LUND UNIVERSITY

Architectural considerations for rate-flexible trellis processing blocks

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

Published in:
2005 IEEE 16th International Symposium on Personal, Indoor and Mobile Radio Communications

DOI:
10.1109/PIMRC.2005.1651606

2005

Link to publication

Citation for published version (APA):
Kamuf, M., Öwall, V., & Anderson, J. B. (2005). Architectural considerations for rate-flexible trellis processing blocks. In *2005 IEEE 16th International Symposium on Personal, Indoor and Mobile Radio Communications* (pp. 1076-1080). IEEE - Institute of Electrical and Electronics Engineers Inc..
https://doi.org/10.1109/PIMRC.2005.1651606

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

Architectural Considerations for Rate-flexible Trellis Processing Blocks

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*— A flexible channel decoding platform should be able to operate in varying communication scenarios, and different code rates have to be supported. Therefore, we present a framework that allows efficient processing of rate-flexible trellises. Using a fundamental computational unit for trellis-based decoding, formal principles are obtained to emulate more complex trellises. In a design example, such a computational block supports both rate $1/c$ convolutional codes and set partition codes with subset selectors of rate $2/3$. Synthesis results show the hardware requirements for two different architectural approaches.**

I. Introduction

With growing application diversity in mobile communications, flexible processing hardware has become crucial. For example, a flexible channel decoding platform should be able to handle at least two decoding modes, one when a high error-correcting capability is required at low E_b/N_0 and another one supporting high data throughput if there are good channel conditions. Furthermore, both modes should be utilizing the same computational kernel to minimize hardware overhead compared to two single processing blocks.

The Viterbi algorithm (VA) is used, among others, to recover encoded information corrupted during transmission over a noisy channel. Its processing complexity increases both with the number of trellis states N and the number of branches connecting these states in one stage. The heart of this algorithm is the add-compare-select (ACS) operation that successively discards suboptimal branches from a code trellis.

Flexible Viterbi decoding processors were studied and presented, for example, in [1] and [2]; however, they were intended solely for use with rate $1/c$ binary convolutional codes, c an integer. These codes are used in a scenario, where the available E_b/N_0 is limited due to disadvantageous channel conditions. The corresponding trellis diagram can be decomposed into a radix-2 (R2) butterfly state interconnect structure depicted in Fig. 1(a). To reduce bandwidth expansion, higher rate codes up to rate 1 are obtained by puncturing the basic $1/c$ code, which preserves the R2 structure of the trellis.

With the introduction of Wireless Personal Area Networks (WPANs) [3], trellis-coded modulation (TCM) has regained attraction as a means of transmitting information at high data rates. This scheme is most efficient for higher (quadrature) constellations beyond QPSK, which carry more than two

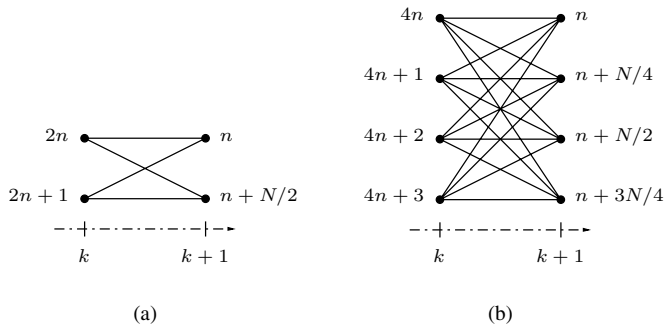

Fig. 1. A radix-2 butterfly (a) and a radix-4 butterfly (b) with state labels for encoders in controller form. N is the number of states in the trellis and $n \in [0, N/2^b - 1]$ in the respective cases, $b = 1, 2$.

information symbols per channel use, thus enabling high data throughput. It makes extensive use of subset selectors that are based on systematic rate $b/(b+1)$ convolutional codes [4]. To date, the most practical codes for TCM used together with a 2-dimensional modulation scheme appear for $b = 2$. The trellis now consists of radix-4 (R4) butterflies, see Fig. 1(b). That is, there are $2^b = 4$ branches leaving and entering each state. Puncturing, however, is not applicable here if code performance is to be fully maintained. This degradation stems from altering the minimum inter-subset distance. Thus, R4 butterflies have to be processed in the decoding steps.

Summarizing these considerations, a flexible channel decoding architecture has to be tailored to efficiently process both R2 and R4 butterflies, while limiting overhead in both area and power consumption.

We propose to map higher-radix butterflies onto a basic R2 butterfly and process them in a time-multiplexed manner, where hardware is reused at the cost of throughput. As an example, we consider the best 8-state rate $1/c$ convolutional codes and the specific TCM code proposed in [3]. It will be shown that only slight modifications are needed to extend the basic R2 architecture. Finally, it should be remarked that almost all practical trellis codes, whether TCM or not, are based on either R2 or R4 butterflies, so the principles that we present cover these codes as well.

In Section II and III, we provide the theoretical basis for rate-flexible processing and the resulting building blocks are

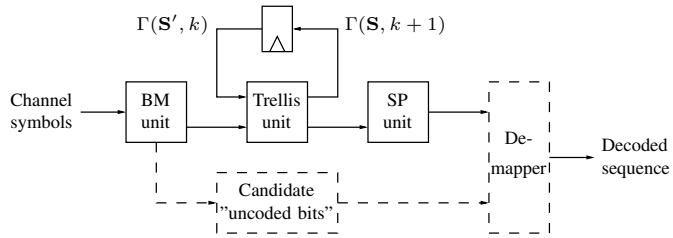

Fig. 2. Block structure of a Viterbi decoder. Additional parts needed for decoding of TCM codes are dashed.

described in Section IV. Two approaches are considered here, an R2-based architecture in Section IV-A and an R4-based one in Section IV-B. The synthesized decoding blocks are then evaluated in the area-time domain in Section V.

II. VA in a nutshell

For the scope of this paper, we briefly revisit the basic building blocks used in the VA. A more thorough, hardware-oriented description is found in [5]. As shown in Fig. 2, there are three main processing blocks in a Viterbi decoder. The branch metric (BM) unit provides measures of likelihood for the transitions in a trellis. These measures are consumed by the trellis unit, where ACS operations on the state metrics $\Gamma(\mathbf{S}', k)$ from instant k form a new set of state metrics $\Gamma(\mathbf{S}, k+1)$ at instant $k+1$. Here, $\mathbf{S}'$ denotes the set of states in a trellis and $\mathbf{S}$ is a permutation of $\mathbf{S}'$ according to the given state interconnection.

The decisions from the trellis unit are collected in the survivor path (SP) unit to reconstruct the information bits that caused the transitions in the survivor path. Additionally, in case of TCM, candidate "uncoded bits" have to be stored that together with the reconstructed information bits form the final decoded sequence.

III. Multi-radix processing

We now turn to the issue of rate-flexibility in the trellis unit and derive a framework for emulating R4 butterflies by means of R2 butterflies. For illustration we take the example of a rate $1/c$ convolutional code combined with a TCM code which subset selector is of rate $2/3$.

A. Considered trellises

The base architecture for the flexible trellis block is an R2 architecture for an 8-state rate $1/c$ convolutional code, that is, the trellis consists of butterflies with state transitions according to Fig. 1(a). Rate $1/2$ codes are often used in today's communication systems since these rates are a good compromise between coding gain and bandwidth efficiency. Also, they usually serve as mother code for high-rate punctured codes.

For the TCM code, we consider signal constellations and subset partitioning as proposed in [3]. There are 8 signal subsets, that is, 8 different branch metrics are to be distributed to the trellis unit that for this 8-state code consists of two R4 butterflies. Half of the branch metrics are applied to the first R4 butterfly and the other half to the second one.

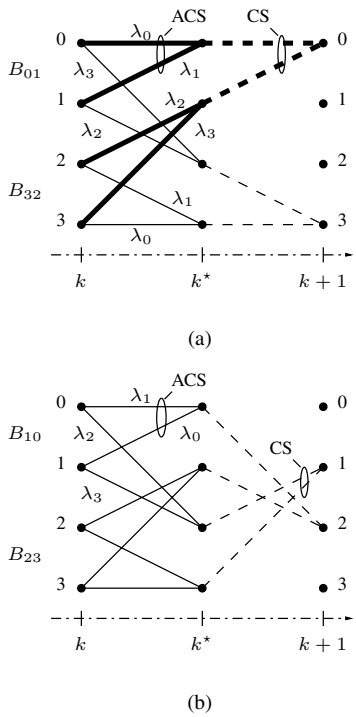

Fig. 3. A decomposed radix-4 butterfly and the two partial computations leading to the updated metrics for states 0 and 3 in (a) and 1 and 2 in (b). As an example, the necessary operations to update state metric 0 are drawn bold in (a).

A TCM encoder is usually realized in observer form since this maintains the constraint length of the underlying code if $b > 1$. However, unlike a controller form realization, which increases the encoder state space from N to $N^{2^{b-1}}$, the state transitions now depend on the coding polynomial. That is, the feedback network has to be flexible if one wants to process both R2- and R4-based codes on a single R2 architecture. This can be done by introducing additional routing resources that modify the permutation in R4 mode to feed back the state metrics in correct order. In this design example, though, the state transitions of the TCM encoder allow the reuse of the trellis feedback connections of the binary $1/c$ code, that is, the permutation transforming $\mathbf{S}'$ into $\mathbf{S}$ is the same in both R2 and R4 mode.

B. Processing framework

In the following, we consider an R4 butterfly which utilizes a set of branch metrics λ_i for $i = 0 \ldots 3$. As to Fig. 1, there are $N/2^b$ butterflies in a trellis, and thus, for $N = 4$ and $b = 2$, there is only one R4 butterfly. Since $n \in [0, N/2^b - 1]$, that is, $n = 0$ in this case, its state labels become $0, \ldots, 3$ as in Fig. 3.

To update one state in an R4 butterfly, one can carry out all six possible partial comparisons in parallel [6]. Four such operations are needed to calculate a complete R4 butterfly as in Fig. 1(b). However, this is not efficient when it comes to hardware reusability in a rate-flexible system due to the arithmetic overhead introduced by such 4-way ACS units.

Instead, we present a way of processing an R4 butterfly given an architecture in which state metrics are updated by means of R2 butterfly units.

Generally, a 4-way ACS can be carried out in two successive steps: first evaluating and discarding a pair of cumulative metrics (ACS), then in the second step, discarding one of the surviving metrics, which corresponds to a compare-select (CS) operation. This procedure is visualized by decomposing the R4 butterfly from Fig. 1(b) into two stages as in Fig. 3. Furthermore, a single R4 stage is split into two R2 butterflies to achieve the cumulation of the state metrics with all four branch metrics, resulting in the structures shown in Fig. 3(a) and (b). Here, the branch metrics are assigned according to the considered TCM subset selector.

To capture these processing steps formally we need the following definition. The state connectivity of an R2 butterfly is defined in Fig. 1(a). Assume that the two states at time k are named u' and v' with state metrics $\Gamma(u', k)$ and $\Gamma(v', k)$, respectively. The two ACS operations leading to two updated state metrics for states u and v at stage $k+1$ are expressed as butterfly operation $B_{u'v'}$. Without loss of generality, the λ_i are distributed as in

$$B_{u'v'} = \begin{pmatrix} \Gamma(u, k+1) \\ \Gamma(v, k+1) \end{pmatrix} = \begin{pmatrix} \min \begin{pmatrix} \Gamma(u', k) + \lambda_0 \\ \Gamma(v', k) + \lambda_1 \end{pmatrix} \\ \min \begin{pmatrix} \Gamma(v', k) + \lambda_2 \\ \Gamma(u', k) + \lambda_3 \end{pmatrix} \end{pmatrix}. \quad (1)$$

It is seen from Fig. 3 that there are four such R2 butterflies between k and $k^\star$, so four operations as in (1) are needed, given that this butterfly operation applies to an update at stage $k^\star$ instead of $k+1$ in this case. For example, B_{01} is shown in Fig. 3(a), that is, $u' = 0$ and $v' = 1$.

Processing the R4 butterfly based on (1) preserves the compatibility with the base R2 architecture. The scheme for obtaining all partial survivors is then expressed as

$$\mathbf{B}' = \begin{pmatrix} B_{01} & B_{10} \end{pmatrix}, \quad (2)$$

where the columns determine the instance of an iteration. So far we have only computed half of the partial survivors needed; to complete the R4 butterfly another unit has to carry out

$$\mathbf{B}'' = \begin{pmatrix} B_{23} & B_{32} \end{pmatrix}. \quad (3)$$

The operations in (2) and (3) guarantee that all state metrics at stage k are added to all branch metrics, that is, 16 partial sums are reduced to 8 partial survivors at intermediate stage $k^\star$ by means of CS operations. The final surviving state metrics at stage $k+1$ are obtained by CS operations on the hitherto surviving metric pairs. Note that the partial survivors are not altered and therefore the final state metrics are not changed compared to a straightforward implementation.

IV. Architectural issues

The model used in this study is a trellis unit that updates N states by means of $N/4$ processing elements (PEs), each consuming and producing 4 state metrics. A PE is configured with butterfly units (BF) of different radices as in Fig. 4.

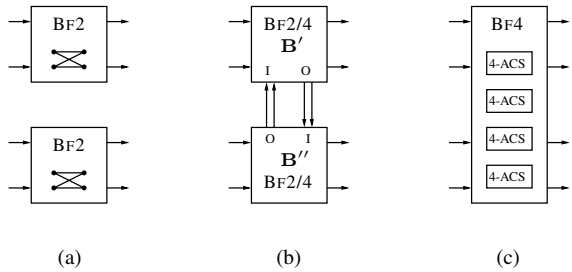

Fig. 4. Butterfly units that are instantiated inside a processing element. A setup as in (a) supports only R2 processing, setups (b) and (c) are rate-flexible.

Either two BF2, two rate-flexible BF2/4 units, or one BF4 unit are employed. Note that the BF4-based architecture can also be configured for rate-flexible processing, whereas a BF2-based design is solely intended for R2 processing and is not discussed further. For a description of these approaches, see [5] and [6].

A. R2-based approach

In the rate-flexible architecture using BF2/4, all partial survivors are calculated during two cycles, and in the third cycle the final update takes place. Two adjacent butterflies are interconnected to exchange the partial survivors according to the desired ordering of state metrics. As an example, the operations to update state metric 0 are drawn bold in Fig. 3(a). The partial survivors needed for the final CS are created in $\mathbf{B}'$ at instance 0 and in $\mathbf{B}''$ at instance 1. Since they reside in different butterfly units, they have to be brought together by means of I/O channels as indicated in Fig. 4(b). The partial survivors have to be stored temporarily and routed for the final CS according to the required ordering of the updated state metrics.

1) Butterfly unit BF2/4*:* Fig. 5(a) shows the rate-flexible butterfly unit BF2/4. Its arithmetic components, that is, the adders and the CS units, are identical to the ones in a BF2 unit, whereas its critical path is slightly larger due to the multiplexers. To cope with a decomposed R4 butterfly, routing resources (shaded in gray) are provided to distribute the partial survivors as dictated by the branch metric distribution and the state transitions. The input multiplexers shuffle the two input state metrics to guarantee their cumulation with all four branch metrics. The multiplexers in front of the CS units select whether the partial survivors at stage $k^\star$ are to be captured into the routing unit PERM, or the final comparison at stage $k+1$ is to be performed. When carrying out (2) or (3), PERM is fed during two cycles and in the third and final cycle the partial survivors are compared. Here, the signals I and O provide the connections to the adjacent butterfly unit to carry out the comparison with the desired pairs of partial survivors. For example, O_0 is connected to I_0 in the adjacent butterfly. The global registers for the surviving state metrics $\Gamma(\mathbf{S}, k+1)$ in R4 mode are only latched every third cycle. The CS operations between $(k, k^\star)$ and $(k^\star, k+1)$ in Fig. 3 are executed in the same CS unit, thus saving hardware. Overall, this unit adds

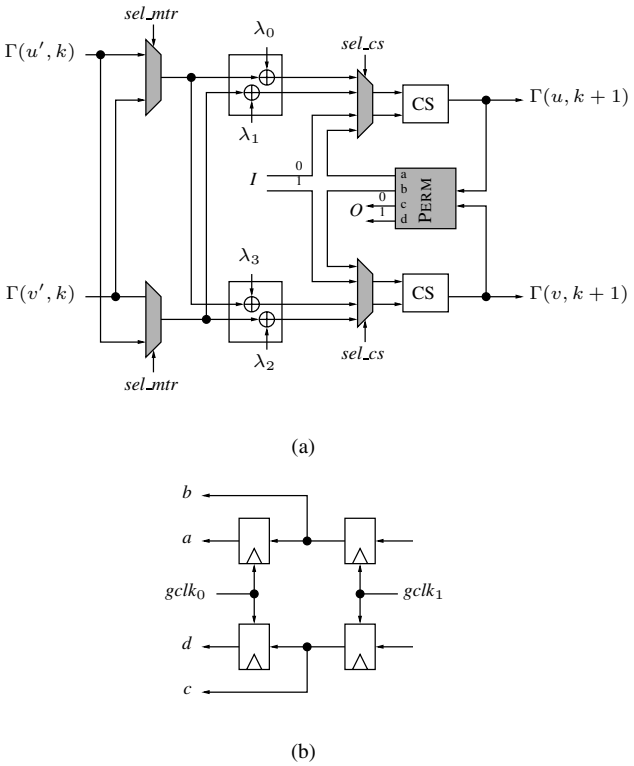

Fig. 5. The flexible butterfly unit BF2/4 in (a). Two such units are needed for an R4 butterfly to be updated in three clock cycles. The shaded blocks are the overhead compared to a BF2 unit. In (b) the routing block PERM is shown. Connections apply for the design example.

four multiplexers and four registers ontop of a BF2 unit, while there is no arithmetic overhead.

2) Routing block PERM*:* The block needed for permutating the partial sums is depicted in Fig. 5(b). It consists of two tapped delay lines. The registers are only clocked when their input data are valid. Since their data are only read when valid, that is, every third clock cycle, they are implemented without reset.

In this design example, PERM carries out the same permutation in both butterflies, that is, the partial survivors in the top rail, a and b, are devoted to the same butterfly unit, whereas the bottom rail survivors, c and d, are routed to the adjacent butterfly unit, see Fig. 4(b). Given (2) and (3), this setup arranges the state metrics in an increasing sequence from 0 to 3, and the design fits seamlessly into the base architecture, that is, the feedback network in Fig. 2 is reused as is.

However, the required order of state metrics can vary for different trellises, depending on the encoder. One can extend this reasoning and notice that the butterflies $B_{u'v'}$ in (2) and (3) can be calculated in any order and the partial survivors needed for the updates are shuffled accordingly. Nevertheless, PERM is highly flexible by providing $4! = 24$ possible permutations to handle these cases. With this approach it is possible to cover a wide range of encoders. If the decoder implementation is to realize different encoders with different routing requirements, PERM has to be programmable.

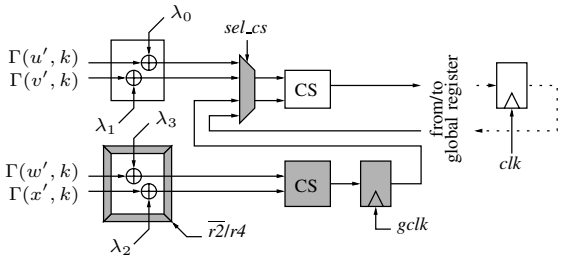

Fig. 6. A flexible 4-way ACS unit for use in BF4 of Fig. 4(c). Four such units are needed for an R4 butterfly to be updated in two clock cycles. The shaded blocks are the overhead compared to a 2-way ACS unit.

It was already mentioned that the permutation transforming $\mathbf{S}'$ into $\mathbf{S}$ is the same for both codes considered. This is not generally true, and additional routing is necessary if the required permutation cannot be obtained by programming the PERM units. On a global level, metrics have to be exchanged between PEs. Essentially, this is done by inserting multiplexers to either choose the R2 or R4 state connections. Depending on the application this can be fixed or programmable.

B. R4-based approach

We compare the presented flexible R2-based approach to an R4 architecture based on BF4 units that utilize four 4-way ACS units as in Fig. 4(c). To account for the intended use in a rate-flexible system, similar control mechanisms have to be provided as in the R2-based approach. Hence, a straightforward two-level-CS implementation is considered. Depending on the desired throughput, a butterfly can be updated in one or two clock cycles, which gives the well-known area–throughput trade-off. Here, we employ a two-cycle update since this maintains the critical path of the R2-based approach and one CS unit can be reused.

Fig. 6 shows the flexible 4-way ACS unit. In R4 mode, the two partial survivors are captured in the first cycle. The global state metric register in the upper path now carries a temporary survivor at time $k^\star$. In the second cycle, these survivors are compared to yield the final state metric at $k+1$. In R2 mode, only the upper ACS path is utilized and to be equally power-efficient, one needs to prevent switching activity in the lower ACS path. This is done by guarding the inputs of the adders with AND-gates, which is illustrated by the gray shading. The static signal $\overline{r2}/r4$, which determines the processing mode, controls whether the addition block is enabled or not. Compared to a conventional 2-way ACS unit, two adders, a CS unit, a register, and a multiplexer are counted as overhead.

Table I shows the necessary hardware to update an R4 butterfly for the two approaches. The R4-based approach lags in terms of arithmetic overhead, that is, adders and CS units. Routing and storing resources, though, dominate in the R2-based approach. The arithmetic overhead weighs more in this case, considering that a CS unit consists of an adder and a multiplexer. Thus, the number of equivalent multiplexers is 16 in both cases, and the number of adders increases to 12 and

TABLE I

NECESSARY HARDWARE TO UPDATE AN R4 BUTTERFLY FOR THE TWO APPROACHES. NUMBER OF MULTIPLEXERS IS EXPRESSED AS NUMBER OF EQUIVALENT 2:1 MULTIPLEXERS. NOTE THAT A CS UNIT CONSISTS OF AN ADDER AND A MULTIPLEXER.

	R2-based	R4-based
Adder	8	16
Multiplexer	12	8
CS	4	8
Register	8	4

24, respectively. Based on these premises, the two approaches are expected to have comparable area requirements.

V. EVALUATION OF SYNTHESIZED DECODING BLOCKS

To test the actual hardware implementations, we used a design kit from Faraday for the UMC 0.13μm CMOS process. All evaluations apply to synthesized cell area, where different throughput requirements are put as design constraints. A modulo normalization technique [7] is used for controlled state metric overflow and we assume 8 bits as wordlength for the state metrics.

Fig. 7 shows the required cell area for synthesized computational blocks that can process both R2 and R4 butterflies. Here, $t_{k\rightarrow k+1}$ stands for processing time for a trellis stage from k to $k+1$. The values are compensated since the BF2/4-based architecture takes 3 cycles for an R4 update, whereas the BF4-based one only needs 2 cycles. For an R2 update, both architectures need one clock cycle.

It is seen that the BF2/4-based architecture becomes somewhat larger than the BF4-based approach as the requirement on $t_{k\rightarrow k+1}$ in R4 mode becomes tighter, that is, less than 5ns. However, the provided throughput at this stage is beyond the speed requirement of considered applications, for example, high data-rate WPANs. In the figure, this means that the actual design space to be considered extends to the far right. Here, the BF2/4-based architecture is more suitable due to the lower area requirement of about 19% (2516μm^2) as indicated by the arrow. Furthermore, this approach already provides routing resources (PERM) to support a wider range of codes, that is, the four state metrics belonging to an R4 butterfly can be shuffled by PERM in any order to maintain compatibility to the feedback connections of the basic R2 architecture. Considering R2 processing only, the BF2/4 architecture is better suited even down to a $t_{k\rightarrow k+1}$ of about 1.5ns.

Furthermore, both designs need a controller that provides control signals for multiplexers and gated clocks. One can state that the one for the BF2/4-based architecture is about three times larger than the one needed when using BF4 units. This is mostly due to the more advanced clock gating in the former design. However, since a controller is instantiated only once, this gives a negligible contribution to the overall area, especially if the state space grows larger. In this design example, the controllers are always smaller than 3% of the total design size.

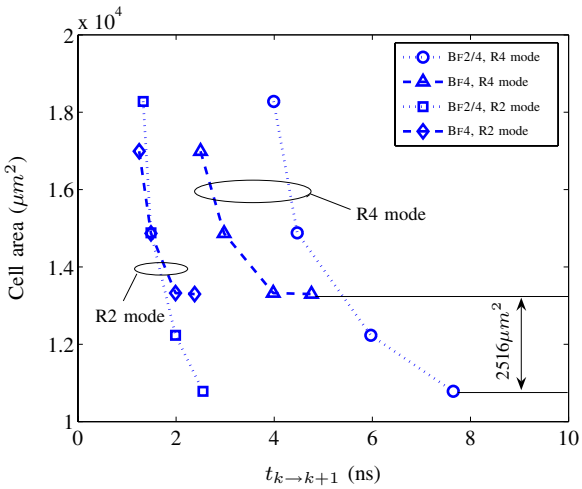

Fig. 7. Cell area versus time for a decoding stage in R2 or R4 mode for architectures based on different radices.

VI. CONCLUSIONS

We presented a framework for processing trellises that are based on butterflies of different radices. In case of a flexible channel decoding platform, both R2 and R4 butterflies have to be taken into account. It is shown how R2 blocks can be reused to compute R4 butterflies. This turns out to be advantageous in terms of area requirements compared to an architecture based on R4 blocks. Up to a critical time for a decoding step in R4 mode of around 5ns, the R2-based approach consumes less area. Additionally, it inherently provides higher flexibility, which has not yet been accounted for in the competing architecture. This makes the rate-flexible architecture based on BF2/4 more suitable for the implementation of a flexible decoding platform.

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] D. E. Hocevar and A. Gatherer, "Achieving flexibility in a Viterbi decoder DSP coprocessor," in *Proc. IEEE Vehicular Technology Conference (VTC-Fall)*, vol. 5, Boston, MA, Sept. 2000, pp. 2257–2264.

[2] J. R. Cavallaro and M. Vaya, "Viturbo: A reconfigurable architecture for Viterbi and turbo decoding," in *Proc. IEEE International Conference on Acoustic, Speech, and Signal Processing (ICASSP)*, vol. 2, Hong Kong, China, Apr. 2003, pp. 497–500.

[3] *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for High Rate Wireless Personal Area Networks (WPANs)*, IEEE Std. 802.15.3, 2003.

[4] J. B. Anderson and A. Svensson, *Coded Modulation Systems*. New York, NY: Kluwer, 2003.

[5] H. Dawid, O. Joeressen, and H. Meyr, "Viterbi decoder: High performance algorithms and architectures," in *Digital Signal Processing for Multimedia Systems*, ser. Signal Processing Series. Marcel Dekker, Inc., Feb. 1999, ch. 17.

[6] P. J. Black and T. H. Meng, "A 140 Mb/s, 32-state, radix-4 Viterbi decoder," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 12, pp. 1877–1885, Dec. 1992.

[7] A. P. Hekstra, "An alternative to metric rescaling in Viterbi decoders," *IEEE Transactions on Communications*, vol. 37, no. 11, pp. 1220–1222, Nov. 1989.