A Framework for the Design Space Exploration of Software-Defined Radio Applications

Thorsten Jungeblut[1], Ralf Dreesen[3], Mario Porrmann[1], Michael Thies[3], Ulrich Rückert[2], and Uwe Kastens[3]

[1] Heinz Nixdorf Institute, University of Paderborn, Germany
[2] Cognitive Interaction Technology - Center of Excellence, Bielefeld University, Germany
[3] University of Paderborn, Germany

Abstract. This paper describes a framework for the design space exploration of resource-efficient software-defined radio architectures. This design space exploration is based on a dual design flow, using a central processor specification as reference for the hardware development and the automatic generation of a C-compiler based tool chain. Using our modular rapid prototyping environment RAPTOR and the RF-frontend DB-SDR[1], functional verification of SDR applications can be performed. An 802.11b transmitter SDR implementation is mapped on our CoreVA VLIW architecture and evaluated in terms of execution time and energy consumption. By introducing application specific instruction set extensions and a dedicated hardware accelerator, execution time and energy consumption could be reduced by about 90 %.

Keywords: Design Space Exploration, Software-Defined Radio, Architecture, VLIW, CoreVA.

1 Introduction

Wireless communication more and more finds its way into our daily life. Mobile devices support complex applications, like high definition television(HDTV), video conferences, or online gaming. Simultaneously, new mobile transmission protocols allow higher throughput and lower latencies. Furthermore, the computational demand of these protocols increases steadily. Instead of using multiple dedicated hardware components for each algorithm, the trend is towards single, reconfigurable, general purpose processing units. The higher integration rate of microelectronic devices allows CPUs to support the required processing power at reasonably low energy consumption. In the beginning of the 1990's, Joe Mitola coined the term *Software Radio Architecture* or *Software-Defined Radio*. In [8] he defines software radio as a set of Digital Signal Processing (DSP) primitives, a metalevel system for combining the primitives into communications systems functions (transmitter, channel model, receiver, ...) and a set of target processors on which the software radio is hosted for real-time communications. Software

[1] DB: daughter board, SDR: software-defined radio.

radio adds the flexibility to the communication systems to allow the dynamic reconfiguration of the used standards. In case of changes in the standards, updates can be performed by simple software updates. Modifications of hardware components are not necessary. High level programming languages like C/C++ make development of the applications hardware independent and more maintainable. Fine grained reconfigurable devices like *Field Programmable Gate Arrays (FPGA)* add additional computational power for highly parallelizable signal processing components. In addition, the general purpose CPU is not only suitable to execute radio-based algorithms, but also the operation system or multimedia applications. The complexity of the radio architecture is highly reduced.

This work presents a framework for the development and design space exploration of software-defined radio applications. Both software application and hardware platform are taken into account. In Section 3 we present a holistic tool flow for the evaluation of the resource efficiency of the hardware-/software combination. The tool flow is based on a central processor specification described in the UPSLA language[6], which acts as a reference for the development of the hardware and the software development tools. In Section 3 we introduce the CoreVA VLIW architecture which plays the role of the central general purpose CPU to execute the radio protocols. In Section 4 we present our rapid prototyping environment RAPTOR, a modular architecture for the evaluation and verification of the radio application. The flexible software-defined radio extension module *DB-SDR* is used for the realization of the RF-parts of the software radio architecture. In Sections 5 and 6 we discuss the potential of our tool flow by mapping a IEEE 802.11b transmitter SDR implementation to our CoreVA architecture and present the results of the optimization by application specific instruction set extensions and hardware accelerators. In Section 7 we conclude our work and propose potential optimizations and future extensions.

2 Related Work

In [7] Joe Mitola describes the potential of a software radio architecture and the necessity of open architecture interface standards (e.g. Common Object Resource Broker (CORBA)). The architecture is defined as a comprehensive, consistent set of functions, components, and design rules. A channel stream is partitioned into five segments : *Antenna Segment*, *IF Processing Segment*, *Baseband Processing Segment*, *Bitstream Segment*, and *Source Segment*. The author presents an estimation of resource requirements. The highest computational demand (>1 billion operations per second) relates to the IF Processing Segment. He takes for example a bandwidth of 10 MHz and a oversampling sampling factor of 2.5. With a computational demand of 100 operations per sample this sums up to 2.5 billion operations per second. To handle the complexity, he suggests to assign parts of the IF Processing Segments to heterogeneous multiprocessing hardware or dedicated chips as well.

[1] presents a tunable software-defined radio receiver from 800 MHz to 5000 MHz implemented in a 90 nm technology. The receiver chip size is $7\,mm^2$

and is sufficient to implement mobile standards from GSM up to 802.11g wireless LAN (up to 40 MHz sample rate).

The authors of [2] describe a programmable baseband platform for mobile and wireless LAN standards. The architecture embeds four single-instruction multiple data (SIMD) DSP cores which are accompanied by dedicated programmable processors for channel en-/decoding and filtering operations and an ARM processor for the execution of the protocol stack. To prove the feasibility of their approach, they investigated IEEE 802.11b as the first communication standard. The architecture operates at a frequency of 300 MHz.

In [10] and [11] a software defined radio prototype based on a multiprocessor architecture is proposed. Several mobile applications as well as IEEE 802.11b wireless LAN is implemented. They use flexible-rate pre-/post-processors, four TMS320C6201 DSPs (200 MHz) and a PowerPC (400 MHz) to handle bandwidths of up to 20 MHz. User data rates of up to 2 MBit/s (IEEE 802.11b, DQPSK) are achieved.

In [4] Intel targets wireless LAN standards in the 802.11 series with its reconfigurable communications architecture embedding a mesh of processing elements and couples to analog front ends implemented in a CMOS technology.

All approaches show the immane computational demand required for baseband processing and the enlargement of the design space to be considered. Our methodology allows a fast and accurate evaluation of SDR based applications.

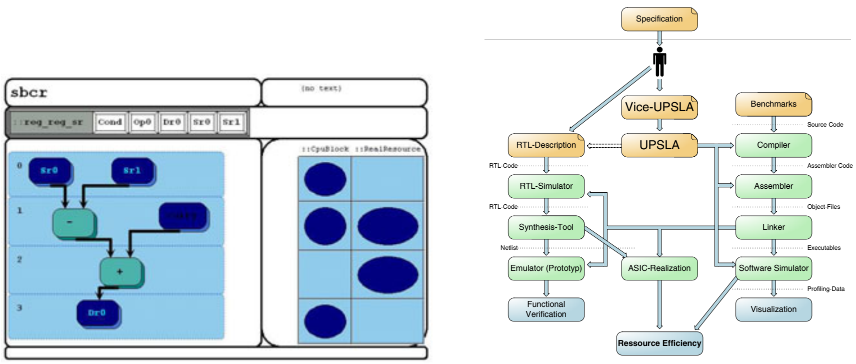
3 Processor Design Based on UPSLA

For development of processor architectures and the corresponding tool chains we use a dual design flow based on the Unified-Processor-Specification-Language (UPSLA) as a reference description (cf. Figure 1b).

UPSLA is a declarative language for the description of processor architectures. The language uses object oriented techniques, which leads to a very compact specification. For sharing of common properties inheritance can be used. Instances with the same behavior are grouped into equivalence classes. This avoids redundancy and enables consistent, global changes on the specification. By this the processor specification can easily be adapted to changes in the architecture. UPSLA is described in textual form. This can be done manually or by a graphical user interface (ViceUPSLA, cf. Figure 1a).

Based on this specification, a complete tool chain can be generated automatically. The tool chain consists of a C-compiler, an assembler, a linker, an instruction set simulator and various debugging and profiling tools.

As a result of this profiling, modifications to the processor architectures can be performed, for example by modifying the instruction set or adding additional functional units (cf. Section 6, [5]). The consistency of the generated tool chain and the hardware description is checked using a validation by simulation approach presented in [3]. Functional verification is performed using our rapid prototyping environment RAPTOR resented in Section 4.



(a) ViceUPSLA for the graphicalesrip-tion of the processor specification

(b) The central processor specifica-tion in the UPSLA languagects as a reference for the hardware devel-opment and the software tool chain

Fig. 1. Dual design flow for the development of processor architectures

The CoreVA architecture. The CoreVA architecture represents a 4-issue VLIW architecture[5]. Using the hardware description language VHDL, CoreVA is specified as a soft macro at register-transfer-level (RTL). The typical harvard architecture with separated instruction and data memory provides a six-staged pipeline (instruction fetch (FE), instruction decode (DE), register read (RE), execute (EX), memory (ME), and register write (WR).

Besides four arithmetic-logical-units (ALUs), two dedicated multiply-accumulate (MLA) units support fast multiplication. Divisions are accelerated by two dedicated division step units. The register file comprises 31 general purpose 32-bit registers, which can be accessed by all four issue slots. The byte-wise addressable memory supports 8-bit, 16-bit and 32-bit data transmissions using natural alignment and little-endian byte-ordering. The operations follow a two- and three-address format and are all executed in one clock cycle. Most instructions have a latency of one clock cycle, except branch, MLA and load operations, which have a latency of two clock cycles. In SIMD (single instruction, multiple data) mode, two 16-bit words can be processed in each functional unit (FU), which leads to an eightfold parallelism. Two 7-bit condition registers support conditional execution for scalar and SIMD operations. If not all FUs can be utilized, a stop bit in the opcode allows to omit empty trailing instruction slots. This leads to more compact code and reduces power consumption, which is very useful for embedded systems, e.g. smart cards. Still, 64% of the opcode space of the CoreVA architecture are free and can be used for instruction set extensions. The CoreVA architecture operates at a clock frequency of 300 Mhz. Area consumption is about $4\,mm^2$ including 32 kByte Level-1 cache for instruction and data. The CoreVA architecture is implemented as a configurable IP-core, for example, the number of functional units, dedicated multipliers/dividers or load/store units can be specified.

4 A Flexible RF Frontend for SDR Applications

For the functional verification of the hardware implementation, we use the RAPTOR2000/X64 rapid prototyping environment [9]. This modular system offers a large selection of FPGA daughter boards or physical interfaces (e.g. Ethernet), which enable the use in real environments.

For the functional verification of software-defined radio applications, we developed an extension module for our RAPTOR family. This module is compatible to the USRP[2] base system and extension modules from Ettus Research[3]. Hereby the extension modules from Ettus Research can directly be reused an RF frontends need not to be developed. By two connectors, these modules can be plugged to the DB-SDR RAPTOR module.

The transceiver modules available from Ettus Research enable the DB-SDR to be used as an adequate RF-transceiver. In general a bandwidth of 30 MHz can be set up. The fully synchronous implementation enables *Multiple Input Multiple Output (MIMO)*. In contrast to the RX- and TX modules, the transceivers include oscillators to enable split frequencies operations. *Frequency hopping* is enabled by dedicated PLLs[4]. Beyond this, all transceiver modules include analog RSSI[5] measuring systems to get direct access to the received field strength. *Automatic Gain Control* is used for a level normalization at 70 dB. All transceiver modules are full duplex capable and the transmit power is adjustable. The receiver modules available at Ettus Research range from 50 MHz to 5 GHz. The Ettus RFX2450 covers frequencies ranging from 2.4 to 2.5 GHz and 4.9 to 5.9 GHz at a maximum transmit power of 100 mW (20 dBm).

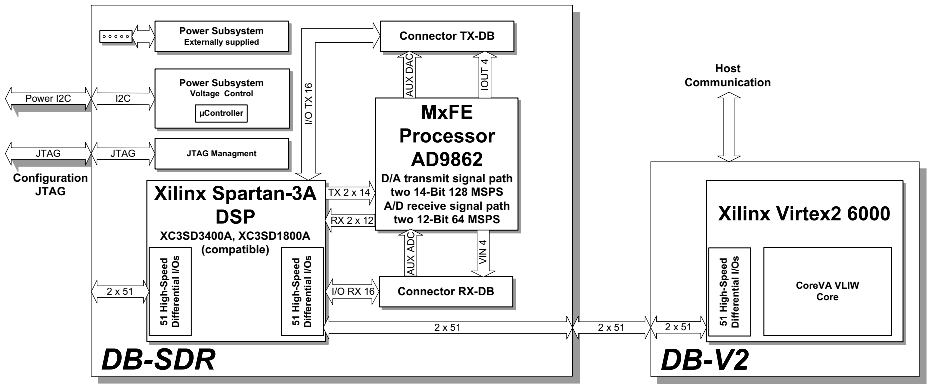


Fig. 2. Block diagram of the DB-SDR Extension module

[2] *U*niversal *S*oftware *R*adio *P*eripheral.

[3] http://www.ettus.com/

[4] PLL: Phase Locked Loop.

[5] RSSI: Received Signal Strength Indication.

AD/DA conversion. Core of the DB-SDR is the AnalogDevices AD9862. The AD9862 is a 12-/14 Bit *mixed signal front-end* (MXFE®) processor for broadband communication. The receive path consists of two 12 Bit A/D converters with a sample rate of 64 MSamples/s to sample a 32 MHz wide frequency band. The upper limit of the AD9862 is 100 MHz. The sending path consists of two high speed 14 Bit D/A converters. The clock frequency of the D/A converters is 128 MS/s, so the Nyquist frequency is 64 MHz. Additional PGAs allow an amplification of up to 20 dB. The outputs are current outputs of 0 to 20 mA. Beside this main AD/DA converters, the AD9862 embeds four *auxiliary analog inputs* with a 10 Bit *low-speed* A/D converter (1.25 MS/s, bandwidth of $\approx$ 200 KHz). This auxiliary analog inputs can be used for example to sample the RSSI signal levels or measuring of temperature or bias. An *low speed* 8 bit D/A converter can be used for example to control external amplifiers.

FPGA. The data preprocessing and the communication with the host system is performed by an Spartan 3 ADSP (XC3SD3400A or XC3SD1800A). This FPGA is also used to configure the AD9862 AD/DA converter and the extension modules by Ettus Research. An UART interface can be used for debugging.

Power management. The power can be supplied by the RAPTOR base board or externally to decouple noise from the host system. A Microship *PIC (Peripheral Interface Controller)* PIC18F6722 controls the initialization of the different voltages.

Clock distribution. Similar to the power supply, the reference clock for the AD/DA converter can be supplied externally or by a CTX286LVCT oscillator. The clock distribution is performed by an Analog Devices AD9513.

5 Case Study: Optimization of an IEEE 802.11b Transmitter SDR Implementation

In this section we will introduce a IEEE 802.11b related physical layer WLAN algorithm. The algorithm is implemented in C-Code and can be executed on our CoreVA architecture. The software processes MAC packets from the higher layers. The output is the discrete representation of the baseband signal to be transferred. Using a R/F-frontend as described in 4, the data is D/A converted and mixed to the ISM[6]-band and send to the receiver. The protocol is compatible to the algorithms of the GNU-Radio[7]. For sake of simplicity we only describe the transmitter part of the 802.11b standard in this section.

The implementation of the data path of the IEEE 802.11b algorithm consists of four function blocks (cf. Figure 3):

– Scrambler
– Differential Encoder

[6] Industrial Scientific Medical.
[7] http://www.gnuradio.org/

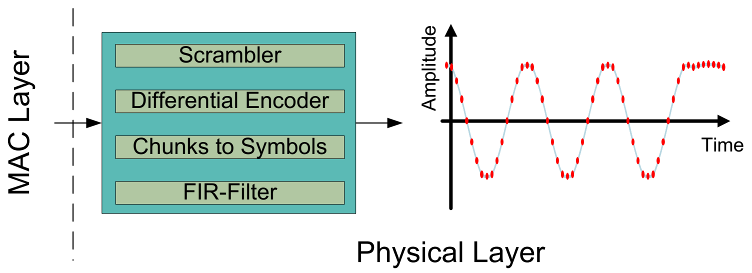


Fig. 3. Function blocks of the IEEE 802.11b algorithm

- Chunks to Symbols
- Interpolation FIR Filter

To enable clock recovery at the receiver, the transmitted signal must not contain long non-constant sequences. In the *Scrambler* the data is scrambled in a way which can be recovered by implementing a linear feedback shift register (LFSR). During the transmission the phase of the signal can be shifted by interference on the channel or by loss of synchronization between transmitter and receiver. If the phase exceeds a threshold a phase ambiguity emerges and the receiver misinterprets the received symbols. In BSPK coding, this effect emerges at phase shifts of more than $+90°$ and less than $-90°$. In QPSK misinterpretation can occur at shifts of $\pm 45°$ because of the fewer distance of the symbols. The *Differential Encoder* avoids these faults in advance by adding dependencies between the current and the subsequent bit. The output of the scrambler is combined bitwise by a modulo-2 operation to the last computed bit. By this only differences of consecutive bits are transferred. If two consecutive bits differ, a 1 is transferred, otherwise a 0. If a continuously phase shift occurs, the interpretation of the symbols might change but the dependencies can be restored in the receiver. Within the function block *Chunks to Symbols* the output of the differential encoder are mapped to the BPSK symbol space of IEEE 802.11b with a phase offset of $\frac{\pi}{4}$. The *Interpolation FIR(Finite Impulse Response)-Filter* converts symbols of the previous blocks to discrete values of the later analog transferred signal. The discrete values are characterized by the property, that a logical 0 is represented by amplitude of 0.361 and a logical 1 is represented by amplitude of -0.361. This signal implies a very wide frequency spectrum due to very steep edges. Thus it cannot be transferred efficiently. For this reason, the signal is shaped with a pulse, to apply to the ISM band. In time division, this pulse shaping is characterized by an interpolation, at which intermediate values are inserted to filter the edges. For the pulse forming, IEEE 802.11b uses the first half wave of a Root-Raised-Cosine function, which is based on an extended Sinus Cardinalis(SI)-function. The Root-Raised-Cosine function has the advantage, that a filter based on it meets the first nyquist criterion. In this work, the pulse forming is realized by a FIR filter of first order and implemented by a matrix multiplication of a vector containing the two subsequent symbols and the so called taps matrix. The taps matrix contains the discrete values of the root-raised-cosine. The number of columns complies with the order of the filter, the number of rows defines the interpolation rate. In this work, an interpolation rate of 8 is set. The result is a vector containing 8 discrete interim values. The matrix is initialized with a root-raised-cosine pulse with a roll-off-factor of 0.5.

Performance Results. The total execution time of the baseband processing of the IEEE 802.11b transmitter SDR implementation for a minimal packet (6 bytes payload of the PLCP frame $\widehat{=}$ 512 bytes baseband data) is 21100 clock cycles. At a clock frequency of 300 MHz this implies a processing time of 70 μs. The `Interpolation FIR Filter` consumes the most computation time (88 %) and is examined in particular in the optimization steps described in Section 6 (Scrambler: 4.5 %, Differential Encoder: 3 %, Chunks to Symbols: 4 %)

6 Example of Design Space Exploration

This section describes the optimization of the IEEE 802.11b transmitter SDR implementation introduced in Section 5. As the algorithm is intended to be executed on our CoreVA architecture on a mobile, energy limited device, we consider the optimization of resource consumption in terms of energy, so first and foremost we try to reduce execution time and power consumption.

The optimization process is divided into three independent intermediate steps affecting different domains of the hardware/software combination:

- Starting from the C-Reference Code, we adapt the implementation of the software to enable the compiler using architecture dependent parameters.
- Analyzing the instruction streams to introduce tightly coupled, application specific instruction set extensions (ISE) to the processor core.
- Adding loosely coupled dedicated hardware extensions

6.1 Optimizing the C-Code

Despite the fact, that a developer expects from an "ideal" compiler to convert C-code with the same behavior to identical (and optimal) machine code, reality does not look the same. The way of describing the functionality of the algorithm impacts the quality of the results. One reason (especially when using parallel systems like VLIW) is that compiling is always a tradeoff between code quality and compiler effort (compile time). On the other hand different implementations of an algorithm supposed to be obviously identically turn out to be different and have a great impact on the compilation result. One example is the non equivalence of `logical shift left` and `division by a power of two` when not using *unsigned* variables. Using variables not adapted to the width of the data path can force the compiler to insert additional instructions to handle overflows, even if this cannot occur with the set of input values used later. Manual unrolling of loops can improve speed, if the compiler cannot prove a fixed number of iterations during compile time. Aligning structures to memory boundaries can reduce clock cycles to calculate memory addresses, on the other hand, this often leads to a tradeoff between performance and memory requirements (In addition, the setup and output delays of a memory macro scales with its size and hereby can impact the maximum frequency of the total system.). Considering these rules the execution time of the IEEE 802.11b algorithm could be reduced by 25 %. Using the evaluation applications of our tool flow allows quick iteration steps in the design space exploration. For example, at this point of optimization, no modifications to the hardware had to be made. The resulting C-code is still portable to other architectures, and most of the modifications would produce performance gains on similar architectures, as well.

6.2 Introducing Instruction Set Extensions

Successive instructions with data dependencies can be combined to one single instruction. These instructions are derived directly from the evaluation applications of our tool flow. As such instruction sequences base on existing fundamental operations of the functional units, their combination often requires a negligible amount of additional hardware resources. For the proposed IEEE 802.11b transmitter SDR implementation, an `Multiply Accumulate` instruction is followed by a data dependent `arithmetic shift right` in 2304 clock cycles. After introducing an extension to the instruction set, the application could be sped up by about 4 %. The critical path of the architecture is not affected. The increase in area and power consumption is negligible and below the variations of non deterministic synthesis processes. As execution time is reduced the required energy for the processing of the algorithm is reduced analogously (cf. Figure 4). Following this example several candidates of instruction pairs can be implemented to further improve the energy balance.

6.3 Loosely Coupled, Dedicated Hardware Extensions

Moving from general purpose optimization methods not exclusively suited for the IEEE 802.11b application, the next step is to introduce dedicated hardware extensions, to which whole parts of the algorithms can be migrated to. Usually, these extensions have higher performance but increased resource requirements compared to instruction extensions. As before, the developer has to consider this tradeoff for example by considering the required energy as a ranking measure. In the example presented in this section, we implemented a hardware accelerator for the processing of 802.11b MAC packets. The packets are transferred to the hardware extension, processed by four dedicated components representing the four components of the reference software implementation described in Section 5 (cf. Figure 5).

The combinatorial logic is decoupled by input and output registers to avoid impacts on the critical path of the processor core. Address decoders control the distribute the data from the CPU to the internal registers of the hardware

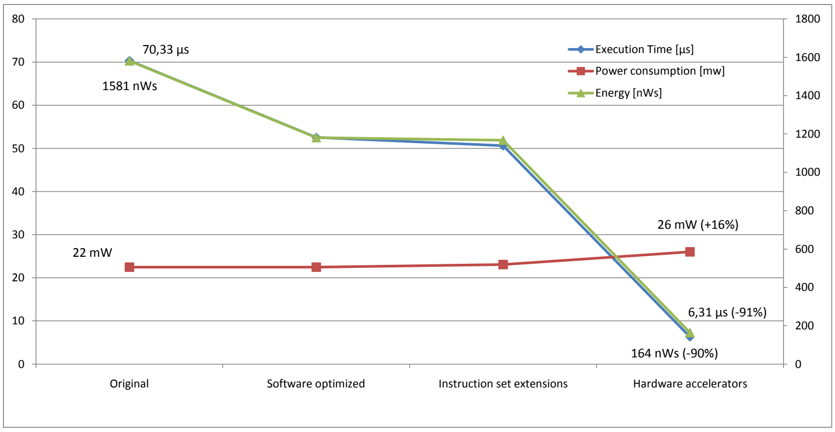


Fig. 4. Optimization results for the IEEE 802.11b transmitter SDR implementation

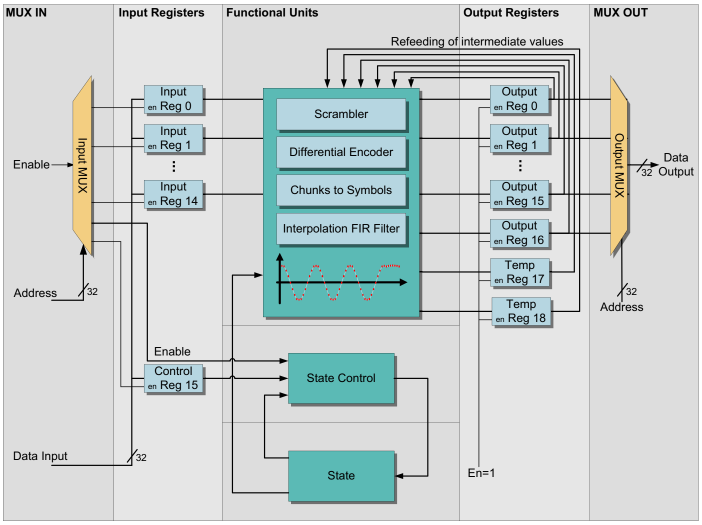


Fig. 5. Block diagram of the IEEE 802.11b hardware accelerator

extension and vice versa. All parts of the hardware extension share the same internal registers for intermediate data. A state machine controls the access to the registers and the processing of the data. The execution time for the processing of a data byte is 5 clock cycles. The final results of the computations are the discrete signal values transferred to the physical layer. The maximum clock frequency of the system including the hardware accelerator is not affected. The area requirements are increased by about 32 %, the power consumption by about

16 %. The execution time of IEEE 802.11b algorithm using the hardware accelerator is reduced by about 90 %. Figure 4 depicts the optimization results the presented optimization steps. By adding the hardware accelerators, the energy consumption could be reduced by about 91 %.

7 Conclusion

We presented a framework for the design space exploration and development of SDR architectures. This framework is based on dual design flow, consisting of the automatic generation of a complete C-compiler tool chain based on a high level reference specification in the UPSLA languagend a conventional RTL-based hardware development. We presented the CoreVA VLIW architecture for the execution of the baseband algorithms. Functional verification is performed using our rapid prototyping environment RAPTOR For this modular architecture a RF-frontend has been developed. This module can be extended with commercial RF-transceivers from Ettus Research. To demonstrate the effectiveness of our tool flow, we presented the mapping of a IEEE 802.11b transmitter SDR implementation to our architecture and optimized the system in terms of execution time and energy consumption by introducing instruction set extensions and dedicated hardware accelerators. These hardware extensions achieve a speed up by a factor of ten and reduce energy by about 91 %.

Acknowledge

Substantial parts of the research described in this paper were funded by the Federal Ministry of Education and Research (Bundesministerium für Bildung und Forschung – BMBF) registered there under grant number 01BU0643 (Easy-C).

References

1. Bagheri, R., Mirzaei, A., Chehrazi, S., Heidari, M., Lee, M., Mikhemar, M., Tang, W., Abidi, A.: An 800MHz to 5GHz software-defined radio receiver in 90nm CMOS. ISSCC Dig. Tech. Papers, 480–481 (2006)
2. Bluethgen, H.M., Grassmann, C., Raab, W., Ramacher, U., Hausner, J.: A programmable baseband platform for software-defined radio. In: Proceedings of SDR FORUM (2004)
3. Dreesen, R., Jungeblut, T., Thies, M., Porrmann, M., Kastens, U., Rückert, U.: A Synchronization Method for Register Traces of Pipelined Processors. In: Analysis, Architectures and Modelling of Embedded Systems, pp. 207–217. Springer, Boston (2009)
4. Halfhill, T.R.: Intel maps wireless future. Microprocessor Report 23, 1–4 (2003)
5. Jungeblut, T., Dreesen, R., Porrmann, M., Rückert, U., Hachmann, U.: Design Space Exploration for Resource Efficient VLIW-Processors. In: University Booth of the Design, Automation and Test in Europe (DATE) Conference (2008)

6. Kastens, U., Le, D.K., Slowik, A., Thies, M.: Feedback driven instruction-set extension. In: Proceedings of ACM SIGPLAN/SIGBED 2004 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES 2004), Washington, D.C., USA (June 2004)
7. Mitola, J.: The software radio architecture. IEEE Communications Magazine 33(5), 26–38 (1995)
8. Mitola III., J.: Software radios: Survey, critical evaluation and future directions. IEEE Aerospace and Electronic Systems Magazine 8(4), 25–36 (1993)
9. Porrmann, M., Hagemeyer, J., Romoth, J., Strugholtz, M.: Rapid Prototyping of Next-Generation Multiprocessor SoCs. In: Proceedings of Semiconductor Conference Dresden, SCD 2009, Dresden, Germany, pp. 29–30 (2009)
10. Shiba, H., Shono, T., Shirato, Y., Toyoda, I., Uehara, K., Umehira, M.: Software defined radio prototype for PHS and IEEE 802.11 wireless LAN. IEICE Transactions On Communications E Series B 85(12), 2694–2702 (2002)
11. Shono, T., Shirato, Y., Shiba, H., Uehara, K., Araki, K., Umehira, M.: IEEE 802.11 wireless LAN implemented on software defined radio with hybrid programmable architecture. IEEE Transactions on Wireless Communications 4(5), 2299–2308 (2005)