Power-efficient layered Turbo Decoder processor

J. Dielissen[1,2], J. van Meerbergen[1,2], Marco Bekooij[1], Françoise Harmsze[1], Sergej Sawitzki[3], Jos Huisken[1], and Albert van der Werf[1]

[1] Philips Research, Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands
[2] Technical University Eindhoven [3] Technical University Dresden
E-mail: {John.Dielissen, Jef.van.Meerbergen}@philips.com

Abstract

Turbo decoding offers outstanding error correcting capabilities, that will be used in wireless applications like the Universal Mobile Telecom Standard[4] (UMTS). However, the algorithm is very computational intensive, and therefore an implementation on a general purpose programmable DSP results in a power consumption which reduces the applicability of turbo decoding in hand-held applications. In this paper we present a solution based on a layered processing architecture. This architecture includes an application specific Very Long Instruction Word (VLIW) processor, a data flow processor, and hardwired execution units in a hierarchical way. The power consumption of this solution is an order of magnitude better than the implementation on a current state of the art, power efficient general purpose DSP.

1 Introduction

The introduction of 'turbo codes' by Berrou et.al. [1] in 1993 opened up new perspectives in channel coding. The outstanding error correcting capabilities and the increasing importance in wireless communication created a large interest in this coding scheme. Due to recent developments in turbo decoding and the continuing miniaturisation of integrated circuits it became possible to apply the turbo decoding algorithm in hand-held applications. UMTS is one of the first standards which applies the turbo decoding algorithm. Turbo decoding will likely be used in many other applications and standards like wireless LANs and storage. This makes the core an attractive candidate for reuse, hence, the core should be designed as a reusable core.

For a 2 Mbit/sec UMTS turbo decoder approximately 8 giga RISC like operations per second are required. If it whould at all be possible to implement such a computation intensive algorithm on a general purpose RISC DSP will result in a power consumption of approximately 1.92 W (8 Gops * 240 mW/Gops[3]), which reduces the operation time of hand-held applications tremendously. It is therefore necessary to apply a more power efficient solution. Since we will not rely on circuit optimisations and a fully hardwired solution is not flexible enough for us we will further exploit architectural freedom.

Many turbo decoding papers on implementation published until now [5, 8] focus on algorithmic simplifications for implementation cost reduction and their influence on the performance of the algorithm. The first ICs, which are available now, execute the so-called SISO, which is only a part of the turbo decoder and is explained in section 2. In this paper we will show the integrated solution of a complete turbo decoder implementation on both the architectural and implementation level.

In the next section we will first explain turbo decoding and the algorithmic simplifications which are applied to reduce the implementation cost. Then in section 3 the internal architecture of the turbo decoder is discussed in which three layers of processing are distinguished. These three layers lead to a layered processing architecture. The implementation of this architecture is shown in section 4. Thereafter the results and conclusion are given.

2 Turbo decoding algorithm

Channel coding addresses the problem that a message, which has to be sent from the transmitter to the receiver, can be corrupted by noise. Additional

information is sent to enable correction of the errors which occur in the channel. This redundant information can be created using a Finite State Machine encoder: depending on the current state and the input bit, a next state and 1 or more output bits are calculated. This type of coding is also known as trellis coding and can be decoded using for example a Viterbi decoder. Such a decoder estimates the sequence of states followed by the encoder. This enables him to correct some of the errors in the received message.

In turbo decoding multiple codes are used. Each encoder adds redundant information using a different bit order of the message. This bit shuffling is called interleaving and improves the bit error rate (BER) performance of the turbo decoder. The message with the original bit order (called systematic) and the redundant information from the encoders (called parity) is transmitted. The interleaved versions of the systematic message are derived in the decoder, as can be seen in Figure 1.

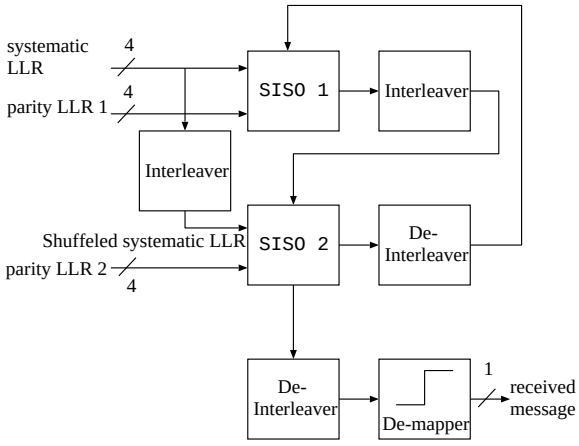

Figure 1. Block schematic of the turbo decoding algorithm.

The turbo decoder will start with decoding the first code. The information obtained during this first decoding step is used as additional information during the decoding of the second code. The usage of this additional information will lead to less errors in this second decoding step. The turbo decoder will use the output of this second decoding step as additional information for re-decoding the first code. This can be continued for a number of iterations, where an iteration is a sequence of the two decoding steps. The additional reduction in BER, which can be obtained in each iteration, reduces and therefore, the number of iterations is often limited to approximately 10. The iterative behaviour of the turbo decoder is shown in Figure 1.

The information on which the turbo decoder operates is the Log-Likelihood Ratio (LLR) of received information: the logarithm of the probability that a 1 is transmitted divided by the probability that a 0 is transmitted. This LLR can be derived from the received signal by means of a single multiplication. The implication of this LLR is the property that only RISC like operation (additions, maximums, and subtractions) are neaded to calculated the algorithm, which leads to a cheaper implementation.

The decoding of the transmitted codes is done using a Soft Input Soft Output decoder. There are several algorithms available for such a decoder. We chose to implement the Max($\star$)-log-MAP [6] algorithm because this algorithm can achieve the lowest BER, allows a trade off between BER performance and implementation cost, and has a regular structure [2]. The algorithm consists of a forward recursion (estimation of the state sequence of the encoder in forward direction), followed by a backward recursion (estimation of the state sequence of the encoder in backward direction), and a soft output calculation (LLR calculation using the forward and backward estimations). All the forward estimations have to be saved (for UMTS 56 bit per state estimation), making the memory for implementation on the full block length (for UMTS 5114 trellis steps) large. Therefore, the algorithm is applied on smaller windows: block length B is divided into a number of windows with length W. This technique is called the sliding window. The number of memory fields is now equal to the window length. The forward recursion for each window is initialised by the last forward recursion in the previous window, as can be seen in the Figure 2.

For initialising the backward recursion (from the light coloured dots in downward direction) either training calculation can be conducted or the Next Iteration Initialisation (NII) technique can be applied [2]. In the training calculation option the initialisation of the backward recursion is done by conducting additional backward recursion calculations. This leads to a more irregular and more computationally intensive algorithm. In the NII technique these backward recursions are initialised with information from the previous iteration, which

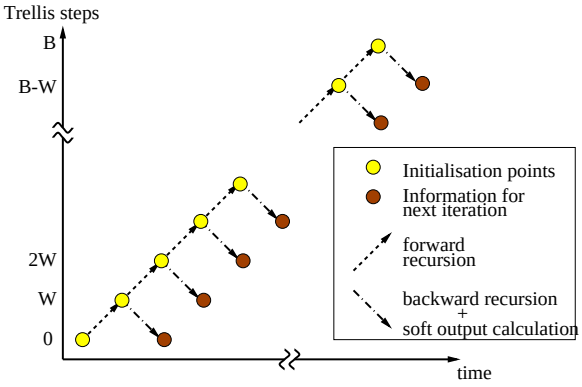

Figure 2. Basic sliding window schedule

is shown in the Figure 2. This leads to a more regular algorithm, at the penalty of an increase in memory locations for storing the state at the end of each window. This last technique is actualy applied in our design and the block schematic is shown in Figure 3.

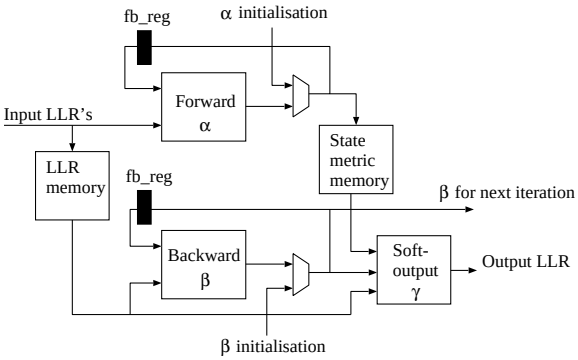

Figure 3. Block schematic of the SISO algorithm

The algorithm is implemented by starting with the forward calculations. During these calculations, the input LLRs and the output information are saved in the LLR memory and the State metric memory respectively. When the forward recursion reaches the end of the first window, it will continue with the next window. At that point the backward and soft output calculations start at the same time and use the saved information. this continues until the end of the trellis is reached. To adapt to all possible block lengths, the SISO algorithm requires the following parameters:

- window length
- number of windows
- last window length

The block length is equal to the window length times the number of windows plus the last window length.

3 Turbo decoding processor architecture

When examining the turbo decoding algorithm the following observations are made: the algorithm is highly irregular, has many conditional constructions like :'if..then..else..', and has to support a variety of options. Most of the execution time is spend in the SISO part, where there are a number of nested loops. There are three functions dominating the inner loop: forward recursion, backward recursion, and soft output calculation. These functions consist mainly of add-compare-select operations.

These observations lead to a hierarchical processing architecture of 3 layers and is depicted in Figure 4.

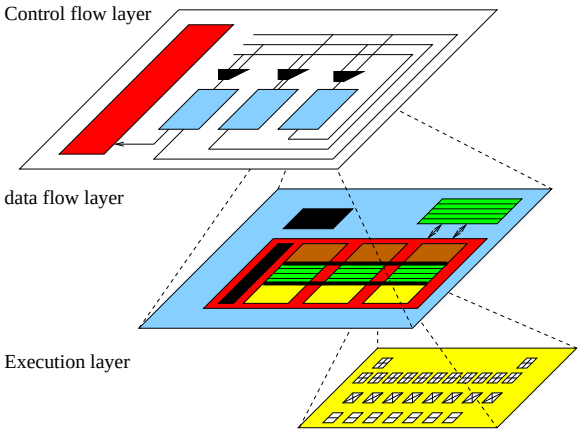

Figure 4. the 3 layers of processing

The highest layer (**control flow layer**), is a sort of router for data streams from and to the SISO unit. In this layer not only the irregularities of the algorithm are solved, but also the consequences of hardware decision (e.g. 1 SISO unit) are dealt with. It has to combines the sequential character of the initialization with the more parallel approach for the streaming the data. This leads to the next list of options that are handled:

- Issues related to standardisation, e.g. UMTS
  - trellis termination (bringing the FSM back to the initial state)
  - variable block length adaption
- Options in implementation
  - manipulating apriori information using source information
  - mapping of both SISOs on one SISO unit
  - mapping the interleaving/De-interleaving on one unit
- Quality of service
  - programmable stopping criteria
  - window length control

An application specific instruction set processor (ASIP) is used to handle this amount of complexity/flexibility in a power efficient way. The algorithm allows for parallelisation of several complex functions (e.g. SISO decoding and Interleaving), leading to a VLIW processor architecture. After defining an initial architecture, the VLIW processor can be programmed, which provides the required flexibility and allows some future extensions. If for example the design needs to be scaled for higher throughput, more units can be added to the architecture. This architecture has a short implementation time and provides an efficient solution for the control flow for different standards.

In the second layer of processing (**data flow layer**), the very regular, high throughput, SISO algorithm is positioned. For this type of streaming based processing we can use architectures that are optimised for data flow, and such architectures are already used in the video application domain.

The calculation units of the data flow layer are in the lowest layer (the **execution layer**). The forward (and backward) calculations are in a recursive loop. This is due to the algorithm, which specifies that the next calculation uses the data produced in the previous calculation, and is visualised by the feed back registers (fb_reg) in Figure 3. It does therefore not give any speed up if these calculations are pipelined. Since these function determine the latency of the SISO it is important to calculate them as fast as possible, preferable within 1 clock cycle. For the soft output calculation unit pipelining can be applied. The data introduction interval for these units remains 1 cycle.

4 Turbo decoder implementation

a VLIW processor architecture template from A|RT Designer is used for implementing the **control layer**. This architecture is shown in Figure 5.

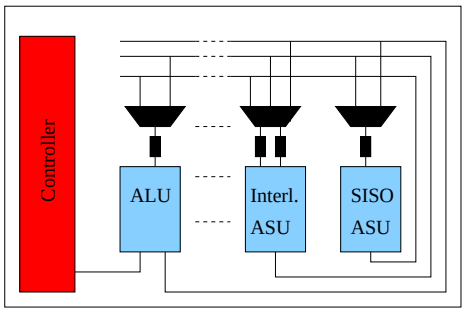

Figure 5. Architecture template for implementing the control flow layer

The architecture consists of a number of standard units like ALUs, address computation units (ACUs), ROMs, and RAMs. Beside these standard units application specific units (ASUs) can be used. The controller of this VLIW architecture consists of a micro code memory, an instruction register and a FSM.

A summary of the instantiated VLIW architecture is given in the next table:

input/output ports	11
ALUs	3
ACUs	2
RAMs	3
ASUs	2
registers	46 fields, 679 bits
micro-rom	149 words of 129 bits

Table 1: Operations in the control flow layer

Address generation for implementing (de-)interleaving is conducted by an ASU. Also, a dedicated ASU is created to execute the SISO function. The VLIW processor controls the ASUs by downloading configuration settings, start commands, compute commands, or No Operations (NOPs).

The SISO ASU is implemented using the tool PHIDEO [7], which was originally developed for video processing. PHIDEO uses a C-like language for specifying operations and data dependencies and

uses pragma's for specifying constraints. These constraints can be:

- schedule constraints
- operation to data-path assignment
- signal to memory assignment
- memory addressing

The architecture template of PHIDEO is shown in Figure 6.

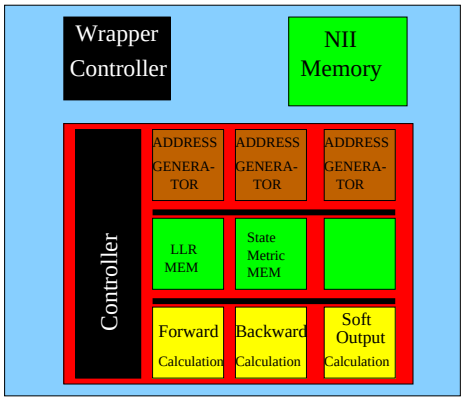

Figure 6. Architecture template for implementing the data flow layer

Since it is not possible to connect the PHIDEO processor directly to the VLIW processor, a wrapper has to be written. This wrapper contains the configuration of the PHIDEO processor and controls the start, reset, and hold pins. The second function of the wrapper is to implement the NII, explained in section 2. Solving this NII in the wrapper is the easiest and cheapest solution.

In PHIDEO the forward, the backward, and the soft output calculations are written down as loops. The next step is to constrain the operations in such a way that the desired schedule, architecture, and memory behaviour is obtained.

The data flow architecture is largely determined by the PHIDEO template and the constraints given. The result is the architecture shown as a block schematic in Figure 3. Because of the large number of bits that have to be stored it is important that they are stored in an efficient way. Figure 7 shows the best possible memory schedule: Both the memory locations and the memory bandwidth are utilised for 100 %. The horizontal bars point out the life time of a variable. The variables which have to be saved during the odd windows have a white colour, the variables for the even windows are coloured black.

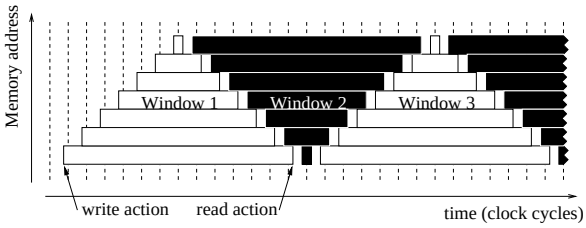

Figure 7. memory schedule

The three functional units of PHIDEO are positioned in the lowest layer: the forward, the backward and the soft output calculations. Figure 8 shows the architecture of the forward recursion.

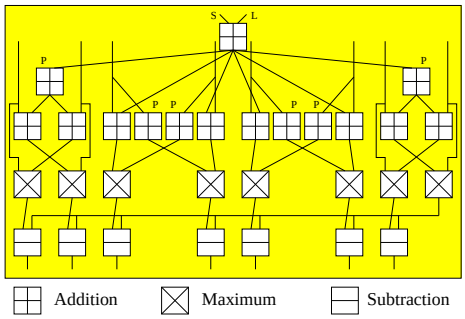

Figure 8. Dedicated fixed architecture for implementing the execution layer

The requirements for the operations in the forward, backward, and soft output calculations(1 clock cycle for calculating all operations and no pipelining) lead to a dedicated, not programmable execution unit.
These units can be written directly in synthesisable VHDL. The number of operations which have to be conducted are shown in the next table.

	addition	Maximum	subtraction
Forward	15	8	7
Backward	15	8	7
Soft output	25	14	1

Table 2: Operations in the execution layer

As can be seen in this table the number of execution operations totals to 100. For a 2Mb/s 10 iteration turbo decoder this requires a processing requirement of 4 Giga operations per second. If we encounter the overhead of (de-)interleaving, sliding window, and

hard decision making the processing requirements increase to approximately 8 RISC like Gops.

5 Results

The turbo decoder is currently described in synthesisable RTL VHDL. For functional verification a mapping to the Xilinx xcv1000e FPGA is done. This mapping showed a logic cells utilisation of 38% and the memory usage of 70%. The total design runs at 15.9 MHz.

For implementation in silicon, a 0.18 μm process is targeted. The design will be realized with standard cells. The total area of these standard cells (excluding the memories and test hardware) sums up to 0.48 mm^2. The decomposition of this cell area over the three processing layers is shown in Figure 9.

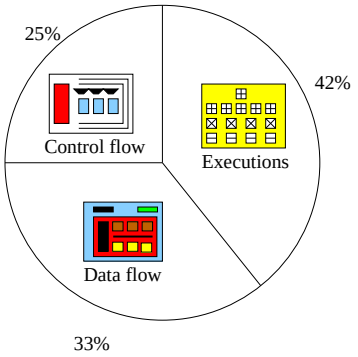

Figure 9. Area decomposition

To obtain an estimation of the power consumption we have used a simulator based on the net-list and power models of cells and wires. These simulations showed that a power dissipation of approximately 35 mW. This estimation, which results in a power efficiency of 0.005 mW/MOPS, is done for a supply voltage of 1.8V and a clock frequency of 50 MHz. The decomposition of the power dissipation over the three processing layers is shown in Figure 10.

6 CONCLUSIONS

In this paper we presented a layered processing architecture for implementing a turbo decoder. We showed that our solution has a power efficiency of 0.005 mW/MOPS. We achieve this result by applying a layered processing architecture, where flexibility is only used in the highest processing layer, the data flow layer is configurable, and the execution layer is completely fixed. No additional power reduction techiques like voltage scaling or low power circuit techniques are applied.

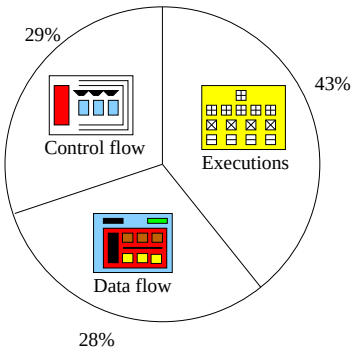

Figure 10. Power decomposition

References

[1] C. Berrou, A. Glavieux, and P Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo codes. In *IEEE Proceedings of ICC '93*, pages 1064–1070, May 1993.

[2] J. Dielissen and J. Huisken. Implementation issues of 3rd generation mobile communication turbo decoding. In *21st Symposium on information theory in the benelux*, pages 9–16, May 2000. Wassenaar, The Netherlands.

[3] TMS320c55X^{TM} DSP Generation http://www.ti.com/sc/docs/products/dsp.

[4] http://www.3gpp.org.

[5] G. Masera, G. Piccinini, M. Ruo Roch, and M. Zamboni. VLSI architectures for turbo codes. *IEEE Transactions on VLSI Systems*, 7(3):369–378, September 1999.

[6] P. Robertson, E. Villebrun, and P. Hoeher. A comparison of optimal and suboptimal MAP decoding algoritms operating in the log domain. In *Proceedings 1995 International Conference on Communications*, pages 1009–1013, 1995.

[7] J.L. van Meerbergen, P.E.R. Lippens, W.F.J. Verhaegh, and A. van der Werf. Phideo: High-level synthesis for high throughput applications. *Journal of VLSI Signal Processing*, 9:89–104, 1995.

[8] Z. Wang, H. Suzuki, and K. K. Parhi. VLSI implementation issues of turbo decoder design for wireless applications. In *IEEE Workshop on Signal Processing Systems*, pages 503–512, 1999.