UNIVERSITÀ DI PISA

Scuola di Dottorato in Ingegneria "Leonardo da Vinci"

Corso di Dottorato di Ricerca in
Ingegneria dell'Informazione

Tesi di Dottorato di Ricerca

Fast Memory-Based Processing in Software-Defined Radios

Autore:

Vincenzo Pellegrini _____

Relatore:

Prof. Marco Luise _____

Anno 2012

Manuscript received February 28, 2012

Accepted March 28, 2012

Sommario

Negli ultimi anni le Software Defined Radio sono state un argomento di ricerca di primo piano nell'ambito dei sistemi di trasmissione radio. Molti e variegati paradigmi implementativi sono stati concepiti e proposti, con soluzioni capaci di spaziare da sistemi basati su Field Programmable Gate Array (FPGA) a implementazioni ottenute mediante un singolo General Purpose Processor (GPP) passando per dispositivi caratterizzati dalla presenza computazionalmente preponderante di un Digital Signal Processor (DSP) o da architetture miste. Tali soluzioni rappresentano punti di equilibrio diversi dell'inevitabile compromesso tra flessibilità e capacità computazionale del sistema di trasmissione implementato, comprimendo in qualche modo l'aspirazione ad un *sistema radio universale* propria del concetto originario dell'SDR. A questo riguardo, le soluzioni SDR basate su GPP rappresentano il modello implementativo maggiormente desiderabile in quanto costituiscono l'alternativa più flessibile ed economica tra tutte le tipologie di SDR. Ciò nonostante, la scarsa capacità computazionale ha sempre limitato l'adozione di questi sistemi in scenari produttivi di vasta scala. Se convenientemente applicati entro il contesto di sviluppo SDR, concetti classici noti in informatica sotto la denominazione collettiva di *space/time trade-off* possono essere di enorme aiuto quando si cerchi di mitigare un simile problema. Traendo ispirazione da detti concetti, nonchè estendendoli ed applicandoli all'abito dell'SDR, questa tesi sviluppa e presenta una tecnica di programmazione specifica per software radio chiamata *Memory Acceleration* (MA) che, mediante un uso estensivo delle risorse di memoria disponibili a bordo di un tipico sistema di calcolo general purpose, può fornire alle SDR convenzionali basate su GPP fattori di accelerazione sostanziali (circa un ordine di grandezza) senza ridurne la peculiare flessibilità. Alcune rilevanti implementazioni di sistemi SDR capaci di lavorare in tempo reale su processori GPP consumer-grade realizzate in tecnica MA sono descritte in dettaglio entro questo lavoro di tesi e fornite come prova della reale efficacia del concetto proposto.

Abstract

In recent years Software Defined Radios (SDRs) have been quite a hot topic in wireless systems research. Many implementation paradigms were conceived and proposed ranging from Field Programmable Gate Array (FPGA) implementations to single General Purpose Processor (GPP) systems while also including mixed architecture and Digital Signal Processor (DSP) based devices. All such choices trade-off system flexibility for computing power, compressing, to some extent, the quest for an *universal radio system* which is inherent to the original SDR concept. In this respect, GPP-based solutions represent the most desirable implementation model as they are the most flexible and cost effective among all types of SDRs. Still, the scarcity of computational power on general purpose CPUs has always constrained their wide adoption in production environments. If conveniently applied within an SDR context, classical concepts known in computer science as *space/time trade-offs* can be extremely helpful when trying to mitigate this problem. Inspired by and building on those concepts, this thesis presents a novel SDR implementation technique called *Memory Acceleration* (MA) that makes extensive use of the memory resources available on a general purpose computing system, in order to accelerate signal computation. Actually, MA can provide substantial (about one order of magnitude) acceleration factors when applied to conventional SDRs without reducing their peculiar flexibility. Notable real-time, consumer-grade GPP SDR implementations that were obtained while developing the MA SDR programming technique are described in detail within this work and provided as factual proofs of the MA concept effectiveness. Opportunity for extending the MA approach to the entire radio system, thus implementing what we call a Memory Based Software Defined Radio (MB-SDR) is as well considered and discussed.

Acknowledgments

Once reached the *incipit* of this PhD thesis, I really feel I have to thank a bunch of people.

My advisor Professor Marco Luise for his guidance, for the perspective he was continuatively able to provide on exciting research and development opportunities throughout the PhD program and, particularly, for that day, some years ago, in which he spoke to my ears the words "Software Defined Radio" for the first time. My co-advisor Filippo Giannetti for the invaluable insight he always provided into a wide variety of radio transmission issues as well as for having shared with me the emotion of "teaching the wonders of the radio". My colleague and friend Mario di Dio, for the daily (and nigthly) effort and time spent together in chasing and improving the performance of state of the art SDRs. Dr. Giacomo Bacci of the Pisa University DSP for Communication Lab, for his constant, precious and always gentle help.

Elisabetta, Bernardo, Monica and Giovanni, without whose love, endurance and support, none of the steps composing this research path could have ever been moved forward.

Pisa, February 2012 Vincenzo Pellegrini

Contents

List of Figures

List of Tables

List of Acronyms

ADC	Analogue to Digital Converter
ATSC	Advanced Television Systems Committee
AWGN	Additive White Gaussian Noise
BCH	Bose Chaudhuri Hocquenghem
BER	Bit Error Rate
COFDM	Coded OFDM
CORBA	Common Object Request Broker Architecture
CPU	Central Processor Unit
DAB	Digital Audio Broadcasting
DAC	Digital to Analogue Converter
DMT	Discrete MultiTone
DRM	Digital Radio Mondiale
DSP	Digital Signal Processing
DSPCOLA	DSP for Communication Lab
DVB-T	Digital Video Broadcasting - Terrestrial version
ESSOR	European Secure SOftware defined Radio
ETSI	European Telecommunication Standards Institute
FEC	Forward Error Correction
FFT	Fast Fourier Transform
FFTW	Fastest Fourier Transform in the West
FM	Frequency Modulation
GF	Galois Field
GNU	GNU's Not Unix (Recursive Acronym) :-)

GRC	GNURadio Companion
GSM	Global System for Mobile communications
HDTV	High Definition TeleVision
HW	Hardware
IEEE	Institute of Electrical and Electronics Engineers
IFFT	Inverse Fast Fourier Transform
IP	Internet Protocol
JTRS	Joint Tactical Radio Systems
LDPC	Low Density Parity Check
LFSR	Linear Feedback Shift Register
LOS	Line of Sight
LUT	Look-Up Table
MA	Memory Acceleration
MB-SDR	Memory-Based Software Defined Radio
MFN	Multiple Frequency Network
MIMO	Multiple Input Multiple Output
MPEG2	Motion Picture Experts Group2
NO-LOS	No Line of Sight
OSSIE	Open-Source SCA Implementation - Embedded
PC	Personal Computer
PCIe	Peripheral Component Interconnect Express
PRBS	Pseudo Random Binary Sequence
QEF	Quasi Error Free
RF	Radio Frequency
RS	Reed-Solomon (Codes)
SIMD	Single Instruction Multiple Data
SCA	Software Communications Architecture
SDH	Synchronous Digital Hierarchy
SDR	Software Defined Radio
SFN	Single Frequency Network

SNR	Signal to Noise Ratio
SW	Software
TDP	Thermal Design Power
TPS	Transmission Parameters Signalling
TS	Transport Stream
UHF	Ultra High Frequency
VHF	Very High Frequency
USRP	Universal Software Radio Peripheral

List of Symbols and Operators

Computation-only implementation of block n	$f_n(...)$
Memory-only implementation of block n	$t_n(...)$
Number of items within the MIDS	l
Cardinality of Alphabet for each item of MIDS	A
Cardinality of input space of block $f_n(...)$	C_{i_n}
Cardinality of output space of block $f_n(...)$	C_{o_n}
Total available computational power	W
Computational cost of block $f_n(...)$	W_n
Computational cost of subsystem within table boundary	W_{TB}
Computational cost of subsystem replaced by table m	W_m
Computational cost of subsystem replaced by table m	W_m
Computational cost for handling table $t_m(...)$	Wm_m
Total size of available memory	M
Total memory footprint of table $t_m(...)$	M_m
Data size of items stored in $t_m(...)$	S_m
Acceleration factor	a
Acceleration efficiency	η
Acceleration efficiency of table $t_m(...)$	η_m
Overall SDR implementation merit parameter	I

Introduction

Motivations for a "different" SDR

The field of SDR is slowly attaining its maturity, especially in military and security-related applications [1]. In spite of this, there is still a wide variety of HW platforms to implement an SDR terminal, with no emerging well-established standard solution(s). Amidst this variety, every platform generally adopts its own judicious mixture of (powerful but not so flexible) FPGAs, dedicated DSP processors (less powerful and sufficiently flexible), and/or (easily programmable and flexible but no so powerful) GPPs [2]. Platforms based almost entirely on GPPs are very attractive for research, development and small-scale market deployment, owing to short development time and extremely low development costs [3]. Indeed, they constitute the closest implementation paradigm to the original SDR concept described in J. Mitola's seminal work [4] back in 1992: the dream of a *"universal radio"* where *all* of the needed signal processing is performed directly over a fully reconfigurable computational back-end. If achieved in practice, this *fully software* architecture could act as an authentic game-changer within the market of radio-frequency (RF) transmission systems, as all production costs ranging from hardware design to Application Specific Integrated Circuit (ASIC) foundry production would be entirely saved. Reference implementations being written in a high level programming language such as C or C++ could then directly become system pre-production prototypes, or even actual complete products. On the other hand, the drawbacks of such solutions are their limited computational power or, seen from another perspective, their low throughput per Watt when compared to equivalent HW-accelerated (e.g. FPGA-based) implementations [5]. In fact, it has always been a universally accepted assumption that a greater amount of generality and flexibility of the radio system has to be paid with heavy losses on power efficiency, due to the necessary usage of general purpose CPUs. Accordingly, SDR implementations

up to the present date have simply aimed to replicate the classical HW-implemented signal processing chains into the software realm. Aim of this research is to prove that, by making use of all the resources available on a general purpose computing system (i.e. not only calculus but also memory), it is possible – at least – to substantially reduce the power efficiency gap that exists between SDRs based on general purpose CPUs and HW implementations of the same radio system. Implications of such results include potential for deploying flexible radio technologies on the industrial scale (due to increase of power efficiency up to levels that would make them a practical alternative to HW solutions) as well as do suggest the possibility to implement fully generic, C/C++ definable *radio signal processing cores* being trivially derived from currently available general purpose CPUs, yet able to deliver very high signal synthesis and demodulation performances.

Principal issues and challenges

Peculiarities of implementing a real-world software defined radio, especially once the full-software approach has been chosen, make the design and complete development of the radio system substantially different from the traditional work-flow, usually involved in producing conventional, hardware (ASIC) based, radio-communication devices.

Different authors in different contexts adopt different definitions for the rather broad class of what is often referred to as *real-time constraints*. The term *real-time* is typically used to indicate both computing time issues and latency issues even if the nature of such constraints is rather heterogeneous. Actually, a practical SDR implementation has to take into account both needs. An SDR must in fact be fast enough in processing (modulating and/or demodulating) its informative radio signal to keep-up with the radio frequency (RF) signal rate mandated by the radio communication standard it implements (computational constraint). Also, it must be quick enough in reacting to a *stimulus* received from the radio frequency channel to comply with external timing requirements imposed by external entities such as (but not limited to) an infrastructured or peer-to-peer network architecture to which the SDR is interfaced (latency constraint). Within this thesis, we will thus respectively distinguish between:

- Computation-yielded real-time constraints
- Architecture-yielded latency constrains

Therefore we will only be using the term *real-time* with respect to computational power related issues, as these constitute the toughest limit keeping the full-software SDR paradigm from widespread adoption. Timing issues depending on the radio system architecture will be instead referred to as *latency* issues. It is worth to consider that, as long as an insufficient processing capability slows down the system in all of its tasks due to the typical interdependence of functional blocks, computation yielded real-time constraints can have an impact over latency performance of the radio system while instead the opposite cannot happen, due to the definition we adopt for these two constraint classes. Also, it must be noted that the customary distinction between *soft* and *hard real-time* is completely orthogonal to the classification introduced above. In software defined radio, all real-time constraints are to be considered *hard real-*

time bounds as long as missing a single deadline set by the implemented radio-communication standard when processing a single given information unit will (both on the transmit and on the receive chains) have catastrophic effects with respect to the possibility of successfully transmitting that unit and, most likely, some of its contiguous information blocks.

Whenever computational real-time requirements are met, which we must assume as the normal operating condition for any SDR, latency issues might only arise from architectural problems. This fact can have a relevant impact particularly when a full-software implementation paradigm is chosen, which falls right into the main scope of this research. GPP-based systems are conceived and designed around a central computing core (or set of cores) whose operation is regulated by a certain *interrupt* scheme. Choices taken in implementing such scheme directly yield latency bounds when it comes to make the fully software SDR systems responsive to external *stimula* acquired through the RF channel (e.g. responding to a transmission request within a certain time which is assigned by some network regulator entity). For these reasons, whenever the SDR system is part of a wider, time-regulated network or when it has to be strictly locked to any sort of external, time-critical event, a *real-time kernel* must be employed within the host operating system and interrupt as well as hardware-yielded latencies (i.e. timing of buses and hardware interfaces) must be cautiously considered.

Also, given the limited computational power available over GPP platforms, suitable techniques must be implemented within the SDR system in order to maximize its computational efficiency (and thus its throughput-per-watt figures). A proposal being formulated within this context is the main subject of the following chapters of this thesis.

Thesis outline

The remainder of this thesis is structured as follows.

In Chapter 1, we present the current state of the art of SDR systems with special attention to the full-software option and to its main issues and criticalities. Both the industrial and academic realms are considered.

In Chapter 2, we describe in full detail the proposed *Memory Acceleration* (MA) SDR programming technique by presenting its logical and quantitative foundations as well as by providing complete examples of its application within real full-software SDR contexts.

In Chapter 3, we describe several complete real-world SDR systems which, along the path of this research, were implemented by resorting to the MA technique, and which we intend as the practical proof of the MA concept.

In Chapter 4, we explore and discuss the idea of extending the memory acceleration design philosophy to the entire software radio system as we imagine and define the concept of a *Memory-Based SDR* (MB-SDR).

Within the customary *conclusions* chapter, we discuss implications of the results which were obtained within this research as well as development and research perspectives that exist within the field of computationally-accelerated software defined radio.

Chapter 1

SDR today: the state of the art

1.1 "Historical" notes

Probably conceived in 1991, the term and the concept "Software Radio" first appear on the world's stage in 1992 at the IEEE National Telesystems Conference thanks to Joseph Mitola's work entitled *"The Software Radio"* [4] where a universal, fully reconfigurable radio system is imagined as composed by no application-specific subsystems and based only upon a general-purpose computational back-end being directly interfaced to the radio medium only through A/D conversion. Since those days, the concept inspired a wide variety of related initiatives in both research and production contexts, which led to a plethora of SDR systems and implementation paradigms. Precisely 20 years, at the time of writing, after Mitola's seminal work, industry and academia, civilian authorities and the military have all become involved into SDR to some extent and for different reasons. Indeed, the label of *early adopters* belongs to the United States military whose involvement in SDR technologies dates back to the pioneering days of the *SPEAKeasy*, phase I and II projects. Such projects consisted in implementing a reconfigurable, multi-standard radio capable of working with ground, naval, airborne and naval forces communication systems (phase I) as well as in obtaining standard-bridging capabilities (i.e. the actual SDR acting as a gateway between heterogeneous RF standards) and in reducing weight and configuration times of former implementations (phase II). The early SPEAKeasy effort extends into nowadays Joint Tactical Radio System: an US military program aiming to obtain

flexible and interoperable communication within a broad set of radio standards. Such system is based on the internationally endorsed Software Communications Architecture (SCA): a highly layered software radio architecture designed in order to obtain remarkable cross-platform portability of the implemented radio systems (*waveforms* in the SCA parlance), which was implemented in a multitude of contexts, including civilian ones. Common Object Request Broker Architecture (CORBA), a special software mechanism allowing pieces of software written in different languages to interoperate is intended as the *behind the scenes magic* which should allow for the very high level of radio application portability and seamless interfacing targeted by SCA.

Indeed, SDR adoption didn't only happen within military environments, a swarm of initiatives also proliferated throughout academic institutions and their research labs as the possibility of gaining cheap and quick access to any region of the spectrum was regarded as a true breakthrough by many. Finally, in recent years, industry as well has proposed a wide variety of software defined radio platforms, spanning the full range of SDR implementation paradigms from FPGA to GPP-based systems. Such platforms, intended for both experimentation and development, are typically used to implement rather narrow-band communication standards and however never attempt the implementation of high capacity signals (where high capacity is assumed as $>$ 10 Mbps) on general purpose CPU systems. Actually, the majority of those systems prefers the choice of hybrid architectures where GPPs only act as the overall regulator and information gateway of the system, while all of the heavy processing effort is performed over FPGAs and/or DSPs that are also on-board. A rather popular example of such concept is the LYRtech Virtex-4 FPGA SDR Development Platform (shown in figure 1.1), which features a so-called co-processor SDR architecture (i.e. FPGA + DSP + microprocessor) in a small form factor, all-in-one, computation-embedded solution. It uses CORBA-enabled middleware for all processing devices and runs stand alone without the need for integration with an host computer system. Another quite paradigmatic device is Sundance SMT8146, again a stand-alone system which, developed by Sundance, inc. and mainly targeted at academic institutions, features both FPGA and a DSP based modules (figure 1.2). Recurrent costs and development effort yielded by such class of systems is typically much higher than that of GPP-based counterparts, also SDR code portability is greatly reduced.

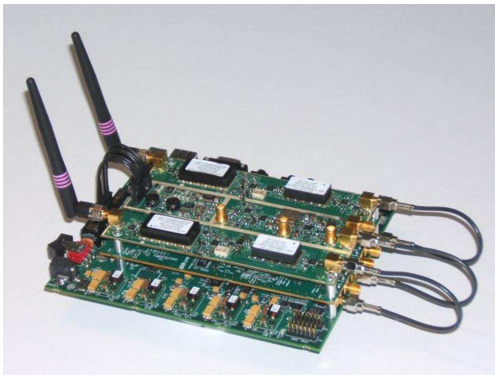

Figure 1.1: *LYRtech Virtex-4 FPGA SDR Development Platform*

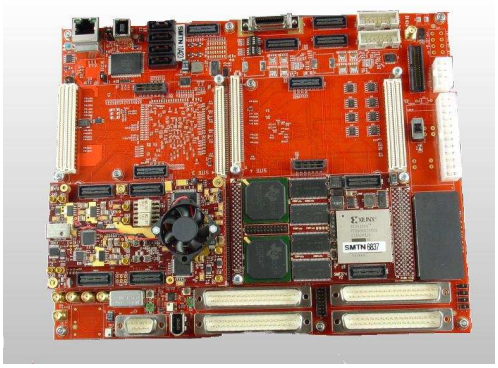

Figure 1.2: *Sundance SMT 8146 SDR Development Platform*

1.2 Academic and amateur research

Above all academic projects, GNU GNURadio [6], an open source community project with remarkable academic participation, and Open-Source SCA Implementation - Embedded (OSSIE) [7], originated and maintained at VirginiaTech, deserve a special mention as long as their GPP-oriented implementation paradigm falls right into the scope of this research. The former is an open source initiative born in 2001, which consists of a software section, the GNURadio framework, and a hardware section produced by Ettus Research LLC [8] named Universal Software Radio Peripheral (USRP). The latter is an open source implementation of the JTRS Software Communications Architecture in the form of an SDR development framework, primarily intended to enable research and education in SDR and wireless communications, but also includes tools for rapid development of SDR components and waveforms applications. The GNURadio framework is a fully open source, class-oriented programming framework where C++ classes are used to implement each and every functional block of the radio signal processing chain (the flow-graph in GNURadio speak) while at runtime the flow-graph set-up and interconnection happens by means of a Python script which is often referred to as the glue of the system. Recently, a graphical tool called GNURadio Companion (GRC) has been implemented and distributed along with the GNURadio source tree, that allows for graphical flow-graph design and instantiation. A full C++ programming architecture has been developed as well in order to make Python language dependence just optional. On the hardware side, starting from the early 2001 prototypes, the USRP system grew rapidly (along with its manufacturing company, ETTUS Research LLC) to a full "ecosystem" of products which today includes as many as 5 SDR peripheral types covering the spectrum from DC to 6 GHz while being capable to handle an instantaneous bandwidth of up to 25 MHz at 16 bit sample resolution (or 50 MHz @ 8 bit). Many independent, even individual, and commercial projects were developed on-board both GNURadio and OSSIE but it was academic wireless systems research that appeared to enjoy such systems the most. Research projects were carried on by using such software environments (very often also autonomous software frameworks directly interfaced to USRP hardware) which implemented a wide variety of communication standards and RF applications. Some examples, taken from an extremely abundant published literature, can be: aeronautical Very High Frequency (VHF) receiver and transmitter

(up to 50 8.33 kHz AM channels) on a SCA-compliant waveform [9], GNSS design, test and analysis [10], RF system verification, validation and measurement [11], communication protocol and device prototyping in Multiple Input Multiple Output (MIMO), cooperative and/or cognitive environments [12], [13], [14], prototyping of radio sensing systems [15]. Other open source community powered implementations also exist. The Open Base Transceiver Station (OpenBTS) [16], is an open source Global System for Mobile communication (GSM) SDR-implemented base station featuring standard 200 kHz GSM traffic and broadcast channels (Gaussian Minimum Shift Keying - GMSK modulation) and being capable of interfacing to standard GSM handsets.

As long as not only academic people happen to love radios, but many individuals around the world have always been fascinated and enthralled by radio experimentation, SDR platforms and implementations also arouse within amateur contexts. This way, such projects were born as High Performance Software Defined Radio (HPSDR) [17], which concentrates on narrow-band, usually analogue, voice radio signals in use for short wave listening and point to point communication by radio amateurs worldwide. The implementation paradigm it adopts is GPP-based and extremely close to that of GNURadio and OSSIE.

It appears quite clearly from examples listed above, that a truly wideband, modern, computation-intensive RF communication chain including state-of-the-art Forward Error Correction (FEC) techniques and being entirely implemented over GPP-based SDRs within an acceptable power budgets is still a target to be reached.

1.3 Commercial systems

Again, also in the industrial SDR domain, which is mostly focused on military applications, typical systems are based upon proprietary hybrid FPGA-DSP architectural solutions, which provide enormous computing resources at the expenses of system power consumption rates, SDR application portability and SDR development costs, for all the reasons that were discussed within the introduction to this thesis. Nevertheless, typical capacities reached by such systems, although significant, and in the Mbps range, remain well below those 10 Mbps we consider acceptable for a truly broadband system. Examples of this fact are state-of-the-art products from major worldwide military SDR players such as the Italian Selex-Elsag and the French Thales group, respectively the *Swave* (figure 1.5) and *FlexNet-Four* (figure 1.6) vehicular software

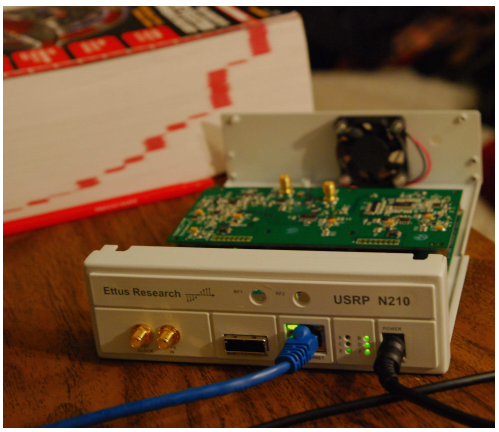

Figure 1.3: *USRP n210, the highest profile USRP currently available from Ettus Research*

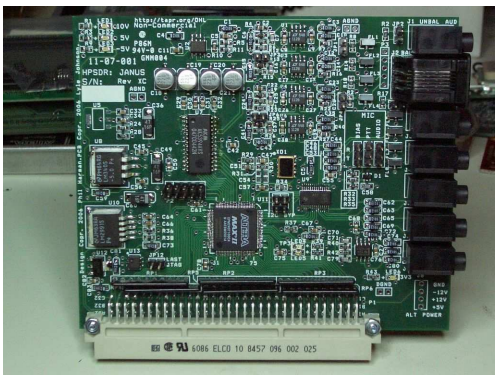

Figure 1.4: *HPSDR hardware peripheral*

Figure 1.5: *Selex-Elsag Swave Vehicular SDR*

radios.

Such systems implement the European Secure SOftware defined Radio (ESSOR) architecture waveforms while also guaranteeing SCA compliance, and obtain maximum capacity (for multimedia applications) of 6 Mbps, while more typical operating data-rates stop at 1.5 Mbps. Such SDR computational performance, when considered in conjunction with the huge computational power available over the presented platforms, is definitely not impressive, this is most likely due to the inefficiency of the middle-ware-encumbered SCA architecture.

1.4 Industrial research

The closest point to the purposes of this research has probably been reached so far by an industrial research project carried out by Microsoft Research Asia in Beijing and called Sora, which plainly stands for "Software Radio". People at Microsoft Research implemented a Peripheral Component Interconnect Express (PCIe) board acting as a radio front-end for transmission and reception (figure 1.7) on-board a standard Personal Computer (PC) (figure 1.8), much alike what happens within the typical GNURadio full-GPP processing + USRP peripheral paradigm. It is a clear, explicit aim of the Microsoft Research Sora project to achieve high performance, state of the art, real-time radio-communication signal processing over GPP-based computing plat-

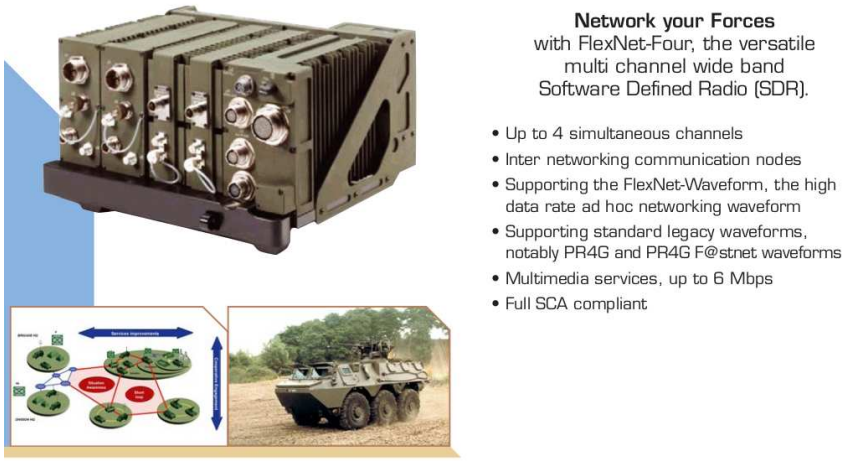

Figure 1.6: *Thales FlexNet-Four Vehicular SDR*

forms such as commodity PCs. Thus, Sora proposes joint usage of multi-threading, look-up tables and Single Instruction Multiple Data (SIMD) programming within a software radio context in order to obtain significant speedups. Within their main work [18], people from the Sora group report some rather good performance numbers obtained while implementing a proof-of-concept software IEEE 802.11 transceiver by means of all the above-quoted techniques. Although good and very interesting, such performance numbers are not easy to evaluate due to the inherently discontinuous nature of 802.11 traffic which can, by itself, in some quite typical conditions, greatly relief the computational burden imposed upon network nodes' radio interfaces.

Within this work, we instead reckoned that multi-thread programming and SIMD instructions are already well known implementation enhancements and can always, rather easily, be applied to any GPP-SDR design. Therefore, based on the assumption that increasing cache size and memory resources on a computing system comes at a much smaller power consumption cost than increasing clock frequency, and therefore offers larger performance improvement margin than any other possible acceleration technique, we tried instead to focus only on the *memory Vs computation trade-off*. We did this by exploring the application of such trade-off to radio signal processing in

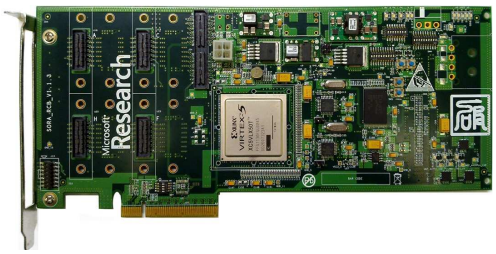

Figure 1.7: *Microsoft Research Asia Sora PCIe SDR Board*

Figure 1.8: *Microsoft Research Asia Sora PCIe SDR Board operating within a standard PC*

depth in order to provide a convenient and rather general SW design criterion which enables fast implementation of any radio chain in a memory-intensive fashion. Finally, it must be adequately emphasized that all the advantages of the Memory Acceleration technique that we are about to introduce *can also be obtained while applying MA along with other computation enhancement strategies* such as multi-threading architectures and SIMD instruction sets, thus leveraging the SDR system performance by the sum of all contributions.

Chapter 2

The memory option: MA research activity

According to all considerations presented in previous chapters, as well as in the quest for an acceleration technique that was both easy to implement and fully-scalable with respect to the available resources, we decided to focus our research effort onto memory-based speedup techniques revisiting classical concepts already known in computer science under the collective denomination of *space/time trade-offs*. In previous literature, space/time trade-offs are intended as a means to reduce the execution time of a certain algorithm either by increasing the degree of HW/SW parallelism of a given implementation (therefore consuming more *space*) [19], [20], or by pre-computing the data produced by some well-determined algorithm and casting it into some tabular form [21] (sacrificing again space, in terms of size of the table to be stored, to gain execution time). Our effort aims instead at obtaining memory implementation of entire radio signal processing algorithms (e.g. convolutional encoders, Reed-Solomon encoders, Viterbi decoders, channel estimators and synchronizers) tending towards the limit where any radio signal processing algorithm can indifferently be implemented either in calculus or just through memory look-ups to suitable tables.

2.1 Description of the Memory Acceleration design rule

2.1.1 System representation and useful quantities

We start our discussion by observing that any radio terminal and, more generally, any system performing signal processing functions, can be represented as the interconnection of a number of constituent functional blocks. Simple systems are arranged as a straightforward cascaded "chain" of elementary blocks, more complicated schemes (possibly with feedback connections) look more like a "mesh" of components and connections, as depicted in figure 2.1 a). Focusing for simplicity on a radio receiver, whatever the mesh of blocks and connection is, the end-to-end signal processing function of our system is equivalent to a mathematical function $f(...)$ which maps a certain amount of soft-valued input symbols (for instance the signal samples collected in a given signal frame) into the corresponding hard-valued demodulated information bits as shown in figure 2.1 b). We call the minimum set of soft channel symbols that can be processed independently from the remainder of the stream the *Minimum Independent Data Set* (MIDS). For the ETSI DVB-T [22] standard (our principal case study) , this would be 4 Orthogonal Frequency-Division Multiplexing (OFDM) frames (i.e. what is called a *superframe* in [22]). We indicate the size (number of items) of the MIDS with symbol l and with A the cardinality of the alphabet each input datum of the MIDS belongs to.

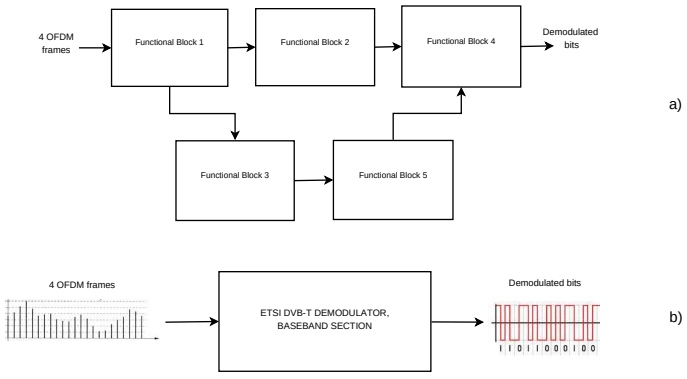

Figure 2.1: *Possible system representations: black box and web of constituent blocks*

The domain of $f(...)$ is then defined as the set of all possible different messages within a MIDS. We also call *input space* the domain of $f(...)$ and C_i the cardinality of such space. Clearly

$$C_i = A^l \tag{2.1}$$

If we could find a convenient analytical expression for the function $f(...)$, we could consider implementing our sample DVB-T demodulator by programming such analytical expression into a computing system via any high-level programming language like C/C++. This would be a *computation-only* implementation of the system, one that is completely located at the *time* end of the time/space trade-off. Such implementation would only (or mainly) take advantage of the *computational* resources being available on a GPP-based platform, with very little attention to the *memory* resources that are available.

After this remark on memory resources, it would be natural to think of replacing our function $f(...)$ with a *tabular* implementation of $f(...)$: a table $t(...)$ containing, for each of the A^l items of the overall input space, the associated output value. This would be a *memory-only* implementation, located at the *space* end of the trade-off, and would not require any (or would require negligible) real-time computation. On the other hand, the size C_i of the table would not be practical for any memory technology available today or in the foreseeable future. The table $t(...)$ could be filled up by running once and forever, at instantiation time (i.e., at the time of initialization or configuration of the terminal) any standard, computation-only, implementation of function $f(...)$ (i.e. the traditional radio system chain) over the entire input space. Such considerations suggest that the path towards optimal SDR implementations lies somewhere in between the two ends, with a hybrid approach that could use *both* computational *and* memory resources to the greatest possible extent (situation graphically represented in figure 2.2).

Let us now come back to the mesh representation of the signal processing functions of our SDR. We call this representation the *0-step* of a procedure underlying MA, that we label *algorithm segmentation*. Such *0-step* may be the direct translation of the signal processing functions described in a communications standard, in a reference implementation, or, in the parlance of SCA-based SDRs [23], in a *waveform*. The implicit assumption in this decomposition is that each of the functional blocks $f_n(...)$ in the mesh is *atomic*, i.e., impossible to break-up in a further mesh of constituent functional blocks. On the contrary, the aim of our *algorithm segmentation* approach is

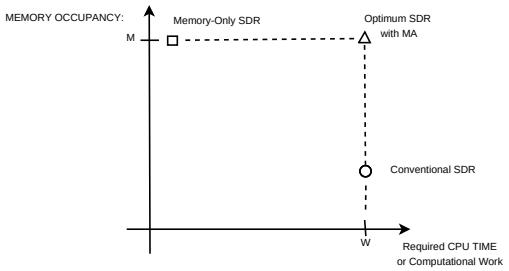

Figure 2.2: *Table boundary after one step, released block is peripheral*

just coming to a further decomposition of a functional block $f_n(...)$, formerly assumed to be atomic, into a chain (or mesh) of constituent sub-blocks $f_{n,p}(...)$, $p = 1, ..., P_n$ whose end-to-end behavior is equivalent to the original function $f_n(...)$. One advantage of this is that the input spaces of sub blocks $C_{i_{n,p}}$ will be different from and significantly smaller than C_{i_n}, provided that the segmentation is performed correctly. Algorithm segmentation cannot be considered as a form of algorithm re-design: as a consequence, algorithm segmentation *does not change the overall computational cost of the segmented algorithm*. We will describe algorithm segmentation with convenient detail in section 2.1.5.

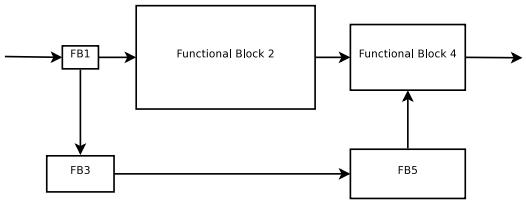

Figure 2.3: *Computational cost weighted functional block representation. Blocks 1 and 4 are* peripheral

2.1.2 Acceleration design

An expedient visual representation of the SDR signal processing mesh is obtained as follows: we call W_n the *computational cost* of the *n-th* functional block (required number of CPU instructions or operations per second (ops)), and we use a graphical

Table 2.1: *MA / MB-SDR symbols and taxonomy*

Symbol	*Meaning*
$f_n(...)$	Computation-only implementation of block n
$t_n(...)$	Memory-only implementation of block n
l	Number of items within the MIDS
A	Cardinality of Alphabet for each item of MIDS
C_{i_n}	Cardinality of input space of block $f_n(...)$
C_{o_n}	Cardinality of output space of block $f_n(...)$
W	Total available computational power
W_n	Computational cost of block $f_n(...)$
W_{TB}	Computational cost of subsystem within table boundary
W_m	Computational cost of subsystem replaced by table m
W_r	Computational cost of not yet memory-accelerated subsystem
Ω_m	Computational cost for handling table $t_m(...)$
M	Total size of available memory
M_m	Total memory footprint of table $t_m(...)$
S_m	Data size of items stored in $t_m(...)$
a	Acceleration factor
η	Acceleration efficiency
η_m	Acceleration efficiency of table $t_m(...)$
I	Overall SDR implementation merit parameter

representation of the SDR in which the size of the functional block is directly proportional to such cost, as in figure 2.3. This gives at a glance an indication of the relative computational weight of each block (function) within the whole radio. Let us also introduce the symbol Ω_m as the total computational cost of *memory management* for table $t_m(...)$, something that has nothing to do with algorithm complexity, but that represents the cost of memory address calculation plus memory access latency (if significant). The latter parameter can be made equivalent to a computational cost by reducing it to CPU time or to equivalent ops/clock cycles.

We come now to the core of the MA technique. Broadly speaking, the aim of MA is aiding the GPP in processing the informative signal through a proper (extensive) usage of memory resources. Such result is obtained by replacing the functional blocks $f_n(...)$ usually implemented according to a purely-computational (*time*) approach (with marginal usage of memory), with pre-computed tables $t_m(...)$ as introduced in 2.1.1. The replacement is done after one or more steps of algorithm segmentation have been carried out, and on the most demanding blocks in terms of computational power only. In subsection 2.1.4, we will introduce the so-called Recursive Table Aggregation Rule (RTAR) that finalizes tables that will have to be implemented into memory, while also calling for algorithm segmentation to be applied upon the convenient functional blocks. In this respect, notice that the *input space cardinality* C_{i_n} of each $f_n(...)$ is usually unrelated to its *computational cost* W_n. Just to make an example, consider the *data deinterleaver* in a DVB-T demodulator, that performs the inverse operation of the interleaver used in the modulator to scatter around the protected bits of an encoded data block in order to protect them from time-correlated errors. The cardinality of the input space of the deinterleaver is the same as that of its own adjacent binary FEC decoder (they bear the same block length), but the computational complexity of the decoder is way larger than that of the deinterleaver.

We already mentioned that, after segmentation, only the most computation-demanding blocks need being implemented in a tabular form. This rule comes from the consideration that the amount of memory resources M is finite, and has to be used in an optimal fashion. Performing memory-acceleration (i.e., tabular implementation) of low-complexity blocks would only result in a waste of memory resources, with negligible impact on processing speed. The MA rule attains an optimum configuration whenever the available memory resources are exhausted, and the maximum number of operations (or the maximum possible amount of CPU time) has been replaced by

memory look-ups implementing the same functions. Replacement of computation-dominated blocks has to follow a hierarchical approach, starting form the most, down to the least demanding, until available memory resources are over.

2.1.3 Table aggregation and cache friendliness

The computational complexity of a tabular implementation $t_m(...)$ is not zero, though. We have to consider in fact the memory management cost Ω_m that can be at times non negligible. From this standpoint, it is apparent that the larger is the number of functional blocks $f_n(...)$ that we collapse into a single tabular implementation $t_m(...)$, the smaller is the total memory management cost Ω of our memory-accelerated implementation. In addition to this, we must also consider that GPP-based platforms have a hierarchical memory structure with smaller and faster caches in the proximity of the computing cores and bigger, slower extended memories in a more peripheral location of the system. The general rule to use efficiently such hierarchical memory arrangement is *storing contiguously in memory information which is used contiguously in time*. This means for instance that a series of consecutive blocks in a processing chain are accelerated very efficiently when their processing is aggregated (as far as possible) into a single table $t_m(...)$ whose internal arrangement reproduces the same cascaded structure of the original chain (RTAR is designed in order to yields this). This happens because if such criterion is observed, either the whole table fits into the CPU cache or any subset of the table is fetched into cache only once and never gets used twice, thus maximizing cache friendliness. Once recognized that aggregating blocks in such a structured way is a virtue *per se*, we introduce in the next subsection a Recursive Table Aggregation Rule. Following this rule, we can on one hand provide the aggregation of as many functional blocks as possible into the same table, while on the other we can perform algorithm segmentation –which still remains a demanding design task– only for those blocks where it is really needed and useful.

2.1.4 Recursive Table Aggregation Rule

Assuming that we have an atomic mesh decomposition of our end-to-end algorithm (our SDR), what is the optimum level of break-up to replace computation-intensive blocks with tables? We try to give an answer to this fundamental question through the RTAR, whose aim is to come to a balanced (optimum) time/space trade-off in the

design of the SDR, possibly providing also cache friendliness. We start by enclosing

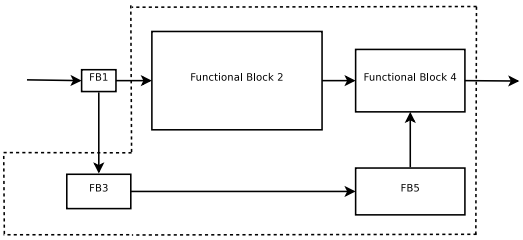

Figure 2.4: *Table boundary after one step, released block is peripheral*

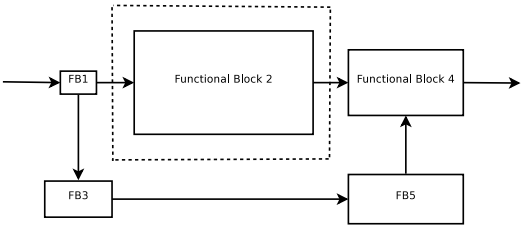

Figure 2.5: *Atomicity limit reached*

the subsystem we intend to memory-accelerate into a closed line that we call the *table boundary* (TB). The connections that crosses the TB represent the input/output interface towards the external world of the subsystem under consideration. An atomic block of the subsystem is called *peripheral* if all of its input *or* all of its output connections cross the TB. Once this is defined, the RTAR proceeds as follows:

1. (Initialization) Define the whole radio the as the (sub-)system to be memory-accelerated. This is equivalent to enclosing the entire radio within a TB. Calculate the size C_i of the table that is necessary to memory-implement the selected subsystem (the whole radio). If the table fits into memory, then go to step 3

2. (Reduction) Identify the computationally-lightest block contained within the TB and *reduce* the subsystem by releasing such block (move it outside the TB as in Fig. 2.4). If the released block is not peripheral, then release also all blocks depending on its output, see Fig. 2.6. Calculate the size C_i of the table

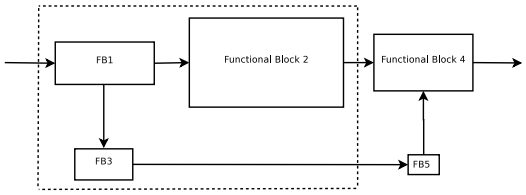

Figure 2.6: *Example of released block (FB5) being non-peripheral. In this case cascaded blocks (FB4) are released as well. At next step of the iterative algorithm, formerly released blocks will be enclosed in the new table boundary*

equivalent to the enclosed system; if the table fits, then go to step 3), otherwise *reduce again* until either i) the table fits (in this case, still go to step 3), or ii) the atomicity limit $f_n(...)$ is reached (figure 2.5). If the latter becomes true, perform *algorithm segmentation* upon the block $f_n(...)$ being currently surrounded by TB and go back to step 2). [1]

3. (Memory-only Implementation) Implement the subsystem being enclosed in the current table boundary by replacing its computation-only functional blocks with an equivalent table $t_m(...)$. If the system still contains blocks not yet implemented in memory, and memory resources are not exhausted, initialize the table boundary for another MA iteration by enclosing all of the remaining computational blocks of the radio system, then go to step 2).

The proposed RTAR rule is admittedly heuristic - an exhaustive approach to algorithm segmentation to find the optimum configuration of the SDR appears unfeasible. Nonetheless, we believe that our rule captures the majority of the achievable MA gain with a manageable approach. In some test cases (conducted upon rather heterogeneous signal processing algorithms), RTAR was shown to provide substantial speedup

[1] Note that, in the case atomicity limit is reached, the block which undergoes algorithm segmentation is the heaviest block of the whole radio, and therefore the sub-blocks obtained from its segmentation still collectively yield the majority of the computational cost of the SDR. Still, as soon as the first of obtained sub-blocks gets released, we cannot guarantee that this keeps true. Thus, whenever one of the sub-blocks obtained from algorithm segmentation is released, it must be checked whether the table boundary encloses a computational cost W_{TB} which is still greater than the cost of any functional block outside the TB. In case this condition becomes false, the TB must be re-initialized to enclose the entire system and the procedure shall continue from step 2).

factors (roughly one order of magnitude) with an acceptable MA design effort.

Cache-friendliness provided by RTAR also constitutes the basis for MA compatibility with parallel programming. Memory access contentions that could indeed happen when loading the required memory table (or table portion) from the external Random Access Memory (RAM) into the core-dedicated caches of a multicore computing system can be made extremely sporadic by performing most of the look-ups within the cache, therefore minimizing the number of fetches being necessary from the RAM. For the reader's convenience, an MA flowchart is sketched in in figure 2.7.

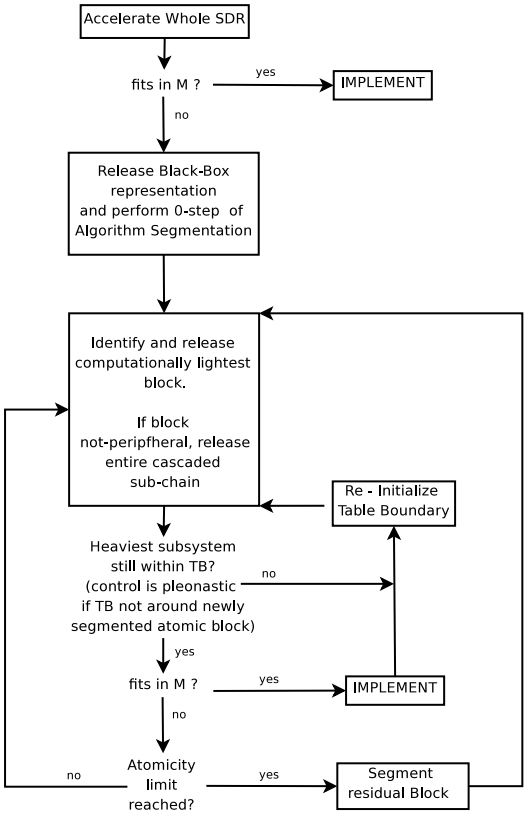

Figure 2.7: *Schematic representation of MA Recursive Table Aggregation Rule. Exit condition on memory exhaustion is not graphically represented for readability*

2.1.5 More on Algorithm Segmentation

As previously stated, *Algorithm Segmentation* is the process of breaking down a single functional block $f(...)$ into its constituent functional sub-blocks or *segments*. This process just identifies the segments within a given block $f(...)$ without actually performing any re-design of the segmented algorithm, so that the computational cost of the segmented system $f(...)$ is not changed. A *segment* is any sub-system of $f(...)$ with a specific and identifiable MIDS over one or more input connections. The output yielded by the processing of such MIDS (the segment output) is in its turn input to another segment (with another, possibly different MIDS) which concurs to build up $f(...)$ as a whole. The gain of the process of segmentation lies just in the difference between the cardinality C_i of the overall MIDS of $f(...)$, and those of the constituent segments, $C_{i,m}$. The MIDS of the segments are often (much) smaller than that of $f(...)$. Typically, the overall MIDS has a size that is given by the *Least Common Multiple* of the segments' MIDS. Therefore, when considering (2.1), it turns out immediately that the set of tabular implementations $t_m(...)$ of the segments is dramatically much less demanding than the (global) table $t(...)$ of the whole subsystem, in terms of memory resources.

Considering the extreme variety of architectures and functions of the signal processing algorithms in an SDR terminal, trying to find an optimal segmentation/aggregation configuration is very hard. We can just say that the best algorithm segmentation is the one providing the smallest *granularity of input spaces* of the obtained sub blocks. This is true because the smaller the granularity is, the closer the RTAR will manage to bring the total memory occupancy of the MA-ed SDR to the memory capacity of the computing platform M. Broadly speaking, the more sub-blocks N_{sb} algorithm segmentation obtains from the given block, the better algorithm segmentation was performed.

To sum up, the joint action of algorithm segmentation and RTAR (i.e., the gist of the MA concept) is to i) decompose the given SDR system down to the finest possible level of computational granularity; ii) generate a re-implementation which uses the available memory resources in order to perform as much computation as possible by means of memory look-ups, and iii) do it with the smallest possible computational cost of memory management.

2.2 Application and performance metrics of MA

2.2.1 MA compatibility with different acceleration techniques

All performance results that will be presented in section 2.3 were obtained by applying MA alone, i.e. by making use of no other performance enhancement technique such as low level (Assembler) programming or code parallelization. Still, such implementation techniques are fully compatible with MA. Compatibility with low level programming is trivial and does not deserve any discussion. For parallel implementation instead, a possible issue lies in the concurrent access to a certain memory area from multiple computing cores. This may call for some form of collision control and consequent performance bottlenecks. Such problem can be substantially mitigated through the "cache friendliness" approach that we mentioned above. Multicore/multiprocessor GPP-based platforms often feature cache memories which are dedicated to each single core as depicted in figure 2.8. Such memories are independently accessed by each of the cores, so that the probability of collision events is zero. Collisions may instead happen when loading the required memory table (or table portion) from the external Random Access Memory (RAM) into the core-dedicated caches as shown in figure 2.8. Appropriate use of a cache-friendly table structure will make these fetches extremely sporadic. Should access contention happen at the RAM-level, it would be rare enough not to degrade system performance. Practical proof of this is provided in [24].

2.2.2 Performance evaluation

To quantify the performance in terms of processing speed-up of the MA technique, we define the *acceleration efficiency* for any functional block $f(...)$ as

$$\eta = \frac{\sum_{n=0}^{N_{sb}-1} W_n - \sum_{m=0}^{N_t-1} \Omega_m}{\sum_{m=0}^{N_t-1} M_m} \tag{2.2}$$

where N_{sb} is the number of the sub-blocks $f_n(...)$ obtained after algorithm segmentation which are implemented in tabular form, and N_t is the number of tables that will be used to produce such an implementation as resulting from the application of RTAR. In other words, the acceleration efficiency η is the ratio between the computational effort being saved by means of the resulting memory-based implementations (reduced by the amount of computational work needed for table management) and the total

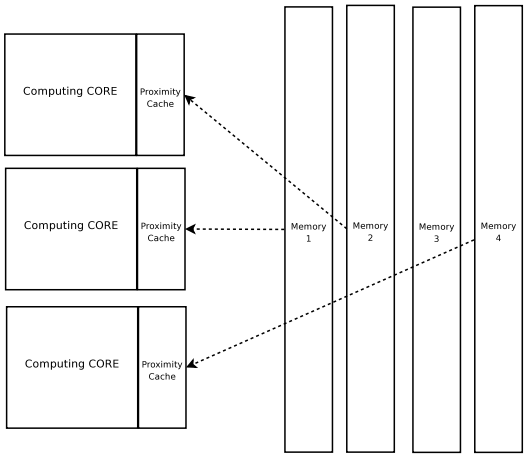

Figure 2.8: *Multiple processing cores, their dedicated caches and table loading from RAM to the core-dedicated cache*

memory footprint being required. A negative value for η indicates that the chosen MA design will *reduce* system performance. Once the acceleration process is completed, it is possible to calculate the obtained *acceleration factor* a as

$$a = \frac{W_r + \sum_{n=0}^{N_{sb}-1} W_n}{W_r + \sum_{m=0}^{N_t-1} \Omega_m} = \frac{1 + \sum_{n=0}^{N_{sb}-1} W_n/W_r}{1 + \sum_{m=0}^{N_t-1} \Omega_m/W_r} \tag{2.3}$$

where W_r accounts for the total computational cost of the remaining blocks which where not implemented in memory.

As previously stated, different algorithms offer different opportunities for segmentation. The consequence of this statement is that it is very difficult to give an upper bound for a. Still, a naive and probably loose upper bound for a can be found assuming that the whole radio can be memory-accelerated. In such condition we get

$$a_{max} = \frac{\sum_{n=0}^{N_{sb}-1} W_n}{\sum_{m=0}^{N_t-1} \Omega_m} \leq \frac{\sum_{n=0}^{N_{sb}-1} W_n}{\sum_{m=0}^{N_t-1} [L_m + (i_m - 1)(x + \sigma)]} \tag{2.4}$$

where N_{sb} is now the total number of computational blocks within the segmented system, L_m is the access latency for each table (that depends on the table size and on the chosen implementation platform), i_m is the number of inputs to each table, x is the computational cost for one multiplication by a constant, and σ is the computational

cost of one sum with a variable. All such quantities, including L_m, can be expressed in terms of number of equivalent elementary operations, or of required CPU time. The term $(i_m - 1)(x + \sigma)$ is a lower bound for Ω_m that only considers the simplest elementary computation of the memory address.

2.3 A case study: the DVB-T Viterbi decoder

2.3.1 MA design for the Viterbi decoder

We wish to describe in this section the results we obtained by applying the MA technique to a fully software, GPP-based implementation of a an ETSI DVB-T [22] receiver called SR-DVB [25], [26], which will be instead described as a whole in section 3.4. Our starting point was an optimized C/C++ source code with no parallelism, developed in-house with a conventional computation-dominated approach. In such reference implementation, the demodulator turned out to run on a set of pre-recorded input signal samples roughly 10 times slower than the real time. R-DVB included all of the ETSI DVB-T demodulation functions (except synchronization and equalization), starting form a stream of previously synchronized and equalized baseband I-Q samples acquired from a standard broadcasting band, and delivering the demodulated MPEG Transport Stream (TS). Our GPP test and reference platform is just a run-of-the-mill Intel quad core Q9400 CPU clocked at 2.66 GHz, our software does *not* make any use of parallelization: all code is single threaded and *only one of the four cores* available on the platform is actually used.

We set forth to apply MA and see if an acceleration factor of at least 10 could be gained, thus allowing for real-time processing of a DVB channel. An all-memory implementation of the entire ETSI DVB-T demodulator as a single memory table (i.e. by applying no RTAR cycle) was of course not feasible. Therefore, we profiled the computational cost of the conventional demodulation chain on a block-by-block basis (akin to what appears in Fig. 2.3), thus performing what we called the *0-step* of algorithm segmentation. It was immediately clear that the K=7 Viterbi Decoder (VD) [27] included in the demodulation chain [22] was by far the heaviest block of the system (followed at a large distance by time/frequency signal synchronization and FFT-based OFDM demodulation) accounting for about 80% of the CPU effort. According to RTAR, the VD was therefore meant to be the first block to undergo memory

acceleration. Such activity produced a Memory Accelerated, novel implementation of the VD itself (the MA-VD, to be further detailed in the following) that relies almost entirely upon memory resources. The MA-VD, together with the subsequent MA implementation of the signal synchronization functions described in section 2.4, was the enabling factor to to obtain a real-time, fully software ETSI DVB-T receiver on a low budget off-the-shelf GPP [24].

After identifying the VD as the most computation-intensive block, we started applying our technique of segmentation and aggregation as in 2.1.4. A classical decomposition of the VD is shown in figure 2.9 where all of the functions are meant to be implemented through pure computation. The blocks of such decomposition implement four different functions [28]: i) Add-Compare-Select (ACS), ii) refresh of the survivors (the memories of the tentative decoded bits for each state of the decoder), iii) update of the accumulated metrics (weights) for each of the survivors, and iv) selection of the likeliest (smallest metric), associated with corresponding output of the decoded bit after a certain decoding latency (depth). After such a coarse-grain segmentation, the input space cardinalities C_{i_n} were still far too large to fit into the available memory. We iterated RTAR until segmentation produced the much finer-grained implementation of figure 2.10, with convenient table boundaries. The constituent sub-blocks then were:

- Branch Metrics Computer (BMC)
- Sum of branch metrics to the accumulated metrics (ADD)
- Accumulated metrics comparison and selection (C & S)
- Storage and update of accumulated metrics (AMM)
- Storage and update of survivors (SM)

The sub-blocks come in number of 64 since the decoder has 64 internal states corresponding to the number of states of the DVB-T convolutional channel encoder. We finally were awarded with candidate tables having reasonable and practical C_i. RTAR was then ready to start re-aggregation of the segmented system into larger tables intended for direct memory implementation. The configuration we attained at, after a number of further RTAR cycles is shown in figure 2.11, where the distinction between memory-only (*space*, $t(...)$) and computation-only (*time*, $f(...)$) implementation is

clearly shown. In such implementation, the ACS function for as many as 4 states is lumped into a single memory look up. Another memory table implements the selection of the likeliest state, while update of decoded bit memories and of current metrics for all states is still being performed through pure computation by the $f(...)1$ and $f(...)2$ blocks respectively. This arrangement is the result of a judicious allocation of memory and computational resources, and the degree of its optimality depends on the particular computing platform that we assumed. More *time* blocks could have been implemented as *space* blocks with a somewhat more "extreme" exploitation of available memory (that we did not want to pursue).

The final result of MA design is that the computational cost required to perform the single memory look-up is by far smaller than the sum of all elementary operations that would be needed in order to derive in a computation-dominated (i.e. time-dominated) fashion the required result. For this reason we may also claim that the described approach is also *power-saving*, and reduces the power efficiency gap between SW and HW implementations of the radio terminal, without losing anything in reconfigurability and flexibility of the system. This comes of course at the cost of larger memory occupancy.

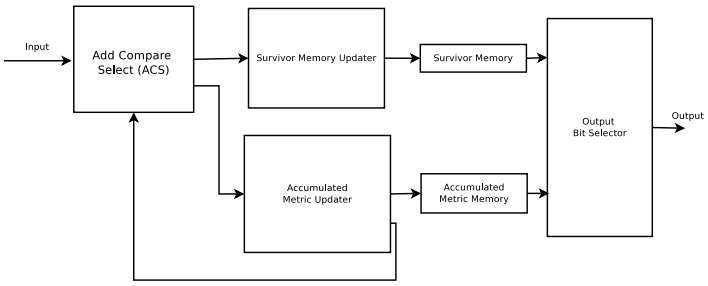

Figure 2.9: *Classical functional block decomposition of the Viterbi Decoder. All displayed functional blocks are implemented through pure computation*

On the reference platform described in section 2.3, the processing (decoding) of a fixed amount of data (namely 631, 701, 504 bits), by the standard C/C++ computation-dominated implementation of the DVB-T VD took about 66.56 seconds, thus yielding a computational requirement of 101.31 nanoseconds per decoded bit. The MA implementation above takes on the contrary 6.35 seconds (10.05 nanoseconds per decoded bit), for a resulting acceleration factor $a_{MA-VD} = 10.4$. The total memory occupancy

of the memory-accelerated implementation is 50.0 MiB which, by today's standards, is negligible on a GPP-based platform. Residual, non memory-accelerated computation is responsible for a computation time of 0.5 seconds, while decoding the same amount of data mentioned above, resulting in a computational load of 0.79 nanoseconds per decoded bit. Such figure is indeed a good estimate of what we called W_r in 2.4. The sum of table-management computational costs Ω_m associated with all the tables implementing the VD is therefore 5.85 seconds, equivalent to 9.26 nanoseconds per decoded bit. The acceleration efficiency η for our implementation[2] turns out to be

$$\eta_{MA-VD} = \frac{101.31 - 0.79 - 9.26}{50} = 1.82 \qquad [\frac{nanosec/decoded\ bit}{MiB}] \tag{2.5}$$

The benchmark CPU exhibiting such acceleration result has a 6M cache size. Tests with smaller CPUs featuring smaller caches (3M, 2M, 1M, 512K) exhibited quasi-linear scaling of the total CPU time required by the MA implementation with the cache size, suggesting an extremely strong dominance of cache size over clock frequency. Considering the nature of the MA technique, such dominance was expected (indeed it was aimed-for), still, the quasi-linear behaviour was pretty surprising.

It is not possible instead to provide an estimate for a_{max} as such maximization is directly dependent on the optimality of algorithm segmentation for the considered algorithm. Optimality of algorithm segmentation for a given DSP algorithm is in fact still an open research issue at the time of writing.

2.4 Other memory-accelerated algorithms

In the last part of this chapter, some results obtained while applying MA to other algorithms are worth to be mentioned as well. Within the synchronization section of the SR-DVB receiver, a carrier frequency fractional (i.e. expressed as a fraction of OFDM subcarrier spacing) offset corrector, which used to be implemented by means of pure calculus, was accelerated by a factor 46.3 after undergoing MA. In short, within this very basic MA application, a single, computationally-implemented, oscillator generating any complex tone that could be required to compensate the estimated fractional offset is *algorithm-segmented* down to a set of oscillators capable of generating

[2]The presented performance results were obtained by running the algorithm under Linux Fedora 10 and compiling the source code with the $g++$ compiler, version $4.3.2 - 7$

only a single frequency. Generated complex tones are spectrally spaced by two times the standard deviation of the fractional frequency offset estimator, thus, the oscillator set is kept finite and small. Each of such sub-blocks (sub-oscillators) is implemented by means of a memory table, therefore obtaining full memory implementation of the fractional frequency offset corrector block as shown in figure 2.12.

We believe that this example, although very simple and providing very little or no innovation wrt current common practice, suggests a different and much more general way to think of Look-Up Tables (LUTs) in SDR implementation. Such generality, as described in section 2.1, can be extremely useful when it comes to use memory in order to speed up much more complicated algorithms.

As discussed above, functional blocks inherently working on large amounts of data (i.e. featuring big MIDS) require well-designed segmentation in order to be conveniently accelerated. When such blocks do perform very basic operations on such broad data sets, they are difficult to segment. Still, it was possible to obtain quite a significant memory-acceleration of a computationally heavy algorithm presenting such challenges, namely the OFDM time and frequency offset estimator described by van de Beek, Sandell and Borjesson in [29]. Memory-accelerated implementations obtained so far provided acceleration factors as big as 11.996 which are expected to grow by means of further development and will be described in future MA-related works. Reference HW/SW platform for this implementation is the same as described above.

We believe that acceleration factors of about one order of magnitude in terms of computational efficiency obtained by applying MA to the two highly diverse signal processing algorithms described above (namely an hard-valued Viterbi decoder and a soft-valued, correlator-based OFDM synchronizer), already suggest the generality of MA approach as well as its applicability to an extremely wide variety of radio signal processing algorithms.

For such a reason, we propose MA as an implementation technique for radio signal processing within SDR systems which can provide substantial boost to their performances and therefore move SDR technologies much closer to market segments they are currently excluded from, because of their poor energy efficiency.

An issue to be discussed for memory-dominated SDRs is the initialization process. Whenever the terminal has to be (re-)configured, the memory tables in the space-implemented blocks have to be (re-)initialized. This can be done by i) bootstrapping

their contents from a local pre-programmed ROM, ii) running once and for ever the computation-only version of the same block on all of the possible values of the input space, or iii) getting the contents via another available communications link from a remote table. Each approach has pros and cons: i) is very fast but increases the memory resources demand on the platform; ii) may be slow since exhaustive calculation of all entries in the memory may take a lot of computing time; iii) does not require any additional computation/memory resource, but is subject to the availability of a live communication link. This by no means diminishes flexibility of the MA SDR, but is something that nonetheless has to be considered.

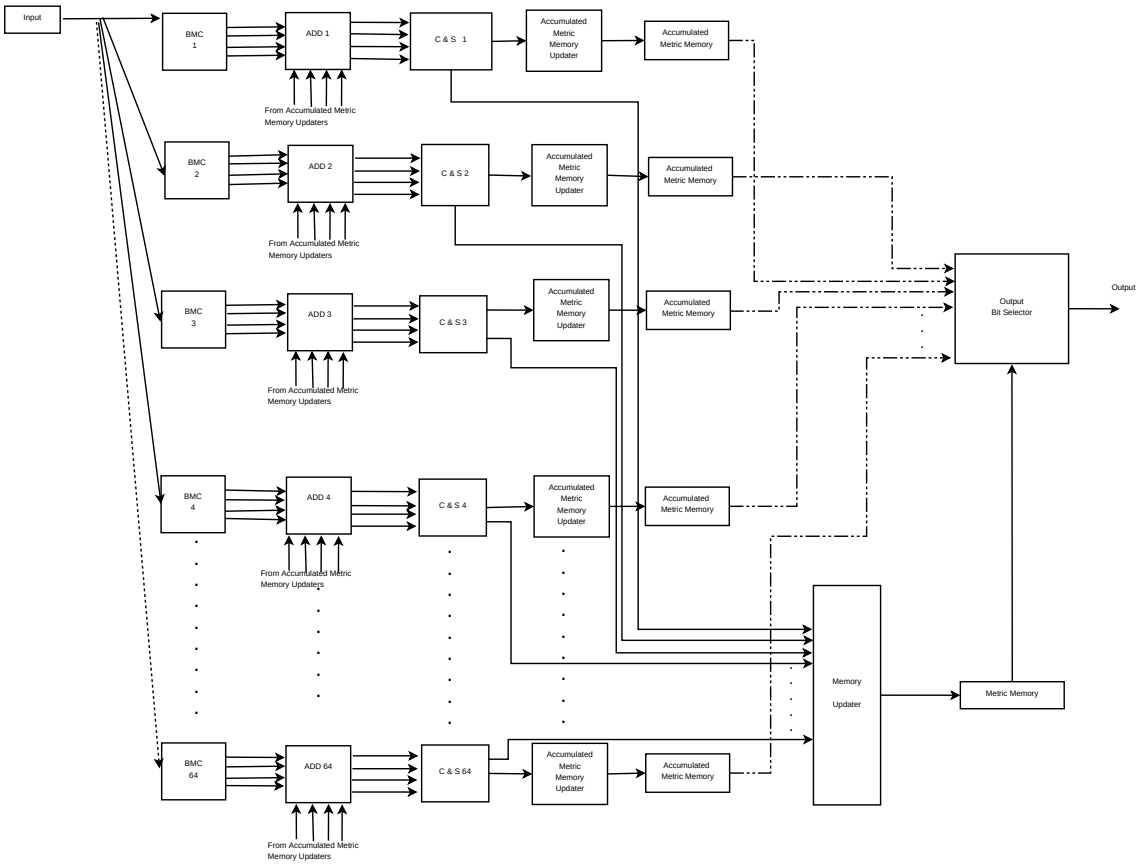

Figure 2.10: *Computation-only implementation of our Viterbi decoder, right after undergoing algorithm segmentation. f(...) indicates a purely computational implementation*

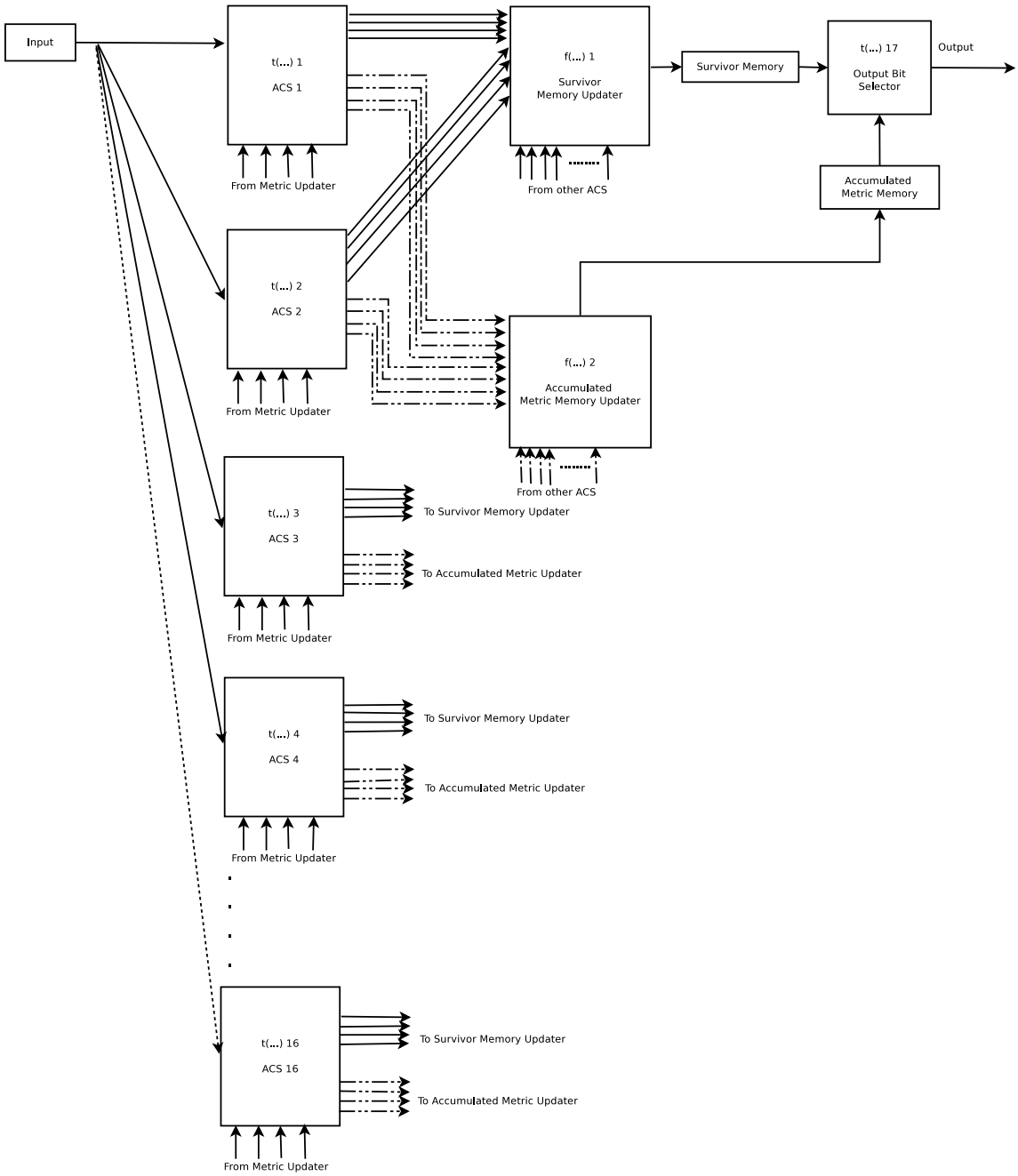

Figure 2.11: *Memory-accelerated Viterbi decoder as returned by RTAR. f(...) indicates a purely computational implementation, t(...) a memory table*

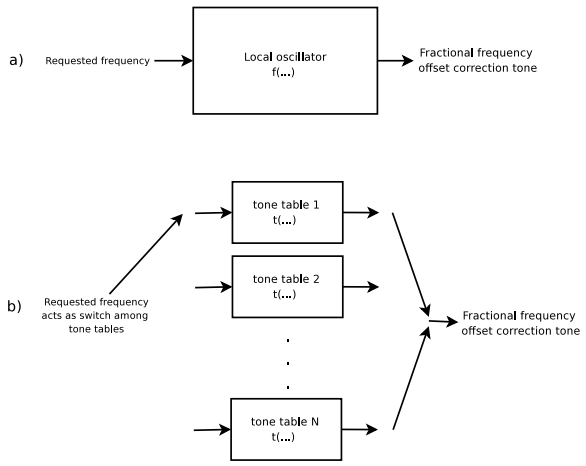

Figure 2.12: *A very basic MA application: segmentation of a computationally-implemented local oscillator a), into a finite set of tabled complex oscillations b)*

Chapter 3

MA application to real-world systems

As a proof of concept of the MA SDR programming technique we provide a description of software defined radios which were developed and implemented according to the MA rule, along the MA development and research path.

All such implementations have close relations to the ETSI DVB family of radio communication standards. This choice was dictated by the need to test MA concepts and strategies against real-world SDR implementations of radio systems which had to be both state-of-the-art (i.e. feature the most modern FEC systems and modulations) and as purely physical layer as possible, in order to minimize all possible uncertainties and interference in performance assessment that could possibly be yielded by interaction with upper layer protocols.

3.1 The ETSI DVB-T transmission standard, some PHY-layer notes

Digital video broadcasting (DVB), in its terrestrial version (DVB-T), is currently the most widely deployed system for delivering standard and high definition video content to digital TV users worldwide. Although born as a European initiative, with the standardization process led by the European Telecommunications Standards Institute (ETSI) [22], DVB-T is today already deployed or adopted in more than 70 countries [30].

The main reasons for the almost worldwide application of ETSI DVB-T lie in the

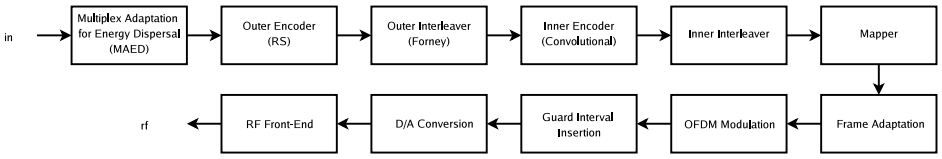

Figure 3.1: *Functional block scheme for the standard ETSI DVB-T transmitter [22].*

high spectral efficiency and in the robust multipath resistance due to the Orthogonal Frequency Division Multiplexing (OFDM) modulation technique.

The ETSI DVB-T system allows one baseband motion picture expert group-2 (MPEG2) transport stream (TS) to be transmitted over the air. (Optionally, two TSs can be combined through the use of hierarchical modulation techniques.) Each TS is designed to carry multiple audio/video and/or data channels. Distribution over single frequency networks (SFNs) is also supported.

3.1.1 Analysis of ETSI DVB-T functional blocks

In this subsection we will briefly analyze the structure of the functional blocks that adapt the baseband MPEG2 signal to the typical multipath radio frequency channel, as described in [22].
Only the transmitter device will be considered, as receive chain blocks are mainly symmetrical to the transmitting ones.

Fig. 3.1 shows the functional block diagram of a DVB-T modulator. Each function block is shortly described as follows:

- the *multiplex adaptation for energy dispersal (MAED)* is the first block of the system, thus it receives the baseband MPEG2 input stream organized in fixed length packets. It then removes time domain correlation from its byte-wise input by performing a bit-level XOR with a pseudo-random binary sequence (PRBS). The proper PRBS is generated via a linear feedback shift register (LFSR) by the generator polynomial:

$$p(x) = 1 + x^{14} + x^{15} \tag{3.1}$$

The initialization of both scrambler and descrambler is bound to the MPEG2 SYNC byte.

- the *outer encoder* performs Reed-Solomon (RS) encoding at byte level and in systematic form. The chosen code is an RS(204,188), obtained by shortening an RS(255,239). The main purpose of the RS code is to mitigate the impact of the error bursts produced at the receiver side by the Viterbi algorithm. Such RS code is capable of correcting up to 8 random erroneous bytes in a received RS word that is 204 bytes long.
  The $GF(2^8)$ Galois Field the code operates within is generated by the following field generator polinomial:

$$p(x) = x^8 + x^4 + x^3 + x^2 + 1 \tag{3.2}$$

  The code is instead generated by:

$$g(x) = (x + \lambda^0)(x + \lambda^1)(x + \lambda^2)...(x + \lambda^{15}), \lambda = 02_{HEX} \tag{3.3}$$

- the *outer interleaver* aims to minimize time correlation between the residual errors at decoding time, it is implemented by a convolutional interleaver based on the Forney approach.
  The interleaved data bytes are composed of error protected packets and delimited by inverted or non inverted MPEG2 sync bytes;

- the *inner encoder* performs convolutional encoding and puncturing to achieve quasi-error-free (QEF) transmission.
  The mother convolutional code, running at a coding rate $r = 1/2$ has generators: $G_1 = 171_{OCT}$ and $G_2 = 133_{OCT}$;

- the *inner interleaver* provides support for hierarchical modulation and fairness in assigning the payload bits to the OFDM carriers, i.e., it prevents certain bits from the original TS from being constantly assigned to the same set of OFDM carriers with unsatisfactory signal-to-noise ratio (SNR). It is made up of two different interleavers, named "bit-wise interleaver" and "symbol interleaver", each driven by different permutation laws and working on different data blocks;

- the *mapper* modulates the encoded bits onto QPSK, 16-QAM, and 64-QAM constellations. Both non-hierarchical and hierarchical modulations are supported, thus allowing two different TSs with different error performance to be transmitted simultaneously;

- *frame adaptation (FA)* provides insertion of all needed reference signals: pilot carriers, used for synchronization and channel estimation, and carriers conveying information upon the transmission parameters being implemented. This operation is called transmission parameter signaling (TPS);

- the *OFDM modulation* block adds virtual carriers, reference signals and performs an inverse fast Fourier transform (IFFT) with 2048 (2k) or 8192 (8k) subcarriers.;

- *guard interval insertion* inserts the guard interval (cyclic prefix) required by the duration of the channel impulse response.

- the *digital-to-analog (D/A) conversion* interpolates the samples, thus producing an analog signal;

- the *RF front-end* shifts the basesband I/Q analog signal to the proper carrier frequency of the desired TV channel.

3.2 Soft-DVB, a software defined DVB-T modulator

First obtained by applying basic space-time trade-off concepts, Soft-DVB is a fully software, real-time, ETSI DVB-T modulator which later, through full application of the MA design rules, reached Ultra Low Voltage (ULV) processor compatible computational performance. It implements either a 7 or an 8- MHz DVB-T channel with non-hierarchical 16-QAM mapping, convolutional coding rate 2/3 and OFDM guard interval 1/4. As stated in [22], this yields a useful bitrate for our transmission of 11.612 Mb/s. The spectrum of the output OFDM digital signal is presented in Fig. 3.2. As can easily be verified, its shape is consistent with typical spectra of OFDM transmission schemes.

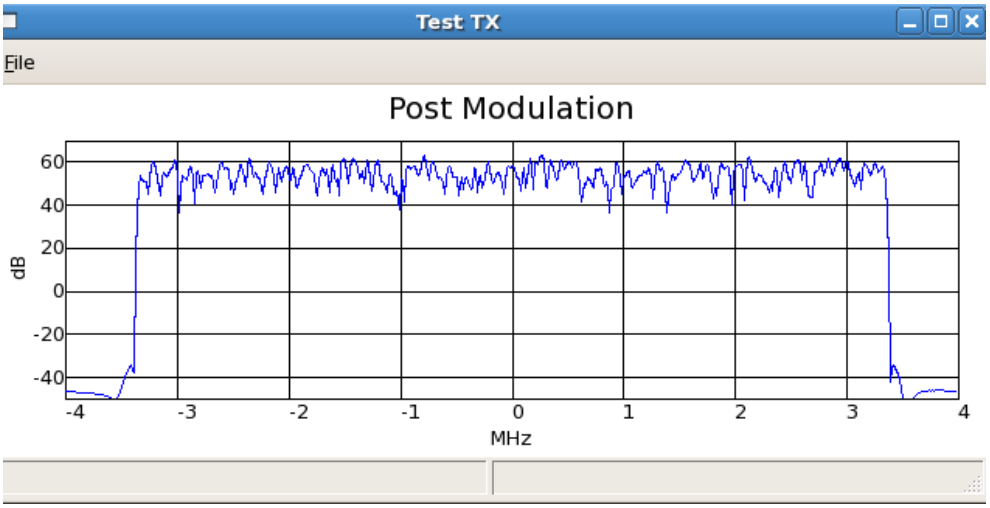

Figure 3.2: *Spectrum of the digital signal at the output of the Soft-DVB SDR*

Experimental tests have been performed using several different receivers including: i) some off-the-shelf digital TV set-top-boxes and ii) several typical USB-pen DVB-T receivers. The output of one of the used test receivers (namely the Access Media STB L3012) is shown in Figure 3.3. All receivers can easily and quickly acquire the MPEG TS synchronization byte and report a BER below 10^{-9}.

The very first Soft-DVB version used to run in 6.8 times the amount of time required by real-time transmission. After initial application of basic space-time trade off concepts and some other computational improvements, the Soft-DVB SDR became capable of running in real-time on a 3.0- GHz Intel® Pentium IV processor, draining 83% of its computational power [31].

We later considered that managing to run a fully standard DVB-T modulator over a 2.5W Thermal Design Power (TDP) embedded CPU would provide good proof for our MA concept within an embedded context as well as constitute a nice and useful piece of SDR equipment. In fact, such a system would enable to quickly set-up an extremely agile and cheap DVB-T broadcasting station using only off-the shelf hardware. An architecture was thus set-up as in figure 3.4 in which all baseband computation is performed over an Intel Atom N270 CPU, located on-board a GMB-N270 Mini-ITX

Figure 3.3: *Output of the Access Media STB1230 test receiver fed by the Soft-DVB transmitted signal*

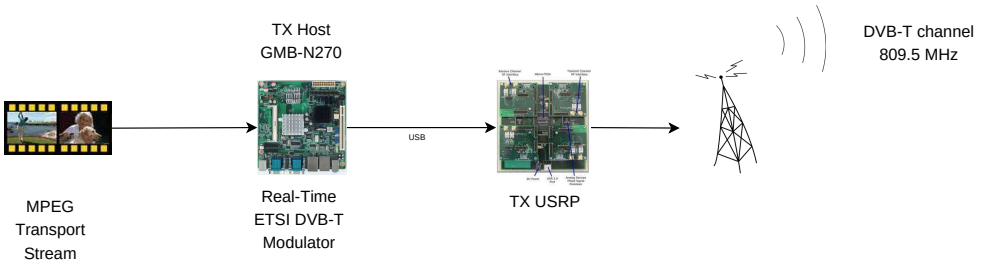

Figure 3.4: *Soft-DVB SDR over the Atom embedded-class CPU. Development setup*

embedded board equipped with 1 GB DDR2 RAM [32]. Baseband signal samples travel along an USB 2.0 bus to the GNURadio project [6] USRP where they are interpolated, converted into the analogue domain, frequency-shifted to the desired DVB-T channel and suitably amplified.

Initially, transmission and reception tests could only be run in *virtual time* as long as, before extensively applying MA, the fully loaded Atom N270 CPU was 6.3 times slower than the real-time. Recursive application of RTAR to the full modulator chain identified the *IFFT*, the *convolutional encoder*, the *OFDM framing*, the *inner interleaver* and the *outer FEC encoder* (Reed-Solomon encoder) as the heaviest blocks in decreasing order of computational weight. It was decided to skip the IFFT block as long as it was implemented since the beginning by using the FFTW library [33] which is itself very efficient and because research on an MA-optimized FFT algorithm, due to its importance, is considered a separate, stand-alone research effort. Proceeding in applying RTAR to the remainder of the chain allowed for memory-implementation of the computationally heaviest blocks in the order mentioned above. After applying MA, the target ETSI DVB-T modulator got capable of running on the chosen ATOM N270 CPU in real-time, while absorbing only about 60% of its total computing capabilities. Overall memory footprint of the MA-implemented system is 342 MiB. Such two quantities are shown in figure 3.5 as they were measured by the Linux Table of Processes (TOP) utility during program execution.

It is worth to mention that RTAR wasn't halted on either of the exit conditions described in 2.1.4 (i.e. on exhaustion of memory resources or on having memory-accelerated the entire chain) but its application was simply stopped when the per-

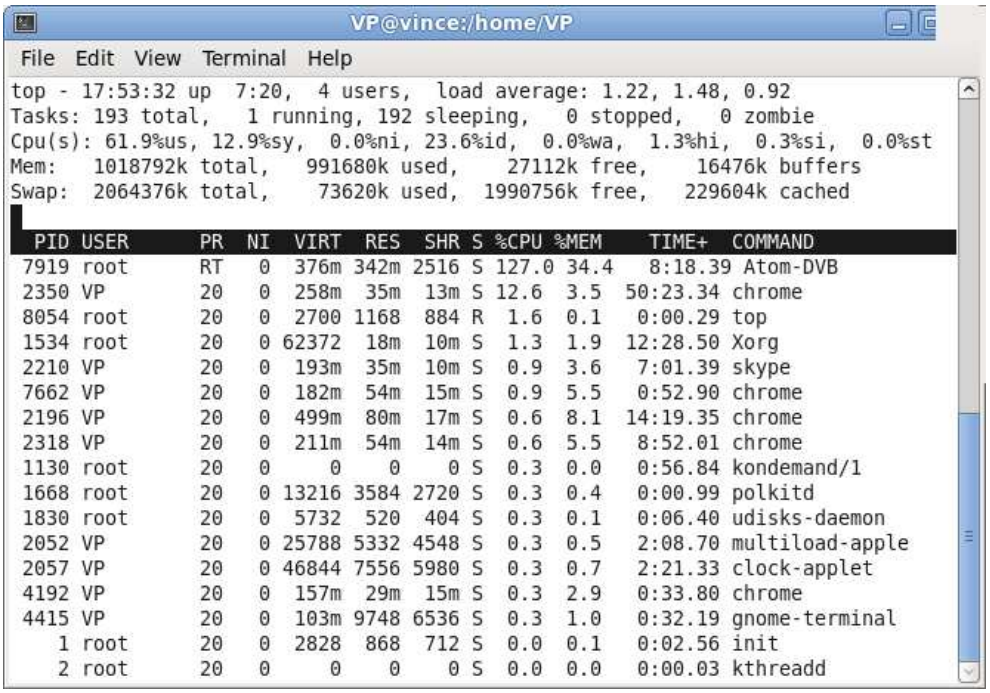

Figure 3.5: *Soft-DVB SDR over the Atom embedded-class CPU. Performance Figures. Total CPU absorption is about 60%. Note that, in the representation adopted by the Table of Processes (TOP), Unix utility, full CPU occupancy for a CPU featuring Intel HyperThreading technology (virtually dual-core) as the Intel Atom N270 is marked as 200% usage*

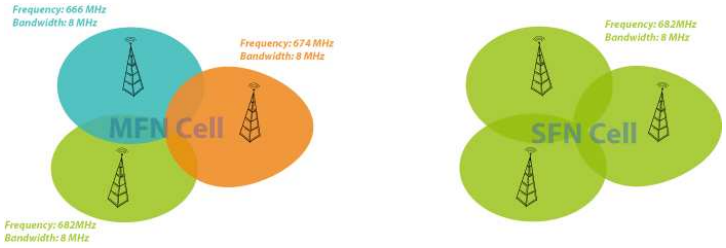

Figure 3.6: *Classical MFN Vs SFN configuration. Courtesy of Enensys*

formance of the accelerated chain was considered significant, that is after achieving real-time with a robust safety margin. Further (most likely marginal) improvement is therefore possible by proceeding in applying RTAR. Moreover, as long as the non-MA implementation of the FFT is currently by far the computationally-heaviest block of the modulator, a substantial performance boost is expected when the MA-FFT block (under development at the time of writing) will be moved to the chain. This result was live-demoed within GLOBECOM 2010 demo session, held at IEEE GLOBECOM 2010 international communications conference in Miami, FL, USA.

3.3 Soft-SFN, a software-defined Single Frequency Network

Latency issues as described in section are among the most critical aspects that should be taken into account when it comes to implementing a full-software, GPP-based SDR. A rather challenging implementation wrt signal synchronization and latency, which naturally arises from choosing the DVB family of standards as a testing field for SDR and MA technology, is the classical DVB-T Single Frequency Network (SFN). SFN, indeed a typical operating mode for today's DVB-T networks, is a particular broadcasting configuration in which all transmitters share the same frequency without destroying each other's signal, therefore dramatically increasing the network's spectral efficiency wrt Multiple Frequency Networks (MFNs), as depicted in figure 3.6.

This is possible by exploiting the multipath resistance which is inherent to the OFDM modulation, as long as, under certain conditions, the potentially interfering

signal originated by a secondary transmitter can be considered equivalent to an echo of the main transmitter's signal and thus turned into an useful information source. The required conditions for this to happen are that the SFN's transmitters are precisely aligned in phase, frequency and time. Particularly, the mutual timing offset at the receiver side must be significantly smaller than the *guard interval* duration of the transmitted signal, which, in Soft-DVB transmission mode, is set at $64\mu s$. As long as GPP systems involve, by their own nature, a lot of latency issues for the reasons which were described in section , suitability of GPP-based SDR for applications involving SFNs or similar, highly synchronization-critical architectures might be seriously questioned. On the other hand, obtaining extremely flexible and cheap modulators based on embedded-grade GPPs that are also SFN-capable could be an extremely interesting possibility in a wide variety of application scenarios. We therefore decided to attempt a full-software SDR SFN implementation and started by assessing the latency imposed by a standard Linux kernel over a commodity PC upon a synchronization signal being distributed via Internet Protocol (IP) over Ethernet, which we found to be safely compatible with practical DVB-T guard intervals. IP over Ethernet was chosen as the means to distribute the timing synchronization among the transmitters due to the extreme cost-efficiency that results from being able to time-synchronize and feed a small-scale SFN with broadcast content without having to use a dedicated satellite downlink or Synchronous Digital Hierarchy (SDH) transport network (as in current implementations).

After setting-up the two-transmitters architecture shown in figure 3.7, we found that our test-bench SFN worked steadily with mobile, hand-held receivers always accepting and correctly demodulating its signal even while they were being rapidly moved throughout the lab room, flooded with signal from both broadcasting nodes. A video clip of the experiment is available online at [34]. We believe this result demonstrates the feasibility of full-software OFDM single frequency networks. Also, as long as both timing and broadcast content are distributed through an IP over Ethernet architecture, this low-cost, alternative distribution system for content and timing synchronization is validated as well. This distribution strategy is expected to be both practical and highly competitive for small, localized SFNs.

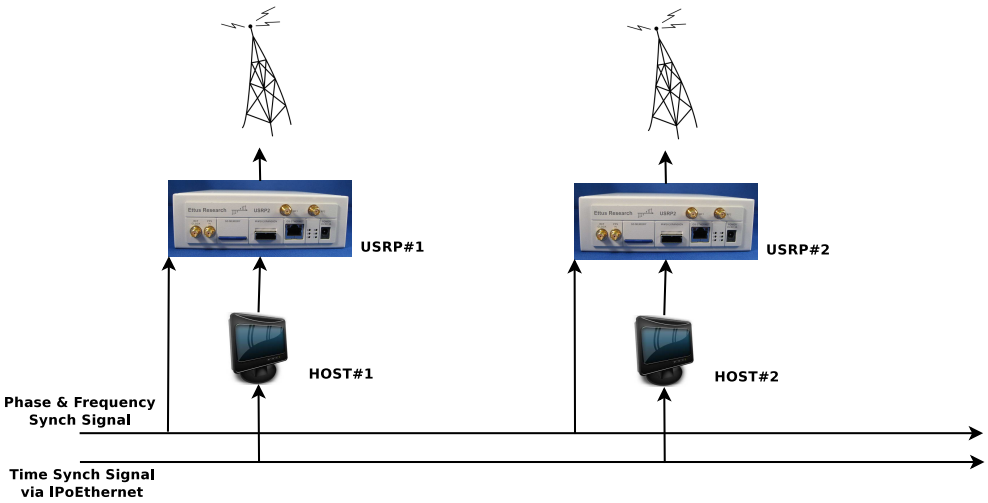

Figure 3.7: *Soft-SFN test setup architecture*

Figure 3.8: *Soft-SFN test setup. A still from the demonstration video, available online at http://www.youtube.com/watch?v=mQ6YorV4VKE*

3.4 SR-DVB, a software defined DVB-T demodulator

SR-DVB is a proof-of-concept, fully-software receiver for OFDM, ETSI DVB-T signals which aims to demonstrate effectiveness of the MA technique not just on the transmitting side but also on the receiving chain. All baseband receiver functions are implemented through MA-accelerated C++ code that was developed from scratch at the University of Pisa, Digital Signal Processing for Communication Laboratory (DSPCoLa) within the research work described in this thesis. The receiver can be seen as the result of the concatenation of two signal processing chains: the synchronization and channel estimation/equalization chain and the channel decoding subsystem, developed in [26] and [25] respectively. Implementation quality of the synchronization and channel estimation/equalization blocks has major impact over the sensitivity and the goodness of the receiver. Blocks within channel decoding subsystem, which account for the majority of the computational cost, implement functions ranging from frame adaptation to TS extraction.

As all other the other implementations being described within this thesis, SR-DVB demodulator lives within an SDR framework called *newRADIO* [35]. Such framework is conceived in order to remove from the actual radio any level of abstraction which is not strictly necessary. The rationale behind this choice is that the complexity within an SDR is always much more computational than logical. Therefore, the switch from an HW implementation to a SW one, though obtained by means of a very basic object abstraction level, provides enough generality and flexibility to the entire SDR system. Development effort is instead needed in minimizing computational overhead as well as in finding practical ways to relieve computing cores from their huge burden in any possible way.

Each functional block of SR-DVB, except the FFT block which is based on FFTW [33], was implemented from scratch in order to have full control of available system resources and to reach, by means of MA application, real-time performance. Within SR-DVB architecture, signal captured by the antenna (its spectrum is visible in figure 3.2) is sent via a 50 Ω coaxial cable to the USRP front-end and then (after undergoing just baseband conversion and sampling) via USB 2.0 to the host PC, a system based on the Intel Q9400 CPU (2.66 GHz). SR-DVB was tested and validated by receiving the Soft-DVB 11.612 Mbps signal which it proved able to correctly demodulate in

real-time while absorbing less than 50% of computational resources available aboard the host PC. Figure 3.9 shows a screenshot of one of the received channels, whereas 3.10 shows statistics and content analysis relating to the entire demodulated MPEG Transport Stream.

Figure 3.9: *Output of the mplayer Linux media player being fed by the MPEG TS received with SR-DVB.*

3.5 Second generation DVB systems, improvements and evolutions

In 2006, just two years after the final release of the currently deployed DVB-T standard [22], work began for defining the forthcoming DVB-T2 system, which has already reached the market in several countries, including the United Kingdom, Italy and Sweden, and is planned to be fully rolled-out in the next few years. Actually, DVB-T2 provides a 30% increased payload capacity and supports High Definition TeleVision (HDTV) delivery via Motion Picture Experts Group 4 (MPEG4) codecs. Physical

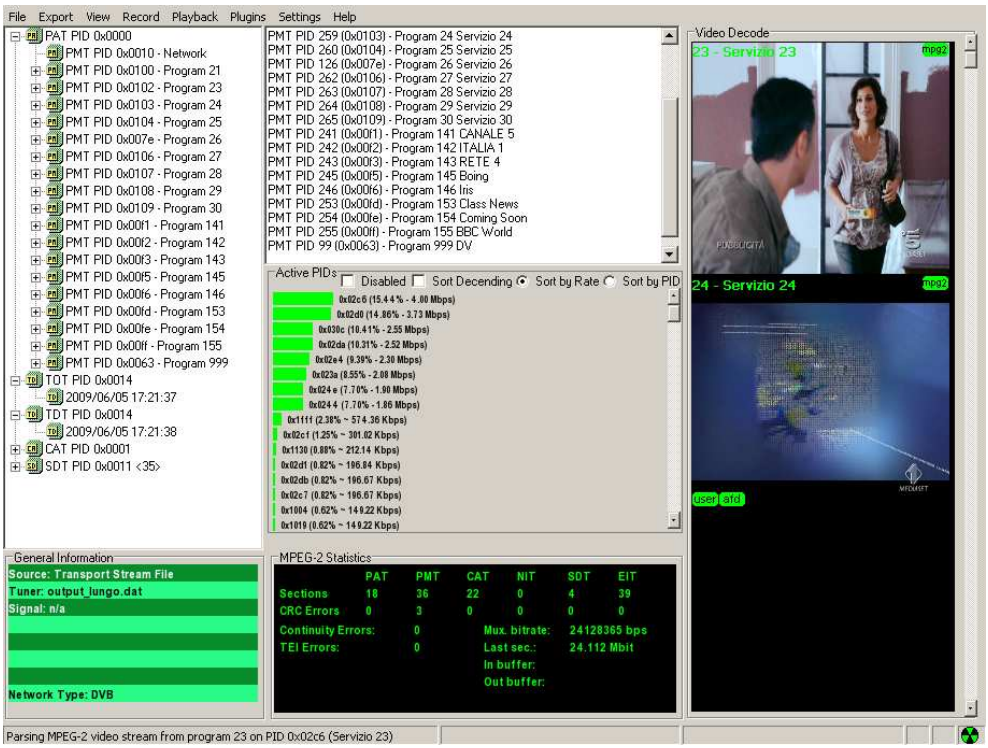

Figure 3.10: *Analysis of the demodulated MPEG Transport Stream*

layer improvements include, but are not limited to, a much stronger (and computationally heavier) FEC made up of concatenated Bose Chaudhuri Hocquenghem (BCH) and Low Density Parity Check (LDPC) codes, higher order constellations, rotated-constellations mapping strategy, improved channel estimation and equalization (figure 3.11). Here is a brief description of the most relevant blocks that distinguish DVB-T2 from DVB-T.

- *FEC codes* probably are the most relevant innovation in DVB-T2 with respect to its first generation counterpart. Here the concatenation of the Reed-Solomon and convolutional codes which was derived, for DVB-T, from the deep-space communication experience and knowledge developed in the '70s and '80s has been replaced by the concatenation of BCH and LDPC codes. As in DVB-T

the Reed Solomon code is responsible for correcting some error clusters which might occasionally appear at the output of the Viterbi decoder [1], in DVB-T2, the BCH completes the error protection provided by the main (LDPC) encoding by lowering its error floor. DVB-T2 codes perform within a fraction of a dB from the Shannon theoretical channel capacity limit, whereas DVB-T FEC stops at about 5 dB from Shannon. This huge step forward in terms of communication performance of the system comes, of course, at the expenses of the computational cost both on the tx and rx sides. The LDPC code is systematic and the encoded block length can be as much as 64800 bits.

- *Constellation maps* up to 256-QAM can be used due to the more robust error protection capabilities provided by the new FEC scheme (DVB-T stops instead at 64-QAM as the maximum constellation size option).

- *Rotated Constellations* are, most likely, the other big innovation introduced into the DVB terrestrial broadcasting system by its second generation family of standards. The rationale behind the idea, implemented in T2, of rotating the mapper's constellation and transmitting symbols on the quadrature channel with a given delay with respect to the I channel is the following. In a rotated constellation, correct recovery of even just one of the two symbol components (either I or Q) is sufficient to unambiguously determine the transmitted constellation point (this is not true for conventional constellations instead). Therefore, if the I and Q components of a rotated constellation symbol are transmitted via separate subcarriers, it might be possible, even if a given OFDM tone is lost due to channel impairments of whatever kind, to still recover all the symbols (provided that the two symbols which were affected by such loss can still rely on the other subcarriers to be received correctly). This new DVB-T2 feature has proved to be of great use when operating upon heavily impaired, frequency selective channels.

- Several *Pilot Subcarrier Patterns* were standardized that offer options for reducing the total amount of symbols used for channel.

- Possible *Guard Interval Lengths* do follow the same strategy

[1] used for convolutional codes on the rx side

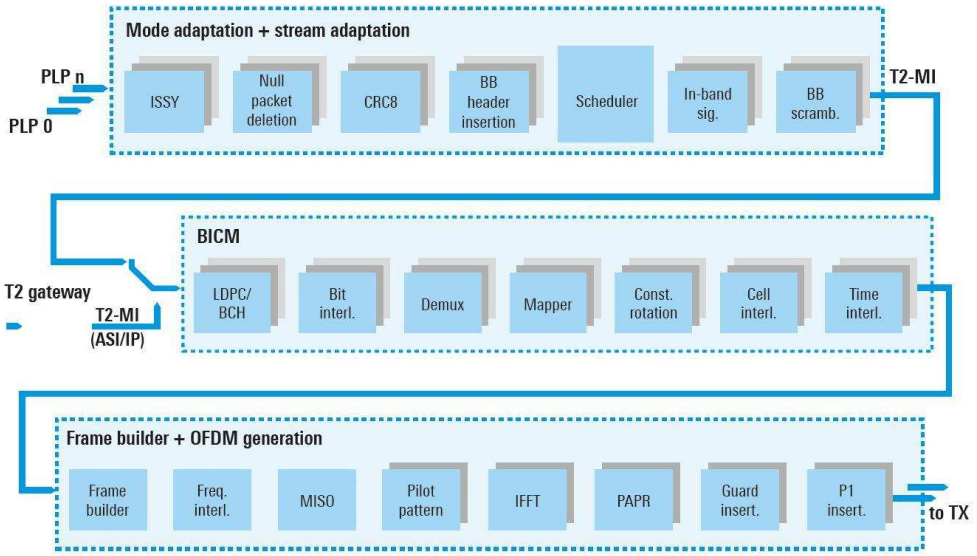

Figure 3.11: *The ETSI DVB-T2 broadcasting chain. Source: wikipedia.org*

- Further *Transport Stream Level* error protection, bandwidth saving and synchronization capabilities were added by also featuring a *brand-new* input preprocessing stage.

With the purpose of demonstrating MA effectiveness and suitability for the implementation of high-profile, high-performance (and therefore highly computation-hungry) transmission standards, a full-software, GPP-based real-time DVB-T2 modulator was developed by massively resorting to the MA acceleration technique, just some weeks after the completion of the official PhD program this thesis relates to.

3.6 Soft-DVB2, a software defined DVB-T2 modulator

Due to the state-of-the-art of commercial wireless transmission systems approaching the single-user channel capacity within a few fractions of a dB, during the first two

months following the end of the PhD program, actually by the time of finalizing this thesis, we considered it worthwhile to test our MA approach also against emerging and future-proof communication standards. It was for this reason that we set off for the implementation of the ETSI DVB-T2 standard for digital TV broadcasting [36]. After about two months of intensive work we got to some results which we describe within this section.

The DVB-T2 standard is inherently much more computationally demanding than its first generation predecessor. This is largely, but not only, due to the much more powerful BCH + LDPC FEC coding that was adopted by the ETSI for second generation DVB systems. Also, the inclusion into the modulator of several additional processing features (e.g. rotated constellations, four stages of interleaving, a whole hierarchy of pilot symbols and information) along with lots of MPEG transport stream pre-processing (Mode Adaptation / Stream Adaptation in the DVB-T2 speak) considerably increased the computational requirement (figure 3.11). Still, by extensively using MA concepts [2], it was possible to obtain a DVB-T2 modulator implementing either a 7 or an 8- MHz DVB-T2 channel with 16-QAM mapping, LDPC coding rate 3/5, 2K FFT size and OFDM guard interval 1/8, yielding a total useful bitrate of 12.567911 Mbps and capable of running in real-time while loading at about 44% an Intel E8400 3- GHz dual core CPU (figure 3.12). The output signal generated by Soft-DVB2 was validated by transmitting it towards a SHARP TU-T2 set-top-box, one of the very first DVB-T2 home receivers worldwide to be commercially distributed, (figure 3.13) which was able to flawlessly demodulate it for test cycles lasting a few hours (figure 3.14). The CPU that was used to carry out such tests is based upon an architecture being already quite obsolete at the time of writing, the computational load imposed by Soft-DVB2 upon a recent (both desktop and mobile grade) Intel core-i7 processor is therefore expected to be much lower. We believe this implementation proves that the MA SDR programming rules are also suitable when it comes to implementing the newest, broadband and *close-to-Shannon* wireless communication standards.

[2] which in this case were not applied upon an existing reference, calculus-only implementation to be used as a starting point, as it had happened with previous systems, but were directly applied starting from the the first draft of the SDR

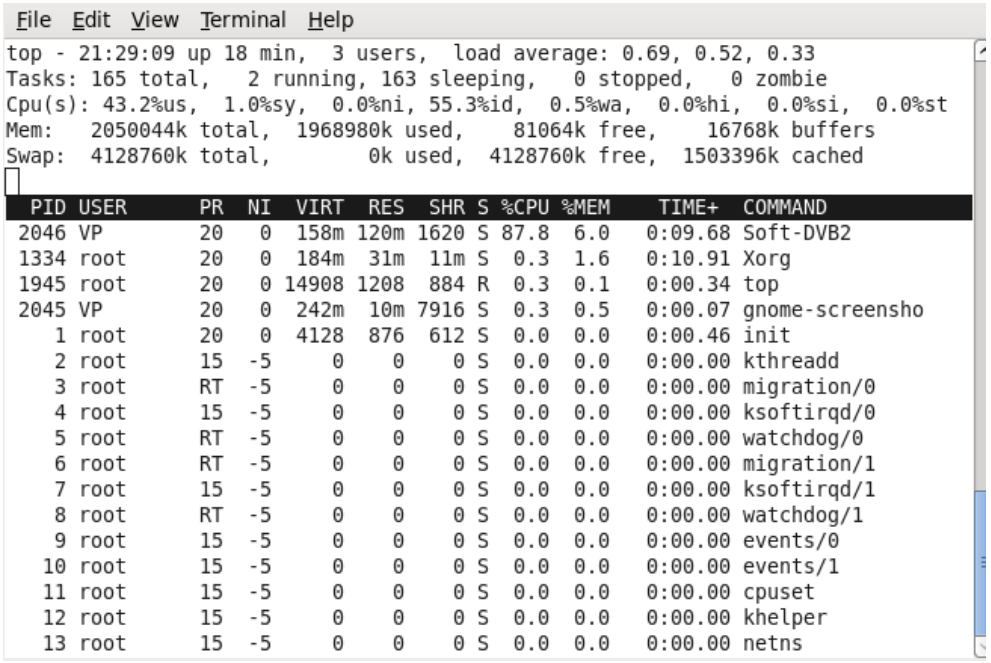

Figure 3.12: *Soft-DVB2 SDR over the Intel E8400 CPU. Performance Figures. Total CPU absorption is about 44%, memory occupancy about 118 MiB. Note that, in the representation adopted by the Table of Processes (TOP) Unix utility, full CPU occupancy for a dual core CPU as the Intel E8400 is marked as 200% usage*

Figure 3.13: *The SHARP TU-T2 set-top-box used as a test receiver*

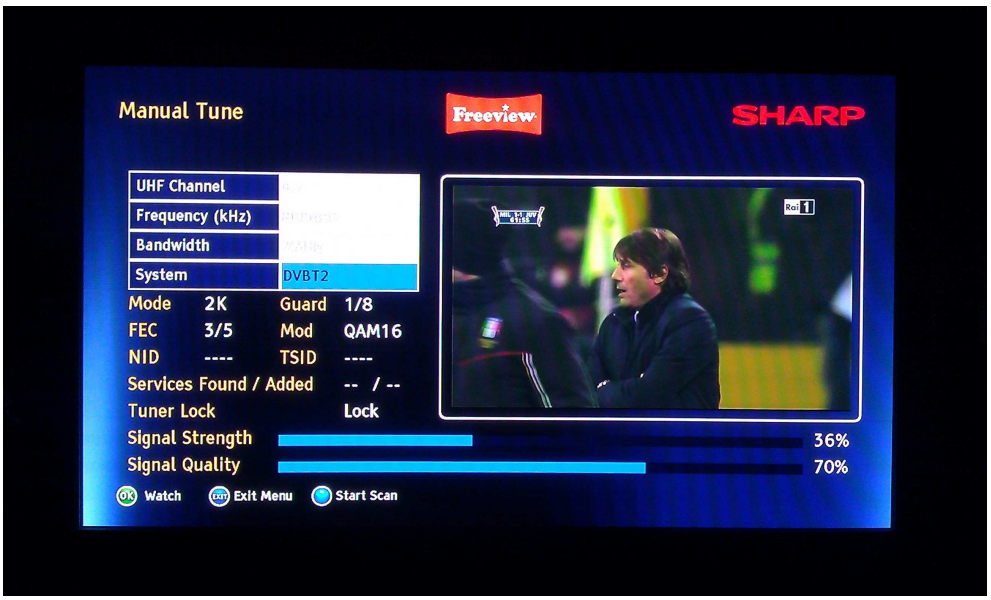

Figure 3.14: *Soft-DVB2 SDR at work. Output of the SHARP TU-T2 test receiver being fed by the Soft-DVB2 transmitted signal*

Chapter 4

MB-SDR, opportunities of a full-memory approach

Based on the performance boost obtained by applying MA approach to our radios, curiosity naturally arises about evaluating the possibility to take the MA concept to its absolute limit: *memory-implementation of the entire radio system*, after – obviously – suitable algorithm segmentation and table aggregation.

At this point it might be useful to discuss a little more in depth *what* actually distinguishes software implementations from hardware ones to the extent that average power efficiency gap between the two is as wide as two orders of magnitude in favour of HW systems (with worst cases reaching three orders of magnitude). Key concept in order to understand this huge separation is *specificity*. Software implementations do rely on general purpose processors, which means that such computing systems feature as-small-as-possible operation granularity. I.e. the simplest arithmetic operations (sums) are *serially* performed a huge number of times and combined in order to produce the required processing. This yields that a given (rather complex) operation will require a very big number of clock cycles to be performed. On the other hand, HW implementations consist of task-specific circuits made up of many synchronous logic components. Thus, in a HW implementation a large number of elementary operations will be *aggregated within a single clock cycle* (i.e. each time the system clock is incremented by one, many elementary operations take place throughout the entire system, in all of its sections), clock frequency then can be kept much lower than what is required by SW systems in order to perform, in real-time, the amount of computation required by a certain radio standard. This is actually where the power efficiency gap between the two implementation classes is generated.

In section 2.1.2 we described MA technique as a way of assisting processing-yielded computation through the usage of memory resources. Indeed, given the above analysis of the difference between HW and SW systems, we could even look at the MA approach as a way of making up for the low operation granularity that characterizes software systems and impairs them compared to their HW counterparts, when it comes to comparing power efficiencies. I.e. as HW systems do aggregate many operations within the same clock cycle by means of implementing several dedicated subsystems all triggered by each and every clock edge, *MA approach aggregates many elementary operations by storing the result of their combination into a single table.* Some clock cycles will then be used in order to access the content of the table but still, if MA was designed correctly, the amount of *equivalent* elementary operations *performed per single clock cycle* will be greatly increased. Based on such considerations, the idea of extending memory implementation to the entire radio system starts getting attractive.

Thus, we define a Memory Based SDR (MB-SDR) as *a software defined radio where most (or all) of the computation is performed by means of suitable memory look-ups.* It might appear at a first glance that such a design choice conflicts with what stated in section 2.1.1, where we claimed that efficient SDR implementations must use up all resources available (calculus and memory) in order to perform their computation. Actually, by implementing an MB-SDR, calculus resources are not left unused, they are used (exclusively or almost exclusively) to cover the computational work $[W_m]$ yielded by memory management. This obviously loosens performance requirement over computing resources and therefore *mandates downsizing of the computing core* in order to have it fully loaded and obtain the required increase in power efficiency. As a result, the role of the computing core of an MB-SDR is just to *move the data around* and *fetch the necessary information* from the suitable memory tables while actual computation truly happens only *in memory.* Considering that, with present technology, an average computer CPU can take about 140 Watts of electrical power while 2 GiB, Double Data Rate 2 (DDR2) RAM modules typically require 4.4 to 5 Watts, this appears to be an interesting strategy to increase power efficiency of SDRs.

It is important to note how such a power efficiency gain has absolutely no impact upon the flexibility of the system. The flexibility and ease of reconfiguration which are peculiar to any SDR are fully conserved by the MA approach as long as resources used for speeding up the computation are provided by memory and not by a different

(less flexible) computation technology, as it always happens whenever performance boost is gained by replacing SW implementations with specialized HWs.

As long as we have defined above an MB-SDR as accommodating in memory *all*, but even just *most*, of the computation yielded by the radio communication system, hierarchical, computational-cost-driven memory-mapping of functional blocks remains a necessity. RTAR is thus kept as the instrument of such prioritization while algorithm segmentation still provides input space cardinality reduction as described for the general case in subsection 2.1.5.

Chapter 5

Conclusions and perspectives

Within this thesis, we introduced the notion of memory-acceleration for the functions of an SDR terminal, and we showed that such technique can bring considerable speedup factors in a fully-SW, GPP-based implementation. We also illustrated how to regard MA as a technique to enhance the energy efficiency of the SDR, in that it replaces many power-consuming CPU operations with a single memory access. With proper segmentation and *in-memory* re-aggregation of signal processing algorithms, MA yields a typical performance boost of *one order of magnitude*, making an SDR energy- and/or computation-efficient without any impact on flexibility/reconfigurability. The acceleration factors presented in this work were obtained with computing architectures and compilers (GNU g++) that are totally unaware of the MA approach and therefore tend to disfavor memory access to privilege pure computation. It is expected that applying MA on computational back-ends that take into account memory management and access optimization results in larger acceleration factors. We believe that the results that were obtained within this research work do prove the possibility to implement state-of-the-art, "close-to-Shannon" radio transmission systems by resorting only to *general-purpose silicon*, or, in other words, by means of pure software over GPP processors. Although hardware implementations still retain a significant advantage in terms of throughput-per-Watt, the application of the MA programming technique shrinks the performance gap substantially. As factually proven by the developed systems, several radio communications applications are already within the reach of memory-accelerated full-software SDRs.

In fact, all modern radio communication applications either

- being non highly-mobile, thus non extremely power-critical
- or involving only the transmission side

can be implemented today in pure software upon reasonable power budgets if the MA technique is used, even if they feature data rates in the range of several tens of Mbps. This includes, for example, vehicular communication systems, receivers and transmitters operating upon typical set-top-box power budgets, fixed (base-station-like) applications, point-to-point links, broadcast applications etc.

Usual, SDR-based signal integrity monitoring applications as well can take a lot of advantage from the application of MA.

5.1 Development perspectives

Although RTAR is expressed and operates in a sufficiently algorithmic way, due to Algorithm Segmentation still requiring a considerable amount of human contribution to be suitably designed, the memory-acceleration of an SDR remains a task involving human effort. An extremely relevant possibility would be instead to achieve automatic implementation of both RTAR (already pretty straightforward) *and* algorithm segmentation (much more complicated but surely viable, at least if segmentation optimality is not strictly required). This development path would lead to the implementation of what we might call an *MA-compiler*: a piece of software accepting as an input a standard implementation of a given functional block of a certain SDR and returning the corresponding memory-accelerated implementation of the same block as its output. Such a system would be extremely useful for SDR research and development as well as a quickly market-deployable tool.

5.2 Research perspectives

Considering the large variety of the processing algorithms typically encountered in radio terminals, we could not give a general criterion to perform segmentation. Finding the optimal segmentation of classical radio signal processing algorithms that leads to the most efficient memory acceleration might be a promising research path. Knowing

optimal segmentation of the most widely used radio signal processing functional blocks, besides being an interesting memory/calculus optimization problem, would make it possible to build an extremely powerful and useful *tool-box* of memory-accelerated algorithms to be assembled together when building an SDR for a given radio application or radio-communication standard.

Bibliography

[1] J. McHale. SDR: here, there, and everywhere.

[2] M.S. Safadi and D.L. Ndzi. Digital hardware choices for software radio (sdr) baseband implementation. In *Proc. ICTTA '06 Information and Communication Technologies*, Damascus, Syria, October 2006.

[3] M. Dickens, B. Dunn, and L.J Nicholas. Design and implementation of a portable software radio. *IEEE Communications Magazine*, 46:58 – 66, August 2008.

[4] Joseph Mitola. The software radio. In *Proc. IEEE National Telesystems Conference, 1992*, pages 15–23, Washington, DC, USA, May 1992.

[5] M. Cummings and S. Haruyama. Fpga in the software radio. *IEEE Communications Magazine*, 37:108 – 112, February 1999.

[6] GNURadio website.

[7] OSSIE website.

[8] Ettus Research LLC website.

[9] Mario Di Dio, Andrea Morelli, and Marco Luise. A real-time, fully-software aeronautical vhf tx/rx waveform based on the sca-compliant ossie/usrp2 platform. In *Proc. WSR12 Int. Workshop on Software Radios*, pages 35–41, Karlsruhe, Gemany, march 2012.

[10] C. Fernndez-Prades, J. Arribas, P. Closas, C. Avils, and L. Esteve. Gnss-sdr: an open source tool for researchers and developers. In *Proc. ION GNSS Conference 2011*, Portland, OR, USA, September 2011.

[11] P. Hndel and P. Zetterberg. Receiver i/q imbalance: tone test, sensitivity analysis, and the universal software radio peripheral. *IEEE Transactions on instrumentation and measurement,*, 59.

[12] P. Zetterberg. Experimental investigation of tdd reciprocity based zero-forcing transmit precoding. *EURASIP Journal on Advances in Signal Processing, special issue on Cooperative MIMO Multicell Networks*, 2011.

[13] T. O'Shea, T. Clancy, and H. Ebeid. Practical signal detection and classification in gnuradio. In *Proc. SDR Forum Technical Conference*, Denver, CO, USA, November 2007.

[14] B. Le, T. W. Rondeau, D. Maldonado, D. Scaperoth, and C. W. Bostian. Signal recognition for cognitive radios. In *Proc. SDR Forum Technical Conference*, Orlando, FL, USA, November 2006.

[15] Jens P. Elsner, Christian Koerner, and Friedrich K. Jondral. Non-parametric modeling with gaussian processes for spatial radio-scene analysis. In *Proc. 5th IEEE Consumer Communications and Networking Conference (CCNC)*, Las Vegas, NV, USA, January 2008.

[16] OpenBTS website.

[17] HPSDR website.

[18] K. Tan, H. Liu, J. Zhang, Y. Zhang, J. Fang, , and G. M. Voelker. Sora: high-performance software radio using general-purpose multi-core processors. *Commun. ACM*, 54:99–107, January 2011.

[19] D. Cociorva, G. Baumgartner, C.G. Lam, P. Sadayappan, J. Ramanujam, M. Nooijen, D.E. Bernholdt, and R. Harrison. Space-time trade-off optimization for a class of electronic structure calculations. *ACM SIGPLAN Notices*, 37:177–186, September 2002.

[20] C.D. Walter. Space/time trade-offs for higher radix modular multiplication using repeated addition. *IEEE Transactions on Computers*, 46:139 – 141, February 1997.

[21] Once upon a time-memory tradeoff.

[22] Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television, November 2004.

[23] C.R. Aguayo Gonzalez, C.B. Dietrich, and J.H. Reed. Understanding the software communications architecture. *IEEE Communications Magazine*, 47:50 – 57, September 2009.

[24] V. Pellegrini, M. Di Dio, L. Rose, and M. Luise. A real time, fully software, etsi dvb-t receiver based on the usrp. In *Proc. Int. Workshop on Software Radios*, Karlsruhe, Gemany, March 2010.

[25] L. Rose. R-dvb: Software defined radio implementation of dvb-t signal detection functions for digital terrestrial television. *Master's Thesis, University of Pisa*, April 2009.

[26] M. Di Dio. Signal synchronization and channel estimation/equalization functions for dvb-t software-defined receivers. *Master's Thesis, University of Pisa*, September 2009.

[27] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13:260 – 269, April 1967.

[28] G.D. Forney Jr. The viterbi algorithm. *Proceedings of the IEEE*, 61:268 – 278, March 1973.

[29] JJ. van de Beek, M. Sandell, and P.O. Borjesson. Ml estimation of time and frequency offset in ofdm systems. *IEEE Transactions on Signal Processing*, 45:1800 – 1805, July 1997.

[30] DVB Project. Official DVB-T deployment data. 2007.

[31] V. Pellegrini, M. Bacci, and M. Luise. Soft-dvb: A fully-software gnuradio-based etsi dvb-t modulator. In *Proc. Int. Workshop on Software Radios*, Karlsruhe, Gemany, March 2008.

[32] Gmb-n270 - intel atom mini-itx with dual vga, 6 com, and dual lan ports.

[33] Matteo Frigo and Steven G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 1381–1384, Seattle, WA, May 1998.

[34] Soft-SFN demo video-clip.

[35] V. Pellegrini and M. Luise. Fully software ofdm modulation in vehicular, highly time-variant channels. an implemented technology and its results. In *Proc. Int. Symposium on Wireless Communication Systems (ISWCS'09)*, Siena, Italy, September 2009.

[36] Digital Video Broadcasting (dvb); frame structure channel coding and modulation for a second generation digital terrestrial television broadcasting system (dvb-t2), November 2009.

List of Publications

International Journals

1. G. Baldini, I.N. Fovino, M. Luise, V. Pellegrini, E. Bagagli, S. Rubino and S. Marcoccio, "The high speed railway infrastructure in Europe and its dependency on GSM-R wireless communications," *International Journal of Critical Infrastructure Protection*, Jan. 2011.

2. V. Pellegrini, M. Di Dio, M. Luise, and L. Rose, "On Accelerating Signal Processing Functions in Software Defined Radios through Efficient use of Available Memory Resources," *submitted to IEEE Transactions on Signal Processing*,

3. V. Pellegrini, M. Di Dio and M. Luise, "On the Usage of Memory Resources for Achieving Real-Time in Wideband Embedded SDRs," *submitted to EURASIP Journal on Embedded Systems*,

International Conferences

4. V. Pellegrini, G. Bacci, and M. Luise, "SOFT-DVB: A fully-software GNURadio-based ETSI DVB-T modulator," in *Proc. Karlsruhe Workshop on Software Radios (WSR)*, Karlsruhe, Germany, Mar. 2008.

5. V. Pellegrini and M. Luise, "Fully Software OFDM modulation in vehicular, highly time-variant channels. An implemented technology and its results," in *ISWCS09, International Symposium on Wireless Communication Systems*, Siena, Italy, Sept. 2009.

6. V. Pellegrini, M. Di Dio, L. Rose, and M. Luise, "A real time, fully software, ETSI DVB-T receiver based on the USRPA real time, fully software, ETSI

DVB-T receiver based on the USRP," in *Proc. Karlsruhe Workshop on Software Radios (WSR)*, Karlsruhe, Germany, Mar. 2010.

7. V. Pellegrini, M. Di Dio, L. Rose, and M. Luise, "On the Computation/Memory Trade-Off in Software Defined Radios," in *Proc. IEEE GLOBECOM 2010*, Miami, USA Dec. 2010.

8. V. Pellegrini, M. Di Dio, L. Rose, and M. Luise, "Computational Potential of the MA Technique," in *Demo session in Proc. IEEE GLOBECOM 2010*, Miami, USA Dec. 2010.

9. R. Malangone, E. Marzilli, C. Malta, F. Senesi, M. Luise, V. Pellegrini, E. Bagagli and S. Rubino "The GRIDES (GSM-R Integrity Detection System) for the Italian ERTMS high speed line," in *WCRR World Congress on Railway Research*, Lille, France May 2011.

10. V. Pellegrini and M. Luise, "Developing Fully-Software Radios Based on the USRP," in *Tutorial accepted at IEEE ICASSP 2011*, Prague, Czech Republic May 2011.

National Conferences

11. V. Pellegrini, M. Di Dio, L. Rose, and M. Luise, "On the Computation/Memory Trade-Off in Software Defined Radios," in *Riunione Annuale GTTI*, Brescia, Italy, June 2010