ECE 641 Final Project

Generalized Discrete Fourier Transform (GDFT) FPGA Implementation

By

Wes Weydig

Andy A. Hernandez

12/8/2010

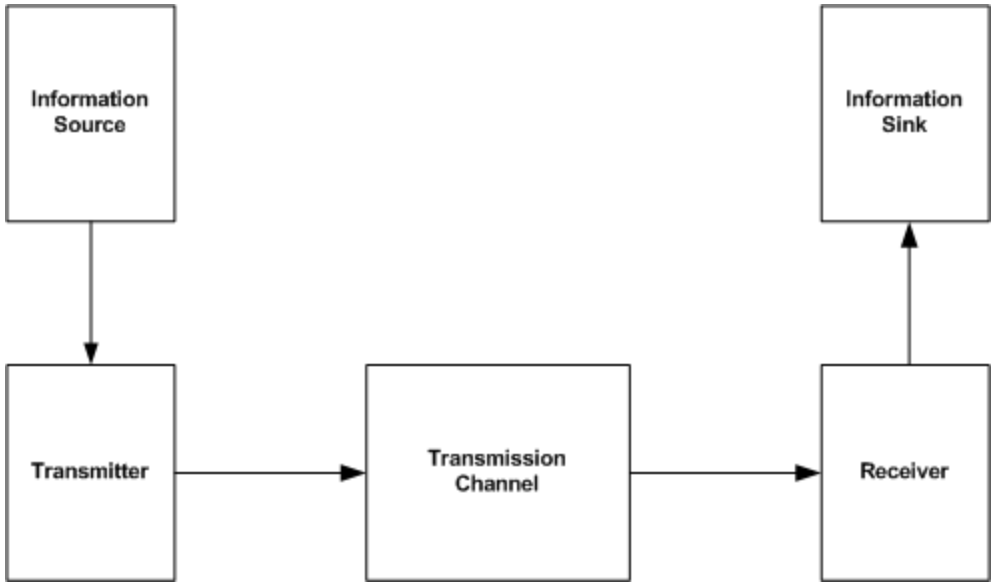
Introduction

The Generalized Discrete Fourier Transform, (GDFT), utilizes non-linear functions in the phase space whereas DFT is linear in the phase space. The GDFT is a constant modulus transform that is a desired property in wireless communications. The non-linearity of the GDFT has been shown to provide performance improvements in wireless applications.

Reference:
http://web.njit.edu/~akansu/gdft.htm

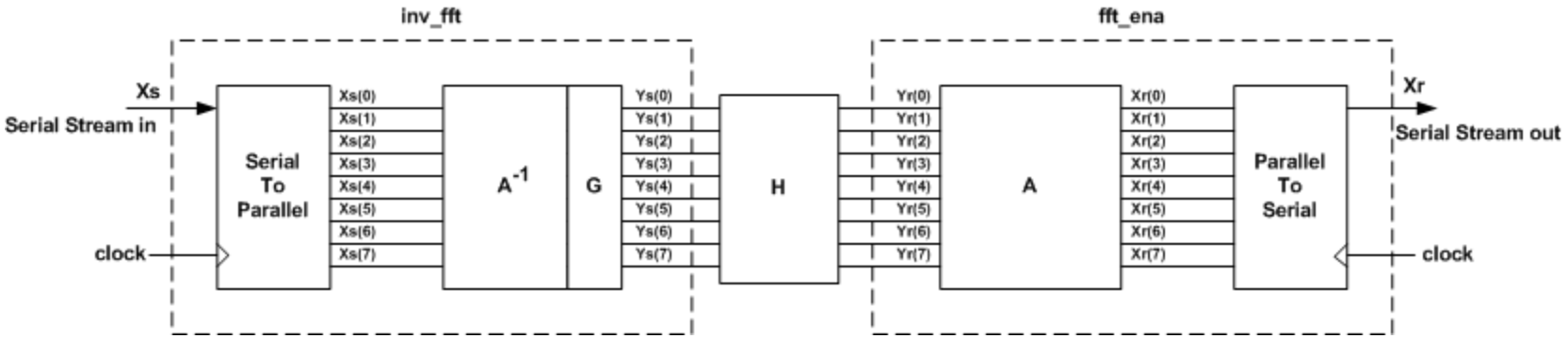
Objective

The objective of this exercise is to implement the GDFT in a Field Programmable Gate Array, FPGA. For this experiment, the underlying concept was to imitate a digital communication system inside the FPGA.



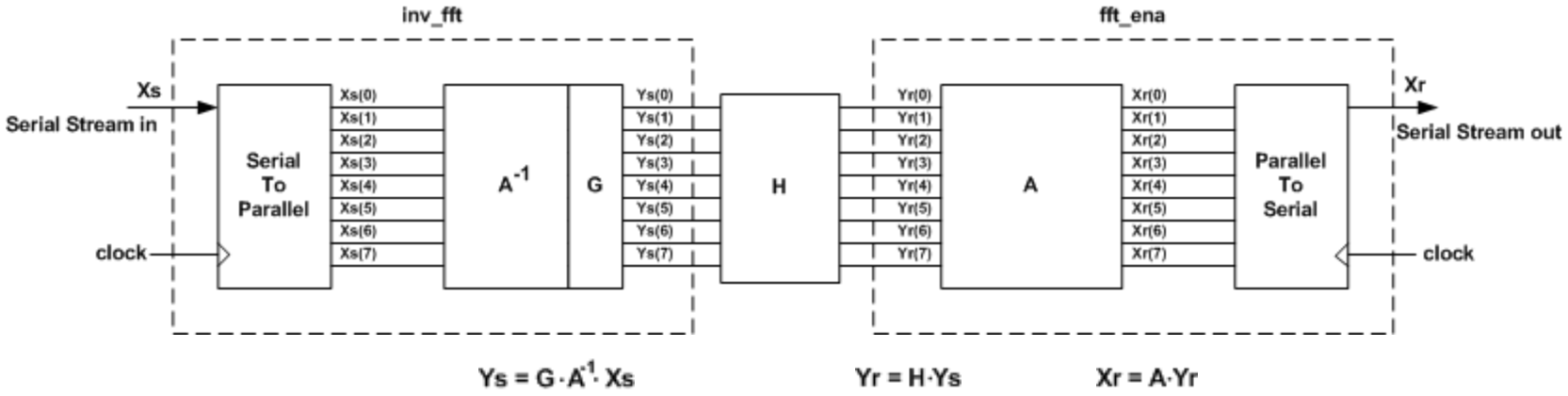
Objective

The block diagram of the final system that is implemented is shown in the figure below.



Objective

The overall objective is for the serial stream at the input to appear at the output unchanged. The following equations demonstrate the concept.



$Xr = A \cdot H \cdot Ys$

$Xr = A \cdot H \cdot G \cdot A^{-1} \cdot Xs$ $\quad G = H^{*}$, $H \cdot G = I$

$Xr = A \cdot I \cdot A^{-1} \cdot Xs$

$A \cdot A^{-1} = I$

$Xr = Xs$

G matrix is made such that it is the conjugate of H matrix. Therefore, H G will compute to the identity matrix.

Implementation

The transmit portion of the design consists of an 8-bit, length-8 ramp signal that is serially input into the IFFT module. For ease of presentation, only real values were used at the input.

x_in_re = [08,10,12,14,16,18,20,22] - decimal values shown

x_in_im = [0,0,0,0,0,0,0,0]

The IFFT output results in a complex signal which computes to

x_ifft_re_out = [15,1,1,1,1,1,1,1]

x_ifft_im_out = [0,2,1,1,0,0,-1,-2]

Implementation

The IFFT block in this project is a modified version of the FFT block that was used in Lab Number 6. This required editing the values of the following constants in the inv_fft block.

from

cc_in: (64, 45, 0, -45);
ccms_in: (64, 90, 64, 0);
ccps_in: (64, 0, -64, -90);

to

cc_in: (64, 45, 0, -45);
ccms_in: (64, 0, -64, -90);
ccps_in: (64, 90, 64, 0);

The final stage of the fft also had to be modified to divide the result by N = 8.

Implementation

The divide function presented a rounding error in the final answer. The solution is to sample the answer, make a decision on whether to round up or not and apply the decision to the answer. This technique has to be implemented every time there is a divide-by function. Out of the IFFT function, the divide by N = 8. The G and H matrices were normalized to 8 bits, so the resulting multiplications had to be divided down by 128. By doing these rounding calculations, the final output values are closer to that of the input. For divide by 8, bit two is registered. This bit is then used to decide whether to add 1 to the answer or to leave the answer alone. When a divide by 128 is calculated, bit 6 is registered and this value is used to decide whether to add 1 or not.

Implementation

H is a diagonal matrix which held arbitrary values of our choosing. These values maintain a magnitude of 1, but change the phase value. The phase values were chosen to be

[pi/6, 2pi/6, 3pi/6, 4pi/6, 5pi/6, 6pi/6, 7pi/6, 8pi/6]

These values were converted to Cartesian in Matlab and normalized to 8 bits. The G matrix is the complex conjugate of H and therefore consists of the negative phase values of H.

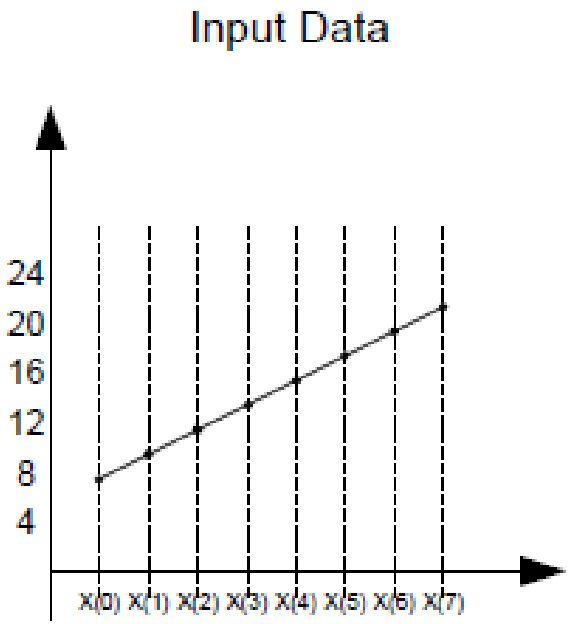
Implementation

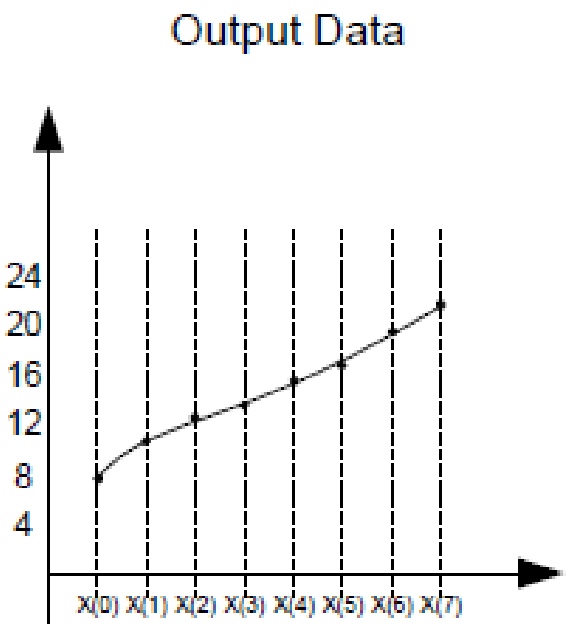
These complex values are set as constants in the design and consist of numbers in the format of a + jb. To perform the multiplication of the G matrix with the resultant IFFT values, FOIL must be computed on each component in the G and IFFT matrices. Therefore, 4 multiplications and 2 additions must be performed per matrix component. This was done in hardware by instantiating 4 Altera LPM multipliers and 2 adder/subtracter blocks. This same method was applied to the H matrix multiplication.

Implementation

Performing all these multiplies, additions, and subtractions in hardware comes with loss of precision. We were able to overcome some of these limitations by rounding the results of each divide stage, but the final output was still not exact. This would most likely be overcome by increasing the length-8 IFFT – FFT blocks to length 64 or 128 which would add more granularity to the divide at the output of the IFFT.

Final Results





[8 10 12 14 16 18 20 22]

[8 11 13 14 16 17 19 22]

Matlab / SignalTap results

Xs =	X_IFFT =	Ys_NORM =	Yr_NORM =	Xr =
22	15.0000	12.8889 - 7.4414i	14.7665 + 0.0000i	21.6576
20	1.0000 + 2.4142i	2.5705 + 0.3384i	0.9844 + 2.3766i	19.6887 + 0.0000i
18	1.0000 + 1.0000i	0.9922 - 0.9922i	0.9844 + 0.9844i	17.7198 + 0.0000i
16	1.0000 + 0.4142i	-0.1402 - 1.0647i	0.9844 + 0.4078i	15.7510 + 0.0000i
14	1.0000	-0.8593 - 0.4961i	0.9844	13.7821 + 0.0000i
12	1.0000 - 0.4142i	-0.9922 + 0.4110i	0.9844 - 0.4078i	11.8132 + 0.0000i
10	1.0000 - 1.0000i	-0.3632 + 1.3554i	0.9844 - 0.9844i	9.8444 + 0.0000i
8	1.0000 - 2.4142i	1.5783 + 2.0569i	0.9844 - 2.3766i	7.8755 + 0.0000i

$$Ys = G \cdot A^{-1} \cdot Xs \qquad Yr = H \cdot Ys$$

H and G were normalized to 8 bits. After the multiplication, the result was divided by 128
Ys_NORM and Yr_NORM are post division

Xs =	yre_post_round =	Ifft_g_real_post_round =	Ys_h_real_post_round =	Yre_out =
22	15	13 - 7i	15 + 0i	22
20	1 + 2i	2 + 0i	1 + 2i	19
18	1 + 1i	1 - 1i	1 + 1i	17
16	1 + 0i	0 - 1i	1 + 1i	16
14	1 + 0i	-1 - 0i	1 + 0i	14
12	1 - 0i	-1 + 0i	1 + 0i	13
10	1 - 1i	0 + 1i	1 - 1i	11
8	1 - 2i	1 + 2i	1 - 2i	8

Questions

?