FPGA implementation of Interpolation-Based Chase BCH Decoding

Kiran k[1]
[1] Final year M.Tech student, E&C Engg. Dept.,
PES College of Engineering, Mandya
kiran.kkiru@gmail.com

Dr. Radhakrishna Rao K A PhD [2]
[2] Professor, E&C Engg. Dept.,
PES College of Engineering, Mandya
karkrao@yahoo.com

***Abstract*— BCH codes are is widely used in digital telecommunication systems, with satellite communications and data recording systems such as CD and DVD. Compared with hard-decision decoding of BCH codes, the soft-decision Chase algorithm can achieve significant coding gain by carrying out decoding trials on 2^{η} test vectors. Earlier the one-pass Chase schemes find the error locators based on the Berlekamp's algorithm and need hardware-demanding selection methods to decide which locator corresponds to the correct code word.**

In this paper, an interpolation-based one-pass Chase decoder is proposed for BCH codes. By using of the binary property of BCH codes, a low-complexity method is developed to select the interpolation output leading to successful decoding. The code word recovery step is also significantly simplified. From architectural analysis, the proposed decoder with $\eta = 4$ for a (4200, 4096) BCH code has higher efficiency in terms of throughput-over-area ratio than the prior one-pass Chase decoder based on the Berlekamp's algorithm

Index Terms— BCH codes, Chase decoding, interpolation.

I. INTRODUCTION

BCH CODES can be found in many applications, including flash memory, optical transport network, and digital video broadcasting. For a BCH code with minimum distance $d_{\min}$, traditional hard-decision decoding (HDD) algorithms, such as the Berlekamp's algorithm, can correct $t = d_{\min}/2$ errors. Through flipping the n least reliable bits and trying 2^{η} test vectors, the soft-decision Chase algorithm can correct up $t + \eta$ errors. It also has better error-correcting performance than the generalized minimum distance (GMD) decoder and the soft decision decoder in [1], which assumes that all but one error are located in the $2t$ least reliable bits.

To reduce the complexity of the Chase BCH decoding, one pass schemes have been proposed to derive the error locators for all test vectors in one run based on the Berlekamp's algorithm [2], [3]. Further simplifications and implementation architectures of this scheme were developed in [4]. It was observed that selecting the error locator corresponding to the correct code word requires expensive parallel Chien search and accounts for a majority part of the overall decoder.

This brief proposes a novel interpolation-based Chase BCH decoder. By making use of the binary property of BCH codes, substantial modifications and simplifications are developed in this brief. In particular, instead of employing expensive parallel Chien search, the selection of the interpolation output leading to successful decoding is achieved by simple evaluation value computation without any error-correcting performance or code rate loss.

In addition, the recovery of each code word bit is done through testing the evaluation values of two low-degree polynomials. From architectural analysis, the proposed decoder with $\eta = 4$ for an example (4200, 4096) BCH code has 2.3 times higher efficiency in terms of throughput-over-area ratio than the Chase decoder based on the Berlekamp's algorithm [4], while achieving the same error-correcting performance.

The structure of this brief is as follows. Section II introduces the Chase decoding. The interpolation-based Chase BCH decoding and its implementation architectures are proposed in Sections III and IV, respectively. Section V provides the hardware complexity analysis, and Simulation results and conclusions are drawn in Section VI and VII respectively.

II. CHASE DECODING

A t-error-correcting (n, k) BCH code over finite field $GF(2p)$ $(p \in Z+)$ is considered in this brief. In the Chase algorithm, 2^{η} test vectors are formed by flipping each of the η least reliable bits in the received word, and decoding is done for each vector. As a result, up to $t + \eta$ errors can be corrected. It can be observed, that the Chase decoding can significantly outperform both the GMD and soft-decision decoding in [1]. Carrying out the decoding for each test vector separately in the Chase algorithm leads to high hardware complexity.

Instead, the error locator polynomials of all test vectors can be
derived using the one-pass schemes in [2] and [3] for BCH decoding. These schemes find the error locator for the first test vector using the Berlekamp's algorithm. Then, the error locator for each successive vector is computed by updating the polynomials derived from the Berlekamp's algorithm. To decide which error locator corresponds to the correct code word, the Chien search is employed previously to select the one whose root number equals its degree.

Binary BCH codes can be considered as subfield sub-codes of RS codes. Interpolation-based one-pass Chase RS decoders have been developed. For an (n, k') RS code, the symbols of a codeword c can be considered as the evaluation values of a degree $k'-1$ message polynomial, $f(x)$, over n distinct nonzero finite field elements $\alpha_0, \alpha_1 \ldots \alpha_{n-1}$.

Assume that the hard decision of the received vector is r. A polynomial $Q(x, y)$ that passes each point, (α_i, r_i), with $(1, k'-1)$ minimum weighted degree can be found through interpolation. If the number of errors does not exceed $(n - k')/2$, then $Q(x, y)$ would have a factor $y-f(x)$, and r is decoded.

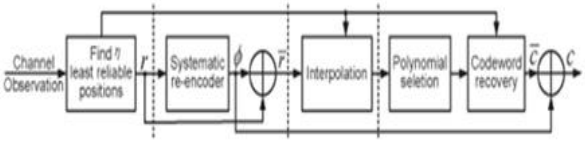

Fig. 1. Re-encoded interpolation-based Chase decoder.

The interpolation-based Chase decoder can be implemented according to Fig. 1. First, a code word φ is derived by applying systematic encoding on the last k' symbols in r. Then, $\bar{r} = r + \varphi = (c + \varphi) + e = \bar{c} + e$ is another code word, $\bar{c}$, corrupted by the same error vector e, and the decoding is done on $\bar{r}$ instead. This follows the re-encoding technique introduced in [5].

The interpolation problem can be solved by the Koetter's algorithm [6]. For high-rate codes, it starts with a pair of polynomials $\{1, y\}$ and iteratively forces them to pass one more point each time with minimum increase in the weighted degree. At the end, the polynomial with lower weighted degree is the interpolation output. Denote the set of the last k' code positions, also called the systematic positions, by S'.

Since $\bar{r} = 0$ for $i \in S'$, the interpolation over the points in these positions can be presolved as $v(x) = \pi i \in S' (x + \alpha i)$, and this factor is taken out of the interpolation process using a coordinate transformation. As a result, the real interpolation only needs to be done on the remaining $n - k'$ points. The test vectors in the Chase algorithm can be arranged in a Gray code manner, i.e., adjacent vectors only have a pair of different points (α_i, r_i) and (α_i, r'_i).

The unified backward–forward interpolation [7] eliminates (α_i, r_i) from a given interpolation result and adds (α_i, r'_i) in one iteration. Accordingly, the interpolation results for all test vectors are computed in one run. Moreover, when a code position in S' needs to be flipped, the coordinate transformation and interpolation can be modified using the methods in [8].

Compared to the schemes based on the Berlekamp's algorithm, the interpolation-based scheme leads to more efficient Chase decoders for RS codes. However, the application of this scheme for BCH codes has not been studied. In the next section, an efficient one-pass Chase BCH decoder based on interpolation is proposed.

III. Interpolation-Based Chase BCH Decoder

An (n, k) t-bit error-correcting BCH code over GF(2^p) is a subfield subcode of an (n, k') t-symbol error-correcting RS code over GF(2^p). In another word, all the (n, k) BCH code words form a subset of the (n, k') RS code words. $n - k' = 2t$, and $n - k$ is equal to or slightly less than pt.

The interpolation based decoding is developed based on the interpretation that the code word symbols are evaluation values of the message polynomial. BCH codes cannot be encoded this way since the evaluation values of a binary message polynomial over finite field elements are usually not binary. Hence, BCH code words are considered as RS code words in order to apply the interpolation based decoding.

Applying RS systematic re-encoding to the last k' code positions, $n - k'$ points remain to be interpolated for each test vector. The same backward–forward interpolation scheme [7], [8] can be adopted to derive the interpolation results of all vectors in one run. Nevertheless, by making use of the property that r is binary in BCH decoding, substantial simplifications can be made on the polynomial selection and code word recovery steps.

Therefore, there are $2pk'$ code words for the $(n,$ $)$ RS code over GF $(2p)$. However, only a small proportion, $2k$ of them, are also code words of the (n, k) binary BCH code. Hence, the chance of returning a binary BCH code word is extremely small if the test vector is undecodable, particularly for long codes. Inspired by this, the interpolation output polynomials can be selected based on whether they will lead to binary code words.

Nevertheless, testing if each symbol in the decoded word is binary requires the code word recovery step to be completed first. Instead, we propose to check only a few symbols that are easy to compute from the interpolation output. Which symbols to test depends on the specifics of the computations used in the decoding.

Adopting the re-encoding technique, the decoding is actually carried out on $\bar{r} = r + \varphi$. Then, the returned code word $\bar{c}$ is added up with φ to compute the code word c. $\varphi i = ri$ is binary for $i \in S'$. In addition, using the code word recovery scheme in [8], $\bar{r}i$ for $i \in S'$ is zero unless the corresponding code position is a root of $q1(x)$. $q1(x)$ has at most t roots and $t << k'$ for highrate codes. Therefore, c_i for $i \in S'$ is mostly binary and cannot be used to tell whether the entire c is binary.

On the other hand, $\bar{c}i$ and φi for $i \in \bar{c}'$ are mostly nonbinary for undecodable cases, and the chance that $\bar{c}i + \varphi i$ $(i \in \bar{c}')$ is binary is extremely small. Here, $\bar{s}'$ denotes the first $n - k'$ code positions. Accordingly, we propose to select the interpolation output whose corresponding $\bar{c} + \varphi$ is binary in the first two symbols. φ is available, and the two symbols of $\bar{c}$ can be easily computed from the interpolation output.

In our simulations, the test vectors are ordered according to reducing reliability as much as possible, and the first interpolation output passing the test is selected.

IV. VLSI ARCHITECTURES

Systematic RS re-encoding can be done by linear feedback shift registers (LFSR's), and an efficient parallel architecture is available in [9]. The modified unified backward–forward interpolator in [8] is among the most efficient and is adopted in the proposed decoder.

The proposed polynomial selection tests whether $\bar{f}(1) + \varphi 0$ and $\bar{f}(\omega) + \varphi 1$ are binary. $\bar{f}(1)$ and $\bar{f}(\omega)$ can be derived using (1). As aforementioned, $v(1)$ and $v(\omega)$ are precomputed and are nonzero and can be computed on the fly using an adder and a multiplier–register loop in at most 2η clock cycles.

The architecture in Fig. 2 tells whether $f(\omega) + \varphi 1$ is binary, and $\bar{f}(1) + \varphi 0$ can be tested by a similar architecture, in which the constant multipliers in the four feedback loops are replaced by hard-wiring since they implement multiplications by one. The unified interpolator in [8] outputs the coefficients of $q0(x)$ and $q1(x)$ serially. $q0(\omega)$ and $q1(\omega)$ are computed by the first and third feedback loops in Fig.2 and are available after $\max(\deg(q0(x)), \deg(q1(x)))$ clock cycles.

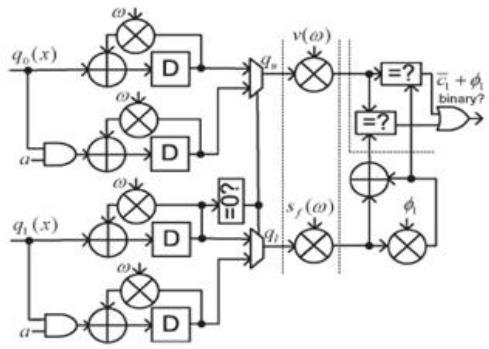

Fig. 2. Architecture for proposed polynomial selection.

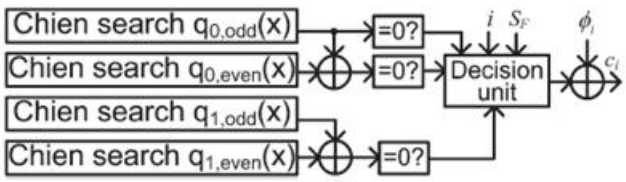

Fig. 3. Architecture for BCH code word recovery.

It is possible that $q1(\omega) = 0$. In this case, $q0(\omega)$ is also zero, and the L'Hopital rule needs to be applied. It says that, if $d(x) = a(x)/b(x)$ and $a(\alpha i) = b(\alpha i) = 0$ for a certain αi, then $d(\alpha i) = a'(x)/b'(x)|\alpha i$. The derivative of a polynomial over finite fields of characteristic two is the collection of the odd-degree terms.

With the help of the signal "a" that flips every clock cycle, $\omega q'\ 0(\omega)$ and $\omega q'\ 1(\omega)$ are computed by the second and fourth loops in Fig. 2, respectively. $\omega q'\ 1(\omega)$ is nonzero since $(x + \omega)$ can only be a simple factor of $q1(x)$, and the common scaler ω does not affect the value of $f(\omega)$. Denote the outputs of the two multiplexors in Fig. 2 by qu and ql.

The testing can be done equivalently as whether $qu\ v(\omega) + ql\ sf(\omega)\varphi 1$ equals zero or $ql\ sf(\omega)$ as shown in Fig. 2 to avoid expensive finite field inverters.

The architecture in Fig. 4 implements BCH code word recovery according to (3). The evaluation values of $q1(x)$ and $q0(x)$ over the k code positions in S are computed through the Chien search. Similarly, the polynomial coefficients are divided into even and odd parts so that the evaluation values Of $q'_0(x)$ are also available. Then, whether $\bar{c}_i$ is '1' or '0' is decided based on the evaluation values and if $i \in SF$. Such a decision unit can be implemented by very simple hardware. To achieve high speed, the Chien search can be implemented by the parallel architecture in [10].

V. HARDWARE COMPLEXITY ANALYSIS

Taking a $t = 8$ (4200, 4096) BCH code over GF(213) as an example, the hardware complexity of the proposed interpolation based Chase BCH decoder with $\eta = 4$ is analyzed and listed in Table I.

TABLE I
HARDWARE COMPLEXITIES OF BCH DECODERS FOR A (4200, 4096) CODE OVER $GF(2^{13})$

	GF Mult.	GF Add.	GF Inv.	GF Squarer	Const. Mult.	Multiplexor (bit)	Memory (bit)	Register (bit)	# of clks	Total equivalent gate count (XOR)
Systematic re-encoder	1	321	0	0	320	0	208	208	230	-
Interpolator	44	42	2	0	2	352	1088	512	228	-
Proposed polynomial selection	10	22	0	0	12	104	0	624	12	-
Proposed codeword recovery	0	323	0	0	320	208	0	949	217	-
Pipelining RAM	0	0	0	0	0	0	13010	0	-	-
Interpolation-based Chase decoder	55	708	2	0	654	664	14306	2293	230	75540
Chase decoder based on Berlekamp's [4]	17	2987	1	9	2976	33891	13146	5551	207	191891
GMD decoder based on Berlekamp's	51	2044	3	16	2000	18876	13666	4628	200	146903
Soft-decision decoder [1]	23	135	0	17	9	1209	106496	2288	69743	124361
HDD	2	545	0	8	544	117	8400	1183	128	34637

Fig. 4. Table 1.

This code is considered as a subfield sub code of a (4200, 4184) RS code over GF(213) in the decoding. To increase the throughput, pipelining is applied along the dashed lines in Fig. 1. Each pipelining stage should take similar number of clock cycles to improve the hardware utilization efficiency. This is achieved through adjusting the parallel processing factors used in the functional blocks. The

critical path of each block is designed not to exceed one finite field multiplier, one adder, and one multiplexor.

The unified backward–forward interpolator in [8] for the systematically re-encoded decoder processes polynomial coefficients serially.

The number of clock cycles spent for an interpolation iteration is $dx + \xi$, where dx is the maximum x- degree of the polynomials during that iteration and ξ is the pipelining latency of the interpolator.

The worst case interpolation latency happens when all the 2η test vectors need to be interpolated. In this case, dx increases from 1 to $(n - k')/2$ gradually during the $n - k'$ forward interpolation iterations for the first test vector and remains at $(n - k')/2$ during the backward–forward interpolation iterations for later vectors.

To further increase the interpolation speed, multiple copies of the interpolator can be employed. When two copies are used, each copy needs to take care of eight test vectors. Accordingly, the worst case interpolation latency is ((1 + 3) + (8 + 3)) × 8/2 × 2 + (8 + 5) × 7 + 1 + 2 + 2 + 3 + 3 + 3 + 3 = 228 clock cycles.

To match the speed of the interpolation, the parallel architecture in [9] can be adopted to implement the systematic RS reencoder. In addition, the interpolation points in SF and s' need to be modified according to the coordinate transformation. The modifications on the points in SF can be done simultaneously as the systematic re-encoding using one multiplier and one adder [8]. The re-encoding and coordinate transformation are completed in [4184/20] + 16 + 4 = 230 clock cycles.

Each backward–forward interpolation iteration takes as short as 13 clock cycles. The polynomial selection should be done in the same amount of time to avoid slowing down the decoder. max{deg($q1(x)$), deg($q0(x)$)} = (n –)/2 = 8 in the backward forward iterations, and the data paths in Fig. 2 are divided into four pipelining stages. Hence, the computation of $c1 + \varphi 1$ can be finished in 12 clock cycles, and another copy of the architecture is used to compute $c_0 + \varphi_0$ in parallel.

Considering the latency of the polynomial selection, the proposed code word recovery should be finished in about 228−12= 216 clock cycles. Accordingly 4096/216= 19 parallel processing is adopted in the Chien search engines in Fig. 3, and 19 copies of the other units are employed.

A GF (213) multiplier can be implemented by 192 XOR gates and 169 AND gates, and a squarer has 20 XOR gates. On average, a constant multiplier over GF (213) needs 40 XOR gates. An inversion in GF (213) can be completed in 13 clock cycles using a squarer–multiplier–register loop.

Each AND or OR gate has 3/4 the area of an XOR, and each multiplexor or storing a bit in memory requires the same area as an XOR. Also, each register takes about three times the area of an XOR. Using these assumptions, the total equivalent gate counts of the BCH decoders are computed and listed in TableI.

Compared to the Chase BCH decoder based on the Berlekamp's algorithm [4], the proposed interpolation-based Chase decoder can achieve around 2.3 times higher efficiency in terms of throughput-over-area-ratio, while having the same error-correcting performance.

V. SIMULATION RESULTS

The proposed circuit is modeled using verilog language and simulated on Xilinx ISE DESIGN suite 14.3 and implemented on Xilinx Spartan-3e FPGA device. Fig.5. shows the simulated waveforms.

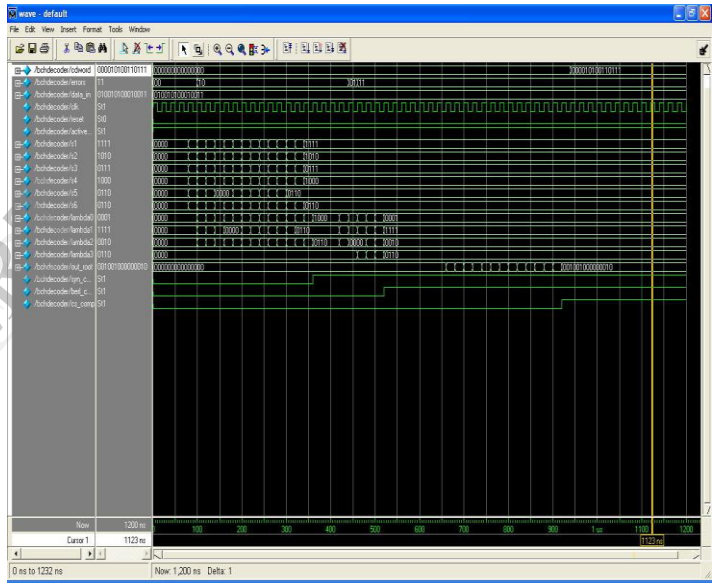

Fig. 5. Simulation Result of Decoder

VI. CONCLUSION

This brief developed an efficient interpolation-based one pass Chase BCH decoder. By making use of the binary property of BCH codes, novel polynomial selection and code word recovery schemes were proposed. In particular, the proposed polynomial selection led to significant complexity reduction without sacrificing the error-correcting performance. Future work will be directed to further speeding up the interpolation based BCH decoder without incurring large area overhead.

REFERENCES

[1] Y.-M. Lin, H.-C. Chang and C.-Y. Lee, "An improved soft BCH decoder with one extra error compensation," in Proceedings of the IEEE International Symposium on Circuits Systems., May 2010, pp. 3941–3944.
[2] Y. Wu, "Fast Chase decoding algorithms and architectures for Reed–Solomon codes," IEEE Transactions on Information Theory, Jan. 2012.
[3] N. Kamiya, "On algebraic soft-decision decoding algorithms for BCH

codes," IEEE Transactions on Information Theory, Jan. 2001.
[4] X. Zhang, J. Zhu, and Y. Wu, "Efficient one-pass Chase soft-decision BCH decoder for multi-level cell NAND flash memory," in Proceedings of the IEEE International Midwest Symposium Circuits Systems, Aug. 2011.

[5] W. J. Gross, F. R. Kschischang, R. Koetter, and P. Gulak, "A VLSI architecture for interpolation in soft-decision decoding of Reed- Solomon codes," in Proceedings of the IEEE Workshop Signal Processing in Systems., Oct. 2002.
[6] R. Koetter, "On algebraic decoding of algebraic-geometric and cyclic codes," Ph.D. dissertation, Dept. Elect. Eng., Linköping Univ., Linköping, Sweden, 1996.
[7] X. Zhang and J. Zhu, "Algebraic soft-decision decoder architectures for long Reed-Solomon codes," *IEEE* Transactions *Circuits Systems II* Oct. 2010.
[8] X. Zhang and Y. Zheng, "Systematically re-encoded algebraic soft decision Reed-Solomon decoder," *IEEE* Transactions *Circuits Systems,* Jun. 2012.
[9] C.-S. Choi and H. Lee, "High throughput four-parallel RS decoder architecture for 60 GHz mm WAVE WPAN systems," Proceedings of the IEEE International *Conference . Electronics,* 2010.
[10] K. Lee, H.-G. Kang, J.-I. Park, and H. Lee, "A high-speed low complexity concatenated BCH decoder architecture for 100 Gb/s optical communications," *J. Signal Processing. Systems,* Jan. 2012

[11] Web source: the error correcting page
[12] Web source: NPTEL
[13] Error control coding by Shu Lin, Daniel J Costello
[14] Fundamentals of Error correcting codes by Cary Huffman and vera pless
[15] Art of Error correcting codes by Robert, Zarangoza

FPGA Implementation of 2D FFT and its Inverse for Image Compression

MANJUNATH K
Department of Electronics and communication Engg.
PESCE, Mandya.

MAHESH GOWDA N M
Assistant Professor,
Department of Electronics and communication Engg.
PESCE, Mandya.

Abstract— Digital Signal Processing Systems have wide range of application in the image processing, voice processing , radars and bio-data signal compression. Basically these applications are developed using fast fourier and inverse fourier transform concepts. The computational complexity associated with these FFT and IFFT algorithm is very high and also speed is one of the important factor we should consider while providing the VLSI implementation. This paper present a new algorithm for 2D FFT and IFFT that overcome the irregularities that present in the butterfly structure VLSI implementation. The design is coded and implemented on Spartan 3 FPGA and it is tested for 2D color images.

Keywords-Discrete Fourier Transform,Fast Fourier Transform ,twiddle factor.

INTRODUCTION

The field of Digital Signal Processing (DSP) has grown enormously in the past decade and is playing a significant role in driving the technology. The DSP applications reach out to our everyday life such as medicine, surveillance, authentication and many more areas. In all these applications, Discrete Fourier has been widely used for efficient implementations. The Fast Fourier Transform is an efficient algorithm to compute the Discrete Fourier Transform (DFT) fast and its inverse. The evaluation of both produces same results but FFT is much faster. The DFT involves N2 complex multiplications and N(N-1) complex additions, where N is 8 for 8x8 pixels block of an image. As the value of N increases, the number complex calculations also increase resulting in high processing time.

A Fast Fourier Algorithm (FFT) was discovered in 1965 by Cooley and Tukey, which reduced the number of calculations drastically and paved the way for real time processing of discrete signals which revolutionized the field of Digital Signal Processing [1]. Most of FFT algorithrns decompose the overall N-point DFT into successively smaller and smaller expressions popularly known as butterfly structure. The FFT algorithrns achieve its computational efficiency through divide and conquer strategy. The fundamental principle of FFT algorithm is that of dividing the computation of DFT sequence of length N into successively smaller DFTs by exploiting the Symmetry property and Periodicity property of DFT. The FFT reduces the number complex multiplications to (N/2) log N_2 and additions to N $\log_2$N. Thus there is a large reduction in the calculation which makes real time processing a reality. Further, the butterfly structure makes hardware realizations difficult since regularity of expressions are broken.

This Paper presents a novel algorithm that can overcome the irregularity encountered in the butterfly structural implementations in VLSI realization. In contrast to the butterfly structure, the proposed algorithm is highly regular and hence suitable for VLSI implementations. The proposed design incorporates a high degree of pipelining and massive parallelism and hence offers high throughputs. Section I presents the basics of Discrete Fourier Transforms. The regular FFT/IFFT algorithm outlined earlier is proposed in Section II. The quantitative and qualitative analysis of the algorithm is summarized and tabulated in Section III successively.

A. 1-D Discrete Fourier Transform Pair

One dimensional signal is described using function that depends on one independent variable with reference to time. Audio signals, radar signals and biomedical signals are examples of one dimensional signal. A transform changes one domain value such as time in to frequency components and makes the signal processing far more easily than the time domain components.

The Discrete Fourier Transform for a 1D signal is given by:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \qquad (1)$$

for n = 0, 1, 2, ……, N-1.

The Inverse Fourier Transform is given by

$$x(n) = 1/N \sum_{k=0}^{N-1} X(k) W_N^{-nk} \qquad (2)$$

for k = 0, 1, 2, ……., N-1.

B. 2-D Discrete Fourier Transform Pair

Two dimensional signals is described using function that depends on two variables. Images are examples of two dimensional signals. The Discrete Fourier Transform for a 2D signal is given by the expression:

$$F(u,v) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} f(x,y) W_N^{(ux+vy)} \qquad (3)$$

for k = 0, 1, 2, ……..N-1.

The Inverse Fourier Transform is given by

$$f(x,y) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} F(u,v) W_N^{-(ux+vy)} \qquad (4)$$

for n = 0, 1, 2, ……., N-1.

Fast Fourier transform algorithrns resorts to a variety of tricks for reducing the time required to compute a DFT [2]. Several methods for computing FFT and IFFT are discussed in Ref. [1]. Decimation In Time (DIT) divides a sequence x (n) in to odd and even sequences in successive stages to realize the whole DFT sequence. Likewise, Decimation In Frequency (DIF) divides the number of frequency components X(k) in to odd and even components to get the DFT in frequency domain [3]. FFT helps to transform the signal from time domain to frequency domain, where filtering and correlation can be performed with fewer operations [4].

A Radix sorting is a fast and stable algorithm that sorts keys, say, numerical digits. The processing of keys begins at the Least significant Digit and proceeds to the Most Significant Digit. In DFT applications, the radix-n algorithm divides a DFT of size N in to n interleaved DFTs of size N/n with each recursive stages. Radix 2 first divides the DFT of size N in to two interleaved DFTs of size N/2 and then computes the DFT of even indexed inputs X2m and odd indexed input X2m+l. These resuIts are combined to produce the DFT of the sequence [5]. The implementation of Radix 2 algorithm is centered on conjugating the twiddle factors of the corresponding forward FFT. It has single path delay feedback pipelined processor [6]. The Radix 2 algorithrns are simplest FFT algorithms. The Radix 2 DIT FFT can be applied recursively to the two length N/2 DFTs to save computation and it has to be successively applied to reach DFTs of length-2 [5]. Floating point arithmetic has been virtually impossible to implement on FPGA based systems due to the inherent complexity of floating point arithmetic. With the introduction of high level languages such as VHDL, rapid prototyping of floating point formats has become possible making such complex structures more feasible to implement [7]. For this a VHDL library has to be formed which includes addition, subtraction, multiplication and divisionmodules. This situation inevitably utilizes large amount of FPGA resources.

Most of the research to date for the implementation of FFT algorithrns has been performed using general purpose processors, Digital Signal Processors (DSPs) and dedicated FFT Processors [3]. However, as Field Programmable GateArrays (FPGAs) have grown in capacity, improved in performance, and decreased in cost, they have become a viable solution for performing computationally intensive tasks, with the ability to tackle applications for custom chips and programmable DSP devices [8].

I. Proposed Regular, FFT Algorithm for Image Processing

Twiddle Factor is a key component in FFT/IFFT computation. It is any of the trigonometric constant coefficients that are multiplied by the data in the course of the FFT algorithrn. They are the coefficients used to combine results from a previous stage to form inputs to the next stage. Twiddle factor term was apparently coined by Gentleman and Sande in 1963. It is the root of unity complex multiplication constants in the butterfly operations and also it describes the rotating vector which rotates in increments according to the number of samples. The twiddle factor is expressed as

$$W_N = e^{-j(2\pi nk)/N} \qquad (5)$$

By Euler's formula, the exponential term can also be written as

$$e^{-j\theta} = \cos\theta - j\sin\theta \qquad (6)$$

The exponential term (j2rrnk/N) of the DFT equation may be written using the Euler's formula:

$$e^{-j\theta} = \cos(2\pi nk/N) - j\sin(2\pi nk/N) \qquad (7)$$

The RHS of the Euler's equation is expressed as (m*n) matrix forms for various values of (u, x) and (v, y) as Sine and cosine terms. The matrices and their transposes may be stored as lookup tables on ROMs and accessed block by block for hardware implementation [9].

As we have observed in the FFT equation, it consists of complex additions, complex multiplications and twiddle factors irregularities to compute the FFT. Also in hardware realization, computation speed is of utmost importance and hence the algorithm should be amenable for parallelism and pipelining to speed up the computation.

The irregularities of the twiddle factors in FFT can

also be overcome by the Cosine and Sine transforms of the signal. The FFT for 2D signal is obtained by adding the Sine and Cosine transforms of the input signal. The transforms in this algorithm is obtained in matrix form by varying the values of u and x from 0 to 7. The transposed Sine and Cosine matrices are obtained by varying the values of v and y from 0 to 7.The required Cosine matrix and Sine matrix for FFT and IFFT are obtained as shown in Eq. (8) and Eq. (9)

$$C(u+1,x+1) = \cos((pi/4)*(u*x)) \quad (8)$$

$$S(u+1,x+1) = \sin((pi/4)*(u*x)) \quad (9)$$

The proposed, regular, 2-D FFT can be obtained by subtracting the Cosine and Sine transform as per Eq. 10

$$F(u,v) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} \{F(u,v)(\cos(2\pi(ux+vy)/N) + j\sin(2\pi(ux+vy)/N))\} \quad (10)$$

for k = 0, 1, 2...., N-1.

Similarly, the IFFT for 2-D is computed by subtracting the Cosine and Sine transform of the FFT of the signal and maybe expressed as:

$$f(x,y) = (1/MN)\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \{F(u,v)(\cos(2\pi(ux+vy)/N) - j\sin(2\pi(ux+vy)/N))\}$$

for n = 0, 1, 2...., N-1 (11)

A. Proposed Regular FFT/IFFT Algorithm

The proposed FFT/IFFT algorithm is also folIows:

1. Read the input file from the host system.
2. Evaluate the (8x8) Cosine and Sine matrix using Equations 8 and 9.
3. The image is accessed as (8x8) block successively, and the Cosine and Sine transforms are obtained.
4. Compute the FFT of the image using Eq. 10.
5. Verify the obtained FFT values with buiIt in FFT functional values of MATLAB.
6. Compute the IFFT of the processed image in step 4 using the Eq. 11.
7. Verify visually the reconstructed image with the original image.
8. Calculate the Power Signal to Noise Ratio (PSNR) of the processed signals for validation. Note: A PSNR value of 35 dB and above implies the reconstructed image is indistinguishable from the original image.

II. Experimental Results and Comparative Study

Due to the importance and use of FFT in many signal and Image processing applications, a novel FFT and IFFT algorithm has been presented in this paper. This algorithm exploits the Regularity, Parallelism and Pipelined architecture for the implementation.

The FFT and IFFT are coded in MATLAB in order to establish the correct working of the proposed algorithrn. It also serves as a reference for checking the hardware results after the algorithm is implemented on FPGA.

The algorithm is designed using MATLAB (R2008a)and applied on to various images. MATLAB is considered as industry standard for this field of research [9]. The reconstructed images are of good quality with PSNR value

Table 1. Reconstructed Quality of SampIe Images using the Proposed FFT/IFFT Algorithrns

2-D (color images)			
Input File	**File Type**	**File Size (pixels)**	**PSNR (dB)**
Dolphins	bmp	532x327	39
Luvre-Paris	bmp	805x531	39
Seminar	jpg	1312x1000	38
Volcano	bmp	528x361	38
Tree	bmp	483x812	35
Cleveland	bmp	806x550	35

a

b

a

b

better than 35 dB. Table 3 shows the output quality of the proposed Regular FFT/IFFT algorithm. Figures 1 and 2 show the sample reconstructed images.

III. CONCLUSION

A new algorithm has been presented in this paper to compute FFT and IFFT of the image signals. The proposed algorithm has been coded in Matlab and verilog. The design is synthesized and implemented on spartan 3 FPGA., and the reconstructed images are good with a PSNR value of 35 dB or above.

REFERENCES

[1] Alan V. Oppenheim, Ronald W. Schafer, John R. Buck,\ Discrete -Time Signal Processing, Prentice Hall, Second Edition, pp. 646-652, 1999.
[2] L. Rabiner B. Gold, Theory and Application 0/ Digital Signal Processing, Prentice-Hall, 1975.
[3] Sneha N. Kherde, Meghana Hasamnis, "Efficient Design and Implementation of FFT", International Journal of Engineering Science and Technology (IJEST), ISSN : 0975-5462, NCICT Special Issue Feb. 2011.
[4] The Design and Implementation of FFT Algorithm Based on the Xilinx FPGA IP Core, The 2nd International Conference on Computer Application and System Modeling (2012).
[5] Shaminder Kaur, "FPGA Implementation of OFDM Transceiver using FFT Algorithm", International Journal of Engineering Science and Technology (IJEST) Vol. 4, No. 04 April 2012.
[6] Asmita Haveliya, "Design and simulation of 32-point FFT using Radix-2 algoritm for FPGA implementation", 2012 Second International Conference on Advanced Computing & Communication Technologies.
[7] Ahmed Saeed, M. Elbably, G. Abdelfadeel, M. I. Eladawy, "Efficient FPGA implementation of FFT/IFFT Processor", International Journal of Circuits, Systems and Signal Processing, Volume 3, 2009.
[8] Nabeel Shirazi, M. Athansa and A.Lynn abbott, "Implementation of a 2-D Fast Fourier Trasform on FPGA based Custom Computing Machine", Proceedings of 1th Reconfigurable Architecture Workshop, Denver 2005.
[9] Taoufik SAIDANI, Yahia SAID, Mohamed ATRI Rached Tourki, "Real Time FPGA acceleration for Discrete Wavelet Transform of the 5/3 Filter for JPEG 2000", SETIT 2012, IEEE, pp. 21-24, March 2012.
[10] I.S.Uzun, A.Amira A.Bouridane, FPGA Implementations of Fast Fourier Transforms for Real-Time Signal and Image Processing.
[11] S. Ram ach an dran, "Digital VLSI Systems Design",Springer 2007.