Energy-Efficient Radix-4 Belief Propagation Polar Code Decoding Using an Efficient Sign-Magnitude Adder and Clock Gating

Oğuz Meteer [1], Arvid van den Brink [1], and Marco J.G. Bekooij [1,2]

[1]Department of Computer Architectures for Embedded Systems, University of Twente, Enschede, The Netherlands
[2]Department of Embedded Software and Signal Processing, NXP Semiconductors, Eindhoven, The Netherlands
{*o.meteer, a.b.vandenbrink*}*@utwente.nl* [1], *marco.bekooij@nxp.com* [2]

2022 25th Euromicro Conference on Digital System Design (DSD) | 978-1-6654-7404-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/DSD57027.2022.00026

***Abstract*—Polar encoding is the first information coding method that has been proven to achieve channel capacity for binary-input discrete memoryless channels. Since its introduction, much research has been done on improving decoding performance, execution time and energy efficiency. Classic belief propagation uses radix-2 decoding, but a recent study proposed radix-4 decoding which reduces memory usage by 50%. However a drawback is its higher computational complexity, negatively impacting energy usage and throughput.**

In this paper we present an energy-efficient radix-4 belief propagation polar decoder architecture that uses a new sign-magnitude adder that does not require conversion to two's complement and back. On top of that we also propose using clock gating of input values by checking if all R inputs of the decoder are zero. These two key contributions lead to a more energy-efficient design that is smaller and has higher maximum clock speed and throughput.

Post-layout simulation results show that compared to the previously proposed 1024-bit radix-4 belief propagation polar code decoder, our decoder uses between 30.22% and 32.80% less power and is 5.2% smaller at the same clock speed. Also, our design can achieve a 15.7% higher clock speed at which it is still up to 10.76% more power efficient and 4.8% smaller.

***Index Terms*—polar codes, belief propagation decoding (BPD), energy efficiency, low-power design, signal processing systems**

I. Introduction

While capacity-approaching codes such as low-density parity-check (LDPC) codes [1] and Turbo codes [2] have seen wide spread use in applications ranging from wireless communication to data storage, the first provable capacity-achieving codes called polar codes were invented by Arıkan [3]. Polar codes are the first in the family of provable capacity-achieving codes for symmetric, binary input memoryless channels [3] as well as arbitrary, discrete and continuous memory channels [4]. Polar codes also have garnered attention from the industry as they were chosen as a channel encoding scheme for uplink and downlink control information for the 5th generation wireless systems (5G) by the 3rd generation partnership project (3GPP) [5].

Numerous decoding algorithms for polar codes have been proposed with the the two main decoding algorithms being successive cancellation (SC) and belief propagation (BP). SC decoding algorithms are serial in nature and therefore require less computation but have much higher latency compared to BP decoding. BP decoding algorithms on the other hand are embarrassingly parallel, making them more suitable for low-latency applications. However, BP decoding also has two disadvantages. The first is higher computational complexity compared to SC decoding and the second is the iterative nature of BP decoding, meaning that the latency and energy usage grows linearly with the amount of iterations.

In this paper we propose two gate level improvements that are applicable to any BP polar code decoder implementation. The first is an efficient sign-magnitude adder that does not convert to two's complement for performing an addition as is normally done, and is therefore smaller, faster and more power efficient. The second is an efficient clock gating condition that disables almost all decoder hardware, significantly decreasing switching activity and dynamic power usage. We also present two radix-4 BP polar code decoder implementations that significantly improve power and energy efficiency compared to [6], which to the best of our knowledge, is the only other radix-4 BP polar code decoder in literature.

The paper is organized as follows. In Section II we give a brief overview of polar codes and the SC and BP algorithms. Then related work is described in Section III. Section IV describes the basic idea behind our proposed solutions. Next, in Section V, we evaluate the reference and proposed implementations, perform post-layout power analysis, and list the power usage, area, maximum frequencies and throughput. Section VI addresses future work. Finally, we conclude the paper in Section VII.

II. Polar Codes

Polar codes, abbreviated as $\mathcal{P}(N,K)$, are defined by the parameters (N, K, $\mathcal{A}$, $u_{\mathcal{A}c}$), where $N = 2^n$, $\forall n > 1$ is the length of the code word, K is the number of information bits, $\mathcal{A}$ is the set of indices of the information bits, and u_{Ac} are the frozen bits [3] which are predetermined hard coded values (often zeroes). Encoding a source vector

2771-2508/22/$31.00 ©2022 IEEE
DOI 10.1109/DSD57027.2022.00026

$u_1^N = (u_1, u_2, \cdots, u_{N-1}, u_N)$ into a code word $x_1^N = (x_1, x_2, \cdots, x_{N-1}, x_N)$ happens using the following equation:

$$x_1^N = u_1^N G_N = u_1^N B_N F^{\otimes n} \tag{1}$$

where G_N is the generator matrix, B_N is the permutation (or bit-reversal) matrix, and $F^{\otimes n}$ is the *Kronecker* power of F. The Kronecker power is defined as:

$$F^{\otimes n} = F \otimes F^{\otimes (n-1)} \tag{2}$$

where $n = \log_2(N)$ and $F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$.

Then x_1^N is transmitted over the polar code constructed virtual channels W^N where the channel output $y_1^N = (y_1, y_2, \cdots, y_{N-1}, y_N)$ is received and fed to the decoder input. The decoder then uses the polar code knowledge $\mathcal{A}$, $u_{\mathcal{A}^c}$ and the received y_1^N to produce the estimate $\hat{u}_1^N$.

A. Belief Propagation Decoding

A belief propagation encoder and decoder can be constructed using a factor graph [7]. The factor graph of a basic, radix-2 computational unit (CU) of belief propagation for polar codes is shown in Fig. 1a. These CUs can be combined in multiple stages to create decoders for larger code words. A factor graph for a decoder with code word length $N = 4$ is shown in Fig. 1b.

Each CU has four terminals, where the terminals on the left and right side are the R and L inputs for right-bound and left-bound messages respectively. They are shown in Fig. 1a and map to the following equations:

$$R_{s+1,j}^{(i+1)} = g(R_{s,j}^{(i)}, L_{s+1,j+2^{n-s}}^{(i)} + R_{s,j+2^{n-s}}^{(i)}) \tag{3}$$

$$R_{s+1,j+2^{n-s}}^{(i+1)} = g(R_{s,j}^{(i)}, L_{s+1,j}^{(i)}) + R_{s,j+2^{n-s}}^{(i)} \tag{4}$$

$$L_{s,j}^{(i+1)} = g(L_{s+1,j}^{(i)}, L_{s+1,j+2^{n-s}}^{(i)} + R_{s,j+2^{n-s}}^{(i)}) \tag{5}$$

$$L_{s,j+2^{n-s}}^{(i+1)} = g(R_{s,j}^{(i)}, L_{s+1,j}^{(i)}) + L_{s+1,j+2^{n-s}}^{(i)} \tag{6}$$

with $1 \leq i \leq I_{max}$, where I_{max} is the maximum number of iterations, s is the stage number, j is the CU index and g is the min-sum approximation function:

$$g(x, y) \approx 0.9 \cdot \text{sign(x)} \cdot \text{sign(y)} \cdot \min(|x|, |y|) \tag{7}$$

BP is an iterative algorithm where messages are processed using the constructed factor graph, where each iteration involves two types of messages: left bound L and right bound R. Each message is represented by a log likelihood ratio (LLR) and they are all initially assigned an LLR depending on their index in the factor graph. The left most R messages are assigned an LLR as follows:

$$R_{1,j} = \begin{cases} 0, & \text{if } j \in \mathcal{A}; \\ \infty & \text{if } j \in \mathcal{A}^c \end{cases} \tag{8}$$

where $\mathcal{A}$ is the set of information bits and $\mathcal{A}^c$ is the set of frozen bits. The right most L messages are assigned an LLR as follows:

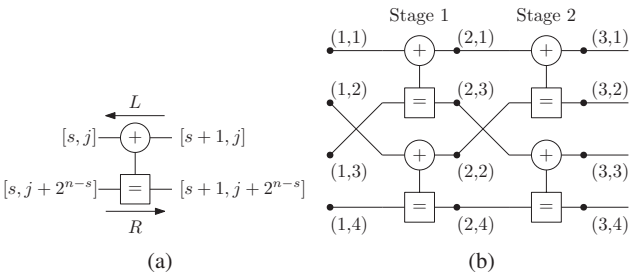


Fig. 1: (a) Belief propagation (BP) computational unit (CU); (b) Polar code factor graph for code length $N = 4$.

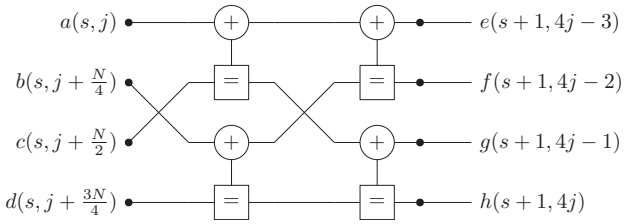


Fig. 2: Factor graph of a radix-4 BP CU.

$$L_{n+1,j} = \ln\left(\frac{\Pr\left[y_j | x_j = 0\right]}{\Pr\left[y_j | x_j = 1\right]}\right) \tag{9}$$

All other, intermediate node messages in the factor graph are initialized with an LLR value of zero.

After iterating for I_{max} iterations, the decoder estimation $\hat{u}_1^N$ is determined using threshold detection at the left most terminals as follows:

$$\hat{u}_j = \begin{cases} 0, & \text{if } L_{1,j} + R_{1,j} \geq 0; \\ 1, & \text{otherwise} \end{cases} \tag{10}$$

B. Stage-Combined Belief Propagation Decoding

While classical BP decoding uses radix-2 CUs, a stage-combined BP decoder for polar codes was introduced in [8] and [9]. They propose a radix-4 BP decoder that has no decoding performance degradation and has two benefits. First, the number of stages processed per iteration is reduced from $(2\log_2(N)) - 1$ to $\log_2(N) - 1$, therefore reducing the decoding latency. Second, the memory storage and the number of read and write operations are reduced from $N(\log_2(N) - 1)$ messages to $N(\log_4(N) - 1)$ messages, making radix-4 decoding more efficient in terms of processing for larger code words compared to radix-2 decoding.

The factor graph of the proposed radix-4 CU is shown in Fig. 2, where the eight terminals are named a through h for simplicity and are mapped to the following equations:

$$
\begin{aligned}
R_e &= g(R_a, g(R_b, L_g) + g(L_f, R_c) \\
&\quad + g(R_d + L_h, g(R_b, R_c) + g(L_f, L_g))) \\
R_f &= g(R_b, R_d + L_h + g(R_c, L_g)) \\
&\quad + g(R_a, g(L_e, R_c + g(L_g, R_d + L_h))) \\
R_g &= g(R_c, R_d + L_h + g(R_b, L_f)) \\
&\quad + g(R_a, g(L_e, R_b + g(L_f, R_d + L_h))) \\
R_h &= R_d + g(R_c, L_g) + g(R_b, L_f) \\
&\quad + g(R_a, g(L_e, g(R_b, R_c) + g(L_f, L_g)))
\end{aligned} \tag{11}
$$

$$
\begin{aligned}
L_a &= g(L_e, g(R_b, L_g) + g(L_f, R_c) \\
&\quad + g(R_d + L_h, g(R_b, R_c) + g(L_f, L_g))) \\
L_b &= g(L_f, R_d + L_h + g(R_c, L_g)) \\
&\quad + g(L_e, g(R_a, L_g + g(R_c, R_d + L_h))) \\
L_c &= g(L_g, R_d + L_h + g(R_b, L_f)) \\
&\quad + g(L_e, g(R_a, L_f + g(R_b, R_d + L_h))) \\
L_d &= L_h + g(R_c, L_g) + g(R_b, L_f) \\
&\quad + g(L_e, g(R_a, g(R_b, R_c) + g(L_f, L_g)))
\end{aligned} \tag{12}
$$

The architecture for all four outputs are shown in Fig. 3. *Type I* blocks implement Eq. 7 and *Type II* blocks are sign-magnitude adders. We would like to note the small discrepancy in [6], where the architecture for outputs R_f/L_b, R_g/L_c and R_h/L_d differ from the decoding equations in Eq. 11 and Eq. 12. This is corrected in our implementation.

III. Related Work

Several studies have proposed algorithmic methods to decrease the average number of iterations and thus latency and energy usage. In [10], two early stopping criteria for use with BP decoding were introduced in [10]. These criteria make use of the fact that after converging, additional iterations in BP decoding do not change the result and that convergence is often reached in less iterations than the maximum iteration count. These criteria detect convergence using the G-matrix after which computation is stopped. As a result, the average number of iterations decreased up to 42.5% at an SNR of 3.5 dB. For their $\mathcal{P}(1024, 512)$ implementation, the average throughput increased between 20% and 50%, the energy usage as well as average latency was reduced between 10% and 30%, while the area overhead only increased between 2% and 5%.

Another early stopping architecture was introduced in [11] based on comparing the last three consecutive hard decoding decisions. They also introduced a double-column architecture where the memory is split into two parts and the amount of computation units (CU) is doubled. In their $\mathcal{P}(1024, 512)$ implementation, this increased the clock period from 3.5 ns to 4.0 ns, but the iteration latency decreases from 19 to 10 clock cycles which results in an increased throughput of 66%. And while there are twice the amount of CUs in the double-column architecture, the increase in area is only 28%.

In addition to early stopping, a different study proposed an energy-efficient BP polar code decoder [12] where a novel scheme based on subfactor-graph freezing was proposed. During each iteration, they propose checking whether subfactor graphs have converged, and freezing the computation of those subfactor graphs. This not only decreases the average number of iterations, but also decreases the average number of computations, resulting in a 60% to 73% improvement in energy-efficiency in their $\mathcal{P}(1024, 512)$ BP decoder implementation.

While these and other studies focused on decreasing the average number of iterations and computations, the CUs are all based on radix-2 decoding. However, a stage-combined BP decoder is proposed in [8] and [9] which introduces radix-4 CUs. Although the computational complexity is increased compared to radix-2 decoding, the amount of required memory is halved. Also, the amount of memory required increases whenever the code word length is quadrupled instead of whenever it is doubled as with radix-2 decoding, making it more attractive for larger code words. Despite the higher decoding complexity, radix-4 presents new optimization opportunities.

These proposed *algorithmic* optimizations still use the basic building blocks such as the g function in Eq. 7 and an adder. In contrast, we propose two *gate level* optimizations that can improve energy efficiency, area, and throughput. Since we propose optimizations to the basic building blocks, they can also be used in combination with any algorithmic optimizations that others have proposed.

IV. Basic Idea

Most, if not all polar code decoder implementations use the sign-magnitude (SM) number format for messages. This is because the absolute values of x and y are required in the $\min(|x|, |y|)$ part of the g function of Eq. 7. Therefore the magnitude part of messages can be directly compared without using hardware to calculate the absolute value, simplifying the implementation of the g function. However, addition of SM numbers is not trivial, because if the signs are different then simply adding the magnitudes yields the wrong answer. The default method is to convert to two's complement (TC), perform the addition and then convert back to SM. This requires a converter that XORs the magnitude bits with the most significant bit (MSB) and adds the MSB to the result. This converter is shown in Fig. 4 and Fig. 5 shows a simple SM adder with these converters. Compared to just a TC adder, this SM adder requires more hardware and the critical path is much longer due to two additional +1 adders.

A. Efficient Sign-Magnitude Adder

We propose an efficient SM adder, shown in Fig. 6. It converts negative values to one's complement (OC) instead of TC, which effectively inverts the magnitude by using only the XOR gates of the converter shown in Fig. 4. Compared to the default SM adder in Fig. 5, it does not require any +1 adders which reduces the critical path, area and energy usage. Connecting the carry output of the TC adder to its carry input and to a full adder is required to calculate the correct result by conditionally enabling the XOR gates at the adder output.

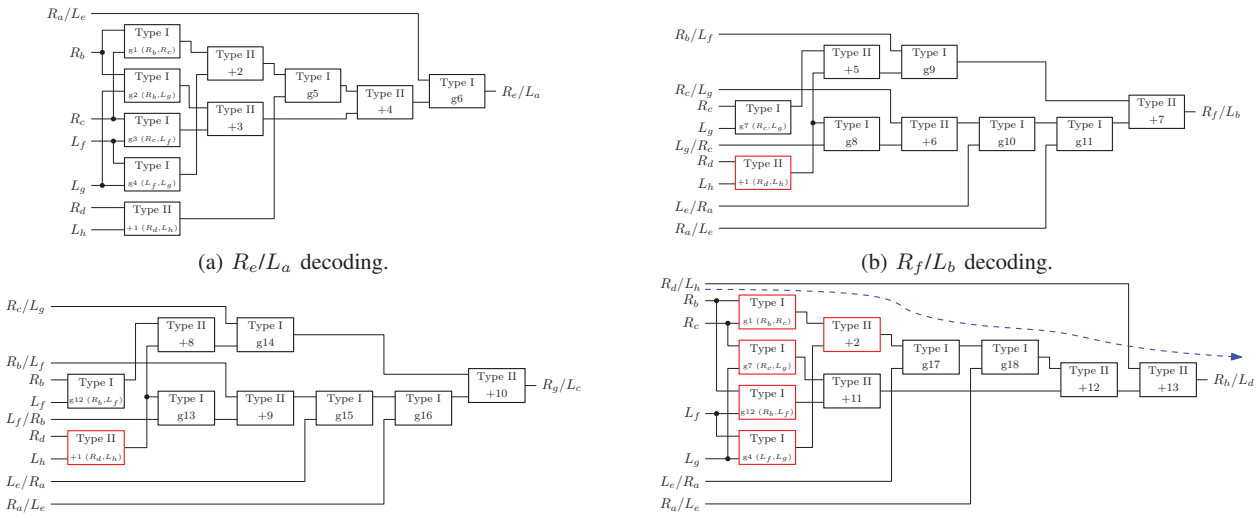


Fig. 3: Radix-4 CU architecture for computing the decoding equations in Eq. 11 and Eq. 12. The red boxes show the shared hardware blocks and the blue arrow shows the critical path.

From the perspective of TC, the OC operation is equivalent to $\overline{A} = -A - 1$. Because of this, using OC can generate incorrect results if two values with different signs are added together. If we perform $S = -A + B$ then using OC gives us $S = \overline{A} + B = -A - 1 + B$ which is 1 off from the correct result. If $S < 0$, then applying the OC on S gives the correct answer since it negates the previously missing 1. This is because applying the OC twice yields the same value, i.e. $\overline{\overline{A}} = \overline{-A-1} = -(-A-1) - 1 = A$. But if $S \geq 0$ then the resulting value is $S - 1$ which requires adding one to the result and thus needs a +1 adder. The solution is to apply the OC to negative inputs, add the magnitudes and then add the carry output to the result.

We note that in Fig. 5 the carry input of the TC adder is not used, meaning that it can be used to conditionally add the missing +1. In our SM adder, we simply connect the carry out (indicated as G_{0-3} in Fig. 6) to the carry in, but in order to not create a combinatorial loop, an adder is needed that calculates the carry output signal independently of the other carry bits. One such adder is the Carry Lookahead adder (CLA) [13] and a 4-bit example of the internals is shown in Fig. 7. We see that the P and G outputs of the partial full adder (PFA) blocks are independent of the C input, which only affects the S output. This is why connecting G_{0-3} to C_0, which eventually generates C_3, does not create a combinatorial loop. A classic CLA uses the G_{0-3} and P_{0-3} signals to generate the carry output whereas in our SM adder, the carry output is the G_{0-3} signal, so the computation of the final P_{0-3} can be omitted. Larger adders can be built in the exact same way as the classic CLA by creating a tree of Carry Lookahead blocks.

B. Clock Gating

A recent study showed that when performing BP decoding, the values 0 and $+\mathbb{U}max$ appear in combinations that can

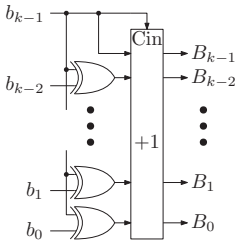


Fig. 4: Hardware for converting to and from sign-magnitude and two's complement.

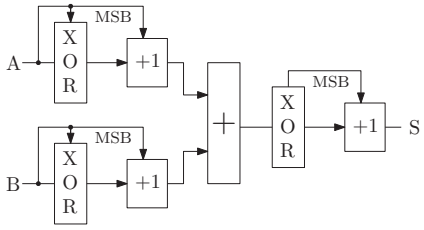


Fig. 5: Trivial addition of two sign-magnitude numbers.

simplify the decoding equations [14], whose two fundamental operations are the g function of Eq. 7 and addition. When at least one of the inputs of g is 0 then the result is also 0 and the g operation can therefore be removed. The same also applies to addition since adding 0 to a value does not change its result. However, when at least one of the inputs to any of the two operations is $+\mathbb{U}max$, then only g can be partially simplified as the $min(|a|, |b|)$ part always yields the scaled version of the input that is not $+\mathbb{U}max$, saving a comparison operation. Also, adding $+\mathbb{U}max$ to a value does not have a predetermined result and therefore the addition cannot be

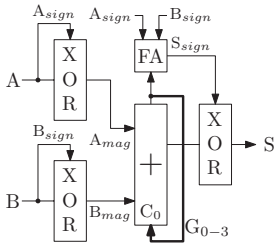


Fig. 6: Our proposed efficient sign-magnitude adder.

TABLE I: Occurrence of the values 0 and +Umax for each input of all CUs for a radix-4 $\mathcal{P}$(1024,512) BP polar code decoder using 15 iterations and 10000 random code words. Shown are the percentages for an SNR of 0.0 dB and 4.0 dB.

Iteration Direction	Right-bound				Left-bound			
Value	0		$+\mathbb{U}max$		0		$+\mathbb{U}max$	
SNR (dB)	0.0	4.0	0.0	4.0	0.0	4.0	0.0	4.0
R_a	29.5%	23.4%	20.1%	50.1%	25.5%	20.1%	16.8%	51.9%
R_b	44.4%	39.6%	17.5%	42.4%	39.3%	33.1%	14.0%	46.1%
R_c	40.9%	32.9%	18.2%	43.6%	35.8%	27.7%	14.7%	47.3%
R_d	55.5%	47.6%	15.2%	27.6%	51.5%	39.7%	12.1%	22.1%
L_e	12.9%	6.8%	0.8%	46.2%	5.3%	0.1%	0.8%	43.8%
L_f	12.9%	6.8%	0.7%	47.3%	5.3%	0.1%	0.7%	44.7%
L_g	12.8%	6.8%	0.8%	47.3%	5.2%	0.1%	0.8%	44.8%
L_h	12.8%	6.8%	0.8%	47.5%	5.3%	0.1%	0.7%	44.9%

removed. For this reason, finding a clock gating condition with input values being 0 is preferred as it can potentially simplify the decoding equations more than input values being $+\mathbb{U}max$.

While radix-2 BP decoding was used in [14], the same also applies to radix-4 BP decoding as it is a property of BP decoding rather than the used radix. However, the impact of those simplifications can vary greatly between radix-2 and radix-4 BP decoding, because the decoding equations for radix-4 have many more operations. This means that potentially, many more operations can be simplified in radix-4 BP decoding. The downside is that there are a great number of combinations with certain inputs and varying amounts of operations that can be skipped, leading to a very large optimization space.

To research this optimization space we have implemented a radix-4 BP polar decoder in Matlab. We analyzed the frequency of these values for each input of each CU in the SNR range of $[0.0, 4.0]$ dB. For brevity, only the results with an SNR of 0.0 and 4.0 dB are shown in Table I and the results with other SNR values lie between the reported extremes.

An efficient clock gating condition should occur often while also simplifying the decoding equations as much as possible. If we focus on optimizations based on input values being 0, then the results show that this occurs much more frequently for R inputs compared to L inputs. However, any single R input alone being 0 does not simplify the decoding equations as much as combinations of multiple R inputs being 0. We therefore analyzed these combinations which can be found in Table II. The condition $R_b \cup R_c \cup R_d = 0$ appears the most often and gives us the following decoding equations:

TABLE II: Occurrence of the value 0 for combinations of R inputs of all CUs for a $\mathcal{P}$(1024,512) BP polar code decoder using 15 iterations and 10000 random code words. Shown are percentages for an SNR of 0.0 dB and 4.0 dB.

Iteration Direction	Right-bound		Left-bound	
Value	0		0	
SNR (dB)	0.0	4.0	0.0	4.0
R_a	29.5%	23.4%	25.5%	20.1%
$R_a \cup R_b$	28.9%	23.4%	24.6%	20.0%
$R_a \cup R_c$	28.4%	23.3%	24.1%	20.0%
$R_a \cup R_d$	29.3%	23.4%	25.0%	20.1%
$R_a \cup R_b \cup R_c$	28.3%	47.6%	51.5%	39.7%
$R_b \cup R_c \cup R_d$	38.3%	32.7%	32.1%	27.5%
$R_a \cup R_b \cup R_c \cup R_d$	28.3%	23.3%	24.0%	20.0%

$$
\begin{aligned}
R_e &= g(R_a, g(L_h, g(L_f, L_g))) \\
R_f &= g(R_a, g(L_e, g(L_g, L_h))) \\
R_g &= g(R_a, g(L_e, g(L_f, L_h))) \\
R_h &= g(R_a, g(L_e, g(L_f, L_g))) \\
L_a &= g(L_e, g(L_h, g(L_f, L_g))) \\
L_b &= g(L_f, L_h) + g(L_e, g(R_a, L_g)) \\
L_c &= g(L_g, L_h) + g(L_e, g(R_a, L_f)) \\
L_d &= L_h + g(L_e, g(R_a, g(L_f, L_g)))
\end{aligned}
\tag{13}
$$

The results also show than whenever R_a is 0 then R_b, R_c and R_d are almost always 0 too making preferable to choose $R_a \cup R_b \cup R_c \cup R_d = 0$ over $R_a \cup R_b \cup R_c = 0$ as the former removes much more hardware and greatly simplifies the decoding equations of Eq. 11 and Eq. 12 to the following:

$$
\begin{aligned}
R_e &= 0 \qquad L_a = g(L_e, g(L_h, g(L_f, L_g))) \\
R_f &= 0 \qquad L_b = g(L_f, L_h) \\
R_g &= 0 \qquad L_c = g(L_g, L_h) \\
R_h &= 0 \qquad L_d = L_h
\end{aligned}
\tag{14}
$$

Although $R_a \cup R_b \cup R_c \cup R_d = 0$ occurs between 7.5% and 10% less often than $R_b \cup R_c \cup R_d = 0$, we choose the former clock gating condition because it greatly simplifies the decoding equations and thus the hardware. An overview of the Radix-4 CU architecture with clock gating is shown in Fig. 8. The common hardware which consists of the input registers and the clock gating condition is shown in Fig. 8a and the hardware for computing Eq. 14 in addition to Eq. 11 and Eq. 12 is shown in Fig. 8b through Fig. 8e.

V. Evaluation

In this section we evaluate a stage-combined, radix-4 BP polar code decoder implementation as a reference (*Ref*) as described in [6] and our two proposed designs: *SM* uses our proposed SM adder, and *SM+GC* uses our SM adder and clock gating. A high level overview of the architecture of *SM+GC* is shown in Fig. 9 which consists of an array of 256 radix-4 CUs, a register file for storage, routers for routing the

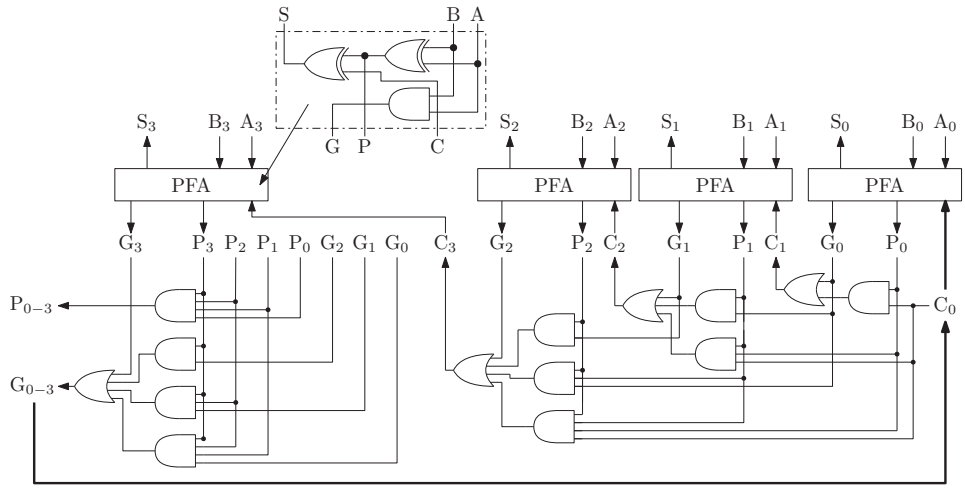
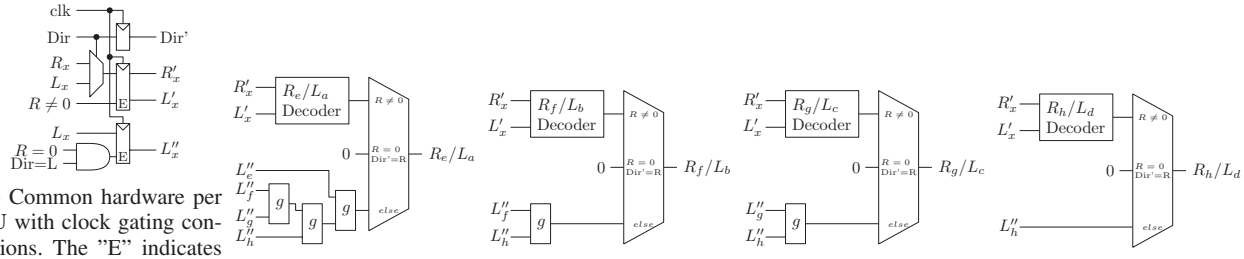


Fig. 7: Internal structure of our proposed SM CLA. In the PFA block we see that the P and G outputs are not dependent on the C input. And since only P and G signals are used to generate the G_{0-3} signal, connecting it to the C_0 input does not produce a combinatorial loop. A 4-bit adder is shown here.



(a) Common hardware per CU with clock gating conditions. The "E" indicates the clock enable input.

(b) R_e/L_a decoding.

(c) R_f/L_b decoding.

(d) R_g/L_c decoding.

(e) R_h/L_d decoding.

Fig. 8: Radix-4 CU architecture with clock gating for computing the decoding equations in Eq. 11, Eq. 12 and Eq. 14.

initial LLR and intermediate messages according to the factor graph and the clock gating hardware. Other implementations are similar but lack the clock gating logic. Only the hardware inside the dashed box was implemented and evaluated. For all implementations we use code length N=1024 and information bits K=512. Similar to [6], we use 7 bit sign-magnitude numbers and instead of a scaling factor of 0.9 in the g function in Eq. 7, a scaling factor of 0.875 is used since it simplifies the hardware implementation. We verified that the decoding performance with these settings is similar to a classic radix-2 BP decoder. Throughput is calculated as:

$$TP = \frac{f \times N}{I_{max} \times S} \tag{15}$$

where f is the clock frequency, N is the code length, I_{max} is the number of BP iterations and S is the number of stages. In our evaluation $N = 1024$, $I = 15$ and $S = 2log_4(N) - 1 = 9$.

We synthesized all implementations using a UMC 65nm standard cell library using Synopsys Design Compiler with *high* synthesis and mapping effort. A typical-typical corner was used with a supply voltage of $V_{dd} = 1.2V$. Place and route was done with Cadence Innovus, and post-layout power calculation was performed using Synopsys PrimeTime.

We created a framework in Matlab to develop and evaluate different kinds of decoders, which also generates stimulus and expected results files. These files were then used by the test bench to verify correctness of the hardware implementations, and were also used for power simulations. All implementations were tested with an SNR in the range of 0.0 dB to 4.0 dB and test vector size of 100 words per SNR.

A. Results

The synthesis results are shown in Table III where we compare the reference implementation (*Ref*) [6], our implementation with just our proposed sign-magnitude adders (*SM*) and our implementation also adds clock gating (*SM+CG*). The reference implementation can run at a maximum clock speed of 142.9 MHz and has a throughput of 1084 MiB/s.

Compared to the *Ref* design, our *SM* implementation has a 18.6% higher clock speed and throughput at 169.5 MHz and 1286 MiB/s respectively. It is also 18.6% smaller because our proposed sign-magnitude adder does not require any conversion hardware to and from two's complement unlike in the *Ref* implementation. When designing for the maximum clock speed of *Ref*, our *SM* implementation is 20.2% smaller.

Our *SM+CG* implementation has a 15.7% higher clock

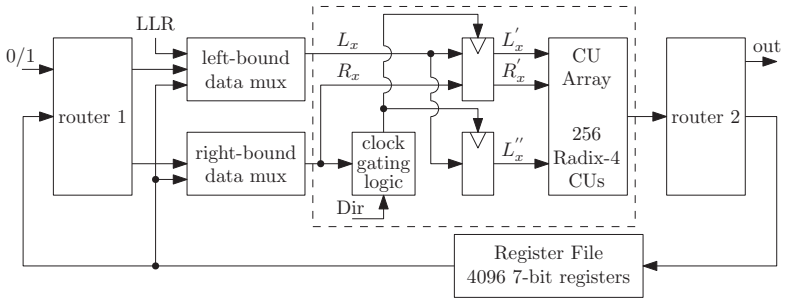


Fig. 9: High level architecture overview of our proposed $\mathcal{P}(1024, 512)$ BP decoder. The hardware inside the dashed box was implemented and evaluated for all designs.

TABLE III: Hardware comparison of radix-4 $\mathcal{P}$(1024,512) BP polar code decoders.

Implementations	Ref. [6]	SM		SM+GC	
Block Length	1024				
Iterations	15				
Message bits	7				
Process (nm)	65				
Supply (V)	1.2				
Frequency (MHz)	142.9	169.5	142.9	165.3	142.9
Core Area (mm^2)	2.126	1.730	1.696	2.023	2.015
Core Utilization	67.0%	65.5%	77.4%	68.8%	76.4%
Throughput (Mb/s)	1084	1286	1084	1254	1084

speed and throughput at 165.3 MHz and 1254 MiB/s respectively, and is 4.8% smaller, whereas it is 5.2% smaller when designed for the same clock speed as the *Ref* design. Even with the clock gating logic and redundant hardware to compute the outputs shown in Eq. 14, our *SM+GC* implementation still has an advantage in terms of area.

The power usage and energy efficiency results of all implementations are shown in Fig. 10a and Fig. 10b respectively. First we note that power usage is heavily dependent on the SNR because with a lower SNR, noise is more likely to cause unnecessary sign changes. And even though the sign-magnitude number format and our proposed SM adder is used, it still requires inverting negative values which leads to a higher switching activity.

Compared to the *Ref* implementation, our *SM* implementation at the same clock speed of 142.9 MHz improves energy efficiency and decreases power usage between 4.90% and 10.94% for an SNR of 4.0 and 0.0 dB respectively. This shows that our proposed SM adder produces less switching activity due to removing the unnecessary adders used by the conversion hardware in Fig. 4. When we look at our *SM* implementation at the maximum clock speed of 169.5 MHz, it uses between 8.66% and 18.35% more power compared to the *Ref* design, but it also runs at a 18.64% higher clock frequency and the same increase in throughput. Therefore its energy efficiency is between almost equal and 8.41% more efficient at 4.0 and 0.0 dB SNR respectively. The lines for *Ref* at 142.9 MHz and *SM* at 169.5 MHz are almost the same at an SNR of 4.0 dB because at that point, the throughput and power usage increase at the same rate.

Looking at our *SM+GC* implementation at the same clock speed as *Ref*, it increases energy and power efficiency between 30.22% and 32.80% for an SNR of 4.0 dB and 0.0 dB respectively. Even with a 15.7% higher clock speed and throughput, the *SM+GC* implementation at 165.3 MHz uses between 6.30% and 10.76% less power than the *Ref* implementation at 142.9 MHz for an SNR of 4.0 and 0.0 dB respectively.

VI. Future Work

It would be interesting to evaluate our proposed techniques for a radix-2 instead of a radix-4 BP decoder. However we suspect that the benefits of clock gating to be lower and of the new sign-magnitude adder to be slightly higher.

VII. Conclusion

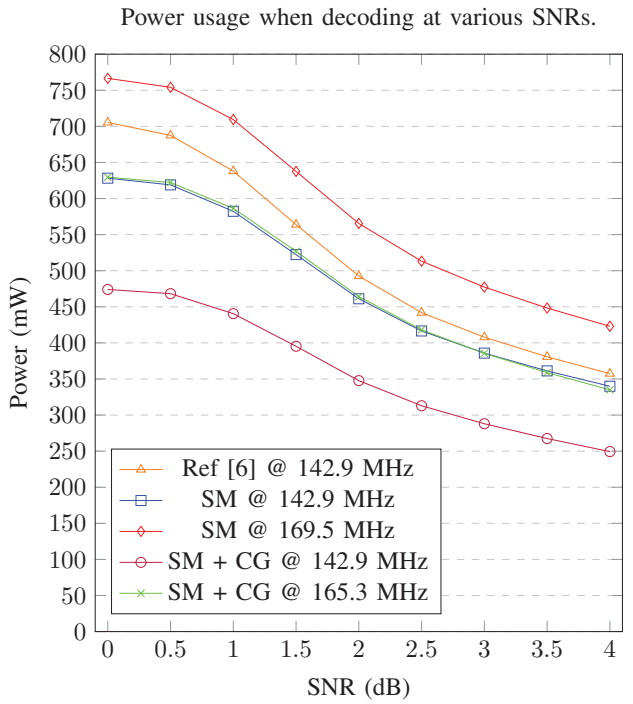
In this paper we have proposed two gate level optimizations for belief propagation decoders, namely an efficient, sign-magnitude adder that does not require conversion to and from two's complement, and clock gating based on the condition that all R inputs are zero. We have introduced two energy-efficient, radix-4 belief propagation polar code decoders where one uses our proposed adder and the other additionally uses clock gating.

Evaluation using post-layout simulation shows that our proposed decoder with clock gating decreases power usage up to 32.80% and is 5.2% smaller compared to a reference radix-4 belief propagation decoder at the same clock speed. And at the maximum clock speed which is 15.70% higher, it is up to 10.76% more power efficient and is 4.8% smaller.

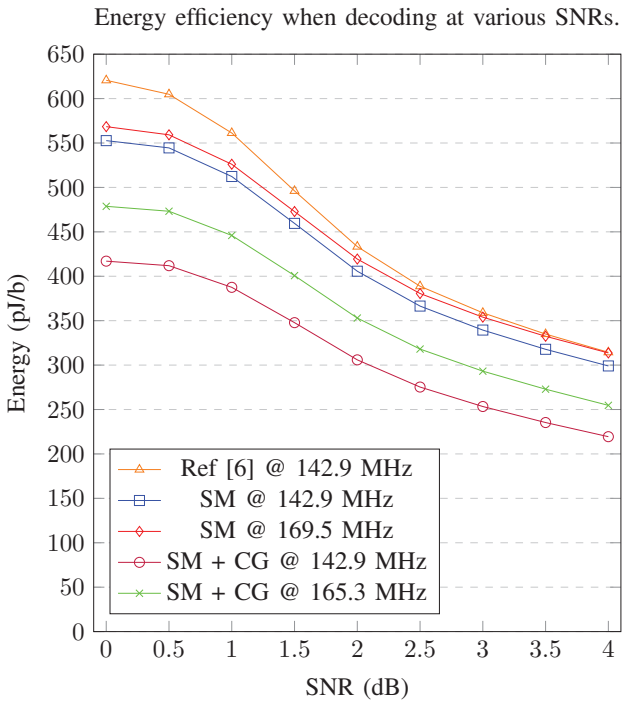
The results demonstrate that our proposed sign-magnitude adder is smaller and results in less switching activity compared to a typically used adder that converts between sign-magnitude and two's complement numbers. The results also show the benefit of clock gating based on the condition that all R input values being zero, as it leads to an even lower-switching activity without compromising power, area or speed compared to the state-of-the-art, stage-combined, radix-4 belief propagation polar code decoder.

Acknowledgment

This work is part of the research program Perspectief ZERO with project number P15-06 Project 3, which is (partly) financed by the Dutch Research Council (NWO).



(a) Power usage with N=1024, K=512, I_{max}=15 and 100 random code words per SNR.



(b) Energy efficiency with N=1024, K=512, I_{max}=15 and 100 random code words per SNR.

Fig. 10: Power and energy figures of the reference, *SM* and *SM+GC* implementations.

References

[1] R. Gallager. "Low-density parity-check codes". In: *IRE Trans. Inf. Theory* 8.1 (1962), pp. 21–28. DOI: 10.1109/TIT.1962.1057683.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima. "Near Shannon limit error-correcting coding and decoding: Turbo-codes". In: *IEEE Intl. Conf. Commun.* Vol. 2. 1993, 1064–1070 vol.2. DOI: 10.1109/ICC.1993.397441.

[3] E. Arıkan. "Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels". In: *IEEE Trans. Inf. Theory* 55.7 (2009), pp. 3051–3073. DOI: 10.1109/TIT.2009.2021379.

[4] E. Şaşoğlu, E. Telatar, and E. Arikan. "Polarization for arbitrary discrete memoryless channels". In: *IEEE Inf. Theory Workshop*. 2009, pp. 144–148. DOI: 10.1109/ITW.2009.5351487.

[5] V. Bioglio, C. Condo, and I. Land. "Design of Polar Codes in 5G New Radio". In: *IEEE Commun. Surveys Tutorials* 23.1 (2021), pp. 29–40. DOI: 10.1109/COMST.2020.2967127.

[6] J. Sha et al. "A memory efficient belief propagation decoder for polar codes". In: *China Commun.* 12.5 (2015), pp. 34–41. DOI: 10.1109/CC.2015.7112042.

[7] E. Arıkan. "Polar codes: A pipelined implementation". In: *Proc. 4th ISBC* (2010), pp. 11–14.

[8] J. Sha, J. Lin, and Z. Wang. "Stage-combined belief propagation decoding of polar codes". In: *IEEE ISCAS Symp.* 2016, pp. 421–424. DOI: 10.1109/ISCAS.2016.7527260.

[9] J. Sha et al. "A Stage-Combined Belief Propagation Decoder for Polar Codes". In: *J. Signal Process. Syst.* 90.5 (May 2018), pp. 687–694. ISSN: 1939-8018. DOI: 10.1007/s11265-016-1181-y.

[10] B. Yuan and K. K. Parhi. "Early Stopping Criteria for Energy-Efficient Low-Latency Belief-Propagation Polar Code Decoders". In: *IEEE Trans. Signal Process.* 62.24 (2014), pp. 6496–6506. DOI: 10.1109/TSP.2014.2366712.

[11] Y. S. Park et al. "A 4.68Gb/s belief propagation polar decoder with bit-splitting register file". In: *Symp. on VLSI Circuits Digest of Technical Papers*. 2014, pp. 1–2. DOI: 10.1109/VLSIC.2014.6858413.

[12] S. M. Abbas et al. "High-Throughput and Energy-Efficient Belief Propagation Polar Code Decoder". In: *IEEE Trans. VLSI Syst.* 25.3 (2017), pp. 1098–1111. DOI: 10.1109/TVLSI.2016.2620998.

[13] C. R. Kime and M. M. Mano. *Logic and computer design fundamentals*. Prentice Hall, 2003.

[14] A. B. Van Den Brink and M. J.G. Bekooij. "Computational Complexity Reduced Belief Propagation Algorithm for Polar Code Decoders". In: *APSIPA Conf.* 2021, pp. 318–323.