Flexible Front-End Processing for Software Defined Radio Applications using Application Specific Instruction-Set Processors

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

Flexible Front-End Processing for Software Defined Radio Applications using Application Specific Instruction-Set Processors

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 University, Germany, name.surname@ice.rwth-aachen.de

***Abstract*—High computational demands of today's wireless communication standards require the design of highly flexible Software Defined Radio (SDR) platforms like the OpenAirInterface ExpressMIMO platform. A DSP engine of major importance is the Front-End Processor (FEP) which deals with the different air-interface operations at the transceiver side. In this paper we propose an Application Specific Instruction-Set Processor (ASIP) architecture for front-end processing and compare it to a programmable DSP engine as well as to other ASIP solutions. For design comparison we mainly focus on architectural differences and the run-time performance in terms of processing time. The synthesis results are provided for different target technologies.**[1]

I. Introduction

Recently, we have witnessed a significant change in the use of mobile phones and other mobile devices. A few years ago these devices focused solely on providing voice communication. In contrast, today's smartphones support a wide range of applications and high data-rate access becomes of major importance. In addition, the requirements of different applications and the variable environment requests the support of multiple wireless communication standards. For example, available smartphones typically include GSM, 3GPP UMTS, WLAN 802.11a/b/g, Bluetooth and most likely LTE in the near future. It is expected that this number increases due to upcoming standards, like LTE Advanced and WiMAX, while at the same time updates of existing standards need to be supported as well.

The high computational demands of such wireless communication standards, especially in the physical-layer (PHY-layer), have been commonly answered by a dedicated subsystem per standard. To allow the execution of the different modes of the given standard, each of them has been implemented by a set of configurable hardware accelerators. By nature, these systems have limited flexibility and can mostly support only the standards they were intended for. Therefore, changes in existing standard specifications or the implementation of new standards require a time-consuming and costly redesign of the hardware architecture.

These issues have given birth to the concept and idea of Software Defined Radio (SDR). Key idea is to provide a flexible SDR platform that can support multiple wireless communication standards in a multimodal fashion. Unfortunately, adding flexibility to a hardware design usually comes with the cost of increased area, increased energy-consumption and/or reduced computational performance. Earlier investigations [1] have illustrated that a large amount of computational complexity can be efficiently implemented by a vector processing unit and SIMD (Single Instruction Multiple Data) instructions. This paradigm is also visible in recently released SDR platforms in commercial products like Femtocells from TI [2] and Freescale [3], as well as in SDR platforms from academia [4]. In contrast to these solutions, the baseband processing of the OpenAirInterface ExpressMIMO platform [5] is split over several independent subsystems, as seen in Fig. 1.

In this paper we focus on the design of a flexible Front-End Processor (FEP) for the ExpressMIMO platform. For this purpose, a thorough comparison between a programmable tool-based Application Specific Instruction-Set Processor (*ASIP*) denoted as the *A-FEP*, a previously designed programmable DSP engine (the *Custom* FEP (C-FEP) in the following) and two other ASIP solutions from academia ([6], [7]) is carried out.

For our ASIP design we used the Language for Instruction-Set Architecture (LISA) [8] which has gained commercial acceptance over the last years. Like the C-FEP, the A-FEP achieves the required real time requirements of latest wireless communication standards when executing the front-end processing part of the physical-layer.

The paper is structured as follows: After presenting the related work in Section III and a brief introduction of the underlying front-end processing algorithms and the functional specification of the A-FEP in Section IV-A, the architecture of the A-FEP is enhanced in Section IV-B. Usually, architectures

[1]The research work leading to this paper has been supported by the European FP7 project ACROPOLIS (Advanced coexistence technologies for radio optimization and unlicensed spectrum)

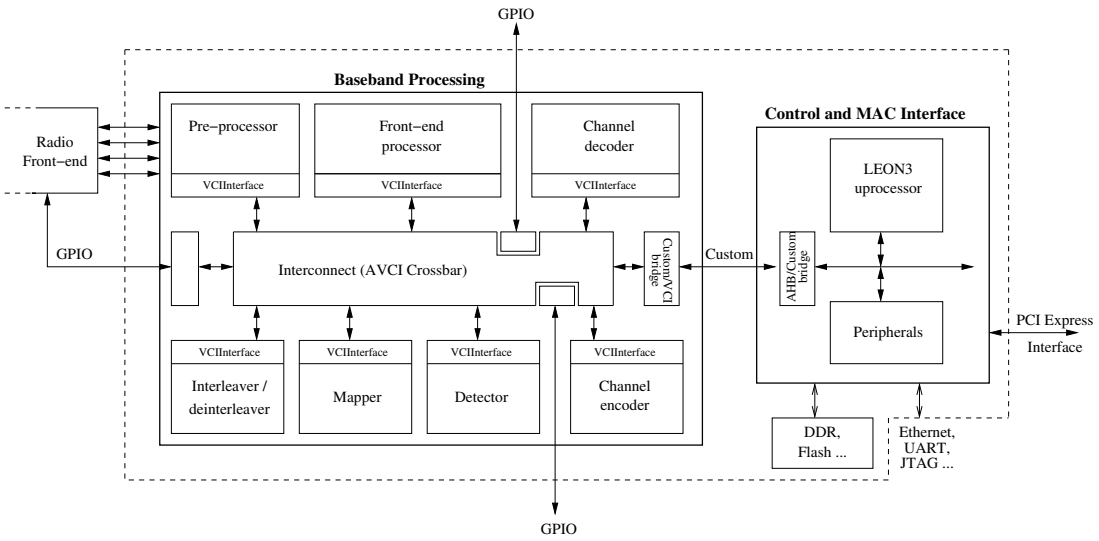

Fig. 1. ExpressMIMO Platform - System Overview

are evaluated in terms of frequency, area, power consumption and the number of MOPS / MIPS (millions of operations / instructions per second). As the latter does not provide a clear information about the processing time of different operations executed on ASIPs, we will provide processing time results based on the actual amount of cycles instead. Two recent academic solutions providing this information are the ASIP developed by ETH Zürich [7] and the one designed by the Cairo University [6]. In Section III, the architectures chosen for comparison are shown before we finally focus on the results of the run-time comparison in Section V.

II. System Overview

The ExpressMIMO platform is flexible SDR platform that supports a wide range of different wireless communication standards like GSM, UMTS, WLAN or DAB. To simplify upgrades to future standards like LTE the baseband processing implemented on a Xilinx Virtex 5 LX330 FPGA is split over different independent programmable DSP engines that are connected via a generic Advanced Virtual Component Interface (AVCI) crossbar [9]. The platform further embeds a SPARC LEON3 processor from Gaisler - Aeroflex [10] running on a Xilinx Virtex 5 LX110 FPGA. LEON3 is responsible for the scheduling of the different baseband tasks as well as for the data transfer between the PHY layer implemented on the baseband engine and the MAC layer running on a host PC. The chosen Operating System (OS) is MutekH [11] whose flat function call convention, flat registers and simplified interrupt handling reduce the latencies significantly when compared to other OS like eCos or RTEMs.

The design of each DSP engine on the baseband FPGA follows the general structure shown in Fig. 2. This standardized DSP shell is composed of a Control Sub-System (CSS), a DMA engine, a processing unit (PU) and the Memory Sub-System (MSS). MSS and PU are custom defined and depend on the functionality of the DSP. Currently all DSP engines are controlled by the LEON3 processor which results in a centralized control flow on the platform. To decrease the resulting communication overhead, optionally a 8 bit micro-controller (UC) coming with a 2 kB data memory can be included in the DSP shell to enable a distributed control flow. During our ongoing work we experienced, that for standards operating on small vector length like IEEE 802.11a where one OFDM symbol includes 80 sub-carriers, the communication overhead leads to a significant performance drop. Therefore we decided to extend the functionality of the A-FEP by a set of General Purpose (GP) instructions to overcome this drawback. The UC can still be kept in the design but only for the programming of the DMA engine.

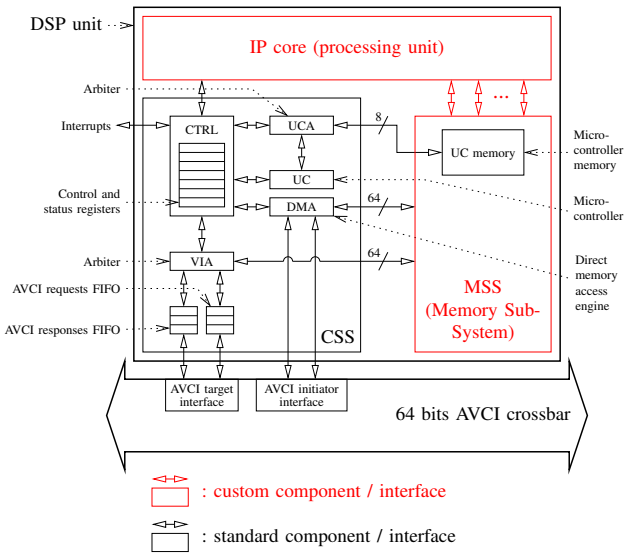

Fig. 2. OpenAirInterface ExpressMIMO standardized DSP shell

III. Related Work

In the context of SDR platforms, one promising design solution are ASIPs that can be seen as a class of microprocessors coming with a specialized Instruction-Set Architecture (ISA). For SDR platform design, ASIPs tend to be suitable candidates as they are meant to fill the gap between General Purpose Processors (GPPs) and Application Specific Integrated Circuits (ASICs) [12]. Being tailored to a specific application, ASIPs exhibit a lower energy consumption than GPPs or Digital Signal Processors (DSPs) while offering a higher flexibility than ASICs at the same time. During the past years, different solutions for front-end processing ASIPs have been proposed. Some of the the architectures focus only on some air-interface algorithms like packet synchronization or channel estimation (e.g. [13] or [14]) while other designs are tailored to the processing of a specific group of standards. One example is [15] where the proposed ASIP solution supports the execution of the 802.15.4a standard only.

Instead, the A-FEP presented in this paper supports a wide range of different air-interface operations like packet / timing synchronization, channel estimation, carrier / coarse frequency offset estimation or data detection for different air-interfaces like OFDM/A or SDMA and is not tailored to a specific standard but to wireless communication standards in general.

For performance evaluation, the A-FEP is compared to three different solutions in Section V:

- The **C-FEP** is a programmable DSP version of the FEP and the ancestor of the ASIP. The comparison of C-FEP and A-FEP is used to highlighten the performance gain obtained on the ExpressMIMO platform when using the latter instead.
- The IEEE 802.11a/n ASIP solution for single and multiple antennas implemented at ETH Zürich and presented in [7] is a well known ASIP architecture of high performance. It is further denoted as **ASPE A**.
- The recently published ASIP solution developed by the Cairo University [6] used for synchronization and acquisition in OFDM receiver systems which is called **Sync-ASIP**.

As the run-time performance analysis carried out is based on the execution time, the different solutions will briefly be described in the following and differences to the presented ASIP architecture are highlighted.

A. C-FEP

A first architecture of the manually designed C-FEP has already been presented in [16]. Since then, its design has been continuously improved to get a higher performance and a higher run-time flexibility. Like for the A-FEP, the C-FEP is embedded in the standardized DSP shell shown in Fig. 2. Instead of using a Program Memory, the DSP engine is programmed through the control registers being part of the CSS. Other differences compared to the A-FEP are listed in the following:

- **Processing Unit:** Besides the vector processing unit the C-FEP additionally embeds a DFT / IDFT unit and supports a component-wise look up table operation to approximate non-linear operations like invert, log, square root, sine, cosine, etc. . The processing core is split over two identical processing units, each embedding twenty-four 25 x 18 bit signed multipliers and twelve 43 bit accumulators, which can either be used to implement two radix-4 butterflies for DFT / IDFT computation or to execute the different vector operations. The resulting pipeline consists of 15 stages and has to be emptied before the next vector operation can be executed. This results in an overhead of 11 to 16 cycles needed for initialization and termination of each vector operation.
- **MSS:** For DFT / IDFT support, the MSS is extended by twiddle factor and temporary data memories for DFT / IDFT computation with an overall size of 52 kB.

On large vectors the ratio between the communication overhead and the processing time is close to zero while it results in a significant performance drop when processing short data sets. In [17] we recently investigated in these effects by considering a centralized control flow using the LEON3 processor. Overcoming this drawback was the main motivation in designing the A-FEP solution being presented in the context of this paper.

The current target architecture of the baseband processing engine on the ExpressMIMO platform is a Xilinx Virtex 5 LX330 FPGA with a speed grade of -2. Although, an ASIC target technology may be considered for a future release[2]. For the FPGA target, processing engine and MSS of the C-FEP have been synthesized with Precision RTL from Mentor Graphics. The design obtains a frequency of 96 MHz by requiring 20119 function generators, 5030 CLB slices, 10945 DFFs, 33 block RAMs and 24 DSP48E slices.

For the ASIC target, only the processing engine of the C-FEP has been synthesized as the new design of the MSS is still part of our ongoing work. The maximum achievable frequency in this case is about 450 MHz and the area is 0.48 mm^2.

B. ASPE A

The ASIP solutions presented in [7] are based on the Adaptive Stream Processing Engine (ASPE) [18] which is a coarse-grained ASIP architecture being optimized for data processing. Main advantages are the shortened design time and the limited run-time reconfigurability for bug fixes resulting in lower costs than other solutions. The ASPE is connected to a GPP taking care of the control and of performance uncritical tasks. In contrast to the ExpressMIMO platform, ASPE includes three different types of building blocks whose number and type can be selected from a library at design time.

1) **Functional Units** (FU) contain the arithmetic operations and can be combined to implement more complex ones. The number of internal pipeline stages is flexible and can be chosen at design time.

[2] 65nm target library, low power and high voltage threshold, characterized for a typical manufacturing process at 1.2 Volts power supply and 25°C temperature

2) **Storage Units** (SU) are used for local data storage. They are connected to the FUs via a run-time configurable network.
3) **Sequencer Units** (SEQ) are used to control the configurable network between FUs and SUs. They further support control related tasks like zero-overhead loops or data dependent control flow.

In [7], two different ASIP solutions are presented that are both tailored to the processing of the IEEE 802.11a/n standard. The first one is a Single Input Single Output (SISO) receiver called ASPE A while the second one supports a 2x2 MIMO (Multiple Input Multiple Output) configuration. In addition, the design of the 2x2 MIMO receiver has further been extended by a second ASPE ASIP (called ASPE B) responsible for MMSE (minimum mean square error) estimation and MIMO detection to achieve a higher performance of the overall design.
Table I illustrates the ASPE A configuration for both designs.

TABLE I
ASPE A CONFIGURATIONS FOR THE IEEE 802.11A/N RECEIVER

Ressource	Quantity	Comments
SEQ	1	Program Memory (512 words a 192 bit) storage of the program control flow storage of the 16 bit command words
FU	1	complex-valued multiply and accumulate unit
	2	complex-valued arithmetic logic units
SU	1	Registerfile (16 registers)
	1	input data buffer (64x32 bit)
	6	data storage (256x32 bit)

By combining the different FUs, the functionality of the ASPE A has been enhanced by a set of different vector operations like CORDIC for instance.

The Synthesis results of both ASIPs are the following: For a 0.13 μm CMOS target process, the SISO receiver configuration obtains a frequency of 160 MHz and requires a silicon-area of 1.9 mm^2. Instead the MIMO receiver has been synthesized for a 0.18 μm CMOS target process. For a target frequency of 160 MHz the silicon area is 7.6 mm^2, although the maximum achievable frequency is 250 MHz

C. Sync-ASIP

The ASIP solution presented in [6] covers synchronization and acquisition of different OFDM standards like HIPERLAN/2, IEEE 802.11a or LTE. The design includes six 12 bit real adders, three 13 bit real multipliers, two 12 bit rounders, two 24 bit accumulators, ten 13 bit multiplexers and two 24 bit shifters that are distributed over three different pipeline stages. The maximum vector length supported is 256 and a maximum of one operation / cycle can be processed. Like ASPE A, the design allows the processing of the CORDIC algorithm as well as maximum likelihood or correlation functions. The MSS is built of 286 word dual-port banks à 24 bit, based on the choice of the maximum correlation length of 256 required for IEEE 802.16e and LTE. The instruction-set is composed of program flow instructions (conditional / unconditional jumps, move, ...), optimized instructions to facilitate the implementation of the synchronization tasks and vector instructions.
For a a 0.18 μm CMOS target process, the obtained frequency is 120 MHz and the area is 1.1 mm^2.

IV. A-FEP ARCHITECTURE

A. Processing Engine Requirements

The front-end processing requirements for the support of OFDM/A, SC-FDMA, W-CDMA and SDMA have already been detailed in [1] and [16]. These papers state that operations to be performed by the FEP on the transceiver side comprise channel estimation, synchronization, carrier / coarse frequency offset estimation and data detection and can be build up from component-wise vector operations and a DFT / IDFT unit. The latter is neglected for the A-FEP and kept as a separate processing engine in the baseband design of the ExpressMIMO platform.

The basic set of vector operations to be supported by the A-FEP is listed in Table II.

TABLE II
A-FEP VECTOR OPERATIONS

Component-Wise Addition	$Z[i] = X[i] + Y[i]$
Component-Wise Product	$Z[i] = X[i] \times Y[i]$
Component-Wise Square Absolute	$Z[i] = \|X[i]\|^2$
Move (MOV)	$Z[i] = X[i]$
Component-Wise Division	$Z[i] = X[i]/Y[i]$
Vector Sum	$Z = \sum X[i]$

Besides, shift, max/min and argmax/argmin operations are provided that can operate independently on the real and imaginary parts of the complex or integer vector elements being processed. In addition, pre- and post-processing value modifications are applied, comprising absolute value, negation, zeroing, rescaling and saturations. The input and output vector elements can be of four different data types: 8 or 16 bit signed integers and complex numbers where real and imaginary parts are 8 or 16 bit signed integers. Type conversions between them are possible and specified through parameters being part of the instruction word.

One major challenge when supporting a wide range of different standards with different properties is to ensure an efficient processing so that all of them meet their real-time constraints. Therefore, the A-FEP comes with a sophisticated and programmable Address Generation Unit (AGU) that allows to build input vectors from non-contiguous data sets in the connected MSS. Symmetrically, the AGU can also be used to store result vectors at non-contiguous locations, allowing component skipping or (periodic) value repetition. Moreover, programmable self-wrapping mechanisms allow to turn MSS sections into circular buffers. Major parts of the MSS are the 4 kB program memory (PM) and the input-output data space which has been designed for the support of standards operating on large vector sizes like LTE or DAB. It is split over four different memory banks, each with a size of 4096 32 bit entries. The maximum vector length that can be processed depends on the data type. For vector elements with a size of

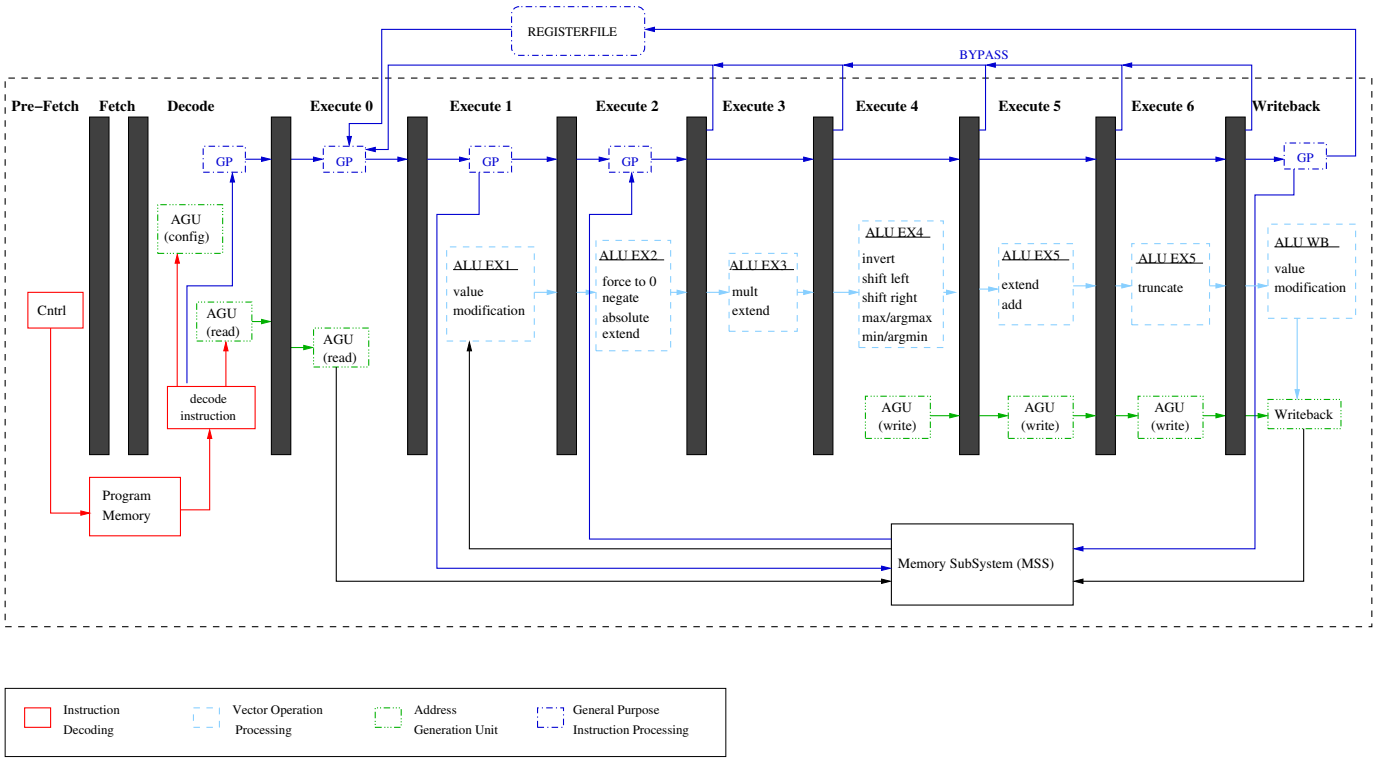

Fig. 3. ASIP Pipeline

32 bit the maximum length is 4096, for vector elements with a size of 16 bit it is 8192 and for vector elements with a size of 8 bit it is 16384.

B. HW Architecture and Instruction-Set

The instruction-set of the A-FEP comprises three different instruction types:

1) **AGU configuration instructions:** These instructions carry the necessary parameters for programming the AGU. In total, six different instructions have been implemented whose number in the program code may vary depending on the amount of parameters to be modified for the subsequent vector processing instruction.
2) **Arithmetic Vector Processing (AVO) instructions:** To fulfill the processing engine requirements, the ASIP supports nine different AVO instructions which are vector multiplication, addition, square, square modulus, sum, shift, move, division and max,min. Maximum supported vector length is 16384 entries for a vector composed of 8 bit vector elements.
3) **General Purpose (GP) instructions:** The GP instruction-set is based on a load-store architecture and supports common instructions like compare, branch or ALU operations. It comes with a registerfile possessing a size of 16 x 32 bit. Further included is a IRQ instruction used to signal to the LEON3 processor the end of a scheduled tasks. Tasks can represent a single instruction or more complex algorithms like packet synchronization. Letting the main control to LEON3 comes with the advantage of a simplified scheduling when processing different standards in a multimodal fashion.

The resulting pipeline structure is illustrated in Fig. 3. It consists of eleven stages and comes with a throughput of two vector elements per cycle. Usually, one instruction per cycle is fetched from the program memory. An exception are the multicycle AVO instructions which may operate on vectors with variable length. During the execution of these instructions, the pipeline registers between PRE-FETCH, FETCH and DECODE are stalled.

Synthesizing the A-FEP for the same FPGA target as the C-FEP, the A-FEP obtains a frequency of 105 MHz by requiring 13122 function generators, 3281 CLB slices, 6433 DFFs, 17 block RAMs and 8 DSP48E slices. For the ASIC target, only the processing engine of the A-FEP has been synthesized as the new design of the MSS is still part of our ongoing work. The maximum achievable frequency in this case is about 550 MHz with an area of 0.18 mm^2.

V. Run-time Performance Comparison

The run-time performance depends on two different factors: the necessary processing time required for the communication between the LEON3 processor and the baseband DSP engines and the pure data processing time or execution time of the DSPs. For a standard operating on small vector lengths (e.g.

IEEE 802.11a/p), the first factor is of major importance while it is more or less negligible for standards like DAB or LTE that operate on large vector lengths. Table III lists the A-FEP execution times for different front-end processing algorithms of a IEEE 802.11p receiver for a frequency of 100 MHz. Packet structure and the applied OFDM decoding procedure have recently been presented in [17] and [19].

TABLE III
A-FEP EXECUTION TIMES FOR A IEEE 802.11P RECEIVER

algorithm	cycles	execution time
energy detection	302	3.06 μs
channel estimation	141	1.41 μs
data detection (16-QAM)	159	1.59 μs
data detection (64-QAM)	273	2.73 μs

For demonstration and to compare the performance of C-FEP, A-FEP, ASPE A and Sync-ASIP, we will take the example of two different packet detector algorithms for OFDM signals.

A. Auto-correlation Based Packet Detection Algorithm

In [7], packet detection is performed over the Short Training Sequence (STS) of the IEEE 802.11a/n receiver whose packet structure is illustrated in Fig. 4. The STS is composed of ten repetitions of a 16 samples sequence.

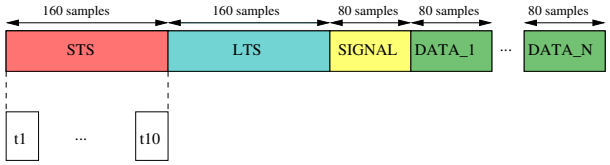

Fig. 4. IEEE 802.11a Packet Structure

The applied algorithm correlates L samples of the received sample stream $r[d]$ (d is the time index) with the subsequent L ones. For a single antenna receiver this can be expressed via the auto-correlation function

$$P_L[d] = \sum_{m=0}^{L-1} (r^*[d+m] \cdot r[d+m+L]). \quad (1)$$

To obtain a higher accuracy, L is set to half of the size of the STS which corresponds to a vector length of 80 samples. In a next step, the average energy of the received sample stream in the actual window is calculated as

$$R_L[d] = \sum_{m=0}^{L-1} |r[d+m+L]|^2. \quad (2)$$

In case

$$|P_L[d]|^2 > \frac{|R_L[d]|^2}{2} \quad (3)$$

the beginning of the packet is found. Otherwise the window over the incoming sample stream is shifted by a predetermined number of samples.
Extending this algorithm to the 2x2 MIMO case, (1) and (2) are performed for both receive chains obtaining two different results. The comparison is performed over the average results stated as

$$P_{L,\text{avg}}[d] = \frac{1}{2} \sum_{j=1}^{2} P_{jL}[d] \quad (4)$$

and

$$R_{L,\text{avg}}[d] = \frac{1}{2} \sum_{j=1}^{2} R_{jL}[d] \quad (5)$$

For the A-FEP, the set of instructions to be executed and the cycle count of each of them for the SISO case is shown in Table IV.

TABLE IV
A-FEP INSTRUCTIONS FOR AUTO-CORRELATION BASED PACKET DETECTION

	instructions	cycles
$P_L[d]$	agu_cfg (6x)	9
	vec_move	L/2 + 4
	agu_cfg (4x)	4
	vec_mult	L/2 + 4
	agu_cfg (2x)	2
	vec_sum	L/2 + 4
$R_L[d]$	agu_cfg (2x)	2
	vec_square_modulus	L/2 + 4
	agu_cfg (2x)	2
	vec_sum	L/2 + 4
	agu_cfg (2x)	2
	vec_square_modulus	L/2 + 4
$\|P_L[d]\|^2 > \frac{\|R_L[d]\|^2}{2}$	nop (7x)	7
	lw	5
	nop	1
	lw	5
	nop	1
	bgt	8

This results in a total amount of $6 \cdot \frac{L}{2} + 72$ cycles including the communication overhead and $6 \cdot \frac{L}{2} + 24$ cycles if only the data processing time is taken into account. When using the C-FEP instead the implementation can be simplified and the code can be written more dense as illustrated in Table V.

TABLE V
C-FEP OPERATIONS FOR AUTO-CORRELATION BASED PACKET DETECTION

	operations	cycles
$P_L[d]$	vector move	L/2 + 11
	vector multiplicaton + sum	L/2 + 12
$R_L[d]$	vector square_modulus + sum	L/2 + 12
$\|P_L[d]\|^2 > \frac{\|R_L[d]\|^2}{2}$	vector square_modulus + sum	L/2 + 11

Considering only the pure data processing time without the communication overhead, the resulting amount of cycles is $4 \cdot \frac{L}{2} + 46$.
To get an idea about how much time is required for control flow processes, different measurements have been carried out on the ExpressMIMO platform operating at a frequency of 100 MHz.

- The time required by the LEON3 to program one vector operation of the C-FEP takes 420 ns. The communication

TABLE VI
AUTO-CORRELATION BASED PACKET DETECTION COMPARISON

Solution	cycles (SISO)	cycles (MIMO)	execution time (SISO)	communication overhead (SISO)	execution time (MIMO)	communication overhead (MIMO)
ASPE A	296	650	2.96 μs	-	6.5 μs	-
A-FEP	264	572	2.64 μs	0.48 μs	5.72 μs	0.64 μs
C-FEP	312	465	3.12 μs	1.2 μs	4.65 μs	1.2 μs

overhead can be reduced when programming the subsequent vector processing during the processing of the previous one. In case the data processing time of this operation is larger than the 420 ns required for processing, the communication overhead has no effect in the overall processing time.

- Using polling, the time till LEON3 reacts on the end of a vector operation in the C-FEP is 436 ns.
- The time necessary for a value comparison including the memcopy of the values to compare from the C-FEP to LEON3 takes 350 ns.

To sum up, Table VI gives the resulting cycles counts and processing times for the A-FEP, the C-FEP and ASPE A for a frequency of 100 MHz. Based on these results it can be observed that for this algorithm, the A-FEP performs slightly better than ASPE A although the architecture of the latter is optimized for the processing of the IEEE 802.11a/n standard. Compared to the C-FEP, the A-FEP reduces the communication overhead significantly due to reduced pipeline delays and due to the GP instructions that reduce the communication overhead with the LEON3 processor. For that reason the A-FEP performs even better for the MIMO case. Considering only the data processing time, the C-FEP performs better as the sum is not an extra operation but included in the last stage of the C-FEP pipeline. In general it can be stated, that the development of algorithmic implementations is simplified using the A-FEP as no explicit synchronization between the processing unit and the LEON3 processor is needed.

B. Energy Based Packet Detection Algorithm

For the first example, we compared the A-FEP to two powerful ASPE A solutions that are able to execute the whole baseband processing of the applied IEEE 802.11a/n standard. In this example, the A-FEP will be compared to a specialized ASIP for synchronization and acquisition, the Sync-ASIP, that has recently been published in [6]. The applied packet detection algorithm is slightly different than the previously presented one. To get a first estimate of the probability that the beginning of the packet is detected, two energy values a_n and b_n are calculated over $L = 64$ elements and divided through each other. In case the result is beyond a certain threshold, the exact beginning of the packet is detected using auto-correlation functions.

The energy values can be computed as

$$a_n = \sum_{m=0}^{L-1} |r_{n-L}|^2 \tag{6}$$

and

$$b_n = \sum_{m=0}^{L-1} |r_{n+L}|^2 \tag{7}$$

while their relation is expressed as

$$m_n = \frac{a_n}{b_n} \tag{8}$$

For the A-FEP, the set of instructions to be executed and the cycle count of each of them is illustrated in Table VII.

TABLE VII
A-FEP INSTRUCTIONS FOR ENERGY BASED PACKET DETECTION

	instructions	cycles
a_n,b_n	agu_cfg (6x)	9
	vec_abs_square	L/2 + 4
	agu_cfg (2x)	2
	vec_abs_square	L/2 + 4
	agu_cfg (2x)	2
	vec_sum	L/2 + 4
m_n	agu_cfg (4x)	2
	vec_cwl	7
	agu_cfg (3x)	3
	vec_mult	4
	nop (7x)	7
	lw	5
	nop	1
	lw	5
	nop	1
	bgt	8

This results in a total amount of $3 \cdot \frac{L}{2} + 68$ cycles including the communication overhead and $3 \cdot \frac{L}{2} + 12$ cycles if only the data processing time is taken into account. When using the C-FEP instead the implementation can be simplified and the code can be written more dense as illustrated in Table VIII.

TABLE VIII
C-FEP OPERATIONS FOR ENERGY BASED PACKET DETECTION

	instructions	cycles
a_n,b_n	vec_abs_square + sum	L/2 + 12
	vec_abs_square + sum	L/2 + 12
m_n	vec_cwl	15
	vec_mult	11

Table IX gives the resulting cycles counts and processing times for the A-FEP, the C-FEP and Sync-ASIP for a frequency of 100 MHz. As expected, the typcial phenomenon can be observed that a weakly programmable ASIP specialized for a specific tasks performs far better than a flexible ASIP solution that is capable to perform a wide range of different operations. Comparing the A-FEP with the C-FEP solution it can be seen, that the A-FEP drastically reduces the communication overhead when processing short data sets.

TABLE IX
ENERGY BASED PACKET DETECTION COMPARISON

Solution	cycles	execution time	communication overhead
Sync-ASIP	31	0.31 μs	-
A-FEP	108	1.08 μs	0.56 μs
C-FEP	114	1.14 μs	2.29 μs

VI. CONCLUSION

In this paper we focused on the comparison of two new designs for the Front-End Processor for the OpenAirInterface ExpressMIMO platform with existing ASIP solutions. Comparing the A-FEP to the C-FEP, we have shown that the A-FEP performs better in terms of processing time due to the reduced communication overhead with the host system and due to reduced internal latencies. This makes this design the appropriate solution for standards with short data sets where the communication overhead plays an important role in the overall system performance. Although the A-FEP supports a wide range of different wireless communication standards, we have shown that it performs slightly better for a packet detection algorithm than a solution presented by ETH Zürich that is tailored to the processing of the IEEE 802.11a/n standard. On the other hand the performance is still worse when compared to an ASIP solution designed for synchronization and acquisition that commes with a reduced instruction-set and is therefore less flexible than the presented A-FEP. Future work includes the design analysis with regards to energy consumption and power dissipation as well as the final integration of the A-FEP on the ExpressMIMO platform.

REFERENCES

[1] K. van Berkel, F. Heinle, P. P. E. Meuwissen *et al.*, "Vector processing as an enabler for software-defined radio in handheld devices," *EURASIP J. Appl. Signal Process.*, vol. 2005, pp. 2613–2625, January 2005. [Online]. Available: http://dx.doi.org/10.1155/ASP.2005.2613

[2] *C6000 High Performance Multicore DSP*, Texas Instruments Inc., 2011.

[3] *MSC8156: Six Core High Performance DSP*, Freescale Semi. Inc., 2011.

[4] "http://www2.imec.be/be_en/press/imec-news/cobra.html."

[5] 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.

[6] M. A. Said, O. A. Nasr, and A. F. Shalash, "Embedded reconfigurable synchronization & acquisition ASIP for a multi-standard OFDM receiver," *Eurasip Journal on Embedded Systems*, vol. 2012, 2012, 10.1186/1687-3963-2012-2.

[7] S. Eberli, "Application-specifc processor for MIMO-OFDM software-defined radio," Ph.D. dissertation, ETH Zürich, 2009.

[8] A. Hoffmann, O. Schliebusch, A. Nohl *et al.*, "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.

[9] "Vsia consortium: http://www.vsi.org/." [Online]. Available: http://www.vsi.org/

[10] "http://www.gaisler.com/leonmain.html."

[11] "http://www.mutekh.org."

[12] A. W. R. L. O. Schliebusch, Gerd Ascheid and H. Meyr, "Application specific processors for flexible receivers," in *Proc. of National Symposium of Radio Science (URSI)*, Poznan (Poland), Apr 2005.

[13] M. Hamdy, O. A. Nasr, and A. F. Shalash, "ASIP design of a reconfigurable channel estimator for OFDM systems," in *Microelectronics (ICM), 2011 International Conference on*, Dec. 2011, pp. 1–5.

[14] T. Schuster, B. Bougard, P. Raghavan, R. Priewasser, D. Novo, L. Van der Perre, and F. Catthoor, "Design of a low power pre-synchronization ASIP for multimode SDR terminals," in *Proceedings of the 7th international conference on Embedded computer systems: architectures, modeling, and simulation*, ser. SAMOS'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 322–332. [Online]. Available: http://dl.acm.org/citation.cfm?id=1776200.1776244

[15] C. Bachmann, A. Genser, J. Hulzink, M. Berekovic, and C. Steger, "A low-power ASIP for IEEE 802.15.4a ultra-wideband impulse radio baseband processing," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, April 2009, pp. 1614–1619.

[16] 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.

[17] C. Schmidt-Knorreck, M. Ihmig, R. Knopp, and A. Herkersdorf, "Multi-standard processing using DAB and 802.11p on software defined radio platforms," in *WSR2012, 7th Karlsruhe Workshop on Software Radios*, to be published.

[18] T. Bösch, "Adaptive stream processor for network multimedia consumer electronic devices," Ph.D. dissertation, ETH Zürich, 2004.

[19] R. Ghaffar and R. Knopp, "Low complexity metrics for bicm siso and mimo systems," in *Vehicular Technology Conference (VTC 2010-Spring), 2010 IEEE 71st*, may 2010, pp. 1 –6.