A Novel VLSI DHT Algorithm for A Highly Modular And Parallel Architecture
PALLALA PRANAYKUMAR1, D.SREELAKSHMI2

1PG Scholar, Dept of ECE, ASCET, Gudur, Andhra Pradesh, India, E-mail: pranay441@gmail.com.
2Assoc Prof, Dept of ECE, ASCET, Gudur, Andhra Pradesh, India, E-mail: doma.sreelakshmi@audisankara.com.

Abstract: A new very large scale integration (VLSI) algorithm for a 2N-length discrete Hartley transform (DHT) that can be efficiently implemented on a highly modular and parallel VLSI architecture having a regular structure is presented. The DHT algorithm can be efficiently split on several parallel parts that can be executed concurrently. Moreover, the proposed algorithms' well suited for the sub expression sharing technique that can be used to significantly reduce the hardware complexity of the highly parallel VLSI implementation. Using the advantages of the proposed algorithm and the fact that we can efficiently share the multipliers with the same constant, the number of the multipliers has been significantly reduced such that the number of multipliers is very small comparing with that of the existing algorithms. Moreover, the multipliers with a constant can be efficiently implemented in VLSI. Image compression is largely possible by exploiting various kinds of redundancies which are typically present in an image. The extent of redundancies may vary from image to image. Image compression algorithms aim to remove redundancy in data in a way which makes image reconstruction possible. This basically means that image compression algorithms try to exploit redundancies in the data, they calculate which data needs to be kept in order to reconstruct the original image and therefore which data can be “thrown away. Image compression and coding techniques explore three types of redundancies: coding redundancy, inter pixel (spatial) redundancy, and psycho visual redundancy. Coding redundancy consists in using variable-length code words selected as to match the statistics of the image itself or a processed version of its pixel values. This type of coding is always reversible and usually implemented using look-up tables (LUTs).

Keywords: Discrete Hartley Transform (DHT), DHT Domain Processing, Fast Algorithms.

I. INTRODUCTION

The Discrete Fourier transform (DFT) is used in many digital signal processing applications as in signal and image compression techniques, filter banks, signal representation, or harmonic analysis. The discrete Hartley transform (DHT) can be used to efficiently replace the DFT when the input sequence is real. In the literature, there are some fast algorithms for the computation of DHT and some algorithms for the computation of generalized DHT. There are also several split-radix algorithms for computing DHT with a low arithmetic cost. Thus, Sorensen et al. and proposed split-radix algorithms for DHT with a low arithmetic cost proposed another split-radix algorithm where the odd-indexed transform outputs are computed using an indirect method. The classical split-radix algorithm is difficult to implement on VLSI due to its irregular computational structure and due to the fact that the butterflies significantly differ from stage to stage. Thus, it is necessary to derive new such algorithms that are suited for a parallel VLSI system.

There are also in the literature several fast algorithms that use a recursive strategy as that for discrete cosine transform (DCT) and that in for generalized DHT. Since DHT is computationally intensive, it is necessary to derive dedicated hardware implementations using the VLSI technology. One category of VLSI implementations is represented by systolic arrays. There are many systolic array implementations of DHT. Systolic array architectures are modular and regular, but they use particularly pipelining and not parallel processing to obtain a high-speed processing. In the literature, highly parallel solutions as those in and were also proposed. In, a highly parallel and modular solution for the implementation of type-III DHT based on a new VLSI algorithm is proposed. In, we have a highly parallel solution for the implementation of DHT based on a direct implementation of fast Hartley transform (FHT). It is worth to note that hardware implementations of FHT are rare. Multipliers in a VLSI structure consume a large portion of the chip area and introduce significant delays. This is the reason why memory-based solutions to implement multipliers have been more and more used in the literature. To efficiently implement multipliers with lookup-table-based solutions, it is necessary that one operand to be a constant. When one of the operands is constant, it is possible to store all the partial results in a ROM, and the number of memory words is significantly reduced from $2^{2N}$ to $2^N$.

In this brief, a new VLSI DHT algorithm that is well suited for a VLSI implementation on a highly parallel and
modular architecture is proposed. It can be used for
designing a completely novel VLSI architecture for DHT.
Moreover, using sub expression sharing technique and
sharing the multipliers with the same constant, the hardware
complexity can be significantly reduced the number of
multipliers being very small, significantly less than that in.
In the proposed solution, we have used only multipliers with
a constant that can be efficiently implemented in VLSI. The
proposed solution is not only appealing by its high level of
parallelism and by using a modular and regular structure but
it can be also used to obtain a small hardware complexity by
extensively sharing the common blocks. The rest of this
brief is organized as follows. In Section II, we present a
DHT Transform. In Section III, we present a Look-Up-
Table. In Section IV, Simulation Results and The
closure and future works are presented in Section V.

II. DHT TRANSFORM
The Hartley transform is an integral transform closely
associated with the Fast Fourier Transform; however that
transforms real-valued functions to real-valued functions, it
absolutely was planned as an alternate to the Fourier
transform by R. V. L. Hartley in 1942, and is one in all
several noted Fourier-related transforms. Compared to the
Fourier transform, the Hartley transform has the benefits of
rewriting real functions to real functions and of being its
own inverse. An FFT could be a thanks to work out
identical result more quickly: computing the DFT of N
points within the naive manner, victimization the definition,
takes O(N2) arithmetic operations, whereas a FFT will work
out an equivalent DFT in exactly O(N log N) operations.
The distinction in speed is huge, particularly for long
knowledge sets wherever N could also be within
thousands or millions. FFTs are of nice importance to a
decent type of applications, from digital signal method and
finding partial differential equations to algorithms for fast
multiplication of enormous integers within the presence of
round-off error, several FFT algorithms are rather more
correct than evaluating the DFT definition directly.

A. Forward and Inverse Transform
The forward and inverse discrete Hartley transform pair is
given by
\[
X_H(k) = \sum_{n=0}^{N-1} X(n) \cos \frac{2\pi nk}{N} \tag{1}
\]
\[
X(n) = \frac{1}{N} \sum_{k=0}^{N-1} X_H(k) \cos \frac{2\pi nk}{N} \tag{2}
\]
Where \( \cos \phi = \cos \phi + \sin \phi \).

\[
H(u,v) = \frac{1}{MN} \sum_{X,Y} \{ f(x,y) \cos(2\pi \frac{X}{M} + \frac{Y}{N}) + \sin(2\pi \frac{X}{M} - \frac{Y}{N}) \} \tag{3}
\]

Where \( f(x, y) \) is the intensity of the pixel at position\((x, y)\)
\( H(u,v) \) is the value of element in frequency domain. The
results are periodic and the cosine+sine (CAS) term is call
“the kernel of the transformation” (“or” basis function”).

The FHT are also used for fast arithmetic operation. In
Fast Hartley Transform (FHT) \( M \) and \( N \) must be power of 2.

Which are much faster than DHT? These equation form
representation is
\[
H(u,v) = \left\{ T(M-u,v) + T(u,N-v) - T(M-u,N-v) \right\} / 2 \tag{4}
\]

For both the FFT and the FHT the complexity is Nlog2N.
An advantage of the DHT is the fact that the DHT is self-
inverse, so that only one software routine or hardware device
is needed for the forward and inverse FHT. For the forward
and inverse FFT of a real signal, two different routines or
devices are required. The DHT is somehow conceptually
easier than the DFT if the signaling is real, however all
operations are applied with the FFT and also the FHT with an
equivalent complexity.

B. Relationship to Fourier Transforms
This transform differs from the classic Fourier transform \( F(\omega) \rightarrow \{ f(t) \} (\omega) \) in the choice of the kernel. In the Fourier
transform, we have the exponential kernel: \( \exp -i\omega t = \cos \omega t -i \sin (\omega t) \) where \( i \) is the imaginary unit. The two transforms
are closely related, however, and the Fourier transform
(assuming it uses the same \( \frac{1}{\sqrt{2\pi}} \) normalization convention)
can be computed from the Hartley transform via:
\[
F(\omega) = \frac{H(\omega) + H(-\omega)}{2} - i \frac{H(\omega) - H(-\omega)}{2} \tag{5}
\]

That is, the important and imagined elements of the Fourier
transform are merely given by the even and odd elements of
the Hartley transform, severally. Conversely, for real-valued
functions \( f(t) \), the Hartley transform is given from the
Fourier transform’s real and imagined parts:
\[
\{ Hf \} = \{ Rf \} - i \{ I f \} = R\{ Ff \} \cdot (1 + i) \tag{6}
\]

Where \( R \) and that I denote the important and imagined
elements of the complicated Fourier transform.

III. LOOK-UP-TABLE
To efficiently implement multipliers with lookup-table-
based solutions, it’s necessary that one quantity to be a
continuing, once one in all the operands is constant, it’s
potential to store all the partial leads to a read-only memory,
and therefore the variety of memory words is considerably
reduced from \( 2^N \) to \( 2^L \). LUT in the main depends on RAM
blocks. we tend to might understand any logic operate from
truth table victimization RAM, this can be done by mapping
the inputs to the address bus and therefore the output is
mapped to the information bus.

Fig.1. Block diagram of LUT.
A Novel VLSI DHT Algorithm for A Highly Modular And Parallel Architecture

These Look-Up-Tables are very helpful to implement the multiplier factor in Field Programmable Gate Array (FPGA). The FPGA is one among the components in VLSI.

A. Parallel Process
Parallel version in the main used when combining a bigger FFT with a three or five purpose FFT, since it's possible to use three or five massive FFTs during a single device.

B. Proposed System
Highly parallel and standard resolution for the implementation of type-III DHT supported a replacement VLSI rule is given. An extremely parallel resolution for the implementation of DHT supported a direct implementation of fast Hartley transforms (FHT). It’s value to notice that hardware implementations of FHT are rare. A new very large scale integration (VLSI) algorithmic rule for a 2\(^{\text{th}}\)-length discrete Hartley transform (DHT) which will be expeditiously enforced on an extremely standard and parallel VLSI design having an everyday structure is conferred. The DHT algorithmic rule will be expeditiously split on many parallel elements which will be dead at the same time. Moreover, the projected algorithmic rule is compatible for the sub expression sharing technique which will be wont to considerably reduce the hardware quality of the extremely parallel VLSI implementation. Using the benefits of the projected algorithmic rule and also the fact that we will efficiently share the multipliers with constant, the quantity of the multipliers has been considerably reduced specified the quantity of multipliers is incredibly tiny examination with that of the existing algorithms. Moreover, the multipliers with a relentless will be expeditiously enforced in VLSI.

IV. RESULTS
A. Simulation Results

The above simulation shows the twiddle factor distributed for the multipliers W0..W3 represents the twiddle factors in memory. f0 ....f3 represents the switching activity of multiplier. The multiplier consists of 1 and -1.

The above simulation shows the twiddle factor distributed for the multipliers W0..W3 represents the twiddle factors in memory. f0 ....f3 represents the switching activity of multiplier. As the multiplier consists of common digits -1 and 1 it can be re-shared.
The above simulation is the real values obtained and the imaginary values are set to zero two successive inputs are given the sel line represents the inverse and normal DHT output.

**Fig6. Multiplication.**

The above simulation is the multiplier output for the twiddle factors and the inputs. the outputs are shown by out1 to out4.

**Fig7.**

The above simulation is the multiplier output for the twiddle factors and the inputs. the imaginary multiplication is done by using the polarity in inverse.

**Fig8. Imaginary multiplication.**

**B. Synthesis Report**

**Fig9. RTL schematic.**

**Fig10. RTL View.**
A Novel VLSI DHT Algorithm for A Highly Modular And Parallel Architecture

<table>
<thead>
<tr>
<th>Table1. Design Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

V. CONCLUSION AND FUTURE SCOPE

A DHT transform of 8 bit input is being implemented with radix 4 implementation. The selection line is the switch for DHT and its inverse. All variable length of inputs have been tested and synthesized. FFT has consumed 8 adders 4 multipliers where as the proposed scheme has only 4 adders and 2 sharing multipliers as one of the twiddle factor is one. Implementation is done in verilog language using Xilinx tool.

Future Scope

Split-radix techniques are very attractive since they provide both compact size and minimum operation counts. As processors evolve, the finite register set limitation also becomes less stringent. Such an environment could be used to exploit the advantages of the larger and more complex algorithms like vector radix techniques. Many promising hybrid techniques have been also developed and deserve attention. Image should also support saving the FFT Buffer to disk as well as its power spectrum. The ability to view and alter the amplitude and phase of images should also be supported. Finally, the dyadic frequency domain operations deserve optimization, since their speed could double with- out too much difficulty.

VI. REFERENCES