Spatially Coupled Serially Concatenated Codes: Performance Evaluation and VLSI Design Tradeoffs

Mojtaba Mahdavi, *Member, IEEE*, Stefan Weithoffer, *Member, IEEE*, Matthias Herrmann, *Member, IEEE*, Liang Liu, *Member, IEEE*, Ove Edfors, *Senior Member, IEEE*, Norbert Wehn, *Senior Member, IEEE*, and Michael Lentmaier, *Senior Member, IEEE*

***Abstract*— Spatially coupled serially concatenated codes (SC-SCCs) are constructed by coupling several classical turbo-like component codes. The resulting spatially coupled codes provide a close-to-capacity performance and low error floor, which have attracted a lot of interest in the past few years. The aim of this paper is to perform a comprehensive design space exploration to reveal different aspects of SC-SCCs, which is missing in the literature. More specifically, we investigate the effect of block length, coupling memory, decoding window size, and number of iterations on the decoding performance, complexity, latency, and throughput of SC-SCCs. To this end, we propose two decoding algorithms for the SC-SCCs: *block-wise* and *window-wise* decoders. For these, we present VLSI architectural templates and explore them based on building blocks implemented in 12 nm FinFET technology. Linking architectural templates with the new algorithms, we demonstrate various tradeoffs between throughput, silicon area, latency, and decoding performance.**

***Index Terms*— Spatial coupling, turbo-like codes, window decoder, coupling memory, VLSI implementation, digital baseband, channel coding, decoding latency.**

I. Introduction

ULTRA reliable communication (URC) is one of the main types of applications in 5G and beyond 5G (B5G) systems, in which a very low bit-error-rate (BER) is needed. Example use cases are self driving cars, factory automation, and remote surgery [1]. The classical turbo codes, parallel concatenated codes (PCCs), suffer from an error floor, where in a moderate to high signal-to-noise ratio (SNR) the decoding performance is only slightly enhanced by increasing SNR. This prevents turbo codes from being used in such applications. Moreover, the demand of ultra reliability is more pronounced for short-length messages where the corresponding code is short and, in consequence, the overall system will work far away from the Shannon limit, which is stated for very long codes.

Recently, *spatial coupling* gives a new perspective of code design and improves the decoding threshold of turbo-like codes [2]. The main idea is to construct powerful long codes using small component codes. As a result, the threshold of an iterative belief propagation (BP) decoder is improved to that of the optimal maximum-a-posteriori (MAP) decoder. This phenomenon is referred to as *threshold saturation* [2], which makes it possible to achieve Shannon limit by coupling the classical turbo-like codes, while without spatial coupling a significant gap exists between BP and MAP thresholds [2].

Moreover, [2] shows that with spatial coupling, serially concatenated codes (SCCs) can achieve better performance than PCCs in both waterfall and error floor regions [2]. For this reason, spatially coupled serially concatenated codes (SC-SCCs) are selected as the focus of this paper. We will show that SC-SCCs can address the demand of very low BER, even for small block lengths, without additional latency and complexity compared to the classical turbo codes. Having considered the mentioned advantages, the spatial coupled codes can be considered as a strong candidate to be used in B5G systems.

Contributions: The main contributions of this paper are as follows.

- In this paper, for the first time, we propose two decoding algorithms for SC-SCCs, called *block-wise* and *window-wise* decoding. Several VLSI architecture templates are presented for an efficient implementation of these algorithms in a 12 nm FinFET technology.
- Due to the spatial coupling the design space of SC-SCC schemes becomes huge compared to the uncoupled ensembles. We have performed an extensive design space exploration for by means of area estimations based on fully placed and routed building blocks, Monte Carlo simulations, and complexity analysis. We investigate the effect of block size, coupling memory, decoding window size, and the number of iterations on the performance, complexity, throughput, decoding latency, and silicon area of the proposed schemes in a 12 nm FinFET technology.

Manuscript received September 1, 2021; revised November 30, 2021 and January 17, 2022; accepted January 27, 2022. Date of publication February 25, 2022; date of current version April 28, 2022. This work was supported by the Ericsson Research Foundation, Sweden. This article was recommended by Associate Editor J.-O. Klein. *(Corresponding author: Mojtaba Mahdavi.)*

Mojtaba Mahdavi was with the Department of Electrical and Information Technology (EIT), Lund University, 22100 Lund, Sweden. He is now with Ericsson, 22362 Lund, Sweden (e-mail: mojtaba.mahdavi@ericsson.com).

Stefan Weithoffer is with the Lab-STICC, UMR CNRS 6285, IMT Atlantique, 29238 Brest, France (e-mail: stefan.weithoffer@imt-atlantique.fr).

Matthias Herrmann and Norbert Wehn are with the Department of Electrical and Computer Engineering, Technical University of Kaiserslautern, 67663 Kaiserslautern, Germany (e-mail: herrmann@eit.uni-kl.de; wehn@eit.uni-kl.de).

Liang Liu, Ove Edfors, and Michael Lentmaier are with the Department of Electrical and Information Technology (EIT), Lund University, 22100 Lund, Sweden (e-mail: liang.liu@eit.lth.se; ove.edfors@eit.lth.se; michael.lentmaier@eit.lth.se).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2022.3149718.

Digital Object Identifier 10.1109/TCSI.2022.3149718

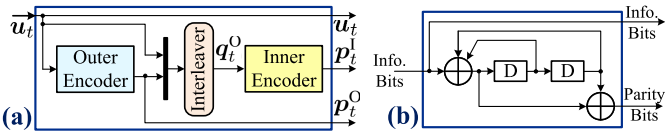


Fig. 1. (a) SCC encoder; black bar represents concatenation, (b) RSC encoder.

- Based on this exploration, we provide several tradeoffs between performance, latency, throughput, and silicon area of the proposed SC-SCC schemes.
- We demonstrate how to perform a fair comparison between different SC-SCC scenarios and also between the coupled and uncoupled turbo codes. As a result, our scheme achieves better performance than the uncoupled ensembles with the same latency and complexity.

II. Background

This section starts by describing the encoder and decoder of uncoupled codes, i.e., SCCs. Then, we demonstrate how to build the SC-SCCs and present the corresponding encoder.

A. SCC Encoder

The SCC *component encoder* consists of two recursive systematic convolutional (RSC) encoders, named as the *outer* and *inner* encoders, which are concatenated in a serial manner using an interleaver as depicted in Fig. 1(a). In this work, the RSC encoder with the generator polynomial of (1, 5/7) and the code rate of $R = 1/2$, shown in Fig. 1(b), is considered as a case study. The encoding procedure of the SCC is as follows.

The information sequence is divided into blocks of length K bits, which enter the outer encoder, as shown in Fig. 1(a). The outer encoder produces the K-bit outer parity sequence, $\boldsymbol{p}_t^{\mathrm{O}}$, for the information block at time t, i.e., $\boldsymbol{u}_t$. Then, the two sequences, $\boldsymbol{u}_t$ and $\boldsymbol{p}_t^{\mathrm{O}}$, will be permuted using the interleaver to generate the $2K$-bit sequence, $\boldsymbol{q}_t^{\mathrm{O}} = \Pi(\boldsymbol{u}_t, \boldsymbol{p}_t^{\mathrm{O}})$. The inner encoder receives $\boldsymbol{q}_t^{\mathrm{O}}$ and produces a $2K$-bit inner parity sequence $\boldsymbol{p}_t^{\mathrm{I}}$. Finally, the SCC encoder output is $\boldsymbol{v}_t = (\boldsymbol{u}_t, \boldsymbol{p}_t^{\mathrm{O}}, \boldsymbol{p}_t^{\mathrm{I}})$, which is called *code block* and transmitted over the channel. Since, the outer and inner trellis lengths are K and $2K$, respectively, the overall code rate of the SCC is $R = 1/4$.

B. SCC Decoder

The SCCs can be decoded using inner and outer decoders, which employ the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm to calculate maximum a posteriori (MAP) estimates of the information bits. In this paper, the SCC decoding is referred to as **Algorithm 1** and an example of its processing flow is shown in Fig. 2. Since the SCC is a kind of block code, each block is decoded independently of the previous and next blocks. Thus, at time t, the decoder receives the log likelihood ratio (LLR) of information, outer, and inner parity sequences $(L_{\mathrm{ch}}(\boldsymbol{u}_t), L_{\mathrm{ch}}(\boldsymbol{p}_t^{\mathrm{O}}), L_{\mathrm{ch}}(\boldsymbol{p}_t^{\mathrm{I}}))$ to perform decoding as follows.

In each iteration, the inner decoder receives three input streams. The first one contains the LLRs of the inner parity sequence, $\mathbf{In1}_{\mathrm{D}}^{\mathrm{I}}(t)6 = L_{\mathrm{ch}}(\boldsymbol{p}_t^{\mathrm{I}})$ and the second one, $\mathbf{In2}_{\mathrm{D}}^{\mathrm{I}}(t)$, is generated by permuting the LLRs of information and outer parity sequences using the *Interleaver*. The third input contains

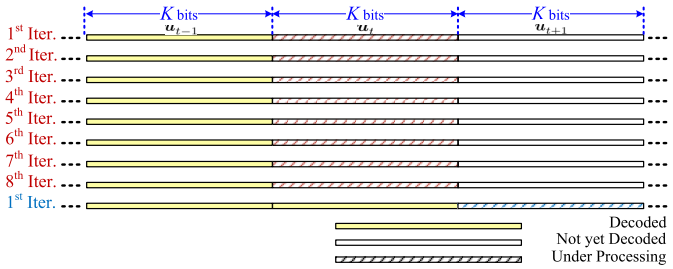


Fig. 2. SCC decoding flow to decode the block at time t after eight iterations.

a-priori LLRs, $L_{\mathrm{a}}(\boldsymbol{q}_t^{\mathrm{O}})$, and is obtained by permuting the extrinsic LLRs, which are generated by the outer decoder, i.e., $\mathbf{In3}_{\mathrm{D}}^{\mathrm{I}}(t) = \Pi(L_{\mathrm{e}}(\boldsymbol{u}_t), L_{\mathrm{e}}(\boldsymbol{p}_t^{\mathrm{O}}))$. The inner decoder output contains the extrinsic LLRs, $L_{\mathrm{e}}(\boldsymbol{q}_t^{\mathrm{I}})$, which will be deinterleaved and used as a-priori information, $L_{\mathrm{a}}(\boldsymbol{q}_t^{\mathrm{I}})$, for the outer decoder. The other outer decoder inputs are the channel LLR values of information and outer parity sequences, $L_{\mathrm{ch}}(\boldsymbol{u}_t)$ and $L_{\mathrm{ch}}(\boldsymbol{p}_t^{\mathrm{O}})$. Finally, the outer decoder produces the extrinsic LLRs, $L_{\mathrm{e}}(\boldsymbol{u}_t)$ and $L_{\mathrm{e}}(\boldsymbol{p}_t^{\mathrm{O}})$, and sends them back to the inner decoder.

The above process is repeated for I iterations for the same block $(\boldsymbol{u}_t)$, striped rectangle in Fig. 2, to eventually make the hard decision for $\boldsymbol{u}_t$. After that, a similar process will be done to decode the next block, $\boldsymbol{u}_{t+1}$ (see last row in Fig. 2).

C. SC-SCC Encoder

The encoder of SC-SCCs is constructed by coupling $m+1$ component encoders, where m is the *coupling memory*. As shown in Fig. 3, each component encoder consists of an outer encoder, an inner encoder, two interleavers, and a demultiplexer, which are connected together as follows. At time instant t the outer encoder receives a block of information bits as its input stream, $\mathbf{In}_{\mathrm{E}}^{\mathrm{O}}(t) = \boldsymbol{u}_t$, and generates the corresponding outer parity sequence $\boldsymbol{p}_t^{\mathrm{O}}$. Then, *Interleaver 1*, permutes the pair of $(\boldsymbol{u}_t, \boldsymbol{p}_t^{\mathrm{O}})$ and generates a $2K$-bit sequence,

$$\boldsymbol{q}_t^{\mathrm{O}} = \Pi_1(\boldsymbol{u}_t, \boldsymbol{p}_t^{\mathrm{O}}). \tag{1}$$

This sequence is divided into $m+1$ portions of equal size, i.e., $\boldsymbol{q}_{t,0}^{\mathrm{O}}, \boldsymbol{q}_{t,1}^{\mathrm{O}}, \ldots, \boldsymbol{q}_{t,m}^{\mathrm{O}}$. The first subsequence, i.e., $\boldsymbol{q}_{t,0}^{\mathrm{O}}$, is used as a part of the input of the inner encoder at time t and the other ones, $\boldsymbol{q}_{t,1}^{\mathrm{O}}, \ldots, \boldsymbol{q}_{t,m}^{\mathrm{O}}$, are used in the next inner encoders at times $t+1, \ldots, t+m$, respectively. Thus, at time t the sequence $(\boldsymbol{q}_{t,0}^{\mathrm{O}}, \boldsymbol{q}_{t-1,1}^{\mathrm{O}}, \ldots, \boldsymbol{q}_{t-m,m}^{\mathrm{O}})$ is generated by the current and previous m component encoders, which is permuted using *Interleaver 2* to create the inner encoder input,

$$\mathbf{In}_{\mathrm{E}}^{\mathrm{I}}(t) = \Pi_2(\boldsymbol{q}_{t,0}^{\mathrm{O}}, \boldsymbol{q}_{t-1,1}^{\mathrm{O}}, \ldots, \boldsymbol{q}_{t-m,m}^{\mathrm{O}}). \tag{2}$$

Finally, the SC-SCC encoder output is $\boldsymbol{v}_t = (\boldsymbol{u}_t, \boldsymbol{p}_t^{\mathrm{O}}, \boldsymbol{p}_t^{\mathrm{I}})$, where $\boldsymbol{p}_t^{\mathrm{I}}$ is the inner parity sequence. In this paper, a code rate of $R = 1/3$ is considered. Thus, the output of the inner encoder is punctured such that only K bits of $\boldsymbol{p}_t^{\mathrm{I}}$ are transmitted.

III. Proposed Decoding Schemes for SC-SCCs

Spatially coupled codes can be decoded using window decoding [3], [4]. In this section, we present two decoding algorithms in the context of SC-SCCs: *block-wise* and

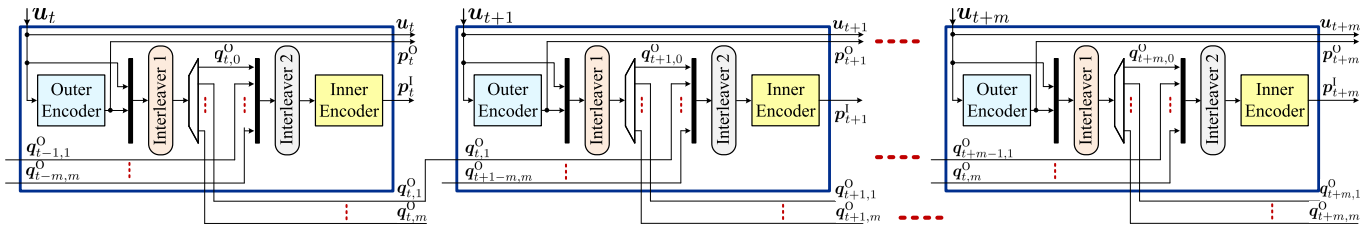


Fig. 3. SC-SCC encoder architecture with coupling memory of m, which is built by spatial coupling of $m+1$ samples of SCC component encoders together.

Algorithm 2 Block-Wise SC-SCC Decoder

$\tilde{\boldsymbol{u}}_t$=**SCSCCDecoder**$(L_{\text{ch}}(\boldsymbol{p}^{\text{O}}_{t'}), L_{\text{ch}}(\boldsymbol{p}^{\text{I}}_{t'}), L_{\text{ch}}(\boldsymbol{u}_{t'}), K, m, W, I_{\text{w}})$
- **for** $I = 1 : I_w$ **do**
  - **for** $t' = t : t + W - 1$ **do**
    - $\mathbf{In1}^{\text{I}}_{\text{D}}(t') = L_{\text{ch}}(\boldsymbol{p}^{\text{I}}_{t'})$
    - $L_{\text{ch}}(\boldsymbol{q}^{\text{O}}_{t'}) = \Pi_1(L_{\text{ch}}(\boldsymbol{u}_{t'}), L_{\text{ch}}(\boldsymbol{p}^{\text{O}}_{t'}))$
    - **for** $i = 0 : m$ **do**
      - $L_{\text{ch}}(\boldsymbol{q}^{\text{O}}_{t',i}) = L_{\text{ch}}(\boldsymbol{q}^{\text{O}}_{t'})(\frac{2Ki}{m+1} : \frac{2K(i+1)}{m+1} - 1)$
    - **end**
    - $\mathbf{In2}^{\text{I}}_{\text{D}}(t') = \Pi_2(L_{\text{ch}}(\boldsymbol{q}^{\text{O}}_{t',0}), \ldots, L_{\text{ch}}(\boldsymbol{q}^{\text{O}}_{t'-m,m}))$
    - $L_{\text{e}}(\boldsymbol{q}^{\text{O}}_{t'}) = \Pi_1(L_{\text{e}}(\boldsymbol{u}_{t'}), L_{\text{e}}(\boldsymbol{p}^{\text{O}}_{t'}))$
    - **for** $j = 0 : m$ **do**
      - $L_{\text{e}}(\boldsymbol{q}^{\text{O}}_{t',j}) = L_{\text{e}}(\boldsymbol{q}^{\text{O}}_{t'})(\frac{2Kj}{m+1} : \frac{2K(j+1)}{m+1} - 1)$
    - **end**
    - $L_{\text{a}}(\boldsymbol{q}^{\text{O}}_{t'}) = \Pi_2(L_{\text{e}}(\boldsymbol{q}^{\text{O}}_{t',0}), \ldots, L_{\text{e}}(\boldsymbol{q}^{\text{O}}_{t'-m,m}))$
    - $\mathbf{In3}^{\text{I}}_{\text{D}}(t') = L_{\text{a}}(\boldsymbol{q}^{\text{O}}_{t'})$
    - $[L_{\text{e}}(\boldsymbol{q}^{\text{I}}_{t'})]$ =**BCJR** $(\mathbf{In1}^{\text{I}}_{\text{D}}(t'), \mathbf{In2}^{\text{I}}_{\text{D}}(t'), \mathbf{In3}^{\text{I}}_{\text{D}}(t'))$
    - $\mathbf{In1}^{\text{O}}_{\text{D}}(t') = L_{\text{ch}}(\boldsymbol{p}^{\text{O}}_{t'})$
    - $\mathbf{In2}^{\text{O}}_{\text{D}}(t') = L_{\text{ch}}(\boldsymbol{u}_{t'})$
    - $L_{\text{e}}(\tilde{\boldsymbol{q}}^{\text{I}}_{t'}) = \Pi_2^{-1}(L_{\text{e}}(\boldsymbol{q}^{\text{I}}_{t'}))$
    - **for** $l = 0 : m$ **do**
      - $L_{\text{e}}(\tilde{\boldsymbol{q}}^{\text{I}}_{t',l}) = L_{\text{e}}(\tilde{\boldsymbol{q}}^{\text{I}}_{t'})(\frac{2Kl}{m+1} : \frac{2K(l+1)}{m+1} - 1)$
    - **end**
    - $L_{\text{a}}(\tilde{\boldsymbol{q}}^{\text{I}}_{t'}) = \Pi_1^{-1}(L_{\text{e}}(\tilde{\boldsymbol{q}}^{\text{I}}_{t',0}), \ldots, L_{\text{e}}(\tilde{\boldsymbol{q}}^{\text{I}}_{t'+m,m}))$
    - $\mathbf{In3}^{\text{O}}_{\text{D}}(t') = L_{\text{a}}(\tilde{\boldsymbol{q}}^{\text{I}}_{t'})$
    - $[L_{\text{e}}(\boldsymbol{p}^{\text{O}}_{t'}), L_{\text{e}}(\boldsymbol{u}_{t'})]$=**BCJR**$(\mathbf{In1}^{\text{O}}_{\text{D}}(t'), \mathbf{In2}^{\text{O}}_{\text{D}}(t'), \mathbf{In3}^{\text{O}}_{\text{D}}(t'))$
  - **end**
- **end**
- $\tilde{\boldsymbol{u}}_t = \text{Sign}(L_{\text{ch}}(\boldsymbol{u}_t) + L_{\text{e}}(\boldsymbol{u}_t) + L_{\text{a}}(\tilde{\boldsymbol{q}}^{\text{I}}_t)(0 : K-1))$

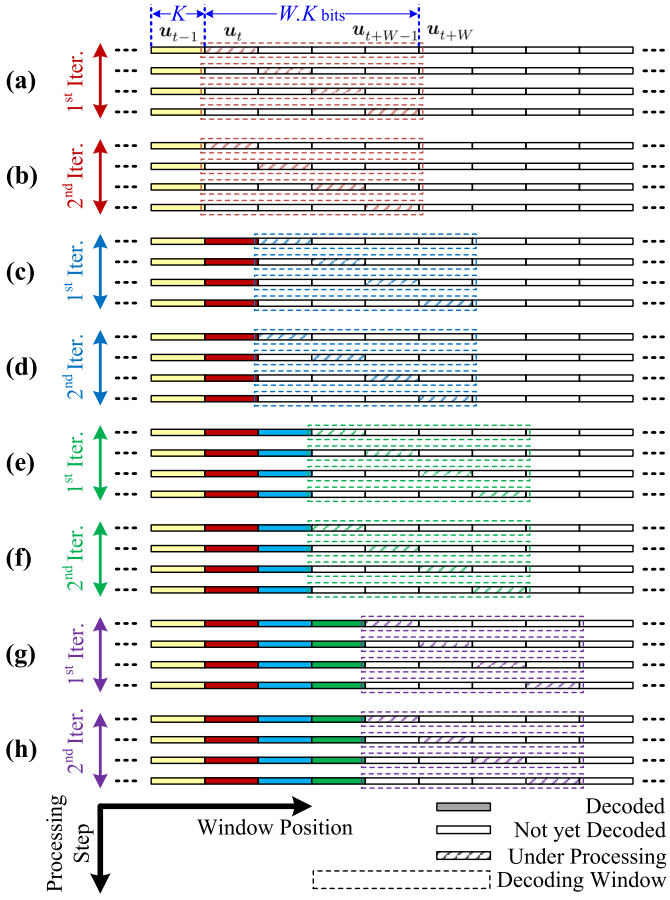


Fig. 4. Block-wise decoding of SC-SCC for $W = 4$ and $I_{\text{w}} = 2$. The dashed rectangles specify the ongoing decoding window, moving from left to right.

window-wise decoding. The corresponding complexity analysis, decoding performance, VLSI architectures, and implementation results are discussed in Section IV, V, VI, and VII, respectively.

A. Block-Wise SC-SCC Decoder

We have defined the *window size*, W, as the number of code blocks to be processed in a decoding window and I_{w} as the *number of iterations per window position*. The proposed block-wise SC-SCC decoder is formulated in Algorithm 2 and its processing flow is illustrated in Fig. 4 for $\{W = 4, I_{\text{w}} = 2\}$, where the window is shown by dashed rectangles, moving from left to right. Let us consider a decoding window of size W, which starts at time instant t and ends at $t + W - 1$. The leftmost block inside a window, which is the first block to be decoded, is referred to as *target block*. The decoding of blocks with time indices $t' = t, t+1, \ldots, t+W-1$, is performed as follows. The decoding is started by the inner decoder, which receives three inputs. The first one contains the channel LLR values of inner parity sequence, $\mathbf{In1}^{\text{I}}_{\text{D}}(t') = L_{\text{ch}}(\boldsymbol{p}^{\text{I}}_{t'})$. To generate the second input, channel LLRs of information and outer parity bits are permuted using *Interleaver 1* as

$$L_{\text{ch}}(\boldsymbol{q}^{\text{O}}_{t'}) = \Pi_1(L_{\text{ch}}(\boldsymbol{u}_{t'}), L_{\text{ch}}(\boldsymbol{p}^{\text{O}}_{t'})). \tag{3}$$

Then, the sequence $L_{\text{ch}}(\boldsymbol{q}^{\text{O}}_{t'})$ is divided into $m+1$ parts of equal size, named as $L_{\text{ch}}(\boldsymbol{q}^{\text{O}}_{t',0}), L_{\text{ch}}(\boldsymbol{q}^{\text{O}}_{t',1}), \ldots, L_{\text{ch}}(\boldsymbol{q}^{\text{O}}_{t',m})$. The first subsequence, i.e., $L_{\text{ch}}(\boldsymbol{q}^{\text{O}}_{t',0})$, is used in the inner decoder at time instant t' and the other ones, $L_{\text{ch}}(\boldsymbol{q}^{\text{O}}_{t',1}), \ldots, L_{\text{ch}}(\boldsymbol{q}^{\text{O}}_{t',m})$, are used in the next inner decoders at time $t'+1, \ldots, t'+m$, respectively. At time instant t', the corresponding $m+1$ subsequences are concatenated together and permuted using

Algorithm 3 Window-Wise SC-SCC Decoder

$\tilde{\boldsymbol{u}}_t$=**SCSCCDecoder**$(L_{\rm ch}(\boldsymbol{p}^{\rm O}_{t'}), L_{\rm ch}(\boldsymbol{p}^{\rm I}_{t'}), L_{\rm ch}(\boldsymbol{u}_{t'}), K, m, W, I_{\rm w})$
  for $I = 1 : I_w$ **do**
    for $t' = t : t + W - 1$ **do**
      $L_{\rm ch}(\boldsymbol{q}^{\rm O}_{t'}) = \Pi_1(L_{\rm ch}(\boldsymbol{u}_{t'}), L_{\rm ch}(\boldsymbol{p}^{\rm O}_{t'}))$
      for $i = 0 : m$ **do**
        $L_{\rm ch}(\boldsymbol{q}^{\rm O}_{t',i}) = L_{\rm ch}(\boldsymbol{q}^{\rm O}_{t'})(\frac{2Ki}{m+1} : \frac{2K(i+1)}{m+1} - 1)$
      end
      $\mathbf{temp}^{\rm I}_{t'} = \Pi_2(L_{\rm ch}(\boldsymbol{q}^{\rm O}_{t',0}), \ldots, L_{\rm ch}(\boldsymbol{q}^{\rm O}_{t'-m,m}))$
      $L_{\rm e}(\boldsymbol{q}^{\rm O}_{t'}) = \Pi_1(L_{\rm e}(\boldsymbol{u}_{t'}), L_{\rm e}(\boldsymbol{p}^{\rm O}_{t'}))$
      for $j = 0 : m$ **do**
        $L_{\rm e}(\boldsymbol{q}^{\rm O}_{t',j}) = L_{\rm e}(\boldsymbol{q}^{\rm O}_{t'})(\frac{2Kj}{m+1} : \frac{2K(j+1)}{m+1} - 1)$
      end
      $L_{\rm a}(\boldsymbol{q}^{\rm O}_{t'}) = \Pi_2(L_{\rm e}(\boldsymbol{q}^{\rm O}_{t',0}), \ldots, L_{\rm e}(\boldsymbol{q}^{\rm O}_{t'-m,m}))$
    end
    $\mathbf{In1}^{\rm I}_{\rm D} = [L_{\rm ch}(\boldsymbol{p}^{\rm I}_{t}), L_{\rm ch}(\boldsymbol{p}^{\rm I}_{t+1}), \ldots, L_{\rm ch}(\boldsymbol{p}^{\rm I}_{t+W-1})]$
    $\mathbf{In2}^{\rm I}_{\rm D} = [\mathbf{temp}^{\rm I}_{t}, \mathbf{temp}^{\rm I}_{t+1}, \ldots, \mathbf{temp}^{\rm I}_{t+W-1}]$
    $\mathbf{In3}^{\rm I}_{\rm D} = [L_{\rm a}(\boldsymbol{q}^{\rm O}_{t}), L_{\rm a}(\boldsymbol{q}^{\rm O}_{t+1}), \ldots, L_{\rm a}(\boldsymbol{q}^{\rm O}_{t+W-1})]$
    $[L_{\rm e}(\boldsymbol{q}^{\rm I})] =$ **BCJR** $(\mathbf{In1}^{\rm I}_{\rm D}, \mathbf{In2}^{\rm I}_{\rm D}, \mathbf{In3}^{\rm I}_{\rm D})$

    for $t' = t : t + W - 1$ **do**
      $L_{\rm e}(\boldsymbol{q}^{\rm I}_{t'}) = L_{\rm e}(\boldsymbol{q}^{\rm I})(2K(t'-t) : 2K(t'-t+1) - 1)$
      $L_{\rm e}(\tilde{\boldsymbol{q}}^{\rm I}_{t'}) = \Pi_2^{-1}(L_{\rm e}(\boldsymbol{q}^{\rm I}_{t'}))$
      for $l = 0 : m$ **do**
        $L_{\rm e}(\tilde{\boldsymbol{q}}^{\rm I}_{t',l}) = L_{\rm e}(\tilde{\boldsymbol{q}}^{\rm I}_{t'})(\frac{2Kl}{m+1} : \frac{2K(l+1)}{m+1} - 1)$
      end
      $L_{\rm a}(\tilde{\boldsymbol{q}}^{\rm I}_{t'}) = \Pi_1^{-1}(L_{\rm e}(\tilde{\boldsymbol{q}}^{\rm I}_{t',0}), \ldots, L_{\rm e}(\tilde{\boldsymbol{q}}^{\rm I}_{t'+m,m}))$
    end
    $\mathbf{In1}^{\rm O}_{\rm D} = [L_{\rm ch}(\boldsymbol{p}^{\rm O}_{t}), L_{\rm ch}(\boldsymbol{p}^{\rm O}_{t+1}), \ldots, L_{\rm ch}(\boldsymbol{p}^{\rm O}_{t+W-1})]$
    $\mathbf{In2}^{\rm O}_{\rm D} = [L_{\rm ch}(\boldsymbol{u}_{t}), L_{\rm ch}(\boldsymbol{u}_{t+1}), \ldots, L_{\rm ch}(\boldsymbol{u}_{t+W-1})]$
    $\mathbf{In3}^{\rm O}_{\rm D} = [L_{\rm a}(\tilde{\boldsymbol{q}}^{\rm I}_{t}), L_{\rm a}(\tilde{\boldsymbol{q}}^{\rm I}_{t+1}), \ldots, L_{\rm a}(\tilde{\boldsymbol{q}}^{\rm I}_{t+W-1})]$
    $[L_{\rm e}(\boldsymbol{p}^{\rm O}), L_{\rm e}(\boldsymbol{u})] =$ **BCJR** $(\mathbf{In1}^{\rm O}_{\rm D}, \mathbf{In2}^{\rm O}_{\rm D}, \mathbf{In3}^{\rm O}_{\rm D})$
    for $t' = t : t + W - 1$ **do**
      $L_{\rm e}(\boldsymbol{p}^{\rm O}_{t'}) = L_{\rm e}(\boldsymbol{p}^{\rm O})(K(t'-t) : K(t'-t+1) - 1)$
      $L_{\rm e}(\boldsymbol{u}_{t'}) = L_{\rm e}(\boldsymbol{u})(K(t'-t) : K(t'-t+1) - 1)$
    end
  end
  $\tilde{\boldsymbol{u}}_t = {\rm Sign}(L_{\rm ch}(\boldsymbol{u}_t) + L_{\rm e}(\boldsymbol{u}_t) + L_{\rm a}(\tilde{\boldsymbol{q}}^{\rm I}_t)(0 : K-1))$

Interleaver 2 to produce the second input of the inner decoder,

$$\mathbf{In2}^{\rm I}_{\rm D}(t') = \Pi_2(L_{\rm ch}(\boldsymbol{q}^{\rm O}_{t',0}), \ldots, L_{\rm ch}(\boldsymbol{q}^{\rm O}_{t'-m,m})). \quad (4)$$

The third input contains the a-priori LLRs, $L_{\rm a}(\boldsymbol{q}^{\rm O}_{t'})$, which is obtained using the extrinsic LLRs as stated in Algorithm 2. Eventually, the inner decoder produces the extrinsic LLRs, $L_{\rm e}(\boldsymbol{q}^{\rm I}_{t'})$, and sends them back to the connected outer decoders.

Similarly, the outer decoder at time t' receives three inputs, where the first and second ones are the channel LLRs of outer parity and information sequences, ($\mathbf{In1}^{\rm O}_{\rm D}(t') = L_{\rm ch}(\boldsymbol{p}^{\rm O}_{t'})$, $\mathbf{In2}^{\rm O}_{\rm D}(t') = L_{\rm ch}(\boldsymbol{u}_{t'})$), and the third one is the a-priori LLRs, $L_{\rm a}(\tilde{\boldsymbol{q}}^{\rm I}_{t'})$. The outer decoder uses them to generate the extrinsic LLRs for information and outer parity bits ($L_{\rm e}(\boldsymbol{u}_{t'})$, $L_{\rm e}(\boldsymbol{p}^{\rm O}_{t'})$) and sends them to the connected inner decoders.

So far the first block, $\boldsymbol{u}_t$, in the current window is processed, which corresponds to the first row of Fig. 4(a). To finish the first iteration, the above process will be applied for the remaining blocks inside the current window, $\boldsymbol{u}_{t+1}, \ldots, \boldsymbol{u}_{t+W-1}$,

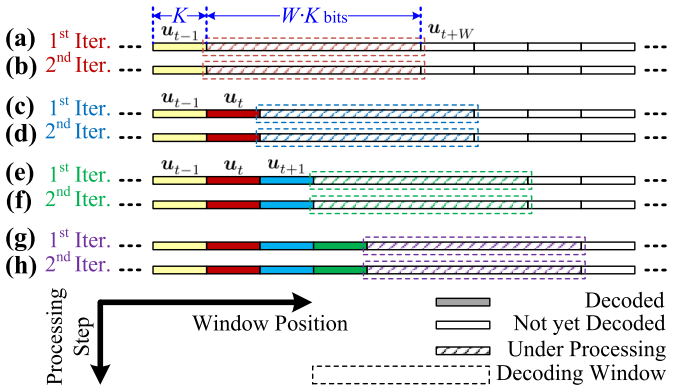


Fig. 5. Window-wise decoding of SC-SCC for $W = 4$ and $I_{\rm w} = 2$. The dashed rectangles specify the ongoing decoding window, moving from left to right.

which correspond to the second, third and fourth rows in Fig. 4(a) (in this example $W = 4$). Then, the same procedure is repeated for the current window in the next iteration, as shown in Fig. 4(b). After $I_{\rm w}$ iterations[1] (in this example $I_{\rm w} = 2$), the hard decision is made to decode the target block, as

$$\tilde{\boldsymbol{u}}_t = {\rm Sign}(L_{\rm ch}(\boldsymbol{u}_t) + L_{\rm e}(\boldsymbol{u}_t) + L_{\rm a}(\tilde{\boldsymbol{q}}^{\rm I}_t)(0 : K-1)).^2 \quad (5)$$

After that, the window is moved by one block, which starts at time $t+1$ and ends at $t+W$, as shown in Fig. 4(c). The same process will be repeated to decode the new target block, $\boldsymbol{u}_{t+1}$, (see Fig. 4(c)-(d)). Similarly, $\boldsymbol{u}_{t+2}$ and $\boldsymbol{u}_{t+3}$ will be decoded as shown in Fig. 4(e)-(f) and Fig. 4(g)-(h), respectively. Thus, all the blocks inside the initial window, in Fig. 4(a), will be decoded after $W \cdot I_{\rm w}$ iterations (8 iterations in this example).

B. Window-Wise SC-SCC Decoder

The second proposed decoding scheme is window-wise decoding, where the main idea is to perform the BCJR through the whole window at once instead of a block-wise manner. This scheme is detailed in Algorithm 3 and an example of its processing flow is shown in Fig. 5 for $\{W = 4,\ I_{\rm w} = 2\}$. To explain this scheme, we consider the same window as the one in Fig. 4, which starts at time t and ends at $t+W-1$. The decoding is started by the inner decoder, which receives three inputs. The first one is the channel LLRs of the inner parity sequences corresponding to all the blocks inside the window,

$$\mathbf{In1}^{\rm I}_{\rm D} = [L_{\rm ch}(\boldsymbol{p}^{\rm I}_{t}), L_{\rm ch}(\boldsymbol{p}^{\rm I}_{t+1}), \ldots, L_{\rm ch}(\boldsymbol{p}^{\rm I}_{t+W-1})]. \quad (6)$$

To generate the second input, the same operations as (3) and (4) will be done for the channel LLRs of information and outer parity sequences at each time instant $t' = t, \ldots, t+W-1$. Then, the generated sequences for the corresponding time instants, i.e., $\mathbf{temp}^{\rm I}_{t'}$, will be concatenated together and used as $\mathbf{In2}^{\rm I}_{\rm D}$. The third input contains the a-priori information,

$$\mathbf{In3}^{\rm I}_{\rm D} = [L_{\rm a}(\boldsymbol{q}^{\rm O}_{t}), L_{\rm a}(\boldsymbol{q}^{\rm O}_{t+1}), \ldots, L_{\rm a}(\boldsymbol{q}^{\rm O}_{t+W-1})], \quad (7)$$

which is constructed in the same way as the second input (see Algorithm 3). The inner decoder generates the extrinsic LLRs, $L_{\rm e}(\boldsymbol{q}^{\rm I})$, which will be sent to the connected outer decoders.

[1]We consider the number of iterations as the stopping criterion of decoder.
[2]The notation $\boldsymbol{a}(i : j)$ represents the i-th entry to the j-th entry of $\boldsymbol{a}$.

At the outer decoder side, the first and second inputs are generated by concatenating the LLRs of the outer parity and information sequences, respectively (see Algorithm 3). To generate the third input, the a-priori LLR sequences of all blocks inside the current window are separately calculated. Then, they are used to create the third input of outer decoder,

$$\mathbf{In3}_{\mathrm{D}}^{\mathrm{O}} = [L_{\mathrm{a}}(\tilde{\boldsymbol{q}}_{t}^{\mathrm{I}}), L_{\mathrm{a}}(\tilde{\boldsymbol{q}}_{t+1}^{\mathrm{I}}), \ldots, L_{\mathrm{a}}(\tilde{\boldsymbol{q}}_{t+W-1}^{\mathrm{I}})]. \tag{8}$$

The outer decoder employs these inputs and produces extrinsic LLRs for the information and outer parity bits, $L_{\mathrm{e}}(\boldsymbol{u}_{t'})$ and $L_{\mathrm{e}}(\boldsymbol{p}_{t'}^{\mathrm{O}})$, and sends them to the connected inner decoders.

At this point, the current window has been processed once, which corresponds to Fig. 5(a). The above process is repeated in the next iteration for the same window as depicted in Fig. 5(b). After I_{w} iterations (in this example $I_{\mathrm{w}} = 2$), the hard decision is made using (5) to decode the K leftmost bits in the window, i.e., $\boldsymbol{u}_t$. Then, the window is moved by K bits, which starts at time $t+1$ and ends at $t+W$, as shown in Fig. 5(c). The same decoding process will be performed to decode the K leftmost bits of the new window, i.e., $\boldsymbol{u}_{t+1}$ (see Fig. 5(c)-(d)). This procedure is continued to decode $\boldsymbol{u}_{t+2}$ and $\boldsymbol{u}_{t+3}$ following Fig. 5(e)-(f) and Fig. 5(g)-(h), respectively. Similar to the Algorithm 2, the whole window in Fig. 5(a) will be decoded after $W \cdot I_{\mathrm{w}}$ iterations (8 iterations in this example).

C. Latency and Constraint Length

In this paper, we have defined two types of latency: *structural latency*, $\mathcal{L}^{\mathrm{S}}$, and *decoding latency*, $\mathcal{L}^{\mathrm{D}}$. The *structural latency* is a code-related parameter and obtained as

$$\mathcal{L}_{\mathrm{SC}}^{\mathrm{S}} = W \cdot K_{\mathrm{SC}}, \quad \text{(bit)} \tag{9}$$

for spatially coupled codes while for the uncoupled ones is

$$\mathcal{L}_{\mathrm{UC}}^{\mathrm{S}} = K_{\mathrm{UC}}, \quad \text{(bit)}, \tag{10}$$

where K_{SC} and K_{UC} are the information block lengths of spatially coupled and uncoupled codes, respectively.

The second type of latency, $\mathcal{L}^{\mathrm{D}}$, is determined by decoding algorithm and corresponding VLSI architecture, which will be analyzed for various decoders in Section VI and VII.

Another code-related parameter is *constraint length*, which specifies the strength of the codes and is defined as

$$\mathcal{C} = K_{\mathrm{SC}} \cdot (m+1), \quad m < W, \ 2K_{\mathrm{SC}}. \tag{11}$$

In the following, K is used without subscript and the context determines if K is related to the uncoupled or coupled codes.

IV. Computational Complexity Analysis

We analyze the complexity of the SC-SCC decoder based on Algorithm 2. To this end, the number of required operations to decode a code block with K information bits is enumerated. Similar results will be obtained for the complexity of Algorithm 3, since both algorithms perform the same operations.

TABLE I

Computational Complexity per Decoded Bit in Log-MAP BCJR Algorithm for One Iteration and One Trellis Step

	# Addition/Subtraction	# Comparison
$\mathcal{O}_{\mathcal{A}}$	$2 \cdot l \cdot 2^{E_{\mathrm{m}}}$	$l \cdot 2^{E_{\mathrm{m}}} - 1$
$\mathcal{O}_{\mathcal{B}}$	$2 \cdot l \cdot 2^{E_{\mathrm{m}}}$	$l \cdot 2^{E_{\mathrm{m}}} - 1$
$\mathcal{O}_{\Gamma}$	$2 \cdot l \cdot 2^{E_{\mathrm{m}}}$	0
$\mathcal{O}_{\mathcal{M}}$	$2 \cdot l \cdot 2^{E_{\mathrm{m}}}$	0
$\mathcal{O}_{\mathrm{APP}}$	1	$l \cdot 2^{E_{\mathrm{m}}} - 2$
$\mathcal{O}_{L_{\mathrm{e}}}$	2	0

A. Computational Complexity of BCJR

In this analysis, the log-MAP BCJR is considered, which has the same decoding performance as the MAP BCJR but less complexity. Let us consider a trellis with $2^{E_{\mathrm{m}}}$ states, where E_{m} is the encoder memory size, e.g., $E_{\mathrm{m}} = 2$ in Fig. 1(b). The probability of transition from the state at time $t-1$, i.e., S_r, to the one at time t, i.e., S_s, called branch metric, is given by

$$\Gamma_t(S_r, S_s) = \frac{1}{2} u_t \cdot L_{\mathrm{a}}(u_t) + \frac{1}{2} L_{\mathrm{c}} \cdot \Big(u_t \cdot L_{\mathrm{ch}}(u_t) + p_t \cdot L_{\mathrm{ch}}(p_t) \Big). \tag{12}$$

Here, $L_{\mathrm{a}}(u_t)$ and $L_{\mathrm{ch}}(u_t)$ are the a-priori and channel LLRs of the information bit u_t, $L_{\mathrm{ch}}(p_t)$ is the channel LLR of the inner/outer parity bit p_t, and L_{c} denotes the channel reliability measure. Having considered a single trellis step, forward and backward recursion metrics, α and β, can be calculated as

$$\mathcal{A}_t(S_s) = \log \alpha_t(S_s) = \max^*(\mathcal{A}_{t-1}(S_i) + \Gamma_t(S_i, S_s)), \tag{13}$$

$$\mathcal{B}_{t-1}(S_r) = \log \beta_{t-1}(S_r) = \max^*(\mathcal{B}_t(S_i) + \Gamma_t(S_r, S_i)). \tag{14}$$

In these equations, $i = 1, \ldots, 2^{E_{\mathrm{m}}}$ and the max* operator is

$$\max^*(a, b) \triangleq \max(a, b) + \log(1 + e^{-|a-b|}). \tag{15}$$

where "max" performs a comparison and the correcting term, $\log(1 + e^{-|a-b|})$, is usually obtained from a look-up table (LUT).[2] Then, the state transition metrics are calculated as

$$\mathcal{M}_t(S_r, S_s) = \mathcal{A}_{t-1}(S_r) + \Gamma_t(S_r, S_s) + \mathcal{B}_t(S_s) \tag{16}$$

Finally, the LLR of a posteriori probability (APP) for each information bit, u_t, is given by

$$L(u_t|\boldsymbol{v}_t) = \max_{S^-}{}^* \ (\mathcal{M}_t(S_r, S_s)) - \max_{S^+}{}^* \ (\mathcal{M}_t(S_r, S_s)), \tag{17}$$

in which S^- and S^+ are the sets of state transitions such that $(S_r, S_s) \in S^-$ and $(S_r, S_s) \in S^+$ are caused by $u_t = 0$ and $u_t = 1$, respectively. In the context of turbo decoders, the extrinsic information, $L_{\mathrm{e}}(u_t)$, is obtained by subtracting the channel LLR and a priori of u_t from the APP in (17). The extrinsic information of inner/outer decoder will be permuted and used as the a priori information of outer/inner decoder.

The number of required operations to calculate $\mathcal{A}$, $\mathcal{B}$, Γ, $\mathcal{M}$, L_{e}, and APP values for each trellis step are listed in Table I.

[2] In practice, eight values of $|a-b|$ between 0 and 5 are stored in the LUT.

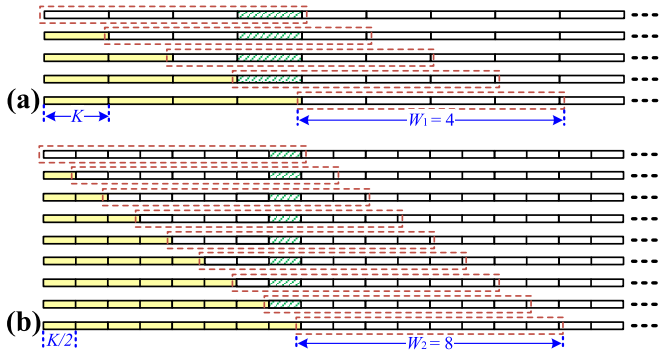


Fig. 6. Window decoding approach for two fixed-latency scenarios with block length and window size of (a) $K_1 = K$, $W_1 = 4$ and (b) $K_2 = K/2$, $W_2 = 8$. The dashed rectangles specify the ongoing decoding window and the blocks, which are located in the left side of the window are already decoded.

In this table, the complexity due to the normalization of $\mathcal{A}$ and $\mathcal{B}$ are included in $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{B}}$, which is $2^{E_\mathrm{m}} - 1$ comparisons and 2^{E_m} additions for each. Thus, the complexity of BCJR to decode an information block of K bits in one iteration is

$$\mathcal{O}_\mathrm{D} = K \cdot (\mathcal{O}_\Gamma + \mathcal{O}_{\mathcal{A}} + \mathcal{O}_{\mathcal{B}} + \mathcal{O}_{\mathcal{M}} + \mathcal{O}_\mathrm{APP} + \mathcal{O}_{L_\mathrm{e}}). \quad (18)$$

B. Fixed-Complexity SC-SCC Decoder

Taking the overlaps between decoding windows in Fig. 4 into account, each block is processed $W \cdot I_\mathrm{w}$ times, regardless of its length. Thus, the computational complexity of proposed SC-SCC decoder to decode a window of size W is

$$\mathcal{O}_\mathrm{SCSCC} = W^2 \cdot (3\mathcal{O}_\mathrm{D}) \cdot I_\mathrm{w}, \quad (19)$$

where the factor 3 is because the inner trellis length is $2K$. Therefore, the computational complexity per decoded bit is

$$\mathcal{O}_\mathrm{bit} = \frac{W \cdot (3\mathcal{O}_\mathrm{D}) \cdot I_\mathrm{w}}{K}, \quad (20)$$

which will be proportional to W and I_w since the decoder complexity ($\mathcal{O}_\mathrm{D}$) is proportional to K, as defined in (18).

Following Algorithm 2, the window is moved by one block after I_w iterations, meaning that the amount of overlap between two successive windows depends on block length. To clarify this concept, Fig. 6(a) and (b) show the decoding flow for two scenarios, $\{K_1 = K, W_1 = 4\}$ and $\{K_2 = K/2, W_1 = 8\}$, which have the same structural latency of $\mathcal{L} = 4K$. Since the window in Fig. 6(a) includes larger blocks, it moves by larger steps and thus has less overlaps. Therefore, if the same I_w is used for both cases in Fig. 6, which is a common assumption in the literature, the scenario in Fig. 6(b) would have higher complexity than the one in Fig. 6(a) since it has larger W.

In order to have a fair comparison between the performance of different SC-SCC schemes, the same complexity must be considered regardless of their block length and window size. To this end, we have defined the effective number of iterations,

$$I_\mathrm{eff} = W \cdot I_\mathrm{w}, \quad (21)$$

which specifies how often the BCJR algorithm is executed to decode a certain code block. The goal is to adjust the I_w such that the same I_eff is achieved for all scenarios, which according to (20) and (21) results in the same computational complexity.

TABLE II

DIFFERENT SCENARIOS OF SC-SCCS WITH THE SAME LATENCY ($\mathcal{L}^\mathrm{S}$), CONSTRAINT LENGTH ($\mathcal{C}$), AND COMPUTATIONAL COMPLEXITY

$\mathcal{L}^{\mathrm{S}\ddagger} = 1024$, $\mathcal{C}^\dagger = 512$	Block Length (K)	128	64
	Window Size (W)	8	16
	Coupling Memory (m)	3	7
	#Iterations per Window (I_w)*	6	3
$\mathcal{L}^\mathrm{S} = 2048$, $\mathcal{C} = 1024$	Block Length (K)	256	128
	Window Size (W)	8	16
	Coupling Memory (m)	3	7
	#Iterations per Window (I_w)	6	3
$\mathcal{L}^\mathrm{S} = 4096$, $\mathcal{C} = 2048$	Block Length (K)	512	2562
	Window Size (W)	8	16
	Coupling Memory (m)	3	7
	#Iterations per Window (I_w)	6	3
$\mathcal{L}^\mathrm{S} = 8192$, $\mathcal{C} = 4096$	Block Length (K)	1024	512
	Window Size (W)	8	16
	Coupling Memory (m)	3	7
	#Iterations per Window (I_w)	6	3
$\mathcal{L}^\mathrm{S} = 16384$, $\mathcal{C} = 8192$	Block Length (K)	2048	1024
	Window Size (W)	8	16
	Coupling Memory (m)	3	7
	#Iterations per Window (I_w)	6	3

‡, †, * Calculated using (9), (11), and (22)

For example, to have the same computational complexity in both scenarios in Fig. 6, the number of iterations per window position in the second scenario should be set to

$$I_{\mathrm{w}_2} = (W_1/W_2) \cdot I_{\mathrm{w}_1}, \quad (22)$$

where W_1 and I_{w_1} correspond to the scenario in Fig. 6(a).

V. PERFORMANCE EVALUATION

To evaluate the performance of SC-SCCs, we have defined five scenarios in Table II. In each of them latency, constraint length, and complexity are fixed while different combinations of K, W, and m are considered. In all cases, $I_\mathrm{eff} = 48$ is used to have the same complexity while $I_\mathrm{w} \geq 1$. The number of iterations per window, I_w, is adjusted following (22). As shown in Table II, less number of iterations per window position is used for smaller K, which is due to the larger overlaps between successive windows as explained in Section IV.

A. Effect of Coupling Memory on the Performance

To improve the decoding performance of SC-SCC, the code strength and thus the constraint length, $\mathcal{C}$, should be increased. To this end, according to (11), either block length or coupling memory should be increased. The first alternative, increasing K, will increase the latency as stated in (9), which is not appealing in many applications. However, the second alternative, increasing m, will not change the latency and complexity (see (9) and (19)). Therefore, it is important to find the optimal coupling memory, which leads to the best performance for a given window size and block length.

We have investigated this concept for Algorithm 3 with $\{\mathcal{L} = 1024, K = 32, W = 32\}$ and $\{\mathcal{L} = 8192, K = 512, W = 16\}$, as an example, where the results are depicted in the results are shown in Fig. 7(a) and (b), respectively. As a result, by increasing the coupling memory up to $m = W/2 - 1$ the waterfall performance will be improved considerably and the

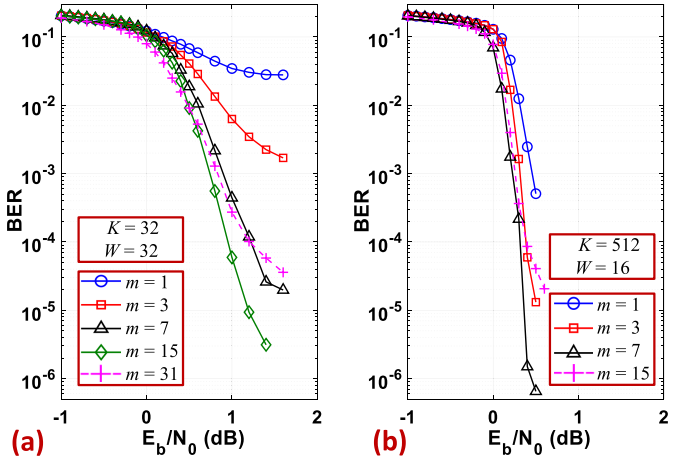


Fig. 7. The effect of coupling memory on decoding performance of SC-SCC schemes. Algorithm 3 with structural latency (a) $\mathcal{L}^{\mathrm{S}} = 1024$ and (b) $\mathcal{L}^{\mathrm{S}} = 8192$ is used. The same complexity, $I_{\mathrm{eff}} = 80$, is assumed in both cases.

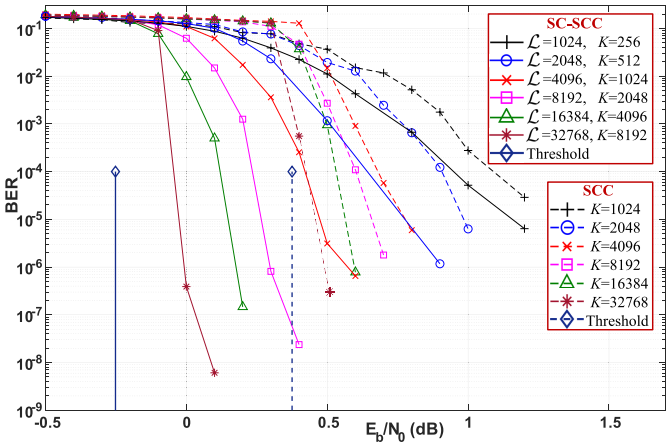


Fig. 8. Performance comparison between proposed SC-SCCs and uncoupled SCCs for different block lengths and structural latencies. Here, Algorithm 3 is used, as an example. The same complexity is assumed for all cases ($I_{\mathrm{eff}} = 80$).

error floor goes down to the lower BERs [5]. More specifically, Fig. 7(a) shows that at the SNR of 1.2 dB the BER can be improved from 3×10^{-2} to 3×10^{-6} if the coupling memory is increased from $m = 1$ to $m = 15$.

However, if $m > W/2 - 1$ the performance will be degraded, as shown with dashed curves in Fig. 7. In such a case the performance of the decoder cannot fully exploit the code. Thus, for a given $\{K, W\}$ the coupling memory of $m = W/2 - 1$ leads to the best performance without compromising latency and complexity. Similar improvement is achieved for Algorithm 2 by using higher m. As described in Section II-C the sequence in (1), which has $2K$ bits, is divided into $m + 1$ portions. Thus, the coupling memory should be chosen such that $m + 1 \leq W$ and divides $2K$.

B. Performance Comparison With Uncoupled Codes

The BER performance of the proposed SC-SCC schemes and the uncoupled ensembles, SCC, are compared in Fig. 8 for different structural latencies and block lengths, where Algorithm 3 is used, as an example. To have a fair comparison, the same complexity, i.e., equal I_{eff}, is considered for all cases.

It can be seen that spatial coupling significantly improves the performance of the SCC and brings it much closer to the capacity. In Fig. 8, the asymptotic decoding thresholds of the SCC and SC-SCC ensembles for the AWGN channel are illustrated using vertical lines. Having considered a fixed block length, the SC-SCC achieves around 1 dB better performance than the corresponding SCC scheme. Moreover, in case of equal latency i.e., $K_{\mathrm{UC}} = \mathcal{L}^{\mathrm{S}}_{\mathrm{SC}}$, the SC-SCC still has around 0.5 dB better performance than the SCC at the BER of 10^{-4}.

It is worth mentioning that, even with a lower latency, the performance of SC-SCC is better than the SCC. For example, the SC-SCC with $\mathcal{L}^{\mathrm{S}} = 8192$ has remarkably better performance than the SCC with $\mathcal{L}^{\mathrm{S}} = 32768, 16384$ in Fig. 8. This means that by just increasing the block length and latency, the SCC cannot achieve better performance than the SC-SCC.

C. Performance Comparison of Different SC-SCC Decoders

The proposed SC-SCC decoding schemes, i.e., Algorithm 2 and Algorithm 3, benefit form the spatial coupling such that the SC-SCCs achieve better performance than the uncoupled SCCs, as shown in Fig. 8. This performance improvement is achieved by employing larger m, without increasing the latency and complexity, as depicted in Fig. 7.

To select the efficient decoder, the decoding performance as well as the hardware-related metrics, which are discussed in Section VI-VII, should be considered jointly. Therefore, it is important to compare the decoding performance of Algorithm 2 and Algorithm 3, since depending on K, W, and m they may have different performance. We have performed this comparison in Fig. 9 for all the cases in Table II. It can be seen that in case of small block lengths, the gap between the BER of Algorithm 2 and Algorithm 3 is noticeable and Algorithm 3 improves the performance considerably. This is due to the fact that executing the BCJR for a very short trellis, which is the case in Algorithm 2, results in a poor performance at the boundaries between blocks. This is because of the unreliable states at the start and end of each trellis, therefore, the bits which are close to the boundaries will have a weak protection. This problem can be resolved by employing Algorithm 3 in which the trellis length becomes large and the boundary states are more reliable. Note that a small or large K is relative to the latency. For example, $K = 128$ is considered as a large K in case of $\mathcal{L}^{\mathrm{S}} = 1024$, while it is small for $\mathcal{L}^{\mathrm{S}} = 8192$.

In these evaluations the BER curves are used, however, it has been shown in [6] that similar results will be obtained if block error rate (BLER) is used as the measure of performance comparison. As an example, we have included one BLER curve for each latency scenario in Fig. 9, which follows the corresponding BER curve in a different SNR range.

D. Performance Comparison With LDPC and Polar Codes

Performing a fair comparison between different code types is challenging since the design parameters are defined differently in each of them. However, to give an intuition, we have selected different code types from the literature, which have similar design parameters as the proposed SC-SCCs.

To this end, we have taken LDPC codes from 5G with $(K, R) = \{(256, 1/4), (128, 1/4), (64, 1/4)\}$. According to

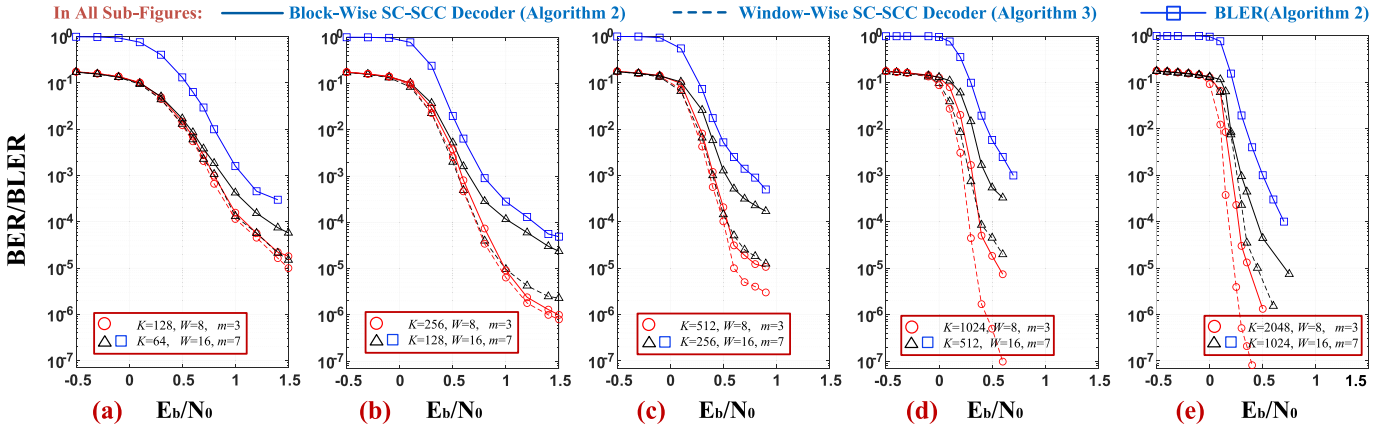


Fig. 9. BER/BLER performance of the scenarios in Table II, with the fixed latency and constraint of (a) $\mathcal{L} = 1024, \mathcal{C} = 512$, (b) $\mathcal{L} = 2048, \mathcal{C} = 1024$, (c) $\mathcal{L} = 4096, \mathcal{C} = 2048$, (d) $\mathcal{L} = 8192, \mathcal{C} = 4096$, and (e) $\mathcal{L} = 16384, \mathcal{C} = 8192$. The same complexity is considered for all scenarios by choosing $I_{\text{eff}} = 64$.

the simulation results in [7], these codes achieve BER $= \{10^{-3}, 9\times10^{-3}, 3\times10^{-2}\}$ at SNR of 1 dB after 200 iterations. However, the BER of SC-SCCs with the same K and $R = 1/3$ falls in the range of $10^{-5} - 10^{-4}$ (see Fig. 9) while they have higher code rate and lower number of iterations, i.e., $I_{\text{eff}} = 64$. Thus, the SC-SCCs will achieve even lower BERs for $R = 1/4$ and higher I_{eff} compared to the mentioned LDPC codes.

Moreover, we have considered 5G polar codes with $(K, R) = \{(256, 1/4), (128, 1/4), (64, 1/4)\}$, which achieve BER $= \{0.07, 0.08, 0.1\}$ at the SNR of 1 dB [7]. Also, for the polar codes with $(K, R) = \{(200, 1/3), (2048, 1/3)\}$ the BLER $= \{0.09, 0.05\}$ is achieved as demonstrated in [8] and [9]. However, the corresponding BER and BLER of SC-SCC with similar K, R, and SNR fall in the range of $10^{-5}-10^{-4}$ and $10^{-4}-10^{-3}$, respectively, as shown in Fig. 9.

VI. Proposed VLSI Architectures for SCC and SC-SCC Decoders

In this section, we investigate the hardware realization of the SCC and SC-SCC decoding algorithms. First, we explain architectural choices for the inner and outer decoders as well as the respective area and latency considerations in Section VI-A. Then, in Section VI-B, we employ these kernels to realize the proposed decoding schemes. Finally, a design comparison at the architecture level is performed in Section VI-C.

A. Hardware Architectures for the Inner and Outer Decoder

The state-of-the-art hardware architectures for BCJR algorithm are based on its sub-optimal variant, *max-Log-MAP* (in the following: *MAP*), where the log-MAP is further simplified [10], [11]. These architectures can achieve a high throughput by employing either *spatial* or *functional* parallelism techniques. Spatial parallelism is predominant in the *parallel MAP* (PMAP) [12]–[15] and *fully parallel MAP* (FPMAP) [16] architectures while the functional parallelism, is dominant in the *pipelined MAP* (XMAP) [17], [18] architecture.[3]

[3]Even though the references mentioned in this section were presented in the context of PCCs, they are applicable to the decoding of SCCs [19].

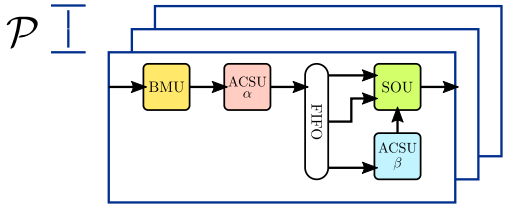


Fig. 10. PMAP decoder architecture schematic.

1) PMAP: In PMAP architectures, the code block with length K bits, is split into $\mathcal{P}$ smaller *sub-blocks* of length $K/\mathcal{P}$ bits (i.e., $K/\mathcal{P} = 1$ for FPMAP). Then, the sub-blocks, i.e., *sub-trellises*, are decoded either by parallel sub-decoder cores (PMAP) or by parallel processing elements (FPMAP). Since the FPMAP is based on a reformulation of the MAP algorithm [16] and is explicitly tailored for PCCs, we do not consider it here and will focus on PMAP. Fig. 10 shows the PMAP architecture, which features $\mathcal{P}$ parallel sub-decoders. Each sub-decoder core is made up of two *add-compare-select units* (ACSUs) to realize forward and backward recursions based on (13) and (14), a *branch metric unit* (BMU) to calculate branch metrics using (12), a *soft-output unit* (SOU) to generate extrinsic information using (17), and *first-in-first-out* (FIFO) buffers to store the backward and branch metrics. For very large sub-blocks, a second BMU can be used to perform a *recomputation* of branch metrics to avoid storing them in the FIFO and the same can be done for forward metrics [12]. In case of $\mathcal{P} = 1$, the architecture is called *serial MAP* (SMAP), where the blocks are processed serially.

In the PMAP architecture, $\mathcal{P}$ sub-decoders work in parallel to decode a block of K bits. In each sub-decoder, the *sliding window* (SW) decoding is employed and the sub-block is further split into smaller portions of size L_{SW} bits.[4] This approach enables a parallel processing of the forward and backward recursions inside each sub-decoder. However, splitting into sub-blocks degrades the decoding performance due to metrics information loss at the initial step of sub-trellises. Thus, the state metrics at the sub-block and sliding window borders need to be estimated. To this end, *Acquisition* (ACQ) technique

[4]In this paper, W refers to the decoding window size in SC-SCC decoders while L_{SW} refers to the sliding window size inside the inner/outer decoders.

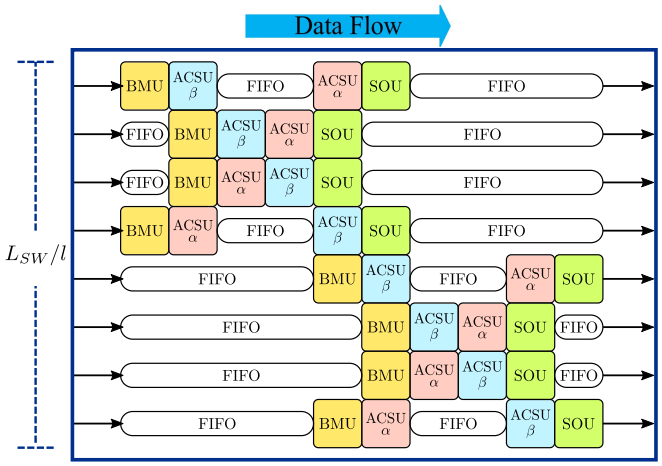


Fig. 11. XMAP decoder pipeline schematic.

[14], [18] can be employed, which carries out a warm-up phase for the state metric computations by doing additional recursion computations of length L_{ACQ}. However, with small sub-block and sliding window sizes, the length of the necessary ACQ calculation is increased, which in turn limits the throughput gain through parallelization [20].

Now, let us move on to evaluate the silicon area and decoding latency of the PMAP architecture. The silicon area occupied by the computational units is

$$A_{\mathrm{PMAP}} = \begin{cases} \mathcal{P} \cdot \left(2 \cdot A^{\mathcal{A}} + A^{\Gamma} + A^{\Lambda}\right) & L_{\mathrm{ACQ}} < L_{\mathrm{SW}} \\ \mathcal{P} \cdot \left(3 \cdot A^{\mathcal{A}} + A^{\Gamma} + A^{\Lambda}\right) & L_{\mathrm{ACQ}} > L_{\mathrm{SW}} \end{cases} \tag{23}$$

where $A^{\mathcal{A}}$, A^{Γ}, and A^{Λ} represent the area of ACSU, BMU, and SOU, respectively. The decoding latency of PMAP decoder, which is made up of $\mathcal{P}$ sub-decoder cores is given by

$$\mathcal{L}^{\mathrm{D}}_{\mathrm{PMAP}} = \frac{K}{\mathcal{P} \cdot l} + \mathcal{L}_{\mathrm{SISO}}, \tag{24}$$

which corresponds to the number of clock cycles needed to decode a code block with information length of K bits. The overall decoding latency is mainly determined by the latency of individual sub-decoder cores to process the sub-blocks of size $K/\mathcal{P}$, which can be expressed as

$$\mathcal{L}_{\mathrm{SISO}} = (L_{\mathrm{SW}} + L_{\mathrm{ACQ}})/l + \mathcal{L}_{\mathrm{P}}\ , \tag{25}$$

where l is the number of trellis steps processed per clock cycle and $\mathcal{L}_{\mathrm{P}}$ is the additional latency due to the pipelined extrinsic computation. The decoding latency, $\mathcal{L}^{\mathrm{D}}_{\mathrm{PMAP}}$, can be improved by employing a higher radix $r = 2^l$ in processing [21].

2) XMAP: The main idea is to split the code trellis into sliding windows and process multiple of them in parallel in a pipeline. To this end, the operations of the MAP algorithm are "unrolled" onto a XMAP decoder pipeline, which is depicted in Fig. 11. In this structure, each row corresponds to the processing of l trellis steps. Delay FIFOs are added to synchronize the pipelines for forward and backward state metric recursion whose characteristic X-shaped overlap gives the architecture its name. Similar to PMAP, the same border initialization method, ACQ, is used for XMAP decoders [17].

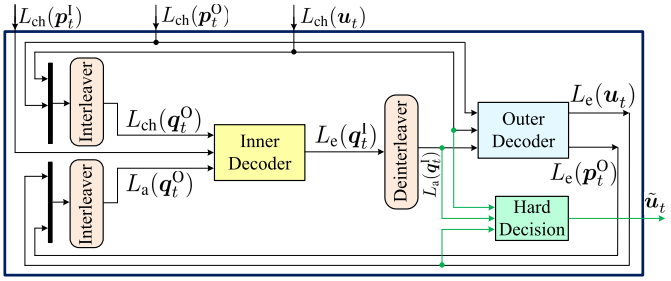


Fig. 12. VLSI architecture of the SCC decoder, corresponding to Algorithm 1.

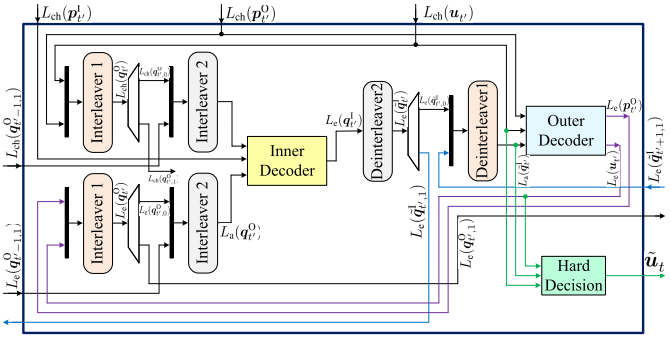


Fig. 13. VLSI architecture to realize the block-wise SC-SCC decoder, following Algorithm 2. Here, t' refers to the blocks at time $t' = t : t+W-1$, which are used to decode the target block, $\boldsymbol{u}_t$, as described in Algorithm 2.

The XMAP decoder is comprised of an acquisition pipeline of length L_{ACQ}/l followed by a decoding pipeline of length L_{SW}/l to realize state metric recursions and extrinsic computation. The acquisition pipeline includes $2L_{\mathrm{ACQ}}$ instances of ACSUs and L_{SW} instances of BMUs, while the decoding pipeline includes $2L_{\mathrm{SW}}$ instances of ACSUs and L_{SW} instances of SOUs. The decoder pipeline is completed by FIFO registers for forwarding the state and branch metrics to the SOUs. Note, that the acquisition pipeline is not shown in Fig. 11.

The total area of the computational units (i.e., BMU, ACSU, SOU) in the XMAP architecture is obtained as

$$A^{\mathrm{XMAP}} = \frac{L_{\mathrm{ACQ}} \cdot 2 \cdot A^{\mathcal{A}}}{l} + \frac{(2 \cdot A^{\mathcal{A}} + A^{\Gamma} + A^{\Lambda}) \cdot L_{\mathrm{SW}}}{l}. \tag{26}$$

The coefficient l in denominators in (26) is because each computational unit processes l trellis steps per clock cycle.

The decoding latency of XMAP decoder, which employs a radix-order $r = 2^l$ to decode a block of K bits is given by

$$\mathcal{L}^{\mathrm{D}}_{\mathrm{XMAP}} = \frac{K}{L_{\mathrm{SW}}} + \mathcal{L}_{\mathrm{Pipe}}\ , \tag{27}$$

where the latency of XMAP pipeline decoder, $\mathcal{L}_{\mathrm{Pipe}}$, is

$$\mathcal{L}_{\mathrm{Pipe}} = (L_{\mathrm{SW}} + L_{\mathrm{ACQ}})/l + \mathcal{L}_{\mathrm{P}}. \tag{28}$$

B. High-Level Decoder Architectures

In this section, we discuss three high-level VLSI architectures to realize the SCC and SC-SCC decoding algorithms. In these designs, only one MAP hardware instance serves as both the inner and outer decoder alternatingly. It can

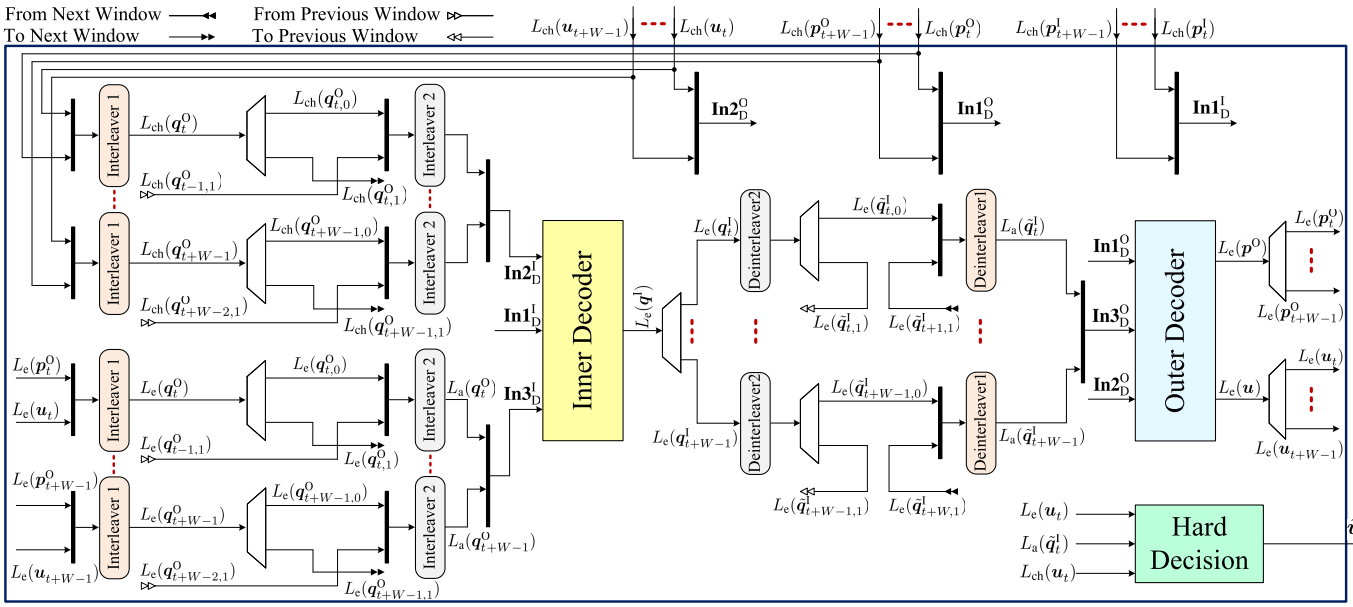


Fig. 14. The VLSI architecture to realize the window-wise decoding approach for the SC-SCC schemes, which is detailed in Algorithm 3.

be realized either by a PMAP, or a XMAP architecture. However, for more clarity with respect to the interleaving and deinterleaving, the inner and outer decoders are drawn separately in Fig. 12–14. Note furthermore, that the individual memories for extrinsic and channel LLRs are not included in Fig. 12–14, since their organization not only depends on the chosen SCC/SC-SCC decoding algorithm, but also depends on the choice of MAP architecture and the chosen parallelism degree.

In this high-level view, the three designs differ in their processing schedule (conforming to either Algorithm 1, 2 or 3) and the trellis lengths treated by the inner and outer decoders.

1) ***Design 1:*** *SCC Decoder Architecture:* Fig. 12 shows the high-level VLSI architecture of *Design 1*, which realizes the uncoupled SCC decoder (i.e., Algorithm 1). This design receives a code block with K_{UC} information bits, which is processed using the inner and outer decoders with the trellis length of $2K_{\mathrm{UC}}$ and K_{UC}, respectively.

2) ***Design 2:*** *Block-Wise SC-SCC Decoder Architecture:* This design employs the block-wise decoding, described in Algorithm 2, to decode the SC-SCCs. The corresponding decoding flow is depicted in Fig. 4. In this scheme, a single component decoder, shown in Fig. 13, is used to decode the whole window in a serial manner.

3) ***Design 3:*** *Window-Wise SC-SCC Decoder Architecture:* The high-level VLSI architecture, shown in Fig. 14, realizes the window-wise SC-SCC decoding which is detailed in Algorithm 3 and its decoding flow is illustrated in Fig. 5. In *Design 3*, the BCJR algorithm runs over the whole window once per iteration, which results in the trellis length of $2K \cdot W$ and $K \cdot W$ for the inner and outer decoders, respectively.

C. Design Comparison

The architectural features of *Design 1–3* are listed in Table III. To have a fair comparison, we consider the same structural latency and computational complexity for all designs. To this end, the block length of *Design 1* is set to $K_{\mathrm{UC}} = W \cdot K$, where W and K are the window size and block length of SC-SCCs. Also, to employ the same I_{eff}, the number of iterations in *Design 1* is set to $I = I_{\mathrm{eff}}$ while in *Design 2* and *Design 3* the number of iterations per window position is $I_{\mathrm{w}} = I_{\mathrm{eff}}/W$.

As a key benefit of SC-SCCs, the decoding performance of *Design 2* and *Design 3* is not limited to the block length. Despite the uncoupled SCC, these designs can achieve the same or even better performance for short block lengths compared to the larger ones, as demonstrated in [5].

In all designs in Table III, it is possible to trade between the decoding performance and throughput by employing the PMAP architecture for the inner and outer decoders. For a given design, the larger inner/outer trellis length results in a better decoding performance at the cost of lower throughput. As specified in Table III, the best decoding performance is achieved using *Design 3* if the trellis length of $2W \cdot K$ and $W \cdot K$ are employed for the inner and outer decoders, respectively. However, the inner and outer decoders of *Design 3* can employ the PMAP with shorter trellis length, named *sub-trellis length* in Table III. It is worth to point out that the sub-trellis length can be an arbitrary value and it is not necessarily equal to the block length. In such case, depending on the degree of parallelism, $\mathcal{P}$, the performance of *Design 3*, the one in the last column of Table III, can be better than that of *Design 2*.

VII. Implementation Estimates and Design Tradeoffs

A. Model Assumptions

The VLSI architectures for SC-SCC decoders extend the design space for decoders of the uncoupled case. Consequently, the effect of design choices like inner/outer decoder architecture, parallelism degree, sub-trellis length, and sliding window length on the figures of merit like core area, latency, and throughput needs to be investigated. Therefore, this paper

TABLE III
DESIGN COMPARISON OF VLSI ARCHITECTURES FOR UNCOUPLED AND COUPLED SCC DECODERS

	Design 1$^{\diamond}$	**Design 2**	**Design 3**	
Decoding Algorithm	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 3
Code Type	SCC	SC-SCC	SC-SCC	SC-SCC
Structural Latency	K_{UC}	$W \cdot K$	$W \cdot K$	$W \cdot K$
Decoding Performance†	Fourth	Third	First	Second
Inner, Outer Trellis Length	$2K_{\mathrm{UC}}, K_{\mathrm{UC}}$	$2K, K$	$2W \cdot K, W \cdot K$	$2W \cdot K, W \cdot K$
Inner, Outer Sub-Trellis Length	$2K_{\mathrm{UC}}/\mathcal{P}, K_{\mathrm{UC}}/\mathcal{P}$	$2K/\mathcal{P}, K/\mathcal{P}$	$2W \cdot K, W \cdot K$	$2W \cdot K/\mathcal{P}, W \cdot K/\mathcal{P}$
Internal Processing	Serial	Serial	Serial	Serial
# Subdecoders per Inner/outer Trellis	$1, \ldots, \mathcal{P}$	$1, \ldots, \mathcal{P}$	1	$1, \ldots, \mathcal{P}$
Decoding Iterations	$I = I_{\mathrm{eff}}$	$I_{\mathrm{w}} = I_{\mathrm{eff}}/W$	$I_{\mathrm{w}} = I_{\mathrm{eff}}/W$	$I_{\mathrm{w}} = I_{\mathrm{eff}}/W$
Inner/Outer Decoder Architecture‡	SMAP, PMAP, XMAP	SMAP, PMAP, XMAP	SMAP, XMAP	PMAP
Main Benefit	High Throughput	Low Area	High Performance	Low Latency
Limited on Block Size $^{\triangleright}$	Yes	No	No	No
Performance Loss for Small Block Size	Yes	No	No	No
Performance Depends on Block Size	Yes	No	No	No

† In this ranking, the same computational complexity and structural latency are considered. $^{\triangleright}$ Possibility of decoding of large block lengths.
$^{\diamond}$ In case of uncoupled codes, the block length $K_{\mathrm{UC}} = W \cdot K$ is considered to have a fair comparison with the SC-SCC, where in this table K is the block length of SC-SCCs (see (9) and (10)). ‡ ACQ technique is employed to improve decoding performance.

TABLE IV
COMPONENT DECODER PARAMETERS FOR AREA AND LATENCY ESTIMATION

$\mathcal{P}$	L_{SW}	L_{ACQ}	$\mathcal{L}_{\mathrm{P}}$	$\mathcal{L}_{\mathrm{I/O}}$	l
$1, 2, 4, 8, 16, 32, 64, 128$	$16, 32, 64, 128$	$0, 8, 16$	4	4	2

TABLE V
PLACE AND ROUTE RESULTS FOR THE COMPUTATIONAL UNITS

	BMU	ACSU	SOU
Area (μm^2) †	572	784	2550
Frequency (MHz)	1000		
V_{dd} (V)	0.72		

† Correspond to the worst case PVT corner of 12 nm technology.

aims at providing guidelines for a down selection of design parameters. We exemplify this with a comparison of three reference designs, i.e., *Design 1–3* in Table III, for three cases of SC-SCC parameters as follows:

- *Case 1:* $K = 1024$, $W = 4$, $I_{\mathrm{w}} = 16$, $m = \mathbf{1}, 3$
- *Case 2:* $K = 512$, $W = 8$, $I_{\mathrm{w}} = 8$, $m = 1, \mathbf{3}, 7$
- *Case 3:* $K = 128$, $W = 32$, $I_{\mathrm{w}} = 2$, $m = 1, 3, 7, \mathbf{15}, 31$

where in each case the coupling memory, m, which according to Section V leads to the best decoding performance is specified in bold. To have a fair comparison, the design parameters are chosen such that *Case 1–3* have the same *structural latency* and *computational complexity*.

The component decoder parameters used in this analysis are listed in Table IV, where the I/O latency ($\mathcal{L}_{\mathrm{I/O}}$) corresponds to the extrinsic-memories delays. It is worth mentioning that, such parameters highly depend on the detailed hardware architecture and their exact values can be specified at the final implementation stage. Since the parallelism in the XMAP architecture comes from the pipelining of the SW decoding, the parallelism degree, $\mathcal{P}$, is fixed to 1 in Table IV while it varies for the PMAP between 1 (i.e., SMAP) and 128.

In case of small K, the parallelism degree of PMAP is reduced to keep the size of sub-blocks ($K/\mathcal{P}$), processed by each sub-decoder, larger than the smallest sliding window size, i.e., $K/\mathcal{P} > 16$. Sub-blocks smaller than 16 bits would lead to a significantly degraded performance due to the lack of accurate state metrics at the sub-block borders [18]. To mitigate this effect in the XMAP architectures, $L_{\mathrm{ACQ}} = 8$ and 16 are considered for small sliding-window/sub-block sizes.

The computational units have been fully synthesized, and then placed and routed in a 12 nm Fin-FET technology for a target clock frequency of 1000 MHz, as detailed in Table V. The quantization for these units is based on a decoder input quantization of 7 bits. The radix-order is fixed to $r = 4$ (i.e., $l = 2$) since it gives the best latency/area tradeoff for max-Log-MAP based decoders [22]. For higher radix-orders, the *Local-SOVA* algorithm can be considered [23]. Since the MAP algorithm is compute dominated, we can give estimations on the silicon area of the decoder architectures on the basis of the total area occupied by the computational units [24]. These estimations do not include the area of the memories, since they are highly dependent on the available memory cuts which would result in a distortion of the comparison. Moreover, it is assumed, that conflict-free interleavers [25] can be found for the SC-SCCs to avoid the memory access conflicts for the high parallelism degrees of component decoders.

Below, in Section VII-B–VII-D, we will evaluate the *silicon area*, *decoding latency*, and *throughput* of *Design 1–3* for the given *Case 1-3* and the component decoder parametrizations in Table IV. These evaluations are based on the area and latency considerations from Section VI-A and on the place and route results in Table V. The goal is to highlight tradeoffs and interplay between code design choices for the proposed SC-SCC schemes and their decoding down to the component decoder level. The corresponding *decoding performance* on the code level were evaluated in Section V.

B. Area and Decoding Latency

We estimate the silicon area and decoding latency for a total of 66 different component decoder configurations (28× XMAP + 38× PMAP) following *Designs 1–3* and considering the code design *Cases 1–3*. The corresponding results for PMAP- and XMAP-based architectures are illustrated in

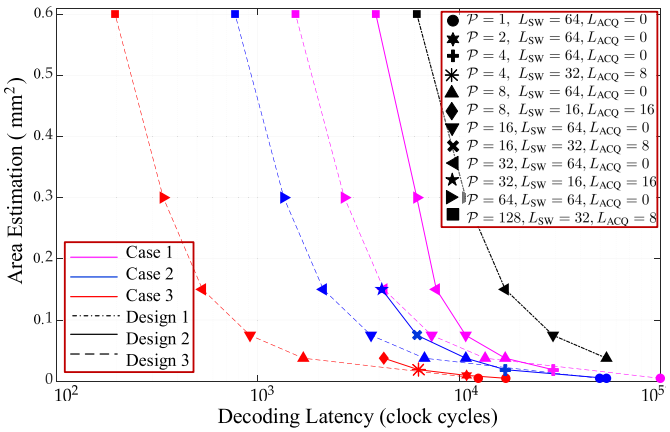


Fig. 15. Area and latency of decoders with SMAP/PMAP component decoder.

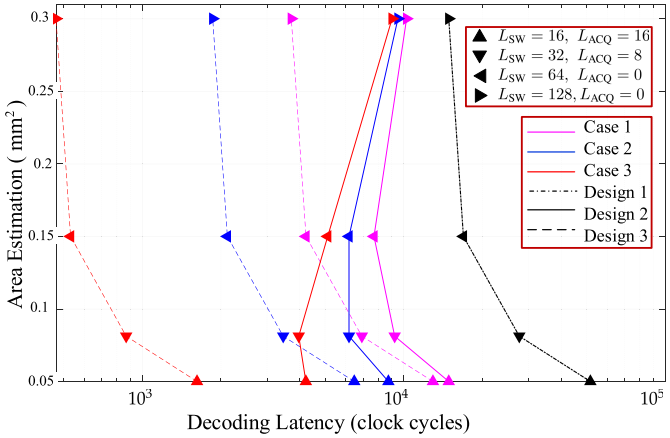


Fig. 16. Area and latency of decoders with a XMAP component decoder.

Fig. 15 and Fig. 16, respectively. The decoding latency can be obtained as

$$\mathcal{L}^{\mathrm{D}} = (\text{Number of Iterations})(\mathcal{L}^{\mathrm{D}}_{\mathrm{Inner}} + \mathcal{L}^{\mathrm{D}}_{\mathrm{Outer}}) + \mathcal{L}_{\mathrm{Ex}}, \tag{29}$$

where $\mathcal{L}^{\mathrm{D}}_{\mathrm{Inner}}$ and $\mathcal{L}^{\mathrm{D}}_{\mathrm{Outer}}$ are the decoding latency of the inner and outer decoders to process the inner and outer trellises. Depending on the inner and outer decoder architectures, $\mathcal{L}^{\mathrm{D}}_{\mathrm{Inner}}$ and $\mathcal{L}^{\mathrm{D}}_{\mathrm{Outer}}$ will be modeled using (24) and (27). In these equations, K is set to the inner/outer trellis length of *Designs 1–3*, which are listed in Table III. Also, the "Number of Iterations" in (29) is equal to I_{eff} for *Design 1* and *Design 2* while it is I_{w} in case of *Design 3*. Moreover, to include the extra latencies, e.g., I/O latency, we consider $\mathcal{L}_{\mathrm{Ex}}$ in (29), which is equal to $W \cdot \mathcal{L}_{\mathrm{I/O}}$ for *Design 2* and $\mathcal{L}_{\mathrm{I/O}}$ for *Design 1* and *Design 3*.

Since a fixed structural latency ($\mathcal{L}^{\mathrm{S}}$) is assumed for the SC-SCC designs, i.e., *Design 2* and *Design 3*, an increase in window size results in a smaller block length (see (9)). Consequently, the designs adapted to the code design *Case 3* exhibit the lowest decoding latency in Fig. 15, since after each pass through the window, K bits will be fully decoded. Another general conclusion from Fig. 15 is that *Design 3* outperforms *Design 2* in terms of decoding latency. The reason for this is two-fold. First, the processing of a large trellis, $W \cdot K$, allows for a higher level of parallelism, $\mathcal{P}$. For example, for *Design 2* and *Case 3*, a parallelization of $\mathcal{P} = 8$ already results in a reduced sub-block size of $K/\mathcal{P} = 128/8 = 16$, for which an acquisition needs to be employed. Second, *Design 3* allows a parallelization of up to $\mathcal{P} = 128$ without reducing the sub-block size below $W \cdot K/\mathcal{P} = 4096/64 = 64$. Note, however, that the parallel decoding of the W sub-trellises within a window, will degrade the performance in comparison to the serial decoding of the individual blocks of size K as is done in *Design 2*. Having considered a certain design in Fig. 15, the decoding latency can be reduced by increasing the level of parallelism for all the code design cases (i.e., *Cases 1–3*) at the expense of larger silicon area.

For the decoders, which employ XMAP as the inner/outer decoder architecture, we obtain similar results, as illustrated in Fig. 16. Here, the decoding latency is smaller for *Design 3* when comparing it to *Design 1* and *Design 2*. In contrast to the PMAP case, increasing the parallelism on component decoder level results in a considerable latency-penalty for *Design 2*, since for large L_{SW}, the number of sliding windows becomes smaller in comparison with the XMAP pipeline length. Therefore, the decoder pipeline cannot be fully utilized and the pipeline latency can no longer be largely hidden.

It is worth to mentioning that in Fig. 15 and Fig. 16, the reference *Design 1*, shown by a dashed black line, has a larger decoding latency than *Design 2* and *Design 3* for all the code design *Cases 1–3*. This is because *Design 1* will output the decoded bits after $W \cdot I_{\mathrm{w}}$ iterations (i.e., 64 iterations in this example), whereas a decoder following either *Design 2* or *Design 3* will output the decoded bits after I_{w} iterations.

C. Area and Throughput

The decoder throughput in terms of decoded bits per second for the presented designs can be obtained from the decoding latencies in clock cycles for a given clock frequency f as

$$\mathcal{T} = \frac{K}{\mathcal{L}^{\mathrm{D}}} \cdot f, \tag{30}$$

where the value of K is determined in a similar way as explained for (29). We consider a clock frequency $f = 1000$ MHz as it was used for the synthesis of the computational units. The resulting throughput estimates are plotted against the area consumption of the *Designs 1–3*, in Fig. 17 and Fig. 18.

In case of *Design 2*, the serial processing of the blocks in a window leads to a reduced throughput compared to *Design 1* and *Design 3*, which process the decoding window in parallel. Notably, in Fig. 17, the throughput difference between the PMAP-based decoders with *Design 2* is only a few Mb/s for various code design cases when comparing at similar levels of parallelism. The throughput estimations for the XMAP-based decoders are illustrated in Fig. 18. The above-mentioned latency penalty is translated into a reduced throughput for *Design 2* with large L_{SW}, which is more pronounced for the code design *Case 3*. Moreover, increasing W from *Case 1* to *Case 3*, yields a throughput penalty for *Design 2* decoders in the order of up to $50 - 60$ Mb/s.

It is worthwhile to mention that a higher throughput can be achieved by employing an unrolled XMAP (UXMAP)

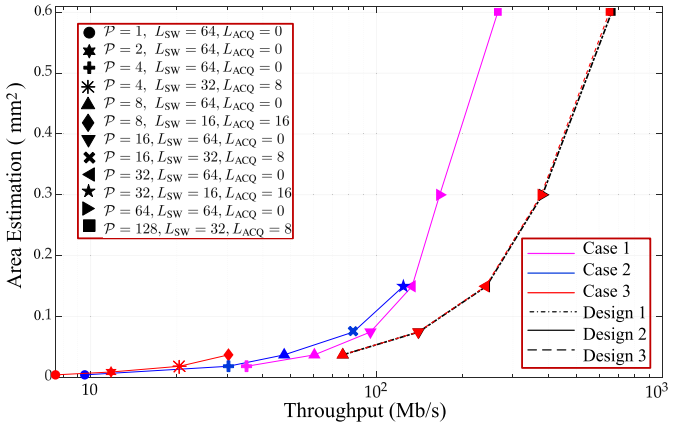


Fig. 17. Area and throughput of decoders with a PMAP component decoder.

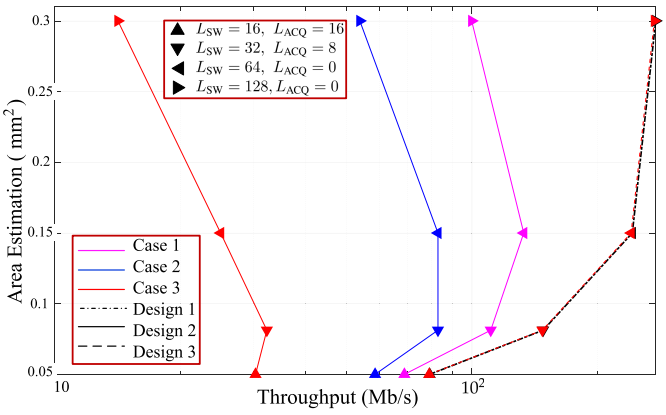


Fig. 18. Area and throughput of decoders with a XMAP component decoder.

architecture. In [26], we have proposed a modified window decoding algorithm along with a UXMAP architecture for SC-SCCs, which achieves a throughput in the range of 100s Gbps.

D. Design Tradeoffs

From the silicon area, decoding latency, and throughput models, several design tradeoffs can be identified.

First, in case of a XMAP component decoder architecture, the window size, W, must be jointly chosen with the sliding window size, L_{SW}, within the component decoder. Also, the number of sliding windows, $N_{SW} = K/L_{SW}$, should be larger than the pipeline length of the XMAP to avoid a low pipeline utilization and latency/throughput degradation. To mitigate this, the combination of spatial and functional parallelism, i.e., multiple parallel XMAP cores, can be used as in [10].

Second, for the PMAP component decoders and *Design 2*, increasing the window size, W, does not significantly impact the throughput. Therefore, the tradeoff can be based on error correcting performance considerations. Seen in connection with the lower latency for the code design cases with higher W, this result emphasizes the viability of SC-SCCs for streaming applications with moderate throughput requirements but high demands on decoding performance (see also Section V).

Overall, the choice between *Design 2* and *Design 3* becomes a tradeoff between decoding performance and throughput, since the latency for a fixed code design case and component decoder architecture is similar.

Finally, it should also be mentioned that the throughput for decoders with *Design 1* illustrated in Fig. 17 and 18 does not account for early stopping. Since we chosen to fix the complexity to have a fair comparison, the number of decoding iterations could be reduced for all decoders with *Design 1* by a factor of up to W, thereby increasing the throughput by the same factor. However, as Section V explained, the decoding performance of the SC-SCC schemes cannot be matched by the uncoupled schemes with the same complexity.

VIII. Conclusion

We proposed two decoding algorithms for the SC-SCCs along with corresponding VLSI architectures. Also, we investigated the effect of block length, coupling memory, window size, and number of iterations on the complexity, performance, throughput, decoding latency, and silicon area of the proposed decoding schemes in a 12 nm FinFET technology.

References

[1] T.-K. Le, U. Salim, and F. Kaltenberger, "An overview of physical layer design for ultra-reliable low-latency communications in 3GPP releases 15, 16, and 17," *IEEE Access*, vol. 9, pp. 433–444, 2021.

[2] S. Moloudi, M. Lentmaier, and A. Graell, "Spatially coupled turbo-like codes: A new trade-off between waterfall and error floor," *IEEE Trans. Commun.*, vol. 67, no. 5, pp. 3114–3123, May 2019.

[3] M. Herrmann, N. Wehn, M. Thalmaier, M. Fehrenz, T. Lehnigk-Emden, and M. Alles, "A 336 Gbit/s full-parallel window decoder for spatially coupled LDPC codes," in *Proc. Joint Eur. Conf. Netw. Commun. 6G Summit (EuCNC/6G Summit)*, Jun. 2021, pp. 508–513.

[4] X. Wu, M. Qiu, and J. Yuan, "Partially information coupled bit-interleaved polar coded modulation," *IEEE Trans. Commun.*, vol. 69, no. 10, pp. 6409–6423, Oct. 2021.

[5] M. Mahdavi, M. Umar Farooq, L. Liu, O. Edfors, V. Owall, and M. Lentmaier, "The effect of coupling memory and block length on spatially coupled serially concatenated codes," in *Proc. IEEE 93rd Veh. Technol. Conf. (VTC-Spring)*, Apr. 2021, pp. 1–7.

[6] M. U. Farooq, A. G. I. Amat, and M. Lentmaier, "Spatially-coupled serially concatenated codes with periodic convolutional permutors," in *Proc. 11th Int. Symp. Topics Coding (ISTC)*, Aug. 2021, pp. 1–5.

[7] G. Liva and F. Steiner. (Jan. 2022). *Pretty-Good-Codes: Online Library of Good Channel Codes*. [Online]. Available: http://pretty-good-codes.org

[8] A. Balatsoukas-Stimming, P. Giard, and A. Burg, "Comparison of polar decoders with existing low-density parity-check and turbo decoders," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, Mar. 2017, pp. 1–6.

[9] D. Hui, S. Sandberg, Y. Blankenship, M. Andersson, and L. Grosjean, "Channel coding in 5G new radio: A tutorial overview and performance comparison with 4G LTE," *IEEE Veh. Technol. Mag.*, vol. 13, no. 4, pp. 60–69, Dec. 2018.

[10] S. Weithoffer, O. Griebel, R. Klaimi, C. A. Nour, and N. Wehn, "Advanced hardware architectures for turbo code decoding beyond 100 Gb/s," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, May 2020, pp. 1–6.

[11] S. Weithoffer, R. Klaimi, C. A. Nour, N. Wehn, and C. Douillard, "Fully pipelined iteration unrolled decoders the road to TB/S turbo decoding," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2020, pp. 5115–5119.

[12] T. Ilnseher, F. Kienle, C. Weis, and N. Wehn, "A 2.15 GBit/s turbo code decoder for LTE advanced base station applications," in *Proc. 7th Int. Symp. Turbo Codes Iterative Inf. Process. (ISTC)*, Aug. 2012, pp. 21–25.

[13] R. Shrestha and R. P. Paily, "High-throughput turbo decoder with parallel architecture for LTE wireless communication standards," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 9, pp. 2699–2710, Sep. 2014.

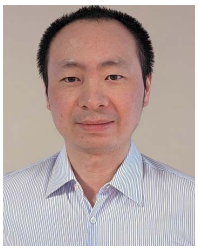
[14] C. Roth, S. Belfanti, C. Benkeser, and Q. Huang, "Efficient parallel turbo-decoding for high-throughput wireless systems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 6, pp. 1824–1835, Jun. 2014.

[15] C.-H. Lin and C.-W. Hsieh, "Low-routing-complexity convolutional/turbo decoder design for iterative detection and decoding receivers," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 11, pp. 4476–4489, Nov. 2019.
[16] A. Li, L. Xiang, T. Chen, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "VLSI implementation of fully parallel LTE turbo decoders," *IEEE Access*, vol. 4, pp. 323–346, 2016.
[17] S. Weithoffer, F. Pohl, and N. Wehn, "On the applicability of trellis compression to turbo-code decoder hardware architectures," in *Proc. 9th Int. Symp. Turbo Codes Iterative Inf. Process. (ISTC)*, Sep. 2016, pp. 61–65.
[18] M. May, T. Ilnseher, N. Wehn, and W. Raab, "A 150 Mbit/s 3GPP LTE turbo code decoder," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2010, pp. 1420–1425.
[19] E. Boutillon, C. Douillard, and G. Montorsi, "Iterative decoding of concatenated convolutional codes: Implementation issues," *Proc. IEEE*, vol. 95, no. 6, pp. 1201–1227, Jun. 2007.
[20] S. Weithoffer, K. Kraft, and N. Wehn, "Bit-level pipelining for highly parallel turbo-code decoders: A critical assessment," in *Proc. AFRICON*, 2017, pp. 121–126.
[21] G. Fettweis and H. Meyr, "Parallel Viterbi algorithm implementation: Breaking the ACS-bottleneck," *IEEE Trans. Commun.*, vol. 37, no. 8, pp. 785–790, Aug. 1989.
[22] S. Weithoffer, C. A. Nour, N. Wehn, C. Douillard, and C. Berrou, "25 years of turbo codes: From Mb/s to beyond 100 Gb/s," in *Proc. IEEE 10th Int. Symp. Turbo Codes Iterative Inf. Process. (ISTC)*, Dec. 2018, pp. 1–6.
[23] S. Weithoffer, R. Klaimi, C. A. Nour, N. Wehn, and C. Douillard, "Low-complexity computational units for the local-SOVA decoding algorithm," in *Proc. IEEE 31st Annu. Int. Symp. Pers., Indoor Mobile Radio Commun.*, Aug. 2020, pp. 1–6.
[24] C. Kestel, M. Herrmann, and N. When, "When channel coding hits the implementation wall," in *Proc. IEEE 10th Int. Symp. Turbo Codes Iterative Inf. Process. (ISTC)*, Dec. 2018, pp. 1–6.
[25] G. Wang, H. Shen, Y. Sun, J. R. Cavallaro, A. Vosoughi, and Y. Guo, "Parallel interleaver design for a high throughput HSPA+/LTE multi-standard turbo decoder," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 5, pp. 1376–1389, May 2014.
[26] M. Mahdavi, L. Liu, O. Edfors, M. Lentmaier, N. Wehn, and S. Weithoffer, "Towards fully pipelined decoding of spatially coupled serially concatenated codes," in *Proc. 11th Int. Symp. Topics Coding (ISTC)*, Aug. 2021, pp. 1–5.

Mojtaba Mahdavi (Member, IEEE) received the M.Sc. degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2010, and the Ph.D. degree from the Department of Electrical and Information Technology (EIT), Lund University, Sweden, in 2021. The title of his Ph.D. thesis is "Baseband Processing for 5G and Beyond: Algorithms, VLSI Architectures, and Co-design." From 2010 to 2013, he was with the Advanced Integrated Circuit Design Laboratory, Sharif University of Technology. In 2020, he was a Visiting Researcher at the University of Kaiserslautern, Germany. In 2021, he joined the Device Platform Research Group, Ericsson Research, Lund, Sweden.

Stefan Weithoffer (Member, IEEE) received the Ph.D. degree from the University of Kaiserslautern, Germany, in 2018, where he worked with his doctoral advisor Prof. Norbert Wehn on implementation issues of high-throughput turbo decoding. He has designed several turbo decoder IP in the course of collaborations with academia as well as industry which have been successfully integrated and manufactured in systems-on-chip (SoC) on 28 nm. Since September 2019, he has been an Associate Professor with the MEE Department, IMT Atlantique.

Matthias Herrmann (Member, IEEE) received the Diploma degree in communication engineering from the University of Kaiserslautern, Germany, where he is currently pursuing the Ph.D. degree in microelectronic system design with the Department of Electrical Engineering and Information Technology. His special research interest includes the design and implementation of high-throughput baseband signal processing components.



Liang Liu (Member, IEEE) received the B.S. degree from the Department of Electronics Engineering, Fudan University, China, in 2005, and the Ph.D. degree from the Department of Microelectronics, Fudan University, in 2010. In 2010, he was a Visiting Researcher with Rensselaer Polytechnic Institute, USA. He joined the Department of Electrical and Information Technology (EIT), Lund University, Sweden, where he held a post-doctoral position in 2010. Since 2016, he has been an Associate Professor with Lund University. His current research interests include wireless systems and digital integrated circuits design.

Ove Edfors (Senior Member, IEEE) received the Ph.D. degree in signal processing from the Luleå University of Technology, Luleå, Sweden, in 1996. He is currently a Full Professor of radio systems with the Department of Electrical and Information Technology (EIT), Lund University, Sweden. His research interests include statistical signal processing and low-complexity algorithms with applications in wireless communications. He has long experience with OFDM and MIMO systems and his current research focus is on how realistic propagation characteristics and hardware limitations influence system performance and baseband processing complexity.

Norbert Wehn (Senior Member, IEEE) holds the Chair for microelectronic system design with the Department of Electrical Engineering and Information Technology, University of Kaiserslautern, Germany. He has more than 400 publications in various fields of microelectronic system design and holds 21 patents. His special research interests include VLSI-architectures for mobile communication, forward error correction techniques, low-power techniques, advanced SoC, memory architectures, 3D integration, reliability issues in SoC, the IoT, and hardware accelerators for machine learning.

Michael Lentmaier (Senior Member, IEEE) is currently an Associate Professor with the Department of Electrical and Information Technology, Lund University. His research interests include the design and analysis of coding systems, graph-based iterative algorithms, and Bayesian methods applied to decoding, detection, and estimation in communication systems. He served as an Editor for IEEE COMMUNICATIONS LETTERS (2010–2013), IEEE TRANSACTIONS ON COMMUNICATIONS (2014–2017), and IEEE TRANSACTIONS ON INFORMATION THEORY (2017–2020). He was awarded the Communications Society & Information Theory Society Joint Paper Award (2012) for his paper "Iterative decoding threshold analysis for LDPC convolutional codes."