Radio Virtual Machine

Vladimir Ivanov (State University of Aerospace Instrumentation, Russia), Markus Mueck (INTEL, Germany), Seungwon Choi (Hanyang University, Korea), Heungseop Ahn (Hanyang University, Korea), Kyunghoon Kim (Hanyang University, Korea), Emilio Calvanese Strinati (CEA, France)

***Abstract*— Reconfigurable Radio assumes that the radio hardware platform is reconfigurable in such a way that the essential radio characteristics (carrier frequencies, spectrum bandwidth, modulation technique, coding technique, output power levels, and etc.) can be changed according to the flexibility of hardware platform that is defined by manufacturers. Corresponding radio hardware platforms are heterogeneous by nature and include computational elements with different level of programmability or reconfiguration. This paper addresses the problem of software portability and its generalization for reconfigurable platforms. Main contribution of this article is to present a novel architecture of the virtual domain-specific machine, namely, Radio Virtual Machine (RVM), which provides an efficient implementation of calculations related to radio signal processing in hardware platforms of different nature.**

Keywords—reconfigurable radio, virtual machine, VM, model of computations, MoC, concurency, heterogenous platform, Radio Access Technology, RAT

I. Introduction

From **a Radio Computer**'s point of view**,** wireless communications can be considered as a domain-specific embedded computing which deals with requirements and algorithms specific for wireless communications and related to radio waves processing. It's assumed that the radio computer interacts with the external world (radio spectrum) with a capability of receiving and emitting radio waves. The main purpose of this paper is to abstract the radio computer from hardware platforms by introducing a virtual machine without losing the efficiency of each particular implementation in hardware and software.

The most well known attempt to solve the problem of **Radio Computing** is the Software Communication Architecture (SCA) [1] standardized by the Joint Tactical Radio System (JTRS) for military applications, which adopts a middleware approach in order to abstract heterogeneous hardware from software. Unfortunately there are drawbacks preventing the wide applications of SCA for other than military uses. First of all, SCA doesn't solve the portability problem, as shown in, for example, [2] where the portability problem was pointed out as a primary problem of SCA. SCA (any version) middleware separates and isolates software from hardware and therefore doesn't allow a joint optimization of hardware and software which is the main source of efficiency for embedded devices. Although the middleware of SCA is quite sophisticated, it is too redundant and, thus, not efficient enough for commercial applications. The development of such middleware is quite costly for civil industry. Historically, SCA was designed based on the distributed computing approach, but the modern terminals are built based on System-on-Chips (SoC) where multiple Intellectual Property (IP) cores are integrated into a single chip. Still, since SoC-based technology does not assume distributed internal communications, it is not reasonable to support the baseline "client-server" model and sophisticated hardware agnostic transactions among software components. Meanwhile, since the "client-server" model proposed by the Object Management Group (OMG) is not formal; thus, it cannot support a formal verification which is critical due to the software complexity for emerging Multi-RAT (multiple Radio Access Technology) Mobile Devices working in a heterogeneous wireless network environment.

An alternative approach of a Radio Virtual Machine (RVM) was the Java Virtual Machine (JVM) [3], [4]. The implementation aspects of the RVM were studied in [5], [6]. But none of these sources treated the problem of an RVM architecture efficiency. The authors rather focused on existing VMs based on the von Neumann architecture and evaluated corresponding overheads. The Radio Computer concept was explicitly expressed in [7], [8] where the architecture, interfaces, hardware resource sharing strategies were described and implemented.

The new impulse for this activity, especially for the standardization of Reconfigurable Radio equipment architectures and interfaces, was provided by the new revision of the European R&TTE Directive [9] allowing 3rd party reconfiguration which has been approved by the European Parliament in 2014. Specifically, the ETSI Technical Committee on Reconfigurable Radio Systems (TC RRS) of the European Telecommunication Standard Institute (ETSI) has been developing standards on the architecture of reconfigurable radio equipment and interfaces required for the simultaneous operation of multiple RATs [10], [11], [17-20]. These standards are based on Radio Computing principles and exploit the RVM approach in order to separate any device reconfiguration from the underlying particular hardware. Details about the system and architecture aspects of this approach can be found in [12] – [14].

In this paper we provide a contribution for the development of a specific concurrent Model of Computation (MoC) and implementation of the MoC as a Radio Virtual Machine supporting the following new features:

- True concurrency;
- Combining message passing and shared memory architecture in arbitrary proportion;
- Vertical and horizontal parallelism;
- Minimal control overhead;
- Aggregation of RVM into more complex RVM;
- Adjusted set of operations.

In Section II, the baseline model of computation is presented. We use the term "scheme" for the MoC. The "denotational semantic" of schemes in terms of the Category Theory [15] is specified. The section provides the basic model definitions. Some quite straightforward statements without proofs are also provided in order to illustrate the main properties of the model.

In Section III the RVM details are specified. The proposed virtual machine implements the scheme behavior.

In the Section IV application of RVM in cellular communications is described. Section V finally gives a conclusion.

II. Model of Computation

A. Schemes

During this section the following notations are used. $\boldsymbol{Z_2}$ is a finite field containing two natural numbers {0, 1}. **Set** is a notation of the category of sets. If f is a map (or a function) than f|A denotes the restriction of f on the set A.

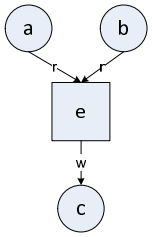


Fig. 1 Scheme example

Let's consider some set E with a discrete topology which we will call *operators* and some set S with discrete topology with the property $E \cap S = \varnothing$. The power set $P(S) = 2^S$ will be called *data*. The set E includes the distinguished element which is called the idle operator and denoted by $0 \in E$. We define the notion of "scheme" as a quiver (E, P(S), r, w) [16] when the set of vertexes is P(S), arrows are operators from E, the source function is r and the target function is w. Functions r and w can be interpreted as operation "read" and "write" data correspondingly.

Definition 2.1: An scheme **N** is a pair **N** = (r, w) where r and w are maps

$$E \underset{w}{\overset{r}{\rightrightarrows}} P(S)$$

Only for the idle operator 0, $r(0) = w(0) = \varnothing$; $\forall e \in E\ r(e) = {}^*e$ is called the *preset* of e, $Im(r) = {}^*E$ and $w(e) = e^*$ is called the *postset* of e, $Im(w) = E^*$.

Fig. 1 illustrates the case when E = {e} and S = {a, b, c}. The map r maps e onto {a, b} $\subset$ P(S) and w maps e onto {c} $\subset$ P(S). We can interpret e as an operator which read data a and b and writes the results of processing in data c.

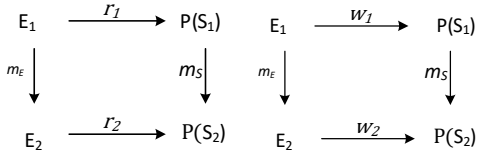
Domain E and codomain P(S) are fixed attributes for r and w so they won't be mentioned in the scheme notation if it doesn't lead to confusion. All maps considered below are the total maps unless defined otherwise.

Definition 2.2: The idle scheme **0** is the scheme for which E = {0} and Im(r) = Im(w) = $\varnothing$.

Definition 2.3: Let $\mathbf{N_1}=(r_1, w_1)$ and $\mathbf{N_2}=(r_2, w_2)$ are schemes then $\mathbf{N_2}$ is a *subscheme* of $\mathbf{N_1}$, notation $\mathbf{N_2} \sqsubseteq \mathbf{N_1}$, if $E_2 \subseteq E_1$, $S_2 \subseteq S_1$ and $r_2 = r_1|E_2$, $w_2 = w_1|E_2$.

Definition 2.4: Let $\mathbf{N_1}=(r_1, w_1)$ and $\mathbf{N_2}=(r_2, w_2)$ are schemes.

1. The scheme *morphism* $\mathbf{m} = (m_S, m_E): N_1 \to N_2$ is defined by maps $m_S: P(S_1) \to P(S_2)$ and $m_E: E_1 \to E_2$ with the following properties:
   a. $m_E(0) = 0$;
   b. the diagrams bellow commute for each r and w maps:



2. Scheme morphism $\mathbf{m} = (m_E, m_S): N_1 \to N_2$ is a scheme *monomorphism* (or epimorphism or isomorphism) if both m_E is monomorphism (or epimorphism or isomorphism) and m_S is monomorphism (or epimorphism or isomorphism) in the **Set** category**.**

The direct corollary from this definition is the following.

Corollary 2.1

1. For any scheme morphism **m**:
   a. $\mathbf{m(0) = 0;}$
   b. $m_S({}^*e) = {}^*m_E(e)$, $\forall e \in E$;
   c. $m_S(e^*) = m_E(e)^*$, $\forall e \in E$;
2. If **m**: $\mathbf{N_2 \to N_1}$ is the scheme morphism than $\mathbf{m(N_2) \sqsubseteq N_1}$.

Proposition 2.1

The collection of schemes and their morphisms create the scheme category $\boldsymbol{\mathcal{N}}$.

Proposition 2.2

1. The idle scheme **0** is a zero object in the category $\boldsymbol{\mathcal{N}}$.
2. The category $\boldsymbol{\mathcal{N}}$ is a category with zero morphisms $\boldsymbol{0_{NM}}$ for any **N**, $\mathbf{M} \in \boldsymbol{\mathcal{N}}$.

As a corollary from this proposition, Ker and Co-Ker objects together with related morphisms k and co-k exist in the category $\boldsymbol{\mathcal{N}}$. They are defined as corresponding equalizer and co-equalizer for any schemes **N** and **M** and morphism $\boldsymbol{f}$**:** $\mathbf{N \to M}$, (Ker, k) = equalizer($\boldsymbol{f}$, $\boldsymbol{0_{NM}}$) and (Co-Ker, co-k) = co-equalizer($\boldsymbol{f}$, $\boldsymbol{0_{NM}}$).

B. Scheme firings

First of all we define an action of $\boldsymbol{Z_2}$ on $\mathcal{N}$.

Definition 2.5: The $\boldsymbol{Z_2}$ -action of α on E and P(S) is defined as a map $\circ: \boldsymbol{Z_2} \times (E \cup P(S)) \to E \cup P(S)$, where for $\alpha \in \boldsymbol{Z_2}$, $(\alpha, p) \mapsto \alpha \circ p$ and if $p = e \in E$ then

1. $\alpha \circ e = 0$ if $\alpha = 0$;
2. $\alpha \circ e = e$ if $\alpha = 1$;
3. $\alpha \circ 0 = 0$, $\forall \alpha \in \boldsymbol{Z_2}$;

and if $p = d \in P(S)$ then

1. $\alpha \circ d = \emptyset$ if $\alpha = 0$;
2. $\alpha \circ d = d$ if $\alpha = 1$;

3. $\alpha\circ\emptyset = \emptyset\ \forall\alpha\in \boldsymbol{Z_2}$.

The action of α on E∪P(S) is designated as αE or αP(S) and αE ⊆E, αP(S) ⊆ P(S).

Definition 2.6: Let's fix a function α:E∪P(S) → $\boldsymbol{Z_2}$ than this function defines a $\boldsymbol{Z_2}$-action α on E and P(S). The action of α on the scheme N is defined as a map ∘: $\boldsymbol{Z_2}\times\mathbf{N}\to\mathbf{N'}$, $(\alpha, \mathbf{N}) \mapsto \mathbf{N'} = \alpha\circ\mathbf{N}$ where for $\mathbf{N}$ = (r, w) the scheme $\alpha\circ\mathbf{N}$ is the pair

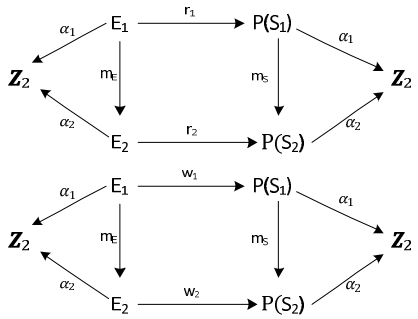
$$\alpha E \xrightarrow[w']{r'} \alpha P(S)$$

where r' and w' are defined as the followings:

- $r'(\alpha(e)\circ e) = \emptyset$ if $\alpha(e) = 0$ and $r'(\alpha(e)e) = r(e)$ in the opposite case;
- $w'(\alpha(e)e) = \emptyset$ if $\alpha(e) = 0$ and $w'(\alpha(e)e) = w(e)$ in the opposite case.

The action of α on $\mathbf{N}$ is denoted as $\alpha\circ\mathbf{N}$ or $\alpha\mathbf{N}$ and $\alpha\mathbf{N} \sqsubseteq \mathbf{N}$.

Definition 2.7: Let $\mathbf{N_1} = (r_1, w_1)$ and $\mathbf{N_2} = (r_2, w_2)$ are schemes with $\boldsymbol{Z_2}$ - action α_1 and α_2 than $m = (m_S, m_E)$, m: $\mathbf{N_1} \longrightarrow \mathbf{N_2}$ is a morphism of scheme with $\boldsymbol{Z_2}$-actions if m is scheme morphism and the following diagrams are commutative:



Proposition 2.3: Schemes with $\boldsymbol{Z_2}$-actions and their morphisms creates the slice category $\mathcal{N}/\boldsymbol{Z_2}$.

Definition 2.8: Let $\mathbf{N}$ = (r, w) be a scheme then

- $D_{ini} = {}^*E\backslash E^*$ is *initializing* data for $\mathbf{N}$;
- $E_{ini} = \{e\in E|{}^*e = \emptyset\}$ are *initializing operators* for $\mathbf{N}$;
- $D_{ter} = E^*\backslash {}^*E$ is *terminating data* for $\mathbf{N}$;
- $E_{ter} = \{e\in E|e^* = \emptyset\}$ are *terminating operators* for $\mathbf{N}$.

Definition 2.9: Let α_{ini} be a $\mathbf{Z_2}$ – action so that $\alpha_{ini}(p) = 1$ for $\forall p\in P\subseteq D_{ini}\cup E_{ini}$ and $\alpha_{ini}(p) = 0$ for other p then $\mathbf{N_{ini}} = \alpha_{ini}\circ\mathbf{N}$ is an *initial state* of the scheme $\mathbf{N}$.

Definition 2.10: Let α_{fin} be a $\mathbf{Z_2}$ – action so that $\alpha_{fin}(p) = 1$ for $\forall p\in P\subseteq D_{ter}\cup E_{ter}$ and $\alpha_{fin}(p) = 0$ for other p then $\mathbf{N_{fin}} = \alpha_{fin}\circ\mathbf{N}$ is a *final state* of the scheme $\mathbf{N}$.

Definition 2.11: Let $\mathbf{N}$ = (r, w) be a scheme then $\mathbf{Z_2}$ – actions $\{\alpha_t\}$ create a family of *firings* if it's defined by the following recursion. Let's denote ${}^*\Delta_{t+1}(e) = \alpha_t(e) - \alpha_t({}^*e)$ and $\Delta^*_{t+1}(e) = \alpha_t(e) - \alpha_t(e^*)$.

- $\alpha_0 = \alpha_{ini}$;
- if $\forall t>0$, α_t was defined then α_{t+1} is defined in the following way:
  - if ${}^*\Delta_{t+1}(e) < 0$ then
    - $\alpha_{t+1}(e) = \alpha_t(e) - {}^*\Delta_{t+1}(e)$,
    - $\alpha_{t+1}({}^*e) = \alpha_t({}^*e) + {}^*\Delta_{t+1}(e)$.
  - if $\Delta^*_{t+1}(e) > 0$ then
    - $\alpha_{t+1}(e) = \alpha_t(e) + \Delta^*_{t+1}(e)$,
    - $\alpha_{t+1}(e^*) = \alpha_t(e^*) - \Delta^*_{t+1}(e)$.
- $\alpha_{t+1}(e) = \alpha_t(e)$ for other $e\in E$ and $\alpha_{t+1}(d) = \alpha_t(d)$ for other $d\in 2^S$

Definition 2.12: Let $\mathbf{N}$ = (r, w) be a scheme then the family of firings $\{\alpha_t\}$ is closed if it is finite and the last element of the family is α_{fin}.

Proposition 2.4

Let $\mathbf{N_1} = (r_1, w_1)$ and $\mathbf{N_2} = (r_2, w_2)$ be schemes, $\{\alpha_t\}$ is firings of $\mathbf{N_1}$ and $\{\beta_t\}$ is firings of $\mathbf{N_2}$, $\mathbf{m} = (m_E, m_S):N_1\to N_2$ is a scheme morphism. If $\mathbf{m}(\alpha_0\mathbf{N_1}) = \beta_0\mathbf{m}(\mathbf{N_1})$ then $\forall t>0$ $\alpha_t = \beta_t\mathbf{m}$ and $\mathbf{m}(\alpha_t N_1) = \beta_t\mathbf{m}(N_1)$.

III. Virtual Machine

A. General Concept

The RVM will be defined in terms of particular schemes. We introduce a set of abstract resources consisting of Abstract Processing Elements (APE) and Data Objects. APEs represent operators described in the MoC above. They abstract computational elements executing any operation from the initially given set of operations which are typical for radio processing. Such a set is called *Radio Library*. Data Objects abstract the memory notion. The Radio Library provides a set of operations which can be carried out with data.

The specific RVM plays a role of the Turing machine in applications related to radio processing. The specific RVM is created for a given scheme according to the following procedure. The RVM allocates corresponding resources such as APEs and Data Objects for each operator and for data from the scheme. The operator semantics are provided by the Radio Library. Only one APE is allocated for each operator and only one Data Object is allocated for each data element. The RVM begins to work immediately after resource allocation and data initialization. All APEs work asynchronously and concurrently. An individual APE executes the allocated operator if all input Data Objects are full. Operation executions correspond to firings defined in the MoC. APEs access Data Objects with operations "read", "read-erase" (re), "read-erase-write" (rew) or "write" (w). After reading input data from Data Objects, the APE executes the allocated operator and, if output Data Objects are empty, then the APE writes processed data. Any full output Data Object blocks the corresponding writing operation. Full data objects and executing APEs are considered as active and they correspond to the schema with $\boldsymbol{Z_2}$–actions. After any writing operation, APEs are considered as inactive. Inactive APEs are returned to the APE's pool. Empty Data Objects which are not connected with APEs are also returned to the Data Object's pool.

B. RVM Architecture

Here we will consider the Radio Virtual Machine (RVM) to be a machine which is capable to execute any concurrent computations expressed by the specific RVM described above.

It plays a role of the universal Turing machine except for radio applications. The architecture of such RVM is represented in Fig. 2. The block "Basic Operations" is downloaded from the Radio Library. Each operation from this block is identified by the corresponding operational code generated as the hash value of the Radio Library operator identifier. The Control Unit (CU) generates configuration codes. The data path of the RVM consists of Data Objects, Abstract Processing Elements and the Abstract Switch Fabric (ASF). Each Data Object (DO) has a unique number. The DO is shown in Fig.3. DOs are configured by the instruction **DO_config** with parameters **Init** and **Set.**

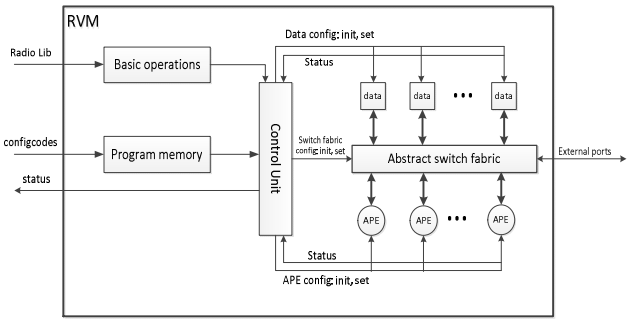


Fig. 2 RVM architecture

The **Init** parameter specifies a method for initializing DOs. It might provide an immediate value or an identifier of the initialization procedure (depending on the implementation). The **Set** parameter defines the DO attributes such as the DO's size (bytes), the number of read/write ports and access time in ns. DOs communicate with APEs through the ASF by the data interface consisting of the data enable signal de, the access type ac ∈ {"r", "re", "rew", "w"} and data. The data enable signal de = 1 just after DO initialization.

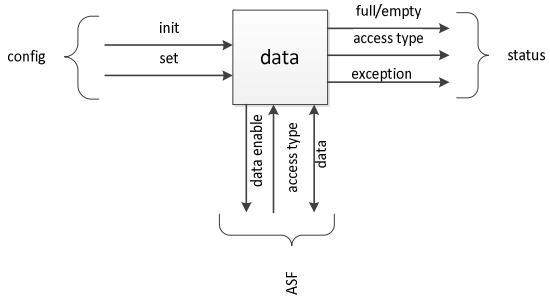


Fig. 3 The Data Object and its interface

APEs establish and convey "access type" (ac) to DO's ports. They dynamically access DOs by read/write operations ("r", "re", "w") during the executed operation "op". The DO's content is available for reading when de = 1 and ac = "r" or "re". The empty DO can be written if de == 0 and ac == w. The signal ac is set by the APE dynamically when the related read/write operation is started in the APE. The internal structure of DOs is recognized by APE's operations. The write operation is blocked when the DO is full. It means de == 1. The status interface provides the DO's status information to the CU and consists of state, access type and exception signals. The state signal=1 if the DO is "full" and 0 in the opposite case, access type=0 in case of the read operation ("r" or "re") and 1 for the write operation.

The APE is shown in Fig. 4. All APEs are numbered and has the following attributes: the number of ports, the execution cost and the time constraint. The execution cost is any number which might be characterized as execution time (ms), power consumption (average or peak, mW), complexity (number of cycles), MOPS/mW. The time constraint is determined by the worst-case activation time of APE on receiving each task. APE's ports connect the APE to the ASF and include data interface with lines: de, ac and data. The APE is configured by the instruction **APE_config** with parameters **Init** and **Set.** All APE's ports are numbered. The **Init** parameter provides the op code operation from the Basic Operations. The APE is moved to the active state with corresponding indication to CU just after the **Init** procedure. The **Set** parameter defines APE's attributes. The APE is inactive when it is not configured and doesn't execute any operation. Status information is delivered after an operation completion and it might be: no exception, configcode flow change, arithmetic overflow/underflow and incorrect operation.

The Abstract Switch Fabric connects APEs and Data Objects. One DO can be connected with multiple APEs. One APE can be connected with multiple DOs. The ASF consists of data ports (internal or external) and processing ports. Data ports connect the ASF and DOs which might be internal DOs or external DOs via the interface lines. Processing ports are connected with APEs via the interface lines. The CU configures the ASF by the **Set** and the **Init** instructions. The **Set** instruction creates data and processing ports. The **Init** instruction establishes the internal connectors between data and processing ports and connects these ports with DOs and APEs.

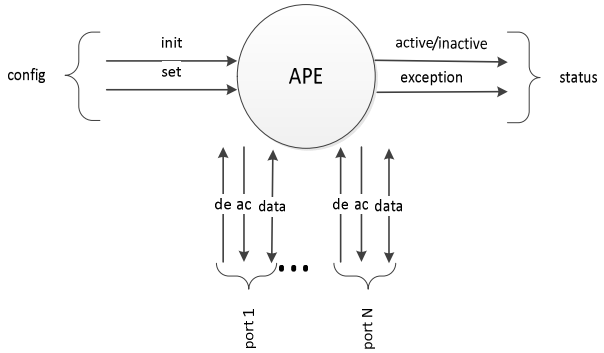


Fig. 4 The Abstract Processing Element and its interfaces

Configcodes are stored in the program memory. Each configcode includes configuration codes for DOs, APEs and ASF. It also includes additional control flags like Last Config Flag (LCF), Next Address Flag (NAF) and optional Next Configcode Address Offset (NCAO). Last Config Flag = 1 if the configcode is the last code in the task. The field NCAO is augmented if NAF = 1.

The Control Unit handles task execution and consists of the parts shown in Fig. 5. The Configuration Counter (CC) points out on the first configcode in a task. The Config Fetcher fetches configcodes from the program memory. The Next Address Block (NAB) calculates the next configcode address.

The Datapath Configuration Block configures the data path. CC is set up at the beginning of the task and is updated after the current configcode execution by NAB as CC = CC + |CC| when no exceptions, where |CC| is the length of the configcode with address in CC and CC = CC + NCAO if a configcode flow change is occurred. NAB detects the end of the task after completion of all configcodes. NAB generates the status signal which might be active, inactive or exception. RVM is active if both the Program Memory and the Basic Operations block were downloaded by corresponding configcodes.

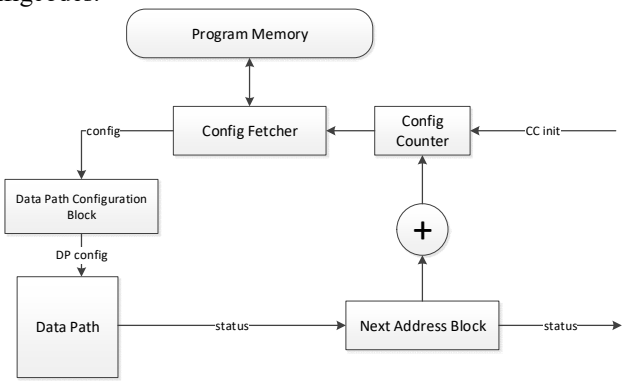


Fig. 5 Control Unit structure

One or a few RVM might be a part of the bigger RVM. In such case the component RVM behaves as the APE and is connected to the RVM ASF by external RVM ports. The component RVM is configured as the APE. The task identifier and corresponding configcodes are downloaded into the Basic Operations block and is taken out during the RVM initialization.

IV. Application of RVM to Cellular Communications

The integration of an RVM into a (Cellular) Modem or Base Station (BS) / Access Point (AP) Platform (based for example on 3GPP 2G/3G/4G, IEEE 802.11a/b/g/n/ac/ad/HEW, WiFi for TVWS, etc.) can be typically done based on the principles outlined in the sequel.

A. Fully programmable platforms

A part (or all) of the digital signal processing is performed within the RVM. The RVM gives full flexibility to the Software Developers for modifying features on the Physical Layer and MAC Layer. This framework still allows for the RVM to have access to (e.g., ASIC based) accelerator components but their usage is optional.

A corresponding Mobile Device Modem architecture is thus designed as follows:

- Example 1: All base-band processing tasks, as well as some tasks of higher layers, are performed in the RVM. The RVM represents a safe execution environment providing mechanisms such as error handling and thus facilitates device (re-)certification;
- Example 2: Some base-band processing tasks, as well as some tasks of higher layers, are performed in the RVM. The "Hard-Wired" part can apply all types of modem design approaches, including fully hardwired solutions, flexible (re-programmable) solutions that allow modification through updates of the firmware, etc. The main difference to the RVM part lies in the fact that the "hard wired" part is not accessible to modifications through RVM code. Also, the RVM environment typically provides safety measures (e.g., handling of exceptions and errors, etc.) which are not part of the hard wired part.

B. Partialy programmable platforms

The RVM is used in order to replace selected base band components, while others still apply the original manufacturer's (e.g., ASIC based) design. The Software Developer is thus able to modify some specific features to which the manufacturer allows access via the RVM. Over the life-time of a product, the manufacturer may choose to open the platform to reprogramming of further components; alternatively, the possibility to alter some specific components may be terminated, for example in case of complications with other device components.

For those base-band blocks which are replaced by designs that are executed in the RVM, the corresponding signalling flow is lead through the RVM while the original hard-wired block is de-activated. Typically, the manufacturer will allow for the replacement of some specific functionalities by RVM 3rd party Software designs, such as antenna selection mechanisms, MIMO encoding/decoding functionalities, forward-error-correction encoding/decoding, etc.

C. Hybrid implementations

A hybrid approach can be applied using a combination of the upper cases. Indeed, the previous cases enable the RVM to replace some or all hard-wired Modem components which were originally designed by the Manufacturer. In a hybrid implementation, the RVM is used to provide some novel functionality that is not yet provided, for example the RVM can be used in order to perform additional encryption / decryption of User Data which is originally not planned to be implemented in the mobile device platform. In this case, in the uplink the data flow of User Data from upper layers is used as an input to the RVM, the RVM performs all encryption operations and feeds then the encrypted data back for further processing. In the downlink the data flow containing received, encrypted data is used as an input to the RVM, the RVM performs all decryption operations and feed then the decrypted data back for further processing.

V. Conclusion

Following the revision of the R&TTE Directive in Europe, the RVM approach proposed in this document is expected to serve as a key enabler for introducing Software Reconfiguration features, in particular the provision of Radio Applications ("RadioApps"), into the market. The RadioApps concept will enable 3rd Party Software Developers to access to

radio parameters (typically on the PHY and MAC layer). Obviously, software reconfiguration features also require the introduction of novel mechanisms for equipment certification. Corresponding architecture and certification standards are currently under development in the ETSI RRS standardization Technical Body. It is expected that those technology solutions will trigger a world-wide proliferation of the corresponding technology.

Acknowledgement

This work was partially supported by Institute for IITP grant funded by the Korea government (MSIT) (No. 2017-0-00723, Development of Software-defined Service-oriented Integrated base station platform using Reconfigurable Radio System Technology.

References

[1] SCA 1.0, JTRS Standard, 2002, last ver. 4.0, Feb, 28, 2012, http://jtnc.mil/sca/Pages/default.aspx

[2] Wireless Innovation Forum Top 10 Most Wanted Wireless Innovations, Document WINNF-11-P-0014, Ver. V1.0.1, October, 7, 2011

[3] "Software Radio Architecture: Mathematical Perspective" by J. Mitola, III, IEEE Journal on Selected Areas in Communications, Vol. 17, No. 4, April 1999

[4] "The Radio Virtual Machine" by M.Gudaitis and J.Mitola III, Oct 31, SDR Forum Workshop, 2000

[5] "The Radio Virtual Machine: A Solution for SDR Portability and Platform Reconfigurability" by Riadh Ben Abdallah, Tanguy Risset and Antoine Fraboulet, IEEE International Symposium on Parallel & Distributed Processing, May 23 – 29, 2009

[6] "Virtual Machine for Software Defined Radio: Evaluating the Software VM Approach" by Tanguy Risset, Antonie Fraboulet, Jerome Martin and Riadh Ben Abdallah, Interbational Conference on Embedded Software and Systems, ICESS, 2010

[7] "Multi-Radio Scheduling and Resource Sharing on a Software Defined Radio Computing Platform" by Kees van Berkel and et.al., Proceedings of the SDR'09, 2009

[8] "Multi-radio coexistence and collaboration on an SDR platform" by Antii Piiponen and et.al., Analog Integr Circ. Sig. Process., Springer, Vol.69, 2011

[9] Proposal for a Directive of the European Parliament and of the Council on the Harmonisation of the Laws of the Member States Relating to the Making Available on the Market of Radio Equipment, Brussels, 17.10.2012, http://eurlex.europa.eu/LexUriServ/LexUriServ.do?uri=COM:2012:0584:FIN:EN:PDF

[10] ETSI EN 303 095 V1.2.1 (2015-06), Reconfigurable Radio Systems (RRS); Radio Reconfiguration related Architecture for Mobile Devices

[11] ETSI EN 303 146-1 V1.2.1 (2015-11), Reconfigurable Radio Systems (RRS); Mobile Device Information Models and Protocols; Part 1: Multiradio Interface (MURI)

[12] Radio Computer Architecture, Regulatory and Security Framework for Next Generation Wireless Mobile Communication, IEEE Wireless Communication Magazine, Vol. 19, Issue 4, 2012 , pp. 9 - 16

[13] "Reconfigurable Radio: Architecture and Programming Model" by Vladimir Ivanov, International Workshop on CR, Seoul, Korea, l, September, 2012

[14] "Reconfigurable Radio Systems as Enabler for Exploiting the Future Heterogeneous Wireless Communications Landscape" by Markus Mueck and et. al., ETSI RRS Workshop, Sophia Antipolis, France, 12, December, 2012

[15] "Category Theory for Computing Science"by Michael Barr and Charles Wells, Prentice Hall, 1990

[16] "Quiver Representations" by Harm Derksen and Jerzy Weyman, Notices of the AMS, February, 2005, vol. 52, N. 2, pp. 200 – 206

[17] ETSI EN 302 969 V1.2.1 (2014-11), Reconfigurable Radio Systems (RRS); Radio Reconfiguration related Requirements for Mobile Devices

[18] ETSI EN 303 146-2 V1.2.1 (2016-06), Reconfigurable Radio Systems (RRS); Mobile Device (MD) information models and protocols; Part 2: Reconfigurable Radio Frequency Interface (RRFI)

[19] ETSI EN 303 146-3 V1.2.1 (2016-08), Reconfigurable Radio Systems (RRS); Mobile Device (MD) information models and protocols; Part 3: Unified Radio Application Interface (URAI)

[20] ETSI EN 303 146-4 V1.1.2 (2017-04), Reconfigurable Radio Systems (RRS); Mobile Device (MD) information models and protocols; Part 4: Radio Programming Interface (RPI)