A PRACTICAL VIEW ON SDR BASEBAND PROCESSING PORTABILITY

T. Kempf, E. M. Witte, V. Ramakrishnan, G. Ascheid
Institute for Integrated Signal Processing Systems,
RWTH Aachen University, Germany
kempf@iss.rwth-aachen.de

M. Adrat, M. Antweiler
Research Establishment for Applied Science (FGAN),
Dept. FKIE/KOM, Wachtberg, Germany,
adrat@fgan.de

ABSTRACT

Software Defined Radios (SDRs) are a promising approach to efficiently use the resources of a wireless communication platform for multi-standard systems. Contrary to this advantage, the SDR concept imposes new and huge design challenges for software and hardware designers. One key issue is the trade-off between energy efficiency, architecture efficiency, flexibility as well as portability. In this paper key aspects of the SDR design space are investigated: The selection of processing elements (e.g. GPPs or DSPs) and the selection of software implementation strategies (e.g. C or Assembly). Measurements and analysis support these trade-off discussions along with the identification of key issues for portable and efficient SDR implementations.

1. INTRODUCTION

The concept of Software Defined Radio addresses key issues for future multi-standard systems, among them the realization of multiple standards on a single platform and the ease of porting a given waveform to multiple SDR systems. Since portability of a waveform can not be defined by a boolean term, the porting effort determines the granularity of portability. Key requirement for future SDRs is to minimize this effort by utilizing e.g. portable C-code implementations. However, incorporation of such flexibility implies overhead costs, e.g. in processing time and energy consumption. Additionally, the performance requirements of wireless communication, in terms of latencies and throughput, put a particular pressure on the implementation of physical layer processing. Due to these conflicting demands, SDR designers have to take various trade-off decisions.
Key aspects of implementation are portability and efficiency, which need to be quantified for trade-off decisions. Possible definitions for efficiency are well known, such as area and energy efficiency. In [1] a definition for portability has been proposed in compliance to the IEEE dictionary [2]. Portability is determined by the porting effort, which needs to be invested to implement a given waveform on a certain SDR hardware platform. Such effort can be specified by the person months spent for implementation. Three main aspects of efficiency are the algorithmic performance, area efficiency and energy efficiency. Focus of this paper is the trade-off decisions between portability and energy efficiency. Figure 1 depicts this trade-off in a qualitative fashion, which will be discussed in detail within Section II.
Implementing a waveform on an SDR platform requires the following steps: First, the waveform has to be partitioned into a task graph, which then can be mapped onto the SDR hardware platform. Recently, an Electronic System Level (ESL) based design concept and workbench for such a seamless SDR design flow starting from a Waveform Description Language (WDL) [3][4] down to the SW/HW implementation has been proposed in [5]. The workbench allows a seamless design flow from specification down to the final prototype implementation along with an iterative exploration of the design space. However, soft- and hardware designers have to incorporate their expert knowledge to find best possible solutions.
In this paper an in-depth analysis of different key aspects influencing portability and efficiency will be performed. The results and investigations can support trade-off design decisions to increase portability and efficiency. Since one main objective of SDRs is the execution of multiple waveforms on one single platform, the analysis does not focus on a particular waveform implementation. It is rather based on generic mathematical kernels, such as a Fast Fourier Transformation (FFT) and FIR filter.
After the subsequent discussion of portability aspects the case study will be introduced. Finally, optimization effects and suggestions to increase portability along with efficiency will be deduced from the determined effects.

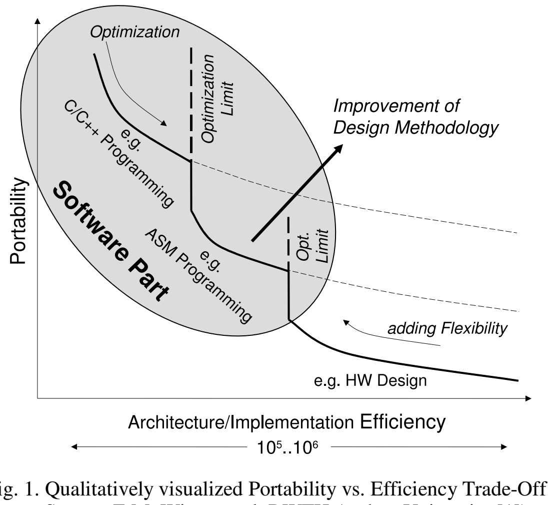

Fig. 1. Qualitatively visualized Portability vs. Efficiency Trade-Off
Source: E.M. Witte, et. al. RWTH Aachen University [1]

This research project was performed under contract with the Technical Center for Information Technology and Electronics (WTD-81), Germany.

2. PORTABILITY ASPECTS

In the domain of baseband processing real-time constraints put a particular pressure on the system development. Therefore, designers have to carefully investigate the complete design space, which is based on the three pillars:

- **Hardware architecture**
- **Software implementation**
- **Algorithm implementation**

One central hardware architecture element of future SDRs are processor cores, since they allow execution of different standards on one platform. While selecting a processor core, designers have to consider that an energy efficiency range of 5 to 6 orders and a performance range of 3 to 4 orders of magnitude exists between a General Purpose Processor (GPP) and physically optimized ASIC [6] (Figure 1). Focusing on programmable processors the energy efficiency can vary by approximately 2 to 4 orders of magnitude (Software Part in Figure 1). From the hardware perspective this part of the design space has been dominated by GPPs and Digital Signal Processors (DSPs). In recent past a new class of processors called *Application Specific Instruction Set Processors* (ASIPs) has been introduced to the markets. Promising developments of SDR specific processing elements have been introduced in [7], [8], [9] and [10]. Preliminary results of such architectures highlight a significant increase of energy efficiency and increased performance. Since these architectures are not yet available to customers, those are not investigated in this case study.

A second key aspect the software implementation for embedded devices is dominated by the C-programming language. Apart from this, Assembly programming can be found in the domain of wireless communication. It is used in performance critical parts mostly found on the physical layer, where most of the computational complexity occurs. The qualitative analysis of the software implementation in Figure 1 illustrates the trade-off between portability and efficiency. Along with increasing efficiency the porting effort rises, since utilization of special features of the underlying platform requires optimizations. Thereby, a design point on the curve moves down-and-right as depicted. Adding flexibility or removing optimizations results in a reduced porting effort, moving the design point to the upper left side. At certain points optimization limits are reached, where the implementation strategy has to be changed.

Algorithmic implementation decisions, the third key aspect, have to consider e.g. quantization, filter length, iteration counts and/or topology. Such modifications can have significant impact on the algorithmic and computational performance and efficiency. Two different kinds of algorithmic optimizations exist. The first ones do not modify the functionality, whereas the second ones modify the functionality of the algorithm, like reducing the algorithmic complexity by e.g. shortening the filter length. However both types of optimizations have to consider the underlying hardware and given constraints. For example, a Fast Fourier Transformation (FFT) can be implemented in radix-2 or different other implementations, which should optimally match the underlying hardware features.

The following case study investigates these main aspects.

Core	C-code	opt. C-code	Assembly
ARM720T	X	--	--
ARM926EJ-S	X	--	--
C55x	X	--	X
C64x	X	X	X

TABLE I: Investigated software implementations

3. CASE STUDY

A. Scenario

Focusing on the software part of the design space, the following case study investigates the key issues: (i) hardware architectures, in this case only processor cores, (ii) waveform implementation options, here only software implementations are considered and (iii) algorithmic implementations. Fundamental algorithmic kernels have been measured on which the trade-offs are discussed in the following:

- *Vector operations*, e.g. vector addition, product
- *Matrix operations*, e.g. transposition, multiplication
- *Filter operations*, e.g. FIR and adaptive LMS filtering
- *Correlation operations*, e.g. autocorrelation
- *FFT operations*, e.g. FFT and IFFT

All investigated algorithms are based on TI's DSP library [13]. Table I depicts the investigated implementation options. The examined software implementations are C-code, optimized C-code and Assembly code implementations. Respectively, different hardware options have been explored: the ARM720T and ARM926EJ-S from the GPP domain and TI's C55x and C64x from the DSP domain.

For comparison the measured execution cycles of each implementation option have been normalized by the processors clock periods to the overall execution time Δt_{exec}. The corresponding implementation efficiency can be computed as the inverse of the energy consumption per task execution given by:

$$\eta(\text{algo,arch,impl}) = \frac{1}{\text{energy per task execution}}$$

Since the power demands of the different processor cores vary and such cores can typically be operated at different

Core	Clock frequency	Avg. power demand
ARM720T	100 MHz	20 mW
ARM926EJ-S	238 MHz	114 mW
C55x	200 MHz	120 mW
C64x	1000 MHz	860 mW

TABLE II: Operation points of the measured processors

	C-code						opt. C-code		Assembly			
	ARM926EJ-S		C55x		C64x		C64x		C55x		C64x	
Algorithm	Δt speed-up	η gain	Δt speed-up	η gain	Δt speed-up	η gain	Δt speed-up	η gain	Δt speed-up	η gain	Δt speed-up	η gain
Vector Addition	3.12	0.55	11.03	1.84	198.15	4.61	198.15	4.61	11.15	1.86	222.92	5.18
Vector Product	3.58	0.63	12.54	2.09	82.16	1.91	152.00	3.53	12.67	2.11	253.33	5.89
Vector Max Value	2.61	0.46	7.70	1.28	51.26	1.19	—	—	24.19	4.03	297.69	6.92
Vector Max Index	2.86	0.50	4.00	0.67	60.68	1.41	66.95	1.56	12.14	2.02	187.90	4.37
Vector Sum Square	2.62	0.46	10.60	1.77	99.96	2.32	—	—	22.11	3.68	375.87	8.74
Matrix Multiplication	3.41	0.60	10.06	1.68	46.85	1.09	—	—	47.64	7.94	363.75	8.46
Matrix Transpose	3.23	0.57	7.11	1.18	39.50	0.92	72.18	1.68	22.43	3.74	177.00	4.12
Autocorrelation	3.23	0.57	26.13	4.35	104.80	2.44	305.37	7.10	60.18	10.03	692.63	16.11
FIR (generic)	3.30	0.58	13.91	2.32	43.88	1.02	—	—	21.23	3.54	287.42	6.68
Complex FIR	3.58	0.63	6.86	1.14	57.97	1.35	—	—	30.99	5.16	332.01	7.72
Adaptive LMS FIR	4.12	0.72	4.53	0.76	65.88	1.53	—	—	24.11	4.02	182.57	4.25
FFT Radix-2	3.43	0.60	3.54	0.59	41.30	0.96	—	—	31.31	5.22	621.15	14.45

TABLE III: Execution time speed-ups and efficiency gains normalized to the C-code implementation on the ARM720T processor (cycle accurate simulation-models: ARM720T and ARM926EJ-S CoWare CCM Model from Modelling Library [11], TI Code Composer Studio v3.3: C55x, C64x [12]) [1]

clock frequencies, a suitable operation point has been selected for each of the cores according to manufacture specifications based on a 130nm technology [14][15] (Table II). External memories and peripherals are necessary for all cores, thus their effects are not considered in the later discussions. The focus is set to processor cores including local memories, caches etc.

B. Measurements

Table III illustrates the measured execution times and efficiencies for each of the investigated algorithmic kernels normalized to the ARM720T execution. The table is structured accordingly to a typical design process starting at a C-code implementation[1]. Here, the three C-code implementations on the ARM926EJ-S, the C55x and the C64x are compared with the ARM720T. Some of the algorithms have been measured exemplarily with optimized C-code. The last four columns illustrate the execution time speed-up and efficiency gain for hand optimized Assembly codes executed on the C55x and C64x DSPs.

- **C-code implementations:** Executing a portable C-code implementation of the algorithmic kernels, the measurements show the expected significant execution time speed-up compared to the ARM720T. The ARM926EJ-S achieves a speed-up factor of 2.61 to 4.12, for the different kernels. The C55x achieves a range of 3.54 up to 26.13, whereas the determined speed-up of the C64x is in the range of 39.50 up to 198.15. Since all cores have a significant higher power demand than the ARM720T the efficiency does not show such improvements. For the ARM926EJ-S even an efficiency degradation of 1/0.46 to 1/0.72 has been identified, whereas the efficiency gain achieved on the DSPs is in the range of 0.59 to 4.61.
- **optimized C-code implementations:** Adding optimizations in terms of compiler directives exemplary to the C-code of the C64x, the compiler of the complex 8-slot VLIW architecture is able to generate code with an improved quality. For example, providing information on pointer alignments and ambiguity are well known techniques to allow more aggressive optimizations. The resulting speed-up compared to the un-optimized implementation has been determined by a factor of approximately 2. It should be noted that this does not apply for all kernels, whereas for the investigated ones a gain of up to 2.9 has been measured. The additional semantic information passed to the compiler is basically compiler independent. Since no standard exists for such directives most compilers implement those differently. Thus they can be considered as compiler dependent.
- **Assembly implementations:** The execution time speed-up and efficiency gain by applying processor specific Assembly code is significant. A speed-up factor in the range of 11.15 to 60.18 has been determined for the C55x as well as of 177.00 to 311.90 for the C64x compared to the respective C-code implementation. The determined efficiency gain has been measured to 1.86 to 10.03 for the C55x and of 4.12 to 16.11 respectively for the C64x. Compared to the C-code implementation the hand-optimized Assembly achieves an efficiency and execution time gain of a factor of 1.0 to 8.8 for the C55x and 1.1 to 15.0 for the C64x.

The following key effects observed by the measured algorithmic kernels are the following: (i) Porting the C-code implementation from the ARM720T to another processor core, whether the ARM926EJ-S GPP or the DSPs, achieves the expected execution time Δt_{exec} speed-up. For the ARM926EJ-S similar results can be found in literature [16]. (ii) For C-code on the DSPs the efficiency η gain is rather low. For the ARM926EJ-S even a degradation has been observed due to an approximately 5 times higher power demand. (iii) Application of compiler directives to the C-code implementation supports the compiler to generate more efficient code. In worst case, no performance gain has been achieved, whereas in best case a factor of 2.91 has been determined. (iv) The typical assumption that the gain of Assembly programming can be neglected can not be supported.

[1] A GNU ARM GCC compiler v4.0.2 and the compiler delivered within TIs Code Composer Studio v3.3 for the DSPs have been utilized.

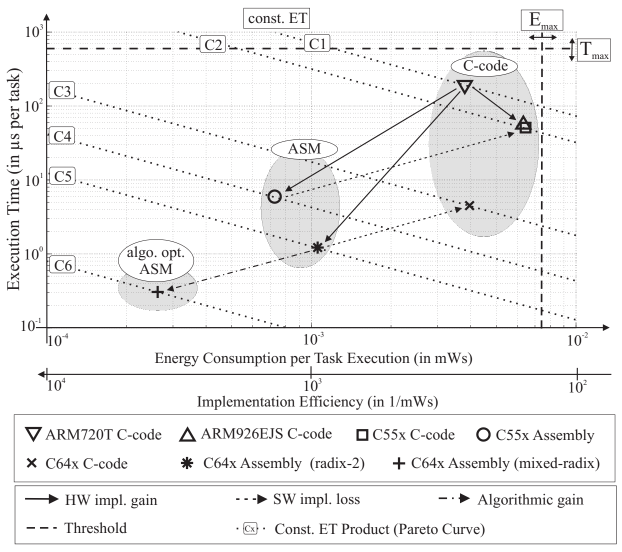

Fig. 2. FFT results (64 points): execution time versus efficiency

The achievable performance gain with architecture specific Assembly programming has been determined to be in the range of one order of magnitude.

The previous discussion has examined the measurements more on a one-dimensional basis, either the execution time Δt_{exec} or the implementation efficiency η. Since those are dependent on each other and the differences between the implementation options, e.g. C-code versus Assembly code, are significant, the tradeoffs shall be analyzed more in detail. This analysis will be performed in the subsequent section on one exemplary kernel, namely the Fast Fourier Transformation (FFT).

C. Trade-off Discussion

The following discussion is based on the measured algorithmic FFT kernel (64-point FFT as radix-2 and mixed-radix implementation with 16 bit precision). Figure 2 depicts the determined execution times Δt_{exec} plotted over the energy consumption per task execution E. Since the implementation efficiency is the inverse of the energy consumption the x-axis can be replaced by the implementation efficiency η.

In hardware design the Area-Timing-Energy Product (ATE Product) is typically taken as cost function for optimizations. Since the focus of this discussion is on energy efficiency and execution time the simplified *Energy-Time Product* (ET Product) will be considered here:

$$ET = \frac{\Delta t}{\eta}$$

A constant ET Product defines a pareto curve [17]. In decimal scale such curves are defined by a hyperbola, which results in double-logarithm scale in a line with slope −1. Along such lines trade-off decisions can be made by trading one cost parameter against the other. Contrary to these trade-offs, optimizations move a design point orthogonal to such a line in the direction of the origin. In Figure 2 the pareto curves C1 to C6 are sketched. For baseband processing tight constraints for latency and energy consumption exist. Such constraints can be illustrated as vertical and horizontal lines within the figure. A vertical line illustrates a maximum allowed latency T_{max}, whereas a horizontal line determines the maximum allowed energy consumption E_{max}. Please note that in wireless communication such constraints are much stricter then the highlighted ones in Figure 2. Thus developers have to apply architecture specific optimizations to achieve those constraints and are forced to make trade-offs.

Three identified key observations will be introduced during the further discussion, which are:

- **HW optimization gain:** The gain which can be achieved by changing from on processor core to another one including necessary individual optimizations to optimally utilize the underlying hardware.
- **SW implementation loss:** The loss which depends on not utilizing the provided features of the underlying processor core, e.g. utilizing C-code where the compiler is unable to generate optimal or near optimal machine code.
- **Algorithmic implementation gain:** The gain which results by changing the implementation characteristic in an algorithmic manner, e.g. a radix-2 versus a mixed-radix FFT implementation.

The discussion starts with the investigation of the measured results depicted in Figure 2 for the C-code executions of the FFT kernel (C-code region). Since orthogonal distances of projected pareto curves highlight optimizations of the ET Product, the measure G(Cx, Cy) defines the length of the orthogonal distance of pareto curve Cx and Cy. Following conclusions of this measure can be derived:

$$\begin{aligned} G(Cx,Cy) = \frac{Cy}{Cx} &> 1 \text{ implementation gain} \\ &= 0 \text{ trade-off} \\ &< 0 \text{ implementation loss} \end{aligned}$$

In comparison with the C55x and ARM926EJ-S to the ARM720T a minor speed-up as well as a minor optimization gain of *G(C2,C1)=2.05* of the ET-Product has been determined. Considering the C-code executed on the C64x a major execution time decrease has been measured, whereas the consumed energy remains constant. The resulting ET-Product gain is *G(C3,C1)=39.67*.

Crossing the C-code optimization limit by writing hand optimized Assembly DSP-code has a huge impact on energy consumption, respectively efficiency, and execution time (ASM code region). The pareto curves C4 of the C55x as well as C5 of the C64x highlight this significant optimization with respect to the curves C1 to C3. Regarding the previous definition of the **HW implementation gain** with respect to the ARM720T, the ARM926EJ-S can achieve a minor optimization gain (*G(C2,C1)*). For the DSPs the HW implementation gain is approximately one order of magnitude for execution time and consumed energy. A resulting large ET Product decrease is determined by *G(C4,C1)=163.43* (C55x) as well as the difference between *G(C5,C1)= 553.51* (C64x).

The **SW implementation loss** can be characterized by the difference between the C-code and the Assembly code implementations of the DSPs. The observed SW implementation loss for the C55x is determined by *G(C2,C4)=0.01*, whereas for the C64x it is *G(C3, C5)=0.07*. This huge SW implementation loss is caused by well known issues for DSP software development [18][19]. Examples for such issues are missing language support for semantic information and missing memory architecture support. Developments in the recent past have addressed these issues by language extensions, like DSP-C [20] and Embedded C [21], which are not yet supported by many compilers.

The **Algorithmic implementation gain** gets particularly visible on the C64x 8-slot VLIW processor. The radix-2 implementation is not suited to fill all slots of the core. By modifying the implementation on the algorithmic level from a radix-2 to a mixed-radix implementation a *4* times speed-up along with a significant optimization of the ET Product of *G(C6,C5)=16.21* can be achieved. This huge impact of a proper algorithmic implementation for the respective processor highlights the demand for an algorithmic reconsideration at implementation time. Therefore, designers can not exclusively focus on optimizing the selected implementation. Instead developers have to consider at the implementation step which algorithmic implementation fits best to the underlying hardware. Unfortunately this reimplementation can have significant porting effort.

Till this point only trade-offs and optimizations regarding execution time and efficiency have been examined. The **porting effort**, respectively portability, defines a third dimension. Considering portability the usage of C-code requires merely a re-compilation. Thus the porting effort for these implementations is rather low. Considering the SW implementation loss designers have to individually evaluate whether to apply cost and time intensive optimization or not. Unfortunately, this effort can not be measured that simply in numbers since it depends on several factors. For example, an experienced developer has to invest less effort than a trainee in code development. To give at least an impression of how much time has to be invested into such optimizations in [22] the development of an optimized FIR filter Assembly code on the ARM Cortex-R4 processor core has been evaluated. This development has taken approximately 20 hours. Thus designers have to consider at each design step whether to invest such cost and time intensive optimizations or not.

Summarizing the observations developers are forced to carefully inspect their taken design decisions. For the software part both two extremes, portable C-code and hardware dependent Assembly code, have pros and cons. C-code provides high portability. Unfortunately, the measured *SW implementation loss* has been measured to be one order of magnitude for execution time and energy efficiency. Including the algorithmic optimizations in total one order of magnitude in efficiency and two orders in performance have been observed for the FFT kernel. On the one hand this achievable gain is huge and can not be neglected. On the other hand the high time and cost investments for optimizations forces developers to decide carefully.

In the subsequent section hints to optimize efficiency and portability will be given.

4. DISCUSSION OF IMPLEMENTATION ASPECTS

Within the case study some points in the SDR design space have been investigated, which are determined by the selection of the underlying hardware architecture, a suitable software implementation scheme and the algorithmic implementation. Key issue for SDR developers is to achieve the given constraints of a waveform along with the contradicting requirements of portability and efficiency. Figure 3 depicts the discussed trade-off decisions by projecting the effects of the following optimizations, starting at C-programming level. For the sake of completeness the improvements by **Coding Style** are sketched. This comprises typical optimizations such as declaration of constant values, usage of inline functions etc., which do not affect the portability.

1) Adding **Compiler Directives** in form of pragmas, provides the compiler with additional information, e.g. about memory disambiguity, memory banks, etc. The semantic information, passed by such directives, is basically compiler-independent. However, no standard exists thus most compilers implement those differently. Therefore, effort has to be invested while porting the code to another platform. Due to this, the projected gain is a falling straight line in Figure 3.
2) Moving from processor independent optimization to architecture dependent optimization, first **Intrinsics and Inline Assembly** will be applied. This way of utilizing specialized assembly instructions within the C-code, allows limited efficiency gain since interfacing to C-code adds complexity and worsens compiler optimizations. The projected effect of such optimization increases the efficiency at the cost of portability.
3) **Assembly** reimplementation of an algorithm can consider architecture specific features and will generate the most efficient code at a high porting effort. Therefore, the projected effect is illustrated as shown in the curves of Figure 1 and 3 as an optimization limit. Please note that Assembly programming typically is an iterative refinement, starting at C-code [23].

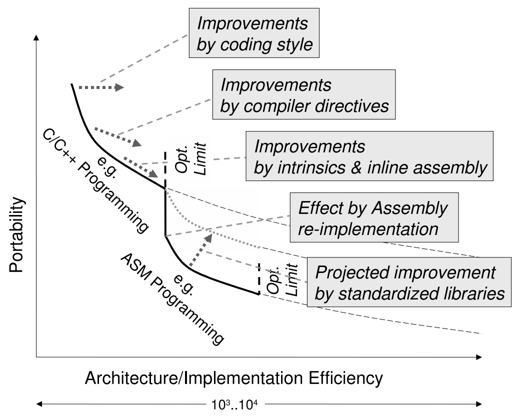
Improvements by coding style
Improvements by compiler directives
Improvements by intrinsics & inline assembly
Effect by Assembly re-implementation
Projected improvement by standardized libraries
Opt. Limit
Opt. Limit
e.g. C/C++ Programming
e.g. ASM Programming
Portability
Architecture/Implementation Efficiency
10³..10⁴

Fig. 3. Projected effects of optimization levels on the SW part

4) A suitable trade-off between Assembly programming and generic C programming is the utilization of **standardized libraries**. Such libraries are available for some processors. However, those are not standardized and hardly suited for heterogeneous SDR platforms.

From the investigated optimizations only libraries provide sufficient portability along with implementation efficiency. Additionally the high porting effort will force developers to switch to more suited development strategies. Broad acceptance and application of such libraries can only be achieved, if those libraries are standardized in functionality and Application Programming Interfaces (APIs). With regard to future standards such libraries will gain more importance. Therefore, the standardization of libraries or even a Waveform Description Language for future SDR development would give great benefit. Unfortunately, this vision can not be achieved in short term; however, we at least believe that the impact on SDR software development would be huge. Additionally, this would significantly increase the potential and acceptance of SDRs.

5. CONCLUSION AND OUTLOOK

SDR development comprises many huge challenges for developers. Two contradicting key issues for SDR design are the portability and efficiency. In this paper a case study has been presented, which highlights this issue. Three effects have been covered: *HW implementation gain*, *SW implementation loss* and *Algorithmic implementation gain*. Since the SW implementation loss can be significant, developers have to consider potential options to increase efficiency along with portability. The identified option is the use of standardized libraries. Such standardized libraries can be in future the foundation for Waveform Description Languages (WDLs). Unfortunately, none of the existing WDL approaches is yet mature enough to be used in the field today. In our future work we will concentrate on this research area.

6. ACKNOWLEDGMENT

The authors would like to thank J. Holzer, C. Hatzig, S. Hartmann and H. Siegmar of the Technical Center for Information Technology and Electronics (WTD-81) for inspiring discussions.

7. REFERENCES

[1] E.M. Witte, et al., “SDR Baseband Processing Portability: A Case Study,” in 5th Karlsruhe Workshop on Software Radios (WSR’08), (Karlsruhe, Germany), March 2008.
[2] “IEEE standard computer dictionary. A compilation of IEEE standard computer glossaries” IEEE Std 610.12-1990, Dec. 1990.
[3] E.D.Willink, “Waveform Description Language: Moving from Implementation to Specification,” vol. 1, Oct. 2001.
[4] M. Gudaitis and R. Hinman, “Practical Considerations for a Waveform Development Environment,” IEEE Military Communications Conference (MILCOM 2001), vol. 1, October 2001.
[5] T. Kempf, et al., “A Workbench for Waveform Description based SDR Implementation,” in Software Defined Radio Technical Conference (SDR’07), (Denver, USA), November 2007.
[6] H. Blume, et al., “Model-based exploration of the design space for heterogeneous systems on chip,” J. VLSI Signal Process. Syst., vol. 40, no. 1, pp. 19-34, 2005.
[7] Y. Lin, et al., “SODA: A High-Performance DSP Architecture for Software-Defined Radio,” Micro, IEEE, vol. 27, no. 1, Jan.-Feb. 2007.
[8] K. van Berkel, et al., “Vector processing as an enabler for software-defined radio in handheld devices,” EURASIP J. Appl. Signal Process., vol. 2005, no. 1, pp. 2613-2625, 2005.
[9] U. Ramacher, “Software-defined radio prospects for multistandard mobile phones,” Computer, vol. 40, no. 10, 2007.
[10] T. Vogt, N. Wehn, “A Reconfigurable Application Specific Instruction Set Processor for Convolutional and Turbo Decoding in a SDR Environment,” in Design, Automation & Test in Europe 2008 (DATE’08), (Munich, Germany), March 2008.
[11] CoWare Inc., “CoWare Modeling IP Library,” http://www.coware.com/products/modellibrary.php
[12] Texas Instruments, “Code Composer Studio™ IDE,” http://www.ti.com/.
[13] Texas Instruments, “TMS320C64x DSP Library Programmer’s Reference (Rev. B),” October 2003.
[14] ARM Ltd. http://www.arm.com/.
[15] Texas Instruments, “TMS320C6455/C6454 Power Consumption Summary,” October 2007.
[16] ARM Ltd., “Performance of the ARM9TDMI™ and ARM9E-S™ cores compared to the ARM7TDMI™core,” http://www.arm.com/pdfs/comparison-arm7-arm9-v1.pdf.
[17] P. Yang, et al., “Energy-aware runtime scheduling for embedded-multiprocessor Socs,” IEEE Design and Test of Computers, vol. 18, no. 5, pp. 46-58, 2001.
[18] K. Kennedy and J. R. Allen, Optimizing compilers for modern architectures: a dependence-based approach. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.
[19] S. S. Muchnick, Advanced Compiler Design & Implementation. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997.
[20] DSP-C Website, http://www.dsp-c.org/
[21] ISO/IEC Working Group JTC1/SC22/WG14, “Programming languages C Extensions to support embedded processors,” http://www.open-std.org/
[22] BDTI Inc., “Evaluating the DSP Capabilities of the Cortex-R4,” InsideDSP.
[23] Alan Anderson, “Programming and optimizing C code, part 1,” DSP DesignLine, http://www.dspdesignline.com/197006981.