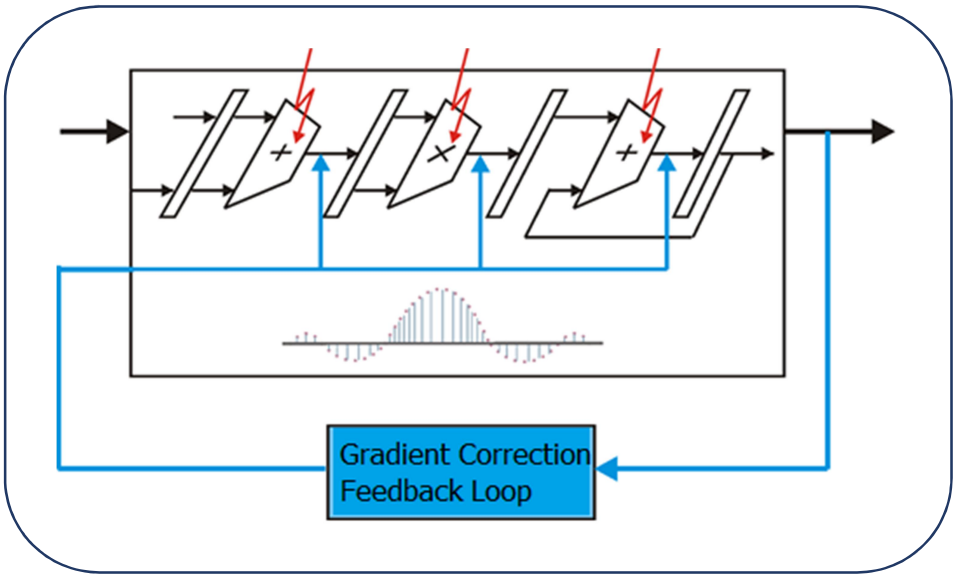


RESPECT-DEM - Reliable Signal Processing Datapaths Design Using Control Techniques Based on Difference Equation Models

Executive Summary

Early Technology Development

OSIP Idea channel

Affiliation:University Politehnica Timișoara

Activity summary:
This activity developed a fault tolerant (FT) mechanism for linear DSP applications. The mechanism uses three ingredients: parity based real number error correction codes (ECC) to store redundant information efficiently, gradient descent correction loops for error estimation, and fine grained check-pointing and roll-back for consistent redundant information encoding. The error correction is performed by a novel gradient descent symbol update method designed for real number codes. The proposed solution has been validated for a 512 point memory-based with stage-by-stage execution FFT architecture implementation.

Publishing Date: 25.11.2022
Contract Number: 4000134184
Implemented as ESA Standard Procurement

List of Authors

Partner	Author & Contact Person for Technical Questions
UPT	Dr. Oana Boncalo, (oana.boncalo@cs.upt.ro) – Any Dr. Alexandru Amaricai

List of Figures

Executive Summary

0.1 Related Work on Error Correction Codes for Processing

Algebraic properties of linear computation have been exploited in enhancing the fault tolerance of hardware architectures [2–7]. If the linearity requirement is met, every linear code defined over a finite field has a corresponding linear real-number code with similar error detection and correction capabilities [7].

State-of-the-art has shown that fault tolerant systems typically have two functions: detection and correction. For the detection part, either mathematical properties of the underlying applications (*e.g.* Parseval's checks for the Fast Fourier Transforms (FFTs) [3]), or syndrome check for error correction codes (ECCs)(*e.g.* [5], [6], [8, 9]) are used. The mitigation of the effect of output errors follows two trends. The first one is to recompute the data [6], [10], while the second is to perform some form of decoding [11], [4], [5].

There are two distinct possibilities for embedding ECC in linear digital circuits: at unit level, and inside the processing units. For the first approach, redundant processing units that are computing linear combinations of input data of the non-redundant ones are employed. Using the linear transformation property, the outputs can be regenerated from the non-redundant units and the redundant units outputs. The linear combinations at the inputs and outputs are dictated by the code's parity check matrix [3], [4], [5]. Thus, p units operate on p independent sets of input data. In practice, this requirement may demand $p\times$ the data memory increase to store the p independent data sets, which is non-trivial overhead in case of real-life applications, such as the 512 point FFT.

The second method is to embed ECCs inside the units. Additional components, such as detection logic, syndrome computation, correction blocks, are added to the original baseline architecture. Thus, a fault tolerant extension of the linear application architecture is built. A set of input data may be mapped to one or several code-words.

The next problem that needs to be solved is how the actual correction, also called decoding in information theory, is done. For ECC which can correct a single error, typically decoding tables are used (e.g. small code-word Hamming (7,4) or HSIAO codes) [3], [4],[5]. These codes are only suitable for unit level correction if the application supports independent data sets, and there are sufficient resources to store them (*e.g.* for the Hamming (7,4) code - 4 independent data sets, and 3 redundant data sets, accounting for $7\times$ input data storage overhead). The scenario where the ECC corrects more than one error requires more sophisticated decoding algorithms, many of which are iterative. The complexity of thee decoding procedures is non-trivial.

Besides the number of corrected errors, another trait that distinguished different research results is the input to the decoding process. Some works take an approach similar to that of information theory, mainly the decoder is loaded with the code-word [12], while other focus on estimating the error[8, 9]. The latter is equivalent to all-zero code-word decoding. The first one has been applied to small scale applications (*e.g.* a third state linear system having a few states). For larger systems, its main drawback is the linear transform extension for computing the output redundant symbols. For large scale applications, such as a 512 point FFT, this linear transform extension, if found for a given code-word, is unsuitable due to being non-sparse, and considerably more complex to implement than the application itself. Hence, the output code-word data is not available. And, since input data storage size is also costly, re-computation of the entire data-set is both lengthy and costly at the same

time, albeit the principle itself is simple.

The case of large size code-words (hundred, thousands of symbols) that can correct more than one error, needs an entire different approach. Pioneering work in this respect has been done by Robert Redinbo [8,9], in proposing an iterative method relying on a output Kalman filter to estimate the error. The works of [8, 9] are true iterative corrections using an extended memory and prolonged execution window to extract the output error from the syndrome. The downside of this approach is, on one hand, the extended execution needed to gather the data window required by the output Kalman filter (in the numerical example 48 steps have been reported for reaching stable state), an on other hand, the storage and computation overhead. No implementation has been made for the proposed solution. All estimates, as well as the corresponding discussion, is supported solely on MATLAB simulations. The extra operations required are estimated roughly. It is not clear how these map to an actual implementation. The extra storage required is not discussed.

This research project investigates the case when:

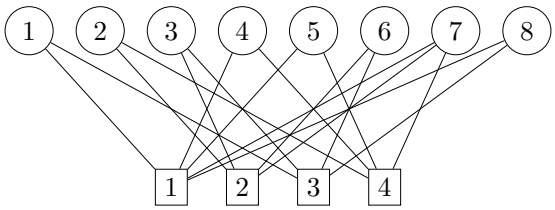
- Large scale linear transform application implemented in a memory-dominant hardware architecture (*e.g.* the use-case is a 512 point FFT);
- Parity based ECC for storing redundant information with hundred-thousand symbol code-words(*e.g.* the use-case is a 768 symbol rate 23 Low-Density Parity-Check code);
- Iterative error estimation using Gradient Descent optimization and Maximum Likelihood decoding principle;
- Data-level (not! data-set) check-pointing with roll-back of it in case of encoder input error;
- More than one error appear during execution;
- Detection by means of syndrome;
- The linear transform extension is not suitable for HW implementation (number of non-zero elements extension greater than the number of non-zero elements in the linear transform application matrix)

Since the target architecture is memory based, the present solution needs to limit the memory overhead (similar to dual memory redundancy), while keeping the additional processing "reasonable". Thus, the final result is a fault tolerant (FT) mechanism suitable for linear DSP applications. The error correction for the proposed approach is performed by a novel gradient descent symbol update method designed for real number codes. These ideas are validated for a 512 point memory-based FFT architecture with stage-by-stage execution.

0.2 Technical contributions

The contributions of this work are as follows:

- Development of the real number gradient descent iterative decoding algorithm;
- Proposal of a novel fault tolerant technique for linear DSP digital designs using gradient descent correction loop for parity based error correction codes, and low-level check-pointing followed by re-execution;
- Efficient implementation and evaluation of the proposed fault tolerant approach for large scale application: the 512 point Fast Fourier Transform (FFT);
- Investigation of automation methods using high level synthesis (HLS) models from Matlab;



$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Figure 1: Tanner Graph with equivalent H matrix representation – Example from [1]

0.2.1 Gradient Descent Iterative Symbol Update

The practical binary decoders for LDPC codes are classified into two groups: either soft-information decoders, i.e. Belief Propagation (BP), Min-Sum (MS) [13], which are very good in term of error correction at the expense of complexity, or hard-decision decoders, i.e. Bit-Flipping (BF)[14], Gradient Descent Bit-Flipping (GDBF) [15], which are usually weaker in terms of error correction, but significantly lower in complexity with respect to BP. The bit-flipping algorithm h proposed by Gallager in [14] is a hard decision algorithm developed for the binary symmetric channel (BSC). The gradient descent bit flipping (GDBF) algorithms have been propose by Wadayama in [15], and it can be regarded as a simplified gradient descent algorithm, derived from a simple nonlinear objective function.

The parity based linear code is defined by a parity matrix H, and it can also be viewed in graph form, called Tanner graph [16]. It is a bipartite graph, comprising of two types of nodes: the variable node set, corresponding to the columns of H, and the control nodes set, corresponding to the rows of the matrix H, while the edges of the graph correspond to non-zero elements of the parity matrix. The information data will be referred hereafter as symbol. A code word x is obtained from an information word a, of size M, by the equation $x = aG$, where G is the code generator matrix satisfying the equation $HG^T = 0$. Let the observable representing the word to be decoded be denoted by the vector $y = (y_1, ..., y_N)^T$. Then, the maximum likelihood (ML) decoding is the search for the code word $x = (x_1, ..., x_N)^T$ having the maximum correlation with the vector y.

Quasy-Cyclic Low-Density Parity-check codes (QC-LDPC) codes are a class of structured Low-Density Parity-Check (LDPC) codes that are obtained by *expanding* a base matrix B by an expansion factor. Specifically, each B matrix entry is replaced by either a circulant permuted matrix, or by a zero matrix [20]. The non-negative entries from the base matrix B are replaced by the unitary matrix shifted by the corresponding B matrix entry value. The negative entries are replaced by the zero matrix. The resulting structure is very favorable for hardware implementations, motivating their use in many standards for communication and memory.

Many of the bit-flipping algorithms employ an inversion function, sometimes called energy function, which estimates the reliability of the received channel samples in order to decide which symbols (bits) are updated (flipped). The inputs of this function are received channel information,= and the hard decision syndrome obtained based on the code's H matrix. The strategy behind the single bit-flip algorithms is to select the least reliable bit and flip it during each iteration. The underlying correction principle is to compute a sum over the adjacent parity check equations for each bit in the code, and flip those with the sum larger than a threshold. This process is repeated until all parity checks are satisfied, or until a maximum number of attempts is made. In multiple bit-flipping algorithms, all the bits with reliability below a given threshold are flipped; hence, multiple bits may be flipped during one iteration in parallel, allowing for faster convergence. It can easily be seen that the BF algorithm has very low complexity since it only requires, in each iteration, a sum of the binary parity-check values for each symbol; however, this strategy has lowest decoding performance with respect to the

BP counterparts.

This work investigates a decoding solution that extracts the errors given that the actual data may be unavailable. It is inspired by the information theory binary gradient descendant bit-flip decoding algorithm [15]. Since data is unavailable due to pipeline, being overwritten by current stage computations, the decoding process can be regarded as an all-zero code-word decoding. The proposed Gradient Descent Symbol Update decoding for real number codes uses a pessimistic update rule (absolute minimum value of possible updates is selected) and the concept of reliable correction direction (computed based on the signs of real parity check values from the syndrome vector) in order to limit the impact of a miss-decision.

LDPC Code Selection

The codes selected in this work need to be compatible with the 512 FFT use-case; hence, the following requirements:

- R_1 the expansion factor z needs to be a power of 2;
- R_2 the number of columns in the generator matrix B corresponding to the useful (data) symbols needs to be a power of 2;
- R_3 encoding needs to be as simple as possible, such that it can be efficiently implemented;
- R_4 the information for the code that needs to be stored inside the encoder/correction loop is as small as possible;
- R_5 existing code, since code design is a non-trivial task;

We identified two standard codes that match the aforementioned requirements: IEEE 802.11 (popularly referred as WiFi) [21], IEEE 802.16 (WiMAX) [22] rate 2/3 codes. These are quasi-cyclic systematic codes, meaning that the parity/auxiliary/redundant symbols are separated from the data symbols in the base matrix B. As shown in the research from[23, 24], the encoding of this standard codes is a three step process: first $H^s \times s$ is computed, where s is the vector used to denote the source symbols, and H^s is the parity matrix obtained by expanding B^s; second, the first parity information symbol is computed; third, the remaining parity symbols are computed exploiting the staircase structure. In order to implement it with no additional overhead, we simplified the double diagonal corresponding to the redundant symbols, to a single diagonal. Therefore, the base matrix of the LDPC code is:

$$B_{Short} = \begin{pmatrix} 25 & 26 & 14 & -1 & 20 & -1 & 2 & -1 & 4 & -1 & -1 & 8 & -1 & 16 & -1 & 18 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ 10 & 9 & 15 & 11 & -1 & 0 & -1 & 1 & -1 & -1 & 18 & -1 & 8 & -1 & 10 & -1 & -1 & 0 & -1 & -1 & -1 & -1 & -1 & -1 \\ 16 & 2 & 20 & 26 & 21 & -1 & 6 & -1 & 1 & 26 & -1 & 7 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & -1 & -1 & -1 & -1 & -1 \\ 10 & 13 & 5 & 0 & -1 & 3 & -1 & 7 & -1 & -1 & 26 & -1 & -1 & 13 & -1 & 16 & -1 & -1 & -1 & 0 & -1 & -1 & -1 & -1 \\ 23 & 14 & 24 & -1 & 12 & -1 & 19 & -1 & 17 & -1 & -1 & -1 & 20 & -1 & 21 & -1 & -1 & -1 & -1 & -1 & 0 & -1 & -1 & -1 \\ 6 & 22 & 9 & 20 & -1 & 25 & -1 & 17 & -1 & 8 & -1 & 14 & -1 & 18 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & -1 & -1 \\ 14 & 23 & 21 & 11 & 20 & -1 & 24 & -1 & 18 & -1 & 19 & -1 & -1 & -1 & -1 & 22 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & -1 \\ 17 & 11 & 11 & 20 & -1 & 21 & -1 & 26 & -1 & 3 & -1 & -1 & 18 & -1 & 26 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 \end{pmatrix}$$

We next evaluate the performance of the $\text{Short}_{2/3}$ code for the GDSU, the binary PGDBF anf GDBF, as well as the binary GDBF and PGDBF versions of the WiFi code. The Output and Symbol error rate statistics are presented in Fig 2, while the average iteration is depicted in Fig 3.

Fig 3 shows that $\text{Short}_{2/3}$ has the best average iteration of all simulated decoders. Fig 2 denotes that the performance of the GDSU is on par with the binary PGDBF using a superior code.

0.2.2 Fault Tolerant Architecture Extension: the GDSU Integration

The use-case baseline architecture is a 512 point FFT having one butterfly processing unit and computing the results in a stage by stage manner similar to the one from [25–27]. For this architecture the data memory is updated (overwritten) at the end of each stage. The ECC code provides protection for the data stored in memory, and against processing errors, such as those that arise during the LDPC encoding. The proposed approach is shown in Fig 4. The operation of the FT components is

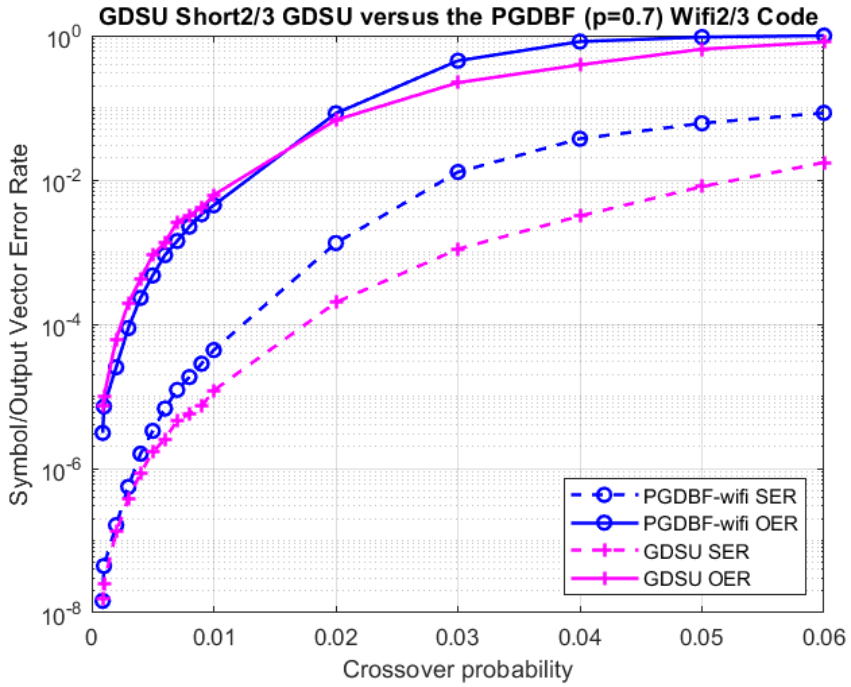


Figure 2: Comparative study of the decoding performance statistics for binary PGDBF for WiFi rate 2/3 code and $\text{Short}_2/3$ GDSU. A total of 100 Output/Frame Errors have been simulated for each point out of a maximum of 100 million frames, and a maximum of 100 decoding iterations.

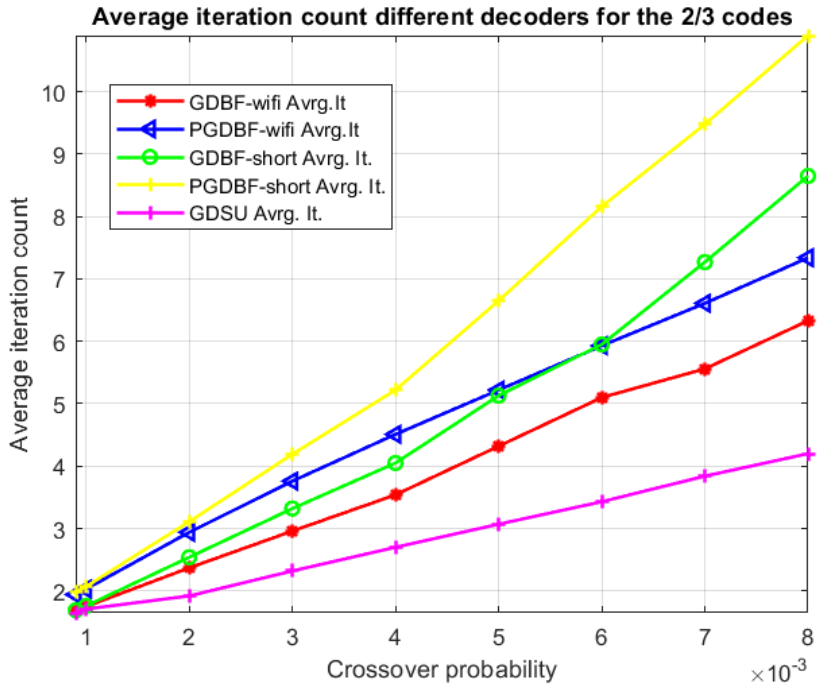


Figure 3: Study of the average iteration count for binary and real codes. A total of 100 Output/Frame Errors have been simulated for each point out of a maximum of 100 million frames, and a maximum of 100 decoding iterations.

as follows: the Syndrome is computed for the stage input data, and in case it fails, a gradient descent optimization loop implemented inside the correction blocks estimates the input stage error (e). From the information theory point of view, this is equivalent to all-zero code-word decoding. The linearity of the FFT transform allows for the stage output data to be updated by an error offset computed by FFT butterfly execution of the $-e$ vector. Data is encoded each time it is saved/updated in the data memory. An additional protection is designed for the butterfly processing unit: double-redundancy for detection, as well as check-pointing and re-execution of mismatch output data for correction. This solution is practical for memory-based architectures, given the small size of processing compared to the storage area requirements.This protection ensures that there are no encoded data errors leading to an erroneous syndrome, and consequently erroneous correction steps in the next FFT stage processing. Note that, in case an error appears, the stage execution (normal execution) freezes, and the gradient descent error correction loop extracts the error, computes and updates data memory location by the error offsets. In case this operation is successful, normal stage execution is resumed.

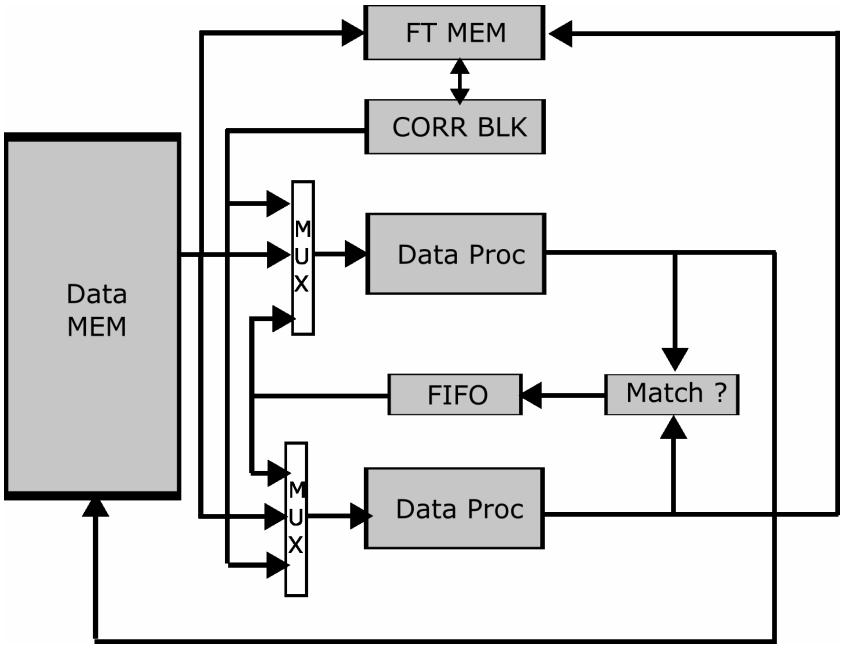


Figure 4: A birds-eye view of the FT FFT with LDPC redundancy and gradient based correction loop.

An important design decision is how many data elements are encoded per clock cycle. We considered two possibilities: the first, which is also implemented in literature assumes that all data symbols are encoded simultaneously (for the 1 butterfly FFT implementation, this means two data symbols), or, the second, to encode each data symbol in a different code-word. We refer to the first as Multiple Data Channel (MDC) approach, while the latter is denoted as Single Data Channel (SDC) approach.

MDC

The MDC is the most constrained. For the 512 point FFT architecture, with 1 butterfly, this approach required careful data access mapping between FFT data from different iterations, and the LDPC code symbols. In addition to this, for the 1 butterfly scenario, quad-port memory blocks are needed (2 reads and 2 writes simultaneously, one for each FFT data). However, if the number of butterflies is increased, the mapping rule becomes more constraint. It is non-trivial, if at all possible, to find a conflict free data mapping rule. Furthermore, the number of required access port for the FT Memory Block also increases proportionally with the number of processing units. Hence, it becomes less suitable for hardware implementations.

SDC

SDC has the advantage that it decouples the application data access flow from the LDPC data access flow. Furthermore, no mapping rule between the application data and LDPC symbols. This solution can be implemented using simple dual port block memories. The disadvantage of SDC is the fact that the computation of the correction offsets is serialized, thus, a minor execution overhead is expected since the number of updates per iteration is relatively small.

0.2.3 Implementations and Results

We implemented a fault tolerant FFT architecture using a modified standard code (WiFi 802.11 rate 2/3). The decoding performance and average iteration count are studied. The real decoding algorithm has comparable frame performance with the best gradient descent counterpart. The cost evaluation is made through a Verilog HDL implementation. The memory requirements is 2× that of the baseline (non-FT) FFT.Logic exceeds the FFT since the underlying baseline architecture consists of a single butterfly processing element, some switches, and the twiddle ROMs. Hence, for FPGA technology (Xilinx Virtex 7) the logic after synthesis is 5977 LUT for the Multiple Data Encoding Channel, and is estimated to 5024 LUT for the Single Data Encoding Channel. We explored the possibility of using HLS for automated code generation and compared the output with the hand-optimized design. The Matlab generated models have shown an overhead of 65%-up to 700%. The worst case is a consequence of the limited ability to model memory block design optimizations. Future work aim at investigating the cost estimates of the FT-FFT architecture for a larger number of butterfly units.

0.3 Conclusions

We have proposed and validated a fault tolerant mechanism relying on ECC and gradient descent optimizations to extract the errors for a linear transform circuit implementation. To the best of out knowledge, this is the first implementation of ECC protection mechanism embedded inside the linear transform. Future work will evaluate the scalability and cost of the SDC solution when integrated in the fault tolerant FFT architectures with a higher degree of parallelism at processing unit level.

Bibliography

[1] O. Boncalo, "Qc-ldpc gear-like decoder architecture with multi-domain quantization," in *2016 Euromicro Conference on Digital System Design (DSD)*, Aug 2016, pp. 244–251.

[2] G. R. Redinbo, "Designing checksums for detecting errors in fast unitary transforms," *IEEE Transactions on Computers*, vol. 67, no. 4, pp. 566–572, 2018.

[3] Z. Gao, P. Reviriego, Z. Xu, X. Su, M. Zhao, J. Wang, and J. A. Maestro, "Fault tolerant parallel ffts using error correction codes and parseval checks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 2, pp. 769–773, 2016.

[4] Y. Xie, C. Yang, C. Mao, H. Chen, and Y. Xie, "A novel low-overhead fault tolerant parallel-pipelined fft design," in *2017 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2017, pp. 1–4.

[5] Z. Gao, P. Reviriego, W. Pan, Z. Xu, M. Zhao, J. Wang, and J. A. Maestro, "Fault tolerant parallel filters based on error correction codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 2, pp. 384–387, 2015.

[6] J. Sloan, R. Kumar, and G. Bronevetsky, "An algorithmic approach to error localization and partial recomputation for low-overhead fault tolerance," in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2013, pp. 1–12.

[7] V. Nair and J. Abraham, "Real-number codes for fault-tolerant matrix operations on processor arrays," *IEEE Transactions on Computers*, vol. 39, no. 4, pp. 426–435, 1990.

[8] G. Redinbo, "Generalized algorithm-based fault tolerance: error correction via kalman estimation," *IEEE Transactions on Computers*, vol. 47, no. 6, pp. 639–655, 1998.

[9] G. Robert Redinbo, "Wavelet codes: Detection and correction using kalman estimation," *IEEE Transactions on Signal Processing*, vol. 57, no. 4, pp. 1339–1350, 2009.

[10] E. S. Sogomonyan, S. Weidling, and M. Goessel, "A new method for correcting time and soft errors in combinational circuits," in *2013 IEEE 16th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2013, pp. 283–286.

[11] A. Dutta and A. Jas, "Combinational logic circuit protection using customized error detecting and correcting codes," in *9th International Symposium on Quality Electronic Design (isqed 2008)*, 2008, pp. 68–73.

[12] M. Ashouei and A. Chatterjee, "Checksum-based probabilistic transient-error compensation for linear digital systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 10, pp. 1447–1460, 2009.

[13] M. P. C. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Transactions on Communications*, vol. 47, no. 5, pp. 673–680, May 1999.

[14] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.

[15] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi, "Gradient descent bit flipping algorithms for decoding ldpc codes," in *2008 International Symposium on Information Theory and Its Applications*, 2008, pp. 1–6.

[16] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, Sep 1981.

[17] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 599–618, 2001.

[18] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. on Information Theory*, vol. 47, no. 2, pp. 619–637, 2001.

[19] T. Richardson and R. Urbanke, "The renaissance of gallager's low-density parity-check codes," *IEEE Communications Magazine*, vol. 41, no. 8, pp. 126–131, Aug 2003.

[20] M. P. C. Fossorier, "Quasicyclic low-density parity-check codes from circulant permutation matrices," *IEEE Transactions on Information Theory*, vol. 50, no. 8, pp. 1788–1793, Aug 2004.

[21] "Wireless LAN medium access control (MAC) and physical layer (PHY) specification," *IEEE Std. 802.11*, 1997.

[22] IEEE-802.16e, "Physical and medium access control layers for combined fixe and mobile operation in licensed bands," 2005, amendment to Air Interface for Fixed Broadband Wireless Access Systems.

[23] Y. Jung, C. Chung, J. Kim, and Y. Jung, "7.7gbps encoder design for ieee 802.11n/ac qc-ldpc codes," pp. 215–218, 2012.

[24] A. Mahdi, N. Kanistras, and V. Paliouras, "A multirate fully parallel ldpc encoder for the ieee 802.11n/ac/ax qc-ldpc codes based on reduced complexity xor trees," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 1, pp. 51–64, 2021.

[25] P.-Y. Tsai and C.-Y. Lin, "A generalized conflict-free memory addressing scheme for continuous-flow parallel-processing fft processors with rescheduling," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 12, pp. 2290–2302, 2011.

[26] Z.-G. Ma, X.-B. Yin, and F. Yu, "A novel memory-based fft architecture for real-valued signals based on a radix-2 decimation-in-frequency algorithm," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 9, pp. 876–880, 2015.

[27] K.-F. Xia, B. Wu, T. Xiong, and T.-C. Ye, "A memory-based fft processor design with generalized efficient conflict-free address schemes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 6, pp. 1919–1929, 2017.