IEEE 802.11p Receiver Design for Software Defined Radio Platforms

Carina Schmidt-Knorreck, Daniel Knorreck, Raymond Knopp
Mobile Communications Department
EURECOM
Sophia Antipolis, France
Email: carina.knorreck@eurecom.fr, daniel.knorreck@eurecom.fr, raymond.knopp@eurecom.fr

***Abstract*—Software Defined Radio platforms are a flexible and cost efficient solution to deal with the increasing number of today's wireless communication standards. One interesting use case can be found in the automotive industry where the IEEE 802.11p standard enables Car-to-Car and Car-to-Infrastructure communication. In the context of this paper we present a physical layer implementation of the 802.11p receiver for the OpenAirInterface ExpressMIMO platform. Our results show that a real-time processing is already possible for most of the modulation schemes when applying a centralized control flow. The results are further extended by recommendations of further design improvements and the derivation of general guidelines for further standard deployment on the platform.**

***Keywords*-flexible HW platform, IEEE 802.11p receiver, SDR**

I. Introduction

The number of today's wireless communication standards is growing rapidly, thus requiring the design of smaller and more sophisticated technologies that support a wide range of different standards. A flexible and cost efficient solution can be found in highly reconfigurable Software Defined Radio (SDR) platforms. These designs do not only cope with the challenging task of multimodal standard processing but are also easy adaptable to future technologies like LTE. This makes these platforms of high interest not only for academia but also for upcoming industrial solutions.

One interesting use case can be found in the automotive industry. Currently, experts focus on the design for Car-to-Car and Car-to-Infrastructure (C2X) communication to further reduce traffic jams and accidents. Imaginable scenarios are among others collision prevention, the monitoring of hazardous vehicles or accident warnings. For this purpose, several channels in Europe and the US have been reserved at the 5.9 GHz and at the 5.8 GHz band. A project of major importance in this domain is the German SimTD project were C2X communication is implemented on the physical (PHY) and on the MAC layer [1]. The most promising standard in this context is IEEE 802.11p [2], which is an enhancement of the well-known IEEE 802.11a standard [3]. It has its origin in 1999 when the US Federal Communication Commission allocated 75 MHz of the DSRC (Dedicated Short-Range Communication) spectrum exclusively for C2X communication. Compared to 802.11a, the bandwidth of 802.11p has been reduced from 20 MHz to 10 MHz. This results in OFDM symbols that are longer in the time domain and thus in systems with large delay spreads to avoid ISI (Inter Symbol Interference). For vehicular use cases where the channels are strong time-varying, this advantage is of major importance as it ensures a reliable reception of the transmitted signal. 802.11p has been in draft form till July 2010 and efficient transceiver design is still an open research topic. This task is quite challenging as the strong latency requirements require a fast baseband processing engine to ensure a high performance. Apart from that there is a high interest in the use of LTE for 802.11p type applications which also increases the requirements for related reconfigurable radio architectures.

The first contribution of our paper is the presentation of an efficient physical layer implementation of the 802.11p receiver for a flexible SDR platform. The chosen target technology is the OpenAirInterface ExpressMIMO platform [4] which has been developed by Eurecom and Télécom ParisTech. In contrast to other SDR platforms, its design is structured in independent DSP engines (Fig. 1) which allows an easy upgrade to future standards. Other advantages include the effective use of spectrum, mobility, increased network capacity, maintenance of cost reduction and faster development of new services. In addition, the platform comes with four A/D and four D/A converters and is thus supporting the multimodal processing of up to eight different channels that are not limited to the automotive context but to wireless communication standards in general. For the time being, the 802.11p receiver is the first prototype that has been developed for the ExpressMIMO platform, but the work on the integration of LTE and DAB has already been started. Therefore we enhanced the presented receiver results by a derivation of guidelines for further standard deployment and identified possible improvements of the DSP engines.

The outline of this paper is as follows: besides a short description of the latest baseband processing design of the ExpressMIMO platform in Section III we outline the basic development methodology of receiver mapping in Section IV. Section V and Section VI elaborate on the porting of the 802.11p receiver to the platform and performance figures are provided in Section VII. Finally, guidelines for further standard deployment are enhanced in Section IX.

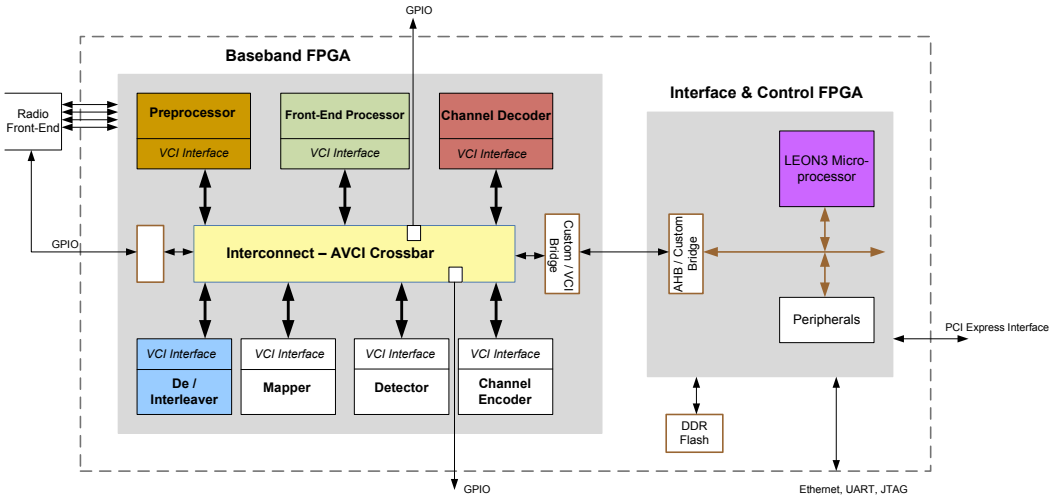


Figure 1. Baseband Architecture of the ExpressMIMO Platform

II. Related Work

Although the 802.11p transceiver has been in draft form till July 2010, there are already different industrial solutions available. One is the LinkBird-MX v3 unit produced by NEC [5] which embeds a Linux machine based on a 64 bits MIPS processor working at 266 MHz and which can be configured either for reception or for transmission. Besides, NXP and Cohda Wireless developed a flexible SDR implementation of WAVE (Wireless Access in Vehicular Environments) called MK3 [6]. It includes among others a GPS module and a CAN bus interface and is based on the NXP MARS platform which has been designed for the automotive context. Another solution is the combination of the WSU (Wireless Safety Unit) platform from DENSO and the Openwave Engine developed by BMW [7]. Besides the physical layer implementation of 802.11p this transceiver supports the required MAC protocols for US, Europe and Japan and includes CAN2.0, Ethernet and a 400 MHz power PC that can process one or two standards in parallel. Finally, [8] presented a transceiver based on GNU radio which has been combined with USRP2.

Like the latter, the ExpressMIMO platform is not limited to the automotive context but to wireless communication standards in general. In contrast to the mentioned solutions, different standards can be supported simultaneously be reusing the same HW architecture. The switch between two channels is performed at runtime and the DSPs are reconfigured depending on the standard in process. Besides, the modular baseband design allows an easy component replacement in case of standard upgrades or the design of new ones. Thus a complete redesign of the architecture is not necessary. Furthermore a C++ library is available that allows to emulate the transceiver in a pure SW environment.

III. System Integration

The design of the ExpressMIMO platform [4] potentially supports a wide range of different standards like GSM, UMTS or LTE as well as their multimodal processing. Its baseband design is split over several independent DSP engines that are controlled by a SPARC LEON3 processor from Gaisler Aeroflex [9]. The connection is established via a generic Advanced Virtual Component Interface (AVCI) crossbar [10]. The architecture of the DSP engines is based on a standardized DSP Shell (Fig. 2) which is composed of a Control Sub-System (CSS), a Processing Unit (PU) and a Memory Sub-System (MSS), where the two latter are custom defined. The CSS is common to all DSPs and is specialized through parameters. It contains among others a local 8 bit microcontroller (UC), a DMA engine, a set of control and status registers plus several arbiters and FIFOs for input-output requests and responses. Furthermore it acts as a gateway with the surrounding host system via two 64 bits wide AVCI compliant interfaces. In addition, a set of input and output interrupt lines is used for signaling and synchronization with the host system.

For the design of transceivers with short data sets like 802.11p, the following characteristics of the CSS are of major importance:

- DMA transfers can operate in parallel to the PUs
- the next command can already be prepared in the control registers while the PU is still busy
- the UC enables a distributed control flow on the platform

For the time being, the UC has not been integrated in the CSS yet. The current version of the receiver is thus orchestrated by a centralized control flow where the whole receiver program is running on LEON3.

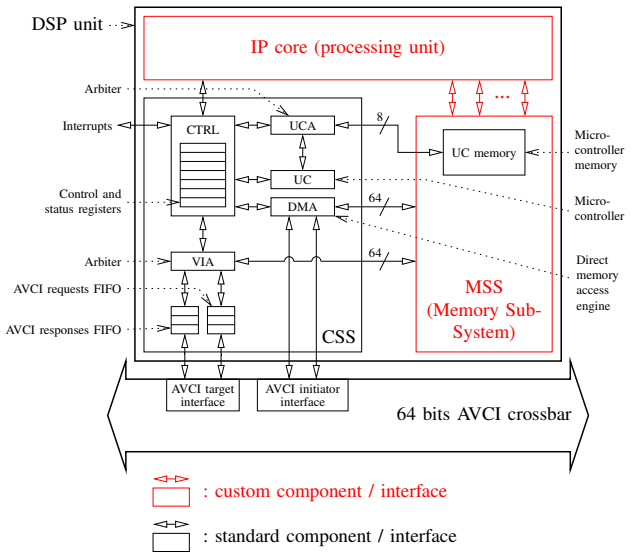


Figure 2. OpenAirInterface standardized DSP shell

To get first insights into the functional behavior of a transceiver on the platform, a C++ library is provided that allows to emulate all baseband processing functions in a bit accurate pure SW environment. The so-called library for ExpressMIMO baseband (*libembb*) is open-source and has already been applied in different European projects (e.g. SACRA [11]).
In the following we only focus on the DSP engines and memories required for the design of the 802.11p receiver:

- **VCI RAM**: The VCI RAM is a baseband memory that stores the permutations tables of the Deinterleaver as well as the incoming signal samples.
- **Front-End Processor (FEP)**: The FEP contains a vector processing unit and a DFT/IDFT unit and computes all operations at the air-interface level like packet synchronization, data detection, etc.
- **Channel Decoder (CHDEC)**: The CHDEC includes turbo decoders and a Viterbi decoder. For the design of the 802.11p receiver the latter one is used.
- **Deinterleaver (DEINTL)**: The DEINTL descrambles the bits to distribute burst errors over the whole received OFDM symbol.
- **(Preprocessor (PreProc))**: The PreProc connects the external radio front-end with the baseband processing engine and contains among others a Sample Rate Converter and real-time interrupt generation. As the work on this DSP is still ongoing, it is neglected in this paper. Including it in a future release will not change the presented performance results, as the PreProc DMA will stream the received samples into the VCI RAM without interrupting the presented receiver processing.

IV. Development Methodology

Receiver design typically starts with the development of purely functional models to analyze the algorithmic part of the receiver and to identify the required data flow arising from data dependencies. In models stemming from data flow networks, no further restrictions are imposed on the partial order of computations whereas sequential models such as C programs merely capture one possible sequence of computations. The entry point to our design flow are sequential, bit accurate C models that are built upon *libembb*. This emulation library is especially tailored to the ExpressMIMO platform and considerably eases the design of new receivers. *libembb* features the same functional primitives that are available on the real hardware and accessible through the same interfaces. Each DSP engine is represented by its own set of bit-accurate C++ functions which account for computations, return values of the processing unit and data transfer operations. Despite the sequential nature of the emulation environment this model can already be leveraged (1) to analyze the data flow between processing units, (2) to expose data dependencies, (3) to obtain first performance results and (4) to identify possible bottlenecks when processing several transceivers simultaneously. First results related to the 802.11p receiver have recently been published in [12].

To obtain cycle accurate performance figures of a set of concurrently executed DSPs, a common approach is the HW/SW co-simulation in discrete event simulators such as Modelsim. Execution parallelism embraces simultaneous computations on DSPs, data transfers of DMAs as well as the pre-programming of commands while the DSP is still busy. Using Modelsim is only appropriate for standards with short data sets as for instance only the initialization time of LEON3 for a standard like DAB is already in the order of 10^5 cycles.
Finally the receiver is validated on the real HW platform by (1) known snapshots and (2) real test signals received through the radio frequency part.

V. Receiver Algorithms and Design

The 802.11p packet structure (Fig 3) is composed of a constant part (Preamble and Signal Field) and the Data Field which consists of variable number of OFDM symbols, each with a size of 80 samples a 32 bit. The constant part has a length of 40 μs and comprises a *Short Training Symbol (STS)* for packet synchronization, a *Long Training Symbol (LTS)* for channel estimation and the *Signal Field* specifying how to decode the transmitted message. In contrast, the length of the *Data Field* is variable. The number of octets in the MPDU requested by the MAC layer varies between 1 and 4095, thus resulting in a Data Field length between 1 and 1366 OFDM symbols. Table I provides the modulation parameters for the applied 10 MHz channel spacing.

Table I
802.11P SPECIFICATION PARAMETERS (10 MHZ CHANNEL SPACING)

Parameter	Value
Number data subcarriers	48
number pilot subcarriers	4
subcarrier frequency spacing	10 MHz/64
Packet length	1 - 1366 DATA symbols
Modulation Schemes	BPSK, QPSK, 16/64-QAM
Code rates	1/2, 2/3, 3/4
Data rates	3, 4.5, 6, 9, 12, 18, 24, 27 Mb/s

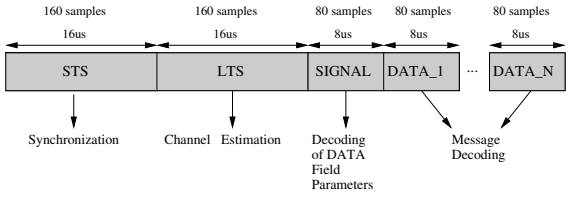


Figure 3. 802.11p Packet Structure

A. Packet Synchronization

OFDM systems are known to be extremely sensitive to timing and frequency synchronization errors. Possible techniques for packet synchronization are auto or cross correlation. Although the preamble of the 802.11p receiver is suitable to both, [13] has shown that the cross correlation performs slightly better. For our receiver we implemented an energy detector combined with an overlapping DFT-based correlator to synchronize over the known STS. The energy detection over the received signal r_x can be expressed as

$$E(X) = \sum_{i=0}^{255} |r_x[i]|^2 \quad (1)$$

In case the result is beyond a predefined threshold, the resulting frequency offset i_s can be computed as

$$i_s = argmax(|q_k|^2) \quad (2)$$

with

$$q_k = IDFT((DFT(r_x[i]) * DFT(STS_{ref})^*)) \quad (3)$$

The window size is 256 complex samples a 32 bit; the shift is performed over the size of one OFDM symbol. This algorithm is accomplished by the FEP only. The comparison to an energy threshold and thus the decision of whether the receiver proceeds to channel estimation is currently in the responsibility of LEON3 but may be delegated to the UC or to a microprocessor on the baseband side in a future version of the receiver.

B. Channel Estimation

For vehicular systems, the channel estimation is of major importance as the applied channels are strongly time-varying. Hence, correlated fading can be observed due to reflections coming from the same object. 802.11p defines two different pilot patters: block and comb pilots. The block pilots carried by the LTS embrace two identical OFDM symbols. In contrast, the four comb pilots are included in each of the Signal and Data Field OFDM symbols. Specific for 802.11p is, that their sign differs between the OFDM symbols. In [14] a detailed comparison of the different types of channel estimators has been carried out. Best performance is attained by the comb-type channel estimator which uses only the four comb pilots and interpolates intermediate values for the remaining carriers. As interpolation entails a considerable computation effort we decided to rely on the block-comb-type channel estimator. The latter computes an initial channel estimate as a function of the block pilots and a subsequent amplitude and phase correction is applied to the four comb pilots. The channel estimator reduces computational complexity while achieving an acceptable performance. The channel estimate based on the block pilots, $\hat{H}$, is calculated by the FEP:

$$\hat{H} = DFT(LTS_{received}) \odot DFT(LTS^*_{reference}) \quad (4)$$

C. Signal and Data Field Detection

For Signal and Data Field detection, three different DSP engines are required: FEP, Deinterleaver and Channel Decoder (Fig. 4). As the parameters describing the latter are to be extracted from the Signal Field, its decoding procedure has to be finished before starting the Data Field detection. These parameters comprise the number of OFDM symbols contained in the Data Field, the modulation scheme (BPSK, QPSK, 16-QAM, 64-QAM), the code rate (1/2, 3/4, 2/3), etc. The symbols of the Data Field permit a parallel execution of the DSPs. For the moment, the Viterbi decoder operates on the complete received message before the result is transferred to the MAC layer. In a future release tail-biting will be included so that this DSP can operate as shown in Fig. 5.
The Data Field can be modulated with four different modulation schemes, each of them exhibiting two different code rates. Operations to be performed in the FEP are channel compensation and data detection.

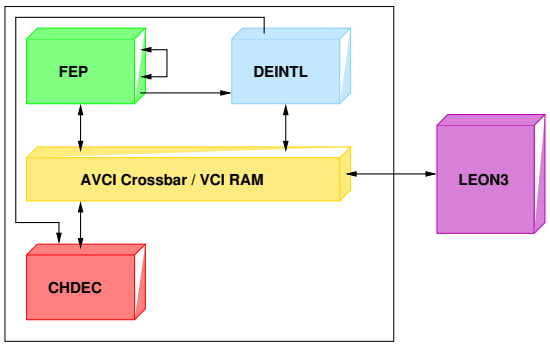


Figure 4. Data and Control Flow for Signal and Data Field

Afterwards the Deinterleaver reverts the two permutations applied by the transmitter to reduce the impact of burst errors. The resulting message after Viterbi decoding is descrambled and CRC checked in LEON3.

1) Channel Compensation: The channel compensation comprises the multiplication with the channel estimate as well as further corrections based on the comb pilots:

$$R_{d,n} = (A(H)e^{-j\hat{\Phi}_n} * R_x) \tag{5}$$

with

$$A(H)e^{-j\hat{\Phi}_n} = \frac{1}{4} \sum_{i=-21,-7,7,21} R^*_{p,i} R_{n,i} \tag{6}$$

and

$$\hat{A}(H) = \sum_{i=-21,-7,7,21} |\hat{H}_i|^2 \tag{7}$$

$R_{n,i}$ are the received and $R^*_{p,i}$ the known comb pilots. The correction of the unknown energy level term stated as $\hat{A}(H)$ is only necessary for 16/64-QAM demodulation where the calculation of the remaining bits is based on $R_{d,n}$.

2) Data Detection (Decoding): For BPSK only the real part of $R_{d,n}$ serves as input to the Deinterleaver. For QPSK, real and imaginary part of $R_{d,n}$ are both significant. For 16/64-QAM, the missing bits are calculated as a function of $R_{d,n}$ as stated in [15].

Data Detection equations to be performed by 16-QAM:

$$R_{d2,n} = \frac{2}{\sqrt{10}}(\hat{H} \odot \hat{H}^*) * \hat{A}(H) - abs(R_{d,n}) \tag{8}$$

Data Detection equations to be performed by 64-QAM:

$$R_{d2,n} = \frac{4}{\sqrt{42}}(\hat{H} \odot \hat{H}^*) * \hat{A}(H) - abs(R_{d,n}) \tag{9}$$

$$R_{d3,n} = \frac{2}{\sqrt{42}}(\hat{H} \odot \hat{H}^*) * \hat{A}(H) - abs(R_{d2,n}) \tag{10}$$

The result of the multiplication of the root term with $\hat{A}(H)$ and $\hat{H} \odot \hat{H}^*$ does not change during the whole Data Field detection and can be computed once after the Signal Field detection is finished.

VI. HW Enhancements

In the design flow, the code written for emulation could directly be compiled for the hardware platform. However, this code has been developed with a sequential execution in mind and currently does not (fully) exploit concurrency on the platform. As emulation is considered to be untimed for the moment, introducing concurrency would not be meaningful anyway. For this reason and to satisfy strong real time constraints, the emulation code is manually revised and optimized before being ported. In our case, improvements include the design of a flexible scheduler to execute the different DSP engines simultaneously and the generation of command words off-line before the receiver is started. As a lightweight operating system for LEON3, we opted for MutekH [16]. Originally, MutekH was designed to support the heterogeneity of nowadays processors and is increasingly used on multiprocessor platforms. For our receiver, MutekH does a good job in low level interrupt handling and initial hardware configurations.

A. Symbol Grouping

One possible optimization is the grouping of Data Field symbols for the FEP to operate on larger vectors. The maximum group size depends on the number of OFDM symbols supplied by the PreProc per acquisition cycle (time till a known number of samples is stored in the output FIFO of this DSP). The operations to be performed in the FEP per OFDM symbol comprise DFT, channel compensation and data detection. The maximum number of operations is saved for a group size of eight. Compared to the case when symbol per symbol is processed, 28 FEP commands can be saved.

B. Scheduling

As the 802.11p standard is extremely latency critical due to the short time available till the acknowledgement packet has to be sent, we renounced the overhead that the usage of POSIX threads would have entailed. We relied on a very simple thread model instead, where one thread is assigned to each concurrent entity on the platform (such as FEP, Deinterleaver, Channel Decoder). Thread management boils down to checking and updating two data items per thread: a pointer to the next chunk of code to be executed (referred to as position pointer) and a pointer to a memory location indicating whether the task is runnable or not (referred to as condition pointer). As the question whether a thread is blocked or runnable coincides with the question whether a DSP engine is running or ready, a condition pointer refers for instance to the dedicated busy flag of a DSP. The position pointer is updated as the algorithm control flow moves on to the next potentially blocking activity on the platform, be it a signal processing operation or a DMA transfer.
Fig. 5 illustrates the scheduling of the different DSP engines. Once, the result of a DSP is available, it is copied into the MSS of the subsequent DSP. During this time, both DSPs are busy. In contrast to the Signal Field, the structure of the Data Field allows to decode different Data symbols in parallel. The processing flow depends on the number of Data symbols available in the FEP MSS:

1) Assuming the case that the PreProc provides one Data symbol as soon as it is received, the scheduler has to wait till the next symbol is available before it can schedule the next task to the FEP (Fig. 5(a)).
2) When more Data symbols are available, the FEP can be started again once it finished the decoding proce-

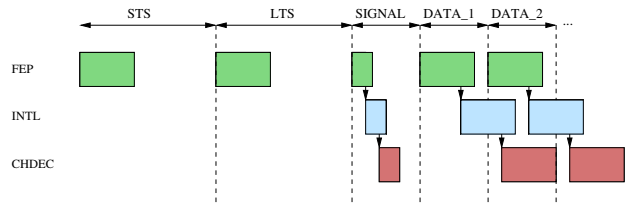
dure of one symbol (Fig. 5(b)). The time the different DSP engines are busy increases as the scheduler has to take care of different DSP tasks in parallel.

C. Command Preparation

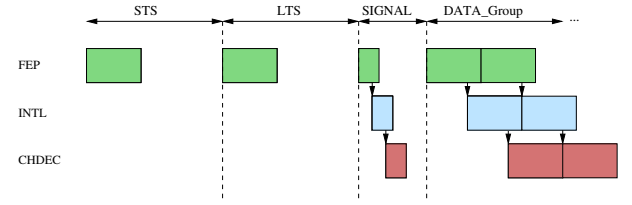
When applying a centralized control flow, the DSP engines are programmed by LEON3. Per parameter, the programming time takes around 425 ns. In case only one parameter has to be written, the communication overhead is negligible, but usual FEP commands, for instance, comprise at least 14 different parameters. This results in a total programming time of at least 6 μs per operation which is almost the duration of one OFDM symbol for 802.11p. An efficient solution can be found in the command preparation where each command is prepared and stored in a local memory before the receiver is started. The required programming time at runtime is significantly reduced to 70 ns per 32 bit command register. In case of the FEP, the programming time is decreased from around 6 μs to 420 ns. For the 802.11p receiver, most of the commands are static and can easily be prepared. In case a parameter is set dynamically at runtime, the performance gain depends on the number of parameters to be set and how they are distributed over the 32 bit command registers. For the 802.11p receiver, maximum one parameter is set dynamically per operation. The related timing overhead is negligible as only one assembly command is required for this operation.

D. Interrupt Handler

MutekH already comes with a very fast interrupt handler. The time measured on the platform from the moment an interrupt is raised till LEON3 reacts and continues with the next assembly command takes about 2.3 μs which is negligible for common standards.



(a) DSP Engine Scheduling when only one Data OFDM Symbol is available



(b) DSP Engine Scheduling when several Data OFDM Symbols are available

Figure 5. 802.11p DSP Processing and Scheduling

For 802.11p instead, this overhead causes a significant performance drop. An alternative is to poll the status registers of the DSP engines which decreases the measured reaction time to 436 ns.

VII. Results

The presented results have been obtained by HW/SW cosimulation in Modelsim. Prior to that, the whole receiver chain has been validated on the platform itself for a reference frequency of 100 MHz. The applied scheduler is based on a Round Robin policy. Detailed results of the Channel Decoder are not provided, as the tail-biting is not included in the current design. Instead the Viterbi decoder is invoked once at the end of the packet detection procedure.

A. Constant Part

Table II lists the DSP processing time and the communication overhead (control flow while the DSPs are not busy) for energy detection, packet synchronization, the calculation of the channel estimate and Signal Field detection. The communication overhead of the first two entries contains the programming time of the operations as well as the value comparison to a known threshold in LEON3 which takes 350 ns. The Signal Field requires three different DSP engines whose busy times including the DMA transfers are summarized in Table II. The additional processing time of the FEP is due to data processing by LEON3 required for the channel compensation algorithm. Furthermore, the estimation of the parameters needed by the Data Field detection (code rate,...) requires an additional processing time of 2.74 μs. The overall processing time of the constant part is about 23 μs plus the time required for an internal DMA transfer in the FEP MSS due to some internal restrictions. Not considering this variable transfer time that may vary between 13 μs and 26 μs, only 40% of the Signal Field processing time is related to the communication overhead. Adding the additional DMA transfer, this value will decrease as most of the supplementary processing time is related to the DMA processing.

B. Data Field

The FEP operations of each Data symbol comprise two main tasks: channel compensation and data detection. The average processing time as a function of the group size for the data detection is illustrated in Fig. 6.

Table II
Runtime Performance Results

	total proc. time	DSP proc time	com. overhead
Energy Detection	2.82 μs	1.51 μs	1.31 μs
Synchronization	8.01 μs	5.76 μs	2.25 μs
Calculation Channel Estimate	1.65 μs	0.82 μs	0.83 μs
Signal Field	11.64 μs	5.26 μs	6.38 μs

Table III
DSP BUSY TIMES SIGNAL FIELD INCLUDING DMA TRANSFERS BETWEEN THE DSPS

	DSP proc time	additional com. overhead
FEP	2.99 μs	2.14 μs
DEINTL	1.62 μs	-
CHDEC	1.27 μs	-

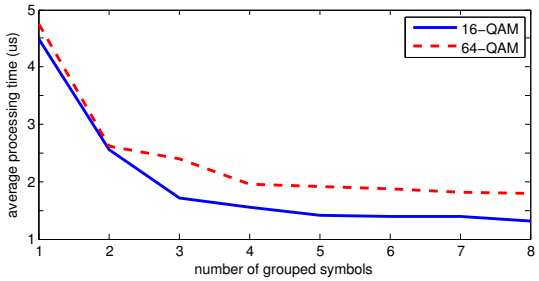


Figure 6. Average Processing Time Data Detection

BPSK and QPSK are not listed, as the result of the channel compensation can directly be used as input for the Deinterleaver. For 64-QAM, the computation of the remaining bits results in a higher processing time of the FEP than for 16-QAM as two more bits have to be calculated. Furthermore it can be observed, that for an increasing group size, a boundary value is reached which is equal to the pure processing time of the DSP plus some delays due to the scheduler.

Fig. 7 gives the average processing time of the Deinterleaver for 16-QAM with a code rate of 3/4 and illustrates the performance loss when applying a Round Robin scheduler. The dotted curve represents the ideal case where tasks of the FEP and Deinterleaver are scheduled instantaneously. The other curve shows the results when the scheduler is applied. Finally Fig. 8 and Fig. 9 illustrate the overall FEP and Deinterleaver processing time including the DMA transfers. As expected, BPSK and QPSK perform best as only $R_{d,n}$ has to be copied from the FEP to the Deinterleaver.

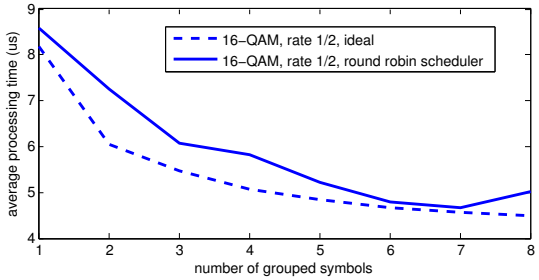


Figure 7. Round Robin Scheduler for 16-QAM

A centralized control flow is sufficient for BPSK, QPSK and 16-QAM as the processing time of the required DSP engines is below 8 μs which corresponds to the duration of one OFDM symbol. More detailed results concerning the DSP processing time and the communication overhead for a group size of eight are given in Table IV and Table V. For the FEP, the DSP processing time takes between 40.1% and 56.03%. Best performs 64-QAM as the additional operations related to the data detection can be programmed while the FEP is busy. Thus only the DSP processing time increases while the additional communication overhead remains almost unchanged. For the Deinterleaver, the processing time takes between 52.41% and 86.76%. The highest value is achieved for 64-QAM with a code rate of 3/4 as the Deinterleaver operates on the largest possible vector in this design with a size of 8*432 samples. It is strongly recommended to improve the Deinterleaver in a future version of the platform as the long processing time causes a significant overhead especially when processing over large vectors.

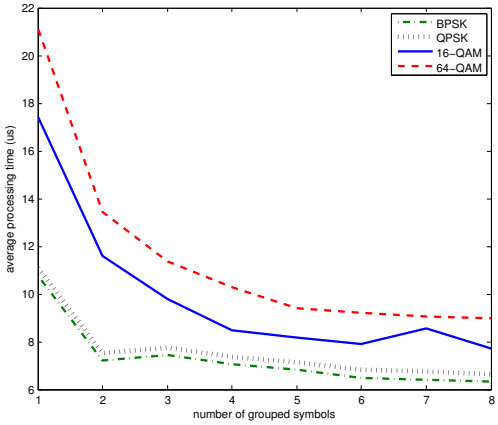


Figure 8. Average FEP Processing Time

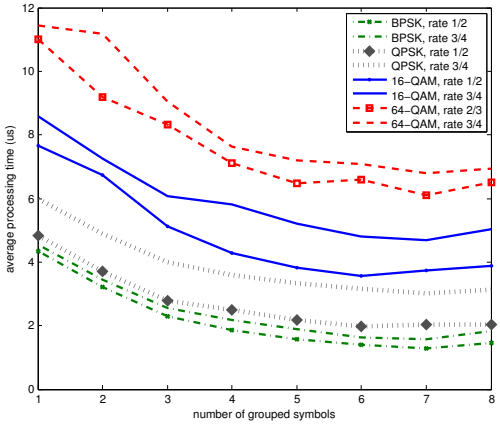


Figure 9. Average Deinterleaver Processing Time

Table IV
FEP BUSY TIMES DATA FIELD (GROUP SIZE OF EIGHT)

	total proc. time	DSP proc time	com. overhead
BPSK	50.62 μs	20.5 μs	30.12 μs
QPSK	51.26 μs	21.14 μs	30.12 μs
16-QAM	61.74 μs	30.67 μs	31.07 μs
64-QAM	71.75 μs	40.2 μs	31.55 μs

Table V
DEINTERLEAVER BUSY TIMES DATA FIELD (GROUP SIZE OF EIGHT)

	total proc. time	DSP proc time	com. overhead
BPSK (1/2)	11.6 μs	6.08 μs	5.52 μs
BPSK (3/4)	14.61 μs	8.48 μs	6.13 μs
QPSK (1/2)	16.36 μs	10.88 μs	5.48 μs
QPSK (3/4)	25.14 μs	15.68 μs	9.46 μs
16-QAM (1/2)	30.96 μs	22.64 μs	8.32 μs
16-QAM (3/4)	40.24 μs	32.24 μs	8 μs
64-QAM (2/3)	52.15 μs	43.36 μs	8.79 μs
64-QAM (3/4)	55.51 μs	48.16 μs	7.35 μs

VIII. GUIDELINES FOR FURTHER STANDARD DEVELOPMENT

Based on the previous results, different recommendations are made when currently processing standards with short data sets on the ExpressMIMO platform. To decrease the communication overhead, different solutions like invoking the local UC in the CSS or a microprocessor or sequencer on the baseband side are possible. Tasks like energy detection could be handled on the baseband side, while the main scheduling is still in the responsibility of LEON3. Furthermore, the communication overhead can be reduced by using polling instead of the interrupt handler and by preparing the commands in advance. For standards with long data sets like LTE, the real-time behavior is still guaranteed even if these recommendations are not considered. E.g. for a FEP vector operation over a size of 4096 samples, the processing time is about 20 μs while the programming time of the DSP stays at a negligible time of 360 ns in this case.

IX. CONCLUSIONS

In this paper, we have presented the latest design of the baseband processing engine of the ExpressMIMO platform as well as an efficient physical layer implementation of the 802.11p receiver. We have shown, that with a centralized control flow, real-time processing can already be achieved for BPSK, QPSK and 16-QAM. To guarantee the same for 64-QAM we suggest further improvements of the FEP, the Deinterleaver and the implementation of a distributed control flow. The latter could be realized by invoking the UCs or by a microprocessor or a sequencer on the baseband side. Based on our results we derived some basic guidelines to facilitate further standard deployment on the ExpressMIMO platform. Our future work includes a thorough comparison of a distributed and a centralized control flow as well as an analysis of the power consumption and the integration of the DAB and LTE transceivers.

ACKNOWLEDGMENT

This work is supported by the DeuFraKo Project PROTON / PLATA. Further thanks go to Renaud Pacalet and Alexandre Becoulet from the System on Chip Laboratory of Télécom ParisTech.

REFERENCES

[1] [Online]. Available: http://www.simtd.org

[2] *Specific requirements part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications amendment 7: Wireless access in vehicular environments*, IEEE Std. 802.11p/D9.0, july 2009.

[3] *IEEE 802.11a: Wireless LAN Medium Access Control and Physical Layer Specifications, High-Speed Physical Layer in the 5 GHz Band*, IEEE, september 1999.

[4] N.-u.-I. Muhammad, R. Rasheed, R. Pacalet, R. Knopp, and K. Khalfallah, "Flexible Baseband Architectures for Future Wireless Systems," in *Proc. DSD'08*, september 2008, pp. 39 –46.

[5] *LinkBird-MX version 3 datasheet*, NEC.

[6] E. Lambers, M. Klassen, A. Kippelaar *et al.*, *DSRC mobile WLAN component*, NXP Semiconductors.

[7] R. K. Schmidt, T. Leinmuller, and B. Boddeker, "V2X Kommunikation," february 2009.

[8] P. Fuxjaeger, A. Costantini, P. Castiglione *et al.*, "IEEE 802.11p Transmission using GNU Radio," in *Proc. WSR'10*, march 2010.

[9] [Online]. Available: http://www.gaisler.com/leonmain.html

[10] *VSI Alliance Virtual Component Interface Standard Version 2 (OCB 2 2.0)*.

[11] [Online]. Available: http://www.ict-sacra.eu

[12] C. Schmidt-Knorreck, M. Ihmig, R. Knopp, and A. Herkersdorf, "Multi-Standard Processing using DAB and 802.11p on Software Defined Radio Platforms," in *Proc. WSR'12*, 2012.

[13] A. Fort, J.-W. Weijers, V. Derudder, W. Eberle, and A. Bourdoux, "A performance and complexity comparison of auto-correlation and cross-correlation for OFDM burst synchronization," in *Proc. IEEE ICASSP'03*, vol. 2, april 2003, pp. II – 341–4.

[14] L. Bernado and, N. Czink, T. Zemen, and P. Belanovic, "Physical Layer Simulation Results for IEEE 802.11p Using Vehicular Non-Stationary Channel Model," in *Proc. IEEE ICC'10*, may 2010, pp. 1–5.

[15] R. Ghaffar and R. Knopp, "Low Complexity Metrics for BICM SISO and MIMO Systems," in *Proc. IEEE VTC'10*, may 2010, pp. 1 –6.

[16] [Online]. Available: http://www.mutekh.org