Design of Low-Power High-Speed Maximum a Posteriori Decoder Architectures

Alexander Worm*, Holger Lamm, Norbert Wehn
Institute of Microelectronic Systems
Department of Electrical Engineering and Information Technology, University of Kaiserslautern
{worm, lamm, wehn}@eit.uni-kl.de

Abstract

Future applications demand high-speed maximum a posteriori (MAP) decoders. In this paper, we present an in-depth study of design alternatives for high-speed MAP architectures with special emphasis on low power consumption. We exploit the inherent parallelism of the MAP algorithm to reduce power consumption on various abstraction levels. A fully parameterizable architecture is introduced, which allows to optimally adapt the architecture to the application requirements and the throughput. Intensive design space exploration has been carried out on a state-of-the-art 0.2 μm *technology, including efficient parallelism techniques, a data flow transformation for reduced power consumption, and an optimized FIFO implementation.*

1. Introduction

For many communication systems, low power consumption is an important design goal; this is not restricted to mobile applications. Low power implementation does not only increase the battery life time, it also allows cheaper packages and increases the reliability of a system.

Important parts of communication systems are channel coding and decoding, which combat errors induced by noisy channels. At the transmitter side, a channel encoder adds redundancy to the data to be transmitted. This redundancy is exploited at the receiver side by the channel decoder to correct the errors. A common measure for the performance of a coding scheme is the bit-error-rate (BER) as a function of the signal-to-noise ratio (SNR) E_s/N_0. Turbo-Codes, first published in 1993 [3], show the best forward error correction performance known up to now. Thus, Turbo-Codes became part of the third generation wireless systems standard, which offers data-rate services up to 2 Mbit/s for internet and multimedia applications. Beyond that, Turbo-Codes are used in or in discussion for WLAN, VDSL, satellite communications, digital broadcasting, optical transmission and hard disk applications. Some of these applications demand fast data transmission in the range of 10–100 Mbit/s and even above. For fiber optics, even channel coding speeds of several Gbit/s are foreseeable.

Turbo-Codes operate on block level, that is the data is separated into blocks. A block is decoded in an iterative procedure [3] by maximum a posteriori (MAP) or soft output Viterbi algorithm (SOVA) component decoders. These are, along with an interleaver, the primary building blocks of a Turbo-decoder. We put emphasis on the decoder, because the complexity of the decoder is much higher than the complexity of the encoder. MAP decoders are superior with respect to communications performance and for that reason preferred in advanced implementations. The MAP decoder throughput is up to 20 times higher (@ 10 iterations) than the Turbo-decoder throughput. Turbo-decoding in the range of 10–100 Mbit/s, which will be quite common for WLAN and VDSL, requires therefore 200 Mbit/s–2 Gbit/s MAP decoding speed. Yet MAP decoding at a speed of 500 Mbit/s is quite conceivable for applications like magnetic recording, even without the multiplicative effect of the Turbo-iterations. Because of the emerging importance of high-speed Turbo-decoders, low power implementation of high-speed MAP decoders is an important objective.

This paper concentrates on the MAP decoder and leaves issues related to the interleaver for future work. We present an in-depth study of design alternatives for low power high-speed VLSI MAP decoder architectures. We try to reduce the power consumption on various abstraction levels. A fully parameterizable architecture is introduced that allows to optimally adapt the architecture to the application requirements and the throughput. In detail we present

- efficient parallelism techniques that can nearly arbitrarily increase the throughput,
- a data-flow transformation that has a great impact on the power consumption,
- optimized FIFO implementations, which are very beneficial for low power high-speed implementations,

*A. Worm is the recipient of a Motorola Partnerships in Research Grant.

- and, most importantly, an intensive design space exploration, based on a fully parameterizable VHDL model, to quantify these techniques with respect to area and power consumption.

The paper is structured as follows: Section 2 discusses related work. Section 3 introduces the MAP algorithm and section 4 presents the different design alternatives. We give results in section 5 and draw conclusions in section 6.

2. Related work

Up to date, some papers on Turbo-decoder implementation with the MAP algorithm have been published (see, for example, [11, 9]), but most of them target modest communication speeds at about 2 Mbit/s. In contrast to high-speed Viterbi decoders, only very few publications exist on high-speed Turbo-decoders ($\geq$50 Mbit/s) and on high-speed MAP decoders. One of those publications that presents a high-speed MAP decoder is [10], which allows a Turbo-decoder implementation of 50 Mbit/s throughput. The paper states that the MAP decoder kernel would be able to run at 1 GHz on a 0.5 μm technology. The recursions are implemented as FIR filters, which implies, however, acquisition for each trellis step and is therefore inefficient from an algorithm point of view (see sections 3 and 4); power consumption is not considered. In another set of papers, Dawid extended the work of Fettweis on the Viterbi algorithm toward the MAP algorithm and proposed architectures for high-speed MAP decoding [5, 4], but power consumption was not a primary concern. In [13], we introduced a new data-flow transformation for high-speed MAP architectures that outperforms Dawid's architecture with respect to area and power consumption, but we could not yet provide a detailed design space exploration for a broad range of architectures, which is the focus of this paper.

Some papers indeed target low power consumption as a primary goal, but they, instead of aiming at high throughput rates, concentrate on memory size and transfer optimization, add control flow to the MAP algorithm or target programmable architectures (see, for example, [9, 6]).

3. The MAP algorithm

This section only contains some fundamentals on the MAP algorithm; for a detailed description see, for example, [1, 4, 8]. The MAP algorithm computes the probability of the source symbol I_k in step k, conditioned on the knowledge of the received distorted symbol sequence $z_0^N = (z_0, z_1, \ldots, z_k, \ldots, z_N)$:

$$\Pr\{I_k | z_0^N\}. \tag{1}$$

The soft-output of the MAP algorithm

$$\Lambda_k = \log \frac{\Pr\{I_k = 1 | z_0^N\}}{\Pr\{I_k = 0 | z_0^N\}} \tag{2}$$

is termed *log-likelihood ratio* (LLR). The probability $P\{I_k | z_0^N\}$ is computed by determining the probability to reach a certain encoder state m after reception of k symbols $z_0^{k-1} = (z_0, z_1, \ldots, z_{k-1})$,

$$\alpha_k(m) = \Pr\{m | z_0^{k-1}\}, \tag{3}$$

and the probability to get from encoder state m' to the final state with the symbols z_{k+1}^N,

$$\beta_{k+1}(m') = \Pr\{z_{k+1}^N | m'\}. \tag{4}$$

The probability of the transition $m \rightarrow m'$ using the source symbol I_k, under knowledge of the received symbol z_k, is called γ_k:

$$\gamma_k(m, m', I_k) = \Pr\{m, m', I_k | z_k\}. \tag{5}$$

The branch metrics γ_k are a function of the received symbols and the channel model. We term the process of computing the γ_k *branch metric calculation*. The probabilities α_k and β_{k+1} can be found recursively over the γ during *forward recursion* and *backward recursion*, respectively. The probability of having sent the symbol I_k in step k is the sum over all paths using the symbol I_k in step k. With $\phi(I_k)$ being the set of all transitions with symbol I_k, we can write:

$$\Pr\{I_k | z_0^N\} = \sum_{(m,m') \in \phi(I_k)} \alpha_k(m) \cdot \gamma_k(m, m', I_k) \cdot \beta_{k+1}(m'). \tag{6}$$

The calculation of equation (2) using equation (6) is termed *soft-output calculation*.

As pointed out in [8], it is mandatory to implement the MAP algorithm in the logarithmic domain (Log-MAP) to avoid numerical problems without degrading the decoding performance. This simplifies the arithmetic operations: multiplication becomes addition, and addition becomes the max*-operator [8]. Throughout the remainder of this paper, the definitions $\delta = \log\alpha$, $\varepsilon = \log\beta$, and $\mu = \log\gamma$ are used for notational convenience.

4. MAP decoder design space

An efficient MAP decoder implementation is the result of a trade-off between communications performance (here: achievable bit-error rate at a signal-to-noise ratio E_s/N_0) and VLSI performance (throughput, area, power consumption). Simplifying the algorithm reduces the implementation complexity but often decreases the communications performance. Typical examples are the Max-Log-MAP simplification and the quantization process. The Max-Log-MAP algorithm [8] is obtained by using the approximation $\max^*(\xi_1, \xi_2) \approx \max(\xi_1, \xi_2)$ and shows therefore a reduced

decoding performance. On the other hand, this algorithm can be implemented with less area and power consumption, if fixed throughput is assumed. A second example for a performance trade-off is the mapping of the algorithm onto a fixed-point number representation, as required for VLSI implementations. Usually, this quantization has an impact on communications performance, but we have shown in [7] that for Turbo-decoding the degradation can be kept within very tight limits. Another important implication of fixed-point number representation is the necessity to avoid overflow, which can be achieved by renormalization. If properly designed, periodic normalization does not alter communications performance, but it can have a significant positive impact on VLSI performance [12]. The present paper, instead, avoids overflow by using saturation arithmetic and by calculating the branch metrics in a way that branch metrics that represent a good match between the anticipated and the received symbol are near zero.

Some architectural transformations, however, increase the VLSI performance of a MAP decoder without or with little effect on the communications performance. The main contribution of this paper is a synthesis-based exploration of such architectural transformations for high-speed MAP decoder architectures on different levels of abstraction that enable a reduction in power consumption. Because these transformations have no or only negligible effects on the communications performance, this paper does not show BER curves.

High-speed MAP decoder implementations demand extensive parallelism to match the architectural with the system throughput. Fortunately, a MAP decoder has inherent parallelism on different abstraction levels: operator level, state level, recursion level and trellis level [2]. This parallelism can be efficiently implemented in an actual architecture by spatial parallelism and/or pipelining, which then enable high-speed architectures.

Power consumption in high-speed MAP implementations is dominated by the dynamic power consumption. The optimizations presented in this paper consider all factors of dynamic power consumption on various abstraction levels:

- reduction of V_{DD}, which is possible because of efficient parallelism techniques that allow to increase the parallelism to compensate the decreased circuit speed (section 4.1);
- a data-flow transformation that reduces the memory and thus the physical capacitance as well as the switching activity (section 4.2);
- efficient FIFO implementations, which reduce the switching activity and physical capacitance (section 4.3).

Note that these techniques have no effect on the communications performance.

Bit-width minimization obviously also reduces power consumption. An efficient quantization scheme is presented in [7] and is therefore not further discussed. Saving power by exploiting data correlation is not applicable here, because the input data of a channel decoder is (ideally) randomly distributed.

4.1. Parallelism

In this paper, emphasis is put on the highest abstraction level, because the introduction of parallelism on trellis level is more efficient than the introduction of parallelism on lower levels of abstraction [4].

A trellis is a common representation of a convolutional code. It is the unrolled state-chart of the encoder where nodes with identical labels are merged. On trellis level, the data-block (frame) can be tiled into a number of windows. Each window can be processed independently from the others, which provides a high degree of parallelism. This technique breaks the well known add-compare-select bottleneck on a higher level of abstraction and enables high-speed architectures as well as voltage scaling. It is referred to as *overlapping sliding windows* or *parallel MAP* and was first presented in [5]. We define W as the window width in trellis steps.

A data-dependency graph visualizes the data flow between arithmetic units. The data-dependency graph of a parallel MAP architecture introduced in [4] is shown in Figure 1 for three neighbored windows of width $W = 6$. We term this type of architecture a D architecture because of the diamond shape at the lower part of each window's data-dependency graph.

Throughout the remainder of this paper, a code constraint length of $K = 3$ is assumed where applicable, by way of example only. A $K = 3$, or 4-state, code is expected to deliver a passable trade-off between communications and VLSI performance. All mentioned principles, however, can be similarly applied to a code with any number of states; the data-flow transformation, in particular, is expected to benefit also larger number of states: although the arithmetic units become more complex, the number of trellis states to be saved increases likewise. For reference, the trellis of a constraint length $K = 3$ code is shown on the top of Figure 1. The trellis steps are numbered from left to right; in the data-dependency graph, time proceeds from top to bottom. The data-dependency edges each correspond to several values. For example, 2^{K-1} values correspond to each δ and ε edge of the data-dependency graph, and a respective number of values corresponds to each μ edge. We distinguish arithmetic units for forward acquisition (explained below), backward acquisition, forward recursion, backward recur-

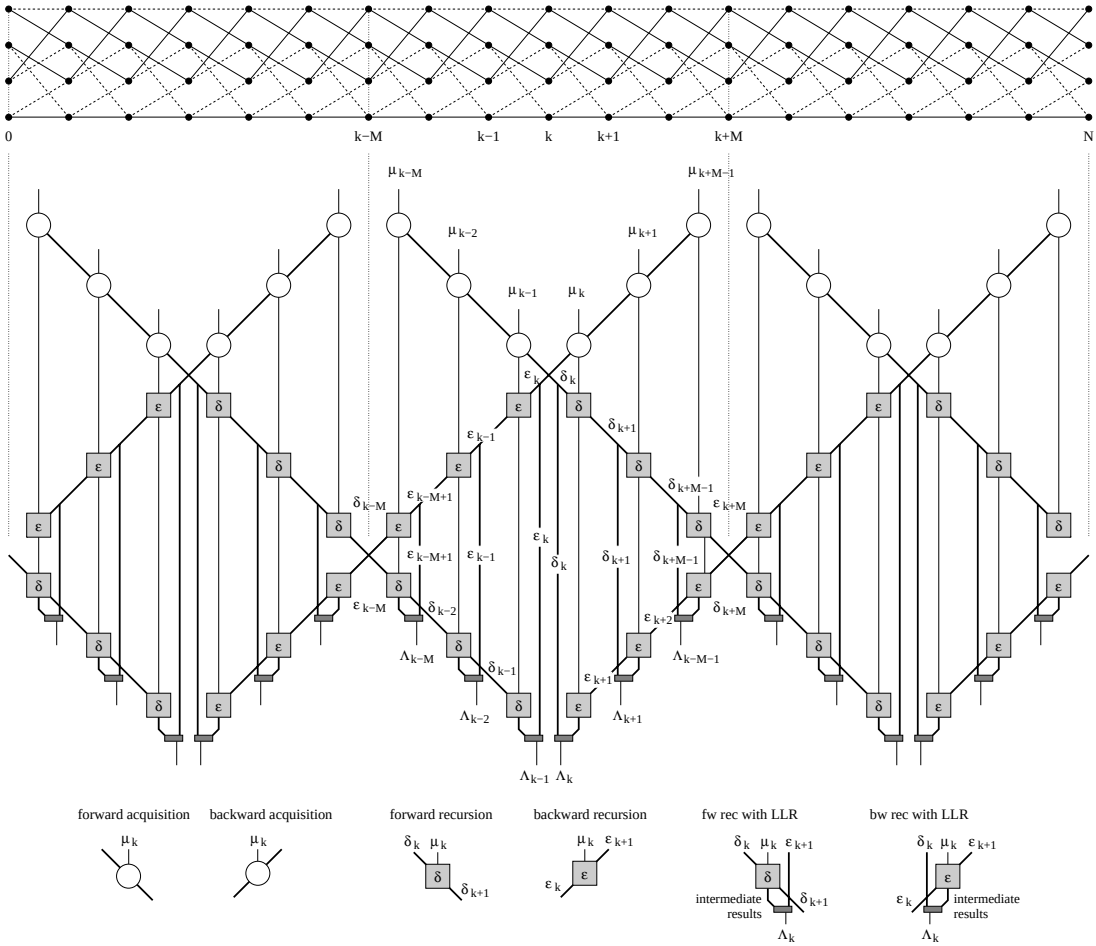

Figure 1. Detailed data-dependency graph of the D architecture ($3W = N$, e. g.)

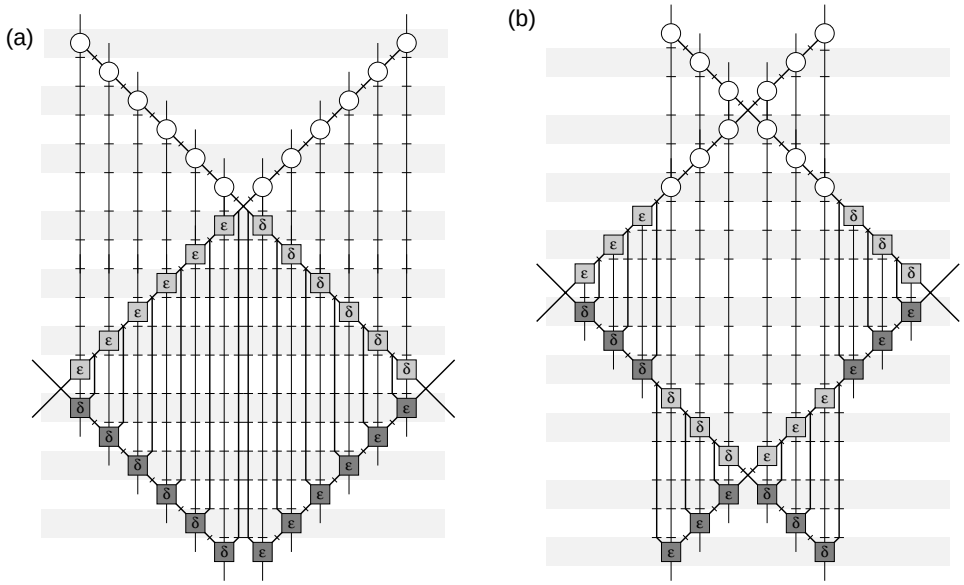

Figure 2. Fully pipelined D and X architectures (for $M = 6, \eta = 2$)

sion, forward recursion with soft-output calculation, and backward recursion with soft-output calculation.

The algorithm works as follows: Within a window, forward and backward recursion start simultaneously at different trellis steps, $k-M$ and $k+M$. The path metrics δ_{k-M} and ε_{k+M} are unknown and therefore set to an arbitrary value. According to theory [4], the path metrics will converge during the first M recursions toward reliable values. This phase is called the acquisition phase and M is called the acquisition depth. At the end of the acquisition phase, the path metrics have become reliable and the values can be stored. After another M steps, both recursions have reached the end of the window and are continued in the next one: ε_{k-M} and δ_{k+M} are fed into the left and right neighboring window, respectively; δ_{k-M} and ε_{k+M} are obtained from these windows, and recursions continue with concurrent calculation of the outputs Λ_k.

As said above, all windows of a frame are processed in parallel. Thus no timing problems arise at the path metric exchange between adjacent windows. Except proper initialization of the path metrics, no special arrangements are necessary at either end of a frame.

Until now, the length of the recursion paths producing reliable values (which equals the window-width W) has been assumed to be twice the acquisition depth M, that is $W = 2M$. In general, we write $W = \eta M$. For $\eta < 2$, the acquisition paths cross the window borders and have to be started in the neighboring windows [13]. The acquisition depth M is determined by system constraints, for example $M = 10$–20 for a constraint length $K = 3$ code. The architect therefore controls η only. For a fixed window-width, η is inverse proportional to M, whereas for a fixed acquisition depth, the window-width is proportional to η.

On recursion level, we assume that the branch metrics have been precomputed. Forward or backward recursion and soft-output calculation are executed in parallel where appropriate.

On state level and on operator level, full parallelism is used. For maximum throughput, the recursions and the soft-output calculation have to be fully unrolled and pipelined, that is a pipeline register is put between successive forward and backward recursion steps. The life-time of the values then directly corresponds to the number of pipeline stages. The data dependencies are mapped on FIFO memories, so memory usage is linearly dependent on the life-time. Thus, the architecture, from the first acquisition to the last result, has a latency of $l = (\eta+1)M-1$ and up to $(\eta+1)M$ windows are processed simultaneously in the pipeline.

In general, memory demand and latency are smaller for small η. On the other hand, the number of arithmetic units increases by lowering η [13]. It is therefore necessary to find a trade-off for minimum area and power consumption by optimizing the window-width W.

The presented fully pipelined architecture can process one block of input data every cycle, because the number of windows is adapted to the block size. This technique yields a very high throughput that is probably too high for many applications. To reduce the throughput, we can either lower V_{DD} or share arithmetic units, that is $\theta > 1$ consecutive recursion steps are carried out on the same hardware [13]. Sharing reduces pipeline lengths and increases the data init interval by θ. For example, the number of arithmetic units is halved for $\theta = 2$, but they have to be enhanced by multiplexers. Shared architectures behave like architectures with a narrower window-width $\frac{\eta M}{\theta}$. Thus, memory is also reduced.

4.2. Data-flow transformation

As mentioned in section 4.1, memory usage in heavily pipelined MAP decoder architectures is proportional to the life-time of a value, which is in our figures visualized as the length of the corresponding data-dependency edge. In the data-dependency graph of the D architecture in Figure 1, the longest δ and ε paths are those of the first few values generated after finishing acquisition, with indices around k. Removing these edges will have great effect on memory.

Figure 2 (a) features a fully pipelined D architecture with $M = 6$ and $\eta = 2$. Soft-output calculation has been merged with recursion for clarity, indicated by darker shading. Small horizontal and diagonal bars denote pipeline registers arranged between the arithmetic units, and along the data-dependency edges to realize FIFO functionality. Note that if the recursions are continued at the bottom of Figure 2 (a), the produced values can be directly used for soft-output calculation. As a result, the recursions creating the upper borders of the rhombus shape do not need to produce reliable δ and ε values. They can be used for acquisition. This leads to the novel X architecture, shown in Figure 2 (b), which was introduced in [13]. The X refers to the X-shaped structure at the bottom.

We have proven in [13] that the new architecture yields significant improvements in memory size. We can show that X-shaped data-flow graphs are also advantageous for heavily (large θ) and even fully shared architectures.

4.3. FIFO implementation

It was mentioned above that considerable savings in area and power consumption can be achieved by using architectures based on X data-flow graphs. These reductions are due to shorter FIFO memories. Additional savings are possible by optimum FIFO realization. Thus, we consider also gated-clocks and ring-buffers. Clock-gating is power efficient only for $\theta > 1$. A ring-buffer improves power consumption for all θ in comparison to register chains, at the

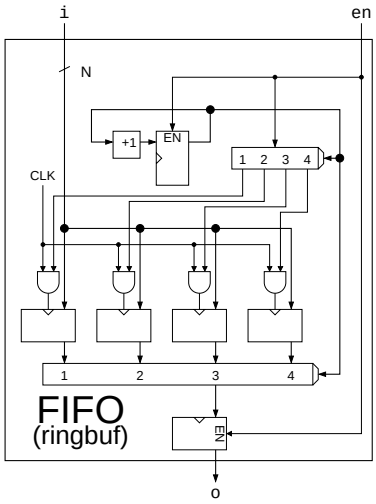

Figure 3. Architecture of ring-buffer FIFO realization

cost of increased area. Ring-buffer FIFO realization moves an address pointer instead of the data and therefore prevents unnecessary toggling. Our ring-buffer implementation, which is shown in Figure 3, uses internal clock-gating in addition. These techniques can drastically reduce the switching activity. Further power reduction is possible by using optimized cells.

5. Results

The high-speed VLSI MAP decoder architecture has been implemented as a parameterizable and fully synthesizable VHDL model. A script generator allows efficient automated synthesis, and area and power characterization. With this method, we have been able to perform a very efficient design-space exploration. We have used a $0.2\,\mu$m state-of-the-art Artisan standard cell library with $V_{DD} = 2.5$ V nominal voltage and the Synopsys tool suite for synthesis and characterization. Worst case parameters were assumed during synthesis (supply voltage $V_{DD} = 2.25$ V, junction temperature $T_j = 110^{\circ}$C, process derating factor of 1.37). Place & route were not explicitly carried out, except for some configurations; we expect no significant area and delay penalty by place & route, since the architecture is dominated by local interconnect. Clock tree and I/O contributions have been excluded, since they contribute only to a constant offset for different architectures. During power characterization, white noise has been assumed at the inputs, which is a reasonable assumption for a channel decoder.

For a block size of 600 bit, a maximum throughput of 45 Gbit/s is achievable at a clock rate of only 75 MHz. Sharing can scale down the throughput to 300 Mbit/s, which corresponds to a Turbo-decoder throughput of some ten Mbit/s.

Bit-width minimization is important for reducing power consumption. For example, we can show that reducing the bit-width by just one bit can reduce area and power consumption by up to about 25%. Omitting path metric renormalization in favor of normalizing the branch metrics was in this context superior to modulo arithmetics. In the following the bit-width of the branch metrics μ_k has been set to $N_\mu = 6$. The required bit-width of the path metrics δ_k and ε_k was found as $N = 7$. Two bits were used as fractional part (decimals).

5.1. Impact of window-width

We prove our earlier presumption that an optimum η might exist by analyzing architectures with various window-widths $W = \eta M$ and $M = 12$. Note that a acquisition depth of $M = 12$ is sufficient, upon tolerable loss in communications performance, for a 4-state code, whereas a code with a larger number of states would require a longer acquisition phase. Our results are given in Figure 4 for configurations without sharing. Each graph features curves for D and X architectures of a Log-MAP and a Max-Log-MAP decoder for a constraint length $K = 3$ code. The results are normalized to one input symbol to allow comparison of different window-widths. Optimum area and power consumption can be achieved for window-widths around $W = 24$. Results for shared architectures are omitted for space limitations. We can show, however, that for shared architectures the optimum W becomes larger. Nevertheless, $W = 24$ is assumed throughout the following discussions.

5.2. Impact of data-flow transformation

Figure 5 clearly shows that the X architecture optimizes power consumption and area for all configurations. For $W = 24$ and $\theta = 1$ ($\theta = 6$) the power consumption of a Max-Log-MAP decoder is reduced by 19% (7%) and area is reduced by 16% (12%). Smaller improvements are obtained for the Log-MAP decoder because of the higher implementation complexity of the arithmetic units. There, power consumption is reduced by 14% (5%) and area is reduced by 10% (8%). The improvements are still significant; thus the X architecture should always replace the D architecture.

5.3. Impact of FIFO implementation

Figure 6 compares the effects of different FIFO implementations on power consumption and area of a Max-Log-MAP decoder. The ring-buffer implementation can considerably reduce the power consumption (-26%) for $\theta = 1$, that is without hardware folding, but it needs more area ($+11\%$) because of the overhead for pointer management.

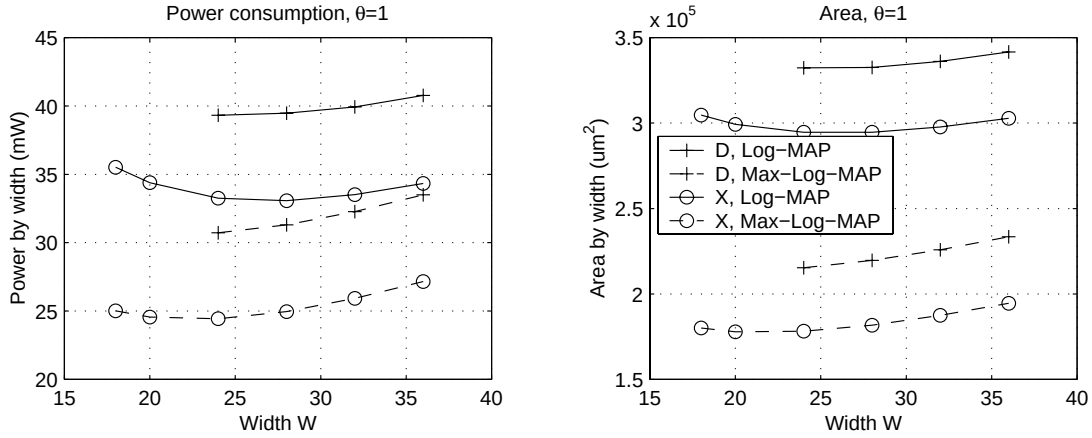

Figure 4. Impact of window width, normalized to one input symbol, $K = 3$

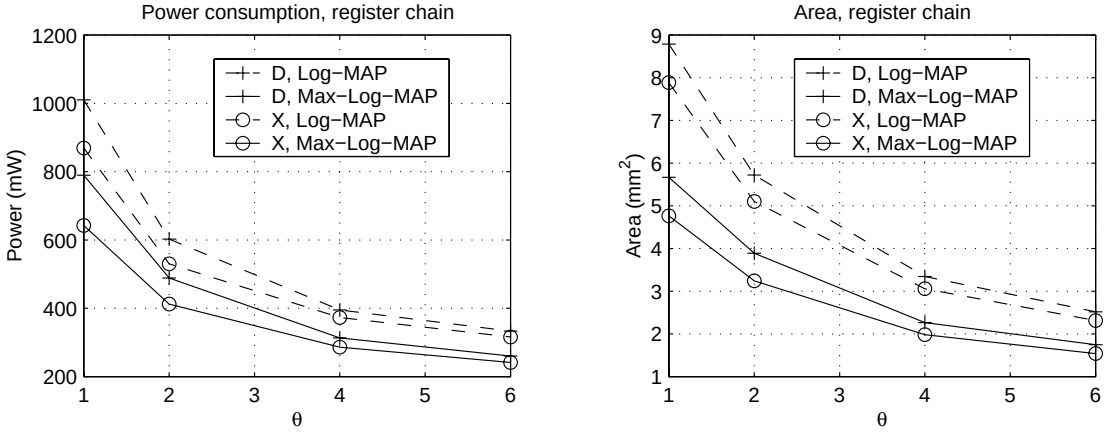

Figure 5. Impact of data-flow transformation, $W = 24$**,** $K = 3$

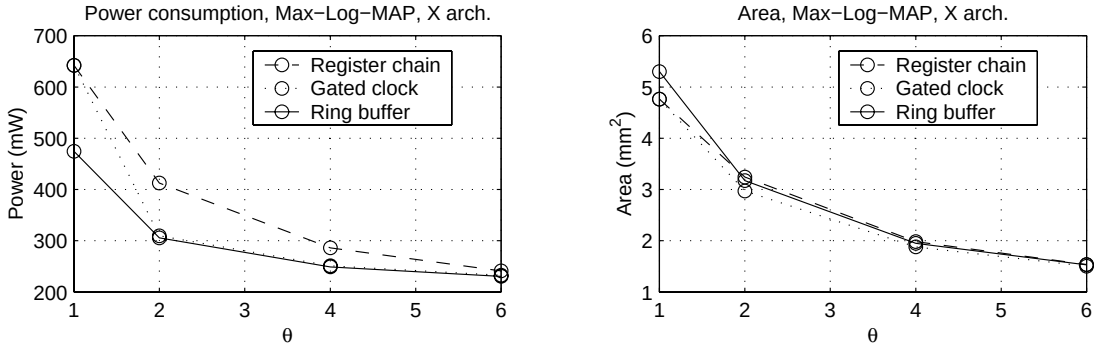

Figure 6. Impact of different FIFO implementations, $W = 24$**,** $K = 3$

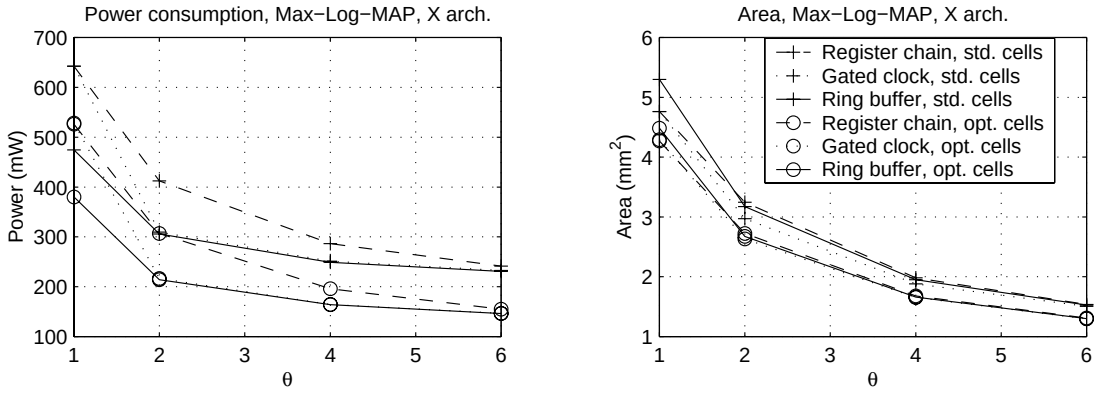

Figure 7. Impact of optimized register cells, $W = 24$**,** $K = 3$

For $\theta = 1$, register chain and gated clock implementation are the same. The picture is different if the architecture uses hardware folding. In terms of power consumption, gated clock implementation becomes clearly superior to register chains and performs virtually as well as ring-buffers. It is also the optimum choice with respect to area.

We have developed special semi-static register cells that are optimized with respect to power consumption. These cells have been fully characterized and included in the library. Results are given in Figure 7: 30% less power consumption has been achieved with the optimized cells, for $\theta = 2$ and using gated clock FIFO implementation. The area savings are in the order of 10%.

6. Conclusions

We have presented MAP architectures that can decode at high throughput rates, but are nevertheless optimized for low power consumption. They are based on standard cells and run at moderate clock frequencies. The key idea is to parallelize on all abstraction levels instead of striving for very high clock frequencies. This enables optimizations that effectively reduce power consumption and allows also voltage scaling. Based on our design space exploration, we have found that a low power high-speed MAP architecture without hardware sharing should use a window width around $W = 24$ (for $K = 3$). It should be based on X shaped data-dependency graphs and implement FIFO memories as ring-buffers. Architectures with hardware sharing can benefit from larger window widths and should use gated clock FIFOs. Optimized register standard cells are always very beneficial.

Future work will be directed at breaking the interleaver bottleneck for parallel MAP architectures in the context of Turbo-decoding. Moreover, we will try to quantify the effects of voltage scaling on the power consumption of high-speed MAP decoder architectures.

References

[1] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate. *IEEE Tr. on Information Theory*, IT-20:284–287, Mar. 1974.

[2] F. Berens, A. Worm, H. Michel, and N. Wehn. Implementation Aspects of Turbo-Decoders for Future Radio Applications. In *Proc. VTC '99 Fall*, pages 2601–2605, Sept. 1999.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes. In *Proc. ICC '93*, pages 1064–1070, Geneva, Switzerland, May 1993.

[4] H. Dawid. *Algorithmen und Schaltungsarchitekturen zur Maximum a Posteriori Faltungsdecodierung*. PhD thesis, RWTH Aachen, Shaker Verlag, Aachen, Germany, 1996.

[5] H. Dawid, G. Gehnen, and H. Meyr. MAP Channel Decoding: Algorithm and VLSI Architecture. In *VLSI Signal Processing VI*, pages 141–149. IEEE, 1993.

[6] F. Gilbert, A. Worm, and N. Wehn. Low Power Implementation of a Turbo-Decoder on Programmable Architectures. In *Proc. ASP-DAC '01*, Jan. 2001.

[7] H. Michel and N. Wehn. Turbo-Decoder Quantization for UMTS. *IEEE Communications Letters*, 2001. Accepted.

[8] P. Robertson, P. Höher, and E. Villebrun. Optimal and Sub-Optimal Maximum a Posteriori Algorithms Suitable for Turbo Decoding. *ETT*, 8(2):119–125, March–April 1997.

[9] C. Schurgers, M. Engels, and F. Catthoor. Energy Efficient Data Transfer and Storage Organization for a MAP Turbo Decoder Module. In *Proc. ISLPED '99*, pages 76–81, Aug. 1999.

[10] F. Viglione et al. A 50 Mbit/s Iterative Turbo-Decoder. In *Proc. DATE '00*, pages 176–180, Mar. 2000.

[11] J. Vogt, K. Koora, A. Finger, and G. Fettweis. Comparison of Different Turbo Decoder Realizations for IMT-2000. In *Proc. Globecom '99*, pages 2704–2708, Dec. 1999.

[12] A. Worm et al. Advanced Implementation Issues of Turbo-Decoders. In *Proc. 2nd International Symposium on Turbo Codes & Related Topics*, pages 351–354, Sept. 2000.

[13] A. Worm, H. Lamm, and N. Wehn. A High-Speed MAP Architecture with Optimized Memory Size and Power Consumption. In *Proc. SiPS 2000*, pages 265–274, Oct. 2000.