734 PROJECT REPORT

64-POINT FFT ALGORITHM FOR OFDM APPLICATIONS USING 8-POINT DFT PROCESSOR (RADIX-8)

PANKHURI

18 MAY 2013

FAST FOURIER TRANSFORM

A Fourier transform takes advantage of symmetry and periodicity properties of DFT (Discrete Fourier Transform) to lower the amount of computation needed. Every DFT can be divided into smaller DFT's. This is basically what an FFT does.

The basic butterfly unit that is the simplest FFT studied is a 2-point FFT. 8-point FFT seems to be a more attractive option for OFDM applications perhaps because it reduces the number of additions and multiplications much more effectively than a 2-point DFT (This is based on what I have observed after reading the papers). Exact counts measuring the numbers of additions and multiplications will be presented in one of the later sections.

Implementation of 64-point DFT has been done. Equation of 64-point DFT is written below. However the method used is more efficient and does not use the same equation written below.

$$X(k) = \sum_{n=0}^{63} x(n)e^{-j2nk/64};\ k = 0 \text{ to } 63$$

64-POINT FFT ALGORITHM DETAILS [1]

In the algorithm used, 64-point DFT is divided into two smaller DFT's each of length eight as per the equation written below.

$$X(k) = X(8r + s) = \sum_{m=0}^{7} W_8^{\,mr}\, W_{64}^{\,ms} \sum_{l=0}^{7} x(8l+m) W_8^{\,sl}$$

where r = 0 to 7 and s = 0 to 7.

So the input complex data, x(8l+m) can be represented as a 2D array where l represents row and m represents column. The first DFT is the column-wise DFT (where only l varies in the equation above). The new matrix obtained after this first step is then multiplied with a constant also shown in the equation. After that, for this intermediate array, a row wise 8-point DFT is calculated to obtain the final result X(8r+s).

PERFORMANCE IMPROVEMENT

The base FFT algorithm (8-point DFT) has been implemented using the Winograd algorithm. This algorithm calculates the DFT in steps that are easy to pipeline. The equations for 8-point DFT are shown below. There are only two complex multiplications, as can be seen.

$$
\begin{aligned}
s(1) &= q(1) + q(2) \\
s(2) &= q(1) - q(2) \\
s(3) &= q(3) - jq(4) \\
s(4) &= q(3) + jq(4) \\
s(5) &= q(5) - j(1/\sqrt{2})q(6) \\
s(6) &= q(5) \pm j(1/\sqrt{2})q(6) \\
s(7) &= (1/\sqrt{2})q(8) - jq(7) \\
s(8) &= (1/\sqrt{2})q(8) + jq(7)
\end{aligned}
$$

$$
\begin{aligned}
y(1) &= s(1) \\
y(2) &= s(5) + s(7) \\
y(3) &= s(3) \\
y(4) &= s(5) - s(7) \\
y(5) &= s(2) \\
y(6) &= s(6) - s(8) \\
y(7) &= s(4) \\
y(8) &= s(6) + s(8)
\end{aligned}
$$

[1]

Therefore when this method is used for 64-point FFT the total multiplications needed for 8-point DFT's is two for each. There are 8 column-wise DFT's and 8 row-wise DFT's, so a total of 16, which means a total of 32 complex multiplications. Added to that is 64 complex multiplications to pre-calculated constants (done immediately after column-wise DFT in the algorithm above).

This is way less than a typical 64-point DFT which would take 4096 such operations and a basic radix-2 64-point DFT which takes 192. Hence there's a significant performance improvement.

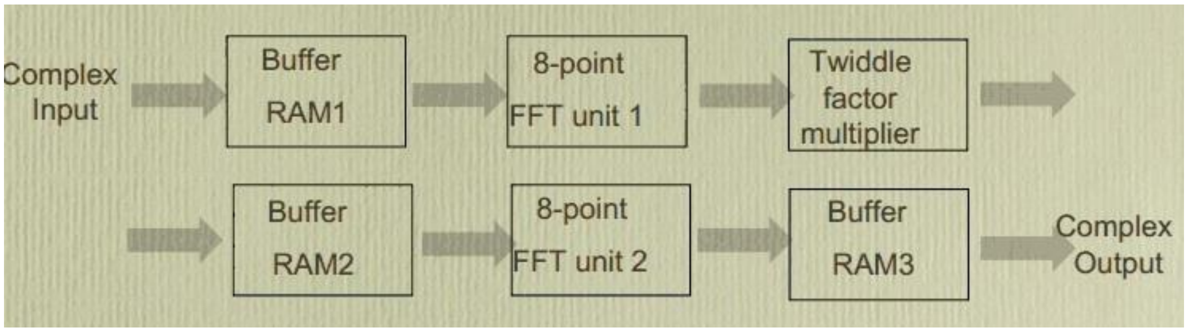
8-POINT FFT PROCESSOR DETAILS

As mentioned above, the baseline processor uses Winograd algorithm. The total number of multiplications is minimized at the expense of increase in additions and some more memory requirement [2-4]. For most of the processor architectures (with some exceptions), multiplications are more expensive, therefore decreasing then at expense of increased additions is a desirable optimization.

Based on the step-wise method followed for Winograd in the equations above, the processor implementation has been pipelined. In every clock cycle one complex number is read from the input data buffer and one is written to the output data buffer. Total of 14 clock cycles are needed by the 8-point DFT. The processor thus designed supports 250 MHz. The measured frequency of the processor designed is quite close and will be mentioned in a later section.

PROCESSOR DESIGN OVERVIEW

The diagram below shows the block schematic of the processor. Synthesis and simulation was done using Altera Quartus II in Verilog HDL on FPGA library.



(Adapted from my own presentation for this project)

DESIGN MODULES

Data buffers: Data buffers convert the 8-inverse form of data (obtained after a DFT) back to natural order. Although first and second buffers are necessary but the third one is optional. However, if it is not used the data output obtained from the processor would be in 8-inverse order: 0,8,16.....56, 1, 9, 17.... so on. So third buffer ensures correct order of output data but does not affect correctness of the operation of processor unlike previous data buffers. These buffers have two banks each, so each can store 2x64 complex data. Quartus module ‘altsyncram’ has been used to design data buffers. While one of the banks is being written to from previous stage, other can be read simultaneously. The processor operation is never blocked because of double banking.

FFT Blocks: FFT blocks are pipelined based on the algorithm described above. Only constant multiplications needed are to the factor 0.707, which has been designed using a bunch or shift and add operations (Please refer the code attached for details). Since we are using sixteen 8-point DFT operations in our processor design algorithm, requirement is for sixteen such modules. But this is avoided by multiplexing and using one each for column and row DFT operations. These are the two units shown in the block diagram. So, the amount of hardware is decreased at the expense of increased latency.

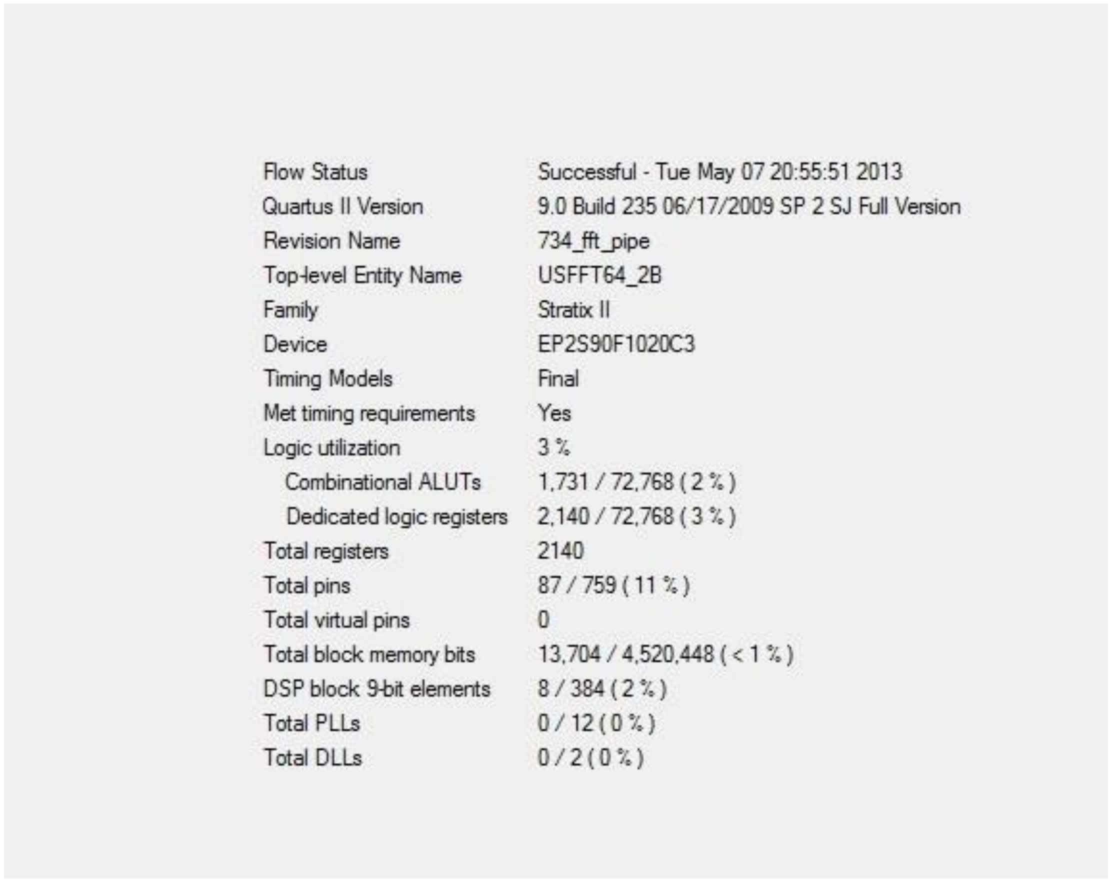
Twiddle factor multiplier: After column-wise DFT operations, the intermediate 2D array obtained in our processor requires multiplication with a constant twiddle factor. The complex multiplication here is done by breaking in into three multiplies and five additions as shown below.

$$(A + jB)(C + jD) = C(A-B) + B(C-D) + j(A(C-D) – C(A-B))$$

This is again an efficient method to reduce the number of multiplications by increasing additions. If regular method is used, there would be one more multiplication and one less addition. The multiplier has been designed using ‘lpm_mult’ mega function.

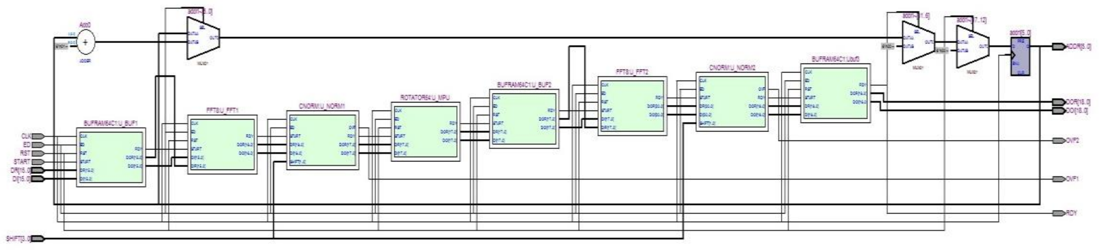
SYNTHESIS

Synthesis results are shown below:

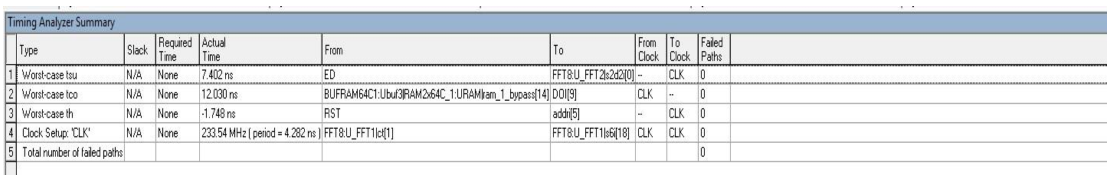


Flow Status	Successful - Tue May 07 20:55:51 2013
Quartus II Version	9.0 Build 235 06/17/2009 SP 2 SJ Full Version
Revision Name	734_fft_pipe
Top-level Entity Name	USFFT64_2B
Family	Stratix II
Device	EP2S90F1020C3
Timing Models	Final
Met timing requirements	Yes
Logic utilization	3 %
Combinational ALUTs	1,731 / 72,768 (2 %)
Dedicated logic registers	2,140 / 72,768 (3 %)
Total registers	2140
Total pins	87 / 759 (11 %)
Total virtual pins	0
Total block memory bits	13,704 / 4,520,448 (< 1 %)
DSP block 9-bit elements	8 / 384 (2 %)
Total PLLs	0 / 12 (0 %)
Total DLLs	0 / 2 (0 %)

The design was tested using a testbench and the results were as expected. The RTL design thus obtained is shown below. It looks similar to algorithmic block schematic.



The timing details are also pasted below. The clock frequency is quite close to that predicted in [1] (also mentioned in the processor details section above)



Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths
1	Worst-case tsu	N/A	None	7.402 ns	ED	FFT8:U_FFT2|s2d2i[0]	--	CLK	0
2	Worst-case tco	N/A	None	12.030 ns	BUFRAM64C1:Ubuf3|RAM2x64C_1:URAM|ram_1_bypass[14]	DOI[9]	CLK	--	0
3	Worst-case th	N/A	None	-1.748 ns	RST	addri[5]	--	CLK	0
4	Clock Setup: 'CLK'	N/A	None	233.54 MHz (period = 4.282 ns)	FFT8:U_FFT1|ct[1]	FFT8:U_FFT1|s6i[18]	CLK	CLK	0
5	Total number of failed paths								0

CONCLUDING REMARKS

LEARNING OUTCOMES

While working on project various implementation issues specific to FFT processor design were encountered. These included - resolving bandwidth issues when multiple stages are involved, reducing multiplier count (pipelining) and reducing total number of multiplications required (dependent on algorithm efficiency). Optimized design was implemented based on the literature study.

Another very important learning outcome was the knowledge of various FFT algorithms used in OFDM applications gained during literature review prior to shortlisting this one. This review also gave a chance to understand various strategies to reduce the computation employed in such algorithms especially the popular use of radix-8.

FUTURE WORK

Modular design of the processor allows it to be used together with other 64-point FFT's to create a larger size. This can be done in a way similar to the way current design has been built using smaller 8-point DFT's.

The same structure can be configured in Xilinx, Altera, Alcatel, Lattice FPGA devices, ASIC.

APPLICATIONS

Applications of the current and similar processor designs lie in OFDM modems, software defined radio, multichannel coding and many other high-speed real time systems.

REFRENCES

[1] Fuster J.J. and Gugel K.S., “Pipelined 64-point Fast Fourier Transform for Programmable Logic Devices”, http://www.add.ece.ufl.edu/papers/fust.pdf

[2] Oppenheim, A.V., Schafer, R.W., Discrete-time Signal Processing, 2nd ed., Prentice Hall, New Jersey, 1999.

[3] Press, W.H., Flannery, B.P., Teukolsky S.A., and Vetterling W.T., Numerical Recipes in C: The Art of Scientific Computing, Cambridge University Press, January 1993.

[4] Smith, Steven, The Scientist and Engineer’s Guide to Digital Signal Processing, California Technical Publishing, 1997.

PROGRAM DETAILS

NOTE: As per discussion during my presentation, I have added the option for user to configure the length of input, so that it can configured as per the input coming from DAC. This makes it more hardware efficient.

Code can be submitted as needed (I am unable to submit more than one files through drop box)