Abstract: This paper presents the Flexible Radio Kernel (FRK), a configuration and execution management environment for hybrid hardware/software flexible radio platforms. The aim of FRK is to manage platform reconfiguration for multi-mode, multi-standard operation, with different levels of abstraction.

A high level framework whose goal is to manage multiple Medium Access Control layers and to enable cooperation algorithms for cognitive radio is defined and described. A low-level environment is also available to manage platform reconfiguration for radio operations. Radios can make use of hardware or software elements, and configuration state is hidden to the high-level layers, offering pseudo concurrency (time sharing) properties independently of the nature of the elements implementation. This study presents a global view of FRK, with a specific focus on its software architecture and its support of configurable hardware. The effectiveness of the approach is demonstrated through a practical example.
An Environment for (re)configuration and Execution Management of Heterogeneous Flexible Radio Platforms

Pierre-Henri Horrein
Christine Hennebert, Frédéric Pétrot

CEA, LETI, Minatec, Grenoble, France
TIMA Laboratory, CNRS/Grenoble-INP/UJF, Grenoble, France

Abstract

This paper presents the Flexible Radio Kernel (FRK), a configuration and execution management environment for hybrid hardware/software flexible radio platforms. The aim of FRK is to manage platform reconfiguration for multi-mode, multi-standard operation, with different levels of abstraction.

A high level framework whose goal is to manage multiple Medium Access Control layers and to enable cooperation algorithms for cognitive radio is defined and described. A low-level environment is also available to manage platform reconfiguration for radio operations. Radios can make use of hardware or software elements, and configuration state is hidden to the high-level layers, offering pseudo concurrency (time sharing) properties independently of the nature of the elements implementation. This study presents a global view of FRK, with a specific focus on its software architecture and its support of configurable hardware. The effectiveness of the approach is demonstrated through a practical example.

Keywords: Flexible Radio, Heterogeneous platforms, Reconfiguration, HW/SW co-design, Cross-Layer protocols

1. Introduction

The increasing number of wireless communication standards and their ever wider access to final users has led to the emergence of new problems in the design of radio devices. Indeed, the arrival of wireless technologies at all communication levels (PAN, LAN, WAN) means that multiple wireless standards can be present on a single device, and as the standards are evolving very fast, the conventional devices are rapidly becoming obsolete. This increased number of standards leads to a saturation of the radiofrequency domain and to the need of as many devices as targeted standard, which is counter productive as the power consumption is becoming a major design issue with the increasing demand for mobility.

Flexible radio is a promising solution to these problems[1][2]. The aim of a flexible radio device is to offer reconfigurable radio operation. Instead of having a static radio device, able to process a single standard, a flexible radio device is meant to process a given range of standards. It can thus solve most of the above mentioned problems:

- the wider range of operation of a single device allows new algorithms to be run in order to optimize RF resources utilization (cognitive radio or opportunistic radio are good examples),
- the centralization of multiple standards in a single device reduces the occupied area,
- in the case of multi-mode operation, the centralization can also reduce power consumption,
- reconfigurability of the device means that updates are possible after the actual physical deployment.

Flexible radio is a very wide concept. It encompasses any solution for radio operation that can be modified without changing the physical system. This definition allows enough freedom for a large number of different solutions to exist. Some FPGA-based flexible radio platform can be modified while out of operation. These platforms are flexible but only statically reconfigurable. Dynamic reconfiguration of FPGA-based platform is very time consuming as of today, and thus not usable with radio time constraints. On the opposite, some solutions are only based on processors. In these solutions, called Software Defined Radio (SDR), all operations are performed using software implementations. Most studies in flexible radio focus on the hardware platform part, providing new reconfigurable computing elements or new architectures to build the radio terminal.

Flexibility in radio operation can lead to great improvements for upper layers algorithms. Yet, increased flexibility means a more complex control of the platform. Flexibility is not achieved only through a flexible platform, it also requires a flexible run-time environment. If using flexible radio devices means spending twice the time in software development, or having unsolvable constraints due to the reconfiguration overhead, flexibility becomes a burden instead of being a solution.

This study aims at presenting a new approach for flexible radio devices management, providing an efficient and flexible...
framework for application developers. The proposed framework allows management of multiple standards instances, from the radio layer to medium access control (MAC) layers, as well as abstractions for flexible radio users. Flexible radio is an application here, the study can be used for any application implementing a dataflow model, like Kahn Process Networks or Synchronous Data Flow, for example. The proposed abstractions also allow the use of heterogeneous platforms, with support for different software execution units and for hardware coprocessors.

The paper is organized as follows. In Section 2, a short presentation of the reconfiguration process and of existing environments is given. In Section 3, a general description of the proposed approach is presented. Details on application management are given in Section 4, while the method used to manage different processing units is described in Sections 5 and 6. In Section 7, a practical case is studied, to validate the proposed approach. Finally, we draw our conclusions in Section 8.

2. Existing solutions

2.1. Reconfiguration overview

Reconfiguration in a flexible radio environment can have different purposes:

- update the standard in order to follow technological evolutions. This is a static reconfiguration, which can be operated while out of operation, and without any performance requirements,
- change the operating standard at low frequency. This reconfiguration occurs when changing cell, for example, but it has very low performance requirements. Only one standard can be active at a time,
- allow multi-mode operation. In this case, the aim is to use a single platform in order to execute several standards with a pseudo concurrency (time sharing).

A study of reconfiguration issues in a pure software system is presented in [3].

The aim of the reconfiguration process is to switch a system from a given state to a new state. Three elements can be defined in a reconfiguration process:

- the initial configuration (ie the state) of the system, and a representation of this state,
- the new desired configuration of the system after the reconfiguration, and a representation of this new state,
- and the protocol used to switch from the initial configuration to the new configuration.

Reconfiguration can occur for different reasons at different levels. It is in fact a hierarchical process, as presented in [4]. Each layer is a different abstraction level, and needs lower level abstractions to be efficient:

- the first and highest level is the network standard level, in which the application is an abstract representation of a standard, with no knowledge of the internal workings of the standard,
- the second level is the functional block level, with the application being a sequence of known operations, but with no knowledge of the implementation of these operations,
- the last level is the implementation level, with a mapping of the functional blocks to the real computing elements of the platform.

Reconfiguration can be defined, and thus occurs, at any of these abstraction levels.

2.2. Reconfiguration management

Different environments for execution and reconfiguration of flexible radio applications have been proposed. Reconfiguration Management Architecture (RMA) [5] is a distributed environment for high level reconfiguration management. The aim is to define a network architecture with the different required elements to manage controlled reconfiguration of the devices in the network. RMA is designed to work with Software Defined Radio.

Platform Hardware Abstraction Layer (P-HAL) [6] is a complete execution and reconfiguration environment for flexible radio. The aim of P-HAL is to define a unified environment for radio application development. It is a multi-layered environment, supporting multi-platform execution by offering a single virtual platform view to the higher layers. A new version of P-HAL, called ALOE [7][8], has been designed in order to support cognitive radio operations, with better monitoring and an improved dynamic reconfiguration process. Yet, in both case, the proposed solution is mainly designed for pure Software Defined Radio, with little support of specialized hardware. Support for FPGA is provided, but not for dynamically (re)configurable elements.

Surfer, presented in [9] and [10] is another environment, using heterogeneous platform to avoid overload of execution units. A single unit can be used as long as it is not used by other applications, but as soon as another application requires the unit, radio execution is done on another unit.

Finally, Software Radio environments are good examples of execution environments. Software Communication Architecture (SCA) [11] for instance uses a centralized CORBA name server to manage reconfiguration and execution, with an Interface Description Language (IDL) for communication management. It is a Software Radio standard, which can have multiple implementations. OSSIE [12] is a free software implementation. A comparison of SCA and ALOE is provided in [7]. GNURadio [13] is another well-known environment. It is a library of signal processing elements which uses a Synchronous Data Flow approach.

All these solutions propose an abstraction of the actual executing platform. SCA and P-HAL even give an abstraction of multiple platforms working together. Yet, they do not fully
answer the problem studied here: defining a flexible radio execution and reconfiguration environment able to provide applications a unified view of any executing platform. This aim demands support for dedicated hardware and specialized processors.

2.3. Hardware/Software cooperation

A flexible radio platform is a mixed hardware/software reconfigurable platform. Some solutions exist in the literature to manage reconfiguration and execution in these platforms. Hardware Thread Interface [14] is an interesting solution to manage mixed FPGA/CPU platforms. Instead of being passive CPU extensions, the FPGAs become full reconfigurable processing units, with special threads running on it, and managed in the system. This solution frees the CPU, and the resulting abstraction eases development. Other studies make use of FPGA, such as [15].

Following the same idea, OveRSoC [16] is a complete HW/SW environment featuring hardware thread for hybrid execution.

P-HAL [6] also describes an interface for reconfiguration management, in the specific context of flexible radio. While this interface is similar to R-HAL, differences exist in the hardware management method, as well as in the dynamic reconfiguration capabilities.

2.4. Reference architecture

In this study, a flexible platform is a complete system, based on:

- General Purpose Processors (GPP);
- Digital Signal Processors (DSP);
- Dedicated Reconfigurable hardware (RHW) elements;
- Input/Output devices (I/Os);
- Memory capabilities (RAM, Flash, ...).

This reference architecture is presented in Figure 1. For the sake of clarity, it is divided in 4 distinct subsystems:

- the processor subsystem with GPPs, DSPs and local memories,
- the RHW subsystem, with all reconfigurable hardware accelerators,
- the I/O subsystem, with all I/O devices not involved in radio processing,
- the memory subsystem, with main global storage capacities and DMA engines.

This is not necessarily the real architecture of the device, only a convenient representation of it. It is assumed that all elements in the platforms are addressable, and that RHW coprocessors are configured through the use of registers. The elements configuration can be changed during operation.

Figure 1: Flexible architecture reference

This means that they must be either stateless, with no intermediate state during processing, or that the state can be reloaded if needed, with writable state registers for example. All the elements are connected through an interconnection method such as a bus or a Network on Chip. While RHW elements are not as flexible as a software implementation, they are more efficient, allowing higher processing throughput for reduced power budget. They are used in different baseband processors, such as the Magali chip [17] or the Leocore [18]. Heterogeneous platforms are also studied in [19].

RHW elements can be of different granularity. They can represent a complete communication chain, or part of a communication chain. They can also represent a single operation such as a configurable Fast Fourier Transform (FFT) hardware accelerator, or a Viterbi decoder. Possible examples of single reconfigurable operators are given in [20].

2.5. Classical software environment

In a classical approach, as presented on Figure 2, an operating system (which can be as light as a simple kernel) is adapted to the platform. Since the aim is to manage communication standards, and to implement media access protocol, a real-time kernel is a good choice. In this real-time kernel, support for the RHW is added, using peripheral drivers. These drivers offer unified interfaces with the devices, in order to set the configurations. The kernel manages the I/O subsystem using the same approach.

In addition to managing the hardware part, the kernel offers useful routines to applications: memory management routines, multitasking possibilities (POSIX threads, process management, ... ) with scheduling of tasks on the different processors, and real-time functions.

Using this classical approach, RHW elements are integrated in the kernel. Managing multiple running standards and reconfiguration of the platform according to the selected communication means is not easy:
Medium Access Control (MAC) protocols can be implemented using the available routines, but the configuration of the platform requires knowledge of the underlying hardware,

- configuration must be done in the application, with no centralization of the knowledge of concurrent standards,

- hybrid execution of hardware and software requires "software ghosts" for the hardware tasks in the system.

Developing a functional and efficient flexible environment for flexible radio management requires kernel integration. The proposed environment, called Flexible Radio Kernel (FRK), is described in the next subsection as an extension of a classical kernel. It aims at offering to radio applications the routines for tasking, memory usage, or I/O management that can be found in any classical system kernel, while wireless networking and reconfigurable hardware support are developed in a framework designed for flexible radio.

3. Flexible Radio Kernel

This section presents the general architecture of the Flexible Radio Kernel (FRK).

3.1. Environment presentation

While other existing environments are mainly designed as SDR environments, FRK is designed as a flexible radio environment with three main goals in mind:

- offer an independent application representation which can be deployed on any platform,

- allow full use of the flexible platform presented in Figure 1, even if the user (developer) has no knowledge of it,

- provide a simple mean to integrate algorithms making use of flexibility.

The framework is built as a library, which means that nothing is being executed specifically for the management behind the scene. It still offers execution management through a runtime part (which is in fact a library with functions to manage the execution), and others services through an off-line part.

The runtime part is based on software layers defined in order to provide control over the platform at all the development levels. As illustrated in Figure 3, these layers can be seen as additions to a classical kernel specifically design to handle flexibility. The classical kernel is still available, in order to manage the unrelated elements (memory, I/Os, ...).

The Protocol Layer (PL) acts as a networking standard scheduler in FRK. It stores all implemented MAC layers, and controls activation or inactivation of each MAC layers. If a MAC layer is active and requires radio processing, it guarantees that the platform is configured as it must be. The PL controller is responsible for managing the cooperation between the different MAC layers, as well as offering a standard selection service to upper layers (based on available QoS for example). The basic element for the PL is the MAC layer instance.

The off-line part is integrated in the R-HAL. It is designed to translate a generic application called a waveform into a plat-
form executable called a Configuration Instance (CI). The translation process is described in Section 4.

As said before, those two additional elements to the kernel are only additions. Other elements from the system kernel are still available for applications. It is also important to note that FRK is not designed to enhance the platform processing power, but to fully benefit from it. Platform limitations still hold. If the platform is not able to process a given operation higher than a given throughput, the use of FRK will not permit reaching this throughput.

In the next sections, specific parts of FRK are detailed.

3.2. Platform Management: R-HAL

The R-HAL is designed to manage reconfigurability in a flexible radio platform. It offers to the PL and other possible layers an abstracted interface for application management. It manages the instantiation of multiple applications and the repartition on the different targets according to the platform capabilities. It is also responsible for the pseudo-concurrency.

The aim of R-HAL is to manage the configuration of the platform following requests from the PL. Different CI can be instantiated at the same time, and multiple configurations can be active if the platform can support multiple wireless communication protocols at the same time. Some configurations can be inactive but instantiated, if the associated standard is standby.

R-HAL offers to the PL an abstraction of the platform. Requests for a new configuration are made following a standardized application representation. This means that the R-HAL has the following main duties:

- translate the application representation into the associated platform configuration, which is an off-line operation,
- manage instantiated configurations at runtime, in order for the PL to be able to use any of them.

When a MAC layer is instantiated on the flexible platform, tasks are created and added to the system scheduler. In the MAC manager, a structure representing the MAC instance (MI) is created, and added to the MAC pool. The MI contains a list of pointers to the tasks associated with the MAC, and pointers to the associated CI provided by R-HAL. A single MAC can use different communication channels, for transmission, reception and control.

Decision on which standard is to use can be taken according to different sources (others can also be thought of), as illustrated here:

- a standard recognizer can be implemented at the input of the flexible radio device, leading to the decision of which standard to use to process the incoming data,
- a high level algorithm can decide to use a specific standard for transmission, based on estimated QoS, on transmission requirements, or on receiver capacities,
- a network coordinator can be used to manage all flexible devices on a network, in order to synchronize and select the best standard from the network point of view.

The MAC controller implements the algorithm taking these sources into account. When a standard is instantiated, information on the standard are registered in the MAC controller. It permanently monitors the state of the different standards. The controller then sends the decisions to the configuration scheduler.

3.3. PL architecture

The Protocol Layer is responsible for executing and managing the MAC layers, as well as sending the requests to the R-HAL. The architecture of the PL, detailed in Figure 5, is made of three main elements:

- a MAC manager, used to schedule and execute the different MAC layers,
- a scheduler, whose role is to decide when reconfiguration occurs, and how it will take place, based on requests from the other actors,
- a MAC controller, used to implement optimization algorithms making use of flexible radio.
The scheduler acts according to the orders from the MAC controller. It centralizes all the operations on the MIs and CIs. When an order comes from the controller, it requests a change in the MAC controller, and in the R-HAL. Corresponding tasks are activated or inactivated.

4. Application management

In this section, details on application management in FRK are given, both for MAC applications and radio applications.

4.1. MAC representation

One of the advantages of flexible radio lies in the new possibilities for medium access management. The reconfigurability of the platform means that wireless communications can be adapted to the environment, and to the requirements (if a low performance communication is sufficient, then it is not necessary to use high data rate, low latency, power greedy standard).

A standard is always defined with a physical layer (managed by R-HAL), and a MAC layer. The MAC layer is designed to support communication security, multiple access to a shared medium, management of devices (addressing), and QoS. A MAC layer can be represented using three tasks and several libraries.

The three tasks are used as follow:

• a first task is used as the reception task, taking care of incoming data from the network,
• a second task is used as the transmit task, managing transmission time, and implementing the packet scheduling method,
• a last task is the control task, used for all the annex MAC operations.

All the libraries are provided for the remaining MAC functions, such as encapsulation, QoS management, or security. Some standards are more complex, using for example several parallel reception and transmission channels. In this case more than three tasks may be used.

4.2. Waveform

The waveform is the generic application in FRK. It is meant to be used on any platform by the PL or any other application using the R-HAL, in order to provide a platform-independent representation of the application. Waveforms in FRK are built using the Operation to Platform Mapping (OPM) library. This library offers a set of predefined operations, which can be instantiated and parameterized according to the need of the application. These operations are for example Fast Fourier Transform (FFT), error coding, filters, ...

An operation in the OPM is defined using two kind of parameters:

• FRK related parameters, which mainly give the structure of the operation, with for example the number of input and output channels,
• operation related parameters, for example the number of points of a FFT or the coefficients of a filter.

Each of the instantiated blocks can then be connected using a specific functions. As a result, a graph representing the application is obtained.

4.3. Configuration Instance and translation

This graph is not understandable as such by the platform. A translation process must take place, in order to transform this generic representation into a platform Configuration Instance (CI). This translation process

Translation is managed by the off-line part of FRK, and is made possible by the target view of the platform. When implementing a target in the environment, possible operations on this target and their implementation must be provided. This means that for each operation defined by the OPM, information on whether it is supported or not by a target is given. If it is supported, a function select_config is implemented, which gives the implementation on the target for the selected parameters. This function is implemented because a given target may support only a selected range of parameters for an operation (for example, only 1024 points for a FFT). It may also have more efficient implementation according to the parameters. A FFT may be implemented in software using a radix-4 algorithm, which is more efficient than a radix-2 algorithm, but can only be used when the number of points is a multiple of 4. The select_config is designed to support this choice. The translation process is presented in Figure 6. The waveform is a simple one with two operations. The demapping operation can be implemented only on target 1. The FFT operation can be implemented on both targets. In order to solve this concurrency, a priority order between targets is given. If target 2 has priority, and the FFT parameters are compatible with the only implementation on this target, then the FFT will be implemented on this target. If target 1 is chosen for FFT, then the select_config function will choose between the two possible implementations.

Once operations have been translated, communications between the operations can be generated. Once again, communication is implemented by targets. For each target, several communication methods can be designed. They follow a predefined
interface, based on a classical FIFO implementation. The translation process parses all connections between operations. For each of the connections, producer target and consumer target are known, since operation translation has been done. A target is chosen between both according to a predefined priority list. For example, the coprocessor target will always be chosen if one of the operation is implemented on this target. The chosen target uses a select_fifo function to select a FIFO implementation compatible with the operation.

The translation result in a CI which can then be managed using the R-HAL interface.

4.4. CI management

The R-HAL interface is the layer responsible for CI management. There are a limited number of actions that may be requested from the R-HAL.

First, CI can be loaded or unloaded. Loading a CI means registering its existence in the R-HAL, and loading all its components in the corresponding targets. Unloading is the reverse operation. A loaded CI is not necessarily executed. It only means that all associated implementations are ready to be executed. In order to execute (or stop the execution of) a CI, it must be activated (or deactivated). Activation means that all the components may now be executed. Loading or unloading an application may be long, depending on the number of operations and on the target usage. Activation and deactivation is a way to reduce time before a request from the PL is taken into account.

The typical use case for these functions is the following:

1. A new standard is being instantiated on the platform, and a new configuration instance is requested. This uses the loading function,
2. The PL selects a standard for a given transmission. In this case, unused standards may be deactivated, to avoid execution units overload, and the running standard may be activated,
3. A standard is being deleted from the platform. In this case, the PL only unload a CI, since it will not be used anymore.

Once loaded and activated, CI are managed in a distributed manner, operation by operation, by the TaME. A priority mechanism is available, with two possible states.

Finally, the last action that can be performed by the R-HAL is the dynamic adaptation of an application. In order to avoid a complete translation and a complete reload of the application, FRK offers a way to modify only a selected operation of a waveform, and to apply the modification on the CI.

The complete FRK development flow, from waveform to actual execution, is presented in Figure 7.

5. Target integration

5.1. TaME interface

The concept of targets is the key to heterogeneous platform management and genericity of applications in FRK. A target is integrated in FRK using a Hardware Abstraction Layer (HAL), and a Target Management Element (TaME), which can be represented as in Figure 8.

The HAL is designed to support the target. For example, the HAL of the hardware coprocessors target is used to manage interrupts from the coprocessors, and to define the representation of these coprocessors. The HAL for software execution must manage the processor.

The TaME is designed to manage operations, which are called configurations at this level in FRK. Each TaME has its own configuration definition, which is necessarily adapted to the target representation in the HAL. A scheduler is also implemented in the TaME, to manage the different configurations. It is interesting to note the scheduling in FRK respects the hierarchical definition of reconfiguration proposed in [4]. High level scheduling is done in the PL, at the standard level. Mid-level scheduling is executed in the TaME. It works at an operation level. Low level scheduling is located in the HAL. It is responsible for configuration management on the actual processing unit.

The TaME implements an API which roughly mirror the R-HAL API. The R-HAL is an interface with the actual targets, with added support for CI.

5.2. FIFO channels

FIFO channels are also located in the TaME. Different implementations can be defined, depending on the possible communications. For example, if the target is hardware coprocessors, some coprocessors may implement a hardware FIFO, while other may use internal memory. FIFO implement a simple interface, based on blocking read and write. When FIFO are read and no data is available, the caller is blocked as long as no data has been written to the FIFO. Since FIFO may be used for inter-target communication, locking and unlocking functions must be
This means that coprocessors are not really integrated through the kernel as HAL, a new HAL is designed. Instead of using the kernel to manage threads, and to define priority based on the upper layers requests, a new HAL is designed. A software task for FRK is represented in Figure 9. The process function is provided by the OPM during the translation. It is the actual processing function. Regardless of the operation being implemented, the process function has a fixed prototype, in which the generic operation is included in order to provide specific parameters. This function is designed to process a given amount of data. Reads from input FIFO channels and writes to output channels are managed by the local controller, which is in fact the main function of the task. The controller reads the required amount of data from input channels, calls the process function, and write the results to the output channels. It also gives required feedback to the TaME scheduler.

The kernel scheduler is managed through the use of locks. Each configuration (task) has a lock which is global for the CI, for fast deactivation or activation of the task. The lock is checked before each execution of the process function. Depending on the constraints, the frequency at which checking a lock availability is done can be reduced, as it can be long and non-deterministic. Added to the CI lock, each priority queues implemented defines a lock, which must be checked according the priority order. The controller of an operation only checks locks for queues with higher priority than the queue of the operation.

6. Hardware coprocessors integration

6.1. Coprocessor TaME

The coprocessor TaME is different from the software TaME. Instead of using the kernel as HAL, a new HAL is designed. This means that coprocessors are not really integrated through a standardized interface like the device interface of POSIX kernels, for example. It also means that the low-level scheduling must be implemented, since a coprocessor may be used to process several operations concurrently.

Coprocessors are represented in the TaME as registers. Two kind of registers can be used:

- pseudo-constant registers are not modified by the coprocessor, they represent the configuration of the coprocessor,
- volatile registers may be modified during execution by the coprocessor, they represent the state.

Addresses of the different registers are given in order to fully represent the RHW element. Based on this representation, a configuration for the coprocessors can be defined as a set of values for the different registers. This representation leads to the following method to manage the coprocessor.

6.2. Configuration switch

The concept of the coprocessor integration in FRK is similar to the context switch of a GPP. Since the coprocessor is defined as a set of read/write registers, it is possible to use what we call a configuration switch. When a new configuration is selected by the TaME scheduler, the running configuration is saved in the corresponding instance. This backup is in fact a copy of all volatile registers of the coprocessor. Once the coprocessor state is saved, the running instance is tagged as non-running, and the new configuration is tagged as running. Values for the pseudo-constant and volatile registers are copied to the coprocessor registers, and the coprocessor is activated.

6.3. RHW FIFO

FIFO channels in the RHW TaME are implemented according to the coprocessor access method, which depends on the system interconnection. It also depends on whether the coprocessor has an integrated HW FIFO, on the presence of DMA channels, and on the speed of the system. Implementing the FIFO is part of porting FRK to a platform.

In the current version of FRK, FIFO are based on a software buffer, which is then copied using a DMA (or the CPU, if there is no DMA) if the corresponding configuration is running. This is the preferred solution, in which connected operations are not necessarily blocked if the configuration is inactive. The HAL must have support for DMA.

6.4. Configuration scheduling

RHW FIFO must have two threshold values for scheduling. The minimal threshold is the minimal required amount of data before the configuration may be processed. The maximal threshold is the amount of data above which the configuration should be executed, in order to avoid blocking the writing operation. These thresholds are used to schedule FIFO consumers. When all input FIFO have reached the minimal thresholds, the operation is tagged as executable. When on of the input FIFO reaches the maximal threshold, and the operation is already executable, it is tagged as candidate.
Each time data is written in a FIFO, the operation status is updated. If the status is modified, the scheduler is called. Reading data does not update the status, in order to avoid specific protection of the information due to possible collisions. Since reading is necessarily done by a running configuration, it is possible to update status when the scheduler is called. Scheduling is done while protected by a TaME lock, which prevents writing to the FIFO.

![Figure 10: Coprocessor TaME scheduler](image)

The scheduler algorithm is presented in Figure 10. The aim is to avoid unnecessary configuration. If the running configuration is candidate, then other configurations can have equal priority, but not higher. No switch is performed. Otherwise, all possible configurations are parsed. If one of them is candidate, it is automatically selected and loaded on the RHW coprocessor. If one of them is executable, and the running configuration is not executable, then a switch is performed. These are the only two cases when a switch is performed.

7. Practical study: Convolutional coder example

7.1. Platform presentation

In order to illustrate and validate the mechanisms in FRK, a simple and practical example is given. The aim is to encode data in a file with a convolutional code, and to write the result in another file. Three operations are used:

- a file source to read data from a file,
- a convolutional coder,
- a file sink, to write data in a file.

The convolutional coder operation is represented by 4 parameters:

- the number of inputs used in the code,
- the memory of the code,
- the number of outputs of the code,
- the output masks, which represents which elements are added in order to obtain a specific output.

The platform used is represented in Figure 11. It is based on an ATME AT91RM9200 system on chip (which uses a ARM processor), and on a FPGA. The FPGA connected to the ARM processor is used to implement hardware convolutional coders. Two kernels have been ported to the platform, and used with FRK:

- a Linux 2.6.23 kernel, which offers a POSIX interface,
- a RTEMS 4.10 kernel, which is a real-time kernel. It also offers a POSIX interface, but FRK was adapted to use the non-POSIX part of the kernel.

![Figure 11: Test platform](image)

Two encoding applications are written using the corresponding OPM operations. The applications are similar, the only difference lies in the coder parameters. Both coders are rate $\frac{1}{2}$.
coders, with 3 memory units. An example of such a code is represented on Figure 12. The two applications are called A and B in the next parts.

7.2. Software case

In a first case, no hardware convolutional coder is available. The translation of the applications results in two full software CI. These CI are then loaded using the R-HAL, and scheduled through blocking FIFO by the kernel.

7.3. Mixed hardware/software

In a second case, a simple hardware convolutional coder is designed, only able to process coder A. The translation of application B still results in a full software CI. The translator, when checking the hardware coprocessor, finds a correspondence for the convolutional coder, but the select_config has no configuration corresponding to coder B parameters.

On the contrary, the translation of application A results in a mixed hardware/software CI, since coder A can be implemented on the hardware coder. File source and sink can only be executed by the software target. As a result, the software target executes 5 of the 6 operations, while the hardware coder processes one operation.

7.4. Hardware case

Finally, the most interesting case is to use a configurable convolutional coder. This coder can process any code with 1 input, 3 outputs, and 3 memory units. It uses 4 registers, one of which is volatile:

- 3 registers are used to give the mask for output processing,
- 1 register is used for memory units contents.

The state of the memory units can be read and written. The coder can thus be used to process both coder A and B. Input and output FIFO are integrated in the coder.

The translation of both applications generates two mixed hardware/software CI, as represented in Figure 13. Both operations and FIFO channels can be seen. In the software TaME, four tasks are processed, which must all be executed by the GPP. In the hardware TaME, two configurations are processed, which are executed by a single coder coprocessor. FIFO are all managed by the coprocessor TaME. They use a software buffer when the configuration is inactive, and the coder FIFO when the configuration is running. FIFO 0 and 1 are input and output of coder A, while channels 2 and 3 are input and output of coder B. The arrows in the figure represent links between FRK entities.

These applications lead to a platform state represented on Figure 14. Coder A is the running configuration. The actual registers values are values corresponding to coder A parameters. FIFO 0 and 1 use the hardware FIFO of the coder, while FIFO 2 and 3 use the software buffers. Sink B has been executed by the software TaME, but FIFO 3 is empty since coder B is inactive, and as a result sink B has been put in a waiting state due to blocking FIFO. Sink A has already written codeA(a) in the file. Source B has started filling channel 2, which reaches its maximal threshold when the E is written. The scheduler is thus called.

Since the running configuration is executable but not candidate, coder B configuration is selected, and a configuration switch occurs, putting the platform in the state represented in Figure 15. Input FIFO of the coder is processed by the coder before switching, in order not to lose data. Output FIFO is copied in the software buffer of channel 1. Once this is done, the volatile register is saved in coder A configuration, and coder B configuration is made running and copied to the hardware coder. Software buffer for channel 2 is copied in the hardware coder.
8. Conclusion

In this study, a complete environment for flexible radio, called FRK, has been presented. This environment is designed to enhance flexibility, by offering easy configuration management.

The environment allows easy integration of heterogeneous processing units thanks to target separation. In this paper, integration of software operations was studied, using kernel tasks. It also features a new way of using hardware coprocessor, using a processor-like approach and configuration switch to manage concurrency. The target view allows easy development of radio applications, using a generic view which can then be translated into a platform dependent configuration.

It also integrates MAC layer in its architecture, in order to enable new algorithm for cognitive radio. The resulting environment can be deployed on any platform, with no application adaptation, as long as the platform is able to provide the required processing power.

FRK is currently under development with encouraging first results in terms of usability. The current development version is able to work along with Linux or RTEMS as a classical kernel. All development is done using the usual C language. At the time when this paper was written, most of the R-HAL part had been developed, and the first PL mechanisms had been deployed. Complete implementation of FRK is still under way. We expect, in a near future, to be able to run complete practical cases on real platforms. Practical cases on real platforms have been run, and a more complete study of the environment especially in terms of overhead is under way. Real-time issues inside FRK are also the focus of our researches.

References


Pierre-Henri Horrein received an engineering degree (master) from Grenoble Institute of Technology in 2008. He is a PhD student in CEA Grenoble. His research interests are in the field of flexible radio and reconfigurable computing, from a software point of view.

Dr Christine Hennebert is currently a Research Engineer at CEA Grenoble, France. She obtained an Engineering (master) degree from ESIGELEC, France in 1992 and a PhD degree from Institut National Polytechnique de Grenoble, France in 1996. Her research work is in the field of implementation of wireless communication applications on innovative and heterogeneous platforms.

Frédéric Pétrot received the DEA (master) and PhD degree in Computer Science from Université Pierre et Marie Curie (Paris VI), Paris, France, in respectively 1990 and 1994, where has been Assistant Professor in Computer Science. He was then one of the main contributors of the Alliance VLSI CAD System and the Disydent ESL environment. F. Pétrot joined TIMA in September 2004, and holds a professor position at the Ensimag, Institut Polytechique de Grenoble, France, where, since 2007, he heads the System Level Synthesis group. His main research interests are in system level design of integrated systems, and include computer aided design of digital system, architecture and software for homogeneous and heterogeneous multiprocessor systems on chip.