JCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.7, Special Issue.1, Jan 2019 E-ISSN: 2347-2693

Hardware Implementation of Fast Recursive Walsh-Hadamard Transform

Pulak Mazumder[1], Rajarshi Middya[2*], Mrinal Kanti Naskar[3]

[1]Department of Electronics and Communication Engineering, Regent Education and Research Foundation, Kolkata 700121, India
[2]Department of Electronics and Telecommunication Engineering, Jadavpur University, Kolkata 700032, India

[*]Corresponding Author: rajarshi.middya@gmail.com, Tel.: +91 9830024086

Available online at: www.ijcseonline.org

Abstract— The Walsh Hadamard Transform is an extremely relevant concept in modern digital image data compression. This paper examines the feasibility of using a tensor product based approach to Walsh Hadamard Transfrom for implementation to hardware architecture using FPGA technology. This paper explains the derivation of a highly parallel and very fast algorithm for the computation of both one-dimensional and two-dimensional transforms using tensor product. Such a fast dedicated hardware design for the Walsh Hadamard Transform will help a wide range of digital signal processing applications.

Keywords— Signal processing, VLSI, FFT, Transforms

I. INTRODUCTION

The convenience of using tensor product for implementation of a DSP algorithm is one of the major driving forces behind this work. When it comes to tensors, scientists and mathematicians have applied it for theoretical studies involving a plethora of disciplines. But a few years before, a seminal review on the utilisation of tensors in DSP algorithms [1] threw light on the unexplored area of algorithm optimisation using this mathe- matical concept. It is now an established fact that the tensor product can be used for the implementation of DSP algorithms due to the strong correlation between these product constructs and CPU architectures. But it is not only mere implementation but rather the optimisation in terms of time complexity that has allured many to pursue this. Although previous works [2] had focussed on modelling FFT algorithms, it was Granta's paper [1] which threw light onto the plethora of algorithms involving recursion that can have similar implementation. Having said these, the major constraint remains the implementation of these derived architectures for practical purposes. This had been greatly aided by Field Programmable Gate Arrays which have experienced a pleasant favouritism from researchers with labo- ratory constrains. Our case revolves around a simple purpose. We implement a very well known transform technique from the area of DSP using tensor product and as a result we were able achieve a parallelisation technique for the algorithms. This has helped us to propose a new architecture for an **a**pplication **s**pecific **i**ntegrated **c**ircuit (ASIC) dedicated to this purpose. It is imperative to mention at the beginning that the architecture we propose is supposed to work in real time. So this could be used as a block during real time processing of digital signals. As a result it might be able to decrease the overall burden on the main processor. This concept has been used recently in the name of co-processors. The **W**alsh-**H**adamard **T**ransform (WHT) is a mathematical construct that finds wide application in the fields of digital signal processing, data compression, and encryption. It also finds application in quantum computer information processing and, it is more often called Hadamard gate in this context. The transform is particularly useful in feature extraction for pattern recognition and digital image processing because of its easy implementation using simple arithmetic stages. Also, the binary nature of the Walsh functions and the Hadamard matrix allow easy implementation. In this work, we use a tensor-product approach. The usage of the tensor product allows implementation of WHT using an algorithm that is both recursive and parallel. Using tensor product allows the decomposition of the WHT matrix into simpler arithmetic stages. Decomposing the one-dimensional input into pairs and applying them to the arithmetic stages allow for parallel execution. The output obtained is stride-permuted and applied back to the same arithmetic stages, continuing the execution recursively. For a two-dimensional input, we design an algorithm that works on the column-major representation of the input. This representation is one-dimensional in nature, allowing the computation of the two dimensional signal similar to the one dimensional one.

Our paper is organised as follows. Section 2 gives a brief glimpse of previous works and where our work stands now. In section 3, we describe the concept of tensor product

© 2019, IJCSE All Rights Reserved

and its use for devising parallel architecture for recursive algorithms. Next, in section 4, we focus on our main work, its procedures, the theoretical deductions and the logical implementation. The results and observations are described in section 5. Lastly, we draw the conclusion of our work in section 6.

II. RELATED WORK

Fast implementation of various transforms have been done by many before. In [3], a fast algorithm for the computation of dis- crete Hartley transform have been given. Although there have been numerous attempts to make the computation of Hartley transform fast and their implementation in VLSI, they gener- ally fall under the categories of direct and indirect. The method described in [3] involves a small number of arithmetic oper- ations with simplified combinational structure and optimistic time complexity. As a result it is well suited for hardware implementation. Another method for the computation of the above transform has been proposed by the same author a few months later [4] in which the algorithms proposed is supposed to be well suited for the subexpression sharing technique thus reducing the hardware complexity for parallel implementation of this algorithm. But so far, the use of tensor product has been limited in such propositions. In [5], an algorithm has been proposed to compute the WHT and the Discrete Fourier Transform simultaneously using a unified butterfly. This is supposed to reduce the number of arithmetic operations than the traditional methods where the transforms are computed separately. The authors have used the Kronecker product technique in sparse matrix factorisation to achieve the butterfly structure. But the objective seemed to have been clouded by the combination of these two transforms instead of making each transform fast individually. But irrespective of the works that have been done so far, the parallelisation of the operations using tensor product for implementation of WHT seemed an utouched area. This motivated us to write the present paper.

III. CONCEPT OF TENSOR PRODUCTS

Previously used in the area of applied mathematics and physics, tensors have a very elegant representation for multi dimensional systems. Although represented as matrices, the definition of tensor products is somewhat different. Considering two tensors ***A*** and ***B*** with structures as

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{bmatrix}; B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1M} \\ b_{21} & b_{22} & \dots & b_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ b_{M1} & b_{M2} & \dots & b_{mm} \end{bmatrix}$$

the tensor product $\otimes$ can be defined as

$$C = A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1N}B \\ a_{21}B & a_{22}B & \dots & a_{2N}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1}B & a_{N2}B & \dots & a_{NN}B \end{bmatrix}$$

When implementing this in a computer program, the fact that a larger matrix C can be represented as the decomposed version of two smaller matrices A and B, becomes very useful. An important property of this tensor product is that the tensor matrix C has its inverse as $A^{-1}\otimes B^{-1}$. So to compute the inverse of a large matrix ($\mathbf{C}$), one can calculate the inverse of its decomposed tensor product (A and B). This acts as an advantage in the case of minimising time complexity.

IV. FORMULATION AND ARCHITECTURE

IV.a Walsh-Hadamard Transform

One-dimensional Transform

The WHT performs an orthogonal, symmetric, involution, linear operation in a set X_N of $N = 2^\alpha$ real numbers. For one dimensional WHT, the set of real numbers is represented as a vector $X_{N\times 1}$ and the transform is given by

$$Y_{N\times 1} = W_N X_{N\times 1} \quad (1)$$

The Hadamard matrix W_N for $N = 2^\alpha$ where α is an integer, can be recursively defined as

$$W_N = \begin{bmatrix} W_{\frac{N}{2}} & W_{\frac{N}{2}} \\ W_{\frac{N}{2}} & -W_{\frac{N}{2}} \end{bmatrix}$$

where $W_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$. Alternatively, W_{2^α} can be defined using tensor product as follows

$$W_{2^\alpha} = W_2 \otimes W_{2^{\alpha-1}}$$
$$W_{2^{\alpha-1}} = W_2 \otimes W_{2^{\alpha-2}}$$

Solving the recurrence for W_{2^α} using the method of substitution, we get

$$W_{2^\alpha} = W_2 \otimes W_2 \otimes W_2 \otimes \cdots \otimes W_2 \otimes W_2$$

Now, the product rule [6,7] implies

$$A_{N1} \otimes \dots \otimes A_{N1} = \prod_{k=1}^{t} \left(I_{N(k-1)} \otimes A_{Nk} \otimes I_{\frac{N}{N(k)}} \right)$$

Modifying the above generalised product rule for the WHT we get,

$$W_{2^\alpha} = \prod_{i=0}^{\alpha-1} \left(I_{2^i} \otimes A_{Nk} \otimes I_{2^{\alpha-i-1}} \right)$$

Further, using commutation theorem, product rule and the properties of stride permutation matrices, we can obtain

$$W_{2^\alpha} = \prod_{i=0}^{\alpha-1} \left(P_{2^\alpha,2} (I_{2^{\alpha-1}} \otimes W_2) \right) \quad (2)$$

Using this result in the definition of the WHT (Eq. 1), we get

$$Y_{N\times 1} = \prod_{i=0}^{\alpha-1} \left(P_{2^\alpha,2} (I_{2^{\alpha-1}} \otimes W_2) X_{N\times 1} \right) \quad (3)$$

This formulation gives way to an efficient algorithm for computing the Hadamard matrix where, $(I_{2^{\alpha-1}} \otimes W_2)X_{N\times 1}$ is a parallel operation and $P_{2^\alpha,2}$ is a 2^α - point stride 2 permutation matrix.

Two-dimensional Transform

The WHT of a square matrix X_N of order N is given by

$$Y_N = W_N X_N W_N^T \quad (4)$$

Since W_N has the same transpose i.e $W_N = W_N^T$, from Eq. 2, we get

$$W_N = W_N^T = \prod_{i=0}^{\alpha-1} \left(P_{2^\alpha,2}(I_{2^{\alpha-1}} \otimes W_2) \right)$$

If this we represent the matrix X_N as a column matrix X^C od order $2^{2\alpha} \times 1$, we can rewrite Eq. 3 for obtaining two-dimensional WHT as

$$Y^C = (W_N \otimes W_N^T) X^C$$

where Y^C is a column matrix representation of Y_N. Now, as $N = 2^\alpha$, from Eq. 4, we can write

$$W_N \otimes W_N^T = W_{2\alpha} \otimes W_{2\alpha}$$

$$\Longrightarrow W_N \otimes W_N^T = W_{2^{2\alpha}}$$

$$\therefore\ Y^C = \prod_{i=0}^{2^{\alpha-1}} \left(P_{2^{2\alpha},2}(I_{2^{2\alpha-1}} \otimes W_2) \right) X^C \quad (5)$$

Eq. 5 gives an efficient algorithm for computation of two-dimensional DWT where $(I_{2^{2\alpha-1}} \otimes W_2)$ is a parallel operation and $P_{2^{2\alpha},2}$ is a $2^{2\alpha}$ - point, stride 2 permutation matrix.

IV.b Proposed Architecture

One-dimensional Transform

Eq. (3) is composed of the following two steps -
(a) The parallel operation $(I_{2^{\alpha-1}} \otimes W_2)X_{N\times 1}$, and
(b) Multiplication of the result of the previous operation with the stride permutation matrix.

The operation in step (a) requires only addition and subtraction of the real values of the vector X - which can be considered to be the input vector. The next step i.e (b) is responsible for changing the order of the output of the previous step. Relating to this, a hardware architecture is presented where each iteration in Eq. 3 is completed in the positive half-cycle and the negative half-cycle witnesses the operation in step (b).

The circuit for 8-point WHT is presented in a block level in Fig. 1. As can be seen, the hardware implementation requires the following devices -
(i) four **A**rithmetic **U**nits (AU) each consisting of an adder and a subtractor for step (a), and
(ii) one **s**tride **p**ermutation **b**lock (SBP) for the operation in step (b)

The output of the SBP i.e the bits Q_1 to Q_8 are again loaded into the input bits X_1 to X_8 and fed back to the AUs in the next clock cycle for computation of the next iteration of the Eq. 3. The number of AUs required depends in the dimension of the input vector, E. If $N = 2^\alpha$, then 2^α number of AU units are required. The number of clock cycles required is equal to the number of iterations in Eq. 3. So, for 2^α point one dimensional WHT, the number of clock cycles required is α.

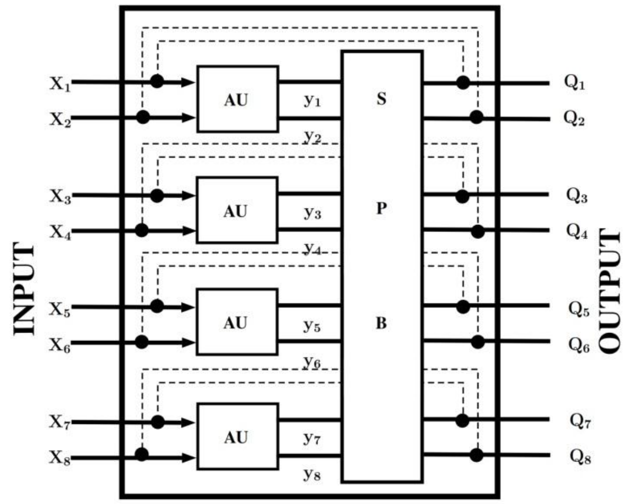

Figure 1 : The Block Diagram of the proposed architecture

Two-dimensional Transform

It can be observed that Eq. 3 and Eq. 5 have the same formulation with α in the former one, replaced by 2^α in the latter. So, the hardware architecture is the same. Only one extra device is required which can preprocess the two dimensional signal and convert it into a single column vector. The single column vector can then be fed to the block for the one dimensional (1-D) transform. One thing that needs to be kept in mind is that the output from the 1-D block would be column vector. So one needs to convert this to a two dimensional form to get the final output.

IV.c Algorithms

Following subsections give a glimpse of the pseudocode for the two variations of the transforms which have been developed.

One-dimensional Transform

(i) $X_N \leftarrow$ input of $N = 2^\alpha$ real numbers
(ii) For every rising edge of the circuit clock, $Y_N \leftarrow AU(X_N)$
(iii) For the corresponding falling edge of the circuit clock, $Y_N \leftarrow SPB(Y_N^T)$
(iv) $X_N \leftarrow Y_N$
(v) Repeat steps (ii) to (iv) for $\alpha - 1$ times.

Two-dimensional Transform

(i) $X_{N\times N} \leftarrow$ input of $N \times N = 2^\alpha \times 2^\alpha$ real numbers
(ii) Convert $X_{N\times N}$ into a column major representation X^C
(iii) For every rising edge of the clock pulse,
$(Y^C)^T \leftarrow AU(X^C)$.
(iv) For the corresponding falling edge of the circuit clock,
$Y^C \leftarrow SPB((Y^C)^T)$.
(v) $X^C \leftarrow Y^C$.

© 2019, IJCSE All Rights Reserved

(vi) Repeat steps (iii) to (v) for $2\alpha - 1$ times.
(vii) Convert the column vector Y^C to the two dimensional matrix $Y_{N\times N}$.

IV.d Time Complexity Analysis

For the analysis presented below, one clock cycle has been considered the unit of time and has been denoted by \`t'.

One-dimensional Transform

The time complexity analysis can be obtained from the algorithm in Sec IVc. The steps (ii) and (iii) get executed in time t. After these steps are executed, step (v) requires them to be executed $\alpha - 1$ times more. So, the total time for execution T is given by $= \left(1 + (\alpha - 1)\right)t = \alpha t$. Again we know that $N = 2^{\alpha}$ i.e $\alpha = \log_2 N$. Therefore from this calculation, the asymptotic representation for the time complexity of the algorithm for one-dimensional transform can be given by $O(\log_2 N)$.

Two-dimensional Transform

In the algorithm for the two-dimensional signal, the time taken for the steps (iii) and (iv) is t. After these steps, step (iv) requires them to be executed another $2\alpha - 1$ times. Ignoring the pre-processing in step (ii) and the post processing in step (vii), the total time required for the computation T can be given by $T = \left(1 + (2\alpha - 1)\right)t = 2\alpha = (2\log_2 N)t$. As the input signal is a matrix of order $N \times N$, the total input size M comes to be N^2. In terms of the total input size, the total time T comes to be $\log_2 M$. Therefore the asymptotic representation of the time complexity for this algorithm can be given by $O(\log_2 M)$.

V. Results and Observations

The proposed architectures have been tested in Verilog HDL. Different bit stream patterns have been employed to see the variation in the time required for computation. Fig. 2 shows the VHDL output for the bit stream of size 24.

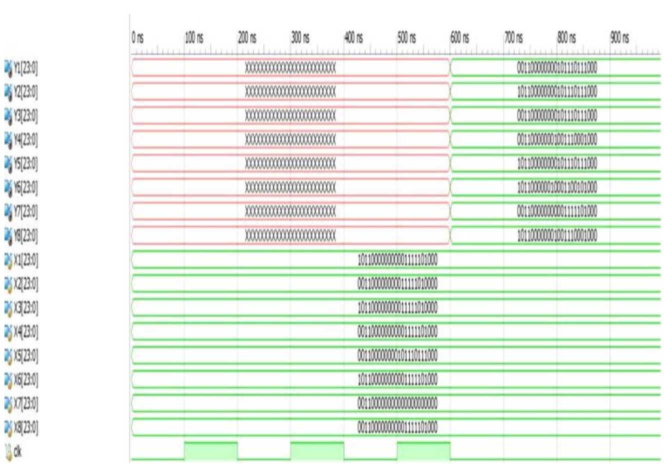

Figure 2 : OUTPUT

As can be seen, after the input X_1 to X_N each consisting of 24 bits are applied to the block, the output is generated in a very small time. The time required for different bit length - 16, 24 and 32 - have been shown in the Table. 1.

Table 1 : Table for time taken for computation of different bit streams of varying length

Input Length	Total Time
16-bit	3.401ns (1.534ns logic, 1.867ns route, 45.1% logic, 54.9% route)
24-bit	5.531ns (5.111ns logic, 0.420ns route, 92.4% logic, 7.6% route)
32-bit	5.642ns (5.111ns logic, 0.531ns route, 90.6% logic, 9.4% route)

VI. Conclusion

The proposed architecture is completely new for WHT as no one has proposed this kind of fast implementation of this transform. The results got from the simulations have been quite promising. The next step of work would be to implement this in an Field Programmable Gate Array for experiment.

References

[1] J. Granata, M. Conner, and R. Tolimieri, "*Recursive fast algorithm and the role of the tensor product*," IEEE Transactions on Signal Processing, vol. **40**, no. **12**, pp. **2921–2930, Dec 1992**.

[2] J. R. Johnson, R. W. Johnson, D. Rodriguez, and R. Tolimieri, "*A methodology for designing, modifying, and implementing fourier transform algorithms on various architectures*," Circuits, Systems and Signal Processing, vol. **9**, no**. 4**, pp. **449–500**, **Dec 1990.**

[3] D. F. Chiper, "*Radix-2 fast algorithm for computing discrete hartley trans- form of type iii*," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. **59**, no. **5**, pp**. 297–301**, **May 2012**.

[4] D. F. Chiper, "*A novel vlsi dht algorithm for a highly modular and parallel archi- tecture*," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. **60**, no. **5**, pp. **282–286**, **May 2013**.

[5] M. T. Hamood and S. Boussakta, "*Fast walsh-hadamard-fourier transform algorithm*," IEEE Transactions on Signal Processing, vol. **59**, no. **11**, pp. **5627–5631**, **Nov 2011**.

[6] J. R. Johnson and A. F. Breitzman, "*Automatic derivation and implemen- tation of fast convolution algorithms*," Journal of Symbolic Computation, vol. **37,** no. **2**, pp. **261 – 293**, **2004**.

[7] M. A. Richard Tolimieri and C. Lu, *Algorithms for Discrete Fourier Transform and Convolution*. Springer-Verlag **New York, 1997**.

© 2019, IJCSE All Rights Reserved

Authors Profile

Mr Pulak Mazumder received his M.Tech from West Bengal University of Tech- nology, Kolkata and his B.Tech degree in Electronics and Communication Engineering from IET, New Delhi, India. Presently he is the Head of the Department of Electronics and Communication Engineering in Regent Institute of Technology, Kolkata, India. His research interests include digital signalprocessing, microwave and em theory.

Mr Rajarshi Middya received his PhD from the School of Mobile Computing and Communications, Jadavpur University, Kolkata, India, his Master's Degree in Science and Technology from Paristech, Paris, France and his B.Tech degree in Electronics and Communication Engineering from West Bengal University of Technology, Kolkata, India. Presently he is attached with the Department of E.T.C.E, Jadavpur University, Kolkata, India. His research interests include wireless sensor networks, signal processing and nonlinear dynamics.

Mr Mrinal Kanti Naskar holds a PhD in Electronics and Telecommunications Engineering from Jadavpur University, Kolkata, India and an M.Tech degree and a B.Tech degree in Electronics and Electrical Communications Department, Indian Institute of Technology, Kharagpur. His research interests include Wire-less Sensor Networks, Computer Architecture and Digital Design. At present, he is a Professor in the Department of Electronics and Telecommunication Engineering, Jadavpur University, Kolkata, India.

© 2019, IJCSE All Rights Reserved