Hardware Realization of Polar Code Successive Cancellation List Decoder

Arunkumar Ravichandran
Department of Electrical
and Computer Engineering
UC San Diego
arravich@eng.ucsd.edu

Debosmit Majumder
Department of Electrical
and Computer Engineering
UC San Diego
dmajumde@eng.ucsd.edu

Abstract

Polar Codes are a family of codes which can achieve Shannon's capacity. While these codes have been theoretically proven to have higher capacity, the practical implementation suffers due to their lesser throughput compared to LDPC codes. This paper explores the design-space to modify the existing architectures of the Successive Cancellation List(SCL) decoder for Polar codes with an aim to improve the throughput and reduce resource utilization on hardware.

1. Introduction

Polar Codes, proposed by Arikan [2] are a major breakthrough in the area of coding theory. These are the first error-correcting codes mathematically proven to achieve Shannon's Channel capacity. They also have efficient encoding and decoding algorithms compared to the other family of codes. Recently, the 3GPP-RAN1-87 meeting approved that Polar Code coding scheme can be used in the 5G uplink control channel [9, 10] in place of turbo codes. Since Polar Codes are shown to be free of error floors when used with the binary-input symmetric memoryless channel, they are being used for wired communications[11] and flash storage systems [16] as well. The reduced complexity of polar code encoding and decoding schemes have made them an attractive choice.

Polar codes are linear block codes with a recursively constructed generator matrix. This recursive construction is carried out in a way that polarizes the probability of correctly estimating bits: some bit estimates become more reliable and others becomes less reliable. A vector containing the k information bits is first expanded into a vector of length N by inserting frozen bits at the appropriate frozen bit locations. Frozen bits are usually set to zero and their optimal locations depend on the channel type and condition.

In [2], Arikan used a Belief Propagation based decoder which even though had a higher throughput had less Bit error rate performance. Successive Cancellation based decoders were also proposed [1] which didn't achieve higher error correction performance. SC decoding schemes have a serial

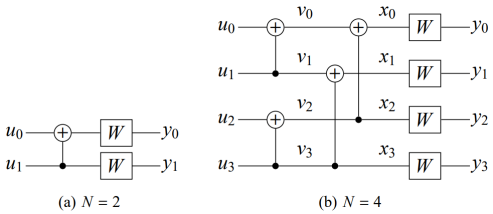

Figure 1. Polar Code Construction of Length N=2 and N=4

nature and hence their throughput is limited. To overcome the sequential nature of these SC decoding algorithms, parallel estimation of multiple bits was done. This improved the throughput of the SC based decoding algorithms [12, 6].

In [7], Tal presented a Successive Cancellation List decoding scheme which improved the error correction performance with a slight reduction in throughput. In this method CRC was added to the block to improve the error correction performance to that of the LDPC decoding scheme. This SCL decoding algorithm [22] is described in terms of likelihoods whose computation is unstable. In [15], LLR based SCL decoding was presented which had higher stability and simplifies sorting step in the SCL decoding scheme.

From the table summarized in [7, 21], we see that LLR-based decoding scheme achieves a higher throughput. By exploring parallelism on hardware, throughput performance can be improved.

It has been proposed in [8] that unrolling of the decoders would improve the throughput. Unrolling is a technique where each processing element can process a different received vector. So in our FPGA implementation we would be unrolling the LLR based SCL decoder.

We have implemented a LLR based SCL decoding scheme for Polar Codes for hardware. The implementation was done through design space exploration in the Vivado HLS tool to obtain a pareto optimal solution of the decoder.

The specific contributions of our paper include:
1. Hardware-friendly LLR based SCL decoding scheme was implemented.
2. Throughput was improved by exploring parallel computation of processing elements in the Vivado HLS tool.
3. The resource usage was reduced by optimizing the memory usage of different variables.

2. Related Work

In [19] Belief Propagation decoder was implemented in a FPGA. BP based decoders are less complex for a low SNR regime. For a higher SNR channel, the BER performance of BP based decoder is less. This BP architecture was compared to the results of a Turbo Code and were shown to perform better over that. Our work builds on the motivation to implement better and simpler decoding architectures suitable for hardware implementations and proposes scope for parallelism.

In [20] a modified SC based decoder was implemented in a FPGA and performance results were compared with the LDPC decoder. This Fast SC Polar decoder requires 5 times fewer LUTs compared to a LDPC decoder, but only achieves half of the throughput. Their decoding scheme was targeted towards Data storage systems and hence this architecture might not be suitable for a wireless communication system.

In [3] comparison of different decoding schemes on FPGA was done. Comparison was made between Polar decoders both in terms of error-correction performance and hardware efficiency against LDPC and Turbo decoders for existing communications standards like 802.11n, 802.11ad and 3GPP. Results of this paper suggest that the error-correction performance of BP and SC decoding are not powerful enough to match the error-correction performance of LDPC or Turbo codes. But more complex algorithms like SCL decoding scheme [22] can be used to achieve higher error-correction performance. Our paper focuses primarily on the SCL decoding scheme with an objective to improve its throuhgput on hardware.

[3] also suggested that increasing the throughput of SCL decoders as a promising research area because SCL decoding scheme occupies less area and offers high error-correction performance.

[14, 13, 18] discusses the hardware implementation of the SCL decoder. A log-likelihood ratio LLR based list decoding implementation is presented to reduce the complexity in [4, 21]. With a change in the SCL decoding scheme, multiple decisions can be made by the SCL decoder in one cycle improving the throughput performance as suggested in [5, 23].

3. Background

3.1. Polar Code Construction

To construct an (N,k) polar code, the k most reliable bits are used to carry the information bits. The rest of the bits are set to a frozen bit 0. This is called as channel polarization where the channels of a Binary Discrete Memoryless channel(B-DMC) is polarized into fully reliable and unreliable.

The polar code construction for 2 lengths is shown in the figure 1[2]. The encoder would decide the location of the reliable and frozen bits based on the channel condition as discussed in [22]. The systematic construction of the Polar code

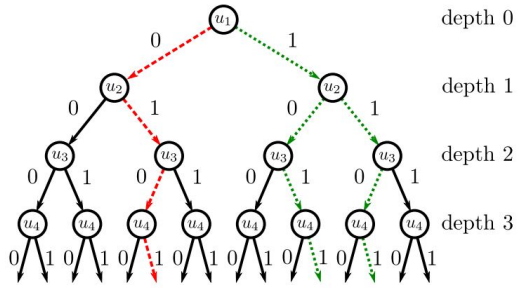

Figure 2. Decoding tree for N = 4 with a possible SC (dashed) and a possible list SC (dotted) decoding trajectories is shown

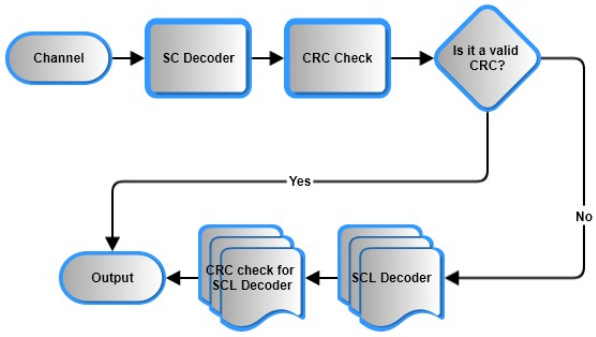

Figure 3. Architecture of the CRC based SCL decoder which we have implemented is shown

is a linear operation where XOR operations are performed on the bits in a recursive way. In a generic Polar Code N-bit encoder, there will be $(N/2)logN$ XOR operations. Time complexity of the encoding operation is $O(NlogN)$. Also an r-bit CRC code is concatenated at the last, so that the last r information bits are used to deliver the CRC checksums of the other K r information bits. This is done to improve the BER performance of the SCL Decoder[21].

3.2. Successive Cancellation List(SCL) Decoding of Polar Codes

The received bits can be realized in the form of a graph with depth N as shown in Figure 2[20]. SC decoding is a search problem trying to find the correct code-word which was sent by traversing the tree. Decisions are made to evaluate if the chosen path is correct or not. In Successive-Cancellation List decoding, the transmitted bits are evaluated using the received bits, y. In the decoder, Log-Likelihood Ratios (LLR) are computed which are helpful in the decision evaluation to find out whether a bit is a frozen bit or reliable bit. In a typical SC decoding, a single path will be evaluated to find the code-word. But in a List Decoding, L paths are decoded simultaneously so that the Maximum Likelihood performance is achieved. The time complexity of this SCL decoder is $O(LNlogN)$ and the space complexity is $O(LN)$. To improve the throughput of the algorithm, CRC is used to remove the least probable paths and allow a higher BER performance.

Input: received codeword, y_1^N
Input: code block length, N
Input: information set, $\mathcal{A}$
Input: frozen bit vector, $u_{\mathcal{A}^c}$
Input: maximum list size, L
Output: estimated information bits, $\hat{u}_{\mathcal{A}}$
Variable: j `//valid CRC vector of SCD`
Variable: k `//valid CRC vector of SCLD`
begin
  $\hat{u}_{\mathcal{A}} \longleftarrow$ **Successive Cancellation Decoding** $(y_1^N, N, \mathcal{A}, u_{\mathcal{A}^c})$
  $j \longleftarrow$ **Cyclic Redundancy Check Decoding** $(\hat{u}_{\mathcal{A}})$
  if j *is true* **then**
    return $\hat{u}_{\mathcal{A}}$
  else
    $\hat{u}_{L,\mathcal{A}} \longleftarrow$ **Successive Cancellation List Decoding** $(y_1^N, N, \mathcal{A}, u_{\mathcal{A}^c}, L)$
    for $l \leftarrow 1$ **to** L **do**
      $k \longleftarrow$ **Cyclic Redundancy Check Decoding** $(\hat{u}_{l,\mathcal{A}})$
      if k *is true* **then**
        $\hat{u}_{\mathcal{A}} \longleftarrow \hat{u}_{l,\mathcal{A}}$
        return $\hat{u}_{\mathcal{A}}$
    $\hat{u}_{\mathcal{A}} \longleftarrow \hat{u}_{1,\mathcal{A}}$
    return $\hat{u}_{\mathcal{A}}$

Figure 4. [2] Pseudocode for the SCL-CRC decoding algorithm is shown

A pseudo-code of the SCL decoder algorithm implemented is shown in Figure 4[21] and a schematic representation of the decoder architecture is shown in figure 3[21].

As shown in Figure 3, the codeword is estimated using the SC decoder and CRC comparison is done. If there is any CRC mismatch, then SCL algorithm with L paths are traversed to get possible codewords. If there is a valid CRC codeword, that is the correct codeword and output is produced. If there is no codeword with a valid CRC, then hard decision is made to output a codeword with higher LLR.

4. Implementation

In this section, we will discuss the design space exploration we did while implementing the SCL Polar Code Decoder in the HLS.

4.1. HLS: Components of the SCL Polar Code Decoder

The SCL decoder has the following elements:
1. Bitonic Sorter
2. Comparator
3. Memory copy decision and transfer
4. Processing Elements
5. CRC Generator
6. Feedback Logic
7. SortAndForward

4.1.1 Bitonic Sorter

To find the best decoding paths from a large probable paths at each depth in the tree, we need a sorter. The inputs to the sorter are the two LLs and the output is the index of the highest L. We use a bitonic sorter as suggested in [17]. Bitonic Sorter is a low-complexity semi-parallel sorting algorithm which is suitable for the FPGA implementation. As two LLs are provided to the BS core, a comparison is made at each level between the LLs. The resource usage of the Bitonic Sorter is shown in Table 1. We pipe-lined the Bitonic Sorter and found it to have a higher rate. Array reshaping was done to reduce the resource usage. The time complexity of the Bitonic Sorter is$O(log^2 L)$ and the space complexity of the Bitonic Sorter is $O(Llog^2 L)$ where L is the list size.

4.1.2 Comparator

Comparator was implemented as a separate core while doing the design space exploration. The inputs to the comparator were computed values from the Bitonic Sorter block comprising of data and index of the decoder tree. In the comparator, min and max data and indices will be filled and updated to the Bitonic Sorter. Thereby the BS can sort from the best list to the worst list. The resource usage for the Comparator is shown in Table 1.

4.1.3 Memory Copy Decision, and Memory Transfer

These are two separate blocks whose functionality is to assist the memory block. The inputs to the memory copy decision block is the memory block and the output is the index of the List. In memory transfer, the data is transferred from one memory block to another. Pipe-lining with array partitioning were performed to increase the throughput of the block. The resource usage of the regular and optimized version of these operations is shown in Table 1.

4.1.4 Processing Element

This is the integral part of the SCL decoder. PE implements the min-sum approximations and also the bottom channel splitting operation of the SCL decoder is done. The inputs to the PE block are two LLR values, partial sum decision and PE block outputs the intermediate LLR value with its precision. Pipe-lining of the PE was done to increase the throughput and memory partitioning was done to increase the number of operating interfaces when parallel operations were being done. The resource usage of the regular and optimized version of the Memory Block is shown in Table 1.

4.1.5 CRC Generator

In the CRC generator block, CRC of a decoded Codeword for a particular L is computed and compared to that of with the CRC value received from the channel. In the SC decoding main block, the CRC value of different lists would be compared with the CRC value available and atleast one list's CRC should be matching the CRC received.

4.1.6 Feedback Logic

Feedback Logic was having a less throughput because of large number of serial operations. To increase the through-

Table 1. Resource usage of individual components in the SCL Polar Code decoder

Functional Element	BRAM	DSP	FF	LUT	Clock Period(ns)	Interval	Throughput(MHz)
bitonic_sorter_unoptimized	0	0	228	643	8.05	22	5.65
bitonic_sorter_optimized	0	0	7	130	6.08	1	164.47
comparator_unoptimized	0	0	11	203	4.64	8	26.93
comparator_optimized	0	0	49	320	4.64	1	215.51
memory_copy_decision_unoptimized	0	0	83	307	8.47	2061	0.57
memory_copy_decision_optimized	0	0	0	78	6.15	1	162.6
memory_transfer_unoptimized	0	0	122	331	8.02	71	1.75
memory_transfer_optimized	0	0	0	78	6.15	1	162.6
processing_element_unoptimized	0	0	464	1672	8.59	5582	0.02
processing_element_optimized	0	0	2641706	784323	8.71	143	0.802
crc_unoptimized	0	0	104	439	6.51	16046	0.009
crc_optimized	0	0	8529	10432	8.22	1	121.65
feedback_logic_unoptimized	2	0	307	810	8.41	40224	0.0029
feedback_logic_optimized	0	0	1024	28733	8.38	2	59.66
sortAndForward_unoptimized	0	0	684	1779	8.05	4275	0.029
sortAndForward_optimized	0	0	6	95	6.51	2	76.8

Table 2. SCL Decoder Resource usage and Throughput

Functional Element	BRAM	DSP	FF	LUT	Clock Period(ns)	Interval	Throughput(Hz)
list_scd_unoptimized	7	0	1916	6831	8.59	25237019	4.61
list_scd_optimized	3	0	2373843	921229	8.71	15409691	7.45

put, we introduced parallelism by using the pipeline pragma in the HLS tool and to reduce the BRAM resource usage array reshaping was done.

4.1.7 SortAndForward

SortAndForward block takes the two LLs. Sorting operation is done by the Bitonic Sorter and after sorting transfer of memory is done from one memory block to another. Decision making is done by the memory copy decision block. We optimized this by pipelining the design and reshaping the arrays into smaller arrays. This reduced the BRAM usage because it concatenates the data into one element while maintaining parallel data access like array partitioning.

4.2. Design space exploration of the SCL decoder

Synthesis of the RTL in the Vivado HLS is a time consuming process if the code is complex. Without any optimization, the resource usage was found to be convenient for hardware implementation. Our hardware-friendly architecture of the SCL decoder kept the resource usage in check despite having less impact on the throughput. Further optimization was tried to improve the throughput by exploring parallelism and this increased the resource usage many-fold. Section 4.1 summarized parallelism among individuals components of the design. Since, the decoder design involves multiple iterations on the input data, loops are predominant parts of the code. Unrolling was performed on the smaller loops along with array partitioning. Bit-width optimization were also performed changing data types into fixed point formats, thus reducing computational complexity. However, it was observed that the top module with all terminal blocks optimized, was exceeding the resource usage and taking time to synthesize. Table 2 shows the throughput and resource usage of the optimized and the original version of the top module.

4.2.1 Testbench

We give the pre-computed LL0 and LL1 as input to the SCL decoder. The block length is 256 and List Size is 2. We have a maximum LL value as 2047 with a precision of 11. We append a 16 bit CRC at the end of the 256 bit block. This CRC is to increase the BER performance. The output from the SCL decoder is the information bits sent by the sender before modulation and encoding. This is compared with the golden output to identify if there are any bit errors.

5. Results

5.1. Results in Software

We calculated the execution time in the software with both the MATLAB code and C++ code. The codes were executed in Intel i7-7700 processor. In this section, for various values of List Size and CRC values we measured the time taken to decode.

MATLAB

We noticed that as we increased the List Size or the size of the CRC bits appended to the Info bits, we found the BER performance to increased. However the execution time was also linearly increasing causing a loss in throughput. The results are presented in Table 2.

Table 3. Execution Time of MATLAB code for different List Size, Block Length and CRC Size

Block Length	*List Size*	*CRC Size*	*Execution Time(s)*
256	1	0	0.59
256	2	16	1.55
1024	4	32	2.5

Table 4. Execution Time of C++ code for different List Size, Block Length and CRC Size

Block Length	*List Size*	*CRC Size*	*Execution Time(s)*
256	1	0	0.45
256	2	16	1.434
1024	4	32	2.25

C++

We found the C++ program to have a higher execution time/higher throughput compared to the MATLAB program. This is because the execution is done in faster way by removing all the wrapper functions provided by the MATLAB. The results are presented in Table 3.

5.2. Results in HLS

In the Vivado HLS tool, we calculated the time taken for the decoder to output the value. The BER performance was similar to the values which we observed in the software and hence we didn't measure that as a factor. However, a design trade-off could be done to achieve higher throughput by sacrificing the BER performance. The resource usage and throughput results of individual blocks and overall SCL decoder block are presented in the Table 1 and Table 2 respectively.

6. Future Work

Implementation of the SCL decoder in the hardware is to be done to corroborate the results obtained from the HLS tool. These are the design considerations which we should keep in mind while implementing the SCL decoder in a specific FPGA.

1. The resource usage shouldn't exceed the available resource area present in the FPGA board. As we see from that Table 1 and Table 2, there are certain elements which when optimized for throughput has a very high resource usage. So a typical FPGA SoC like Zynq would not fit the SCL decoder netlist. However higher end USRPs which has higher area can fit these SCL decoder and we would like to implement the SCL decoder and check the throughput performance.
2. Different SCL decoding architectures can be explored on the hardware. Polar Code decoders are still in their developmental stage and requires improvements in both logical/mathematical aspect and also in the hardware implementation aspect. By implementing different architectures and optimizing them for performance, and also exploring different algorithms for basic operations like sorting, comparing which are less resource utilizing.
3. Data transfer between the host and the FPGA should be limited as the data transfer via the AXI bus is a costly process.

Acknowledgement

I would like to thank Prof. Ryan Kastner and the TAs of the course CSE 237C who provided valuable suggestions while working on this project.

References

[1] A. Alamdar-Yazdi and F. R. Kschischang. A simplified successive-cancellation decoder for polar codes. *IEEE communications letters*, 15(12):1378–1380, 2011.

[2] E. Arikan. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on Information Theory*, 55(7):3051–3073, 2009.

[3] A. Balatsoukas-Stimming, P. Giard, and A. Burg. Comparison of polar decoders with existing low-density parity-check and turbo decoders. In *Wireless Communications and Networking Conference Workshops (WCNCW), 2017 IEEE*, pages 1–6. IEEE, 2017.

[4] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg. Llr-based successive cancellation list decoding of polar codes. *IEEE transactions on signal processing*, 63(19):5165–5179, 2015.

[5] A. Balatsoukas-Stimming, A. Raymond, W. Gross, and A. Burg. Hardware architecture for list sc decoding of polar codes. *arXiv preprint arXiv:1303.7127*, 2013.

[6] K. Chen, K. Niu, and J. Lin. Improved successive cancellation decoding of polar codes. *IEEE Transactions on Communications*, 61(8):3100–3107, 2013.

[7] P. Giard, G. Sarkis, A. Balatsoukas-Stimming, Y. Fan, C.-y. Tsui, A. Burg, C. Thibeault, and W. J. Gross. Hardware decoders for polar codes: An overview. In *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*, pages 149–152. Ieee, 2016.

[8] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross. 237 gbit/s unrolled hardware polar decoder. *Electronics Letters*, 51(10):762–763, 2015.

[9] Huawei and HiSilicon. Details of the polar code design. *3GPP TSG RAN WG1 Meeting87*, 2016.

[10] Q. Incorporated. Ldpc rate and compatible design overview. *3GPP TSG-RAN WG1 86bis*, 2016.

[11] S. B. Korada. Polar codes for channel and source coding. 2009.

[12] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross. A semi-parallel successive-cancellation decoder for polar codes. *IEEE Transactions on Signal Processing*, 61(2):289–299, 2013.

[13] C. Leroux, A. J. Raymond, G. Sarkis, I. Tal, A. Vardy, and W. J. Gross. Hardware implementation of successive-cancellation decoders for polar codes. *Journal of Signal Processing Systems*, 69(3):305–315, 2012.

[14] 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*, pages 1665–1668. IEEE, 2011.

[15] B. Li, H. Shen, and D. Tse. An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check. *IEEE Communications Letters*, 16(12):2044–2047, 2012.

[16] Y. Li, H. Alhussien, E. F. Haratsch, and A. A. Jiang. The performance of polar codes for multi-level flash memories. *Polar*, 4:3, 2014.

[17] J. Lin and Z. Yan. An efficient list decoder architecture for polar codes. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(11):2508–2518, 2015.

[18] 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*, pages 205–208. IEEE, 2012.

[19] A. Pamuk. An fpga implementation architecture for decoding of polar codes. In *Wireless Communication Systems (ISWCS), 2011 8th International Symposium on*, pages 437–441. IEEE, 2011.

[20] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross. Fast polar decoders: Algorithm and implementation. *IEEE Journal on Selected Areas in Communications*, 32(5):946–957, 2014.

[21] A. Süral. *An fpga implementation of successive cancellation list decoding for polar codes*. PhD thesis, bilkent university, 2016.

[22] I. Tal and A. Vardy. List decoding of polar codes. *IEEE Transactions on Information Theory*, 61(5):2213–2226, 2015.

[23] B. Yuan and K. K. Parhi. Low-latency successive-cancellation polar decoder architectures using 2-bit decoding. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(4):1241–1254, 2014.