Reconfigurable Turbo/Viterbi Channel Decoder in the Coarse-Grained Montium Architecture

Gerard K. Rauwerda, Gerard J.M. Smit, Casper R.W. van Benthem
University of Twente, department EEMCS
P.O. Box 217, 7500 AE Enschede, The Netherlands
g.k.rauwerda@utwente.nl

Paul M. Heysters
Recore Systems
P.O. Box 217, 7500 AE Enschede, The Netherlands
paul.heysters@recoresystems.com

***Abstract*— Mobile wireless communication systems become multi-mode systems. These future mobile systems employ multiple wireless communication standards, which are different by means of algorithms that are used to implement the baseband processing and the channel decoding. Efficient implementation of multiple wireless standards in mobile terminals requires energy-efficient and flexible hardware. We propose to implement both the baseband processing and channel decoding in a heterogeneous reconfigurable system-on-chip. The system-on-chip contains many processing elements of different granularities, which includes our coarse-grained reconfigurable MONTIUM architecture. We already showed the feasibility to implement the baseband processing of OFDM and WCDMA based communication systems in the MONTIUM. In this paper we implemented two kinds of channel decoders in the same MONTIUM architecture: Viterbi and Turbo decoding.**

***Index Terms*— Software Defined Radio, System-on-Chip, Channel decoding, MONTIUM, Coarse-grained reconfigurable hardware**

I. INTRODUCTION

A complete hardware based radio system has limited utility since parameters for each of the functional modules are fixed. A radio system built using Software Defined Radio (SDR) technology extends the utility of the system to a wide range of applications using different link-layer protocols and modulation/demodulation techniques. SDR provides an efficient and relatively inexpensive solution to the design of multi-mode, multi-band, multi-functional wireless devices that can be enhanced using software upgrades only.

SDR-enabled devices (i.e. mobile terminals) are designed such that the characteristics of the device can be reprogrammed. So, the same hardware can be adapted to perform different functions at different points in time. This means that in the same SDR framework, different communication standards can be supported, i.e. multi-standard communications. Another advantage of the SDR template is the possibility to implement real adaptive systems. Traditional algorithms in wireless communications are rather static. The recent emergence of new applications that require sophisticated adaptive, dynamical algorithms based on signal and channel statistics to achieve optimum performance has drawn renewed attention to run-time reconfigurability.

Implementation of SDR requires a flexible hardware architecture. Since baseband processing in the wireless receiver is computationally intensive, the processing power of the terminal's hardware architecture has to satisfy these demands. Moreover, wireless terminals are battery-powered, which emphasizes the importance of energy-efficiency in wireless receivers.

Figures presented in [1], [2] show that error decoding in a wireless receiver is as computationally intensive as baseband processing. This means that both baseband processing and error correction algorithms for multi-mode communication systems can benefit from efficient implementations in heterogeneous reconfigurable hardware.

This paper presents the implementation of an adaptable channel decoder, which is capable of both Viterbi and Turbo decoding. The decoder has been implemented in the coarse-grained reconfigurable MONTIUM architecture. Besides the channel decoding algorithms, we already presented implementations of the baseband processing for receivers based on OFDM and WCDMA techniques [3], [4].

In Section II we point to relevant research. Basic channel coding theory is presented in Section III. Section IV describes briefly the proposed heterogeneous reconfigurable system-on-chip, including the MONTIUM. The implementation of an adaptive Turbo and Viterbi decoder is done in Section V. Utilization of the generic channel decoder is discussed in Section VI. Conclusions are presented in Section VII.

II. BACKGROUND

Research in the Software Defined Radio (SDR) context focuses strongly on multi-mode communication systems, as for example in the *MuMoR* [5] project. Furthermore, energy-efficiency becomes ever more important, since mobile terminals are battery-operated. The *EASY* project [6], for instance, aims at developing a power/cost-efficient System-on-Chip (SoC) implementation, which handles the baseband processing

of wireless LAN systems. Their heterogeneous SoC consists of an embedded FPGA and ARM processor.

Conventional reconfigurable processors are bit-level reconfigurable and are far from energy-efficient. *Pleiades* at the University of California, Berkeley, is exploring reconfiguration of coarse-grained application-specific building blocks with an emphasis on low-power computations [7]. Furthermore, PACT [8] proposes an *extreme processor platform (XPP)* based on clusters of coarse-grained processing array elements. Silicon Hive [9] offers coarse-grained reconfigurable block accelerators (e.g. *Avispa* and *Moustique*) and stream accelerators (e.g. *Bresca*) for high performance and low-power applications.

Coarse-grained building blocks are considered in heterogeneous reconfigurable SoCs. These SoCs consist of mixed-grained reconfigurable blocks, suitable for implementation of multi-mode communication systems. The *Smart chipS for Smart Surroundings* [10] and *AMDREL* [11] projects anticipate these objectives.

In [2] it has been reported that 50% or more of the computational complexity in the wireless receiver is due to error coding algorithms, like Viterbi or Turbo decoding. This complexity counts for about 60% of the energy consumption in the digital processing part of a typical wireless receiver [1]. Consequently, power reduction should be achieved in the error decoding part of the receiver. New physically oriented design methodologies are proposed in ASIC design for Viterbi [12] and Turbo decoders [13]. Power reduction by exploiting variations in system characteristics due to changing noise conditions are the focus in [14], [15]. The latter can be achieved using dynamic reconfiguration in reconfigurable hardware.

In [16] a channel decoder chip was proposed that is compliant with the 3^{rd} Generation wireless standard. However, this chip is a dedicated solution for the 3^{rd} Generation UMTS system, which cannot be used in other wireless communication standards. In [17] another reconfigurable architecture for Viterbi and Turbo decoding was reported. That architecture, *Viturbo*, can be configured to decode convolutionally coded data and Turbo coded data.

III. Channel coding

Shannon described in [18] that it is possible to reliably send information over a communication channel with a transmission rate, which is limited by the Shannon capacity (or Shannon limit). The Shannon limit is the absolute limit, where no improvement on the bit error rate (BER) can be made without increasing the energy of the bits. Shannon described his theorem, however he did not give a solution to reliably send information. Many error-correction code schemes have been proposed until now. Among others, convolutional codes are widely used in communication systems as error-correction codes. These error-correction codes enable reliable communication of information over a noisy, distorted communication channel by adding redundant information [19].

A convolutional code is generated by passing the data through a finite state shift register. The contents of the shift register (i.e. the state of the shift register) determines the output code. So, the encoding of information can be represented with a state machine. As an example, Figure 1 shows a convolutional encoder and its state machine. The *rate* of this encoder is $R = 1/2$. Three different bit-values – the current value and two old values – are used in this encoder to create the code. Hence, the *constraint length* of the encoder is $K = 3$. The encoder in Figure 1 is called Non Systematic Convolutional (NSC) because the input data bits are not directly connected to the output code bits.

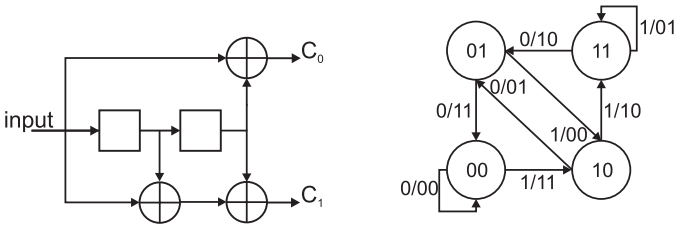

Fig. 1. Convolutional encoder (left) and its state machine (right).

Convolutional code decoding algorithms are used to estimate the encoded input information, using a method that results in the minimum possible number of errors. In [20] Viterbi originally described his maximum-likelihood sequence estimation algorithm, commonly known as the Viterbi algorithm. The job of the decoder is to estimate the path through the trellis that was followed by the encoder. A trellis diagram simply shows the progression of the state of the encoder for different symbol times. Figure 2 depicts part of the trellis for the NSC encoder of Figure 1. The trellis shows for each time instant all possibles states of the encoder, and all possible state transitions.

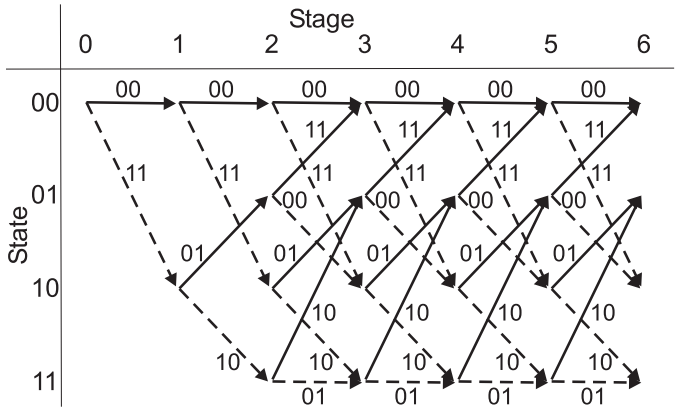

Fig. 2. Trellis diagram with 4 states.

Turbo codes, a new family of convolutional codes were proposed in [21], [22]. These codes are built using concatenation of two Recursive Systematic Convolutional (RSC) codes and their performance is close to the Shannon limit. The recursive codes have a feedback loop in the convolutional encoder, which causes the state of the encoder to be dependent on the state as well as the input. Figure 3 depicts the basic building

blocks of the turbo encoder. The Turbo encoder consists of two RSC encoders and an interleaver. One encoder receives the input data bits directly, while the second encoder receives the input data bits in a shuffled way. The shuffling is performed by the interleaver. The interleaver operates on input blocks of a fixed length, which is specified by dedicated standards. For UMTS the interleaver block size is specified between 40 and 5114 bits [23].

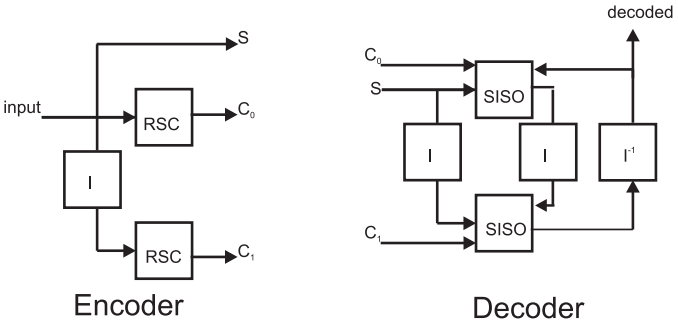

Fig. 3. Turbo encoder (left) and decoder (right).

The decoding of Turbo codes is performed in an iterative way. The decoder consists of a de-interleaver and two decoder blocks. These decoders are mostly referred to as Soft-Input-Soft-Output (SISO) decoders. Each SISO decoder estimates the Log-Likelihood Ratio (LLR), which denotes the logarithm of the probability that a '1' is transmitted divided by the probability that a '0' is transmitted, based on its input signals. These input signals of the SISO decoder are the parity input and systematic input, which is also called the *intrinsic information*, and the feedback information derived by the previous SISO decoder, which is called the *extrinsic information*. Each iteration of Turbo decoding will add extra information to make a better decision on the decoded bit stream.

IV. Heterogeneous reconfigurable hardware

Implementation of SDR requires a flexible hardware architecture. Traditional SDR approaches are implemented on *homogeneous* flexible architectures, like General Purpose Processor (GPP) or Digital Signal Processor (DSP) [24]. Since baseband processing in the wireless receiver is computationally intensive, the terminal's hardware architecture has to be very powerful. Moreover, as wireless terminals are battery-powered, the importance of energy-efficiency of the hardware architecture is emphasized.

A common objection to the traditional homogeneous flexible architecture is its relative energy in-efficiency. However, *heterogeneous* reconfigurable hardware, consisting of processing elements with *different* granularities, is designed with these constraints – flexibility, performance and energy-efficiency – in mind.

The idea of heterogeneous processing elements is that one can match the granularity of the algorithms with the granularity of the hardware. Four processor types can be distinguished: *general purpose*, *fine-grained* (e.g. FPGA), *coarse-grained* (e.g. Montium [25], [26]) and *dedicated* (e.g. ASIC). Figure 4 depicts a heterogeneous reconfigurable hardware template, consisting of processing elements with different granularities. Matching the granularity of the reconfigurable hardware with the algorithm provides flexibility at the right level:

- *general purpose*
  The general purpose processor is the most flexible hardware architecture. It can be programmed to perform almost any algorithm. General purpose processors are well suited for control-oriented functions. Due to the large overhead in control, these processors are not very energy-efficient.
- *fine-grained*
  Fine-grained reconfigurable devices are bit-level programmable. Because of the configurability at bit-level, the configuration overhead is large. Fine-grained reconfigurable devices are perfectly suited for prototyping and to implement encryption algorithms.
- *coarse-grained*
  Coarse-grained reconfigurable devices are flexible at word-level. Multipliers, adders, etc. are hardwired in these devices. Because only coarse functional blocks have to be configured, the configuration overhead is small. These architectures are more suited for data-oriented functions, like algorithms performed in the DSP domain.

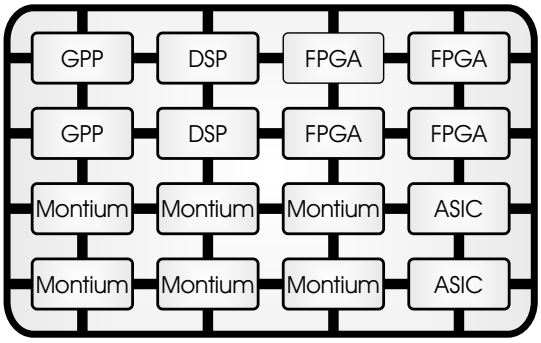

Fig. 4. The Chameleon SoC template.

The proposed tiled System-on-Chip (SoC) template, Chameleon [26], consists of the above mentioned processor types (Figure 4). The tiles are interconnected by a Network-on-Chip (NoC). Both SoC and NoC are dynamically reconfigurable, which means that the programs (running on the reconfigurable tiles) as well as the communication channels are defined at run-time. The coarse-grained reconfigurable tiles in the Chameleon SoC template are Montium tile processors [25], as depicted in Figure 5.

A. The Montium reconfigurable architecture

The Montium is a coarse-grained reconfigurable processor and targets the 16-bit digital signal processing (DSP) algorithm domain. At first glance the Montium architecture bears a resemblance to a VLIW processor. However, the control

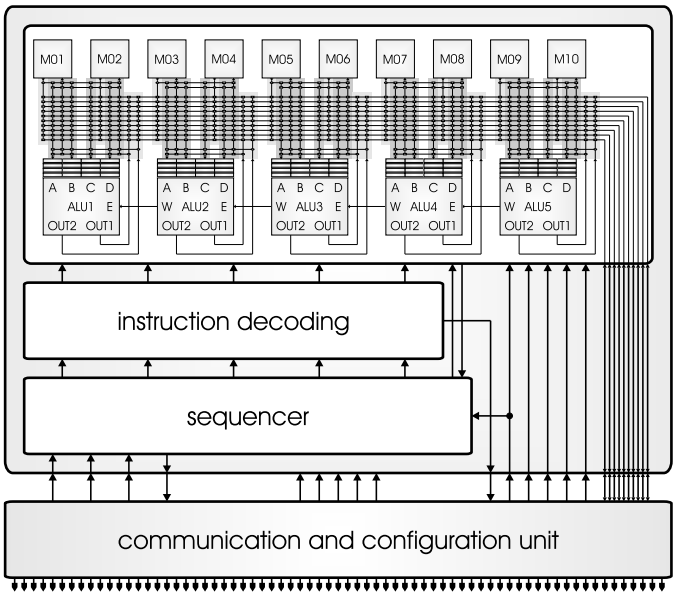

Fig. 5. The MONTIUM tile processor.

structure of the MONTIUM is very different. For (energy-) efficiency it is imperative to minimize the control overhead. This can be accomplished by statically scheduling instructions as much as possible at compile time.

The lower part of Figure 5 shows the Communication and Configuration Unit (CCU) and the upper part shows the reconfigurable Tile Processor (TP). The CCU implements the interface for off-tile communication. The definition of the off-tile interface depends on the NoC technology that is used in the SoC. The CCU enables the MONTIUM to run in 'streaming' as well as in 'block' mode [27].

The TP is the computing part that can be configured to implement a particular algorithm. Figure 5 reveals that the hardware organization of the tile processor is very regular. The five identical ALUs (ALU1 $\cdots$ ALU5) in a tile can exploit spatial concurrency to enhance performance. This parallelism demands a very high memory bandwidth, which is obtained by having 10 local memories (M01 $\cdots$ M10) in parallel. The small local memories are also motivated by the locality of reference principle. A reconfigurable Address Generation Unit (AGU) accompanies each memory. It is also possible to use the memory as a look-up table for complicated functions that cannot be calculated using an ALU, such as sine or division (with one constant). The data path has a width of 16-bits and the ALUs support both signed integer and signed fixed-point arithmetic. The ALU input registers provide an even more local level of storage. Locality of reference is one of the guiding principles applied to obtain energy-efficiency in the MONTIUM. A vertical segment that contains one ALU together with its associated input register files, a part of the interconnect and two local memories is called a Processing Part (PP). The five Processing Parts together are called the Processing Part Array (PPA). A relatively simple sequencer controls the entire PPA. The sequencer selects configurable PPA instructions that are stored in the decoders of Figure 5.

Development tools are essential for quick implementation of applications in the MONTIUM architecture. The intention of the MONTIUM development tools is to start with a high-level description of an application (in C/C++ or Matlab) and translate this description to a MONTIUM configuration. Until now, majority of the applications are implemented in the MONTIUM using a MONTIUM compiler. The configurations of the MONTIUM can manually be created/changed by entering the exact configuration data bits in the configuration editor. The mapping of applications at design time has been automated by using the MONTIUM configuration description language. This language enables structured application development using MONTIUM and precompiler instructions.

V. ADAPTABLE DECODER IMPLEMENTATION

The Turbo and convolutional code schemes have been adopted by many wireless communication standards. In the 3^{rd} Generation UMTS system both coding schemes are employed. Turbo coding has been used for data channels and convolutional coding for voice channels [23].

In [16] a channel decoder chip was proposed that is compliant with the 3^{rd} Generation wireless standard. However, this chip is a dedicated solution for the 3^{rd} Generation UMTS system, which cannot be used in other wireless communication standards. We implement both the Turbo and the Viterbi decoder in our coarse-grained reconfigurable MONTIUM architecture, which can also be used to implement the baseband processing [3], [4]. The flexible coarse-grained reconfigurable MONTIUM enables the implementation of flexible baseband processing and flexible channel decoding in multi-mode communication systems. The unified channel decoder chip in [16] has been implemented in older CMOS technology, therefore we cannot fairly compare the implementation with the MONTIUM implementation. In [17] another reconfigurable architecture for Viterbi and Turbo decoding was reported. That architecture, *Viturbo*, can be configured to decode convolutionally coded data and Turbo coded data. The *Viturbo* decoder is only aimed for channel decoding, whereas the MONTIUM architecture is more flexible and suitable for baseband processing as well. The area of the MONTIUM is slightly larger than the *Viturbo* decoder (~ 250 kGates vs. ~ 200 kGates).

The configurations for the MONTIUM of the Viterbi and the Turbo decoder are implemented using the MONTIUM configuration description language.

A. Turbo decoder

The SISO decoders in the Turbo decoder can be implemented using several algorithms [28]. We implemented the Max-Log-MAP (MLM) algorithm in the MONTIUM. This algorithm has a regular optimized structure and achieves near-optimal Bit Error Rate (BER).

The MLM algorithm consists of three processing phases: *forward recursion*, *backward recursion* and *LLR calculation*. The information from the forward and backward recursion are used to estimate the LLR information. Because the LLR

calculation can be done while the backward recursion is performed, the backward estimations do not need to be stored in memory. However, all the forward estimations have to be stored in memory. Hence, in order to be compliant with the 3^{rd} Generation UMTS standard, at most 5114×8 forward estimates have to be stored for full block length. The required memory to store the forward estimates can be reduced by applying the *sliding window* approach [29]. This approach divides the full block into smaller blocks, *windows*, on which the algorithm is applied. The number of forward estimates that needs to be stored is now equal to the window length.

In the forward recursion a branch metric is computed for each possible state transition in the trellis. The branch metric, $\gamma_{ij}[k]$, of the transition from state i to state j at time k is

$$\gamma_{ij}[k] = S_{ij}[k] \cdot L[k-1] + Q_{ij}[k] \cdot P[k-1] \qquad (1)$$

$S_{ij}[k]$ is the data bit and $Q_{ij}[k]$ is the parity bit associated with a transition from state i to state j at time k. $L[k-1]$ denotes the intrinsic input, and $P[k-1]$ gives the parity input. (2) and (3) show the formulas to calculate the forward state and backward state metrics, respectively. During forward recursion the trellis is read in forward direction, while during backward recursion the trellis is read in the opposite direction. Where $A_x[k]$ denotes the forward state metric of state x at time k, and $B_x[k]$ identifies the backward state metric of state x at time k.

$$A_j[k] = \max(A_{i_1}[k-1] + \gamma_{i_1 j}, A_{i_2}[k-1] + \gamma_{i_2 j}) \qquad (2)$$

$$B_i[k] = \max(B_{j_1}[k+1] + \gamma_{i j_1}, B_{j_2}[k+1] + \gamma_{i j_2}) \qquad (3)$$

$$\begin{aligned} \lambda[k] &= \max_{i,j} \left(A_i[k] + B_j[k+1] + \gamma_{ij}[k]\right)\Big|_{S_{ij}[k]=1} \\ &- \max_{i,j} \left(A_i[k] + B_j[k+1] + \gamma_{ij}[k]\right)\Big|_{S_{ij}[k]=0} \end{aligned} \qquad (4)$$

The LLR calculation is performed immediately after the backward recursion. The LLR calculation is done according to (4). The operations to be performed for the calculation of (2), (3) and (4) can be done in a very regular way. So, these operations are suitable to be performed in the MONTIUM architecture.

1) Throughput: The Turbo codes used in the UMTS communication system have constraint length $k = 4$, which means that 8 states exist in the trellis of the Turbo code. Hence, 8 forward recursion and 8 backward recursion have to be performed consecutively for each time instant of the trellis. The parallelism of ALUs and memories in the MONTIUM provides resources to calculate the forward and backward recursion in 4 clock cycles for one time instant of the trellis.

The intermediate forward state metrics are stored in the local memories of the MONTIUM. Immediately after the calculation of the backward state metrics, the LLR is calculated. The LLR calculation in the MONTIUM is performed in 4 clock cycles per time instant of the trellis. Consequently, 8 clock cycles are required to apply the MLM algorithm for one time instant of the trellis.

Due to the limited size of the local memories in the MONTIUM, the maximum block sizes that can be used for Turbo decoding is also limited. However, the limited memory in the MONTIUM is not problematic when using the sliding window approach.

The maximum channel data rate of the UMTS communication system is 1.92 Mbps, which means the the maximum Turbo frame of 5114 bits has to be processed in 2.66 ms. In order to perform Turbo decoding with 10 iterations, the MONTIUM should run at a speed of 110 MHz. The inner and outer decoder are applied during one Turbo decoding iteration without considering the interleaving process.

2) MONTIUM *configuration:* The total configuration size of the MONTIUM Max-Log-MAP implementation is 1262 bytes. This configuration can be loaded in the MONTIUM's configuration memory in 6.36 μs when the configuration clock frequency is 100 MHz.

B. Viterbi decoder

We implemented a fully flexible Viterbi decoder in the MONTIUM, based on a hybrid Register Exchange/Traceback approach [30]. The *rate*, R, as well as the *constraint length*, k, and the *decision depth*, d, of the decoder can be adapted within certain boundaries. These boundaries depend on the size of the local memories inside the MONTIUM. The results in this paper are based on the DAB system, $R = 1/4$ and $k = 7$ with a *decision depth*, $d = 50$.

1) Throughput: The implementation of the Viterbi algorithm in the MONTIUM results in a decoder that processes one stage of the trellis in 42 clock cycles for the DAB case. The data processing of one stage consists of branch metric calculation and path metric updating. Whenever 10 stages of the trellis have been processed, the Viterbi decoder decides on the decoded bit sequences of the foregoing stages.

In the implemented DAB decoder, always 10 bits are generated during the survivor decision phase. On average 47 clock cycles are required to decode one output bit. The output rate of the Viterbi decoder in the MONTIUM is 2.1 Mbit/s using a clock frequency of 100 MHz. This is sufficient for DAB, which requires an output rate of 1.8 Mbit/s. We also included in Table I the number of clock cycles that are needed for the Viterbi decoder in the general case with *constraint length*, k, *rate*, R, and *decision depth*, d.

2) MONTIUM *configuration:* The total configuration size of the MONTIUM Viterbi implementation is 1356 bytes. This configuration can be loaded in the MONTIUM's configuration memory in 6.78 μs when the configuration clock frequency is 100 MHz.

Once the MONTIUM is configured as Viterbi decoder, only partial reconfiguration has to be performed in order to adjust the *constraint length*, *decision depth* or *rate*. Especially the *decision depth* depends heavily on the conditions of the

TABLE I

CYCLE-COUNT OF THE VITERBI IMPLEMENTATION IN THE MONTIUM.

	branch metric	path metric	survivor decision
DAB decoder $d = 50, k = 7, R = 1/4$	8	34	50
general decoder d, k, R	$\frac{1}{2}2^{\frac{1}{R}}$	$2^{k-2}+2$	$2^{k-2}+2+\frac{d}{16-k-1}\times 3$

wireless channel. Thus, adjusting the *decision depth* can be typically performed on run-time via dynamic reconfiguration.

VI. ADAPTIVE DECODER SYSTEM

Trends in wireless communication systems show that the underlying DSP algorithms become more adaptive. Three levels of adaptivity in the digital signal processing of wireless receivers can be identified: *standards level*, *algorithm-selection level* and *algorithm-parameter level*.

The support of multiple wireless communication standards introduces a first level of adaptivity in the wireless terminal because the terminal can switch between wireless communication standards. For example when packet data transport is performed over UMTS and a WLAN hotspot becomes available the terminal can switch from UMTS to a WLAN standard. This is referred to as *standards level* adaptivity. Standards level adaptivity has an impact on the digital signal processing in the wireless terminal because the wireless communication standard defines the DSP functions that have to be performed to implement the standard.

Although a wireless communication standard usually defines the DSP functions, which have to be performed to implement the standard, it usually does not define the algorithms that have to be used to implement these functions. The communication system can therefore select an algorithm from a set of algorithms that implement the same DSP function. Therefore, this second level of adaptivity is referred to as *algorithm-selection level* adaptivity.

For a specific algorithm, there are also opportunities for adaptivity by changing parameters of the algorithm. This third level of adaptivity is called *algorithm-parameter level* adaptivity.

The implemented Viterbi and Turbo decoder can be used to make an adaptive channel decoder. The three levels of adaptivity also apply for the adaptive channel decoder:

1) *Standards level*: Different channel decoders are used for different wireless communication standards. In the Digital Audio Broadcasting (DAB) standard or UMTS standard convolutional coding is employed. But in the UMTS standard Turbo coding is also applied. Consequently, the usage of the Viterbi or Turbo decoder depends on the requested communication standard.
2) *Algorithm-selection level*: Within the UMTS communication system both convolutional and Turbo coding schemes are defined. For data channels the Turbo coding schemes are used and for voice channels convolutional coding schemes are applied. This means that in an UMTS mobile terminal both the Viterbi and the Turbo decoder need to be available. Another kind of algorithm-selection adaptivity can be the usage of different algorithms for the SISO decoders in the Turbo decoder. The kind of algorithm that will be used, depends heavily on the conditions of the wireless environment. Power reduction by exploiting variations in system characteristics due to changing noise conditions are also the focus in [14], [15].
3) *Algorithm-parameter level*: The characteristics of the decoders can be varied by changing implementation parameters. The implemented Viterbi decoder on the MONTIUM is universal in the sense that different *rates*, *constraint lengths* and *decision depths* can be handled. The Turbo decoder can also handle different *rates* and *constraint lengths*. Moreover, the number of Turbo *iterations* can be adjusted. The parameters *rate* and *constraint length* are dictated by the different wireless communication standards. The *decision depth* and number of Turbo *iterations* are however depending on the quality of the wireless environment, where the communication system is applied. Reducing the decision depth and the number of iterations yield enormous energy reduction.

On one hand an adaptive channel decoder has to be utilized because of multiple standards that are implemented in mobile terminals. Hardware costs in the mobile terminal are reduced by reusing reconfigurable hardware to implement multiple decoding algorithms. An adaptive channel decoder also yields large energy savings on the other hand. Thus, the decoding algorithm that performs best in a given environment with minimal effort can be applied by reconfiguring the hardware to the desired functionality.

VII. CONCLUSION

Future mobile wireless communication systems will become multi-mode communication systems. To realize these multi-mode systems efficiently, the hardware should be reused for different functionality. We showed the possibility to map the Turbo decoder and the Viterbi decoder on the same coarse-grained reconfigurable hardware architecture.

We showed that the coarse-grained reconfigurable MONTIUM is suitable for implementation of channel decoding algorithms. We presented the implementation results of the

Viterbi algorithm as well as the Max-Log-MAP algorithm, used in Turbo decoding, in the same MONTIUM.

The configuration overhead of the decoders is relatively small, as the configuration files of both decoders are small. Hence, changing the functionality of the channel decoder from Turbo to Viterbi, or vice versa, can be done dynamically, because of the short reconfiguration times. Depending on the desired communication standard, one can configure the hardware in the mobile terminal to implement the right channel decoder. The reconfiguration time of the Viterbi or Turbo decoder implementation is less than 7 μs.

The results, presented in this paper, show that the MONTIUM architecture is flexible. In earlier publications [3], [4] the authors already showed that the MONTIUM is suitable to efficiently implement the baseband processing of different wireless communication standards. For future work we will analyse the impact of the flexibility on the power consumption of the adaptable channel decoder.

ACKNOWLEDGEMENT

This research is supported by the EU FP6 project "Smart chipS for Smart Surroundings" and the Freeband Knowledge Impulse programme, a joint initiative of the Dutch Ministry of Economic Affairs, knowledge institutions and industry.

REFERENCES

[1] B. Bougard, S. Pollin, G. Lenoir, W. Eberle, L. Van der Perre, F. Catthoor, and W. Dehaene. Energy-Scalability Enhancement of Wireless Local Area Network Transceivers. In *Proceedings of the Fifth IEEE Workshop on Signal Processing Advances in Wireless Communication*, Lisboa, Portugal, July 2004.

[2] K. Masselos, S. Blionas, and T. Rautio. Reconfigurability requirements of wireless communication systems. In *Proceedings of the IEEE Workshop on Heterogeneous Reconfigurable Systems on Chip*, Hamburg, Germany, April 2002.

[3] Paul M. Heysters, Gerard K. Rauwerda, and Gerard J. M. Smit. Implementation of a HiperLAN/2 Receiver on the Reconfigurable Montium Architecture. In *Proceedings of the 11th Reconfigurable Architectures Workshop (RAW 2004)*, Santa Fé, New Mexico, USA, April 2004.

[4] Gerard K. Rauwerda and Gerard J. M. Smit. Implementation of a Flexible RAKE Receiver in Heterogeneous Reconfigurable Hardware. In *Proceedings of the 2004 IEEE International Conference on Field-Programmable Technology*, pages 437–440, Brisbane, Australia, December 2004.

[5] MuMoR project. `http://www.mumor.org`.

[6] EASY project. `http://easy.intranet.gr`.

[7] Pleiades project. `http://bwrc.eecs.berkeley.edu/Research/Configurable_Architectures/`.

[8] PACT XPP Technologies . `http://www.pactcorp.com`.

[9] Silicon Hive. `http://www.siliconhive.com`.

[10] Smart chipS for Smart Surroundings project. `http://www.smart-chips.net`.

[11] AMDREL project. `http://vlsi.ee.duth.gr/amdrel/`.

[12] Tobias Gemmeke, Michael Gansen, and Tobias G. Noll. Implementation of Scalable Power and Area Efficient High-Throughput Viterbi Decoders. *IEEE Journal of Solid-State Circuits*, 37(7):941–948, July 2002.

[13] Guido Masera, Gianluca Piccinini, Massimo Ruo Roch, and Maurizio Zamboni. VLSI Architecture for Turbo Codes. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7(3):369–379, September 1999.

[14] Russell Henning and Chaitali Chakrabarti. An Approach for Adaptively Approximating the Viterbi Algorithm to Reduce Power Consumption While Decoding Convolutional Codes. *IEEE Transactions on Signal Processing*, 52(5):1443–1451, May 2004.

[15] Jagadeesh Kaza and Chaitali Chakrabarti. Design and Implementation of Low-Energy Turbo Decoders. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(9):968–977, September 2004.

[16] Mark A. Bickerstaff, David Garrett, Thomas Prokop, Charles Thomas, Benjamin Widdup, Gongyu Zhou, Linda M. Davis, Graeme Woodward, Chris Nicol, and Ran-Hong Yan. A Unified Turbo/Viterbi Channel Decoder for 3GPP Mobile Wireless in 0.18-μm CMOS. *IEEE Journal of Solid-State Circuits*, 37(11):1555–1564, November 2002.

[17] Joseph R. Cavallaro and Mani Vaya. VITURBO: A Reconfigurable Architecture for Viterbi and Turbo Decoding. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'03)*, volume II, pages 497–500, Hong Kong, China, April 2003.

[18] C. E. Shannon. A Mathematical Theory of Communication. *Bell Systems Technical Journal*, 27:379–423,623–656, 1948.

[19] Shu Lin and Daniel J. Costello. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, 1983.

[20] Andrew J. Viterbi. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, April 1967.

[21] Claude Berrou, Alain Glavieux, and Punya Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo-Codes. In *Proceedings of IEEE ICC'93*, volume 2, pages 1064–1070, Geneva, Switzerland, May 1993.

[22] Claude Berrou and Alain Glavieux. Near Optimum Error Correcting Coding And Decoding: Turbo-Codes. *IEEE Transactions on Communications*, 44(10):1261–1271, October 1996.

[23] 3^{rd} Generation Partnership Project. Technical Specification Group Radio Access Network; Multiplexing and channel coding (FDD). 3GPP TS 25.212 v4.3.0 (2001-12), January 2002.

[24] Roel Schiphorst. *Software-Defined Radio for Wireless Local-Area Networks*. PhD thesis, University of Twente, Enschede, The Netherlands, 2004.

[25] Paul M. Heysters, Gerard J. M. Smit, and Egbert Molenkamp. A Flexible and Energy-Efficient Coarse-Grained Reconfigurable Architecture for Mobile Systems. *Journal of Supercomputing*, 26(3):283–308, November 2003.

[26] Paul M. Heysters. *Coarse-Grained Reconfigurable Processors – Flexibility meets Efficiency*. PhD thesis, University of Twente, Enschede, The Netherlands, September 2004.

[27] M.D. van de Burgwal, G.J.M. Smit, G.K. Rauwerda, and P.M. Heysters. Hydra: an Energy-efficient and Reconfigurable Network Interface. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'06)*, Las Vegas, Nevada, USA, June 2006.

[28] Patrick Robertson, Emmanuelle Villebrun, and Peter Hoeher. A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain. In *Proceedings of IEEE ICC'95*, volume 2, pages 1009–1013, Seatle, USA, June 1995.

[29] John Dielissen, Jef van Meerbergen, Marco Bekooij, Françoise Harmsze, Sergej Sawitzki, Jos Huisken, and Albert van der Werf. Power-efficient layered Turbo Decoder processor. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 246–251, Munich, Germany, March 2001.

[30] Charles M. Rader. Memory Management in a Viterbi Decoder. *IEEE Transactions on Communications*, 29(9):1399–1401, September 1981.