Application-Specific Instruction-Set Processor for Digital Front-End Processing in Wireless Communications

Carina Schmidt-Knorreck* , Renaud Pacalet† , Andreas Minwegen‡ , Uwe Deidersen‡,
Torsten Kempf‡, Raymond Knopp* , Gerd Ascheid‡

* Mobile Communications Department, EURECOM, Sophia Antipolis, France,
carina.knorreck@eurecom.fr, raymond.knopp@eurecom.fr
† TELECOM ParisTech, Sophia Antipolis, France, renaud.pacalet@telecom-paristech.fr
‡ Institute for Communication Technologies and Embedded Systems, RWTH Aachen, Germany, name.surname@ice.rwth-aachen.de

***Abstract*—High computational demands and multimodal processing of wireless communication standards rise the need of flexible Software Defined Radio (SDR) platforms. A key part is the Front-End Processor responsible for the computation of different air-interface algorithms at the receiver and transmitter side. In this context, Application Specific Instruction-set Processor (ASIPs) allow to tradeoff the flexibility of General Purpose Processors against the performance of ASICs. In this paper we present a flexible high performance ASIP for front-end processing that has been designed for the OpenAirInterface Platform using the LISA language. Architectural details are given as well as synthesis results for different target technologies. Besides the runtime performance of the ASIP is analyzed for different air-interface operations.**

I. Introduction

Since the 1880ths when the first wireless telephone service was patented, the number of different wireless communication standards has grown rapidly. Today, smartphones provide more and more applications and a high data-rate access has become of major importance. The requirements of these applications embrace different wireless communication standards like GSM, 3GPP UMTS, WLAN 802.11a/b/g. Important criteria in design are not only cost, power, area or speed but also the easy adaptation to future wireless standards like LTE. Besides, the evolution of existing standards is still in the focus of research and leads to improved algorithms, smaller designs and a better performance. To deal with all these issues is a complex task and still a hot topic. One solution can be found in the concept for Software Defined Radio (SDR) [1]. The key idea is the support of multiple standards by just one flexible platform. One of them is the digital baseband processing platform being developed by Eurecom and TELECOM ParisTech [2] whose multimodal design supports a high number of different air interfaces. In contrast to other platforms, the baseband processing is split over several independent processing engines which are synchronized by a 32-bit Sparc processor. The partitioning between HW and SW follows a general cost-and-complexity versus speed trade-off. All processing engines are parameterizable and meet the latency requirements of the computationally most intensive task.

In this paper we focus on the implementation of an Application Specific Instruction-set Processor (ASIP) for flexible front-end processing that has been designed using the Language for Instruction-Set Architectures (LISA) [3]. The Front-End Processor (FEP) processing engine is designed to deal with the different air-interface operations at the receiver and transmitter side. Today, air interfaces used by the different standards include Orthogonal Frequency Division Multiplexing / Multiple-Access (OFDM/A), Single Carrier FDMA (SC-FDMA), Wideband Code Division Multiple Access (W-CDMA) and Space-division multiple access (SDMA). The set of air-interface operations to be performed by the FEP comprises Channel Estimation, Synchronization, Carrier Frequency Offset Estimation, Data Detection and Coarse Frequency Offset Estimation.
The ASIP design approach is an attempt to find a compromise between two extremes. ASICs on the one side are optimized architectures for a specific application with high performance and low power consumption. General Purpose Processors (GPPs) on the other side are very flexible and fully programmable, but with high power consumption and low performance. ASIPs still exhibit a high degree of flexibility in combination with a higher performance. In addition, a tool-based design comes with and a shorter design time when compared to the conventional RTL design flow. As stated in [4], the flexibility of ASIPs allow fast design modifications to respond to the evolution of standards, world-wide compatibility, changes of user requirements depending on the quality of service, etc. as their hardware design flow provides a fast and efficient way to generate VHDL or Verilog code from a High Level Language (HLL) tool. For our design, the LISA language has been chosen, which has gained commercial acceptance over the last years.

After a short overview of the entire system in section II and a brief introduction of the different air-interface operations

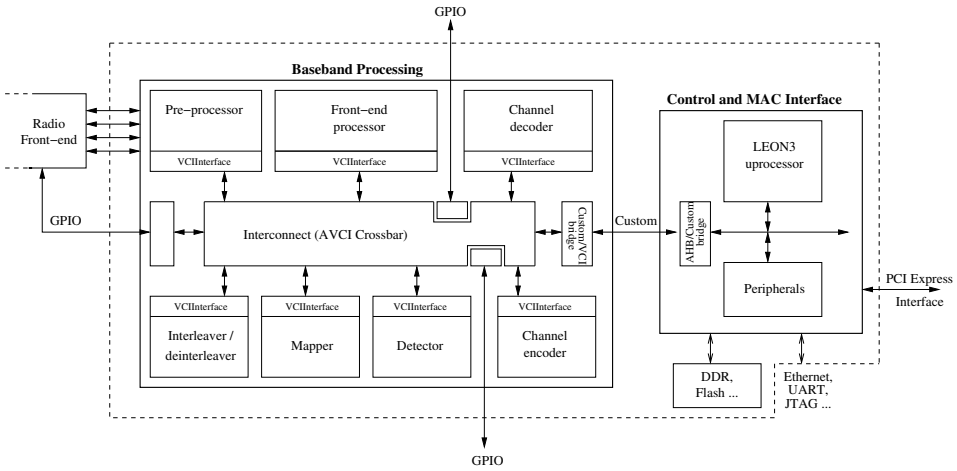

Fig. 1. OpenAirInterface Baseband Architecture

to be implemented at the transceiver side in section III, the architecture of the ASIP is presented in section IV-B. Frequency and area results for different targets are provided and the ASIP is analyzed by comparing several air-interface algorithms for different standards in terms of execution time in section V.

II. System Overview

The ASIP being presented in this paper is part of the OpenAirInterface Platform (Fig. 1) which is a generic prototype architecture for SDR applications. This platform supports a wide range of different standards like 3GPP UMTS, WLAN 802.11a/g/p, WiMAX, GSM and can easily be adapted to future standards like LTE, meeting all their throughput and latency requirements. As the current version of the platform is a prototype architecture and not meant for large-scale production, FPGAs have been chosen as target technology, due to their reduced design cycle, their higher flexibility and lower costs. In the future, ASICs will be considered as the target of interest to minimize the required area and to achieve a higher performance.

The baseband processing of the platform implements the digital part of the physical layer and is split over several independent processing engines, like Channel Decoder or Front-End Processor (FEP), that are connected via a generic Advanced Virtual Component Interface (AVCI) crossbar ([5], [6]). Each processing engine is parameterizable, allowing the reuse of the same architecture for different standards. The advantages of this hardware configurability are the effective use of the spectrum, mobility, increased network capacity, cost reduction, faster deployment of new standards and an easy improvement of existing standards. The baseband processing is controlled by a Sparc LEON3 microprocessor from Gaisler that programs all processing engines and that reacts on interrupts set by them to signal their new availability.

The architecture of all processing engines follows the same general structure that is shown in Fig. 2. It is composed of a Control Sub-System (CSS), a processing unit (PU) and a Memory Sub-System (MSS). The CSS is configured by a set of parameters and the same for all processing engines. It includes a DMA, a local micro-controller, a set of control and status registers plus several arbiters and FIFOs for input-output requests and responses. For signaling and synchronization with LEON3, several interrupt lines are used. In contrast, MSS and PU are custom defined and depend on the functionality of each of the processing engines.

To keep this structure, the ASIP designed for the FEP has to be part of the PU whereas its program memory is part of the MSS. The other elements remain unchanged and allow an update of the program code as well as the transfer of input and output samples while the ASIP is running.

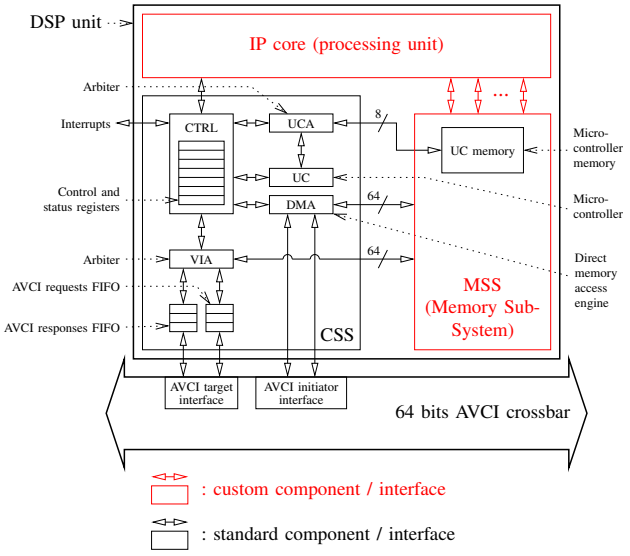

Fig. 2. DSP Unit

III. Frontend Processing Algorithms

The air-interfaces of todays standards include Orthogonal Frequency Division Multiplexing / Multiple-Access (OFDM/A), Single Carrier FDMA (SC-FDMA), Wideband Code Division Multiple Access (W- CDMA) and Space-division multiple access (SDMA). The resulting set of air-interface operations to implement at the receiver and transmitter side comprise Channel Estimation, Sychronization, Carrier / Coarse Frequency Offset Estimation or Data Detection. As these operations depend on the standard, a flexible design requires a parameterizable architecture that does not change when switching to another standard. In literature it already been shown (among others in [7]), that all air-interface operations can be build up from a DFT/IDFT unit and a set of functions which are

- Component-Wise Addition of Vectors
- Component-Wise Division
- Component-Wise Product of Vectors
- Dot Product
- Energy Calculation
- Maximum / Minimum, Argmax / Argmin Calculations

IV. ASIP Design

A. Functional Specification

The ASIP has been designed to replace the existing vector processing unit of the FEP [8]. As shown in Fig. 3, the DFT/IDFT is kept as a separate unit inside the processing engine. The interface between the ASIP and the MSS is custom defined and establishes the connection between the ASIP, the Program Memory with a size of 4kBytes and the input-output data space. Values read from the MSS are available after a delay 3 three cycles. This delay results from three registers inside the MSS that are used to separate the arbiter logic from the memory access. Besides, the MSS is output registered to avoid a critical path between MSS and PU.

To increase the flexibility and thus the programmability of the design, the instruction set of the ASIP should not contain complex operations like Energy Calculation or Dot Product (please refer to section III). Instead, these air-interface operations have to be split into a set of different vector operation instructions. It can be seen that the Energy Calculation of a vector $X[i]$, which is defined as

$$E(X) = \sum_{i=0}^{N-1} |X[i]|^2$$

can be computed by a vector square modulus and a vector sum. Similarly, the Dot Product of two vectors $X[i]$ and $Y[i]$is defined as

$$X.Y = \sum_{i=0}^{N-1} X[i] \times Y[i]$$

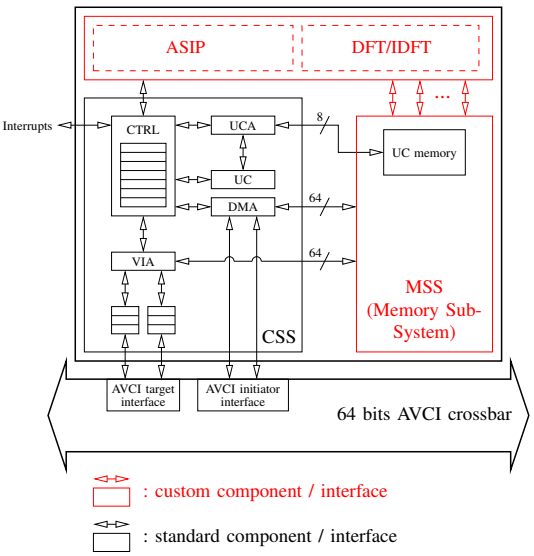

Fig. 3. FEP DSP Unit including the ASIP

and can be computed by a component wise vector multiplication and a vector sum.

The resulting instruction set comprises nine different vector operation instructions that operate over vectors with a length of maximum 4096 elements:

- Component-Wise Addition:
  $Z[i] = X[i] + Y[i]$
- Component-Wise Product:
  $Z[i] = X[i] \times Y[i]$
- Component-Wise Square of Modulus:
  $Z[i] = |X[i]|^2$
- Copy a vector from one MSS location to another:
  $Z[i] = X[i]$
- Component-Wise Filter by a Lookup Table (CWL):
  $Z[i] = Y[X[i]]$
- Component-Wise Square:
  $Z[i] = X[i]^2$
- Vector Sum:
  $Z = \sum X[i]$
- Vector Right/Left Shift:
  $Z[i] = X[i] >> l$, $Z[i] = X[i] << l$
- Vector Max/Argmax or Min/Argmin:
  $Z = max(X[i])$, $Z = min(X[i])$

Based on the vector operation that is processed, one or two vectors are read in from the MSS and one vector is written back. The obtained throughput of the design is two samples per cycle. Supported data types are 8- or 16-bits signed integer vectors and complex vectors whose real and imaginary part are 8- or 16-bits signed integers. It is possible to process two input vectors with different types and to write back the result in a third type. The type conversion depends on parameters that are part of the vector operation instruction word. Furthermore, pre- and post-processing value

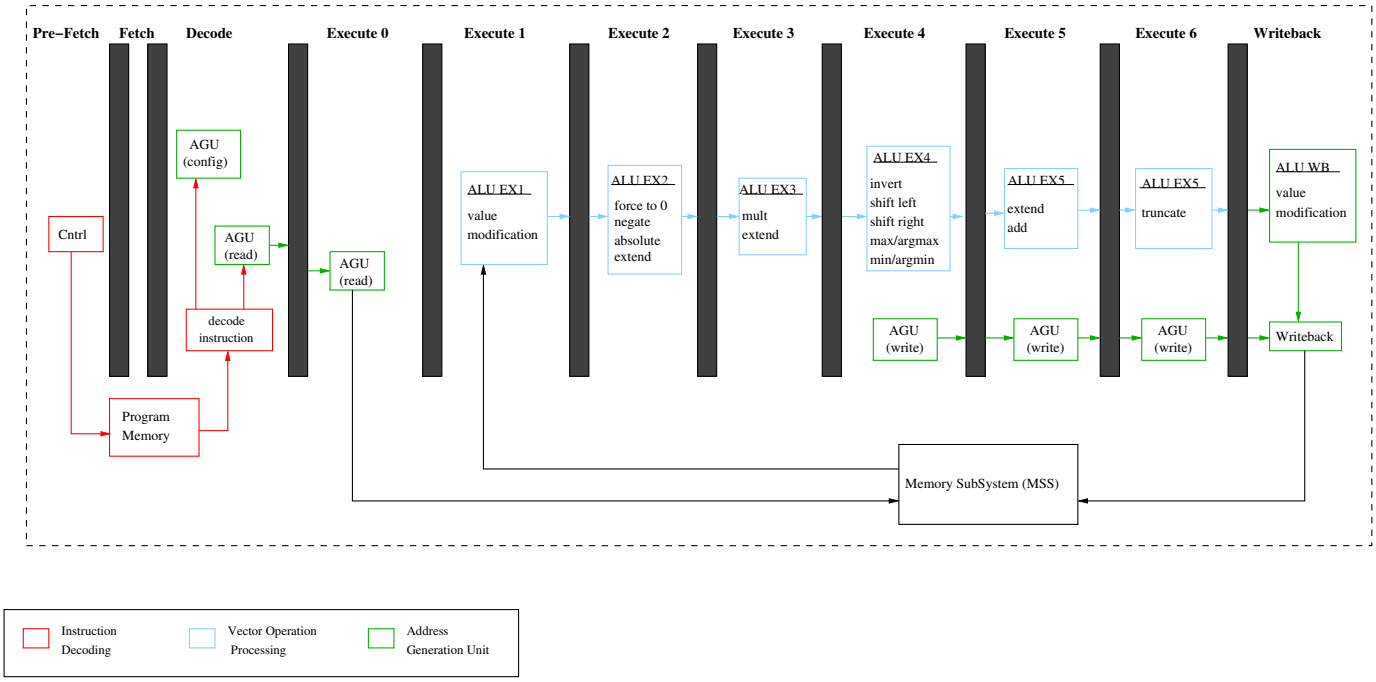

Fig. 4. Pipeline of the ASIP for FEP

modifications are applied. The first one comprises absolute value calculation, negation, zeroing and rescaling while the latter supports downscaling and saturations.

Using a programmable addressing scheme, input vectors can be read from non-contiguous addresses in the MSS. The same applies to the result vector. Supported are skipping or repetition of addresses. Besides, addresses can be periodic or may wrap around boundaries inside the MSS. These boundaries can divide one bank of the input-output data space (size of 4096x32bit) into a half, a quarter or an eighth and allow to turn these sections into circular buffers.

With this design, additional functions like a component-wise subtraction of vectors or a component-wise division can easily be implemented by using the provided instruction set. To program a vector division, the CWL and the component-wise multiplication instruction are needed: performing the CWL instruction a value of vector X is first read in before its eleven MSB are used to obtain the value $Y[X[i]]$ from the MSS where Y represents a Look-Up Table. The result is then stored back in the MSS and can then be used for a component-wise multiplication to get the division result.

B. Architecture

The pipeline of the ASIP consists of eleven stages to achieve a high performance of the design (Fig. 4). Based on the delay of the MSS, the instruction word stored in the Program Memory is decoded after three cycles. Two different types of instructions have been implemented: configuration instructions for address generation and vector operation instructions. The length of the instructions is set to 32 bit. Therefore the address generation parameters have to be split over six different instructions. The vector operation parameters can be provided in one single instruction.

The instruction set of the ASIP comprises nine different multicycle vector processing instructions to fulfill the functional requirements of the FEP that have been stated in section IV-A. Except for the CWL instruction, the number of cycles required for processing are $l/2 + 14$, where l is the length of the vector.The CWL instruction ($Z[i] = Y[X[i]]$) introduces an additional delay of four cycles as $X[i]$ has first to be read from the MSS before its 11 LSB are used to access the value of $Y[i]$.

Furthermore, the instruction set contains some configuration instructions for the Address Generation Unit (AGU). The number of these instructions (between 3 and 6) can vary and depends on the amount of parameters to be updated. The AGU operates in parallel, while a vector processing instruction is executed.

V. Results

A. Synthesis Results

The processor generator software tool (Synopsys Processor Designer [9]) provides a fast and efficient way to generate VHDL code out of the LISA description. This code can be directly used as an input for synthesis tools for different targets.

The tool generated code has first been synthesized with Precision RTF from Mentor Graphics for a Xilinx Virtex 5 LX330 FPGA with a speed grade of -2. The achievable frequency when synthesizing the ASIP together with its MSS is 123

MHz after place and route. Area results are provided in Table I.

	Function Generators	CLB Slices	DFFs	Block RAMs	DSP48E
ASIP	10269	2568	6408	17	8

TABLE I
AREA, FPGA TARGET

Furthermore the ASIP has been synthesized for a 65nm CMOS standard cell target technology. The library is a low power, high voltage threshold one and is characterized for a typical manufacturing process at 1.2 Volts power supply and 25C temperature. The required silicon area of the ASIP (without MSS) is about 0.12mm2 with a frequency of 550 MHz.

B. Runtime Performance

The runtime performance for the ASIP is given for a set of different air-interface operations of OFDM signals. These operations comprise Energy Detection (ED), Time Synchronization (SYNC) and Channel Estimation (CE). For the calculation of the processing time it is assumed, that the DFT has the same throughput than the one of the open-source implementation of the FEP which is $2 + (13 + 2^n/8) \times \lceil n/2 \rceil$ clock cycles.

1) Energy Calculation: To compute the energy of a given vector two vector operation instructions have to be performed: vector square modulus + vector sum. These instructions take 284 clock cycles. For a frequency of $f = 123$MHz this would result in a processing time of 2309 ns, while for a frequency of $f = 550$MHz the processing time would be about 516 ns.

2) Time Synchronization: The Synchronization is achieved by a sliding FFT window that is correlated with a know waveform before the peak inside the window is detected. Using the ASIP, the order of operations to be performed is as follows: FFT, component wise product, IFFT, vector square modulus, vector max/argmax. In best case, this algorithm is performed three times which results in a total number of 2112 clock cycles. For a frequency of $f = 123$MHz this would result in a processing time of 17171 ns (5728 ns per iteration), while for a frequency of $f = 550$MHz the processing time would be about 3840 ns.

3) Channel Estimation: The channel estimation is achieved by a convolution of the received pilot sequencewith its known reference. These instructions take 89 clock cycles. For a frequency of $f = 123$MHz this would result in a processing time of 724 ns, while for a frequency of $f = 550$MHz the processing time would be about 162 ns.

Taking the example of the 802.11p receiver, the Short Training Symbol (STS) used for Synchronization has a length of 160 samples which corresponds to 16us for a 10MHz channel spacing. The algorithm to be performed is a combination of an Energy Detector and the Time Synchronization. The sliding window has a size of 160 samples (16us). In worst case, Energy Detection and two iterations of the Time Synchronization have to be performed in one cycle. The resulting processing time is 13.765us for the FPGA target which would leave some time for other processes on the platform.

VI. CONCLUSION AND FUTURE WORK

In this paper we focused on an ASIP design of the Front-End Processor processing engine for the OpenAirInterface platform. The presented architecture is very flexible, exhibits a high performance and complies to the real time requirements of the latest wireless communication standards.

Our future work includes the extension of the instruction set to run the ASIP as a stand-alone processing engine. Furthermore this design will be compared to the open-source custom implementation of the FEP in terms of flexibility, area and performance for different target technologies.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the FP7-ICT ACROPOLIS (Advanced coexistence technologies for radio optimization in licensed and unlicensed spectrum) project.

REFERENCES

[1] J. Mitola, "The software radio architecture," *Communications Magazine, IEEE*, vol. 33, no. 5, pp. 26 –38, may 1995.

[2] N.-u.-I. Muhammad, R. Rasheed, R. Pacalet, R. Knopp, and K. Khalfallah, "Flexible baseband architectures for future wireless systems," in *Digital System Design Architectures, Methods and Tools, 2008. DSD '08. 11th EUROMICRO Conference on*, sept. 2008, pp. 39 –46.

[3] A. Hoffmann, O. Schliebusch, A. Nohl, G. Braun, O. Wahlen, and H. Meyr, "A methodology for the design of application specific instruction set processors (asip) using the machine description language lisa," in *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, ser. ICCAD '01. Piscataway, NJ, USA: IEEE Press, 2001, pp. 625–630. [Online]. Available: http://dl.acm.org/citation.cfm?id=603095.603223

[4] J. R. Cavallaro and P. Radosavljevic, "Asip architecture for future wireless systems: Flexibility and customization," in *Wireless World Research Forum (WWRF)*, Jun. 2004.

[5] "VSIA consortium: http://www.vsi.org/." [Online]. Available: http://www.vsi.org/

[6] "Vsi alliance virtual component interface standard version 2 (ocb 2 2.0)."

[7] K. van Berkel, F. Heinle, P. P. E. Meuwissen, K. Moerman, and M. Weiss, "Vector processing as an enabler for software-defined radio in handheld devices," vol. 2005. New York, NY, United States: Hindawi Publishing Corp., January 2005, pp. 2613 –2625.

[8] N.-I. Muhammad, K. Khalfallah, R. Knopp, and R. Pacalet, "Reconfigurable dsp architectures for sdr applications," in *Electronics, Circuits and Systems, 2007. ICECS 2007. 14th IEEE International Conference on*, dec. 2007, pp. 971 –974.

[9] "http://www.synopsys.com/systems/blockdesign/processordev/pages/default.aspx."