# TC: Throughput Centric Successive Cancellation Decoder Hardware Implementation for Polar Codes

Tiben Che, Jingwei Xu and Gwan Choi
Department of Electrical and Computer Engineering
Texas A&M University, College Station, Texas 77840
Email: {ctb47321, xujw07, gchoi}@tamu.edu

***Abstract*—This paper presents a hardware architecture of fast simplified successive cancellation (fast-SSC) algorithm for polar codes, which significantly reduces the decoding latency and dramatically increases the throughput. Algorithmically, fast-SSC algorithm suffers from the fact that its decoder scheduling and the consequent architecture depends on the code rate; this is a challenge for rate-compatible system. However, by exploiting the homogeneousness between the decoding processes of fast constituent polar codes and regular polar codes, the presented design is compatible with any rate. The scheduling plan and the intendedly designed processing core are also described. Results show that, compared with the state-of-art decoder, our design can achieve at least $60\%$ latency reduction for the codes with length $N = 1024$. By using $Nangate\ FreePDK\ 45nm$ process, proposed design can reach throughput up to $5.81\ Gbps$ and $2.01\ Gbps$ for $(1024, 870)$ and $(1024, 512)$ polar code, respectively.**

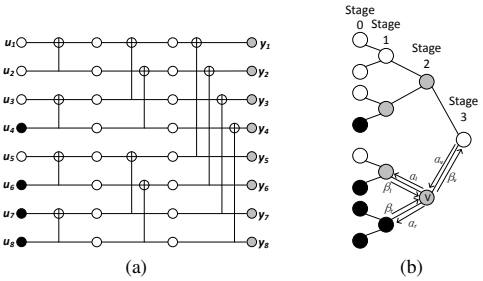



Fig. 1. (a) Encoder of $(8, 4)$ polar code, (b) Tree presentation of $(8, 4)$ SC decoder

## I. Introduction

Recently, polar codes [1] have received significant attention due to its capability to achieve the capacity of binary-input memoryless symmetric channels with low-complexity encoding and decoding schemes. Successive cancellation (SC) [1], list successive cancellation (List-SC) [2] and belief propagation (BP) [3] are the three most common proposed decoding schemes. Among these, SC decoder is the most promising for practical hardware implementation since its low $O(NlogN)$ complexity, where $N$ is the length of the code. Thus, many relevant hardware designs are proposed [4] [5] [6].

However, algorithmically, SC decoder suffers from high latency. Typically, for conventional SC decoder, its latency $(2N - 2)$ increases linearly with respect to the code length. This is a significant challenge since polar codes work well only at very long code lengths. A lot of works have been done to reduce the latency of SC decoder from both hardware and algorithm aspects. In [7], a pre-computation method is used to reduce decoding latency from $2N - 2$ to $N - 1$. In [8], three approaches, the dedicated 2-bit decoder for the last stage of SC decoding, overlapped-scheduling and look-ahead techniques are applied, which eventually results in a $3N/4-1$ latency. In [9] and [10], by observing the tree architecture of SC decoding, certain patterns of constituent codes are found. These constituent codes can feed back the hard decision information immediately without traversal, which can significantly reduce the latency of decoding some polar codes with a given architecture. This approach is refer to as fast-SSC decoder. Moreover, a processors-array based structure for FPGA implementation is also proposed in [10].

In this paper, a novel low latency hardware architecture of polar code decoding using fast-SSC algorithm is presented. Although fast-SSC algorithm naturally lacks flexibility for multiple rates, the proposed design overcomes this disadvantage by utilizing the similarity between the decoding processes of fast constituent polar codes and regular polar codes. Corresponding scheduling plan is presented in this paper. We also provide the design details of the $processing\ unit$ (PU) which is compatible with both regular polar code and constituent polar code. The comparisons with other commonly discussed SC decoders are given. For example, Compared with the 2b-SC-Precomputation decoder, the fastest ASIC design of SC decoder to best of our knowledge, the proposed design can achieve at least $60\%$ latency reduction for polar code with length $N = 1024$. The analysis of latency reduction with respect to code rates is also presented. It shows proposed architecture can yield a significant latency reduction especially at high code rate (code rate $> 0.8$). This is very promising for modern communication or data storage systems where high rate codes are desired. Synthesis results using $Nangate\ FreePDK\ 45nm$ process shows the proposed design can reach throughput of up to $5.81\ Gbps$ and $2.01\ Gbps$ for $(1024, 870)$ and $(1024, 512)$ polar codes, respectively.

This paper is organized as follows. The relative background are reviewed in section II. Then, the hardware implementation of proposed system is described in section III. After that, the synthesis results and relevant comparisons are discussed in section IV. Finally, the conclusion is in section V.

## II. Background

### A. Polar Code and Tree analysis of SC Decoding

As described in [1], a polar code is constructed by exploiting channel polarization. Mathematically, polar codes are linear block codes of length $N = 2^n$. The transmitted codeword $\boldsymbol{x} \triangleq (x_1, x_2, \cdots, x_N)$ is computed by $\boldsymbol{x} = \boldsymbol{uG}$ where $\boldsymbol{G} = \boldsymbol{F}^{\otimes \boldsymbol{m}}$, and $\boldsymbol{F}^{\otimes \boldsymbol{m}}$ is the $m$-th Kronecker power of $\boldsymbol{F} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$. Each row of $G$ is corresponding to an equivalent polarizing channel. For an $(N, k)$ polar code, $k$ bits that carry source information in $\boldsymbol{u}$ are transmitted using the most reliable $k$ channels. These are refer to information bits. While the rest $N - k$ bits, called frozen bits, are set to zeros and are placed at the least reliable channels. Determining the location of the information and frozen bits depends on the channel model and the channel quality is investigated in [11]. Fig. 1a shows an example of $(8, 4)$ polar code encoder, where the black and white nodes stand for the information bits and frozen bits, respectively.

Polar codes can be decoded by recursively applying successive cancellation to estimate $\hat{u}_i$ using the channel output

$y_0^{N-1}$ and the previously estimated bits $\hat{u}_0^{i-1}$. This approach is naturally represented by a binary tree whose each node corresponds to a constituent code. The number of bits in one constituent node in stage $m(m = 0, 1, 2...)$ $N^m$ is equal to $2^m$. Fig. 1b shows an example of $(8, 4)$ polar code. $\boldsymbol{\alpha}$ stands for the soft reliability value, typically is log-likelihood ratio (LLR), and $\boldsymbol{\beta}$ stands for the hard decision. $\boldsymbol{\alpha_l}$ and $\boldsymbol{\alpha_r}$ are the message passing from parent node to left and right child, and can be computed according to Eq. (1) and Eq. (2), respectively.

$$\begin{aligned}\alpha_l[i] &= f(\alpha_v[i], \alpha_v[i + N^m/2]) \\ &= sign(\alpha_v[i])sign(\alpha_v[i + N^m/2]) \\ &\cdot\ min(|\alpha_v[i]|, |\alpha_v[i + N^m/2]|)\end{aligned} \tag{1}$$

$$\begin{aligned}\alpha_r[i] &= g(\beta_l[i - N^m/2], \alpha_v[i], \alpha_v[i - N^m/2]) \\ &= (-1)^{\beta_l[i-N^m/2]} \cdot \alpha_v[i - N^m/2] + \alpha_v[i]\end{aligned} \tag{2}$$

At stage 0, $\beta_v$ of a frozen node is always zero, and for information bit its value is calculated by threshold detection of the soft reliability according to

$$\beta_v = h(\alpha_v) = \begin{cases} 0, \ if\ \alpha_v \geqslant 0 \\ 1, \ otherwise \end{cases} \tag{3}$$

At intermediate stages, $\boldsymbol{\beta_v}$ can be recursively calculated by

$$\beta_v[i] = \begin{cases} \beta_l[i] \oplus \beta_r[i] \ if\ i \leq\ N^m/2 \\ \beta_r[i - N^m/2] \ otherwise \end{cases} \tag{4}$$

### B. Fast-SSC Algorithm

The main idea of fast-SSC algorithm is illustrated in [7], [9] and [10]. By identifying certain pattern of constituent polar codes, the hard decision $\boldsymbol{\beta_v}$ of each constituent node can be determined immediately, without traversing the entire subtree, once the constituent polar code is activated. Such arrangement significantly reduces the decoding latency. For a length $N$ constituent code in non-systematic polar codes, $\hat{\boldsymbol{u}}_N$ is calculated by $\hat{\boldsymbol{u}}_N = \boldsymbol{\beta_{vN}} \cdot G_N$, where $G_N$ is the generator matrix for length $N$ polar code. We adopt four types of constituent polar codes in our design. These are $\mathcal{N}^0$, $\mathcal{N}^1$, $\mathcal{N}^{SPC}$ and $\mathcal{N}^{REP}$, which are called fast constituent polar codes.

$\mathcal{N}^0$ and $\mathcal{N}^1$ are refer to those constituent codes which only contain frozen bits or information bits, respectively. For $\mathcal{N}^0$ codes, we can set $\boldsymbol{\beta_v}$ to 0 immediately. For $\mathcal{N}^1$ node, $\boldsymbol{\beta_v}$ can be directly determined via threshold detection Eq. (3). $\mathcal{N}^{SPC}$ and $\mathcal{N}^{REP}$ are two kinds constituent codes containing both frozen bits and information bits. In a length $N$ $\mathcal{N}^{SPC}$ codes, only the first bit is frozen. It renders the constituent codes as a rate $(N-1)/N$ single parity check (SPC) code. This code can be decoded by performing parity check with the least reliable bit which has the minimum absolute value of LLR. First, get the hard decision $HD_v$ of $\boldsymbol{\beta_v}$ via threshold detection. Then, calculated the parity by

$$parity = \sum_{i=1}^{N^m} \oplus HD_v[i]. \tag{5}$$

and, find the index of the least reliable bit via

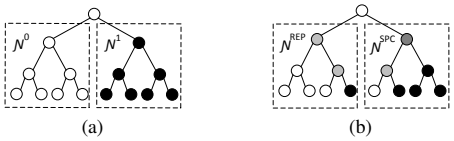
$$j = arg\min_i |\alpha_v[i]|. \tag{6}$$

Eventually, $\boldsymbol{\beta_v}$ is decided by

$$\beta_v[i] = \begin{cases} HD_v[i] \oplus parity, \ when\ i = j \\ HD_v[i], \ otherwise \end{cases} \tag{7}$$

In a length $N$ $\mathcal{N}^{SPC}$ codes, only the last bit is information bit. In this case, all the $\beta_v[i]$ should be the same and are reflections of the information contained in the only one information bit. Thus, the decoding algorithm starts by summing all input LLRs and $\boldsymbol{\beta_v}$ is calculated as

$$\beta_v[i] = \begin{cases} 0, \ when\ \sum \alpha_v[i] \geqslant 0; \\ 1, \ otherwise \end{cases} \tag{8}$$



(a) (b)

Fig. 2. (a) An example of $\mathcal{N}^0$ and $\mathcal{N}^1$ in a 8-bit polar code tree, and (b) An example of $\mathcal{N}^{SPC}$ and $\mathcal{N}^{REP}$ in a 8-bit polar code tree

Fig. 2 gives the examples of tree presentations of these four kinds constituent polar codes.

## III. Hardware Implementation

In this section, a novel hardware implementation of fast-SSC decoder is presented. For a polar code with a given length, different code rate yields different distribution of constituent polar codes. A thoughtfully-composed architecture should have the capability and flexibility to deal with different rates. Thus, by exploiting the homogeneousness between the decoding processes of fast constituent polar codes and regular polar codes, our design supports a variety of rates. The scheduling scheme based on the proposed architecture is also discussed. Additionally, we develop an approach for sharing and reusing computational elements to achieve higher hardware efficiency.

### A. System Overview

As introduced in [5], tree architecture or line architecture for SC decoder is the most common. Line architecture has a higher hardware utilization but needs increased complexity in control module and memory access. Thus, we adopt tree architecture in our design. Fig. 3 shows an overview of proposed system when code length = 16. $Processing\ unit$ (PU) performs the $f$ and $g$ functions in Eq. (1) and Eq. (2), respectively, and its arithmetic part is used to decode $\mathcal{N}^{SPC}$ and $\mathcal{N}^{REP}$ as well. Pre-computation technique is also used, which allows the $f$ and $g$ functions update in the same clock cycle. The PU used in stage 0 has a slight difference with ordinary PU. We denote it with $PU_0$ in the figure. According to Eq. (6), the minimum LLR value needs to be found. The comparator tree is used to perform this since it inherently exists in the tree architecture of PUs. A judicious scheduling permits obtaining the minimum value at $stage\ 0$ and recording the choice of smaller input for each PU at each stage. After that, a backward operation implemented by a series of $parity\ transmit\ unit$ (PTU) can help to locate the minimum one among the length $N$ $\mathcal{N}^{SPC}$ constituent polar codes. Design details are illustrated in section III-C. The estimation of current bit in SC decoding is bases on the information of previous decoded bits ($\boldsymbol{\beta}$). This information is also called partial sum. Thus, a $partial\ sum\ generator$ (PSG) which can co-operate with decoding pipeline is also needed. We adopt the PSG introduced in [12] in our design, and it is compatible with our system. Thus, the design of PSG is not discussed in this paper.

### B. Dataflow, latency and flexibility analysis

In terms of tree presentation, SC decoder conventionally process one node in each clock cycle. Traversal of a subtree contained $N$ leaf nodes needs $2N - 2$ clock cycles. By using

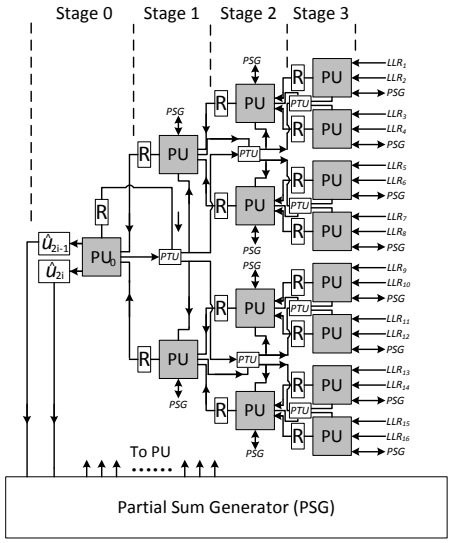



Fig. 3. Overview of proposed system when code length = 16

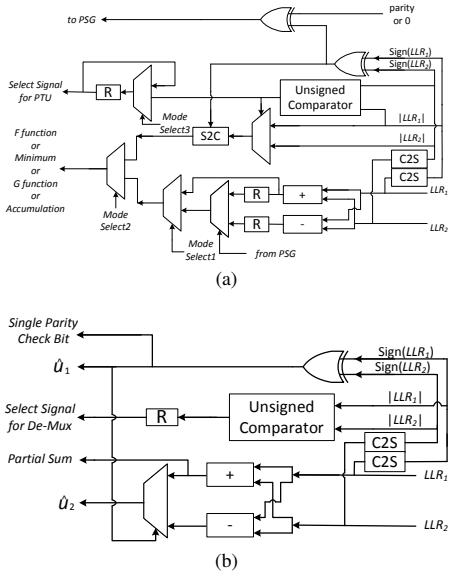



Fig. 4. (a) Design details of PU, (b) Design details of $PU_0$

pre-computation as introduced in [7], which calculate the $f$ function and all the possible result of $g$ functions in the same clock cycle, the latency can be reduced to $N-1$. In our design, if this subtree is belong to fast constituent polar codes, the latency can be further reduced.

For $\mathcal{N}^0$, the $\boldsymbol{\beta_v}$ are all set to 0, and for $\mathcal{N}^1$, the $\boldsymbol{\beta_v}$ are determined by hard decision of input LLRs. Both of the two computations need only one clock cycle after they are activated. For $\mathcal{N}^{SPC}$, according to Eq. (5), Eq. (6), and Eq. (7), only three operations needed. Finding the minimum LLR can be done by a comparator tree, which is naturally existed in SC decoder with tree architecture since every PU has a comparator for Eq. (1). For $N$ LLRs, finding the smallest one use $Log_2N$ clock cycles. Meanwhile, we can obtain the parity bit when the minimum LLR is found, which will be explained in the next subsection. After that, one more clock cycle is need for signal parity check which is done by a $XOR$ gate. Thus, totally, decoding a length $N$ $\mathcal{N}^{SPC}$ constituent polar codes need $Log_2N+1$ clock cycles. For $\mathcal{N}^{REP}$, according to Eq. (8), an accumulation operation is needed. Similar to the comparator tree, an adder tree also exists in SC decoder within the tree architecture since every PU has an adder for Eq. (2). For a length $N$ $\mathcal{N}^{REP}$ constituent polar code, it needs $Log_2N$ clock cycles to decode.

$\mathcal{N}^0$ and $\mathcal{N}^1$ have time complexity $O(1)$ and $\mathcal{N}^{SPC}$ and $\mathcal{N}^{REP}$ have time complexity $O(log_2N)$. Compared with commonly discussed SC architecture in [5], [7] and [8], which all have linear time complexity $O(N)$, we can benefit significantly from proposed scheduling scheme in term of latency, especially with very large $N$. The latency reduction of $N = 1024$ polar code with different rate will be presented in the next section.

The main challenge for fast-SSC decoder is that the architecture subject to the rate of codes. This is due to the reason that polar codes with different rates do not have the uniform distribution of constituent polar codes. Proposed design overcomes this obstacle by exploring the similarity between the decoding architecture of fast constituent and regular polar codes. The specific designed PU allows the tree architecture to deal with both fast constituent and regular polar codes, which means the entire decoding processing can run smoothly no matter what the distributions of constituent codes are. This architecture is independent and does not relay on the distribution of constituent codes. This property provides the flexibility for multiple rates. To switch from one rate to another rate, only the control signals for given PUs need to be modified.

### C. Processing Unit Design

Fig. 4a shows design details of PU. A single PU can perform $f$ and $g$ functions in Eq. (1) and Eq. (2), respectively. Also a PU tree can help to find the minimum values or do accumulation for multiple inputs. In Fig. 4a, $S$ stands for $signed\ magnitude\ number$ and $C$ stands for $2's\ complement\ number$. Unlike the PU design in [8], in which data are initially stored as signed magnitude form, our design use 2's complement as initial form. We do this for two reasons. 1). According to synthesis result, the critical path of PU is along with the $g$ function path. By moving number system convert modules to the $f$ function path, which means using 2's complement as initial data form, the critical path is still along with $g$ function path, but with significant reduction. 2). Compared with four number system convert modules are used in [8], only three are used if use 2's complement number. This is more hardware efficient. The benefits of this modification can be seen in section IV.

For each PU, two LLRs are fed simultaneously. Since we use the pre-computation technique, $f$ and $g$ functions are calculated at the same time, and which one needs to be output is determined by $mode\ select$ 2. According to Eq. (2), there are only two types of possible results for $g$ function, sum or difference. Its final result depends on the corresponding partial sum. So two registers are used here to hold the most recently computed values until the corresponding partial sum is calculated. When it calculates the sum for decoding $\mathcal{N}^{REP}$, only additions are needed. The datapath is decided by $Mode\ select$ 1 signal. When $f$ function is performed, according to Eq. (1), both 2 inputs are divided into two parts: sign bit and unsigned number. Each part is processed separately first, and then results of two parts are combined together to obtain the updated value. $C\ to\ S$ and $S\ to\ C$ modules are needed before and after comparisons, respectively. When it deals with $\mathcal{N}^{SPC}$, the result of comparison should be recorded using a register as the $select\ signal$ for PTU. Since the processing of searching minimum value lasts several clock cycles, there should be a feedback of the register to

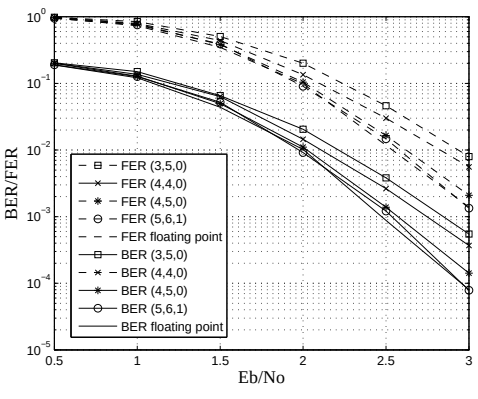



Fig. 5. Effect of quantization on the BER/FER performance of $(1024, 512)$ code

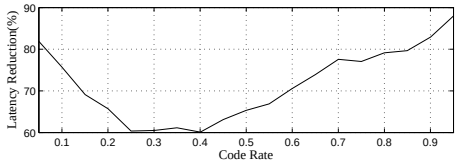



Fig. 6. Latency Reduction $vs$. Code Rate

hold this value for the later clock cycles. The input source is chosen by $Mode\ select$ 3 signal. Since every PU does $exclusive\text{-}or$ operation to the sign bit of two inputs, according to Eq. (5), the sign bit of the final value in stage 0 should be equal to the parity. Eq. (7) can be performed using an $XOR$ gate. The PU that contains the minimum LLR receives the parity check bit and the others receive 0s. The transmission of parity check bit is done by the PTU which is a two input two output module. One input is the $parity\ check\ bit$ (PCB) and the other is the $select\ signal$ (SS). The parity check bit is transmitted via $output$ 1 (O1) or $output$ 2 (O2) bases on the values of SS. Table. I shows the truth table of PTU. We can obtain the logic expression of O1 and O2 as: $O1 = PCB\ and\ \overline{SS}$ ,$O2 = PCB\ and\ SS$. This can be done by two $and$ gates and one $Inverter$.

TABLE I. TRUTH TABLE OF PTU

| PCB | SS | O1 | O2 | PCB | SS | O1 | O2 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |

The PU in $stage0$, as denote $PU_0$ in Fig. 3, has a simpler architecture. Fig. 4b shows the design details of $PU_0$. Since only one more clock cycle need for single parity check, there is no feed back to this register. Furthermore, $\mathcal{N}^{SPC}$ cannot exist in $stage0$. So top part in Fig. 4a which is relative to single parity check can be removed. For $g$ function and $\mathcal{N}^{REP}$ , the output of $f$ function can be feed back to it immediately, and the sign bit of the result of adding is the partial sum for $\mathcal{N}^{REP}$.

### D. Fixed point analysis

Fig. 5 shows the effect of quantization on the $(1024, 512)$ polar code. For channel outputs and inner LLRs, we use separate quantization schemes. The quantization schemes are shown in $(C, L, F)$ format. Where $C$, $L$ and $F$ are the number of bits used for presenting channel output, inner LLRs and fraction parts of both channel output and LLRs, respectively. Since no multiplication or division used, which means the length of fraction does not change, channel outputs and inner LLRs use the same fraction precision. As the result of the trade-off between hardware efficiency and decoding performance, we choose $(4, 5, 0)$ quantization scheme in our design.

## IV. HARDWARE ANALYSIS AND COMPARISON

Table. II shows the hardware comparisons between proposed design and other state-of-the-art designs. All the candidates are $(n, k)$ SC decoder with tree architectures, and they all use $q$-bit quantization for inner LLRs. All the throughputs and hardware complexity (HC) are normalized to the SC decoder in [4], and the hardware complexity is estimated based on the synthesis results. The latency for proposed design is a range with respect to the code rates change from $0.05$ to $0.95$. From this table, we can see that our proposed design achieves the highest throughput per unit of hardware complexity. The exact latency depends on the code rate. Fig. 6 shows the latency reduction of the proposed design along with code rates from $0.05$ to $0.95$. The reduction is relative to the 2b-SC-Precomputation decoder which so far is known to be the fastest. The figure shows at least $60\%$ latency reduction can be achieved by our proposed design. This is very promising for many applications where high rate channel codes are needed, such as for data storage system.

TABLE II. HARDWARE COMPARISON OF DIFFERENT $(n, k)$ SC DECODER WITH $q$-BIT QUANTIZATION FOR INNER LLRS USING TREE ARCHITECTURE

| Hardware Type | [7] | [4] | [8] | Proposed Design |
|---|---|---|---|---|
| # of PU | $n-1$ | $n-1$ | $n-1$ | $n-1$ |
| # of PTU | 0 | 0 | 0 | $2/n-1$ |
| # of 1 bit REG | $\approx 3qn$ | $\approx qn$ | $\approx 3qn$ | $\approx (3q+1)n$ |
| HC | 1.3 | 1 | 1.3 | 1.31 |
| Latency (clock cycle) | $n-1$ | $2n-2$ | $0.75n-1$ | $\approx (0.1 \sim 0.3)n$ |
| Throughput | 2 | 1 | 2.67 | $\approx 6.69 \sim 22.26$ |
| Throughput/HC | 1.53 | 1 | 1.74 | $5.1 \sim 16.99$ |

Additionally, we implemented the proposed design with $Verilog$ for the polar code with length=1024 and synthesized it using $Nangate\ FreePDK\ 45nm$ process with $Synopsys\ Design\ Complier$. We calculated the throughput for $(1024, 870)$ and $(1024, 512)$ polar codes. Table III shows the synthesis result for $(1024, 870)$ and $(1024, 512)$ polar codes. Notice that the maximum frequency is higher than that reported in [8] which use the same process as our design. Our design in theory should have a lower maximum frequency since we have one more Mux delay for regular and fast constituent polar codes. This performance improving is attributable to the modification we have done to PU as described in section III-C.

TABLE III. SYNTHESIS RESULT FOR $(1024, 870)$ AND $(1024, 512)$ POLAR CODES

| | |
|---|---|
| Silicon Area ($\mu m^2$) | 275899 |
| Max Frequency (GHz) | 1.04 |
| Latency (1024,870) (clock cycle) | 156 |
| Throughtput(1024,870) (Gbps) | 5.81 |
| Latency (1024,512) (clock cycle) | 266 |
| Throughtput(1024,512) (Gbps) | 2.01 |

## V. CONCLUSION

In this paper, we proposed a hardware architecture of fast-SSC algorithm for polar codes. By exploiting the similarity between the decoding processing of fast constituent and regular polar codes, proposed design overcomes the disadvantage of fast-SSC decoder that lacking decoding flexibility with respect to multiple code rates. Corresponding scheduling plan and the intendedly designed PU are also described. Result shows that proposed design significantly increase the decoding throughput of polar codes compared with other state-of-art SC decoders.

## REFERENCES


[1] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *Information Theory, IEEE Transactions on*, vol. 55, no. 7, pp. 3051–3073, 2009.

[2] I. Tal and A. Vardy, "List decoding of polar codes," in *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 1–5.

[3] J. Xu, T. Che, and G. Choi, "Xj-bp: Express journey belief propagation decoding for polar codes," *arXiv preprint arXiv:1504.06025*, 2015.

[4] C. Leroux, I. Tal, A. Vardy, and W. J. Gross, "Hardware architectures for successive cancellation decoding of polar codes," in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1665–1668.

[5] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, "A semi-parallel successive-cancellation decoder for polar codes," *Signal Processing, IEEE Transactions on*, vol. 61, no. 2, pp. 289–299, 2013.

[6] A. Mishra, A. Raymond, L. Amaru, G. Sarkis, C. Leroux, P. Meinerzhagen, A. Burg, and W. Gross, "A successive cancellation decoder asic for a 1024-bit polar code in 180nm cmos," in *Solid State Circuits Conference (A-SSCC), 2012 IEEE Asian*. IEEE, 2012, pp. 205–208.

[7] C. Zhang, B. Yuan, and K. K. Parhi, "Reduced-latency sc polar decoder architectures," in *Communications (ICC), 2012 IEEE International Conference on*. IEEE, 2012, pp. 3471–3475.

[8] B. Yuan and K. K. Parhi, "Low-latency successive-cancellation polar decoder architectures using 2-bit decoding," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 61, no. 4, pp. 1241–1254, 2014.

[9] A. Alamdar-Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE communications letters*, vol. 15, no. 12, pp. 1378–1380, 2011.

[10] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast polar decoders: Algorithm and implementation," *Selected Areas in Communications, IEEE Journal on*, vol. 32, no. 5, pp. 946–957, 2014.

[11] I. Tal and A. Vardy, "How to construct polar codes," *Information Theory, IEEE Transactions on*, vol. 59, no. 10, pp. 6562–6582, 2013.

[12] C. Zhang and K. K. Parhi, "Low-latency sequential and overlapped architectures for successive cancellation polar decoder," *Signal Processing, IEEE Transactions on*, vol. 61, no. 10, pp. 2429–2441, 2013.