Chapter 2
Fast Low-Complexity Hardware Decoders for Low-Rate Polar Codes

Abstract In this chapter, we show how the state-of-the-art low-complexity decoding algorithm can be improved to better accommodate low-rate codes. More constituent codes are recognized in the updated algorithm and dedicated hardware is added to efficiently decode these new constituent codes. We also alter the polar code construction to further decrease the latency and increase the throughput with little to no noticeable effect on error-correction performance. Rate-flexible decoders for polar codes of length 1024 and 2048 are implemented on FPGA and ASIC. Over the previous FPGA work, they are shown to have from 22 to 28% lower latency and 26 to 34% greater throughput when decoding low-rate codes. On 65 nm ASIC CMOS technology, the proposed decoder for a $(1024, 512)$ polar code is shown to compare favorably against the state-of-the-art ASIC decoders. With a clock frequency of 400 MHz and a supply voltage of 0.8 V, it has a latency of 0.41 µs and an area efficiency of 1.8 Gbps/mm^2 for an energy efficiency of 77 pJ/info. bit. At 600 MHz with a supply of 1 V, the latency is reduced to 0.27 µs and the area efficiency increased to 2.7 Gbps/mm^2 at 115 pJ/info. bit.

2.1 Introduction

While the Fast-SSC [55] algorithm represents a significant improvement over the previous decoding algorithms, the work in [55] and the optimization presented therein targeted high-rate codes. In this chapter, we propose modifications of the Fast-SSC algorithm and a code construction alteration process targeting low-rate codes. We present results using the proposed methods, algorithms and implementation. These results show a 22–28% latency reduction and a 22–28% throughput improvement with little to negligible coding loss for low-rate moderate-length polar codes.

The rest of this chapter is organized as follows. Section 2.2 discusses polar code construction alteration along with our proposed method leading to improved latency and throughput of a hardware decoder. In Sect. 2.3, modifications to the original Fast-SSC algorithms are proposed in order to further reduce the latency and increase the decoding throughput. Sections 2.4 and 2.5 present the implementation details

P. Giard et al., *High-Speed Decoders for Polar Codes*,
DOI 10.1007/978-3-319-59782-9_2

along with the detailed results on FPGA. Section 2.5 also provides ASIC results for our proposed decoder decoding a $(1024, 512)$ polar code for a comparison against state-of-the-art ASIC decoders from the literature. Finally, Sect. 2.6 concludes this chapter.

2.2 Altering the Code Construction

2.2.1 Original Construction

As mentioned in Chap. 1, a good polar code is constructed by selecting which bits to freeze, according to the type of channel and its conditions [7, 44, 61, 65]. Figure 2.1 shows the decoder tree corresponding to the $(1024, 512)$ polar code constructed using the technique of [61] where only the node types defined in Table 2.1 are used with the same constraints of [55]. The polar code was optimized for an E_b/N_0 of 2.5 dB. The *F*, *G*, *G_0R*, *Combine* and *Combine_0R* blocks are constrained to a maximum of $P = 512$ inputs meaning that, for nodes with a length $N_v > P$, $\lceil N_v/P \rceil$ cycles are required. The *Rep*, *RepSPC* and 01 blocks are all executed in one clock cycle. Finally, the SPC-based nodes—0SPC and RSPC—use pipelining and require $\lceil N_v/P \rceil + 4$ clock cycles. Thus the decoding latency to decode the tree of Fig. 2.1 using the algorithm and implementation of [55] is 220 Clock Cycles (CCs) and the information throughput is 2.33 bits/CC.

Altering a polar code to further trim the decoder tree can result in a significant latency reduction, without affecting the code rate. By making these modifications however, the error-correction performance is degraded. Although, as will be shown in the next section, the impact can be small, especially if the number of changes is limited.

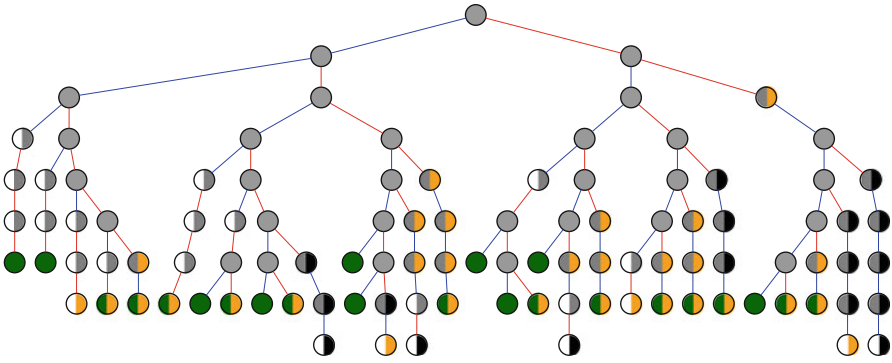

Fig. 2.1 Decoder tree for the $(1024, 512)$ polar code built using [61] and decoded with the nodes and operations of Table 2.1

Table 2.1 Decoder tree node types supported by the original Fast-SSC polar decoder [55]

Name	Color	Description
0R	White and gray	Left-half side is frozen
R1	Gray and black	Right-half side is all information
RSPC	Gray and yellow	Right-half side is an SPC code
0SPC	White and yellow	Left-half side is frozen, right-half side is an SPC code
Rep	Green	Repetition code, maximum length N_v of 16
RepSPC	Green and yellow	Concatenation of a repetition code on the left and an SPC code on the right, $N_v = 8$
01	Black and white	Fixed-length pattern $N_v = 4$ where the left-half side is frozen and the right-half side is all information
Rate-R	Gray	Mixed-rate node

2.2.2 Altered Polar Code Construction

In all SC-based decoders, the size of the decoder tree depends on the distribution of frozen and information bit locations in the code. Arıkan's original polar code construction only focuses on maximizing the reliability of the information bits. Several altered polar-like code constructions have been proposed in the literature [10, 25, 74] and their objective is to trade off error-correction performance for decoding complexity reduction by slightly changing the set of information bits, while keeping the code rate fixed. The main idea behind all the altered code constructions is to exchange the locations of a few frozen bits and information bits in order to get more bit patterns that are favorable in terms of decoding latency. In all cases, care must be taken in order to avoid using bit locations that are highly unreliable to transmit information bits.

The method in [25] first defines a small set of bit locations which contains the $n_s - h$ least reliable information bit locations along with the h most reliable frozen bit locations. Then, in order to keep the rate fixed, it performs an exhaustive search over all $\binom{n_s}{h}$ possible combinations of the n_s elements containing exactly h frozen bit locations and selects the combination that leads to the smallest decoding latency. In [10], the altered construction problem is formalized as a binary integer linear program. Consequently, it is shown that finding the polar code with the lowest decoding complexity under an error-correction performance constraint is an NP-hard problem. For this reason, a greedy approximation algorithm is presented which provides reasonable results at low complexity even for large code lengths. A similar greedy algorithm is presented in [74] for polar codes with more general code lengths of the form $N = l^n$, $l \geq 2$.

2.2.3 Proposed Altered Construction

The methods of [10, 74] only considered rate-0 and rate-1 nodes. As such, the results can not be directly applied to Fast-SSC decoding, where several additional types of special nodes exist. For this reason, in this work we follow the more general exhaustive search method of [25], augmented with a human-guided approach.

More specifically, bit-state alterations that would lead to smaller latency are identified by visual inspection of the decoder tree for the unaltered polar code. This list of bit locations is then passed to a program to be added to the bit locations considered by the technique described in [25]. Hence, two lists are composed: one that contains frozen bit locations proposed by the user as well as locations that were almost reliable enough to be used to carry information bits, and one that contains the information bit locations proposed by the user and the locations that barely made it into information bit locations.

The code alteration algorithm then proceeds by gradually calculating the decoding latency for all possible bit swap combinations. A constrained-size and ordered list of the combinations with the lowest decoding latency is kept. Once that list needs to be trimmed, only one entry per latency value is kept by simulating the error-correction performance of the altered code at an E_b/N_0 value of interest. The entry with the best FER is kept and the others with the same latency are removed from the list. That list containing the best candidates is further trimmed by removing all candidates that feature both a greater latency and worse error-correction performance compared to those of their predecessor. Similarly to the technique of [25], our proposed technique does not alter the code rate as the total number of information and frozen bits remains the same.

2.2.3.1 Human-Guided Criteria

The suggested bits to swap are selected to improve the latency and throughput. Thus, these bit swaps must eliminate constituent codes for which we do not have an efficient decoding algorithm and create ones for which we do. We classify the selection criteria under two categories: the bit swaps that transform frozen bit locations into information bit locations and bit swaps that do the opposite. The former increase the coding rate while the latter reduce it.

In addition to the node type definitions of Table 2.1, the below descriptions of criteria use the following types of subtrees or nodes:

- R1-01: subtree rooted in a R1 node with a 01 leaf node, may contain a chain of R1 nodes
- Rep1: subtree rooted in a R1 node with a leaf Rep node; in Sect. 2.3, that subtree is made into a node where the left-half side is a repetition code and the right-half side is all information
- R1-RepSPC: subtree rooted in a R1 node with a RepSPC leaf node, may contain a chain of R1 nodes

- Rep-Rep1: subtree where the rate-*R* node has a left-hand-side and right-hand-side nodes are Rep and Rep1 nodes, respectively
- 0-RepSPC: subtree rooted in a 0R node with a leaf RepSPC node; in Sect. 2.3, that subtree is made into a node where the left-half side is frozen and the right-half side is a RepSPC node

Dedicated hardware to efficiently decode Rep1 and 0RepSPC nodes are presented in Sect. 2.3.

From Frozen to Information Locations

1. Unfreezing the second bit of a 01 node that is part of a R1-01 subtree creates an RSPC node.
2. Changing an RepSPC into an RSPC node by adding the second, third and fifth bit locations.
3. Changing a RSPC node into a R1 node by changing the SPC code into a rate-1 code.

Criterion 1 is especially beneficial where the R1-01 subtree contains a chain of R1 nodes, e.g., Pattern 5 in Fig. 2.2. Similarly, Criterion 2 has a significant impact on R1-RepSPC subtrees containing a chain of R1 nodes, e.g., Pattern 3 in Fig. 2.2.

From Information to Frozen Locations

4. Changing a 0R-01 subtree into a Repetition node.
5. Freezing the only information bit location of a Rep node to change it into a rate-0 code.
6. A specialization of the above, changing a Rep-RepSPC subtree into a 0-RepSPC subtree by changing the left-hand-side Rep node into a rate-0 node.
7. Transforming a Rep-Rep1 subtree into a 0-RepSPC subtree by changing the left-hand-side Repetition code into a rate-0 code and by freezing the fifth bit location of the Rep1 subtree to change the rate-1 code into an SPC code.

Consider the decoder tree for a $(512, 376)$ polar code as illustrated in Fig. 2.2a, where some frozen bit patterns are circled in blue and numbered for reference. Its implementation results in a decoding latency of 106 clock cycles. That latency can be significantly reduced by freezing information bit locations or by transforming previously frozen locations into information bits.

Notably, five of the bit-swapping criteria—leading to latency reduction—described above are illustrated in Fig. 2.2a. The patterns numbered 1 and 2 are repetition nodes meeting the fourth criterion. Changing both into rate-0 nodes introduces two new 0R nodes. The patterns 3–6 are illustrations of the fourth, second, sixth and first criteria, respectively.

Figure 2.2b shows the resulting decoder tree after the alterations were made. The latency has been reduced from 106 to 82 clock cycles.

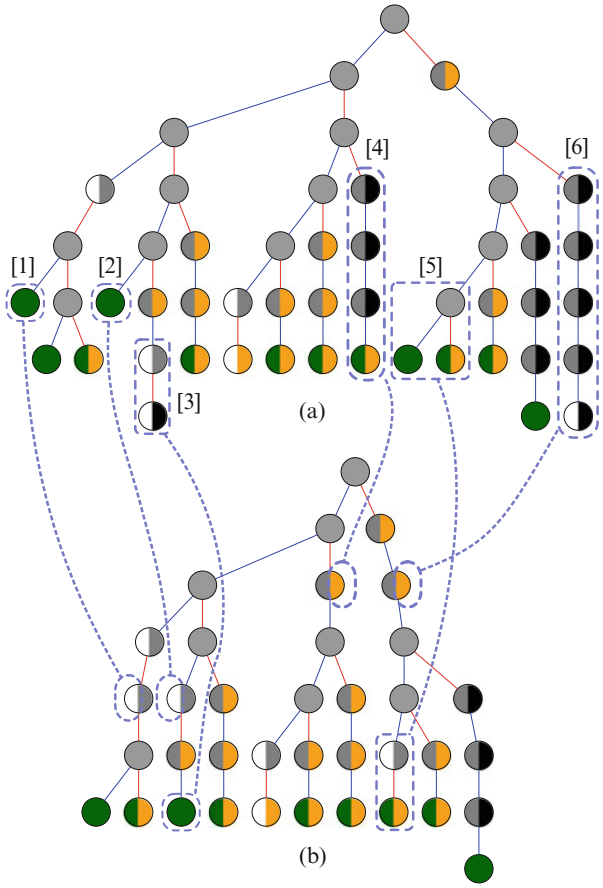

Fig. 2.2 Decoder trees for two different $(512, 376)$ polar codes, where (**a**) and (**b**) are before and after construction alteration, respectively

2.2.3.2 Example Results

Applying our proposed altered construction method, we were able to decrease the decoding latency of the $(1024, 512)$ polar code illustrated in Fig. 2.1 from 220 to 189 clock cycles, a 14% improvement, with 5 bit swaps. That increases the information throughput to 2.71 bits/CC, up from 2.33 bits/CC. The corresponding decoder tree is shown in Fig. 2.3.

The error-correction performance of the $(1024, 512)$ altered code is degraded as illustrated by the markerless black curves in Fig. 2.4. The loss amounts to less than 0.25 dB at a FER of 10^{-4}. For wireless applications, which are usually the target for codes of such lengths and rates, this represents the FER range of interest.

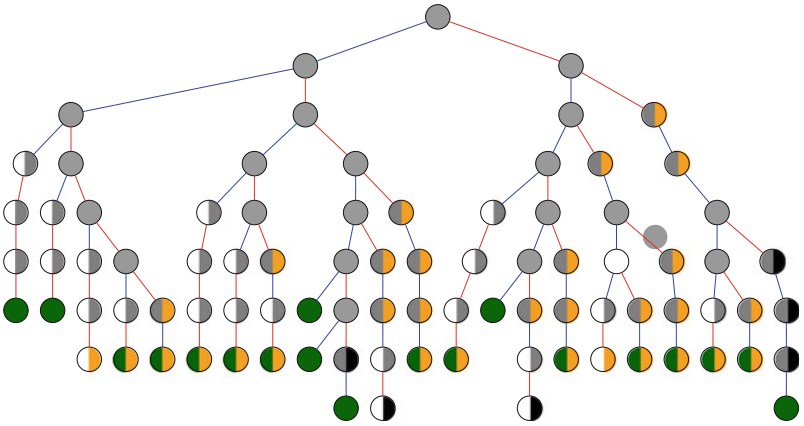

Fig. 2.3 Decoder tree for the altered $(1024, 512)$ polar code

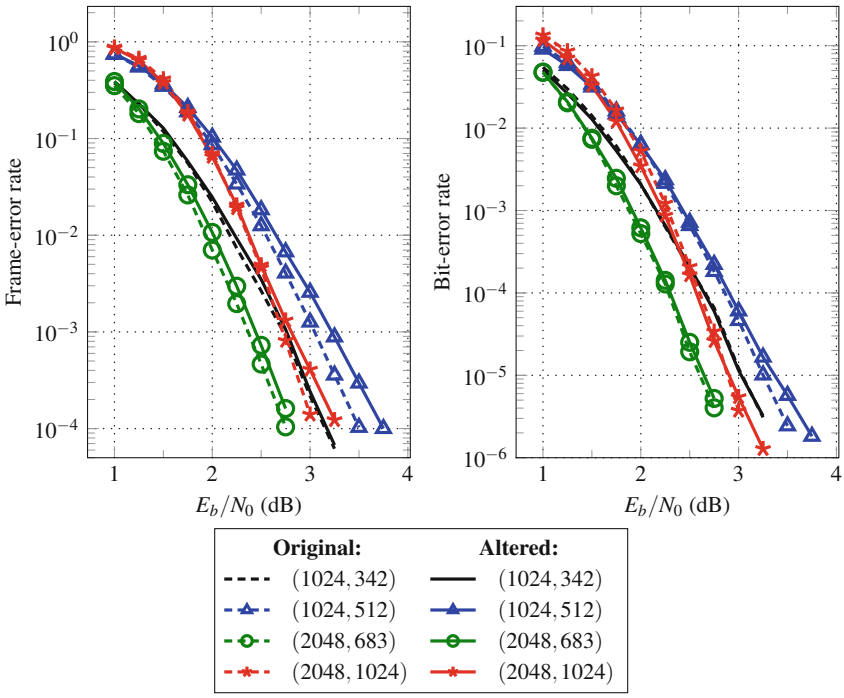

Fig. 2.4 Error-correction performance using BPSK over an AWGN channel of the altered codes compared to that of the original codes constructed using the Tal and Vardy method [61]

Figure 2.4 also shows the error-correction performance of three other polar codes altered using our proposed method. In the case of these other codes, the alterations have a negligible effect on error-correction performance.

2.3 New Constituent Decoders

Looking at the decoder tree of Fig. 2.3, it can be seen that some frozen bit patterns occur often. Adding support for more constituent codes to the Fast-SSC algorithm will result in a reduced latency and increased throughput under the constraint that the corresponding computation nodes do not significantly lengthens the critical path of a hardware implementation. As a result of an investigation, the constituent codes of Table 2.2 were added. Furthermore, post-place and route timing analysis showed that the maximum length N_v of a Repetition node could be increased from 16 to 32 without affecting the critical path.

The new decoder tree shown in Fig. 2.5 has a decoding latency of 165 clock cycles, a 13% reduction over the decoder tree of Fig. 2.3 decoded with the original Fast-SSC algorithm. Thus, the information throughput of that polar code has been improved to 3.103 bits/CC.

Table 2.2 New functions performed by the proposed decoder

Name	Color	Description
Rep1	Green and black	Repetition code on the left, Rate-1 code on the right, maximum length N_v of 8
0RepSPC	White and lilac	Rate-0 code on the left, RepSPC code on the right, $N_v = 16$
001	$\frac{3}{4}$ white and $\frac{1}{4}$ black	Rate-0 code on the left, 01 code on the right, $N_v = 8$

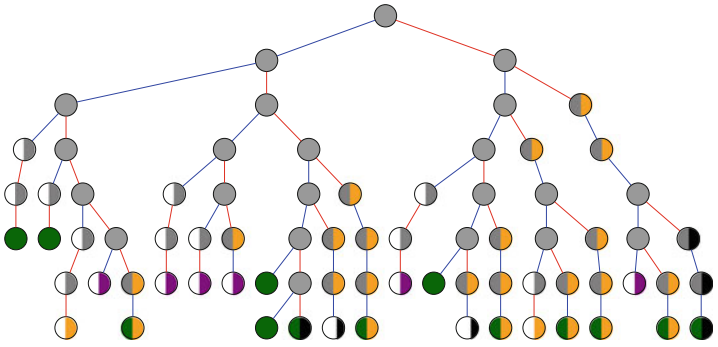

Fig. 2.5 Decoder tree for the altered polar code with the added nodes

Table 2.3 Frozen bit patterns decoded by leaf nodes

Name	Pattern
Rep	0001
	0000 0001
	0000 0000 0000 0001
	0000 0000 0000 0000 0000 0000 0000 0001
Rep1	0001 1111
0SPC	0000 0111
RepSPC	0001 0111
0RepSPC	0000 0000 0001 0111
01	0011
001	0000 0011

To summarize, Table 2.3 lists the frozen bit patterns that can be decoded by leaf nodes. It can be seen that the smallest possible leaf node has length $N_v = 4$ while our proposed decoder tree shown in Fig. 2.5 has a minimum length $N_v = 8$. In other words, Fig. 2.5 is representative of the patterns listed in Table 2.3 but not comprehensive.

2.4 Implementation

2.4.1 Quantization

Let Q_i be the total number of bits used to represent LLRs internally, Q_c be the total number of bits to represent channel LLRs, and Q_f be the number of bits among Q_i or Q_c used to represent the fractional part of any LLR. It was found through simulations that using $Q_i.Q_c.Q_f = 6.5.1$ quantization led to an error-correction performance very close to that of the floating-point number representation as can be seen in Fig. 2.6.

2.4.2 Rep1 Node

The Rep1 node decodes Rep1 codes—the concatenation of a repetition code and a rate-1 code—of length $N_v = 8$. Its bit-estimate vector β_0^7 is calculated using operations described in the previous sections. However, instead of performing the required operations sequentially, the dedicated hardware preemptively calculates intermediate soft values.

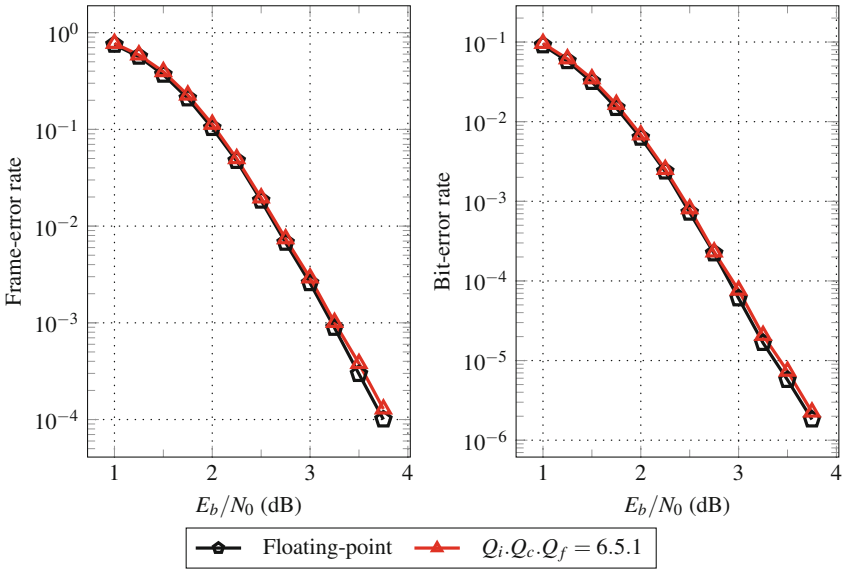

Fig. 2.6 Impact of quantization on the error-correction performance of the proposed (1024, 512) polar code

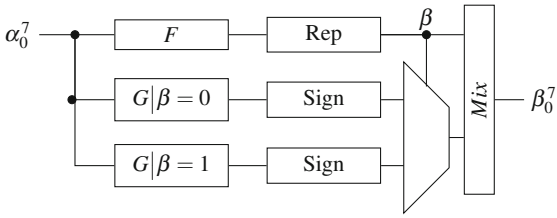

Fig. 2.7 Architecture of the Rep1 Node

Figure 2.7 shows the architecture of the Rep1 node. It can be seen that there are two G blocks. One preemptively calculates soft values assuming that the Rep block will output $\beta = 0$ and the other for $\beta = 1$. The Rep block provides a single bit estimate corresponding to the information bit the repetition code of length $N_v = 4$ it is decoding. The outputs of the G blocks go through a Sign block to generate hard decisions. The correct hard decision vector is then selected using the output of the Rep block. Finally, the bit estimate vector β_0^7 is built. The highest part, β_4^7, is always comprised of the multiplexer output. The lowest part, β_0^3, is either a copy of same output or its binary negation. The negated version is selected when the output of the Rep block is 1.

Calculations are carried out in one clock cycle. The output of the F, G and Rep blocks are not stored in memory. Only the final result, the bit-estimate vector β_0^7, is stored in memory.

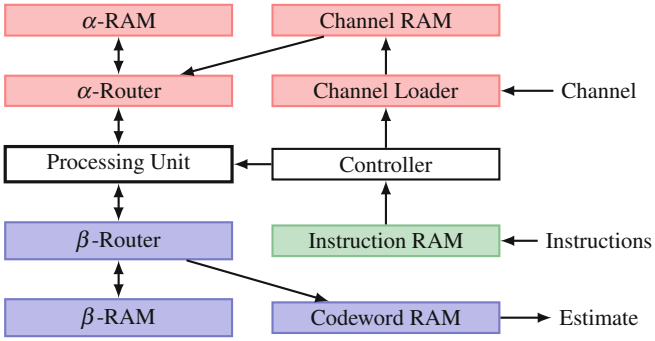

Fig. 2.8 High-level architecture of the decoder

2.4.3 High-Level Architecture

The high-level architecture of the decoder is presented in Fig. 2.8. Instructions representing the polar decoding operations to be performed are loaded before decoding starts. When the decoder is started, the controller signals the channel loader to start storing channel LLRs, 32 LLRs (160 bits) per clock cycle, into the channel RAM. The controller then starts to execute functions on the processing unit. The processing unit reads LLRs from the Channel or α-RAM and writes LLRs to the α-RAM. It reads or writes hard decisions to the β-RAM. The last *Combine* operation writes the estimated codeword into the Codeword RAM, a memory accessible from outside the decoder.

The decoder is complete with all input and output buffers to accommodate loading a new frame and reading an estimated codeword while a frame is being decoded. The required memory could be made smaller if the nominal throughput required is lower. The loading or outputting of a full frame takes fewer clock cycles than the actual decoding, we have a pipelined operation; under normal operation, the decoder should not be slowed down by the Input/Output (I/O) operations.

2.4.4 Processing Unit or Processor

The core of the decoder is the processing unit illustrated in Fig. 2.9 and based on the Fast-SSC implementation of [55]. Thus, the processing unit features all the modules required to implement the nodes and operations listed in Table 2.1 and described in Sect. 2.3. Notably, the 01 and RepSPC blocks connected to the G block implement the 001 and 0RepSPC nodes, respectively, where the all-zero vector input is selected at the multiplexer m_0. The critical path of the decoder corresponds to the 0RepSPC node i.e. goes through G, RepSPC, the multiplexer m_3, *Combine* and the multiplexer m_2. It is slightly longer than that of [55].

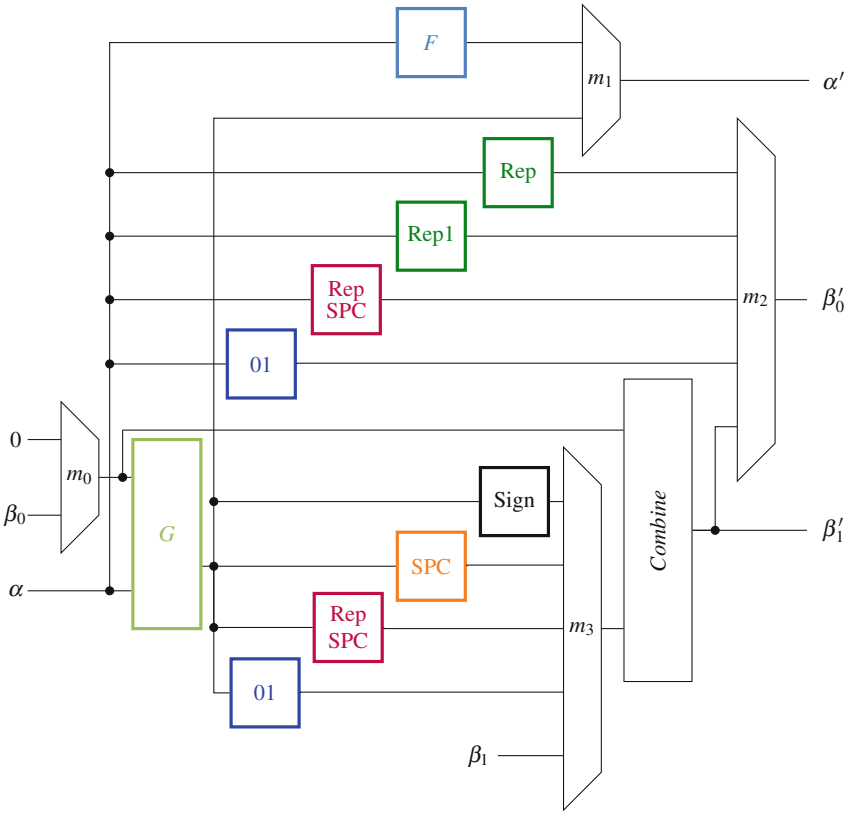

Fig. 2.9 Architecture of the processing unit

2.5 Results

2.5.1 Verification Methodology

A software model was used to generate random codewords for transmission using Binary Phase-Shift Keying (BPSK) over an Additive White Gaussian-Noise (AWGN) channel. The functionality of the designs was verified both at the Register-Transfer Level (RTL) and at the post-place and route level through simulations. Finally, the same frames were also decoded on an FPGA using an FPGA-in-the-loop setup. For all E_b/N_0 values, a minimum of 100 frames in errors were simulated.

2.5.2 Comparison with State-of-the-Art Decoders

In this section, post-fitting results are presented for the Altera Stratix IV EP4SGX530KH40C2 FPGA. All results are worst-case using the slow 900 mV 85 °C timing model. Table 2.4 shows the results for two rate-flexible implementations for polar codes of length 1024 and 2048, respectively. The decoder of [48] is also included for comparison.

Looking at the results for our proposed decoders, it can be observed that the number of Look-Up Tables (LUTs) and registers required are very similar for both code lengths. However, the RAM usage differs significantly where decoding a longer code requires more memory as expected. Timing reports show that the critical path corresponds to the 0RepSPC node.

Table 2.4 also compares the proposed decoders against the decoder of [48] as well as the original Fast-SSC implementation [55]. The latter was resynthesized so that the decoder only has to accommodate polar codes of length $N = 1024$ or $N = 2048$ and is marked with an asterisk (*) in Table 2.4.

Our work requires at most a 1.4% increase in used LUTs compared to [55]. The difference in registers can be mostly attributed to register duplication, a measure taken by the fitter to shorten the critical path to meet the requested clock frequency. The Static Random-Access Memory (SRAM) usage was also increased by 2.3%.

Table 2.5 shows the latency and information throughput of the decoders of Table 2.4 when decoding low-rate moderate-length polar codes. It also shows the effect of using a polar codes with altered constructions—as described in Sect. 2.2—with all Fast-SSC-based decoders. For both [55]* and our work, the results listed as 'altered codes' have the same resource usage and clock frequency as listed in Table 2.4 since these decoders can decode any polar code of length $N = 1024$ or $N = 2048$ by changing the code description in memory.

Applying the proposed altered construction alone, Table 2.5 shows that decoding these altered codes with the original decoders of [55] results in a 14–21% latency reduction and a 16–27% throughput improvement. From the same table, it can be seen that decoding the unaltered codes with the updated hardware decoder integrating the proposed new constituent decoders, the latency is reduced by 4–10% and the throughput is improved by 4–10%.

Table 2.4 Post-fitting results for rate-flexible decoders for moderate-length polar codes

Implementation	N	LUTs	Regs.	RAM (kbits)	f (MHz)
[48]	1024	1940	748	7.1	239
[55]*	1024	23 020	1024	42.8	103
	2048	23 319	5923	60.9	103
this work	1024	23 353	5814	43.8	103
	2048	23 331	5923	61.2	103

Table 2.5 Latency and information throughput comparison for low-rate moderate-length polar codes

Implementation	Code (N, k)	Latency		Info. T/P (Mbps)
		(CCs)	(μs)	
[48]	(1024, 342)	2185	9.14	37
	(1024, 512)	2185	9.14	56
[55]*	(1024, 342)	201	1.95	175
	(1024, 512)	220	2.14	240
	(2048, 683)	366	3.55	192
	(2048, 1024)	389	3.78	271
Altered codes	(1024, 342)	173	1.68	204
	(1024, 512)	186	1.81	284
	(2048, 683)	289	2.81	243
	(2048, 1024)	336	3.26	314
This work	(1024, 342)	193	1.87	183
	(1024, 512)	204	1.98	259
	(2048, 683)	334	3.24	211
	(2048, 1024)	367	3.56	287
Altered codes	(1024, 342)	157	1.52	224
	(1024, 512)	165	1.60	320
	(2048, 683)	274	2.66	257
	(2048, 1024)	308	2.99	342

Combining the contribution of both the altered construction method and the new dedicated constituent decoders, the proposed work achieves the best latency among all compared decoders. For the polar codes of length $N = 1024$, the throughput is 5.7–6.1 times greater than that of the two-phase decoder of [48]. Finally, the latency is reduced by 22–28% and the throughput is increased by 26–34% over the Fast-SSC decoders of [55].

Table 2.6 presents a comparison of this work against the state-of-the-art ASIC implementations. Our ASIC results are for the 65 nm CMOS GP technology from TSMC and are obtained with Cadence RTL Compiler. Only registers were used for memory due to the lack of access to an SRAM compiler. Normalized results for the decoders from the literature are also provided. For consistency, only results for a (1024, 512) polar code are compared to match what was done in the other works. It should be noted that [49] provides measurement results.

From Table 2.6, it can be seen that both implementations of our proposed decoder—at different supply voltages—are 46 and 42% the size of the BP decoder [49] and the combinational decoder [14], respectively, when all areas are normalized to 65 nm technology. Our work has two orders of magnitude lower latency than the BP decoder of [49], and two to five times lower latency than [72]. The latency of the proposed design is 1.05 times and 0.7 times that of [14], when operating at 400 and 600 MHz, respectively. The BP decoder [49] employs early termination

Table 2.6 Comparison of state-of-the-art ASIC decoders decoding a (1024, 512) polar code

	This work		[49][a]	[14]	[72]
Algorithm	Fast-SSC		BP	SC	2-bit SC
Technology	65 nm		65 nm	90 nm	45 nm
Supply (V)	0.8	1.0	1.0	1.3	N/A
Oper. temp. (°C)	25	25	≈ 25	N/A	N/A
Area (mm^2)	0.69	0.69	1.48	3.21	N/A
Area @65nm (mm^2)	0.69	0.69	1.48	1.68	0.4
Frequency (MHz)	400	600	300	2.5	750
Latency (μs)	0.41	0.27	50	0.39	1.02
Info. T/P (Gbps)	1.24	1.86	2.4 @ 4dB	1.28	0.5
Sust. Info. T/P (Gbps)	1.24	1.86	1.0	1.28	0.5
Area Eff. (Gbps/mm^2)	1.8	2.7	1.6 @ 4dB	0.4	N/A
Power (mW)	96	215	478	191	N/A
Energy (pJ/bit)	77	115	203 @ 4dB	149	N/A

[a]Measurement results

and its throughput at $E_b/N_0 = 4$ dB is the fastest followed by our proposed design. Since the area reported in [49] excludes the memory necessary to buffer additional received vectors to sustain the variable decoding latency due to early termination, we also report the sustained throughput for that decoder. The sustained throughput is 1.0 Gbps as a maximum of 15 iterations is required for the BP decoder to match the error-correction performance of the SC-based decoders. Comparing the information throughput of all decoders—using the best-case values for BP—it can be seen that the area efficiency of our decoder is the greatest. Lastly, the power consumption estimations indicate that our decoders are more energy efficient than the BP decoder of [49]. Our proposed decoders are also more energy efficient than that of [14]. However, due to the difference in implementation technology, the results of this latter comparison could change if [14] were to be implemented in 65 nm.

2.6 Conclusion

In this chapter, we showed how the original Fast-SSC algorithm implementation could be improved by adding dedicated decoders for three new types of constituent codes frequently appearing in low-rate codes. We also used polar code construction alterations to significantly reduce the latency and increase the throughput of a Fast-SSC decoder at the cost of a small error-correction performance loss. Rate-flexible decoders for polar codes of lengths 1024 and 2048 were implemented on an FPGA. Four low-rate polar codes with competitive error-correction performance were proposed. Their resulting latency and throughput represent a 22–28% reduction and a 26–34% improvement over the previous work, respectively. The information throughput was shown to be 224, 320, 257, and 342 Mbps at approximately

100 MHz on the Altera Stratix IV FPGAs for the $(1024, 342)$, $(1024, 512)$, $(2048, 683)$ and $(2048, 1024)$ polar codes, respectively. On 65 nm ASIC CMOS technology, the proposed decoder for a $(1024, 512)$ polar code was shown to compare favorably against the state-of-the-art ASIC decoders. With a clock frequency of 400 MHz and a supply voltage of 0.8 V, it has a latency of 0.41 μs and an area efficiency of 1.8 Gbps/mm^2 for an energy efficiency of 77 pJ/info. bit. At 600 MHz with a supply of 1 V, the latency is reduced to 0.27 μs and the area efficiency increased to 2.7 Gbps/mm^2 at 115 pJ/info. bit.

http://www.springer.com/978-3-319-59781-2

High-Speed Decoders for Polar Codes

Giard, P.; Thibeault, C.; Gross, W.J.

2017, XVIII, 98 p. 34 illus., 29 illus. in color., Hardcover
ISBN: 978-3-319-59781-2