A Turbo Maximum-a-Posteriori Equalizer for Faster-than-Nyquist Applications

Mohamed Omran Matar*, Mrinmoy Jana*, Jeebak Mitra†, Lutz Lampe*, and Mieszko Lis*
*University of British Columbia †Huawei Canada
*{momran,mjana,lampe,mieszko}@ece.ubc.ca †jeebak.mitra@huawei.com

***Abstract*—Faster-than-Nyquist (FTN) transmission employs non-orthogonal signaling to improve spectral efficiency over conventional orthogonal transmission at the Nyquist rate. However, FTN signaling also introduces inter-symbol interference (ISI), which must be mitigated through additional signal processing.**

In this paper, we present a scalable FPGA-based architecture for a turbo maximum-a-posteriori (MAP) equalizer based on the Bahl-Cock-Jelinek-Raviv (BCJR) algorithm to mitigate the ISI. In contrast to many existing hardware implementations of BCJR, where the algorithm performs turbo decoding over a binary alphabet for an already-equalized channel, our FPGA design applies BCJR to non-binary signal constellations and for the task of equalization. To the best of our knowledge, this is the first published FPGA-based BCJR equalizer implementation suitable for FTN applications, where a binary forward-error-correction decoder is employed in tandem with the equalizer.

Through careful tradeoff selection and optimization of the design space, our implementation achieves a maximum per-PE throughput of 602 Mbps, and a total of 15.4 Gbps within the constraints of a Xilinx UltraScale+ (xcvu13p) device.

I. Introduction

The increase in the amount of data traffic sent through the communication network infrastructure has created a tremendous strain on coherent optical transport networks. Therefore, substantially higher data rates, for example, per carrier data rates of 1 Tbps and more are being targeted in the next generation optical systems [1]. The associated high spectral efficiencies of communication warrant the consideration of spectrally efficient techniques such as Faster-than-Nyquist (FTN) signaling [2–4]. From a practical implementation perspective, FTN is particularly advantageous for transmission systems such as coherent optical communication, where practical constraints make applying higher-order modulation formats challenging. However, the benefits of FTN come at the price of introducing inter-symbol interference (ISI), which requires successful mitigation through additional digital signal processing.

In communications literature, a number of low-complexity ISI mitigation techniques have been investigated for FTN systems, in the form of linear [5] and frequency-domain equalization [6, 7] at the receiver, or pre-equalization at the transmitter [8–12]. However, the performance of these methods is not sufficient when the ISI due to FTN is severe. Instead, in this paper we focus on mitigating ISI via the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm based maximum a-posteriori probability (MAP) equalization [4, 5, 13, 14], which provides improved performance, and we investigate the requirements of a practical and efficient hardware implementation in an FPGA.

To date, very few papers have focused on hardware implementation of FTN systems [3]. Hardware designs have been proposed for transmitters [15, 16] as well as low-complexity equalizers [17]; however, transmitters are much simpler than decoders and low-complexity equalizers are significantly outperformed by MAP-based approaches [4, 17].

Hardware designs for the MAP algorithm abound in the literature, implemented either as ASICs [18, 19] or in FPGAs [20–24]. These MAP architectures are primarily designed for decoding conventional convolutional and turbo codes operating on a binary alphabet, and therefore have a limited number of BCJR states. The relatively small state-space facilitates storage in flip-flop registers, which allows prior designs to simultaneously access the entire BCJR state space in the same clock cycle. Other prior hardware MAP decoder designs have been proposed for decoding non-binary turbo codes [25]; however, the short codeword lengths and limited dynamic ranges of the state variables again permit an inefficient storage allocation scheme.

In contrast, BCJR equalization for FTN systems requires a significantly larger number of states due to the use of higher-order modulation formats such as 16-ary quadrature amplitude modulation (16-QAM) for increased spectral efficiency. For this reason, the existing hardware decoder implementations [18–24] cannot directly perform ISI equalization in FPGA-based systems, as they would run out of flip-flop resources even in the largest FPGAs available today: for example, a flip-flop-only equivalent to the equalizer we propose here would offer only a quarter of the bandwidth of our BRAM-based design in a system with 64 BCJR states.

In this paper, we propose an efficient architecture for an FPGA-based *sliding-window max-log MAP* algorithm [26] for ISI mitigation, which is (a) directly applicable to FTN receivers that exchange soft information with a low-density-parity-check (LDPC) decoder in a turbo fashion [5], and (b) scalable to 64 BCJR states and 64k LDPC codeword length on a Xilinx UltraScale+ FPGA (xcvu13p), achieving a per-PE throughput of 602 Mbps and an on-device total of 15.4 Gbps.

In contrast with previous architectures, we carefully optimize the mapping of BCJR states to on-chip BRAM memories and schedule operation to maximize BRAM port utilization, which allows our design to support large BCJR state spaces. We also describe tradeoffs between BCJR state variable bitwidths and the need for periodic normalization, which allow us to achieve high BRAM utilization. To the best of our

Appears in the Proceedings of the 28nd IEEE Int'l Symposium on Field-Programmable Custom Computing Machines (FCCM 2020)

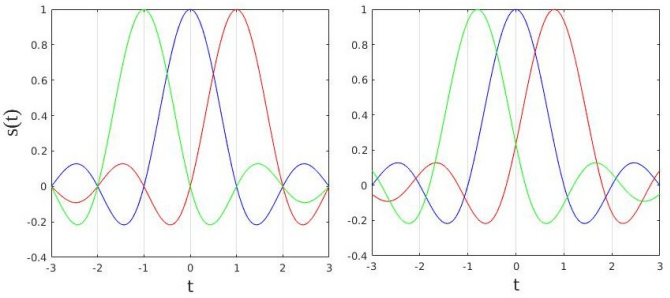

Fig. 1. Left: Nyquist sampling, where each symbol is sampled when the interference from the adjacent symbols is absent (at integer values of t). Right: Faster-than-Nyquist sampling (symbol spacing reduced by 20%), leading to ISI from the adjacent symbols.

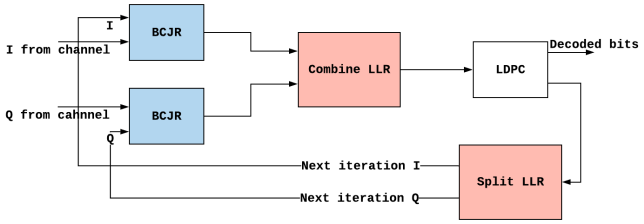

Fig. 2. Turbo BCJR equalization in conjunction with LDPC decoding. The equalizer and the soft-in-soft-out decoder exchange soft information in the form of extrinsic log-likelihood ratios in every turbo iteration, separately for each orthogonal real dimension, i.e. the in-phase (I) and quadrature (Q) components of the complex symbols [5].

knowledge, the proposed architecture is the first published FPGA implementation of MAP equalization capable of direct application in an FTN transmission system.

II. Theoretical Background

Faster-than-Nyquist signaling. Conventional data transmission in communication systems relies on orthogonal linear modulation, known as "Nyquist" signaling [27], in which adjacent pulses do not interfere with each other. FTN signaling [2–4] deliberately violates the orthogonality condition imposed by Nyquist's criterion to transmit the symbols "faster," resulting in increased spectral efficiency, i.e., bits/s/Hz. The equivalent baseband-transmitted FTN signal can be written as

$$s(t) = \sum_{l} x[l]p(t - l\tau T),$$

where τ is the FTN-acceleration factor, l is the symbol index, $\frac{1}{\tau T}$ is the symbol rate, and p is the T-orthogonal pulse-shaping filter, typically a root-raised-cosine filter.

The benefits of FTN come at the cost of introducing ISI, which stems from the overlapping pulses between the adjacent symbols. This is illustrated in Figure 1, for the example of $\tau = 1$ (Nyquist) and $\tau = 0.8$ (FTN). Therefore, an efficient FTN transmission necessitates successful mitigation of the ISI via sophisticated equalizer design. For this, we have adopted the MAP equalizer structure from [5], which iteratively exchanges extrinsic log-likelihood ratios (LLRs) with an LDPC decoder, as shown in Figure 2.

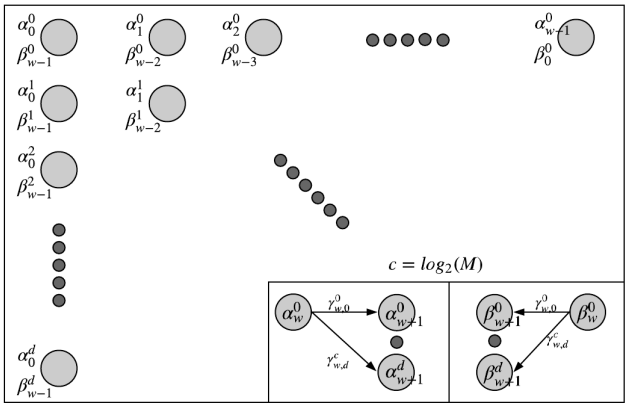

Fig. 3. The forward-backward trellis diagram and data dependencies required to compute α and β values.

MAP equalization and BCJR. The BCJR algorithm, originally proposed for decoding binary convolutional codes in [28], performs a forward-backward recursion over a state transition map or *trellis*. In this paper, we focus on a sliding-window max-log MAP algorithm, which enables hardware-friendly small-block-length processing [26].

Since the FTN pulse p is real-valued, the BCJR equalizer can be applied to real and imaginary parts of the data separately, which greatly reduces the state space [4]. As the first step of the algorithm, a branch metric γ is computed in the log domain, corresponding to the BCJR-state s at time index k, and constellation symbol index c with $m(c)$ being the real-valued constellation symbol according to [13, Eq. (7)] as

$$\gamma(s, c, k) = LLR_{in}(c, k) + [y(k) - x(c, s)]\, m(c),$$

where $LLR_{in}(c, k)$ is the a-priori symbol LLRs fedback from the LDPC decoder in the previous turbo iteration, $y(k)$ is the k^{th} received symbol, and $x(c, s)$ is the noise-free transmitted symbol after passing through the ISI channel. Thereafter, γ is used to recursively compute the forward and backward metrics α and β according to [13, Eqs. (15)–(16)] as

$$\alpha(s, k) = \max_{c} \left[\alpha(s, k-1) + \gamma(s, c, k)\right], \text{ and}$$
$$\beta(s, k) = \max_{c} \left[\beta(s, k+1) + \gamma(s, c, k)\right].$$

The computational schematics are illustrated in Figure 3. Next, α, β, and γ are combined to compute the output symbol-level LLRs as

$$LLR_{out}(c, k) = \max_{d} \left[\alpha(s, k) + \beta(s, k) + \gamma(s, c, k)\right].$$

Finally, the symbol LLRs are converted to extrinsic bit-LLRs, which are provided as inputs to the LDPC decoder [5].

In our FPGA implementation, we use a 16-QAM modulation corresponding to 4 real-valued symbols per real dimension and 3 taps for the ISI channel, resulting in a $4^3 = 64$-state BCJR for each of the in-phase (I) and quadrature (Q) components. Moreover, we evaluate varying window sizes w and overlap lengths o for the sliding-window implementation, as shown in Figure 4.

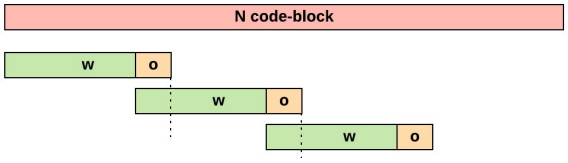

Fig. 4. Sliding window w with overlap o for a block of N symbols for equalization corresponding to one LDPC codeword.

III. Architecture

Sources of parallelism. There are several sources of parallelism in the MAP estimation algorithm. Within each window, the γs can be computed in parallel, as can the output LLRs once αs, βs, and γs are known. The α computation for a given symbol, however, depends on the αs for the previous symbol, so must be computed sequentially (and vice versa with β); the only parallelism for α and β computation is therefore across the encoder states d, a factor of 64.

To obtain additional parallelism, we would ideally also make codewords very short and process many independent codewords at the same time. However, LDPC codes are less efficient with short codewords: for example, the DVB-S2 digital TV standard uses codewords of length 64k.

To overcome this, we employ a sliding-window technique [29] where the MAP equalizer is run independently on small windows of ≥ 128 symbols which are overlapped by ≥ 16 symbols as shown in Figure 4 to ensure smooth α and β propagation. This offers additional parallelism (since windows can be processed independently) at the cost of additional computation (since the overhead segment is recomputed).

Numeric representation and normalization. In decoders that operate on a binary alphabet, the values of α, β, and γ have a very limited range; consequently, very few bits are required to represent them. In contrast, using BCJR in the non-binary context of an FTN equalizer involves values with a much wider range, dramatically increasing the BRAM footprint.

Ranges can be reduced somewhat by normalizing α and β values after every step [30]. This results in the fewest bits per value, but requires computing the minimum and maximum values across the BCJR state space before starting each parallel step — a drastic reduction in parallelism.

In our architecture, we normalize αs and βs periodically rather than after every step (see Figure 5), and trade this off against the number of bits needed per value (see Section IV).

Compute schedule. The computation schedule of the proposed MAP decoder processing element (PE), illustrated in Figure 5, is split into two phases. In the first phase (left-hand side), no LLRs can be produced because there is no cell with both α and β ready. The computation therefore proceeds by propagating αs left-to-right and βs right-to-left, and computing the γs in the order required by the αs and βs. At the end of this phase, all γs have been computed, as well as the first half of the αs and the second half of the βs.

In the second phase, the middle columns of the window have both αs and βs available, so those LLRs can be computed. At the same time, the remaining αs and βs are computed.

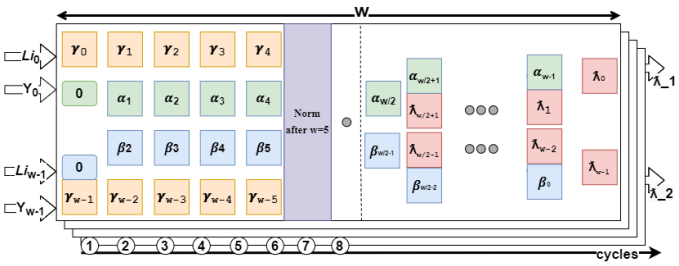

Fig. 5. The proposed MAP decoder computation schedule for one window. Colours represent different functional units, while the depth shows hardware replicated to take advantage of the independent computations across the d and c dimensions. λ denotes output LLR values.

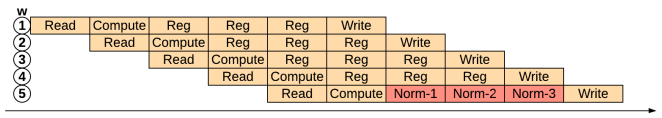

Fig. 6. Pipeline stalls (*Reg*) due to normalization (normalization interval 5).

Across the encoder state (d) and constellation (c) dimensions, the computations are independent, so we replicate functional units within the PEs across d and c. Finally, each window can be computed separately, allowing us to replicate PEs spatially on the FPGA across the w dimension.

Within each window, α and β computation is limited by the periodic normalization. We implement the min/max computation needed for normalization as a three-cycle pipeline overlapped with writing back the computed αs and βs; if values are normalized every five cycles, this allows five αs and βs to be computed in ten clock cycles (see Figure 6).

Memory mapping and scheduling. Unlike prior, much smaller FPGA implementations [20–24], we use BRAMs to store the α, β, and γ values, which requires optimizing memory allocation and carefully scheduling BRAM accesses.

To compute one α (or β), we need to read one previous α (or β) and c values of γ. Since computations across the d dimension are independent, d αs (and βs) can be computed in parallel, requiring d parallel reads of α (or β) in each PE.

We map these to BRAMs as shown in Figure 7. In the Xilinx FPGA we use, BRAMs have 36-bit lines. If values are 9-bit fixed-point, we need $\frac{d}{4} = 16$ BRAMs to allow one PE to access d arbitrary αs in parallel, and another 16 for the βs. For each α (or β) read to produce another α (or β), a full constellation

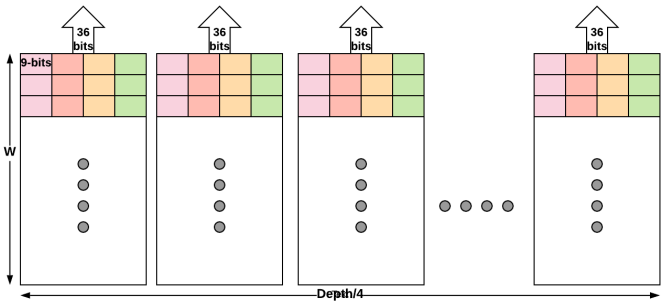

Fig. 7. BRAM partitioning example with a port width of 36 bits and 9-bit fixed-point values.

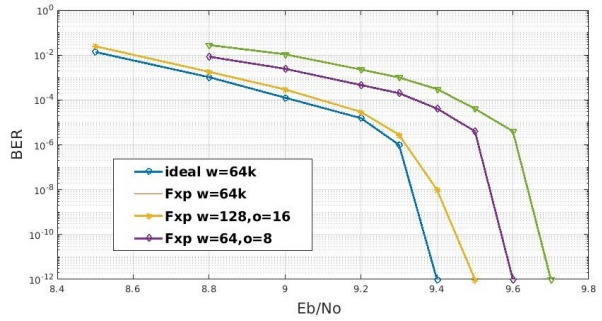

Fig. 8. BER vs. E_b/N_0 in dB comparing idealized floating-point MAP against fixed-point (fxp) MAPs with different sliding-window sizes (w) and overlaps (o), after six LDPC–BCJR iterations.

of c γs is read at once, so multiple γs can be stored in each 36-bit line (four 9-bit values, three 12-bit values, and so on).

IV. Results

To validate design parameters such as bitwidths and window size, we used Matlab to model a bit-accurate TX+RX FTN system with acceleration $\tau = 0.84$ and 16-QAM modulation and an LDPC code of length 64k from the DVB-S2 digital TV standard [31]. We implemented the receiver design in Vivado HLS on a Xilinx UltraScale+ FPGA (xcvu13p).

Ideal vs. hardware MAP. Figure 8 shows bit error rate (BER) vs. bit-energy-to-noise power spectral density ratio (E_b/N_0) for our FTN system with (i) an idealized MAP decoder using a window of the entire transmission block (64k symbols) and 64-bit floating-point α, β, and γ (blue), and (ii) 9-bit fixed-point versions of our hardware implementation with normalization every five steps and different window and overlap sizes (green, purple, and yellow). The 128-symbol sliding window with a 16-symbol overlap (purple) results in abundant parallelism at the cost of only 0.2 dB of SNR drop.

Parallelism vs. throughput. Table I shows how varying window sizes affects the throughput of one BCJR iteration achievable for equalizing the block of received samples corresponding to one 64k-bit codeword of the LDPC code at an FTN acceleration factor $\tau = 0.84$, assuming no area constraints. Smaller window sizes permit more per-window PEs, and higher potential throughputs of a full PE array. (We constrained the clock to 250 MHz but sometimes achieved a higher frequency, which causes small variations in the throughput/PE.)

Table II shows the resource utilization of a single PE for a variety of sliding windows. LUT utilization does not scale linearly with each sliding window (consistent with prior work [23]). Optimal BRAM utilization is achieved at window size 1024; with larger sizes, each α, β, and γ arrays exceed the size of one BRAM block and require more BRAMs.

TABLE I
Without area limits, smaller windows permit more PEs and throughput.

max. # PEs	window size	throughput/PE	max. potential throughput
450	128	**436** Mbps	**196 Gbps**
240	256	427 Mbps	102 Gbps
128	512	427 Mbps	54 Gbps
64	1024	430 Mbps	27 Gbps
32	2048	396 Mbps	12 Gbps

TABLE II
Resource utilization for different sliding windows.

window size	LUTs	CLBs	registers	18-kbit BRAMs
128	55312	10416	24901	130 (2.4%)
256	56200	10084	24907	130 (2.4%)
512	56509	9902	24936	130 (2.4%)
1024	57661	10562	24978	130 (2.4%)
2048	56344	9915	24983	216 (4%)

Normalization. Tables III and IV show the resource footprints and throughputs as bitwidths are traded off against normalization intervals under the constraint that BER is negligible at an E_b/N_0 of 9.6 dB. While the best per-PE BRAM utilization occurs at 9 bits, frequent normalizations required by the small bitwidths create an Amdahl bottleneck and limit full-FPGA throughput. Higher bitwidths make normalization less frequent and increase per-PE throughput, but fewer PEs fit on the FPGA, and throughput also suffers. The highest total throughput is at 12 bits: normalizations are rare, while BRAMs are fully utilized (3 values per 36-bit line) and BRAM/LUT utilization percentages are balanced, allowing 27 PEs to fit.

TABLE III
Bitwidth vs. normalization: single-PE area (window sizes 128 and 360).

	# bits	norm. int.	LUTs	DSPs	registers	18-kbit BRAMs
window 128	9	5	50164 (2.9%)	752	24901	130 (2.4%)
	10	10	51379 (3.0%)	752	27853	194 (3.6%)
	11	20	54993 (3.2%)	752	28527	194 (3.6%)
	12	40	60739 (3.5%)	752	27355	194 (3.6%)
	13	80	65146 (3.8%)	752	28202	194 (3.6%)
window 360	9	5	50432 (2.9%)	752	24901	130 (2.4%)
	11	20	56157 (3.2%)	752	28527	194 (3.6%)
	12	40	61622 (3.6%)	752	27355	194 (3.6%)
	13	80	66805 (3.9%)	752	28202	194 (3.6%)
	15	320	75286 (4.4%)	752	28202	194 (3.6%)

TABLE IV
Bitwidth vs. normalization: throughput (window sizes 128 and 360).

	# bits	norm. int.	throughput/PE	#PEs	total throughput
window 128	9	5	436 Mbps	33	14.4 Gbps
	10	10	517 Mbps	27	13.9 Gbps
	11	20	557 Mbps	27	15 Gbps
	12	40	570 Mbps	27	**15.4 Gbps**
	13	80	**602 Mbps**	25	15.05 Gbps
window 360	9	5	433 Mbps	32	14 Gbps
	11	20	541 Mbps	27	15 Gbps
	12	40	566 Mbps	27	**15.31 Gbps**
	13	80	582 Mbps	25	14.74 Gbps
	15	320	**591 Mbps**	21	12.53 Gbps

V. Conclusion

The scalable MAP decoder architecture we have presented is, to the best of our knowledge, the first FPGA design to scale to the sizes required by FTN applications. The architecture combines careful optimization for BRAM utilization, as well as a tradeoff between normalization intervals and bit-widths used to store intermediate values, and achieves up to 602 Mbps per PE or a total of 15.4 Gbps in a full Xilinx UltraScale+ (xcvu13p) device.

References

[1] T. A. Strasser and J. L. Wagener, "Wavelength-selective switches for roadm applications," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 16, no. 5, pp. 1150–1157, Sep. 2010.

[2] J. E. Mazo, "Faster-than-Nyquist signaling," *The Bell System Technical Journal*, vol. 54, no. 8, pp. 1451–1462, 1975.

[3] J. B. Anderson, F. Rusek, and V. Owall, "Faster-Than-Nyquist Signaling," in *Proc. IEEE*, vol. 101, no. 8, 2013, pp. 1817–1830.

[4] F. Rusek, "Partial response and faster-than-nyquist signaling," Ph.D. dissertation, Department of Electrical and Information Technology, 2007.

[5] M. Jana, L. Lampe, and J. Mitra, "Interference Cancellation for Time-Frequency Packed Super-Nyquist WDM Systems," *IEEE Photonics Technology Letters*.

[6] S. Sugiura, "Frequency-Domain Equalization of Faster-than-Nyquist Signaling," *IEEE Wireless Commun. Lett.*, vol. 2, no. 5, pp. 555–558, October 2013.

[7] S. Sugiura and L. Hanzo, "Frequency-Domain-Equalization-Aided Iterative Detection of Faster-than-Nyquist Signaling," *IEEE Trans. Veh. Technol.*, vol. 64, no. 5, pp. 2122–2128, May 2015.

[8] F. Rusek and J. B. Anderson, "Non Binary and Precoded Faster Than Nyquist Signaling," *IEEE Trans. Commun.*, vol. 56, no. 5, pp. 808–817, May 2008.

[9] M. Jana, A. Medra, L. Lampe, and J. Mitra, "Pre-equalized Faster-than-Nyquist Transmission," *IEEE Trans. Commun.*, vol. 65, no. 10, pp. 4406–4418, October 2017.

[10] M. Jana, L. Lampe, and J. Mitra, "Precoded Time-Frequency-Packed Multicarrier Faster-than-Nyquist Transmission," in *IEEE Int. workshop on Signal Proc. advances in Wireless Commun. (SPAWC)*, 2019.

[11] D. Chang, O. Omomukuyo, X. Lin, S. Zhang, O. Dobre, and V. Ramachandran, "Robust Faster-than-Nyquist PDM-mQAM Systems with Tomlinson-Harashima Precoding," *IEEE Photon. Technol. Lett.*, vol. 28, no. 19, October 2016.

[12] M. Jana, L. Lampe, and J. Mitra, "Dual-polarized Faster-than-Nyquist Transmission Using Higher-order Modulation Schemes," *IEEE Trans. Commun.*, vol. 66, no. 11, pp. 5332–5345, November 2018.

[13] G. Colavolpe and A. Barbieri, "On MAP symbol detection for ISI channels using the Ungerboeck observation model," *IEEE Commun. Lett.*, vol. 9, no. 8, pp. 720–722, August 2005.

[14] M. Secondini, T. Foggi *et al.*, "Optical Time-Frequency Packing: Principles, Design, Implementation, and Experimental Demonstration," *J. Lightw. Technol*, vol. 33, no. 17, pp. 3558–3570, September 2015.

[15] M. R. Perrett and I. Darwazeh, "Flexible Hardware Architecture of SEFDM Transmitters with Real-Time Non-Orthogonal Adjustment," in *IEEE Int. Conf. on Telecommun.*, 2011.

[16] P. N. Whatmough, M. R. Perrett, S. Isam, and I. Darwazeh, "VLSI Architecture for a Reconfigurable Spectrally Efficient FDM Baseband Transmitter," *IEEE Trans. on Circuits and Sytems I: Reg. Papers*, vol. 59, no. 5, pp. 1107–1118, February 2012.

[17] D. Dasalukunte, "Multicarrier Faster-than-Nyquist Signaling Transceivers," Ph.D. dissertation, Lund University, 2012.

[18] S. Belfanti, C. Roth, M. Gautschi, C. Benkeser, and Q. Huang, "A 1Gbps LTE-advanced turbo-decoder ASIC in 65nm CMOS," in *2013 Symposium on VLSI Circuits*, June 2013, pp. C284–C285.

[19] S. Weithoffer, C. A. Nour, N. Wehn, C. Douillard, and C. Berrou, "25 years of turbo codes: From Mb/s to beyond 100 Gb/s," in *International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, Dec 2018, pp. 1–6.

[20] L. F. Gonzalez-Perez, L. C. Yllescas-Calderon, and R. Parra-Michel, "Parallel and configurable turbo decoder implementation for 3GPP-LTE," in *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, Dec 2013, pp. 1–6.

[21] R. G. Maunder, "A fully-parallel turbo decoding algorithm," *IEEE Transactions on Communications*, vol. 63, no. 8, pp. 2762–2775, Aug 2015.

[22] M. I. del Barco, G. N. Maggio, D. A. Morero, J. Fernandez, F. Ramos, H. S. Carrer, and M. R. Hueda, "FPGA implementation of high-speed parallel maximum a posteriori (MAP) decoders," in *Argentine School of Micro-Nanoelectronics, Technology and Applications*, Oct 2009, pp. 98–102.

[24] A. Li, P. Hailes, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "1.5 Gbit/s FPGA implementation of a fully-parallel turbo decoder designed for mission-critical machine-type communication applications," *IEEE Access*, vol. 4, pp. 5452–5473, 2016.

[25] Y. Du, "Hardware Implementation of Non-Binary Turbo Code for DVB/RCS," Master's thesis, Concordia University: Gina Cody School of Engineering and Computer Science, 2003.

[26] P. Salmela, S. Harri, and J. Takala, "A Programmable Max-Log-MAP Turbo Decoder Implementation," *VLSI Design*, vol. 2008, 10 2008.

[27] H. Nyquist, "Certain Topics in Telegraph Transmission Theory," *Trans. on the American Inst. of Elec. Engg.*, vol. 47, no. 2, pp. 617–644, April 1928.

[28] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Trans. Inf. Theory*, vol. 20, no. 2, pp. 284–287, March 1974.

[29] S. Huettinger, M. Breiling, and J. Huber, "Memory efficient implementation of the BCJR algorithm," in *International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, 2000.

[30] Yufei Wu, B. D. Woerner, and T. . Blankenship, "Data width requirements in SISO decoding with module normalization," *IEEE Trans. Commun.*, vol. 49, no. 11, pp. 1861–1868, Nov 2001.

[31] "Digital Video Broadcasting (DVB): Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2)."