Multi-Standard Processing using DAB and 802.11p on Software Defined Radio Platforms

Carina Schmidt-Knorreck, Matthias Ihmig†, Raymond Knopp, Andreas Herkersdorf†
Mobile Communications Department, EURECOM
2229 route des Cretes
06904 Sophia-Antipolis Cedex, France
{Carina.Knorreck, Raymond.Knopp}@eurecom.fr

†Institute for Integrated Systems
Technische Universität München
Arcisstraße 21, 80333 München, Germany
{Matthias.Ihmig, Herkersdorf}@tum.de

***Abstract*—The processing of nowadays wireless communication standards requires the design of Software Defined Radio platforms to minimize area and costs. One interesting use case can be found in the automotive industry where the multimodal processing of the DAB and 802.11p standards are of major interest. In this paper, we analyze their resource consumption on the OpenAir-Interface platform and show that both standards fulfill their realtime requirements. Based on a detailed runtime analysis, we will also give guidelines for a sophisticated scheduling algorithm.**

I. Introduction

Today, the increasing number of wireless standards in mobile communications requires the design of high performance technologies supporting nowadays and future standards. Therefore, highly configurable Software Defined Radio (SDR) platforms are needed, which are able to cope with the challenging task of multimodal standard processing. For industry, SDR platforms are of high interest as only one hardware architecture is designed that acts as a transceiver for a wide range of different standards. Thus, hardware cost and integrational complexity are reduced compared to former technologies.

One interesting use case of such platforms can be found in the automotive industry. The combination of local environment data and traffic data enables not only new safety functions like informing the driver about critical situations within its local environment, but also keeps him up to date with regional traffic information. Standards of interest are Car-to-Car/Infrastructure communication (IEEE 802.11p [1]) and TPEG information (Digital Audio Broadcasting, ETSI-DAB [2]). A major project in this domain is the German SimTD project [3] where Car-to-X communication is implemented not only on the PHY but also on the MAC layer. Furthermore this project includes a real test case using cars, traffic lights, etc. In contrast to our approach, the transceivers for DAB and 802.11p are separated, thus facilitating the implementation of the standards as no sophisticated resource management is needed.

The necessary task scheduling on SDR platforms is still an open research topic. For its design, it is of main importance to have performance key figures at hand. Their interpretation in order to derive first guidelines for the design of an adequate scheduling algorithm is the main target of this paper.

We have chosen the OpenAirInterface platform being developed by Eurecom and Telecom ParisTech [4] as our target platform. Its baseband design is split over independent DSP engines than can be processed simultaneously. Other advantages include the effective use of spectrum, mobility, increased network capacity, maintenance of cost reduction, faster deployment of new standards and faster development of new services. More details about the architecture of this platform are provided in Section II-A. An efficient scheduling of the two standards according to their specification leads to a different resource consumption on the platform. The derivation of the performance figures is detailed in Section II-B, the results of this runtime distribution analysis are explored in Section III. Considering only the processing time, both standards fulfill their realtime requirements. In Section IV, we additionally consider a control flow and we show the differences between a global control of the baseband processing by a LEON3 microprocessor from Gaisler and a distributed control by additionally using microcontrollers. Finally, guidelines for a sophisticated scheduling algorithm are provided in Section III-C.

II. OpenAirInterface Platform

A. Architecture

The OpenAirInterface Platform being developed by Eurecom and Telecom ParisTech [4] is a prototype SDR architecture designed to support a wide range of different standards like WLAN, WiMAX, GSM, UMTS and also LTE in the near future. Its baseband design (Figure 1) is split over several independent DSP engines (Front-End Processor, Channel Decoder, ...) being controlled by a SPARC LEON3 processor from Gaisler - Aeroflex [5]. This design allows an easy component replacement in case updates are required. The connection between the DSP engines is established via a generic Advanced Virtual Component Interface (AVCI) crossbar ([6], [7]). In this paper we consider only the relevant DSP engines needed to process the DAB and 802.11p standard, which are the Preprocessor responsible for signal allocation, the Front-End Processor including a DFT and a vector processing unit, the Channel Decoder and the Deinterleaver.

The architecture of the DSP engines is based on the standardized IP shell shown in Figure 2. It is composed of

- a **Memory Sub-System (MSS)** depending on the functionality of the DSP engines. It contains the input / output data space and local memories
- a **Processing Unit (PU)** containing the functionality of the DSP engine
- a **Control Sub-System (CSS)** that is common to all DSP engines. It is specialized through parameters and contains a local micro-controller (UC), a DMA engine (DMA), a set of control and status registers plus several arbiters and FIFOs for input-output requests and responses. The CSS can be seen as a gateway responsible for the communication with the entire system. It has two 64-bits wide interfaces: a slave interface used for read and write requests to the MSS and to the internal control and status registers plus a master interface used for DMA transfers. Both interfaces are AVCI compliant. Furthermore several input and output interrupt lines are used for signaling and synchronization with the host system.

The whole baseband processing can be emulated using the C++ *library for Express-MIMO baseband* (*libembb*). Although the work on this library is not finished yet, it is already applied in different European projects. The source code will soon become open source. Thanks to this library, the receiver code can easily be tested in a pure SW environment or can be used to access the different DSP engines on the platform. In the first

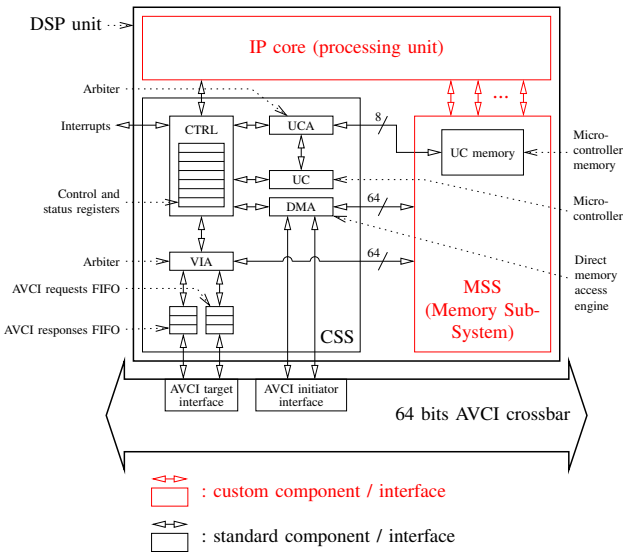

Fig. 2. Standardized IP Shell

case, each DSP engine is represented by its own set of bit-accurate C++ functions. These functions include the representation of the PU as well as error messages and memcopy access functions to represent the DMA transfers. Porting the application to the target platform, the code skeleton remains unchanged, thus resulting in an optimized development process.

B. Performance Estimation

A detailed performance evaluation and runtime distribution analysis of the two standards is necessary to determine whether the wireless standards can actually run in realtime on the platform and to have a basis for further discussions on an adequate scheduling. For this analysis, several factors have to be considered:

- **Execution time on the LEON3 processor**, including control-flow, rarely-used trigonometrical functions as well as time for configuration and start of the baseband DSP engines. The estimation on the development host would require a software simulator, such as TSIM for LEON3 [8].
- **Execution time of the DSP engines** which is easier to estimate as their internal structure is already known.
- **Communication time** to transfer data between the LEON3 and the baseband processing. Time can be estimated by the amount of transferred data between the LEON3 DDR and the baseband memory space at a certain bus speed.

Common receiver design typically starts with the development of purely functional C models to analyze

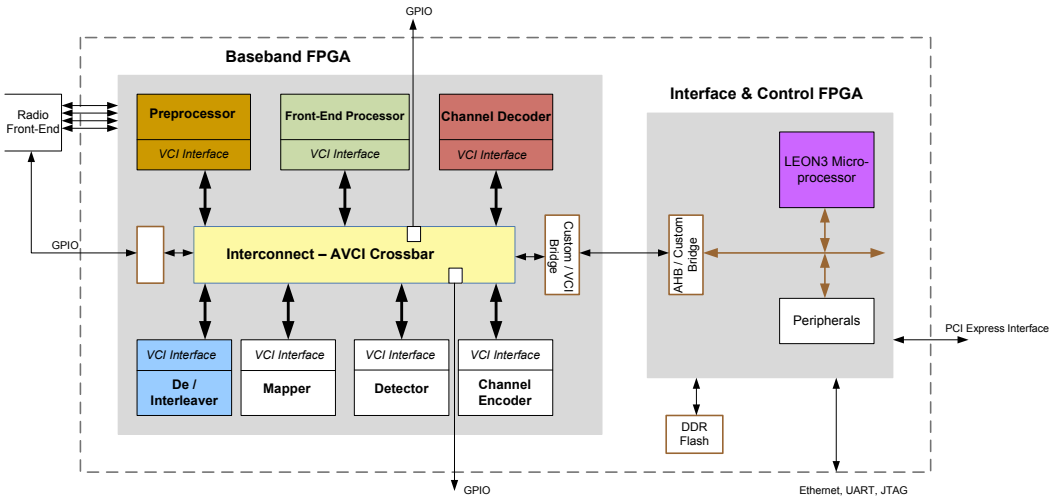

Fig. 1. Baseband Architecture of ExpressMIMO platform

the untimed algorithmic part of the transceiver. Therefore both receivers have been first modeled using *libembb*. For the DAB receiver, we adapted the solution presented in [9] to our needs. A major advantage of this approach is that the basic algorithm relying on *libembb* primitives does not have to be reworked to run on the target platform.

To obtain first estimates of the receiver performance, a common approach is the cycle accurate HW/SW co-simulation, e.g. in Modelsim. This solution is appropriate for transceivers with a short packet or frame length. But for standards like DAB, it is too time consuming as results are obtained on a cycle accurate level. For instance only the initialization of the LEON3 takes already a significant amount of time in the order of 1e5 cycles. Therefore we annotated *libembb* with timing information by implementing a flexible cycle counter. The necessary equations to calculate the processing time of the different DSP engines are shown in Table I. The vector length n is given in units of bytes. Not listed is the Preprocessor which provides the received samples in real-time and who raises an interrupt after k received samples (802.11p: $k = 640$ corresponding to 64us). Although the LEON3 processing time is disregarded in this model, it provides the user with efficient means to assess the transceiver's real-time behavior on the OpenAirInterface platform.

TABLE I
CYCLE COUNTS FOR DIFFERENT VECTOR LENGHTS n FOR EXPRESSMIMO PERFORMANCE MODEL

Operation	Cycles
FEP-FFT	$n \cdot log_4(size)/8 + 16$
FEP-Vector Operations	$n/2 + 16$
(De)Interleaver	$n + 16$
Channel Decoder (Viterbi)	$n/2 + 16$
Memcopy between DSP Engines	$n/8 + 24$
Memcopy between LEON3 and DSP	$n/4 + 24$

III. RUNTIME DISTRIBUTION ANALYSIS

Although all wireless systems must be real-time capable, the actual requirements depend on the type of the wireless system:

- **Latency critical:** For bidirectional packet-based communication systems, such as WLAN IEEE 802.11, the acknowledgement packet must be sent within a specified time. Thus the decoding time for the incoming packet from baseband up to higher layers is bound by a strict latency requirement. Processing time depends on packet size, modulation scheme and interarrival time.
- **Non-latency critical:** Unidirectional or frame-based wireless systems have no latency requirements, but all incoming data must be processed at the frame-rate, i.e. all processors and DSP engines used for decoding must stay under 100% load to fulfill realtime-requirements. Broadcast systems provide a continuous data stream with deterministic timing and processing

In the following, we investigate in a a detailed runtime analysis of the 802.11p and the DAB standard as well as the comparison of the obtained results. For all estimations, the DSP engines and the DMAs are assumed to run at 150 MHz.

A. IEEE 802.11p

The work on the IEEE 802.11p standard [1] was launched in November 2004 to make the IEEE 802.11a standard suitable for vehicular communication. The resulting standard includes the communication between two different vehicles (car-to-car - C2C - communication) as well as the communication of a vehicle with its surrounding area (car-to-infrastructure - C2I - communication). It operates in the licensed ITS band of 5.9 GHz (5.85-5.925 GHz). Table II shows the modulation parameters for a 10 MHz channel spacing.

TABLE II
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	BPSK, QPSK, 16/64-QAM
Coding rates	1/2, 2/3, 3/4
Data rates	3, 4.5, 6, 9, 12, 18, 24, 27 Mb/s

The 802.11p packet structure can be seen in Figure 3. It is composed of a constant part and a variable part. The constant part has a length of 40us and is composed of a *Short Training Symbol (STS)* used for packet synchronization, a *Long Training Symbol (LTS)* used for Channel Estimation and the *SIGNAL Field* containing information about how to decode the transmitted message. In contrast, the length of the *DATA Field* is variable and depends on the length of the transmitted message. The number of octets in the MPDU requested by the MAC layer varies between 1 and 4095, thus resulting in a DATA Field length of 1 and 1366 Data symbols, each with a length of 80 complex samples.

Packet synchronization (SYNC) and the calculation of the channel estimate (CE) are done on the FEP. To detect the beginning of the packet, a sliding FFT window over 256 complex samples is used. For the Signal field detection (SIG) three different DSP engines (FEP, Deinterleaver and Channel Decoder) are needed. The required tasks cannot be processed in parallel to the Data field detection as the necessary parameters for the

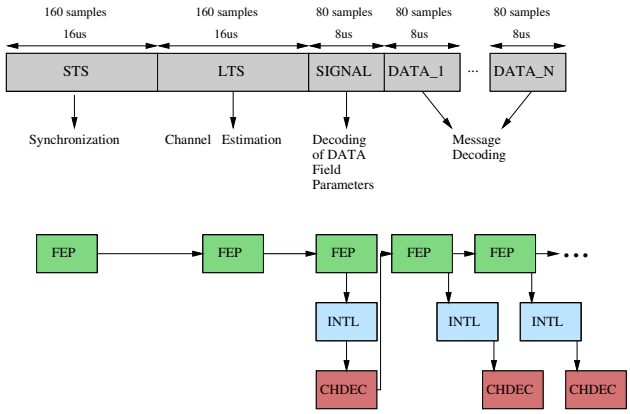

Fig. 3. 802.11p Packet Structure and mapping of the receiver on the OpenAirInterface Platform

latter are coded in the Signal field. These parameters are among others the number of data symbols in the Data field, the modulation scheme and the coding rate. The execution and memcopy runtime for the constant part is shown in table III. In contrast to the FEP, the Deinterleaver and Channel Decoder operations represent only one task.

TABLE III
TASK RUNTIME FOR DSP ENGINES AND MEMCOPY / CONSTANT PART

Task	DSP	Memcopy	total
SYNC	12920 ns	-	12920 ns
CE	653 ns	333 ns	986 ns
SIG (FEP)	1053 ns	855 ns	1900 ns
SIG (DEINTL)	667 ns	403 ns	1000 ns
SIG (CHDEC)	267 ns	365 ns	632 ns

The variable part of the 802.11p packet comprises the Data symbols which can be modulated in eight different ways: BPSK (rate 1/2, 3/4), QPSK (rate 1/2, 3/4) 16-QAM (rate 1/2, 3/4) and 64-QAM (rate 2/3, 3/4). Like for the Signal field, FEP, Deinterleaver and Channel Decoder are used to decode each of the symbols. Due to the tail-biting option of the Channel decoder, the three DSP engines can be processed in parallel, thus reducing the overall processing time. In contrast to the constant part, the processing time depends now on the modulation scheme and on the coding rate. Table IV gives an overview of the DSP processing time per symbol including possible memcopy operations. In case of QAM modulation, the FEP has to calculate some values first before the Data field detection starts. These times are denoted as *FEP (init)*. In contrast to the FEP, the Deinterleaver and Channel Decoder operations represent

only one task while the FEP operations include channel compensation as well as data detection.

TABLE IV
TASK RUNTIME FOR DSP ENGINES INCLUDING MEMCOPY PER DATA SYMBOL

DSP	**BPSK rate 1/2**	**BPSK rate3/4**	**QPSK rate 1/2**	**QPSK rate 3/4**
FEP	2282 ns	2282 ns	2282 ns	2282 ns
DEINTL	1070 ns	1290 ns	1554 ns	1994 ns
CHDEC	632 ns	735 ns	837 ns	1042 ns
DSP	**16-QAM rate 1/2**	**16-QAM rate 3/4**	**64-QAM rate 2/3**	**64-QAM rate 3/4**
FEP (init)	1219 ns	1219 ns	1499 ns	1499 ns
FEP	3166 ns	3166 ns	4006 ns	4006 ns
DEINTL	2520 ns	3200 ns	4367 ns	4807 ns
CHDEC	1247 ns	1657 ns	2067 ns	2312 ns

Figure 4 gives an overview of the overall runtime distribution for the minimum and maximum 64-QAM decoded packet (coding rate is 3/4) under the assumption that the packet starts with the first sample of the test signal. For the Signal field detection, it is the FEP who is consuming most of the processing time. In case of BPSK processing, this trend would remain unchanged even for the DATA field detection. But when processing a 64-QAM modulated packet, the Deinterleaver generates up to six times more samples compared to a BPSK packet of similar length.

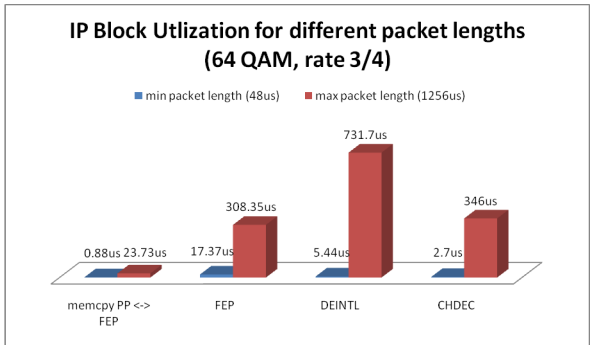

Fig. 4. Runtime Distribution - 802.11p

B. Digital Audio Broadcast

The ETSI Digital Audio Broadcast (DAB) [2] was launched in 1995 by the BBC as replacement for the traditional FM radio and has been on-air in Europe since that time. While recent adaptions, such as DAB+, update the audio codec from MPEG1 to MPEG4 HE AAC, the underlying physical layer processing mechanisms remain the same. The DAB standard defines 4 transmission modes, mode-I being the most commonly used in Band III (174-204 MHz). Table V shows the corresponding modulation parameters.

TABLE V
ETSI DAB SPECIFICATION: TRANSMISSION MODE I

Parameter	Value
Frame duration	96ms, 76 Symbols
Symbol duration (total, useful, guard)	1.246ms, 1ms, 246us
Null Symbol duration	1297ms
Transmission bandwidth	1.536 MHz
OFDM type	2048-FFT, 1535 used
Modulation	D-QPSK
Brutto Bitrate	2.4 Mbps

The DAB receiver chain consists of several tasks, as shown in Figure 5. Functionality is briefly described below; more details on each block can be found in [10].

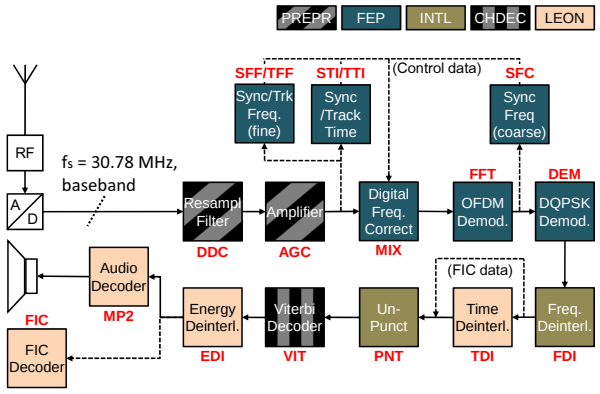

Fig. 5. Task model and mapping of DAB receiver on ExpressMIMO platform

The DAB receiver has two modes: *Synchronize* and *Receive*. After a frequency change, initial time synchronization (STI), coarse (SFC) and fine frequency estimation (SFF) find and lock to the DAB broadcast signal. When synchronized, tracking time (TTI) and fine frequency offset (TFF) are used to track changes, e.g. due to variations in mobile environments. After downsampling the complex baseband to 2.048 MSps and tracking, the estimated fine frequency is corrected (MIX) digitally, as the analog mixer has limited granularity (10 Hz are required for a DAB OFDM symbol). Energy calculation for STI and TTI and dot-product calculation are done on the FEP as well as the demodulation for OFDM (FFT) and for differential QPSK (DEM). Frequency deinterleaving (FDI) of each symbol uses the hardware interleaver. Instead, the time deinterleaver

(TDI) runs on LEON3, as the depth of 384ms spans several frames and thus requires more memory than available on the baseband FPGA - unpuncturing (PNT) is again possible on the hardware interleaver. Viterbi decoding (VIT) is again done in hardware on the channel decoder engine. Energy dispersal (EDI), audio decoding (MP2) and extraction of additional information from the Fast Information Channel are also done on the LEON3.

While all tasks require runtime on the LEON3 processor, most of them use the baseband DSP engines for acceleration. Available memory on the DSP engines is limited to 4x4 kSamples which corresponds to the maximum size of the FEP memory. For this reason the context is saved and restored in the LEON3 DDR to decode a DAB frame. This requires intensive memcopy operations between LEON3 DDR and the DSP block memories and generates an overhead of 55%-85%.

The total runtime required for execution and memcopy for each function is shown in table VI and Figure 6.

TABLE VI
TASK RUNTIME FOR DSP ENGINES AND MEMCOPY FOR 1SEC AUDIO DATA

Task	DSP	Memcopy	total
SFF	0.1ms	0.48ms	0.58ms
STI	0.33ms	3.26ms	3.59ms
SFC	0.3ms	1.09ms	1.39ms
TTI	1.06ms	4.80ms	5.86ms
TFF	1.60ms	10.7ms	12.3ms
FFT	2.54ms	7.43ms	9.97ms
DEM	2.95ms	17.6ms	20.6ms
FDI	4.37ms	19.1ms	23.5ms
PNT	6.08ms	4.65ms	10.7ms
VIT	1.52ms	4.65ms	6.17ms

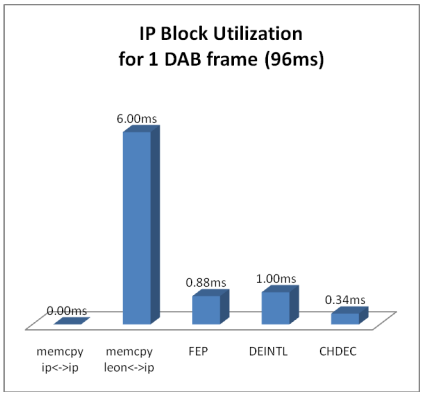

Fig. 6. Runtime Distribution - DAB

C. Discussion

Most of the tasks to be performed by both receivers run on the FEP. These tasks comprise the packet / timing synchronization, channel compensation and data detection. However, Deinterleaver and Channel Decoder are activated only from time to time but run over larger vectors if compared to the FEP. Figure 7 and 8 illustrate the runtime distribution of the FEP tasks. For 802.11p, the execution time varies from 0.12us to 0.96us while they are between 0.14us to 13us for the DAB.

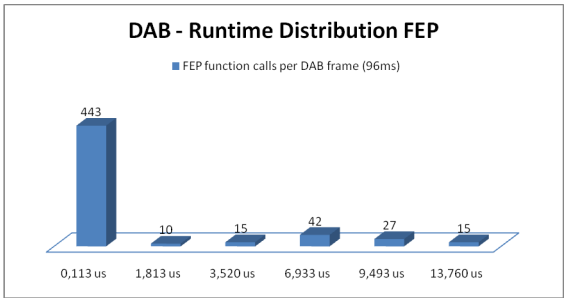

Fig. 7. Runtime Distribution (FEP) - DAB

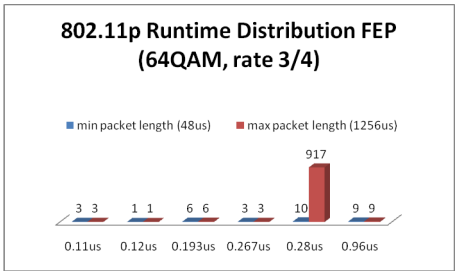

Fig. 8. Runtime Distribution (FEP) - 802.11p

Considering only the processing time, the latency requirements are fulfilled if only one standard is processed. But what if the two should be processed simultaneously? A scheduler being able to cope with these different standards has to be dynamical and should be based on an earliest deadline first policy. Challenges in its design are

- the limited local memory resulting in a context storage when switching between the standards
- the fact that micro operations are scheduled while the deadline is related to the macro packet level
- the consideration, that the starting time of the next 802.11p packet is not known in advance

Due to the strong latency requirements of the 802.11p standard, it is recommended to group vector operations to reduce the time consuming memcopy transfers and the communication overhead. Furthermore DAB operations have to be split to guarantee the real-time behavior of the 802.11p. To avoid unnecessary context savings, this has to happen at runtime only if the FEP is required by the 802.11p. As seen in Figure 7, the longest operations are vector operations, which can be split easily. This is not the case of the FFT taking 9.49us.

IV. Application control

In the previous sections, the results were only based on the pure processing time of the DSP engines and on the time required for memcopy transfers. Not considered was the application control on the platform which can be established in two different ways. Considering a **global control flow**, the LEON3 is responsible for the signal processing task distribution and the synchronization of the different DSP engines. The latter can be enabled by writing the corresponding control registers being part of the CSS. Once a scheduled task is performed, the DSP engine raises an interrupt. The time between this event and its treatment on LEON3 is about 12.5us and is thus not an appropriate solution for standards with strong latency requirements. An alternative could be the polling of the status registers. Now, the time till the event is treated on LEON3 takes less than 2us.

A better solution is the consideration of a **distributed control flow**. In this case the local microcontrollers inside the standardized IP shells are used besides the LEON3, thus allowing the processing of complex tasks inside the DSPs without interaction with the LEON3. Taking the example of a data detector (16-QAM) required by the 802.11p standard, the control flow time can be decreased by more than 100us, under the assumption that 8-bit microcontrollers are used.

The choice of the appropriate deployment depends on whether the performance gain justifies the increase in synchronization complexity. However, a distributed control flow is more suitable for standards with a short packet or frame length to fulfill the realtime requirements.

A detailed analysis of the different control flows while processing the two standards on the OpenAirInterface platform is part of our ongoing work.

V. Conclusion and Future Work

In the content of this paper we focused on the runtime distribution of the 802.11p and the DAB standard on the OpenAirInterface platform. Our results have shown that considering only the processing time on the DSP engines, the latency requirements of both standards are fulfilled. Furthermore we investigated in the right choice of an appropriate control flow and gave guidelines for a sophisticated scheduling algorithm whose implementation is part of our ongoing work. Besides, further publications about the receiver processing on the real HW of the OpenAirInterface platform are planned.

Acknowledgments

This work is supported by BMW Group Research and Technology, the German BMWi (Federal Ministry of Economics and Technology) within the DeuFraKo project PROTON (Programmable telematics on-board unit) / PLATA (PLAteforme Télématique multistandard pour l'Automobile). Furthermore, the research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement SACRA n 2490. Thanks go also to Lothar Stolz and his C-based reference implementation of DAB.

References

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

[2] ETSI 300 401, "Radio broadcast systems; Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers," June 2006.

[3] "http://www.simtd.org," .

[4] 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.

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

[6] "VSIA consortium: http://www.vsi.org," .

[7] "VSI Alliance Virtual Component Interface Standard Version 2 (OCB 2 2.0)," .

[8] Aeroflex Gaisler, "TSIM ERC32/LEON simulator," Jan. 2012.

[9] Lothar Stolz, "An Optimized Software-Defined Digital Audio Broadcasting (DAB) Receiver for x86 Platforms," in *Proceedings of the 7th Karlsruhe Workshop on Software Radios*, 2012.

[10] M. Ihmig, Nicolas Alt, and A. Herkersdorf, "Implementation and Fine-grain partitioning of a DAB SDR receiver on an FPGA-DSP platform," in *Proceedings of the 6th Karlsruhe Workshop on Software Radios*, March 3/4 2010.