[FFT_2XRADIX4]

Demo: FFT_2xRadix4

Fast Fourier Transform: Using 2- Radix 4 Butterflies

Date	Revison
04/03/2011	1.0.1

IMPULSE ACCELERATED

FFT_2xRadix4

Fast Fourier Transform: Using 2- Radix 4 Butterflies

1 Introduction

There are many different FFT algorithms in use for FPGAs. This demonstration shows usage of an in-place iterative Radix 4 FFT, which uses 2 Radix 4 butterflies and 8 memory banks. The pipelining and memory bank access indexing allows for a doubling of the throughput, compared to a single Radix 4 algorithm.

1.1 Overview

The FFT transform is computed on an input array of N-complex numbers, **x**, producing an output array of N-complex numbers **X** . This is notated by: X = FFT(x), with the defining linear equation given by:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi i\left(\frac{k}{N}\right)} \quad , \qquad 0 \le k \le N-1$$

The FFT equation is well known, and implementing it in FPGA hardware in an efficient manner is the purpose of this demonstration.

1.1.1 Single Radix-4 Kernel Algorithm

The initial method used to compute X= FFT(x) , encapsulated by the FFT256_Float Impulse Example is based on using a recursive algorithm that passes over the data $\log_4(N)$ times, applying a single Radix-4 Kernel (which is just a 4-point FFT with arithmetic optimization).

First, we lay the incoming data into the working “in-place” array in a Radix-4 reversed order. This is done so that we may process all passes “in-place”. An added benefit is that the result of the final pass leaves the output array in natural output order.

In the first computational pass, the 4-point FFT Kernel function (Rad4) computes a series of small 4-point FFTs, in-place, with the proper indices given by the Index Generator. Effectively, at the end of the first pass, we’ve computed N/4 -4-point FFTs.

The second pass does the same thing, passing over the data with the Rad4 kernel, in-place, but with different indices, produced by the Index Generator. At the end of the second pass, we’ve effectively computed N/16 – 16-point FFTs.

Notice that due to the in-place nature of the computation, there is an opportunity to use multiple memory channels that support single-cycle Read/Write access. The only constraints are that: 1) There can be no memory Read or Write conflicts for any of the memory channels during any of the passes and 2)The Twiddle Array ROM memory holding the Sin and Cos tables for the exp(-2*pi*i(k/N) factor must also support the necessary number of simultaneous reads. The Index Generator has been adjusted to meet constraint #1 and dual-port RAMs are sufficient to meet constraint #2.

This process continues for $\log_4(N)$ passes, until we've computed our answer, 1(N/N) – N-point FFT .

1.1.2 Double(2x) Radix-4 Kernel Algorithm

To double the throughput, we can use 2- Rad4 kernels passing over the array data simultaneously for each pass, assuming that we can adjust the Index Generator to support 8-memory channels, with the same no-R/W conflict constraint for the 8 memory channels.

This is done by extending the single Rad4 alogirthm's memory channel indexing function:

getBank(i) = bank number(0-3) where the sample i is stored for a single Rad4 method.

The sample data is stored in eight separate arrays in order to access eight samples every cycle. The sample data must be carefully laid out in the arrays so that the eight values needed in each iteration are always in different arrays. The getBank8 macro maps the element number to an array/bank number (0-7) in a way that ensures this.

1.1.3 Software

The software Impulse C test bench first generates a data stream in test_producer(), then computes the hardware model 2xRad4 FFT of the data in fftproc(), and then streams the results back to the test_consumer() to display the results.

1.1.4 Hardware

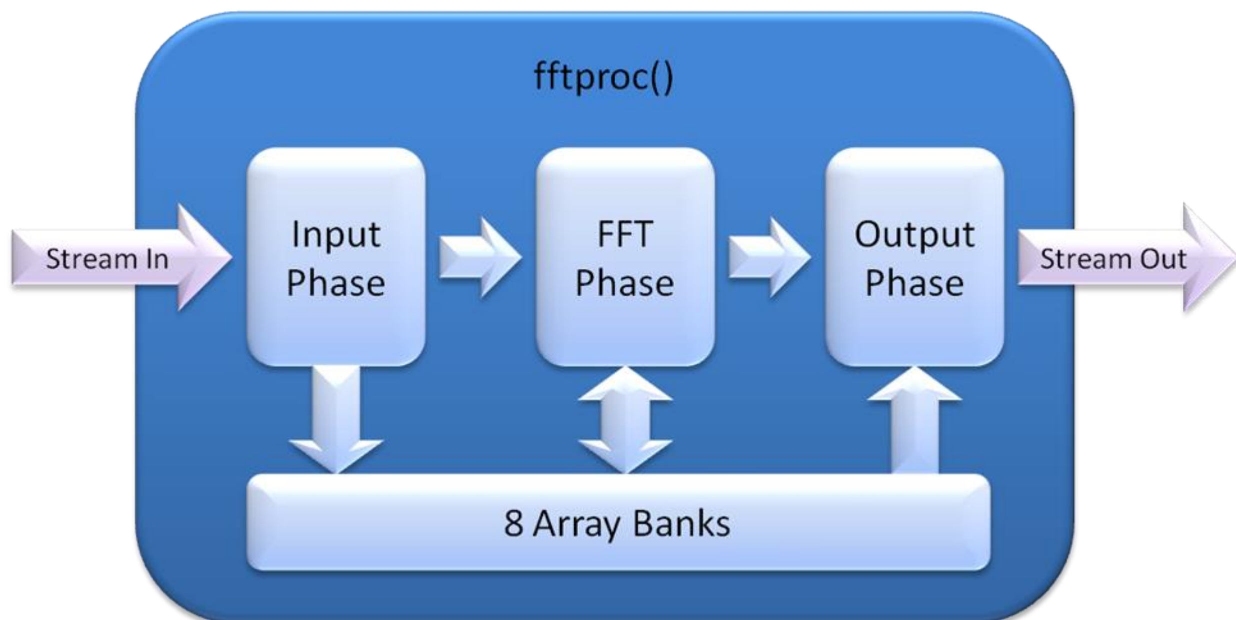
The hardware model is in fft_hw.c, and uses #include arrays of twiddle coefficients, one each for the different point FFT sizes. Currently, the hw model supports point sizes N=256 and N=4096, which is controlled in the file fft_parm.h by commenting/uncommenting the constants: FFT256 and FFT4096.

```
// Choose the point size here 256 or 4096, and then recompile.
//#define FFT256
//
#define FFT409
```

1.1.5 Processing Flow

Below is a diagram of the FFT processing flow performed in fftproc(). There are three basic phases to the designed flow:

1) Input Phase: Data is streamed into the FFT process and stored into the 8 array banks in bit-reversed order which allows the output to appear in natural order on completion of the FFT calculations.
2) FFT Phase: The 8 array banks are processed in multiple passes by the 2 x Rad4's until complete.
3) Output Phase: The FFT result is streamed out of the FFT process



2 ML605 Demo

2.1 Overview

For simplicity, the FFT project uses the CoDeveloper "Xilinx Microblaze FSL (VHDL)" Platform Support Package to generate a pcore that can be connected directly to a Microblaze within an EDK design. The Microblaze is used to write data to the FFT and read out the results using FSL streams and display measured performance on the serial terminal.

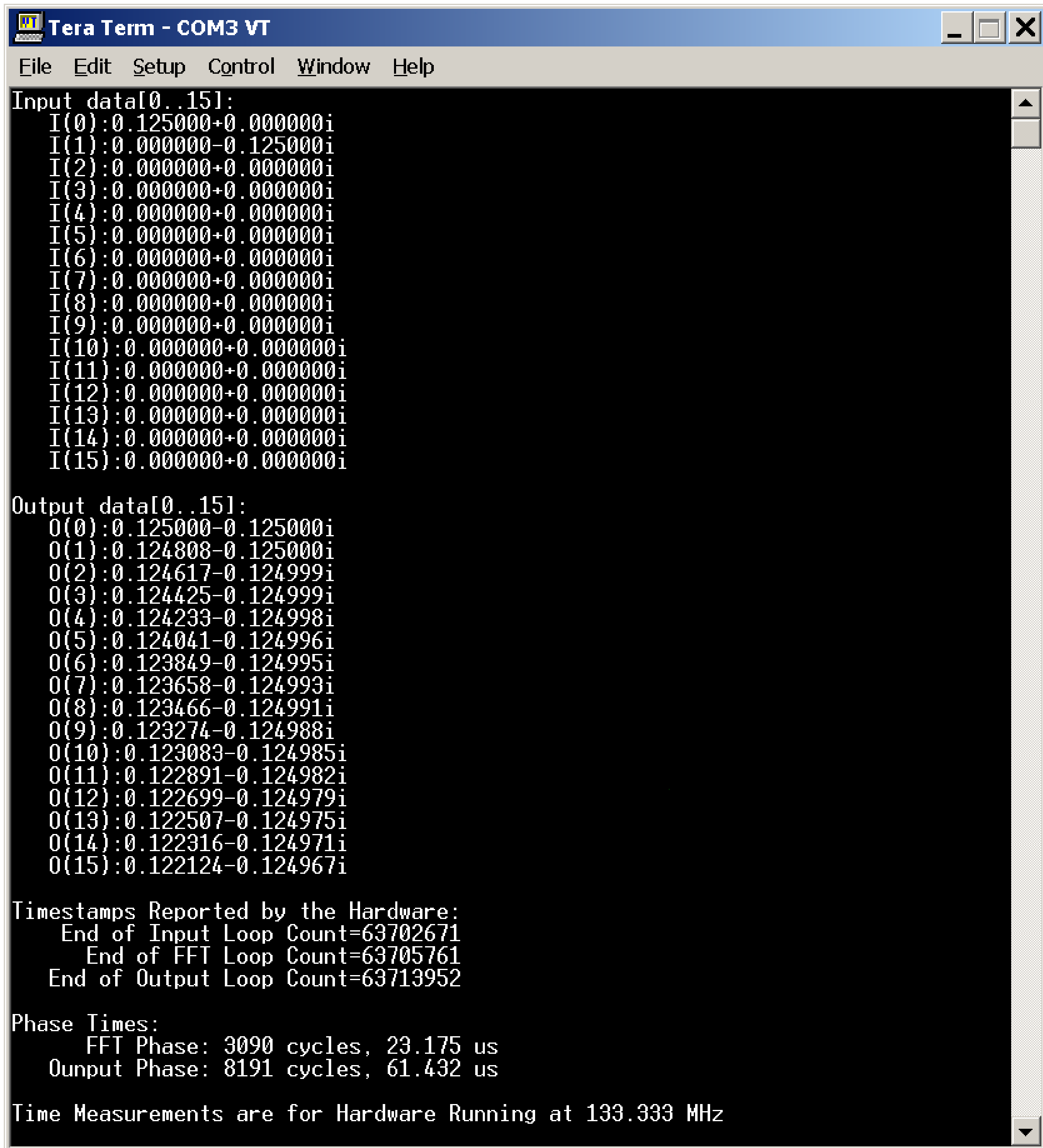
2.2 Measuring Performance

For accurate performance measurement, the project has been instrumented by including the cycle counter process timer_proc() that simply increments the global variable CycleCounter on every cycle. The fftproc() then periodically captures the value of CycleCounter as a timestamp during the different processing phases and outputting the values after the FFT resulting data.

The Microblaze reads the timestamps and then calculates and reports the number of cycles and time in microseconds for each phase.

2.3 Terminal Output

Below is a screen capture of the serial terminal output



2.4 About Impulse

Impulse C is the most widely used C to FPGA toolset. It includes tools for analysis, optimization, refactoring and export which enable software developers to quickly move C to FPGA hardware. More information is available at www.ImpulseC.com or by contacting info@ImpulseC.com. Generally it helps to provide some information about the type of design under consideration for hardware acceleration, the target platform (if any) and overall design goals. Founded in 2002, Impulse C is used by organizations from Toyota to NASA, and by schools from MIT to Paderborn.