MULTI-RADIO SCHEDULING AND RESOURCE SHARING ON A SOFTWARE DEFINED RADIO COMPUTING PLATFORM

Kees van Berkel, ST-Ericsson, Adv. R&D, kees.van.berkel@stericsson.com
Artur Burchard, NXP Research, artur.burchard@nxp.com
David van Kampen, ST-Ericsson, Adv. R&D, david.van.kampen@stericsson.com
Pjotr Kourzanov, NXP Research, peter.kourzanov@nxp.com
Orlando Moreira, ST-Ericsson, Advanced R&D, orlando.moreira@stericsson.com
Antti Piipponen, Nokia Research Center, antti.piipponen@nokia.com
Kalle Raiskila, Nokia Research Center, kalle.raiskila@nokia.com
Sverre Slotte, Nokia Research Center, sverre.slotte@nokia.com
Marinus van Splunter, NXP Research, m.van.splunter@nxp.com
Tommi Zetterman, Nokia Research Center, tommi.zetterman@nokia.com

ABSTRACT

Beyond multi-standard operation, SDR for consumer handsets should also aim at multi-radio operation: supporting e.g. HSPA, DVB-T, and WLAN active simultaneously on a shared hardware platform ("radio computer"). In essence, we propose an SDR operating system that provides a virtual platform for individual radios. By means of a technology demonstrator we address dynamic multi-radio operation, incl. key challenges such as unifying critical interfaces, resource management under real-time constraints, and efficiency.

1. INTRODUCTION AND SDR VISION

A smartphone today typically has separate baseband hardware resources for 2G, HSPA, WLAN, BT, GPS, and FM. Wireless communication standards (cellular, connectivity, broadcast, and positioning) are still evolving *and* diversifying rapidly. Roadmapping chipsets for such dynamic markets has become very challenging.

In response to this diversity, digital wireless transceivers reuse standard hardware components like microcontrollers and buses. Hardware architectures tend to converge towards "heterogeneous multi-core", where individual semiconductor companies internally standardize on cores, interconnect, memory, debug, and power management [1].

A next step on the "SDR staircase" [Fig. 1] is multi-mode combos: baseband architectures that can be configured for different wireless standards in run-time. Supporting multiple radios simultaneously ("multi-radio") could be based on multiple baseband architectures, both HW and SW defined ones. By sharing hardware resources during run-time

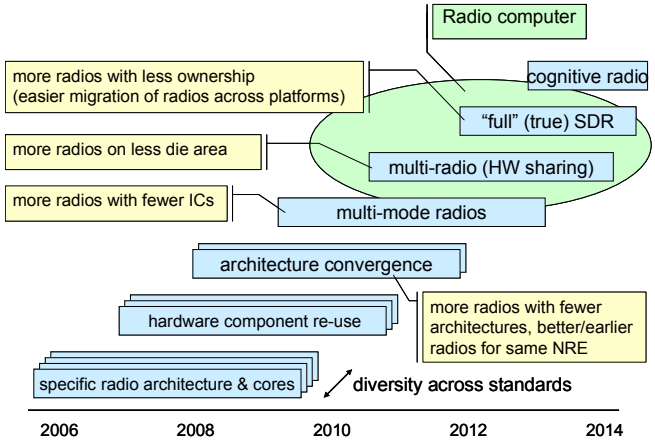

Figure 1 – SDR introduction and evolution

die area can be saved. In the most extreme case, a single hardware platform supports multi-radio by dynamically loading, starting, and stopping individual radio applications, not unlike computer applications. Hence, we propose to call such device a *radio computer* [Fig. 2]. On top of it a *radio operating system* manages all hardware resources.

Resource management of multiple (radio) applications on a (heterogeneous) multi-core architecture under hard real-time constraints (tens of µs to 1 ms) is an open research problem. We believe this is only possible when *all* interfaces are carefully unified and restricted [Fig. 2]:

1. the interface between the user applications and the radio OS ("Multi-Radio Access Interface", MURI);
2. the interface between the radios and the radio OS ("Unified Radio Application Interface", URAI);
3. a "Radio Programming Interface" (RPI) including the programming model for the radio baseband processing;
4. the interface to the RF transceivers (not shown).

Interfaces 1-3 are considered by ETSI for standardization [2].

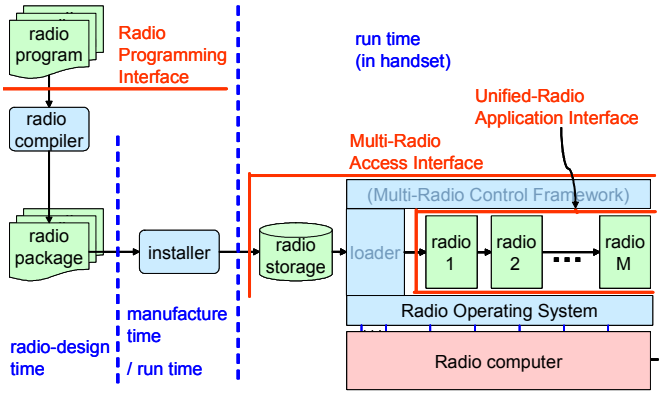

Figure 2 – SDR design/run-time architecture

The RPI leads to a degree of decoupling of radio applications (software) from the underlying hardware. This decoupling allows hardware evolution (improved micro architectures, faster CMOS nodes) to be managed independently from software evolution (improved algorithms, new features, next generation standards). Over time, the lifetime of and investments in radio software can easily exceed those of the radio hardware. With open standards, it is conceivable that radio applications will be provided by third-party vendors ("independent radio-software vendors").

Ideally, a radio compiler accepts a generic (platform-neutral) radio application (software) and generates a so-called *radio package* comprising the code and configurations of all hardware components. In practice, a degree of manual tuning and optimization towards the specifics of the radio computer will be required in order to meet strict cost and power targets.

The key R&D challenges towards this multi-radio SDR vision are:

- a radio OS and SDR Functional Architecture providing a unified interface to diverse radios and vice versa;
- efficient resource management, supporting multi-radio operation, while respecting the hard real-time constraints of the individual radios;

such that radios can be designed and verified independently of one another.

This paper describes a multi-radio architecture as well as an SDR technology demonstrator. The radio computer scope is limited to baseband subsystem. In Section 2 we elaborate the above challenges and describe our multi-radio SDR architecture in more detail. The actual demonstrator and five demonstrations are described in Section 3. In a concluding section we assess the results and look forward to future work.

2. MULTI-RADIO SDR ARCHITECTURE

The large number of radios and the even larger number of radio combinations makes it impractical to optimize resource management for fixed radio combinations. Hence, it must be possible to design and verify radios independently. Radios running simultaneously on a radio computer share hardware resources, but should not interfere with the correct operation of one another.

2.1. SDR functional architecture

For our functional architecture we have developed a unified view on a diversity of radios [3]. Individual radios follow a life cycle, that is, a sequence of *administrative states*: uninstalled, installed, loaded, and active. Furthermore, an active radio can be in one of several radio-specific *operational states* (characterized by it communication behavior and resource use), with a specified set of allowed state transitions triggered by the user or across the radio link. Examples of operational states are: idle, synchronizing, camping, and communicating.

The MURI[1], see also Fig. 3, provides services to support transitions of administrative states (e.g. Install Radio, Load Radio) as well as connection management and flow control services (e.g. Start Scanning, Associate, Add Flow).

The URAI[1] defines the corresponding services for individual loaded radios and resource management services. Typical URAI services of radio systems include: Start Scanning, Associate, Synchronize Radio Time [4, 2].

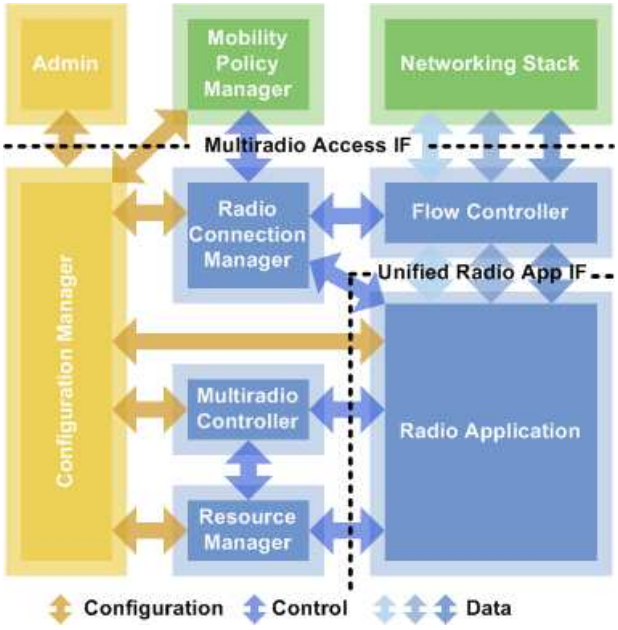

Figure 3 – SDR functional architecture

[1] In [4] MURI and URAI were called MRAI and URSI.

Resource requirements in the active state of a radio may differ considerably, so partitioning the radio operation into operational states enables efficient resource sharing among multiple radios. The Resource Manager's Change Operational State service admits a new state only when the foreseen resource needs can be met by the available resources, as elaborated below.

When radios exhibit rather coarse-grained burstiness (e.g. DVB-H bursts, GSM time slots, WLAN packets, etc) a form of coordination may help to avoid spectral interference, or to allow sharing of e.g. RF resources. The Radio Access service of the Multi-Radio Controller (MRC) provides such coordination.

2.2. Multi-radio resource management

The key to real-time multi-radio resource sharing is the notion of *resource budget*, capturing the resource needs (processor and accelerator loads, memory usage, interconnect bandwidth occupation, etc.) of a radio for each operational state. At compile time these budgets are calculated as tightly as possible. In run time these budgets form the basis for admission control and isolation. [5]

These budgets have the form of vectors of resource needs. Admission control of an additional radio is based on comparing the combined resource needs specified by the active plus new resource vectors with the available resources. This scheme assumes that resource fragmentation and overheads can be accounted for accurately.

Radios differ widely in their granularity of basic tasks. E.g. OFDM symbols for broadcast standards like DVB-T and connectivity standards like 802.11g differ by two orders of magnitude. Worse even, latency constraints for 802.11g are expressed in the µs range, where natural task durations of other radios may be in the ms range. Hence for larger tasks a form of preemption is required. Accordingly, the most fine-grained radio dictates the overall timing granularity, and hence determines the task switching and scheduling overheads. Efficiency dictates that scheduling is as much as possible performed at compile time.

2.3. Radio programming model

A programming model for the real-time part of a radio must:

1. be sufficiently expressive to allow concise descriptions of physical-layer processing of a large variety of radios;
2. allow the calculation of tight budgets and providing guarantees that radios will respect these budgets;
3. be platform neutral.

Synchronous Data Flow (SDF) [6] is a promising candidate with its explicit description of parallelism and its data-independent communication behavior. With worst-case execution times for its atomic actions, it can be analyzed rigorously for guaranteed minimum throughput [7] as well as for maximum latency [8]. Unfortunately, SDF is not sufficiently expressive to capture forms of data-dependent behavior, e.g. when synchronizing on a packet header or when processing packets of different payload sizes. In order to support such mild forms of data-dependence at a relatively coarse grain size, we have introduced specialized constructs to support so-called modes and mode transitions [5]:

- a *mode switch* directs a token to one of several outputs specified by a value at its control input port,
- a *mode select* can forward a token from one of several inputs specified by a value at its control input port,
- a *tunnel* connection asynchronously shares data between actors in different modes.

This resulting computation model is called *mode controlled data flow* (MCDF), and can be analyzed rigorously [5]. An example of an MCDF graph is given in Section 3.2. The proposed Radio Programming Interface can be seen as a component model supporting MCDF combined with the API of URAI services.

2.4. Radio computer

The computational load required for the radios in a smart-phone today amounts to many tens of GOPS, counting algorithmic operations (typically multiply, add). This load will quickly grow to several 100s of GOPS when LTE is included. With a power budget of only several 100s of mW, a heterogeneous multi-core architecture is unavoidable [1]. Promising architectures are typically based on low-cost microcontrollers, powerful SIMD machines [9, 10, 11, 12], and configurable accelerators.

Given the heterogeneity of the hardware architecture, it is essential to provide a largely homogeneous abstraction towards the resource manager, for example based on:

- budget-based schedulers for the programmable and/or configurable cores,
- FIFO support for the MCDF inter-actor synchronization and communication;
- memory management for all local and shared memories.

Depending on the specifics of the hardware, these mechanisms can be implemented in a more centralized form or in a (partially) distributed form.

Ideally, the run-time isolation of different radios is enforced through mechanisms like preemption and MMUs. Only then it can be avoided that a misbehaving radio interferes with another radio. For the short term such mechanisms may well prove too costly.

3. SDR TECHNOLOGY DEMONSTRATOR

The goal of our SDR technology demonstrator is to show:

- that multiple radios can be dynamically installed/ loaded/started/stopped/unloaded, through a unified multi-radio access interface;

- that multiple radios can run in a coordinated fashion;
- that individual radios respect real-time constraints;
- and that hardware resources are shared efficiently.

This section introduces the hardware platform and describes a sequence of five demonstrations towards this goal.

3.1. Hardware platform

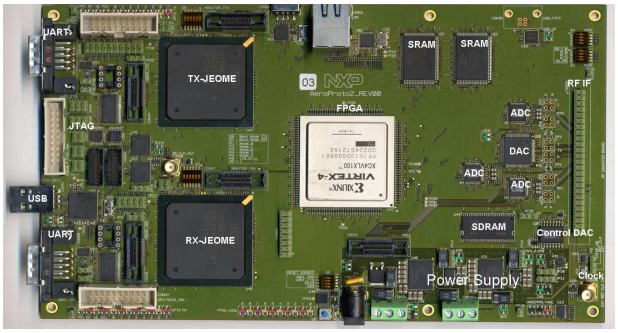

Figure 4 – Photograph of the hardware platform

The hardware platform consists of a PC (running Linux) connected to a board developed for prototyping wireless transceivers. This board (Fig. 4) comprises 2 EVPs [11] plus 3 ARM processors. See also Fig. 7. Channel decoding ASIPs are not present, so they are supposed to be hosted from the ARM. The available version of the EVP does not support interrupts, and hence no preemptive scheduling. Two of these boards can be connected by a flat cable to emulate bi-directional radio traffic.

3.2. Modeling radios and their resource usage (Demo 0)

In our technology demonstrator the various radios have not been programmed in full detail. Instead, so-called *radio resource models* capture all essential interfaces as well as all run-time resource needs (cycle counts, memory footprints, etc.) for a given resource allocation. The non-RT protocol models have been modeled with Rhapsody as communicating state machines. The RT baseband models are programmed as MCDF graphs in an input format called LIME [13]. Each node ("actor") component is programmed in C extended with an API to support the MCDF primitives (e.g. selective multi-ports). An actor can be annotated with a *worst-case execution time (WCET)*, assuming mapping on a particular core. The data-flow graph is described in XML.

Fig. 5 shows a somewhat simplified resource model of a WLAN receiver. While "parsing" the input sample stream, the receiver switches along four modes: packet synchronization (1); header processing (2); payload processing (3); CRC checking (4). Tunnels (T) are used e.g. to pass header information from header analysis to payload demodulate actor.

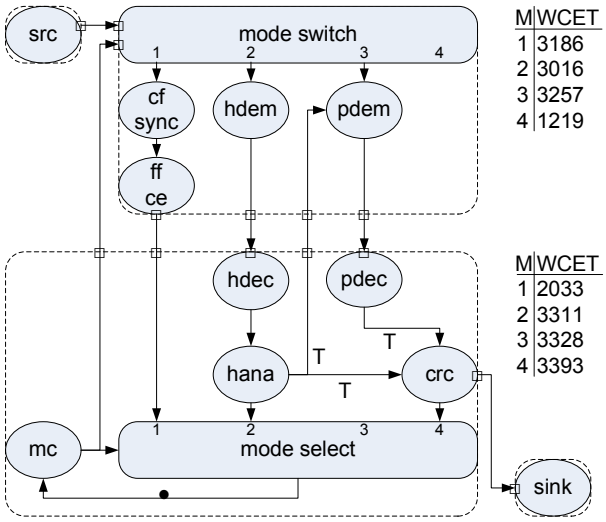

Figure 5 – MCDF graph of a WLAN receiver

Detailed resource models have also been created for DVB-T, DVB-H, TD-SCDMA, and LTE. In all cases our MCDF programming model proved sufficiently expressive.

3.3. Compiling a radio (Demo 1)

The radio compiler [5] accepts the resource model and produces a radio package, comprising of a collection of *dynamically loadable* executables for the various HW cores (including relocation information) and a transceiver configuration file including the resource budget, see Fig. 6.

The Heracles scheduler accepts an analysis model (obtained by parsing the LIME graph, actor components and an actor-to-core assignment), temporal constraints and the hardware platform description. It:

- statically schedules actors that run on the same core and clusters them into a single task to reduce overheads for scheduling and communication; in Fig. 5 dashed lines indicate task clusters;
- computes scheduler settings, and FIFO sizes;
- does rigorous temporal analysis of the MCDF graph;

In Fig. 5 the mode-specific WCET in ns for the EVP (top) and ARM (bottom) cluster tasks is provided. Analysis shows that the application meets its RT constraints (assuming sin gle radio and round-robin schedulers).

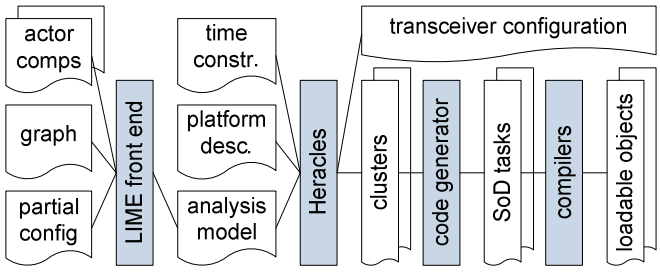

Figure 6 – Radio compiler

The mode sequence *M1, M1, M2, N×M3, M4* takes 26.74 µs, with *N* being 1 for the shortest packet, which is less than the duration of the packet plus the SIFS latency constraint, starting the processing after one OFDM symbol period: *12 + N×4 + 16 - 4* µs.

Per cluster the code generator creates a task shell that calls the actor functions and handles all inter-task communication using the FIFO library of the SoD ("Sea of DSP") streaming framework on which the tasks will run (see Section 3.4). It further efficiently generates MCDF constructs like mode switches with its conditional execution of actors.

3.4. Creating and running a single radio (Demo 2)

Fig. 7 summarizes the run-time software components of the baseband subsystem. On the EVPs and J-ARMs radio tasks run on top of SoD streaming kernels, which provide round robin scheduling and FIFO communication support. At load-time of a radio, SoD tasks are created and added to the schedulers from a function preloaded in the core's instruction memory. Further, ports of the created tasks are connected by FIFOs. This is done by Baseband Resource Manager (BBRM) calling the SoD Network Manager (NM based on the transceiver configuration in the radio package). The NM, BBRM, and other non-real-time functions run on the F-ARM on top of µC/OS. The dynamics of BBRM interfacing with the components in the SDR SFA (largely mapped on a PC in our demonstrator) will be discussed in Section 3.5.

SoD is a light-weight OS. The code size of the ARM kernel is 1.6 KB, for the NM it is 16.8 KB. The overhead of task switching in the round-robin scheduler is 30 clock cycles on the EVP. The overhead of calling the SoD synchronization functions is around 2×30 cycles per port for an ARM task. This synchronization overhead is negligible for slow-paced, long packet processing radios like DVB-T (<3% for 2K mode). For the WLAN ARM cluster (bottom of Fig. 5) in payload processing mode the two active ports give a synchronization overhead of 116 cycles, i.e. 44% of the total execution time. This percentage improves when the ARM runs at a higher clock frequency, as the WCET of this mode includes a fixed 930 ns on an assumed hosted channel decoder ASIP. The sync overhead can be reduced to 59 cycles by inlining these functions in the shell.

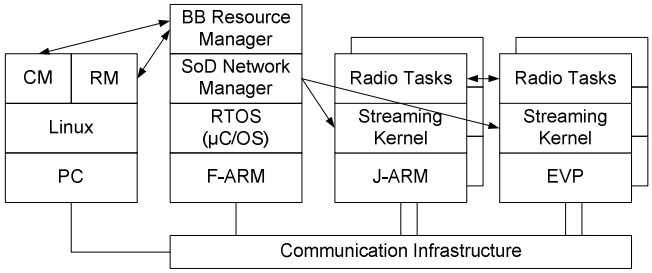

Figure 7 – Run-time software

3.5. Loading and running multiple radios (Demo 3)

The SDR functional architecture (Fig. 3) with its system components and services has been specified in UML using Telelogic Tau [4]. Based on this specification the system components have been prototyped in Telelogic Rhapsody using its UML to C++ code generator. The SFA system components run on the Linux PC as depicted in Fig. 7.

Using a menu-based GUI, the services of the MRAI are called. For example, the Configuration Manager (CM) is invoked to install a radio package into a radio repository on the host PC. After installation, the loading of a radio is requested to the Resource Manager via the CM. The loading of a radio will go through the admission control procedure of subsystem resource management (e.g. BBRM). After successful admission check, the resources are allocated (memory, scheduler settings) and the radio is created (for BB in SoD as explained in Section 3.4). Dynamic loading in the baseband subsystem includes the fetching of loadable objects from the radio repository (via CM), relocating its sections to the allocated memory segments and linking unresolved references in the code (e.g. to the SoD libraries).

A loaded radio can be activated by the Radio Connection Manager through the GUI. In the next phase scanning for communication peers is requested (e.g. WLAN peers in an ad-hoc network). With any found peer an association can be created. Then a data flow is mapped onto such association. The GUI interfacing with the Flow Controller allows to stream different forms of content (e.g. audio, video) over these flows.

With our demonstrator we can for example sequentially set up three radio pairs on two connected boards with separate flows (while the activated ones already stream data we add other radios without disruption) and finally they all run simultaneously without interference.

BBRM prototype requires 24.5KB of instruction memory. The data memory dominates its memory footprint, with a size of 49.8KB, of which 28.6 KB are spent on caching configurations and job information. For a radio containing 16 tasks and 22 FIFOs, the loading of a job takes 201 ms, 196 ms of which were taken by memory allocation.

3.6. Controlling multi-radio (Demo 4)

The MRC (see Fig. 3) plays a key role in solving radio co-existence conflicts (e.g. spectrum interference) but can also be used for sharing resources e.g. the RF transceiver. It does so through dynamic scheduling of the radio spectrum access requests from the individual radios. In the demonstrator setup, the boards have no RF transceivers. Instead, two boards were connected together by a cable, mimicking a shared RF transceiver resource. The transmitting radio computer executes three radios, which are used by three different user applications of different priority (radio 1: RT video

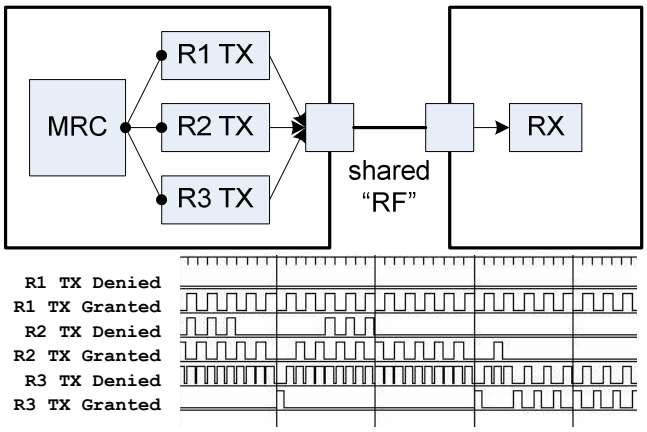

Figure 8 – A MRC use case (top) and execution (bottom)

stream; radio 2: RT audio stream; radio 3: file transfer), as depicted in Fig. 8 (top). Fig. 8 (bottom) also shows the decisions made by MRC. The first radio system doing periodic requests is always granted access for the requested intervals as it has highest priority. The second and third radios are denied access when their request intervals overlap with the first. So when first two radios are active, radio 3 gets very few requested slots granted. When traffic of radio 2 reduces, the extra free time is dynamically granted to radio 3. Each radio, as well as the application using it, is executed in isolation without knowledge about other radios, and without any other coexistence mechanism between radios. It is possible to start new radios so that new and old radios adapt themselves automatically to the changing situation and operate together without knowing the combinations beforehand.

3.7. Resource sharing among multiple radios (Demo 5)

Our software architecture and radio compiler flow have been designed to enable platform resource sharing among multiple RT constrained radio applications. In the current platform with RR schedulers we can demonstrate this for radios working on the same timing granularity, e.g. running 2x DVB-T or 2x WLAN receiver resource models without interference. This is backed up by analysis results. N.B. for the 2x WLAN case we had to scale the execution times of our resource models by a factor of two ("scaled real-time") as the processors operate at rather low clock frequencies: EVP 183 MHz, ARM 61 MHz. As discussed, running DVB-T in parallel with WLAN, would require a preemptive scheduler. Assuming a TDMA scheduler with a 4µs time period, then 13.7% of the load would be spent on context switching assuming 100 cycles (scaled to 50) for context save/restore on EVP running at 183 MHz. This overhead is not insignificant, but still an affordable price for this multi-radio use case.

4. CONCLUSION

Key SDR handset challenges include the abilities to:

- port radios as software-defined entities across multiple platform and platform instances, and
- run multiple radios simultaneously on shared resources,

while maintaining real-time guarantees and with acceptable overheads. Note that multi-radio goes beyond SCA [14].

In this paper we report an SDR technology demonstrator that shows that these challenges can be addressed by:

- unifying and standardizing the radio access interface and the radio application interface;
- constraining the radios to MCDF;
- managing shared resources at multiple levels: operational states, access intervals (burst, slots), and MCDF modes.

These essential multi-radio capabilities are demonstrated on an available hardware platform, and amount in essence to providing a virtual radio computer to individual radios. This is work in progress. Future work includes the demonstration of real radios (rather than resource models), forms of preemptive scheduling, de-fragmentation of resources, multi-radio capabilities of RF interfaces, and more.

5. REFERENCES

[1] K. van Berkel. Multi-Core for Mobile Phones. In *Proc. of DATE 2009*, 2009.
[2] ETSI TR 102 680. Reconfigurable Radio Systems (RRS); SDR Reference Architecture for Mobile Device". 2009.
[3] A. Ahtiainen et al. Architecting Software Radio. In *Proc. of the SDR Forum 2007*.
[4] A. Ahtiainen et al. Multi-radio Scheduling and Resource Sharing on a Software Defined Radio Computing Platform. In *Proc. of the SDR Forum 2008*.
[5] O. Moreira et al, "Data-flow-driven Software Architecture for Software Defined Radio", (manuscript) 2009.
[6] E.Lee and D.Messerschmitt. Synchronous Data Flow. In Proc. of the IEEE, Vol. 75, No. 9, pp 1235-1245, 1987.
[7] S. Sriram et al. *Embedded Multiprocessors: Scheduling and Synchronization,* Marcel Dekker Inc., 2000.
[8] O. Moreira and M. Bekooij. Self-timed scheduling analysis for real-time applications. *In EURASIP Journal on Advances in Signal Processing*, Vol. 2007.
[9] U. Ramacher. Software-Defined Radio Prospects for Multistandard Mobile Phones. *In IEEE Computer,* Vol. 40, no. 10, pp. 62-69, Oct. 2007.
[10] J. Glossner et al. A Software-Defined Communications Baseband Design. In *Communications IEEE*, Vol 41, no 1, pp 120-128, 2003.
[11] K. van Berkel et al. Vector Processing as an Enabler for Software-Defined Radio in Handheld Devices. In *EURASIP J. on Applied Signal Processing* Vol. 2005, 16, pp. 2613-25
[12] Y. Lin et al. SODA: A Low-power Architecture for Software Radio. In *IEEE Proc. ISCA 06*, 2006.
[13] P. Kourzanov, LIME, http://sourceforge.net/projects/sub-lime
[14] http://sca.jpeojtrs.mil.