LUND UNIVERSITY

Area and power efficient trellis computational blocks in 0.13m CMOS

Kamuf, Matthias; Öwall, Viktor; Anderson, John B

Published in:
IEEE International Symposium on Circuits and Systems (ISCAS)

DOI:
10.1109/ISCAS.2005.1464595

2005

Link to publication

Citation for published version (APA):
Kamuf, M., Öwall, V., & Anderson, J. B. (2005). Area and power efficient trellis computational blocks in 0.13µm CMOS. In *IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 344-347). IEEE - Institute of Electrical and Electronics Engineers Inc.. https://doi.org/10.1109/ISCAS.2005.1464595

Total number of authors:
3

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Area and Power Efficient Trellis Computational Blocks in 0.13μm CMOS

Matthias Kamuf and Viktor Öwall
Department of Electroscience
Lund University
SE-221 00 Lund, Sweden
Email: {mkf, vikt}@es.lth.se

John B. Anderson
Department of Information Technology
Lund University
SE-221 00 Lund, Sweden
Email: anderson@it.lth.se

***Abstract*— Improved add-compare-select and branch metric units are presented to reduce the complexity in the implementation of trellis-based decoding architectures. These units use a complementary property of the best rate $1/2$ convolutional codes to reduce both area requirements and power consumption in a silicon implementation with no loss in decoding performance. For a 0.13μm CMOS process, synthesized computational blocks for decoders that can process codes from memory 2 up to 7 show up to 17% savings in both cell area and power consumption.**

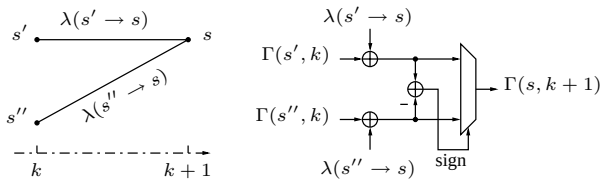
I. Introduction

Trellis-based decoding is a popular method to recover encoded information corrupted during transmission over a noisy channel. For example, the Viterbi algorithm (VA) [1] and the BCJR algorithm [2] are two schemes that work on an underlying trellis description of the encoded sequence. Note that the BCJR in the logarithmic domain (logMAP algorithm [3]) is commonly used in turbo decoding schemes.

Basic computations in either algorithm involve branch metric (BM) calculations and add-compare-select (ACS) operations. In case of the VA, an ACS operation successively discards branches that are not part of the survivor path. In case of the logMAP, this operation corresponds to an Add-MAX* operation [4] to recursively calculate forward and backward state metrics. However, this is basically an ACS operation with an added offset (ACSO) to correct for the Jacobian logarithm. Thus, the presented results for the ACS hold for the ACSO as well.

Almost all good rate $1/n$ convolutional codes, n an integer, have the property that the code symbol labels on the two branches into each trellis node are complementary. In an earlier paper [5], we used this property and presented architectural simplifications for these units. Here, we apply this idea to see how these arithmetic savings translate into area and power savings in a silicon implementation.

We start by briefly reviewing notation and necessary modifications in the following section. Then, architectural issues for a hardware realization are addressed in Section III. We also present a variation that trades area for speed. As a case study, a computational kernel for Viterbi decoding is synthesized. The synthesis results in Section IV confirm the benefits of these approaches compared to a traditional setup. Also, power savings are estimated at gate level.



(a) (b)

Fig. 1. A trellis node (a) and its respective ACS computational unit (b) for a rate $1/n$ code. Two additions and one comparison (subtraction) are needed to determine the new state metric.

II. Notation and simplification

A channel symbol is quantized with q bits and denoted $y_i \in [0, 2^q - 1]$ for $i = 0, \ldots, n-1$. This symbol is the output of a discrete memoryless channel with binary input x_j and transition probabilities $P(y_i|x_j)$. The expected code symbol $c_i(s' \to s)$ along the branch from state s' to state s is derived by the mapping $x_0 \mapsto 0$ and $x_1 \mapsto 2^q - 1$.

The metric $\lambda(s' \to s)$ denotes the likelihood that a transition from state s' to s occurred at time k, see Fig. 1(a). In the additive white Gaussian noise channel the optimal distance measure is the squared Euclidean distance between the expected code symbols and the received noisy channel symbols. However, given the preceding symbol constraints this measure simplifies to a superposition of n channel symbols using

$$\lambda_i(s' \to s) = \begin{cases} y_i, & c_i(s' \to s) = 0 \\ (2^q - 1) - y_i, & c_i(s' \to s) = 2^q - 1 \end{cases} \quad (1)$$

and the complete branch metric becomes

$$\lambda(s' \to s) = \sum_{i=0}^{n-1} \lambda_i(s' \to s). \quad (2)$$

The ACS operation is expressed as

$$\begin{aligned} \Gamma(s, k+1) = \min\{&\Gamma(s', k) + \lambda(s' \to s), \\ &\Gamma(s'', k) + \lambda(s'' \to s)\} \end{aligned} \quad (3)$$

and illustrated in Fig. 1(b). $\Gamma(s, k+1)$ is the updated metric of state s at time $k+1$, based on the preceding state metrics,

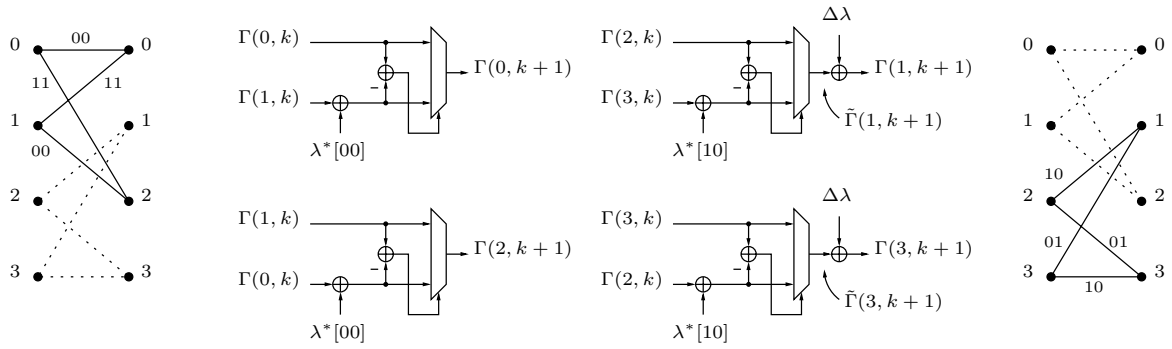


Fig. 3. Improved ACS setup for a (7,5) code. The respective trellis nodes are shown on either side together with the expected symbol labels $[x_0\, x_1]$ along the branches. $\lambda[00]$ was subtracted from all updated state metrics.

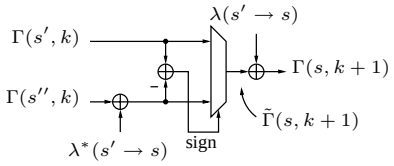


Fig. 2. Transformed ACS unit for a rate $1/2$ code. This unit needs one addition less to determine the outcome of the comparison, $\tilde{\Gamma}(s, k+1)$.

$\Gamma(s', k)$ and $\Gamma(s'', k)$ at time k, and the respective metrics $\lambda()$ along the branches $(s' \to s)$ and $(s'' \to s)$.

We consider rate $1/2$ convolutional codes, $n = 2$, whose code symbol labels on the two branches into each trellis node are complementary. Since the branch metrics linearly depend on the code symbols they also share this property and one branch metric can be expressed by means of the other, that is,

$$\begin{aligned}\lambda(s'' \to s) &= 2\,(2^q - 1) - \lambda(s' \to s) \\ &= \lambda(s' \to s) + 2\,[(2^q - 1) - \lambda(s' \to s)].\end{aligned} \tag{4}$$

We define the modified branch metric $\lambda^*(s' \to s)$, which is a signed number, as

$$\lambda^*(s' \to s) \equiv 2\,[(2^q - 1) - \lambda(s' \to s)]. \tag{5}$$

This expression is calculated by bit-inverting $\lambda(s' \to s)$ with the most significant bit (MSB) excluded followed by a left-shift. Substituting (5) together with (4) into (3) and taking out the common factor $\lambda(s' \to s)$ from the comparison we get

$$\begin{aligned}\Gamma(s, k+1) = \lambda(s' \to s) + \\ \min\{\Gamma(s', k), \Gamma(s'', k) + \lambda^*(s' \to s)\}.\end{aligned} \tag{6}$$

Introducing $\tilde{\Gamma}(s, k+1)$ as the new outcome of the $\min$ operation, see Fig. 2, (6) becomes

$$\Gamma(s, k+1) = \lambda(s' \to s) + \tilde{\Gamma}(s, k+1). \tag{7}$$

Compared to (3) there is one addition less needed to determine the outcome of the comparison. In order to retain the numerical relation between interconnected state metrics in a trellis with different $\lambda()$ we have to add this factor after having determined $\tilde{\Gamma}(s, k+1)$. However, one can subtract this factor from all state metrics and it will be shown that in that case half the ACS units do not need this correction, that is, $\Gamma(s, k+1) = \tilde{\Gamma}(s, k+1)$.

III. Improved architectures

The branch metric $\lambda(s' \to s)$ can take four different values for a rate $1/2$ code, namely $\lambda[x_0\, x_1]$ for every possible combination of symbols $x_j \in \{0, 1\}$. The complementary metrics to $\lambda[00]$ and $\lambda[10]$ that are needed in a conventional ACS unit are $\lambda[11]$ and $\lambda[01]$, respectively. In the improved approach only two metrics are needed since the other two can be calculated according to (4), that is, $\lambda[11]$ is expressed by $\lambda[00]$ and $\lambda[01]$ by $\lambda[10]$. The factor $\lambda(s' \to s)$ of Fig. 2 to be added in an ACS unit is therefore either $\lambda[00]$ or $\lambda[10]$.

From the preceding considerations the hardware savings become apparent by looking at an example, an ACS unit setup for decoding a (7,5) code in Fig. 3. In this picture, the state metric corrections in the two ACS units on the left become obsolete since $\lambda[00]$ was subtracted from all updated state metrics. The correction factor to be used in the two ACS units on the right is then

$$\begin{aligned}\Delta\lambda &= \lambda[10] - \lambda[00] \\ &= (2^q - 1) - 2\,y_0.\end{aligned} \tag{8}$$

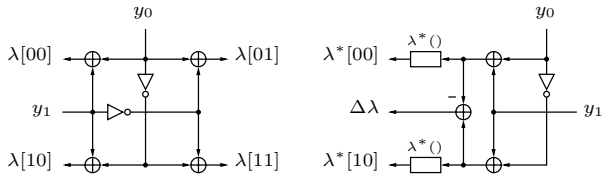
Consequently, for rate $1/2$ codes that have the complementary property half the ACS units save one addition compared to a conventional setup. Therefore, the number of required additions is $5 \cdot 2^{m-1}$, where m is the encoder memory. Compared to a conventional ACS unit setup ($3 \cdot 2^m$ additions), the reduction in arithmetic complexity is 17%.

The necessary distance measures $\lambda[\,]$ are provided by BM units. Fig. 4 shows both a conventional and the improved BM unit. The former (a) needs four additions and two inverters to calculate the four branch metrics. The latter (b) only needs two additions and three inverters to calculate two branch metrics considering that the calculation of $\Delta\lambda$ can be simplified with $y_0 = (2^q - 1) - \overline{y}_0$ and therefore (8) becomes

$$\begin{aligned}\Delta\lambda &= 2\,\overline{y}_0 - (2^q - 1) \pmod{2^q} \\ &= 2\,\overline{y}_0 + 1.\end{aligned} \tag{9}$$

This expression is calculated by a left-shift of y_0 followed by a bit-inversion (MSB excluded).

Note that the critical path in half the ACS units is increased by the delay of an addition. However, this problem is solved



(a) (b)

Fig. 4. Conventional (a) and modified (b) BM unit for a rate $1/2$ code. Note that the subtraction for calculating $\Delta\lambda$ simplifies to a left-shift followed by a bit-inversion of y_0 (MSB excluded), see (8) and (9).

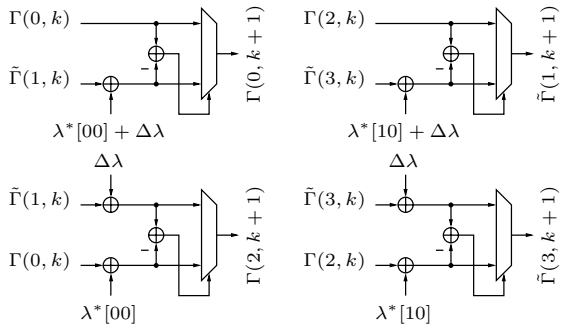


Fig. 5. Retimed ACS setup for a (7,5) code. Note that $\Delta\lambda$ in this case is delayed by one clock cycle in the BM unit.

by delaying the correction into the next computation cycle and the original critical path of the ACS unit is maintained. In this case, the additions for the correction that are after the multiplexers in Fig. 3 move into different ACS units into the comparison path instead, see the retimed ACS unit in Fig. 5. Besides storing $\Delta\lambda$ in the BM unit, two new correction factors $\lambda^*[00]+\Delta\lambda$ and $\lambda^*[10]+\Delta\lambda$ are needed for this architecture. These additions are not carried out in the ACS units since this again would increase the critical path. They are instead precalculated in the BM unit; the complexity is moved from the ACS units to the BM unit which is instantiated only once instead of 2^m times. Fig. 6 shows this BM unit which is the BM unit from Fig. 4(b) appended with two additions and a register. If the extra delay introduced by these additions can not be tolerated, the datapath can always be pipelined since it is purely feedforward.

IV. Implementation and Synthesis Results

In this case study, we implemented Viterbi computational blocks for best feedforward rate $1/2$ convolutional codes [6] up to memory $m = 7$. The output of an ACS unit is both the surviving path into the respective state and the updated state metric, see Fig. 7. We neglect survivor memory management since this part of the decoder does not differ between the conventional and improved architectures. The BM and ACS units are described in a VHDL model at register-transfer level based on generic parameters.

The well-known state metric normalization techniques are still valid since differences among the state metrics remain the same. A modulo normalization scheme is used and the state metrics become

$$\lceil \log_2\{2\,(2^q-1)(m+1)\}\rceil \tag{10}$$

bits wide. The comparison in the ACS unit is implemented with the modified comparison rule [7]. Channel symbol wordlength is $q = 3$ since this gives negligible degradation in decoding performance compared to infinite precision.

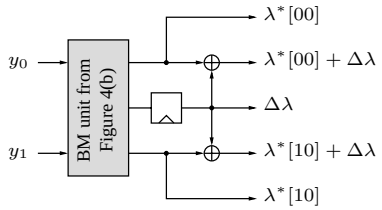


Fig. 6. BM unit for the ACS setup in Fig. 5.

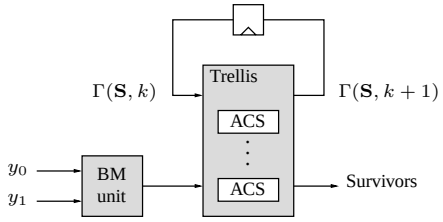


Fig. 7. Block diagram of the model in the case study, $\mathbf{S} = [0 \ldots 2^m - 1]$.

We used a design kit from Virtual Silicon for the UMC[1] 0.13μm CMOS process. Power figures were obtained by Synopsys Power Compiler using toggle information from a gate level simulation run with state- and path-dependent cell information, and random input stimuli. Both dynamic (switching and short-circuit power) and leakage power are included in the results. However, it turns out that the contribution from leakage power is negligible in this study. At this design stage, it is assumed that the contribution from clock tree and interconnection, which is relevant for absolute area and power numbers, is the same in both architectures since we are only interested in the relative savings between architectures.

The improved versions are compared to their respective conventional setup with regard to their application area. For applications with relaxed timing requirements, area and power comparisons are done for the architecture in Fig. 3 together with the respective BM units. Synthesis tests showed that the area-delay product curve is flat down to a delay of about 3.5 ns, which is set as a constraint to the critical path. The power simulation is carried out at a clock frequency of f_{clk}=250MHz. For the retimed architecture of Fig. 5 this delay reaches further down to 2 ns. Here, we only synthesized the ACS units in order to investigate the impact of the saved adder in every unit. For the power simulation f_{clk}=400MHz is assumed.

Table I lists the synthesis results of the cell area for a conventional ACS setup together with the BM unit from

[1]United Microelectronics Company

m	2	3	4	5	6	7
Area (μm^2)	3697	6876	15120	29723	58930	115974

TABLE I

CELL AREA FOR A CONVENTIONAL BM/ACS SETUP.

m	2	3	4	5	6	7
Area (μm^2)	2923	5846	13328	26657	53315	106601

TABLE II

CELL AREA FOR A CONVENTIONAL ACS SETUP.

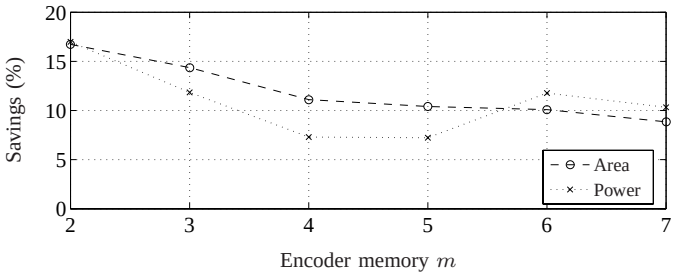


Fig. 8. Area and power comparison (@V_{dd}=1.2V, f_{clk}=250MHz) between a conventional (Table I) and the improved BM/ACS setup from Fig. 3 and 4(b).

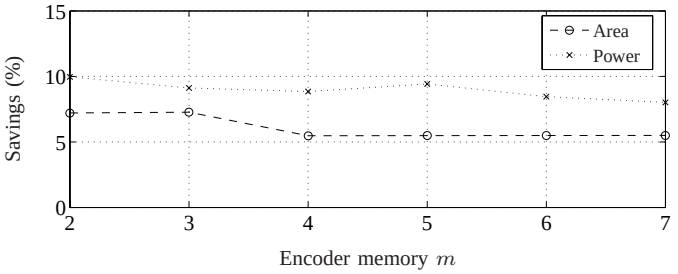


Fig. 9. Area and power comparison (@V_{dd}=1.2V, f_{clk}=400MHz) between a conventional (Table II) and the retimed ACS setup from Fig. 5.

Fig. 4(a). In comparison with it, Fig. 8 shows the possible savings in cell area and power consumption when the improved architecture from Fig. 3 is employed. As mentioned earlier, the arithmetic complexity is reduced by 17%, which is true for $m = 2$. However, with increasing m the percental savings decrease since both area and power overhead introduced by the registers gets bigger. At $m = 4$, the state metric register wordlength is increased by one bit, refer to (10). Thereafter the combinational power savings catch up with this initial penalty and again reach 12% at $m = 6$.

Fig. 9 shows the comparison results when the setup from Fig. 5 is used. Note that compared to Fig. 8 the power figures were obtained at a higher clock frequency due to the shorter critical path. Also, the adders incorporating the correction factors in the ACS units on the top are one bit bigger than the ones in the conventional architecture. Again, the improved setup saves both area and power with 7% and 10%, respectively.

If speed requirements allow use of the computational kernel in a time-multiplexed fashion, the savings increase compared to a parallel implementation; for example, for $m = 6$, there are achievable savings of 10% in cell area, however, a time-multiplexed architecture using a $m = 2$ kernel could gain an extra 7%.

It should be pointed out that in the above comparison optimization steps from the synthesis tool are suppressed, thus preserving the proposed architectural structures. The designs use the same register cells and combinational logic blocks, that is, adders and comparators, are implemented in ripple-carry style. However, it turns out that enabling optimization does not alter the achieved results.

Finally, the results can also be applied to variations of the synthesized kernel such as in a logMAP decoder. The hardware effort of the look-up table that corrects for the Jacobian logarithm in an ACSO is the same in both implementations and hence introduces a constant overhead. Since this decoder is commonly used in turbo decoding schemes, the encoder memory does not exceed 4 and hence cell area savings can be as high as 12% per computational unit.

V. CONCLUSIONS

We showed that both area requirements and power consumption of trellis computational kernels can be reduced by making use of the complementary code symbol property of all good rate $1/2$ convolutional codes. In our case study, area savings vary between 17% and 9% and power savings from 17% to 7% are reported in a 0.13μm CMOS process. Furthermore, iterative decoding schemes can easily employ this simplification since their logMAP decoders are principally based on the same computational kernel.

ACKNOWLEDGMENTS

This project is supported by the Swedish Socware program, the EU Pacwoman project, and the Competence Center for Circuit Design at Lund University.

REFERENCES

[1] G. D. Forney, Jr., "The Viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, Mar. 1973.

[2] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Transactions on Information Theory*, vol. 20, no. 2, pp. 284–287, Mar. 1974.

[3] P. Robertson, E. Villebrun, and P. Höher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *Proc. IEEE International Conference on Communications (ICC)*, Seattle, WA, June 1995, pp. 1009–1013.

[4] E. Boutillon, W. J. Gross, and P. G. Gulak, "VLSI architectures for the MAP algorithm," *IEEE Transactions on Communications*, vol. 51, no. 2, pp. 175–185, Feb. 2003.

[5] M. Kamuf, J. B. Anderson, and V. Öwall, "A simplified computational kernel for trellis-based decoding," *IEEE Communications Letters*, vol. 8, no. 3, pp. 156–158, Mar. 2004.

[6] R. Johannesson and K. S. Zigangirov, *Fundamentals of Convolutional Coding*. Piscataway, NJ: IEEE Press, 1999.

[7] C. B. Shung, P. H. Siegel, G. Ungerboeck, and H. K. Thapar, "VLSI architectures for metric normalization in the Viterbi algorithm," in *Proc. IEEE International Conference on Communications (ICC)*, vol. 4, Atlanta, GA, Apr. 1990, pp. 1723–1728.