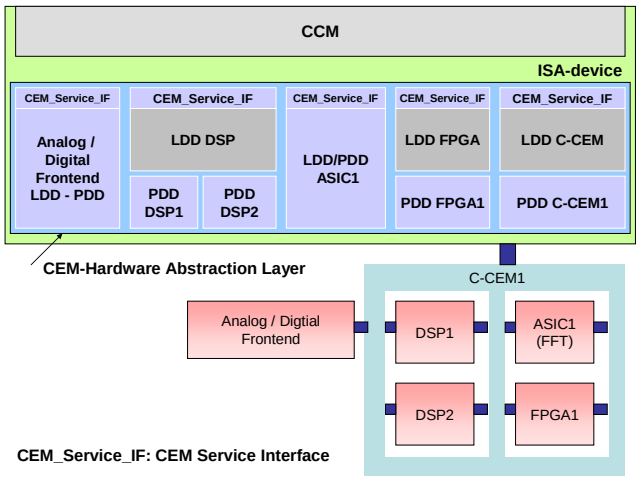

Figure 1: Processing CEMs and Communication CEMs connected to the CEM HAL.

A. *Processing-CEM (P-CEM)*

In an E²R device physical layer processing is done with state-of-the-art signal processing devices, ranging from custom ASICs and RF-front-end to FPGAs, programmable accelerators and general purpose DSPs. Different technologies can be used for these physical layer modules. These data processing devices build the class of P-CEMs in the E²R environment. System architects can build a heterogeneous system on chip, system in a package (multiple silicon chips connected using e.g. flip-chip technology, delivered in a single package) or printed circuit board from these P-CEMs.

In an hierarchical abstraction, the P-CEMs are divided into five basic subclasses: programmable logic devices and compute fabrics (e.g. FPGA), DSPs, Accelerators, ASICs and RF-Modules. Later in this paper we will describe the concept of Logical Device Drivers (LDD) and Physical Device Drivers (PDD). A LDD is provided for each specific class of P-CEM while a PDD adapts these classes to a specific (e.g. proprietary) implementation of the technology.

All these P-CEMs have interfaces to exchange data with other CEMs and to store data in external memories. P-CEMs can be connected with shared or global memory using First-In-First-Out (FIFO) buffers or interrupts based streaming data C-CEMs.

B. *Communication-CEM (C-CEM)*

The C-CEM provides physical interfaces to connect the different P-CEMs to the master ISA device, which runs the CCM. It will also provide a configurable physical interconnection fabric for routing data and control information between the different P-CEMs.

The C-CEM will have a set of ports defined. The number of ports and the number of physical connections in each port will depend on the individual implementation. Each of the P-CEM will connect to the C-CEM through these ports. In Figure 2, a C-CEM with 2 Ports P1 and P2 is shown. The port configuration and assignment is reconfigurable and will be configured according to the interface specifications of the P-CEM, which are provided by the PDD. The C-CEM communicates with the CCM through the CEM_Service_IF, which is composed of two separate interfaces: Configuration Interface and Functional Interfaces.

Apart from providing the physical interfaces between the P-CEMs and the ISA device, C-CEMs will also provide interfaces to transfer data between the different P-CEMs which will be present in the system. The topology of the interconnect architecture for these transfers will be selected by the CCM or an entity which will form the configuration processing chain and map this chain on to the underlying P-CEMs, in order to execute a particular RAT.

In order to satisfy this need for inter-CEM communication and data transfer, while modelling the C-CEM, different interfaces one each for a particular P-CEM are realized [6]. This individual interfaces will be instantiated only when the particular P-CEM is present in the system configuration.

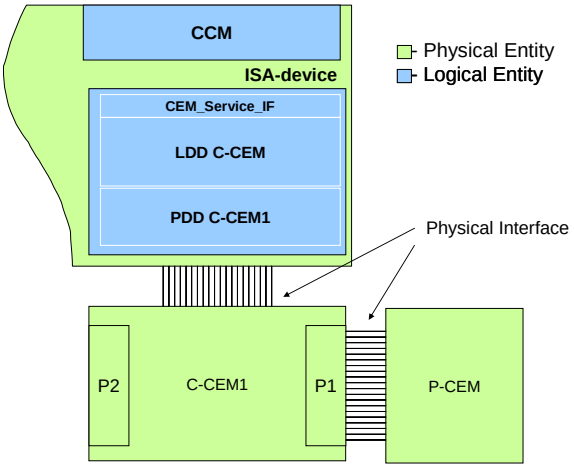

Figure 2: Communication CEM Interfaces

III. CEM - Abstraction Layer

To realize the two different views on the CEM two modules are provided for system abstraction: A physical device driver (PDD) acts as software glue to access the CEM-hardware through the communication fabric. The logical device driver (LDD) realizes the functional and configuration view on the CEM. The LDD links the current configuration of the CEM with the functional capability. For the configuration view the CEM Service Interface of the LDD provides a Configuration Interface (CIF) for hierarchical configuration and implementation of functions. The Functional Interface (FIF) of the CEM Service Interface supports the functional view on the CEM. To implement the functional and configuration interface a set of processing engines within the LDD and PDD are necessary. Hence, it consists of three key components:

1) CEM_Msg_Handler

This entity reads the received configuration language data messages intended for FIF and CIF; it handles the communication with the overlaying CCM. If required, a possible implementation of the CEM_Msg_Handler is a Message Passing Interface [7] module.

2) Configuration Interpreter:

The configuration interpreter is able to interpret the received data on the FIF depending on the actual configuration of the CEM.

3) Config_datastore (CDS):

A small data storage to store the current configuration of a specific CEM class. This database is very small to reduce power of the device. Hence, it stores just the necessary data to interpret received data on the functional interface pertaining to actual device configuration.

The architecture of a CEM hardware abstraction layer module is shown in Figure 3.

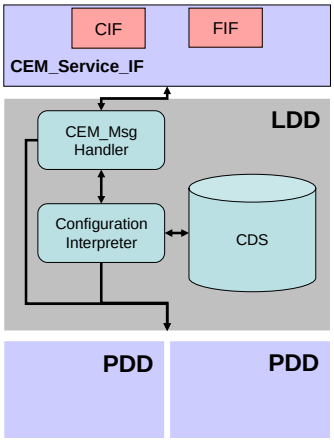

Figure 3: CEM Abstraction Layer CEM_Msg_Handler, Configuration Interpreter and CDS.

A. Object Orientated Architecture

The architecture facilitates an object-orientated approach. Abstraction and modularization through data hiding and encapsulation of structure and behaviour are emphasised. This allows more natural mapping of domain concepts to the reconfiguration architecture design. The creation of extensible frameworks through inheritance and polymorphism improves the reusability. Inheritance creates strong dependencies between a base class and its derived classes.

The device drivers must be loaded at the application side before access to the hardware can be obtained. Device drivers can be loaded and unloaded anytime. Devices can be used synchronously or asynchronously. Asynchronous device usage is preferred, since it is more CPU efficient. Partial reconfiguration of components can be achieved without downtime of other components which are not reconfigured, if this is supported by the underlying hardware. Therefore, the device driver at the kernel side library is often split into two libraries: LDD and PDD.

B. Logic Device Driver

The LDD, encapsulating the logical functions of a device e.g., on and off, and read and write, containing all the complexity of typical device usage usually in the form of a state machine. This part contains all common code. This is a polymorphism bound driver. Each CEM class has similar LDD per CEM. The LDD is not manufacturer dependant and collects CEM common parts/functionality.

The LDD hides the implementation of the function and allows CCM to set up parameters and new Functions without talking directly to the hardware. The LDD instantiates on demand and uses the operations supplied by the PDD to access the CEM-hardware.

The CEM LDD is also specific to that particular class of CEM. CEMs of a single class can share the same LDD package.

C. Physical Device Driver

The PDD is carrying out the functions on a specific device. This should be as thin as possible to improve portability to new devices.

It is the proprietary module inside the CEM Abstraction Layer. It contains manufacturer dependant parts and might disappear for standardised resources. A CEM could be replaced by a CEM from another vendor by just providing a new PDD for the ISA-device.

The PDD is supplied for each CEM and contains operations to configure the CEM (i.e. download a bit stream or object) as well as operations to write data to and from the resource. This includes writing instantiation and control data, but also configuring data channels for C-CEM.

IV. E²R COMPATIBLE DEVICES

A. Network on Chip (C-CEM)

Several flavours of C-CEMs are possible, depended on the targeted throughput and latency requirements. An E²R compatible reconfigurable modem does not limit the physical implementation. It has just to be able to carry out configuration transactions to be considered as a C-CEM. Implementations can consist of busses, networks, or direct connections.

For high throughput Network-on-Chip (NoC) components may very soon exist on many System-on-Chip (SoC) devices. The reason for this is that more and more communication partners on one chip exchange information with each other. The so far implemented bus structures can only connect a limited number of communication partners efficiently but they do not scale to higher numbers. A second problem comes from deep submicron technology where long global wiring become undesirable due to their low performance, higher power consumption and noise phenomenon [8]. Also, a large part of power consumption of the chip is consumed in the clock tree. In addition to that, clock skew is a very big problem in today's chip designs. It takes a great effort to solve these skew problems.

These difficulties can be solved with NoCs. A network may allow an asynchronous communication between the different parts of the system. These dedicated parts are clocked locally. That approach is called Globally Asynchronous Locally Synchronous (GALS).

In general, NoCs are comparable with existing Wide Area Networks in wide areas, for example, as used for inter-computer communication, but several aspects have to be done in other ways. In particular, the specification of the physical layer has to accommodate the on chip submicron effects and problems, which are different from other communication mediums.

Another fact is that protocols used for NoCs should have fewer overheads and enable a relatively simple implementation. This is required because otherwise large parts of the chip area will be spent just for the network.
In such a NoC case the CCM via the C-CEM would configure the routing table and manage Quality of Service (QoS) pre-settings.

B. RTOS services API for DSP-CEM (P-CEM)

To allow implementation of components of different RATs on a same DSP, it is necessary to define a standard Operating Environment (OE). An Application Programming Interface (API) for use in a modem to access the Real Time Operating System (RTOS) of such OE has been identified. Because the API is particularly suited to the implementation of modem components, it has been named the “Modem Interface towards RTOS”. It is part of the technical services supporting the modem components. miRTOS enables, through a POSIX-oriented profile, to give the modem waveform application abstraction from the RTOS available on the DSP CEM.

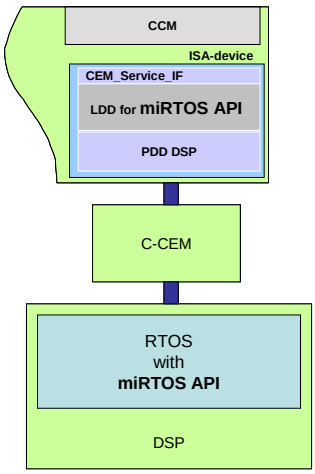

Figure 4: RTOS services API connected to the CEM hardware abstraction layer

A LDD will be provided for the RTOSes with miRTOS API to utilize the additional functionality of the DSP.

As shown in Figure 4, the PDD is provided for the DSP itself and the LDD has a logical connection to the miRTOS API.

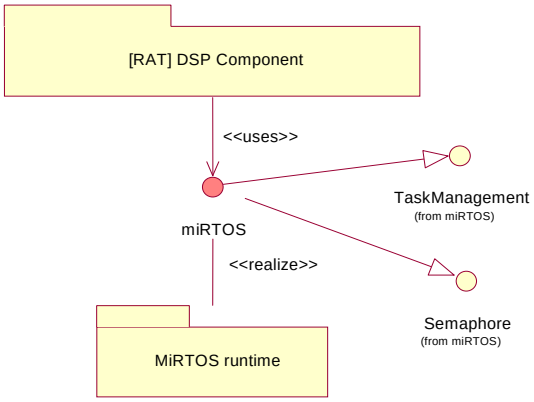

Figure 5: Definition of miRTOS interface

Figure 5 shows the *miRTOS runtime* package, which represents the software delivery realizing the miRTOS API on the processor hosting the component. The interface can be sub-divided into two main interfaces: *Task Management* and *Semaphore*, which are described below:

1) Task management

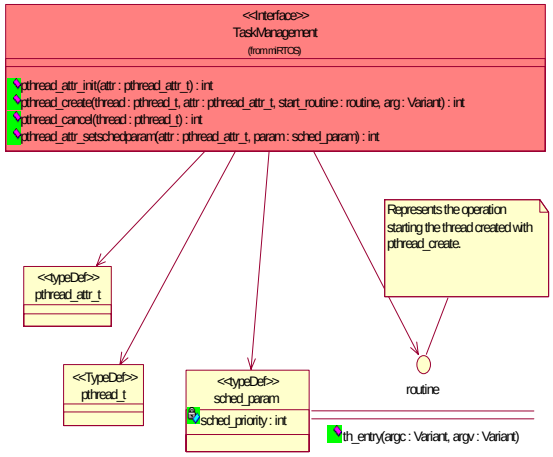

Figure 6: Interface TaskManagement (from miRTOS)

This interface (Figure 6) supports the creation of a task with the appropriate attributes (pthread_create). This interface also allows the cancellation of this created task (pthread_cancel).

In POSIX standard, the task (or thread) attributes are handled through an attributes object initialized with default values by the pthread_attr_init function before the creation of the task. Several tasks can share the same object attributes.

2) Semaphores

Semaphores are used to synchronize a task with an event or with another task, and also to send a message to a task.
MiRTOS semaphore functions permits:

- to create and initialize a semaphore,
- to use a semaphore,
- to destroy a semaphore.

A semaphore can be dedicated to the synchronization of a given task with another task or with an Interrupt Service Routine (ISR).

Semaphore functions can also be used to send a message to a given task. A memory space and a semaphore must be defined, dedicated to this task. The message to be sent is put in the memory space, and then the semaphore is released to notify the task of the message reception.

V. Configuration Process

Example: C-CEM Configuration

Here it will be described for the C-CEM how new configuration data is downloaded and how the enabled or embedded functions are configured.

The C-CEM is realized on a reconfigurable logic resource, the C-CEM resource. The main tasks in the configuration of the C-CEM resource are

1) *Configuration of the Ports*
2) *Configuration of the Interconnect Fabric*
3) *Reconfiguration of the Interconnect Fabric*

And later if sufficient reconfigurable resources are available on the C-CEM resource after realizing the desired C-CEM functionality, the CCM may configure a P-CEM in the surplus resources of the C-CEM resource.

The C-CEM ports are configured depending on the P-CEM connected to that Port. The port configuration can be one of a set of standard interface profiles (PCI, PCIx, AMBA, …) which will be stored in the configuration database of the CCM. For a particular port, the P-CEM's PDD provided by the manufacturer of the P-CEM installed on that port, can specify one of the standard configuration profile for the interface port or define its own user configuration. The specified port configuration will then be downloaded on to the C-CEM thro the Port Configuration interface (which will be part of the CIF in the CEM LDD) and configure the port, thus completing its physical interface to the ISA device.

After configuration of the C-CEM ports, the physical interconnect functionality is configured on the C-CEM resource. This configuration is again performed through the CIF interface of the CEM LDD. This configuration sets up the interconnect fabric which will be based on the data transfer and control requirements of the RAT being set up.

The dynamic reconfiguration of the interconnect fabric is performed through the specific interfaces [9] realized as part of the FIF interface.

The reconfiguration of the any P-CEM realized in the C-CEM resource is performed as any other P-CEM through the interfaces realized as part of FIF interface.

VI. Conclusion

An abstract configuration interface is needed for E²R devices in order to configure different RATs. Following an object oriented approach, a detailed architecture of the HAL has been presented, which provides an abstract service interface to higher layers for configuration of different CEM classes. The hardware abstraction needed between CCM and CEM can be achieved by means of logical and physical device drivers. Only the latter is aware of and specific to the underlying hardware / implementation of the CEM. A particular strategy for dynamic configuration of C-CEMs has been introduced.

The dynamic or partial reconfigurability, which can be achieved, independent from the underlying hardware components needs to be validated using SystemC models for various CEMs.

Acknowledgement

This work has been performed within the framework of the EU funded project E²R. The authors would like to acknowledge the contributions of their colleagues from the E²R consortium.

References

[1] ST-2003-507995 E²R Project, http://www.e2r.motlabs.com

[2] M. Bronzel, H. Seidel, J. Brakensiek, et al., "Functional Elements in E2E Reconfigurable Equipment", Proceedings of the 13th IST Mobile and Wireless Communications Summit (IST Summit). Lyon, France, 27.-30. June 2004.

[3] R. Burgess, S. Mende, "The Role of Configuration Data and a Configuration Control Module in an End-to-End (E²R) Software Radio System", IST Mobile Summit 2005.

[4] Software-Based Communication Domain Task Force http://sbc.omg.org/

[5] C. Dolwin, S. Mende, J. Brakensiek, "The Role of the Configuration Control Module in an End to End Reconfigurable System", SDR Forum Technical Conference, Phoenix, USA, 15-18.11.04.

[6] S. Naveen, B. Steinke, H. Seidel, et al. "Reconfigurable Modem Architecture and Its Abstraction in an End to End Reconfigurable Communication Network", WG6 Reconfigurability, WWRF#12 Meeting, Toronto, Canada, Nov. 04.

[7] MPI - Message -Passing Interface: http://www-unix.mcs.anl.gov/mpi/

[8] A. Jantsch, Hannu Tenhunen 'Networks on Chip', Kluwer Academic Publishers, 2003

[9] B. Steinke, H. Seidel, M. Halimic, and S. Naveen, "Hardware Abstraction Architecture based on Configurable Execution Modules for Functional System Chains", WG6 Reconfigurability, WWRF#13 Meeting, Cheju, Korea, March 05.