MATLAB based Design for an 8-point DFT formed on Products of Rademacher Functions

Roslidar Roslidar and Zulfikar Zulfikar
Department of Electrical and Computer Engineering, Syiah Kuala University, Banda Aceh 23111, Indonesia
roslidar@unsyiah.ac.id

Abstract—**This article discusses a DFT 8-point design based on Rademacher functions. The design is conducted based on previous research, a DFT 4-point model, for hardware. In general, the design requires two calculations of DFT 4-point, four counting processes of DFT 2-point, and three multiplying processes of twiddle factor. The adjustment has been made to change the path process from concurrent to sequential in order to adapt to the software execution command. The designed algorithm is run in MATLAB. The result shows that there is no difference between the proposed DFT and the one provided in MATLAB. Confidently, this design can be an alternative in transforming information signal into frequency domain using DFT technique.**

Index Terms—**Rademacher Functions; Radix-2; Sequential; Concurrent.**

I. Introduction

The rapid growth in computation technology requires high performance in processing various information. Therefore, it is requisite to provide a good signal processing tools. The technique used in signal processing called a Digital Signal Processing (DSP) applies digital format. In digital process, it is required to transform the usual time domain signal to other domain to obtain certain characteristics of the processed signal.

Fourier transform holds an important role in transforming signal domain. For certain application, a practical transformation that is derived from Fourier transform is frequently used. The information in digital format is usually transformed into frequency domain using Discrete Fourier Transform (DFT). This transformation method became popular when Cooley and Turkey introduced divide and conquer realisation in 1965 [1]. This technique then is known as Radix-2. Since that time, thousand paper discussed the practice of realisation method had been published [2]

Fourier transform can be realised by either using software or hardware. In the application that proposedly designed for Fourier signal processing, the realisation is usually conducted in hardware to reduce the selling price. However, in multitasking application, it is more efficient to conduct Fourier realisation using a software. This multitasking application requires a processor that cost the device. In later application, it mentioned various software that have been developed to calculate the Fourier transform [3]-[9]: Fourier calculation method using Sun Performance Libary (SUNPERF); Fortran public domain code by T. Ooura (1996), C language by J Green (1996), and C language by R. H. Krukar (1990); the Fortran FFTPACK library [3]; a Fortran split-radix FFT by Sorensen [4]; a Fortran FFT by Singleton [5]; Temperton's Fortran GPFA code [6]; Bailey's "4-step" FFT implementation [7]; Sitton's QFT code [8]; and the four1 routine from (NRF) [9].

Then, in 1998, a new algorithm was developed by Martin & Frigo [10] to process the Fourier transform calculation. It is the fastest technique compared to prior existing technique. This technique is used in MATLAB programming until now.

Another alternative realisation of Fourier Transform is by combining the processing technique with Walsh transform. The objective is to integrate the simple technique in Walsh transformation. Some researchers have developed the combination concept of both transformation techniques [11]-[13]. One of the research is the factoring of T intermediate transformation to calculate the DFT coefficient [11]. The further integrated technique is Fast Walsh-Hadamard Transforms (FWHT) using Radix-4 [12]. Later, joint counting technique is developed based on Radix-2 [13].

Furthermore, combination technique has also been realised with hardware using FPGA. The combination method was introduced in 2015 [14]. Hardware realisation using FPGA successfully combined both models for 4-point transformation. The realisation was conducted based on the similarity of Fourier matrix and Walsh matrix. Then, the combination of Fourier and Walsh design for 8-point length also has been published [15]. The technique was designed for hardware realization, therefore real and imaginary numbers were stored in the different buffer. The design only required one multiplexer with twiddle factor.

Nevertheless, the combination technique such in [14] and [15] has some drawbacks. When the transformation process is conducted for a large number of point (for example 32-point), it requires many multiplying processes with twiddle factor. Accordingly, the design of software to simplify the multiplication with twiddle factor is needed. This article introduces a software design of DFT calculation process by combining Fourier and Walsh transformation. This calculation method using Rademacher function as this function is usually used in Walsh transformation calculation process. The proposed DFT 8-point design in this article uses decimation in time (DIT) or Radix-2 approach.

II. Design of DFT

The process of transforming the signal into the frequency domain is powerfully done in Fourier transformation. In the frequency domain, the acquired data or signal characteristic is important information. Thus, the Fourier transformation is widely applied in computation device. However, the transformation process is very complicated as it requires many circuit or program. While transforming signal using Walsh transformation is very easy and requires less circuit or program. However, there is very few information about characteristic resulted in Walsh transformation [16]. This is the reason why Walsh transformation is rarely being used and almost being forgotten.

The proposed DFT 8-point is designed to combine the regular Fourier transform technique with that of Walsh transformation. The purpose is to extract the simple technique to simplify the DFT transform. Walsh transformation is run in a simple technique using products of Rademacher functions. This technique is adapted to do the calculation process of Fourier transformation using DFT. Some adjustments are needed to obtain the transformation result using the combination technique.

Unlike Walsh transformation that uses input only integer numbers, Fourier transformation includes non-integer numbers too. In this way, the calculation based on hardware is not easy to process. DFT 4-point design and realisation based on Rademacher functions into FPGA has been published [14]. The design only processed the real and integer numbers. Even though the transformation generates imaginary numbers, in the calculation process, both numbers are stored in the separated buffer. DFT 8-point design for hardware implementation was published [15]. The design constraint the calculation both on integer and non-integer numbers so that it required broad circuit.

Figure 1 shows the diagram block of calculation process in Fourier transformation for a discrete model known as Discrete Fourier Transformation (DFT) for 8-point [2]. The figure displays the design of DFT 8-point with two DFTs 4-point, four DFTs 2-point, and three multiplication processes with twiddle factor. The first DFT 4-point (the upper part) is used to calculate the even data input (x_0, x_2, x_4, x_6). Whereas, the odd data input (x_1, x_3, x_5, x_7) will be processed through the second DFT 4-point (the lower part). Both calculation results become the input to four DFTs 2-point. In other words, all input into DFT 2-point is a joint output of DFT 4-point. However, some output of DFT 4-point at the lower part (L_1, L_2, L_3) have to be multiplied by twiddle factors. The output of DFT 2-point in sequence (X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7) is considered as the transformation result of DFT 8-point.

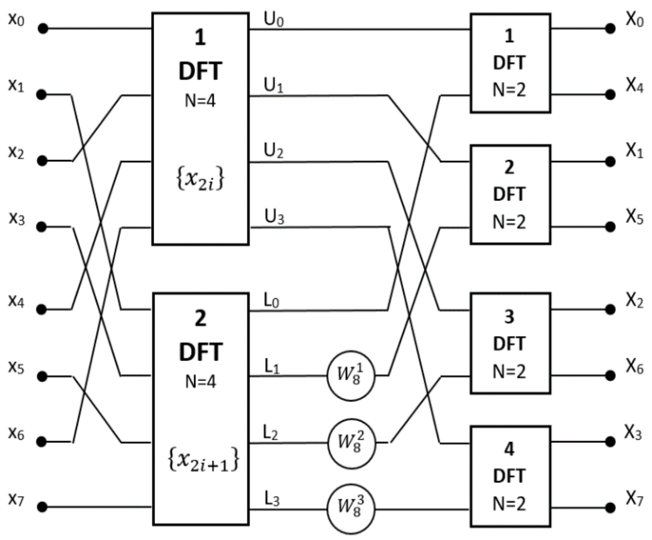


Figure 1: The design of DFT 8-point in general [2]

In order to simplify the calculation, the designed software run in MATLAB will be called from the smaller functions. It requires two calculation functions of DFT 4-point and that of 4 counting of DFT 2-point.

A. Design of 4-point DFT

DFT 4-point design refers to the algorithm that has been published previously [14]. The algorithm is designed to imitate Walsh transformation approach. Input data is multiplied by a matrix to obtain the transformation outcome. In calculation process of Walsh transformation, the matrix contains +1 or -1. While in DFT calculation process, the matrix contains +1, -1, +j, dan –j. The following are the matrices in both transformation methods.

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

Walsh matrix DFT matrix

There are similarities in both matrices. First, all elements in the first row of both matrices consist of +1. Second, both matrices are composed of integers. Third, rows other than the first row contains positive and negative numbers in the same amount (two positive numbers and two negative numbers). The challenge is the position of the positive and negative numbers are not at the same point. Another challenge is there are four imaginary numbers in DFT matrix.

Figure 2 describes the flow chart of calculation program in the proposed DFT 4-point. Part of the designed flowchart adopts the calculation process published previously [14]. Some modifications are needed as the previous design based on hardware works concurrently while here the DFT 4-point exerts sequentially.

The program is started by giving a matrix to hold the calculation result. The output is separated into two-part, imaginary (*Im*) and real (*Real*). Then, the initialisation of *Rad* variable that represents the Rademacher function and *i* variable that counts the total loop to be executed is determined. Because there are four numbers of data input, then there will be four looping processes. To simplify the multiplication process of Rademacher functions, which usually operates with logic exclusive-or (*xor*), *Rad* has to be changed into the binary number (*R* variable), and the value will be added by 1 for every undertaken looping. The following step is the multiplication process of Rademacher functions. The multiplication process is represented by *XOR* variable which is the multiplication product of *R(0)* and *R(1)*.

Every time the looping occurs, one number will be inserted from data input *D* for calculation. Then, *F* variable is used to accommodate the input number temporarily. The variable will take in the positive or negative values from data input depends on the Rademacher *R* function. *F(1)* mainly holds the positive values from each data input because the elements in the first row from the Fourier or Walsh matrix are all positive. Later *F* variable is accumulated into a Real matrix.

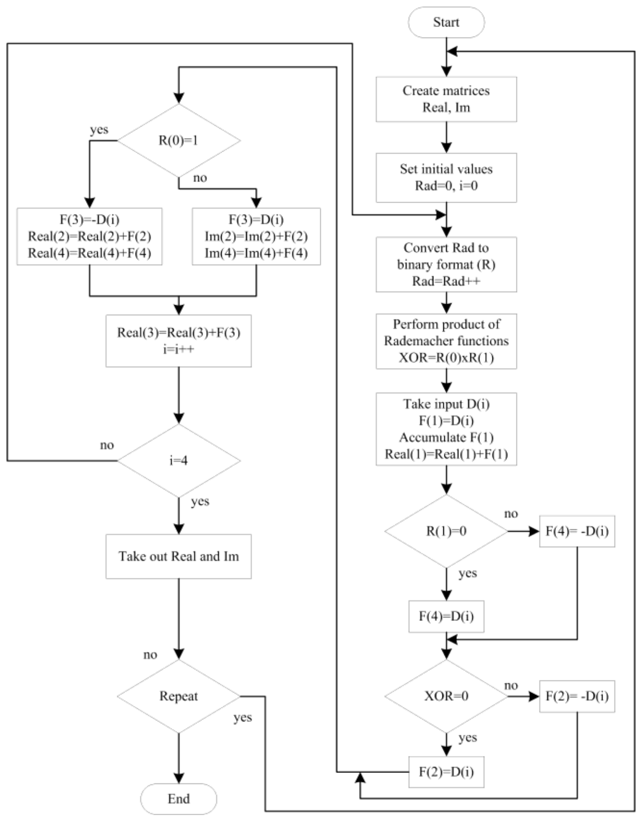

Start
Create matrices Real, Im
Set initial values Rad=0, i=0
Convert Rad to binary format (R) Rad=Rad++
Perform product of Rademacher functions XOR=R(0)xR(1)
Take input D(i) F(1)=D(i) Accumulate F(1) Real(1)=Real(1)+F(1)
R(1)=0
no
F(4)= -D(i)
yes
F(4)=D(i)
XOR=0
no
F(2)= -D(i)
yes
F(2)=D(i)
R(0)=1
yes
no
F(3)=-D(i) Real(2)=Real(2)+F(2) Real(4)=Real(4)+F(4)
F(3)=D(i) Im(2)=Im(2)+F(2) Im(4)=Im(4)+F(4)
Real(3)=Real(3)+F(3) i=i++
i=4
no
yes
Take out Real and Im
no
Repeat
yes
End

Figure 2. The flow chart of designed DFT 4-point based on Rademacher multiplying functions

The next flow is decision process that is the process of deciding which input values are to be considered (positive or negative). Three conditions are used as the reference, either *R(1) = 0, R(0) = 0*, or *XOR = R(0)R(1) = 0*. If the Rademacher function *R(1)* is equal to logic 0, then *F(4)* variable will hold the positive value from data input *D*. Otherwise, it will take in negative value (*–D*). The second decision is whether the *XOR* variable has a logic 1 or 0. If the variable has logic 0, then *F(2)* variable will take in the positive data input, or else it will hold negative data input.

The last decision is whether *R(0)* has logic 1 or not. If the variable has logic 1, then variable *F(3)* will take in the negative value temporarily, accumulate *F(2)* in *Real(2)* variable, and accumulate *F(4)* in *Real(4)* variable. If *R(0)* condition is not in logic 1, then the *F(3)* variable will keep the positive data input value. The temporary value stored in *F(2)* and *F(4)* will be accumulated in *Im(2)* variable and *Im(4)* variable, respectively.

After the decision process completed, the next step is accumulation process of *F(3)* into *Real(3)* variable. Concurrently, *i* variable will be added by 1. The value of *i* variable will be the reference in doing the looping, whether the looping process will be repeated. If the program has taken four looping, then the accumulated numbers (stored in *Real* and *Im* matrix) will be pulled out and considered as the output of the DFT 4-point transformation.

B. Design of 8-point DFT

Figure 3 shows the programming flow of executing functions in DFT 4-point, DFT 2-point, and multiplication process with twiddle factor to form the transformation process of data input into the frequency domain. In the designed program, DFT 4-point will be executed twice, and DFT 2-point will be executed four times corresponding to the design of DFT 8-point that is shown in Figure 1.

The program flows as follows. First, the program starts by reading data input x in matrix size 8x1. The data will divide into two parts, even (*U*) and odd (*L*). Next, the program calls DFT4 function with matrix *U* as the input to the function. The calculation output processed by DFT4 holds in *UR* variable for real number and *UI* for the imaginary number. It follows by retrieving DFT4 function to process data input from matrix *L*. The output of this process will be saved in *LR* and *LI* variables for real number and the imaginary number, respectively.

Afterwards, the DFT2 function is executed four times for calculation process in DFT 2-point. The input for each DFT2 function is the output from DFT4 which processed formerly. For example, *UR(1), UI(1), LR(1),* and *LI(1)* will be the input of DFT2 function (DFT2#1 process). However, the numbers stored in the second, third, and fourth column in matrix *LR* and *LI* are not inserted directly as the input data to DFT2. The numbers have to be multiplied with twiddle factor.

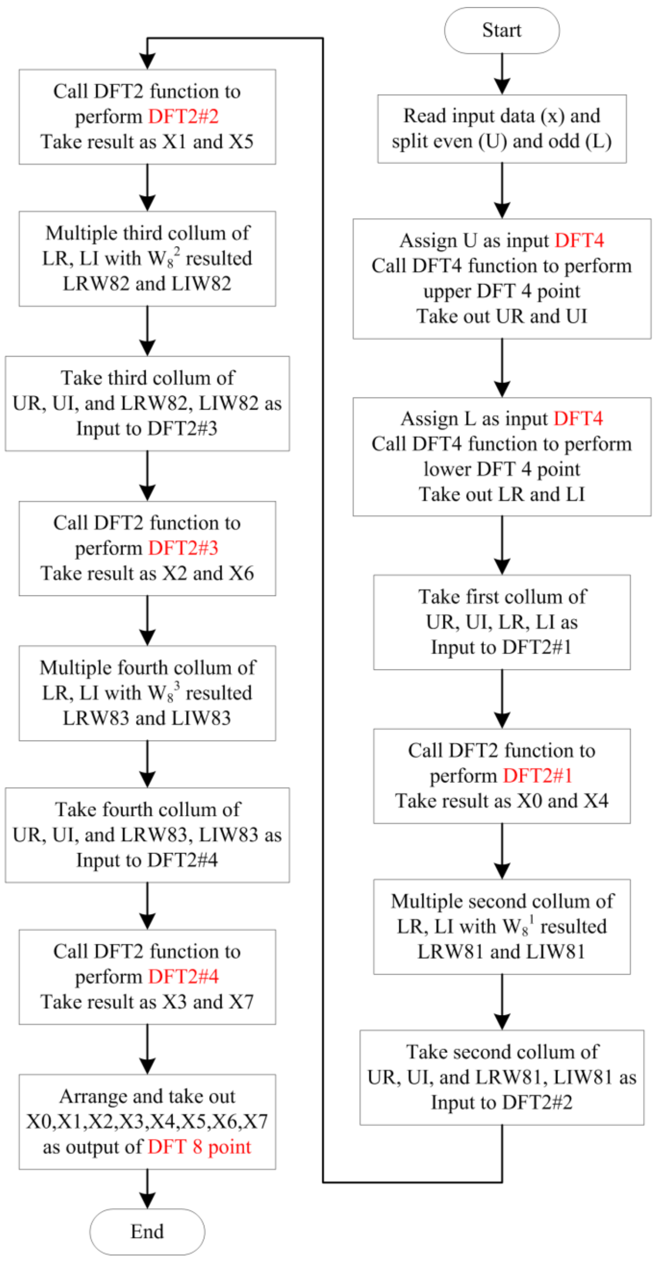

Start
Read input data (x) and split even (U) and odd (L)
Assign U as input DFT4 Call DFT4 function to perform upper DFT 4 point Take out UR and UI
Assign L as input DFT4 Call DFT4 function to perform lower DFT 4 point Take out LR and LI
Take first collum of UR, UI, LR, LI as Input to DFT2#1
Call DFT2 function to perform DFT2#1 Take result as X0 and X4
Multiple second collum of LR, LI with W_8^1 resulted LRW81 and LIW81
Take second collum of UR, UI, and LRW81, LIW81 as Input to DFT2#2
Call DFT2 function to perform DFT2#2 Take result as X1 and X5
Multiple third collum of LR, LI with W_8^2 resulted LRW82 and LIW82
Take third collum of UR, UI, and LRW82, LIW82 as Input to DFT2#3
Call DFT2 function to perform DFT2#3 Take result as X2 and X6
Multiple fourth collum of LR, LI with W_8^3 resulted LRW83 and LIW83
Take fourth collum of UR, UI, and LRW83, LIW83 as Input to DFT2#4
Call DFT2 function to perform DFT2#4 Take result as X3 and X7
Arrange and take out X0,X1,X2,X3,X4,X5,X6,X7 as output of DFT 8 point
End

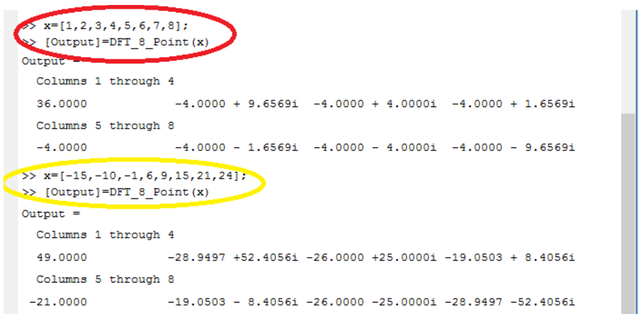
Figure 3: The flow of calculation process in DFT 8-point

The numbers in the second column of matrix *LR* and *LI* are multiplied by twiddle factor W_8^1. Then the outcome is kept in *LRW81* and *LIW81*. Both of the value, as well as the numbers on the second column from matrix *UR* dan *UI*, becomes the input into DFT2 (DFT2#2 process). The result of the function is stored in *X1* and *X5* variables. The following step is multiplying *LR(3)* and *LI(3)* by twiddle factor W_8^2. The output of this multiplications (*LRW82* and *LIW82*) along with *UR(3)* and *UI(3)* becomes the data input for calculation function in DFT2 representing DFT2#3 process. The numbers in the fourth column of matrix *LR* and *LI* are multiplied by twiddle factor W_8^3, and the output is taken in *LRW83* and *LIW83*. Both of the values including the numbers in column four from matrix *UR* and *UI* becomes the input to DFT2 (DFT2#4 process).

The result of four executions of the DFT2 function is the final output of the transformation using DFT 8-point. However, the result has to be rearranged to form a correct sequence of DFT 8-point output. The output of each DFT2 function has been arranged in appropriate order, which is [*X0, X1, X2, X3, X4, X5, X6, X7*] as shown in the last process of the flowchart in Figure 3.

III. Results and Discussions

The designed algorithm of DFT 8-point based on Rademacher function is programmed on MATLAB. The execution result has been analysed and some of them shown in Figure 4. Some data input is used to figure out the truth of program compilation.

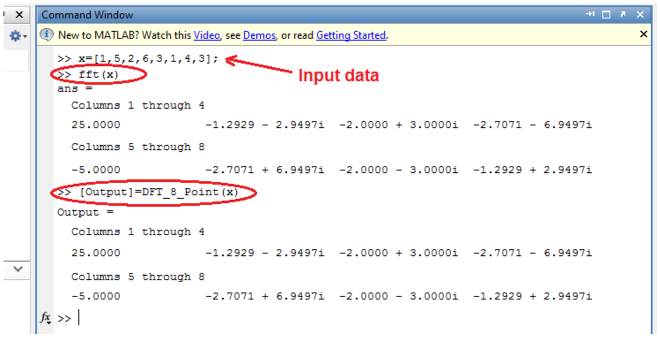


```
>> x=[1,2,3,4,5,6,7,8];
>> [Output]=DFT_8_Point(x)
Output =
  Columns 1 through 4
  36.0000           -4.0000 + 9.6569i  -4.0000 + 4.0000i  -4.0000 + 1.6569i
  Columns 5 through 8
  -4.0000           -4.0000 - 1.6569i  -4.0000 - 4.0000i  -4.0000 - 9.6569i
>> x=[-15,-10,-1,6,9,15,21,24];
>> [Output]=DFT_8_Point(x)
Output =
  Columns 1 through 4
  49.0000          -28.9497 +52.4056i -26.0000 +25.0000i -19.0503 + 8.4056i
  Columns 5 through 8
 -21.0000          -19.0503 - 8.4056i -26.0000 -25.0000i -28.9497 -52.4056i
```

Figure 4: The screenshot of the execution result of DFT 8-point program

Figure 4 shows MATLAB display of programming execution of two different input data. The first given data is x = [1, 2, 3, 4, 5, 6, 7, 8] (marked in red circle) and the second is x = [-15, -10, -1, 6, 9, 15, 21, 24] (marked in yellow circle). The program is executed by a simple command; *[Output]=DFT_8_Point(x)*. The figure represents the exact result of proposed program execution.

Then, the program is compared to the result of discrete Fourier transform provided in MATLAB library. The program is stored in a file named *fft.m*. The program algorithm is just like the one in the method published by Martin and Frigo [10]. Figure 5 shows the comparison of execution result using designed DFT 8-point program and the program provided in MATLAB. Both of the program have the same input which is x = [1, 5, 2, 6, 3, 1, 4, 3]. The output of both executions is exactly the same. This concludes that the design of DFT 8-points based on Rademacher function has successfully been conducted and the output produced by the compiled program resembling the expected result.


Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
Input data

```
>> x=[1,5,2,6,3,1,4,3];
>> fft(x)
ans =
  Columns 1 through 4
  25.0000           -1.2929 - 2.9497i  -2.0000 + 3.0000i  -2.7071 - 6.9497i
  Columns 5 through 8
  -5.0000           -2.7071 + 6.9497i  -2.0000 - 3.0000i  -1.2929 + 2.9497i
>> [Output]=DFT_8_Point(x)
Output =
  Columns 1 through 4
  25.0000           -1.2929 - 2.9497i  -2.0000 + 3.0000i  -2.7071 - 6.9497i
  Columns 5 through 8
  -5.0000           -2.7071 + 6.9497i  -2.0000 - 3.0000i  -1.2929 + 2.9497i
fx >> |
```

Figure 5: The screenshot of the execution result of the designed DFT 8-point and provided a program in MATLAB library

IV. Conclusions

DFT 8-point design based on Rademacher function has been constructed formed on DFT 4-point model for hardware realisation which has been published in [14]. Some adjustment has been made to revise the execution pattern from concurrent to sequential. The algorithm design is run under MATLAB programming. The output is resembling the one resulted in DFT program provided in MATLAB. Potentially, this design can be an alternative in calculating the Fourier transform. Further study should be done to enhance DFT algorithm with more input, such as 16-point, 32–point, etc.

References

[1] J.W. Cooley and J.W. Tukey, "An algorithm for the machine computation of the complex Fourier series," Mathematics of Computation, vol. 19, pp. 297–301, Apr. 1965.

[2] P. Duhamel and M. Vetterli, "Fast Fourier transforms: a tutorial review and a state of the art," Signal Processing, vol. 19, pp. 259–299, Apr. 1990.

[3] P. N. Swarztrauber, "Vectorizing the FFTs," Parallel Computations (ed. G. Rodrigue), Academic Press, pp. 51–83, 1982.

[4] H. V. Sorensen, M. T. Heideman, and C. S. Burrus, "On computing the split-radix FFT," IEEE Transactions on Acoustics, Speech and Signal Processing, vol. 34, pp. 152–156, Feb. 1986.

[5] R. C. Singleton, "An algorithm for computing the mixed radix fast Fourier transform," IEEE Transactions on Audio and Electroacoustics, vol. AU-17, pp. 93–103, June 1969.

[6] C. Temperton, "A generalized prime factor FFT algorithm for any n = 2p3q5r," SIAM Journal on Scientific and Statistical Computing, vol. 13, pp. 676–686, May 1992.

[7] D. H. Bailey, "A high-performance FFT algorithm for vector supercomputers," Intl. Journal of Supercomputing Applications, vol. 2, no. 1, pp. 82–87, 1988.

[8] H. Guo, G. A. Sitton, and C. S. Burrus, "The quick discrete fourier transform," in Proc. IEEE Int. Conf. Acoust., Speech, and Signal Proc., Apr. 1994.

[9] W. H. Press, S. A. Teukolsky, and W. T. Vetterling, Numerical Recipes in Fortran: The Art of Scientific Computing. New York, NY: Cambridge University Press, 1992

[10] M. Frigo and S. G. Johnson, "FFTW: an adaptive software architecture for the FFT," Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, Seattle, WA, pp. 1381-1384 vol.3, 1998.

[11] S. Boussakta, and A. G. J. Holt, "Fast algorithm for calculation of both Walsh-Hadamard and Fourier transforms (FWFTs)," Electron. Letter, vol. 25, no. 20, pp. 1352-1354, 1989.

[12] Monir T. Hamood and, Said Boussakta, "Fast Walsh–Hadamard–Fourier transform algorithm," Trans. Signal Processing, vol. 59, no. 11, pp. 5627-5631, November 2011

[13] Teng Su, and Feng. Yu, "A Family of Fast Hadamard–Fourier Transform Algorithms," Signal Processing Letters, vol. 19, no. 9, pp. 583-586, September 2012.

[14] Zulfikar and H. Walidainy, "A novel 4-point discrete Fourier transforms circuit based on product of Rademacher functions," 2015 International Conference on Electrical Engineering and Informatics (ICEEI), Denpasar, 2015, pp. 132-137.

[15] Zulfikar and H. Walidainy, "Design of 8-point DFT based on Rademacher Functions," International Journal of Electrical and Computer Engineering vol. 6, no. 4, pp. 1551-1559, 2016

[16] M. Y. Zulfikar, S. A. Abbasi, and A. R. M. Alamoud, "FPGA Based Processing of Digital Signals using Walsh Analysis," Proceeding of IEEE International Conference on Electrical, Control and Computer Engineering (INECCE 2011), pp: 440-444, 21-22 June, Pahang, Malaysia, 2011.