Implementing an Adaptive Viterbi Algorithm in Coarse-Grained Reconfigurable Hardware

Gerard K. Rauwerda, Gerard J.M. Smit
University of Twente, department EEMCS
P.O. Box 217, 7500 AE Enschede, the Netherlands
g.k.rauwerda@utwente.nl

Werner Brugger
Atmel Germany GmbH, Design Center Ulm
Lise-Meitner-Str. 15, 89081 Ulm, Germany

***Abstract*— Future mobile terminals become multi-mode communication systems. In order to handle different standards, we propose to perform baseband processing in heterogeneous reconfigurable hardware. Not only the baseband processing but also error decoding differs for every communication system. Therefore we implemented an adaptive Viterbi decoder in the coarse-grained MONTIUM architecture. The *rate, constraint length* and *decision depth* of the decoder can be adjusted to different communication systems. We showed that the flexibility in the coarse-grained reconfigurable architecture has an estimated cost of 24 times more energy consumption compared to a dedicated solution.**

***Index Terms*— Coarse-grained reconfigurable hardware, System-on-Chip, MONTIUM, Viterbi**

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.

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 algorithms based on signal and channel statistics to achieve optimum performance has drawn renewed attention to run-time reconfigurability [1].

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. Heterogeneous reconfigurable hardware, consisting of processing elements with different granularities, is designed with these constraints – flexibility, performance and energy-efficiency – in mind.

In this paper we will use reconfigurable hardware to implement the Viterbi decoder that is used in wireless communication receivers. Figures presented in [2], [3] show that error decoding in a wireless receiver is as computationally intensive as baseband processing. This means that one should consider optimized implementation of both baseband processing and error correction algorithms for multi-mode communication systems in heterogeneous reconfigurable hardware. We already reported the implementation of baseband processing for different wireless communication systems in the same reconfigurable hardware [4], [5].

In Section II we discuss issues related to our research. The proprosed heterogeneous reconfigurable architecture is shortly described in Section III. Section IV introduces the main aspects of channel coding, while the Viterbi decoder is discussed in Section V. The implementation of the Viterbi algorithm in coarse-grained reconfigurable hardware is given in Section VI. Results on the implementation are discussed in Section VII. In Section VIII conclusions are drawn.

II. BACKGROUND

Conventional reconfigurable processors are bit-level reconfigurable and are far from energy-efficient. *Pleiades* at the University of California, Berkeley, is exploring reconfiguration of coarser-grain application-specific building blocks with an emphasis on low-power computations [6]. Furthermore, PACT [7] proposes an *extreme*

processor platform (XPP) based on clusters of coarse-grained processing array elements. Silicon Hive [8] offers coarse-grained reconfigurable block accelerators and stream accelerators 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* [9] and *AMDREL* [10] projects anticipate these objectives.

In [3] it has been reported that 50% or more of the computational complexity in the wireless receiver is due to error coding algorithms, like Viterbi decoding. This complexity counts for about 60% of the energy consumption in the digital processing part of a typical wireless receiver [2]. Consequently, one realizes that power reduction should be achieved in the error coding part of the receiver. So, new physically oriented design methodologies are proposed in ASIC design for Viterbi decoders [11]. But, one also focusses on power reduction by exploiting variations in system characteristics due to changing noise conditions [12]. These measures can be achieved using dynamic reconfiguration in reconfigurable hardware [13]. However, applying these measures in coarse-grained reconfigurable hardware have not been reported yet.

III. Heterogeneous reconfigurable hardware

The idea of heterogeneous reconfigurable hardware is that one can match the granularity of the algorithms with the granularity of the hardware. We distinguish four processor types: *general-purpose* processor, *fine-grained reconfigurable* hardware, *coarse-grained reconfigurable* hardware and *dedicated* hardware.

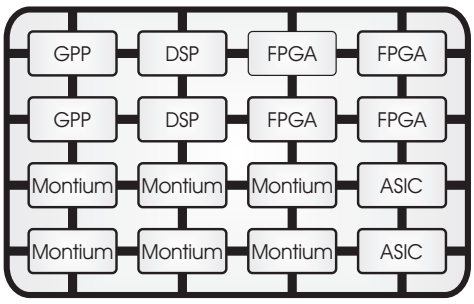

Fig. 1. The Chameleon SoC.

We propose a tiled System-on-Chip (SoC), called Chameleon [14], which consists of the above mentioned processor types (Figure 1). The coarse-grained reconfigurable tiles in the Chameleon SoC will be Montium tile processors [14], as depicted in Figure 2. 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.

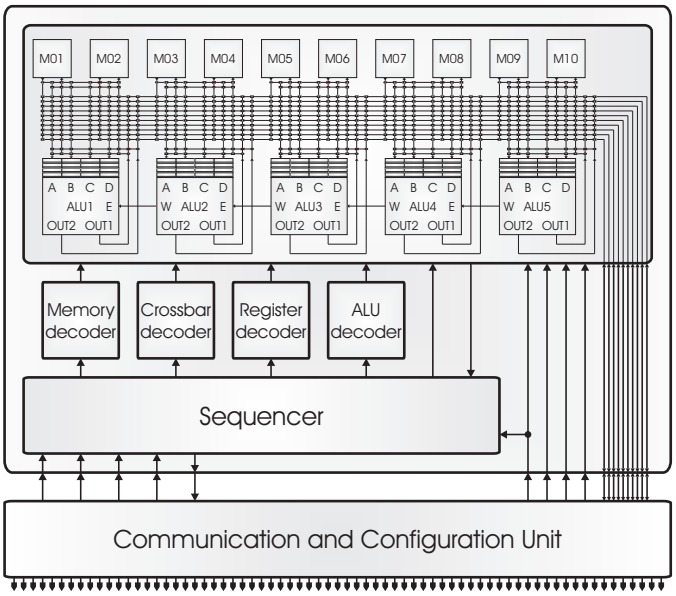

Fig. 2. The Montium tile processor.

A. The Montium tile processor

The Montium is an example of a coarse-grained reconfigurable processor. It 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 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 2 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 TP is the computing part that can be configured to implement a particular algorithm. Figure 2 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. 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 2.

Each local SRAM is 16-bit wide and has a depth of 512 positions, which adds up to a storage capacity of 8 Kbit per local memory. 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).

A single ALU has four 16-bit inputs. Each input has a private input register file that can store up to four operands. The input register file cannot be bypassed, i.e. an operand is always read from an input register. Input registers can be written by various sources via a flexible interconnect. An ALU has two 16-bit outputs, which are connected to the interconnect. The ALU is entirely combinational and consequentially there are no pipeline registers within the ALU. Neighbouring ALUs can also communicate directly on level 2. The West-output of an ALU connects to the East-input of the ALU neighbouring on the left. The East-West connection does not introduce a delay or pipeline, as it is not registered.

IV. CHANNEL CODING

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.

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) and the input data determines the output code. So, the encoding of information can be represented with a mealy state machine. As an example, Figure 3 shows a possible convolutional encoder and its state diagram. 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 $K = 3$.

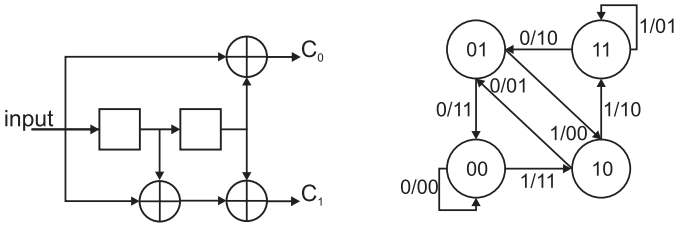

Fig. 3. Convolutional encoder and its state diagram.

A trellis diagram simply shows the progression of the state of the encoder for different symbol times. Figure 4 shows a small trellis diagram, which can be generated from the state diagram of Figure 3. At each time instant 4 states are shown, which correspond to the encoder states.

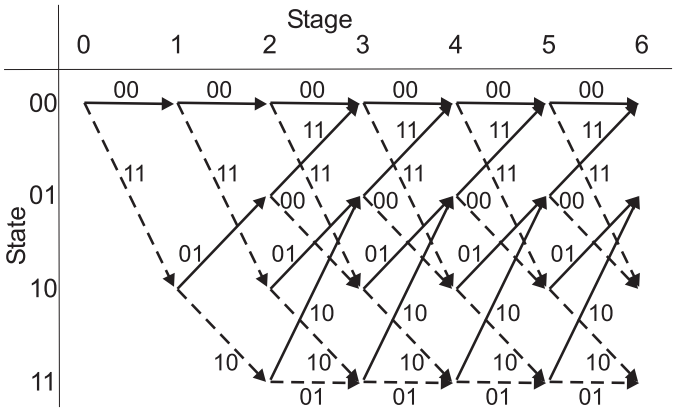

Fig. 4. Trellis diagram with 4 states.

V. VITERBI ALGORITHM

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 [15] 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.

Notice that transitions from state ’00’ to ’00’ and ’10’ are possible as well as from state ’01’ to ’00’ and ’10’. These four transitions form a so-called Viterbi butterfly. A generalized description of the Viterbi butterflies is given in Figure 5. The Viterbi algorithm performs all operations on these butterflies.

The Viterbi algorithm can be divided in three parts: the *branch metric unit*, which performs the branch metric

calculation, the *add-compare-select unit*, which performs the path metric updating, and the *survivor memory unit*, which updates the survivor sequence.

A. Branch metric calculation

The *branch metric unit* determines all branch metrics in the trellis. Branch metric calculation will be performed on Viterbi butterflies. Figure 5 shows the butterflies, which exist in a trellis. The branches connecting to state $S_{i/2}$ correspond always to a decoded '0' as output. The branches connecting to state $S_{(i+N)/2}$ correspond always to a decoded '1' as output.

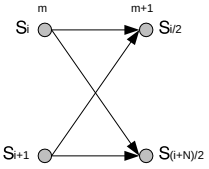

Fig. 5. Viterbi butterfly.

All four branches in the butterfly have a particular codeword assigned. The Euclidean distance is calculated between the received codeword and the codeword, which is assigned to the branch. The codeword length of a *rate* $R = 1/4$ decoder is 4, resulting in the Euclidean distance,

$$\gamma = |y_0 - c_0|^2 + |y_1 - c_1|^2 + |y_2 - c_2|^2 + |y_3 - c_3|^2$$

with y_x the x^{th} bit of the received codeword and c_x the x^{th} bit of the assigned codeword.

B. Path metric updating

The path metrics at the output state are calculated by adding the path metric at the input state and the corresponding branch metric. The path with the minimum path metric is selected as the survivor and the corresponding path metric is assigned to the output state. The example in Figure 6 shows that the branch metrics of the branches have been determined (left picture). In the middle picture the path metrics are determined using the path metrics of the input states and the calculated branch metrics. In the picture is seen that for both output states the survivor origins from the second input state. The path metrics of the survivors for both output states are $3 + 1 = 4$ and $3 + 2 = 5$, respectively.

This operation requires typically a *min()* operation, selecting the minimum of two values. Moreover the originating state (S_i or S_{i+1}) has to be determined, since this information is used in the *survivor memory unit*. The operation is known as *add-compare-select* (ACS); the path metrics and the branch metrics are added, the results are compared and the survivor is selected.

C. Survivor sequence updating

Two approaches are often used to record survivor branches: *traceback* (TB) and *register exchange* (RE) [16]. Basically, the difference between the two approaches is the manner in which information about the survivors is stored during the intermediate steps.

1) Traceback: The TB approach stores the survivor branch of each state. So, the decoded output bit of the corresponding state is stored in every stage. When the survivor branch of each state is known, then the survivor path through the trellis can be reconstructed. Concatenation of decoded output bits in every stage in reversed order of time generates the decoded output sequence of the survivor path through the trellis.

2) Register exchange: The RE approach stores the entire decoded output sequence for each state during path metric updating. Therefore, in each stage of the trellis the decoded output sequence is known and there is no need to traceback.

The right picture of Figure 6 shows the RE approach. The decision bit ('0' or '1') at the output of the butterfly is appended to the bit sequence of the survivor ('10111').

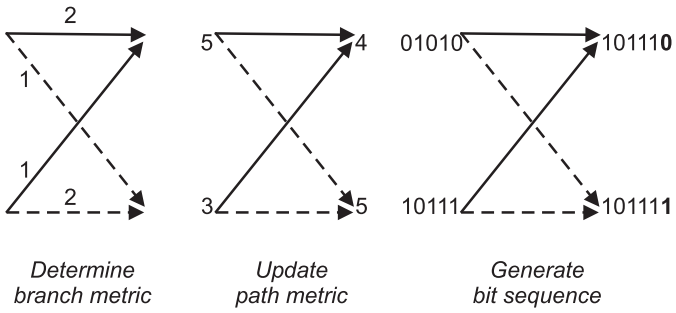

Fig. 6. Example explaining the three parts of the algorithm.

VI. Implementation

We implemented the Viterbi algorithm in the coarse-grained reconfigurable Montium. The mapping of the Viterbi algorithm is highly influenced by the specifications of the reconfigurable hardware:

- survivor sequence updating will be performed with the RE approach, because bit sequences can be stored more efficiently than individual bits.
- due to limited (configurable) instructions, separate memories for input and output states of the Viterbi butterfly are defined and no in-place computation [16] is done.

The bit sequence estimation is performed in a regular pattern. A piece of pseudo-code of the implementation is depicted in Figure 7.

```
@start:
   // perform Viterbi algorithm for next 10 stages
   for stage = 1:10
      calculate_branch_metrics();
      update_path_metrics();
      append_decision_bit_to_bit_sequence();
   end;

   // search the state with minimum path metric and
   // look-up from RE contents the survivor sequence
   search_minimum_path_metric();
   generate_minimum_survivor_sequence();

goto @start;
```

Fig. 7. Pseudo-code of the Viterbi decoder.

A. Branch metric unit

The *rate*, R, of the convolutional code has its impact on the branch metric calculation. The *rate* of the Viterbi decoder can easily be adapted by changing the instructions of the *branch metric unit* in the MONTIUM. Because of the flexibility inside the MONTIUM, one can easily compute a branch metric based on 2, 3 or 4 code bits in parallel. Once the branch metrics are calculated, the values are stored in the local registers of the ALUs and they can be used for the *add-compare-select* task.

B. Add-compare-select unit

The number of Viterbi butterflies is always equal to $\frac{1}{2} * 2^{k-1}$. Hence, the Viterbi decoder in a DAB system with *constraint length* $k = 7$ has to handle 32 butterflies per stage of the trellis.

The *add-compare* operation that has to be performed in the Viterbi butterflies can be easily implemented in the MONTIUM. This requires only an addition and a *min()* operation, selecting the minimum of two values. The path metrics are not only updated in the *add-compare-select* unit, but the decoded bits at the output states are also generated. For every upper state, $S_{i/2}$, of the Viterbi butterfly a '0' is generated as output bit and for every lower state, $S_{(i+N)/2}$, of the Viterbi butterfly a '1' is generated. Moreover, the Viterbi decoder also has to know what the surviving path in the butterfly is. In other words, the originating state of the surviving path through the Viterbi butterfly has to be determined. This is a task of the *select* part in the *add-compare-select* unit.

In the current MONTIUM architecture, described in [14], the *compare-select* operation can be performed by using conditional calls in the sequencer program of the MONTIUM. Drawback of this approach is that only one *select* operation can be performed at a time. In the Viterbi butterfly, however, two *compare-select* operations have to be done; one for each output state of the Viterbi butterfly.

The problem, which arises with the *compare-select* operation, is that the operation merges the data and control path. Control-oriented functions are actually not part of the MONTIUM algorithm domain. Since the *compare-select* operation is very important in the Viterbi algorithm, we propose to add the *compare-select* operation to the ALUs of the MONTIUM. This enables the *compare-select* operation to run independently from the sequencer instructions.

The *compare-select* functionality can be added to the MONTIUM with minor changes in the existing architecture. This enables the *compare-select* operation to run independently from the sequencer instructions. As a result of this minor change in the architecture a larger coverage of the wireless communication domain is obtained. Baseband and channel coding algorithms of different wireless communication standards can now be performed in the same coarse-grained reconfigurable MONTIUM architecture.

C. Memory unit

The Viterbi algorithm comes together with storing two kinds of information. Firstly, the path metric of each state in the trellis has to be stored. Secondly, the decoded bit sequence for each state in the trellis has to be stored. In [16] these kinds of information are referred to scores and hypotheses, respectively.

1) Path metrics: Since many values have to be read and written instantaneously, we chose to read the input information from two memories and to write the results back to two (different) memories. The functions of the memory pairs are interchanged in the consecutive stages of the trellis. The Viterbi implementation in the MONTIUM uses in total four local memories in order to store all path metrics. This implementation with separate input and output memories strikes with the ideas in [16], where in-place computation is proposed.

The in-place computation minimizes the required memory size, but the organization of the memory addressing is irregular for different stages in the trellis. Consequently, many memory addressing instructions are required to implement the in-place computation in the MONTIUM. Furthermore, many (small) local memories are already available in the MONTIUM, which eliminates the need for in-place computation.

2) Survivor sequences: The RE approach for storing the survivor sequences is applied, because bit sequences can be stored more efficiently than separate bits, since the data path of the MONTIUM is 16-bits wide.

The length of the survivor sequence depends on the *decision depth* used in the Viterbi decoder. As a rule of thumb, a *decision depth* of five times the *constraint length* is in practice sufficient. When puncturing is applied to the convolutional code, the *decision depth* will become larger [17]. Since the survivor sequences can only be stored in sequences of 16 bits in the MONTIUM, the entire survivor sequence should be stored in multiple memory addresses when the *decision depth* of the Viterbi decoder is larger than 16. The advantage of such an approach is that we do not need to handle all the bits of the survivor sequence. By using memory pointers – a method that is frequently used in computer science – only the last part of the survivor sequence has to be dealt with.

VII. RESULTS

We implemented a fully flexible Viterbi decoder. The *rate*, R, as well as the *constraint length*, k, and the *decision depth*, d, of the decoder can be adapted. The results in this paper are based on the DAB system [18], $R = 1/4$ and $k = 7$ with a *decision depth* $d = 50$.

A. 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. The data processing of one stage consists of branch metric calculation and path metric updating.

The survivor sequences are stored using the RE approach, combined with using memory pointers. Figure 8 depicts the memory organization of the Register Exchange contents. For the DAB system the memory pointers are 6 bits wide, so 10 bits in the memory are available to store the decision bits. Each pointer is a handle to the memory address of the previous stage. In this way the complete survivor sequence can be constructed.

Whenever 10 stages of the trellis have been processed, the Viterbi decoder decides on the decoded bit sequences of the foregoing stages. Hence, after processing 10 stages of the trellis, the Viterbi decoder has to look-up the bit sequence of the path with the minimum path metric. This procedure requires a search operation through the 64 path metrics and costs 35 additional clock cycles. The look-up operation requires 15 additional clock cycles.

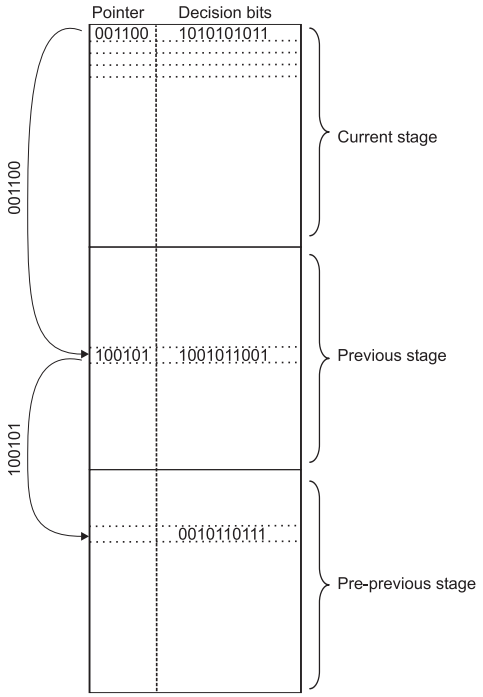

Fig. 8. Register Exchange memory organization with pointers.

In the implemented DAB decoder, always 10 bits are generated during the survivor decision phase. So, 10 bits are generated in $10 \times 42 + 50 = 470$ clock cycles. 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.

B. 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 clock frequency is 100 MHz.

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 wireless channel.

C. Energy consumption

Although energy figures for the MONTIUM are available [14], no exact figures can be given for the implemented Viterbi decoder. We know that the normalized power consumption of the MONTIUM is about 0.5 mW/MHz during Multiply-Accumulate (MAC) operations. However, MAC operations are hardly performed in the Viterbi decoder, thus these will be worst-case numbers. The DAB Viterbi decoder uses on average 47 clock

cycles per bit decision. Hence, the energy consumption of the implemented Viterbi decoder is 23.5 nJ/bit.

Furthermore we estimated that the energy consumption of the Viterbi decoder implemented in the ARM9 is about 5 μJ/bit [19]. However, the ARM9 does not have enough processing power to deliver an output rate of 1.8 Mbit/s for DAB.

A hardwired Viterbi decoder for DAB was implemented in ASIC by Atmel. From discussions with developers from Atmel we know that their implementation of the Viterbi decoder uses about 2 mW in 0.13 μm technology with an output rate of 1.8 Mbit/s. Hence, the average energy consumption is about 1 nJ/bit.

TABLE I

ENERGY/TECHNOLOGY COMPARISON.

	Energy [nJ/bit]	Technology [μm]
ASIC	1	0.13
MONTIUM	24	0.13
ARM9	5000	0.13

VIII. CONCLUSION

We implemented an adaptive Viterbi decoder in the coarse-grained MONTIUM processor. The implemented Viterbi decoder is adaptive in many ways. Depending on the used communication system, one can configure the flexible hardware with the right parameters like *constraint length* and *rate*. Furthermore, the *decision depth* of the Viterbi decoder can be configured. Via dynamic reconfiguration one can change the decision length during operation of the Viterbi algorithm.

We estimated the worst-case energy consumption of the DAB Viterbi decoder in the MONTIUM at 24 nJ/bit. Compared to an ASIC implementation of the DAB Viterbi decoder, flexibility costs about 24 times more energy. Implementing the same decoder in a general purpose processor requires about 5000 times more energy than an ASIC implementation.

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] Jordy Potman, Fokke Hoeksema, and Kees Slump. Tradeoffs between Spreading Factor, Symbol Constellation Size and Rake Fingers in UMTS. In *Proceedings of PRORISC 2003*, pages 543–548, Veldhoven, the Netherlands, November 2003.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

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

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

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

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

[11] 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.

[12] 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.

[13] Sriram Swaminathan, Russell Tessier, Dennis Goeckel, and Wayne Burleson. A Dynamically Reconfigurable Adaptive Viterbi Decoder. In *Proceedings of the 2002 ACM/SIGDA 10th International Symposium on Field-Programmable Gate Arrays*, pages 227–236, Monterey, California, USA, February 2002.

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

[15] 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.

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

[17] Richard D. Wesel, Xuething Liu, and Wei Shi. Trellis Codes for Periodic Erasures. *IEEE Transactions on Communications*, 48(6):938–947, June 2000.

[18] ETSI. Radio Broadcasting Systems; Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers. ETSI EN 300 401 v1.3.3 (2001-05), May 2001.

[19] Michiel Horsman, Thijs Mutter, Marcel van der Veen, Maarten Witteman, and Karel Walters. Energy comparison of Viterbi algorithm on ARM and XScale architecture. University of Twente, Enschede, the Netherlands, 2004. Internal report.