

Implementation of a Dual Analog Decoder

Jae Young Hur, Stephan Wong, and Sorin Cotofana

Computer Engineering Laboratory

Faculty of Electrical Engineering, Mathematics and Computer Science

Delft University of Technology, the Netherlands

E-mail: {[J.Y.HUR](mailto:J.Y.HUR@ewi.tudelft.nl) | [J.S.S.M.WONG](mailto:J.S.S.M.WONG@ewi.tudelft.nl) | [S.D.COTOFANA](mailto:S.D.COTOFANA@ewi.tudelft.nl)}@EWI.TUDELFT.NL

Abstract— When considering forward error correction (FEC) algorithms for telecommunication applications, the BCJR algorithm is the sub-optimal MAP (maximum a posteriori) soft decoding algorithm. For the case of high code rate, BCJR decoding based on the trellis in the dual domain is computationally much less complex than in the original domain. Previously, designs of dual domain primitive circuit elements and high level simulations have been presented but no realization and validation have been reported. In this paper, we present the implementation of a dual analog decoder with high code rate. The implemented decoding algorithm performs three procedures which are branch weight generation, trellis calculation, and soft output generation. The Boxplus circuit performing multiplication and the current mirror are used as basic building blocks in the implementation. The decoder is simulated for a 16-length convolutional code with memory 1 and code rate 2/3. To validate the design we assumed a realistic channel environment and compared the physical circuit behavior (simulated in SPICE) with the algorithm level functionality (simulated in MATLAB). The SPICE simulation results indicate that each of the functional blocks has the distortion less than 3%. The implemented decoder has the same error correction performance as the one predicted by the high level simulations. In addition, the distorted behavior due to the non-ideal characteristic of physical devices is also discussed.

Keywords— Forward Error Correction, BCJR Algorithm, Dual Domain, Analog Design.

I. INTRODUCTION

In high speed applications such as optical transmissions or magnetic recoding, we need high rate codes since the channel is rather good. Among several algorithms to decode convolutional codes, the algorithm proposed by Bahl, Cocke, Jelinek, Raviv (BCJR algorithm) [1] is known to be the sub-optimal *soft* decision decoding algorithm for calculating maximum *a posteriori* probabilities of symbols. Regarding the error correction capability, *soft* decision decoding has the better performance (2 ~ 3 dB coding gain) compared to *hard* decision decoding [2]. However the computational complexity overhead in the *soft* decoder has been the main bottleneck for the implementation.

The error correction channel decoding in modern communication applications is one of the most computation-

ally complex operations (for example, 5 billions operations of Turbo decoding in WCDMA). Now the issue is a tolerable decoding complexity when the codes are implemented. Suppose the information sequence \mathbf{u} and the code word \mathbf{x} encoded by a memory m and code rate $R = \frac{k}{n}$ convolutional encoder. In one trellis section, there are 2^{k+m} branches per trellis section in the original domain while the number of branches is 2^{n-k+m} in the dual domain. In case of $\frac{k}{k+1}$ codes, transformation of the codes into the dual domain reduces the complexity compared to that in the original domain. For example, as k increases given a constant memory, the branch complexity in the original domain grows exponentially, while in the dual the domain, however, the branch complexity remains constant as shown in Table I. The BCJR decoding algorithm in the dual domain contains many multiplications and summations of *soft* bits, which are *tanh* function of the corresponding L -value [1]. When it comes to the implementation in hardware, analog circuits like a differential pair and a diode have exactly the *tanh* relationship between the differential input voltage and the differential output current. Therefore, a differential pair based circuit is the appropriate structure to implement such an algorithm. Such circuits have been introduced already in [6][7]. It has been reported that the analog implementation has speed, area, power advantages compared to digital implementations [7][9]. Moreover, an analog decoder does not need additional hardware, e.g., A/D converter. An analog decoder for memory 1 and code rate 1/2 convolutional codes was implemented into hardware [7]. In addition, a circuit design for memory 1, code rate 2/3 and high level simulation for the dual decoders were shown in [3]. To validate the design of [3], in this work, a dual analog decoder with memory 1 and code rate 2/3 is implemented in SPICE. The simulation results for the systematic bits indicate that each of the functional blocks is fully verified.

The paper is organized as follows. In Section II, theoretic parts of the log-likelihood algebra and the dual BCJR algorithm are reviewed. In Section III, the analog circuit building blocks utilized in this work are described and simulated. In Section IV, simulation results for the *branch*

Number of branches per section	Original domain	Dual domain
m = 1, R = 1/2	4	4
m = 1, R = 2/3	8	4
m = 2, R = 3/4	32	8
m = 2, R = 5/6	128	8
m = 2, R = 7/8	512	8

TABLE I

COMPARISON OF THE NUMBER OF BRANCHES PER TRELLIS SECTION BETWEEN ORIGINAL AND DUAL DOMAIN.

weight generation block, the *trellis calculation* block and the *soft* output generation block are described. Subsequently, the whole decoder is simulated for different signal to noise ratios (SNRs) on the channel. Finally, the conclusion and future work are described in Section V.

II. DUAL BCJR ALGORITHM

A. Soft Decoding Algorithm

A.1 Log-Likelihood Algebra

We consider that a sample \mathbf{x} of a random variable X is transmitted over an AWGN (Additive White Gaussian Noise) channel yielding $\mathbf{y} \in \mathbb{R}$ at the receiver. In an antipodal transmission, where 0 is mapped into +1 and 1 is mapped into -1, the random variable X has two outcomes $x \in \{+1, -1\}$ with the probabilities $P(x = +1)$ and $P(x = -1)$. The *log-likelihood ratio* $L(X) \in \mathbb{R}$, called *L-value* of X is defined as the logarithm of a probability ratio, $L(X) = \ln \frac{P(x=+1)}{P(x=-1)}$. Each probability $P(x = +1)$ and $P(x = -1)$ can be described as $P(x = +1) = \frac{1}{1+e^{-L(X)}}$ and $P(x = -1) = \frac{1}{1+e^{+L(X)}}$. We define the *soft* bit $\lambda(X)$ as the expectation of a random variable X , $\lambda(X) = E(X) = (+1) \cdot P(x = +1) + (-1) \cdot P(x = -1) = \frac{1}{1+e^{-L(X)}} - \frac{1}{1+e^{+L(X)}} = \tanh\left(\frac{L(X)}{2}\right)$. It follows that $L(X) = 2 \tanh^{-1}(\lambda(X))$.

The addition of binary numbers $x_1 \oplus x_2$ is required to perform the *soft* decoding algorithm. With above definitions, the *soft* bit $\lambda(X_3)$ of the sum $x_3 = x_1 \oplus x_2$ of two statistically independent binary numbers x_1 and x_2 is given as $\lambda(X_1 \oplus X_2) = E\{X_1 \cdot X_2\} = E\{X_1\} \cdot E\{X_2\}$. Therefore, $\lambda(X_3) = \lambda(X_1) \cdot \lambda(X_2)$. The *L-value* of the binary addition $x_3 = L(X_1 \oplus X_2) = 2 \tanh^{-1}[\lambda(X_1) \lambda(X_2)]$ is

$$L(X_3) = 2 \tanh^{-1} \left[\tanh\left(\frac{L(X_1)}{2}\right) \tanh\left(\frac{L(X_2)}{2}\right) \right].$$

This operation is called the “*Boxplus*” operation, $L(X_1) \boxplus L(X_2)$, which is including *soft* bit multiplication of each variable x_1 and x_2 and the conversion into the corresponding *L-value*.

A.2 Convolutional Codes

Convolutional encoders are widely used in encoding schemes because of the simple hardware structure which is composed of shift registers and *xor* gates. Fig. 1(a) depicts a simple example of the convolutional encoder which has 1 memory, code rate 2/3. The trellis representation Fig.1(b) depicts the memory state transition with respect to the time step. Every possible state transition has its branch in the trellis whose label is the input/output sequence of the encoder. There are 2 memory states at each time step. l denotes the identifier of the specific trellis subsection $\in \{1, 2, \dots, L\}$. And s denotes the particular state at $(l-1)^{th}$ trellis and s' is at l^{th} trellis subsection.

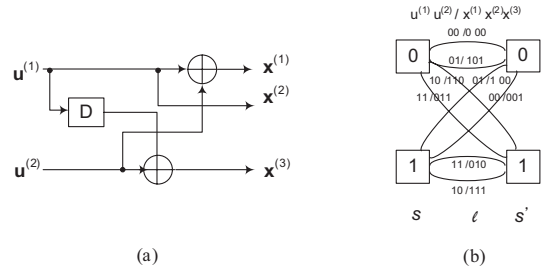


Fig. 1. Memory 1 and code rate 2/3 convolutional encoder (a) Convolutional encoder (b) Trellis representation.

Typically, when we decode a convolutional code, termination bits are added to ensure that the code finishes in a particular state such that the last bits are still protected. In a tailbiting code, the encoder is made to start and end in the same state. The decoding can be initiated at any state and any subsection and it works in a circular structure. For the decoding on the tailbiting trellis, there is no loss in the code rate [5].

A.3 Soft Decision Decoding

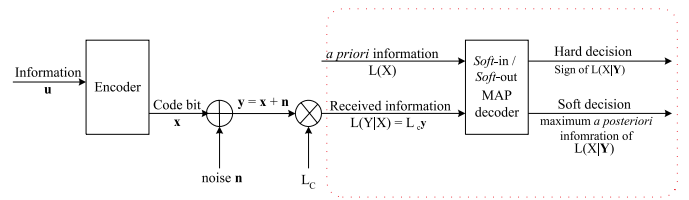


Fig. 2. *Soft-in/Soft-out* MAP decoding.

In the data transmission system depicted in Fig. 2, an information code sequence \mathbf{x} encoded from \mathbf{u} are transmit-

ted to the channel with specific SNR $= \frac{E_b}{N_0}$. $L(x)$ is called the *a priori* information. Received sequence $\mathbf{y} = \mathbf{x} + \mathbf{n}$ are passed through the matched filter and multiplied by L_C . The quantity $L_C = 4 \frac{E_S}{N_0}$ is called the *channel state information* obtained from the AWGN channel estimation. A High L_C value implies that the channel is highly reliable. Our purpose is to estimate $\hat{x} \in \{+1, -1\}$ of a symbol x by minimizing the symbol error probability $P(\hat{x} \neq x)$. The sign of the L -value $L(X|\mathbf{Y})$ indicate the decoded *hard* bit. *Soft-in/Soft-out MAP* decoding algorithm finds the maximum *a posteriori* information $L(X|Y)$. The magnitude $|L(X|Y)|$ measures the reliability of this *hard* decision, where increasing $|L(X|Y)|$ yields a increasing reliability of decision. The original BCJR algorithm [1] is based on the probability propagation. In the original domain, each branch transition has its own probability. From the trellis structure, probabilities of each state in the forward and backward directions are also calculated. With these values, the algorithm generates the *soft* output which corresponds to a L -value for every code bit.

B. Dual BCJR Algorithm

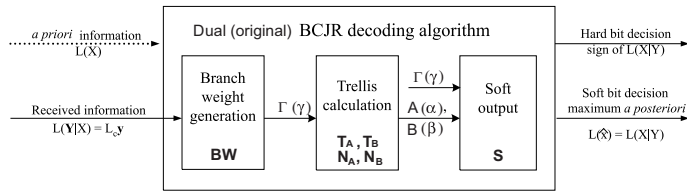


Fig. 3. Overview of dual (original) BCJR algorithm.

In the dual domain, the branch bit weights and the *soft* output generation are derived differently. The *Poisson summation* of the *Fourier* transform changes formulae of the original BCJR into the dual BCJR algorithm. Derivations to the dual domain can be found in [8]. There are 3 sub-tasks in the BCJR decoding process illustrated in Fig. 3.

B.1 Branch Weight Generation Block

The individual branch bit weight for the dual code is as follows :

$$\Gamma_l(s, s', i) = \begin{cases} 1 & \text{if } x_l^{(i), \perp} = +1 \\ \tanh\left(\frac{L(y_l^{(i)} | x_l^{(i)})}{2}\right) & \text{if } x_l^{(i), \perp} = -1, \end{cases} \quad (1)$$

where $x_l^{(i), \perp}$ denotes the i^{th} code bit in a dual trellis. $L(y_l^{(i)} | x_l^{(i)})$ denotes the L -value of a code bit which is a the matched filter output scaled with *channel state information*. The $\Gamma_l(s, s', i)$ denotes the *branch weight* of

the $i^{th} \in \{1, 2, \dots, n\}$ code bit located at branch subsection l and y^i denotes the received value from the channel, corresponding to the i^{th} code bit. This *branch weight* is the *soft* bit of the received value $L_C y$, The total *branch weight* of each branch is derived from the multiplication of each *branch weight* of the individual bits. High *branch weight* path implies the high probable path in the decoding step. The total *branch weight* can be derived as $\Gamma_l(s, s') = \prod_{i=1}^n \Gamma_l(s, s', i)$, which is the *soft* bit.

B.2 Trellis Calculation Block

Each state value A, B at a trellis subsection l can be calculated from the multiplication of a state value at a trellis subsection $l - 1$ and the *branch weight* at a trellis subsection l according to the trellis structure. That is,

$$A_l(s') = \sum_s A_{l-1}(s) \Gamma_l(s, s') \quad (2)$$

$$B_{l-1}(s) = \sum_{s'} B_l(s') \Gamma_l(s, s'). \quad (3)$$

$A_l(s')$ denotes the state value at branch subsection l and $A_{l-1}(s)$ denotes the state value at branch subsection $l - 1$. Similarly, the state value for backward recursion B can be obtained. $B_{l-1}(s)$ denotes the state value at branch subsection $l - 1$ and $B_l(s')$ denotes the state value at subsection l . In the tailbiting trellis, the starting distribution of first trellis is arbitrary. Therefore, no initialization is required. In the dual domain, the zero state at every branch subsection has to be kept to be 1. Therefore the state values can be normalized by dividing every state value by zero state value, which means that $A'_l(0) = 1$, $A'_l(1) = \frac{A_l(1)}{A_l(0)}$.

B.3 Soft Output Generation Block

The *branch weight* Γ and state values A, B can be used to obtain the maximum *a posteriori* conditional L -value in *MAP* decoding. This *a posteriori* information depends on the received channel output. In case of the feed-forward encoder as depicted Fig. 1, the *soft* output generation can be represented as follows [3].

$$L(\hat{x}_l^{(j)}) = \ln \left(\frac{\sum_{s|x_l=+1} \alpha_l(s) \cdot \beta_l(s)}{\sum_{s|x_l=-1} \alpha_l(s) \cdot \beta_l(s)} \right) \quad (4)$$

In Fig. 4, the bit error rate (BER) curve which represents an algorithm behavior of memory 1 and code rate 2/3 for the dual BCJR algorithm is depicted.

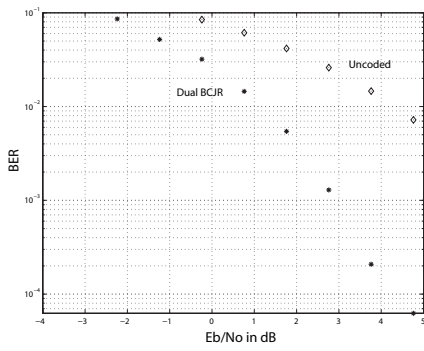


Fig. 4. Algorithm level simulation : BER curve of memory 1 and code rate 2/3 dual BCJR algorithm.

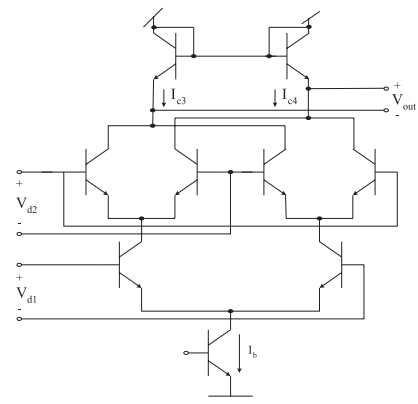


Fig. 6. General *Boxplus* circuit.

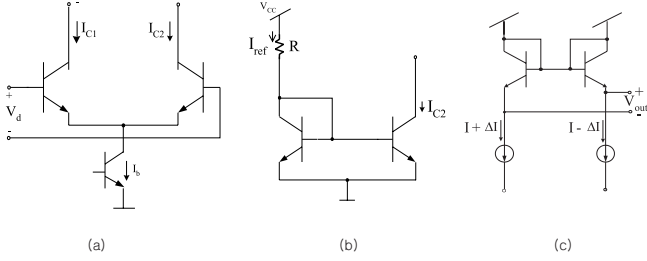


Fig. 5. Basic circuit fragments (a) Differential pair (b) Current mirror (c) Hyperbolic tangent inverse.

III. ANALOG DECODER CIRCUIT BUILDING BLOCKS

In the previous section, we reviewed the algorithm behavior. The aim in this section is to find circuit blocks which perform arithmetic operations (addition, subtraction, multiplication, division) of *soft* bits as well as *tanh* operation, while minimizing circuit resources. In analog circuits, a differential pair based *Boxplus* circuit [4] exhibits such a characteristic. In addition, the current mirror circuit is used as well. A differential pair is composed of two Bipolar Junction Transistors (BJTs) as depicted in Fig. 5 (a). According to the input differential voltage V_d , each collector current I_{C1} , I_{C2} can be described as $I_{C1} = \frac{I_b}{1+e^{-\frac{V_d}{V_T}}}$ and $I_{C2} = \frac{I_b}{1+e^{+\frac{V_d}{V_T}}}$ respectively. Differential current $\Delta I_C = I_{C1} - I_{C2}$ can be obtained as $\frac{\Delta I_C}{I_b} = \tanh\left(\frac{V_d}{2V_T}\right) = \lambda(X)$. Fig. 5 (b) depicts a simple *nnpn* type current mirror (current source). The reference current I_{ref} is generated by V_{CC} and R and ideally the same current I_{C2} is mirrored. Fig. 5 (c) shows the hyperbolic tangent inverse circuit. The hyperbolic tangent inverse (HTI) circuit consists of two diodes. The differential voltage V_{out} is $V_T \ln \frac{I_+ + \Delta I}{I_- - \Delta I}$. It follows that $V_{out} = 2 V_T \tanh^{-1} \left(\frac{\Delta I}{I} \right)$.

A. *Boxplus* Circuit

The *Boxplus* circuit is composed of the Gilbert multiplier and a HTI circuit. The Gilbert multiplier is a stacked configuration of differential pairs with a cross coupling of the collector outputs in the upper stage as depicted in Fig. 6, which consists of 8 transistors and one current source. The bias current I_b is split with respect to V_{d1} . Four collector currents in upper stages are the split currents from bias current I_b with respect to V_{d1} and V_{d2} . Consequently, the Gilbert multiplier cell is used for generating a *differential current* corresponding to the result of *soft* bit multiplication [6]. The differential voltage V_{out} of the hyperbolic tangent inverse circuit is given by $V_{out} = 2 V_T \tanh^{-1} \left[\tanh\left(\frac{V_{d1}}{2V_T}\right) \tanh\left(\frac{V_{d2}}{2V_T}\right) \right]$.

B. *Summation and Subtraction* Circuit

The summation and subtraction circuits of *soft* bits are required in the *trellis calculation*. Fig. 7 shows the *connectivity* for summation and subtraction, respectively. From the figure, $I_1 - I_2 = (I_{L1} - I_{L2}) + (I_{R1} - I_{R2}) = \Delta I_0 + \Delta I_1$ and $I_3 - I_4 = (I_{L1} - I_{L2}) - (I_{R1} - I_{R2}) = \Delta I_0 - \Delta I_1$.

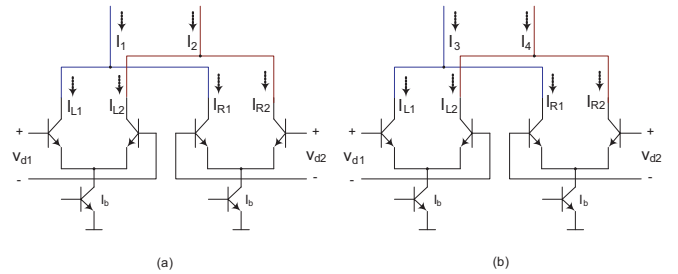


Fig. 7. Summation (a) and subtraction (b) of *soft* bits.

Table II summarizes the analogy between log-likelihood algebra and circuit quantity. *LLR* denotes log-likelihood ratio.

Algebra	LLR $L(X)$	Soft bit $\lambda(X)$
Range	$(-\infty, \infty)$	$(-1, 1)$
Circuit element	voltage diff $\frac{\Delta V}{V_T}$	current diff $\frac{\Delta I}{I_b}$

TABLE II
ANALOGY BETWEEN LOG-LIKELIHOOD ALGEBRA AND
CIRCUIT QUANTITY.

IV. IMPLEMENTATION OF DUAL ANALOG DECODER

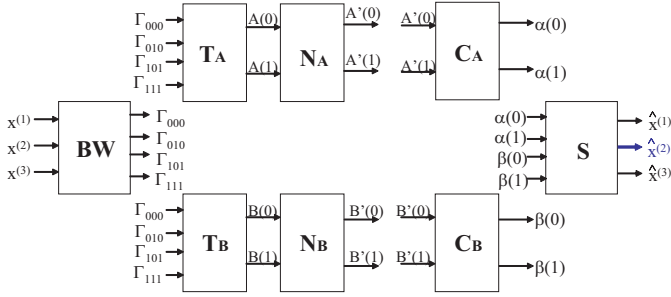


Fig. 8. Block diagram of implemented dual decoder.

A memory 1 and code rate $R = 2/3$ dual decoder, which is implemented in this work is illustrated in Fig. 8. The systematic bit is the concern for the decoding. The blocks **BW**, **T_A**, **N_A**, **C_A** and **S** stand for *branch weight*, *trellis calculation*, *normalization*, *soft out*, respectively. Each block is designed with respect to the trellis structure in Fig. 9. Each trellis subsection is associated with 3 received code bits and 4 *branch weights* Γ_{111} , Γ_{101} , Γ_{010} and Γ_{000} are generated in the **BW** block. And **T_A** and **N_A** blocks calculate 2 state values $A'_l(0)$ and $A'_l(1)$ after normalization for each trellis subsection. The same circuit is used for $B'_l(0)$, $B'_l(1)$ calculation. Subsequently, $A'_l(0)$ and $A'_l(1)$ are transformed into $\alpha(0)$ and $\alpha(1)$ of the original domain in **C_A**. The systematic *soft* outputs of each trellis subsection $L(\hat{x}^{(2)})$ are generated in the **S** block.

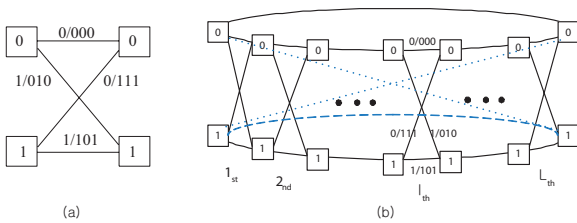


Fig. 9. Memory 1 and code rate 2/3 (a) Trellis diagram in dual domain (b) Tailbiting trellis diagram with length L .

A. Branch Weight Generation Block

From the discussion in Section (II-B.1), each *branch weight* can be derived as :

$$\begin{aligned}\Gamma_{111} &= \tanh\left(\frac{L_C y^{(1)}}{2}\right) \tanh\left(\frac{L_C y^{(2)}}{2}\right) \tanh\left(\frac{L_C y^{(3)}}{2}\right) \\ \Gamma_{101} &= \tanh\left(\frac{L_C y^{(1)}}{2}\right) \tanh\left(\frac{L_C y^{(3)}}{2}\right) \\ \Gamma_{010} &= \tanh\left(\frac{L_C y^{(2)}}{2}\right) \quad \text{and} \quad \Gamma_{000} = 1.\end{aligned}$$

The *tanh* function can be implemented using the *Boxplus* or differential pair. The case of a branch with *branch weight* (*tanh* function) 1 implies that the received L -value $L(y_l^{(i)}|v_l^{(i)})$ is infinite, which means that the differential voltage is infinite. Fig. 10 depicts that how it can be implemented as an example of the differential pair. The right transistor is omitted and the other left transistor is replaced by a wire to realize the effect of an infinite voltage so that all current flow through the left path. To generate

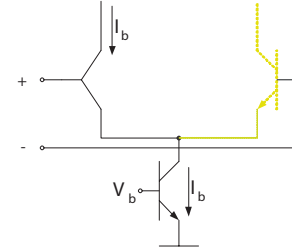


Fig. 10. Differential pair in case of branch weight of 1.

the *branch weight* for 3 code bits, serially connected *Boxplus* blocks can be used. Only a modification as in Fig. 10 needs to be performed for the case of code bit 0 whose *branch weight* is 1. Fig. 11 shows the **BW** block diagram, which receives 3 differential voltages (L -values) $L_C y^{(1)}$, $L_C y^{(2)}$ and $L_C y^{(3)}$ and generates $V_{\Gamma_{111}}$, $V_{\Gamma_{101}}$ and $V_{\Gamma_{010}}$. $V_{\Gamma_{111}}$ is generated in the **BW111** circuit in Fig. 11, which performs the multiplication of 3 *soft* bits. When it comes to the case of two code bits having a 1, one *Boxplus* circuit itself can be used. For the generation, $\Gamma_{010} L_C y^{(2)}$ is equal to Γ_{010} . That is, input voltage $V_{L_C y^{(2)}}$ directly goes to the *trellis calculation* block. In case of Γ_{000} , no circuit is required to implement a infinite voltage.

B. Trellis Calculation Block

According to the trellis structure of the dual decoder in Fig. 9, the memory state values are updated by

$$\begin{aligned}A_l(0) &= A_{l-1}(0) \Gamma_{000} + A_{l-1}(1) \Gamma_{111} \\ A_l(1) &= A_{l-1}(0) \Gamma_{010} + A_{l-1}(1) \Gamma_{101}.\end{aligned}$$

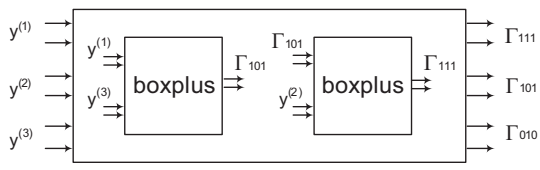


Fig. 11. Branch weight generation block **BW** (Γ_{000} is omitted).

There are multiplications of *soft* bits, which are current state value and the *branch weight* value, and summation of *soft* bits. Therefore, the *trellis calculation* block \mathbf{T}_A can be implemented using the *Boxplus* circuit and summation connectivity. For the case of a state value of a 0 state, the normalized state value is kept to be 1, which means that the corresponding L -value is infinite. The circuit for the *branch weight* of 000 needs to be modified to implement the infinite voltage too. The modified circuits for $A_l(0)$ and $A_l(1)$ are depicted in Fig. 12 which constitute \mathbf{T}_A block [3].

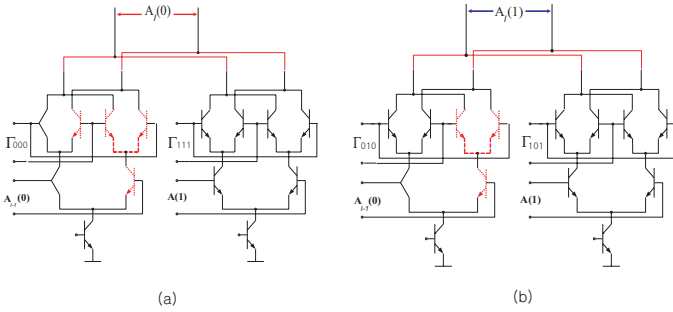


Fig. 12. Trellis calculation block (\mathbf{T}_A).

C. Normalization Block

A normalization in the dual BCJR algorithm is performed by dividing the state values by $A_l(0)$, i.e., after normalization $A_l'(0) = \frac{A_l(0)}{A_l(0)} = 1$ and $A_l'(1) = \frac{A_l(1)}{A_l(0)}$. Generating $A_l'(0) = 1$ in the next state is to simply omit 1 transistor in the differential pair, therefore we only need to consider the normalization for $A_l(1)$. Let $A_l(0) = a_0$ and $A_l(1) = a_1$ before normalization. What we require is the differential current ΔI to get $A_l'(1) = \frac{a_1}{a_0}$, that is $\frac{\Delta I}{I_b} = \frac{a_1}{a_0}$. The hyperbolic tangent inverse circuit will convert the differential current ΔI into the required ΔV .

Now we need the circuit to generate $a_0 + a_1$ and $a_0 - a_1$. In Fig. 13 [3], which simply are current summations and subtractions. The block diagram to perform the summation and subtraction is shown in Fig. 13. The upper current mirror copies the currents from \mathbf{T}_A block and outputs ideally the identical currents. The current difference circuit generates the difference between two input currents.

The hyperbolic tangent inverse circuit converts the differential currents $I_{d1} - I_{d2}$ into differential voltage ΔV as $= 2V_T \tanh^{-1} \left(\frac{a_0 + a_1}{a_0 - a_1} \right)$, where the ΔV is the required voltage to get the normalized value for $A_l(1)$.

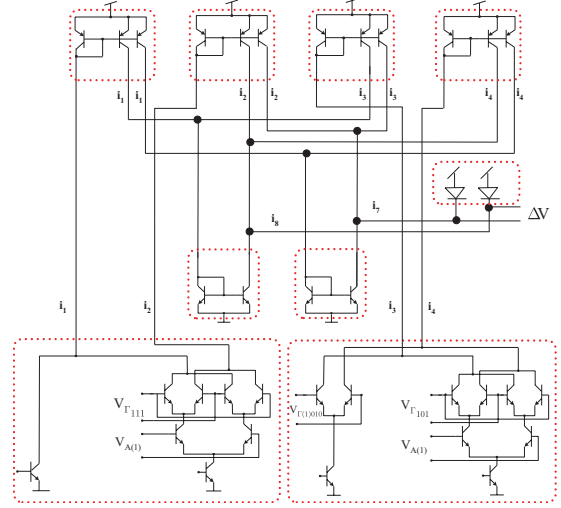


Fig. 13. Block diagram (\mathbf{N}_A) for normalization of $A_l(1)$.

D. Conversion Block from Dual into Original Domain

The state values $A_l(0)$ and $A_l(1)$ in the dual decoder have the relationship, $\alpha_l(0) = 0.5(A_l(0) + A_l(1))$ and $\alpha_l(1) = 0.5(A_l(0) - A_l(1))$. Therefore, the circuit for the summation and the subtraction of *soft* bits is required. In addition, the current difference circuit which converts the differential current into single current is required as well. Fig. 14 depicts the circuit [3].

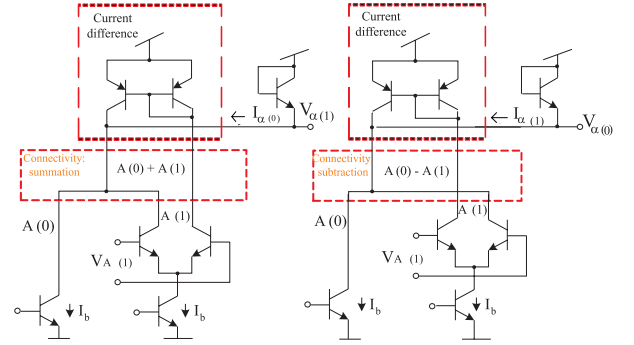


Fig. 14. Conversion block \mathbf{C}_A from dual into original domain.

E. Soft Output Generation Block

For the memory 1 and code rate 2/3 dual decoder, the *soft* output is

$$L(\hat{u}) = \ln \left(\frac{\alpha(0)\beta(0)}{\alpha(1)\beta(1)} \right).$$

To do this, the circuit for multiplication of the probabilities α and β is needed. Subsequently the hyperbolic tangent inverse circuit converts the differential current $I_{\alpha(0)\beta(0)} - I_{\alpha(1)\beta(1)}$ into the differential output voltage $V_{L(\hat{u})}$ corresponding to the decoded *soft* bits. Fig. 15 [6] shows the circuit, where the 4 currents in upper stage represent 4 possible combinations of α and β . $I_{\alpha(0)\beta(0)}$ is the only term which contributes so that decoded *hard* decision of information bit is 0. And $I_{\alpha(1)\beta(1)}$ is the only term which contributes so that the decoded *hard* decision of information bit is 1. The other 2 terms are not necessary so they are connected to V_{CC} . Therefore the L -value of the decoded *soft* output is $L(\hat{u}) = \frac{V_{out}}{V_T} = \ln \left(\frac{\alpha(0)\beta(0)}{\alpha(1)\beta(1)} \right)$.

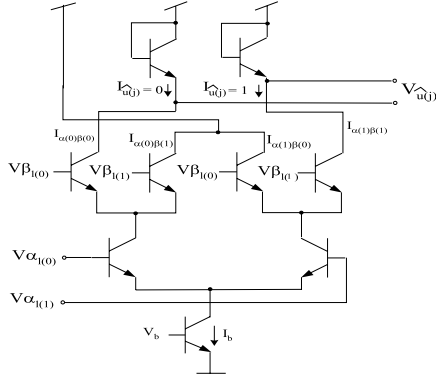


Fig. 15. *Soft* output generation block S.

F. Simulation Results of Individual Blocks

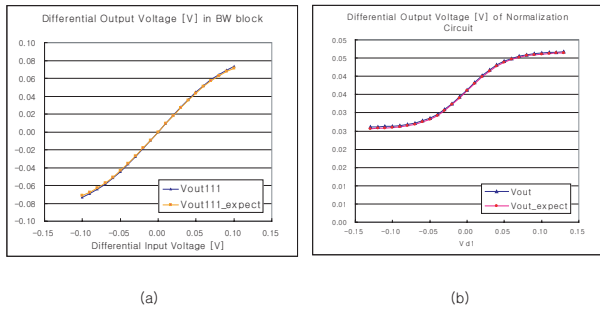


Fig. 16. Simulation results of branch weight (a) and trellis calculation (b) blocks.

In this subsection, **BW** block and \mathbf{T}_A , \mathbf{N}_A blocks are simulated, as those are much complex than others. Fig. 16 (a) illustrates the expected behavior and the circuit behavior in **BW** block. V_{out111} and V_{out111_expect} denote the circuit voltage of Γ_{111} and the estimated voltage of Γ_{111} , respectively. 2 out of 3 inputs are set to be 0.1 V, corresponding to L -value 3.85. 1 out of 3 inputs is swept from -0.1 V to 0.1 V. The simulated behavior has less than 2% errors. In Fig. 16 (b), the expected behavior and

the circuit behavior after \mathbf{N}_A block are illustrated. V_{out} and V_{out_expect} denote the circuit voltage of $A'(1)$ and the estimated voltage of $A'(1)$, respectively. 3 inputs Γ_{111} , $\Gamma_{101}, \Gamma_{010}$ are set to be 0.12, 0.2, 0.6, respectively. The voltage for $A_{l-1}(1)$ is swept from -0.13 V to 0.13 V. The simulated behavior has less than 1% error.

G. Overall Simulation

In this subsection, we describe how the complete decoder is constructed and simulated. The encoder and a channel environment are assumed to be same in all simulations between the high level simulation and low level circuit simulation. The simulation is performed on WinSpice [12] and the model parameters are assumed to be the *default*. The decoded outputs generated in the high level simulation are compared to those generated in the circuit simulation. Several environments are considered to verify whether the decoder circuits work properly. Input stimulus for the transient analysis is the pulse which has a rise time of 50us, fall time of 50 us, a duration of 500 us and the total period = 700 us. Different information and different channel SNRs are used for simulations.

Fig. 17 depicts one of the simulation results. Left graph shows the expected L -value distribution and the right graph shows the circuit simulated L -value distribution. The hard decoding results are the same. In addition, the *soft* decoded L -value distribution of the implemented circuit is close to the expected one. Only some differences between two distribution are found.

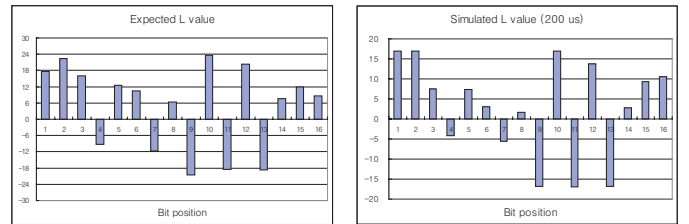


Fig. 17. Comparison between L -values of expectation and simulation considering random information sequence and noisy channel with $\frac{E_b}{N_o} = 2.76$ dB.

Table III shows the summary of the circuit simulations. TER denotes the transmission error rate (number of errors/transmitted bits) and DER denotes the decoding error rate (number of errors/decoded bits). Expected values are from high level (MATLAB) simulations as shown in Fig. 4. Simulation results show that implemented circuit follows the error rate with respect to the expected behavior.

Input	Channel SNR($\frac{E_b}{N_0}$)	L_C	TER	DER(Expect)	DER(Circuit)
Zero	1.76 dB	6	1/16	0/16	0/16
Random	1.76 dB	6	2/16	0/16	0/16
Random	2.76 dB	7.55	1/16	0/16	0/16
Random	0.76 dB	4.776	1/16	0/16	0/16

TABLE III
EXPERIMENTAL RESULTS ON THE DECODER CIRCUIT SIMULATIONS.

V. CONCLUSION

A memory 1 and code rate $2/3$ dual analog decoder with 16 length has been implemented and simulated in circuit level. The number of transistors required for multiplication, addition (subtraction) and normalization (division) are 8, 4 and 18, respectively. Furthermore, the \tanh operation can be efficiently implemented using 2 transistors. The contribution of this paper is the validation of the design of a code rate $2/3$ dual analog decoder [3]. As a result of circuit simulation, each of the functional blocks has distortions within 3%. The decoding error correction rate follows the expected decoding error rate properly with some L -value reduction, while the *soft* bit reduction is negligible (within 5%). It can be noted that the same design methodology as used in this work is applicable to the dual decoder design with memory > 2 and code rate $> 2/3$, e.g., memory 2 and code rate $3/4$, memory 2 and code rate $5/6$ or memory 2 and code rate $7/8$.

The following efforts can be done to improve the circuit behavior :

- Optimize the bias current and voltage
- Optimize the MOS transistor size W, L to get the required voltage.
- Design the more efficient normalization circuits.

Acknowledgement This work is based on the master graduation project at institute of communications engineering in Munich University of Technology. Most of the ideas came from the discussion and previous works of 2 supervisors Andrew Schaefer and Matthias Moerz. Therefore authors would like express the acknowledgement for that.

REFERENCES

- [1] L.R. Bahl, J. Cocke, F. Jelinek and J. Raviv, "Optimal decoding of linear codes for minimising symbol error rate," in *IEEE Trans. Info. Theory*, pp. 284–287, March 1974.
- [2] J. Hagenauer, "Soft-in/soft-out : The benefits of using soft decisions in all stages of digital receivers", in *proc. of 3rd Int. Workshop on DSP Techniques Appli. Space Commun., Noordwijk, The Netherlands, Sep. 1992.*
- [3] A. Schaefer, "Analogue decoding for high rate codes with short block length" Master Thesis, Munich University of Technology, Institute for Communications Engineering, September 2000.
- [4] J. Hagenauer, "Decoding of binary codes with analog networks," in *Proc., International Workshop on Information Theory*, San Diego, California, USA, pp. 13–14, February 1998.
- [5] J.B. Anderson and S.M. Hladik, "Tailbiting MAP decoders," in *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 2, pp. 297–302, February 1998.
- [6] M. Moerz, "Analogue decoders and their implementation in VLSI", Diploma Thesis, Munich University of Technology, Institute for Communications Engineering, February 1999.
- [7] M. Moerz, T. Gabara, R. Yan and J. Hagenauer, "An analog 0.25 μm BiCMOS tailbiting MAP decoder," in *Proc., IEEE Int. Solid-State Circuits Conference (ISSCC 2000)*, San Francisco, California, USA, pp. 356–357, February 2000.
- [8] Ch. Weiss and J. Berkmann, "Suboptimum MAP-decoding of tailbiting codes using the dual trellis", in *Proc. 3rd ITG Conf. Source and Channel Coding*, München, Germany, pp. 199–204, January 2000.
- [9] F. Lustenberger, M. Helfenstein, H.A. Loeliger, F. Tarköy and G.S. Moschytz, "An analog decoding technique for digital codes," in *Proc., IEEE Int. Symp. Circuits and Systems*, vol. II, pp. 428–431, Orlando, Florida, USA, May/June 1999.
- [10] H.A. Loeliger, M. Helfenstein, F. Lustenberger and F. Tarkoey, "Iterative sum product decoding with analog VLSI," in *Proc., IEEE International Symposium on Information Theory*, Cambridge, MA, USA, p. 146, August 1998.
- [11] P. Gray and R.Meyer, *Analysis and Design of Analog integrated Circuits*, 3rd Ed., Singapore: Wiley & Sons, 1992.
- [12] WinSpice3 User Manual, <http://www.winspice.com>
- [13] J.Y.Hur, "Design and Simulation of Dual Analog Decoder", Master Thesis, Munich University of Technology, Institute for Communications Engineering, October 2002.
- [14] R. Johannesson and K. Zigangirov, *Fundamentals of Convolutional Coding*, New York: IEEE Press, 1999.
- [15] Lin and Costello, *Error Control Coding: Fundamentals and Applications*, Englewood Cliffs: Prentice Hall, 1983.