EE Times Connecting the Global Electronics Community

designlines PROGRAMMABLE LOGIC

Design How-To

How to build a fast, custom FFT from C

Alan Coppola, OptNgn, and Brian Durwood, Impulse

5/11/2011 11:58 AM EDT
3 comments

Tweet 0 Share +1 0

FPGAs are great vehicles for FFTs. You can get solid libraries from the manufacturer and be quickly on your way. For even faster performance, engineers are splicing their other processing code in-line with FFT libraries. This is particularly effective when a system is to be optimized for a specific frequency or profile.

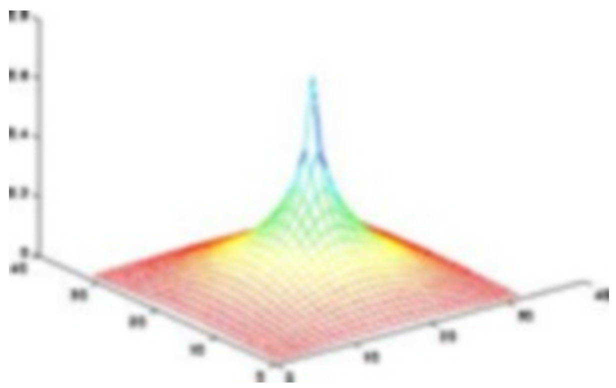
Recently, C-based FPGA FFT libraries have been announced that open up more options, since working from C is a particularly natural way to describe and experiment with log, exponential, and angle iteration.

Customizing an FFT provides engineers and software developers the opportunity to embed part of their processing logic, in-line, in the actual transform. This type of digital signal processing is useful for scaling or for situations in which the coefficients might need to be tweaked or go through extensive iterations to determine the optimal configuration. Customizing FFTs like this used to require sophisticated VHDL or Verilog coding, but not anymore…

First, one asks, why bother to use C-to-FPGA to implement a custom FFT? Well, three common and one newish cases where a custom FFT makes sense are as follows:

1. **Restriction:** Focusing on one specific range of frequency values. For instance, the military often filters for a specific frequency or profile. A custom FFT can be engineered to put all of its resources on just that. For instance a chirp-Z transform targets specific values, increasing speed and reducing resource use by eliminating other parts of a more generic FFT approach.
2. **Derivative:** Real FFTs and others which create derivatives of FFTs, which extend an architecture to cover related signal features that are of interest. For instance Real 2D FFTs can be used to process video directly, in hardware.
3. **Different Application Domains:** For instance NUFFT (Non-uniform FFTs) are used to address variable resolution and transforming curved coordinates as with medical or toroidal imaging.
4. **Polar Calculations:** Polar FFTs are finding a home in compensating for object rotation, registration, or perspective in "intelligent video". In this deployment, C-based algorithms can be used to pre- or post-process streaming signals. Examples range from military usage in tracking incoming missiles to 3D movie image generation. To

customize, engineers splice specific Fourier-Mellin transformations into the FFT framework.

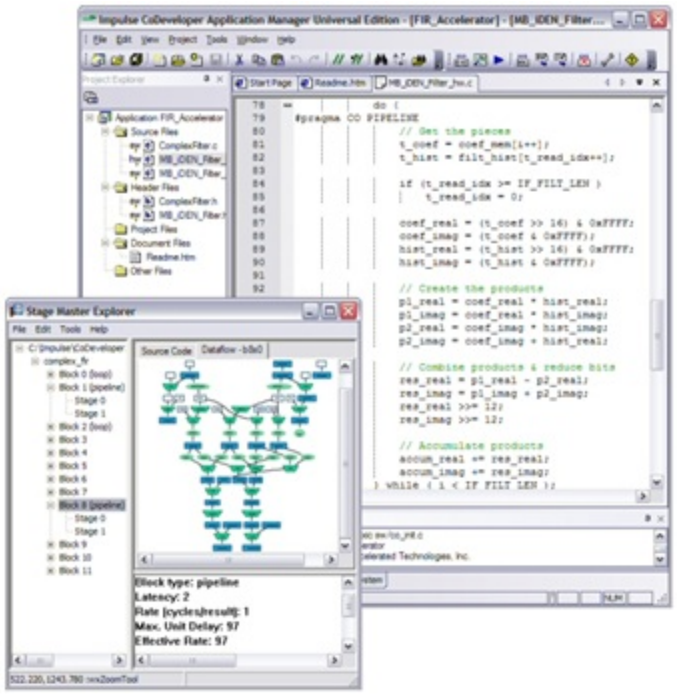


A generic FFT

Steps to an FFT
FFT libraries optimized for FPGAs integrate with the **Impulse** software-to-hardware compiler, allowing software developers to quickly prototype, optimize, and deploy high performance systems requiring a range of FFT sizes and precisions.

The FFT libraries produced by **OptNgn Software** use newer algorithmic techniques to fully utilize the FPGA's higher potential parallelism (over microprocessors and GPUs) in random addressing of memory locations.

These libraries are applicable to a wide range of applications that include image processing, national security, medical electronics and industrial signal processing. The FFT libraries are used as callable blocks from C code. This lets software developers more easily invoke a specific hardware library function, modify parameters, and intermix the library element with other code to form a larger system design that may include C, Verilog, and VHDL.


Impulse CoDeveloper Application Manager Universal Edition - [FIR_Accelerator] - [MB_IDEN_Filter...
Stage Master Explorer
Source Code
Dataflow - b0s0

```
        do {
#pragma CO PIPELINE
            // Get the pieces
            t_coef = coef_mem[i++];
            t_hist = filt_hist[t_read_idx++];

            if (t_read_idx >= IF_FILT_LEN )
                t_read_idx = 0;

            coef_real = (t_coef >> 16) & 0xFFFF;
            coef_imag = (t_coef & 0xFFFF);
            hist_real = (t_hist >> 16) & 0xFFFF;
            hist_imag = (t_hist & 0xFFFF);

            // Create the products
            p1_real = coef_real * hist_real;
            p1_imag = coef_real * hist_imag;
            p2_real = coef_imag * hist_imag;
            p2_imag = coef_imag * hist_real;

            // Combine products & reduce bits
            res_real = p1_real - p2_real;
            res_imag = p1_imag + p2_imag;
            res_real >>= 12;
            res_imag >>= 12;

            // Accumulate products
            accum_real += res_real;
            accum_imag += res_imag;
        } while ( i < IF_FILT_LEN );
```

Block type: pipeline
Latency: 2
Rate (cycles/result): 1
Max. Unit Delay: 97
Effective Rate: 97

The Impulse tools accept all of these forms of design description. The Impulse design flow supports verification using standard C tools such as Visual Studio and Eclipse. The Impulse tools facilitate the partitioning of code between hardware and processing elements of the system.

Applications remain device independent through most of the development process. After functional verification, application developers specify a target FPGA or platforms, and the design portions to be executed in the FPGA are exported to Xilinx ISE, Altera Quartus or Synplicity Synplify to be compiled to the FPGA.

Available fixed- and floating-point FFT library elements include:

- 1D/2D transforms
- Data flow: Burst or continuous
- Radix kernels: Powers of a prime
- Fixed point precision: 12-24 bits (higher bit widths on request)
- Floating point precision: Single precision IEEE-754 compatible (32 bits)
- Dimension sizes: D1 and D1xD2, where D1 and D2 range from 12-1296 in powers of 2^a, 3^b, and 5^c; also a <= 7, b <= 4 , and c <= 2.
- C/Matlab micro-architectural models are available (these models are Root-Mean-Square (RMS) error accurate).
- Netlist: Vendor-independent synthesizable VHDL or Verilog (ask about other formats).
- Size/performance: Highly competitive. Minimum multiplies. Based on prime factor Winograd and Number Theoretic Transform Kernel technology.

The new C-based Double (2x) Radix-4 Kernel reference design goes a step further. It comes pretty much ready to run such that custom logic can be “spliced” in.

Methodology

1. We began with known good design sample from Green Mountain computing.
2. We extended it into a library. To make the design to work for all point sizes the designers chose to create a library of common parameterized FFT results to use for verification. We chose values from 256 and up.
3. We accelerated over the software version by parallelizing the code into 2 radix-4 butterfly kernels so as to double the processing throughput. This pattern may be repeated for even further acceleration.
4. We verified by comparing to an existing C code FFT. This “golden” standard enabled us to constantly verify that the hardware accelerated output stayed equivalent to the original software filtered code. This is called a multi-modal test bench as it combines standard FFT libraries and the Impulse C verification tools to confirm equivalence.

Optionally, data may be interfaced to the host via PCIe drivers, providing a robust, common, understandable architecture that does not require the developer to understand or implement drivers on their own.

Details

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:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi i(\frac{k}{N})} \qquad 0 \leq k \leq N-1$$

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

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 .

Double(2x) Radix-4 kernel algorithm
To double the throughput, we can use two 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 achieved 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 by 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.

Software and hardware
The software Impulse C test bench first generates a data stream in test_producer(), then

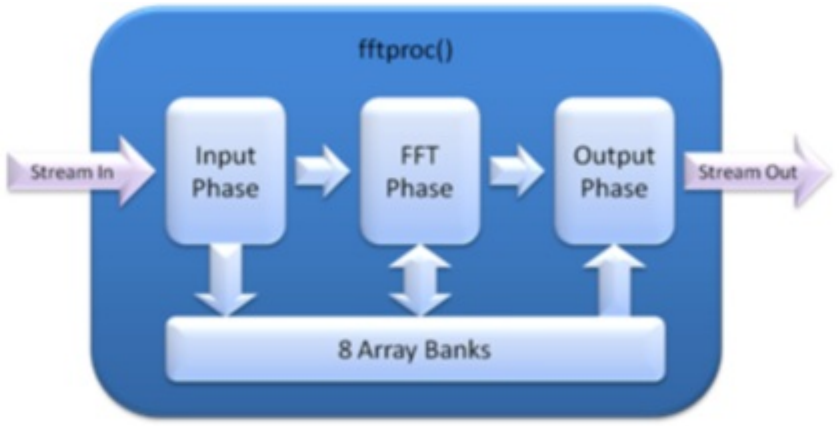
computes the hardware model 2xRad4 FFT of the data in fftproc(), and then streams the results back to the test_consumer() to display the results.

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 hardware 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 point size here 256 or 4096 then recompile
//#define FFT256
//
#define FFT4096
```

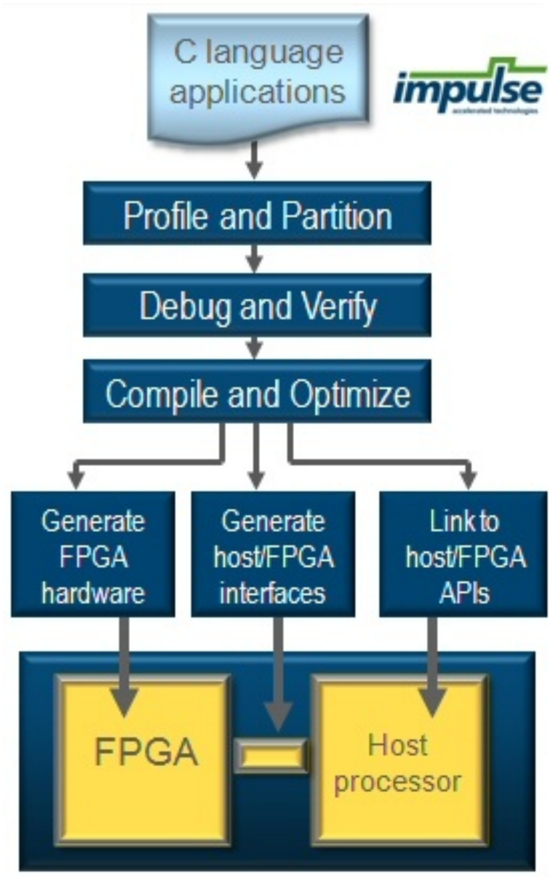
Processing flow
Below is a diagram of the FFT processing flow performed in fftproc(). There are three basic phases to the designed flow:

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



Connecting to the host
For this design the engineers needed a robust, high bandwidth connection from the FPGA to the host. PCIe fit the bill, but in order to obtain "hot rod" level performance the engineers selected a PCIe driver from Accelize. That driver enabled…

The overall C-to-FPGA tool flow: Most work is done in Visual Studio or the equivalent and unlike older processes where it was required to freeze code before marrying to hardware, a system architect identifies the likely HW/SW partitioning points and directs a portion of the project to compile to hardware.



C-to-FPGA flow

The code block may be imported from a legacy algorithm. In this case it is profiled for opportunities to be refactored for parallelism. If the code is original it can be built using “wizards” which facilitate by establishing I/O, memory and other boundaries for the code. In either case the tool set becomes an iterative environment wherein the design can be optimized for stage delay and parallelism.

Once the level of desired parallelism is achieved the big variance from typical software development occurs. The file is exported to synthesis using synthesizable VHDL as an intermediate file format. Depending on file size, the synthesis to FPGA hardware can take hours.

Another use of the synthesizable VHDL is export to ModelSim or a comparable VHDL simulator. This allows a level of simulation beyond desktop. Verification at this level reduces the risk of failure at the board level and is “insurance” typically endorsed by half or more of the sophisticated HW/SW teams we work with.

Import or create C code, analyze flow for bottlenecks, and export to ModelSim and Waveform Verification

Architecture options
Floating-point vs. fixed-point: Floating-point lets you ignore the dynamic range of the input data but takes more resources. If the range is limited, fixed-point can often achieve equivalent results in less resources.

Single vs. double floating point: Double is possible but would take up significant resources, although with FPGAs growing as they are this is becoming less of an issue. Also, a double implementation allows direct comparison with 64-bit double precision software FFTs.

For compactness and performance, one architectural consideration is to embed other functions within FFT math; for example, to push some algorithm logic into the FFT. For instance, this makes it easier to tweak the twiddle coefficients to size them based on the application.

Also, windowing and scaling can be integrated into the FFT to reduce the resources used. Design begins device-independent and highly portable to a selection of FPGAs. It can be compacted on the cheapest possible device or ported to a larger FPGA that provides headroom on the chip for system-on-chip for further integration. This also provides a means to later increase the size of the FFT to increase the resolution or frequency domain.

What of the future?
Intelligent video will use more custom FFTs and in more ways. Convolution processors will be inexpensive, low resource, and common. Applications like face recognition, object tracking, and threat identification will become more usable and available on cheaper devices. For instance, the Fourier-Mellin transformation allows you to see how an object got rotated. Within an image stream, this can be used to find objects in a way that are scale and angle independent. In this manner a face recognition program can quickly capture data as a person walks by a camera. The term Polar refers to the ability to work in radii and angles in addition to the normal values offered by an FFT.

Heterogeneous FPGA programming will increase in which VHDL, Verilog, and C are combined in one set of design files. Elements within the code will be directed to compile to run in FPGA hardware or in software on optional FPGA cores. This driver may be system-on-chip deployment and its associated reliability, power efficiency and small footprint. Or the driver may be redeployable modules. At any rate, the HW/SW line will only continue to blur, and software engineers will continue to dive into the hardware domain.

About the authors

Dr. Alan Coppola is President of OptNgn Software, LLC, (**www.optngn.com**) which produces FFT libraries for FPGAs. He has worked at Cypress Semiconductor, Intel, Mentor Graphics and has been in academia.

Dr. Coppola has been part of teams that successfully launched a number of EDA platforms and products used by design engineers around the world, including the first VHDL programmable logic platform. Dr. Coppola holds degrees in mathematics from the University of Connecticut, and SUNY at Binghamton.

Brian Durwood is Co-founder of **Impulse Accelerated Technologies Inc.** (a private company), makers of the popular Impulse C to FPGA optimizing compiler.

Mr. Durwood has also served as a VP for a Tektronix subsidiary and Product Managed Data I/O's Universal programmer line and ABEL software. Mr. Durwood is a graduate of Brown and Wharton.

EMAIL THIS PRINT COMMENT

More Related Links

Max's BADASS Display: A Comedy of Errors

ESIstream vs. JESD204B for Ultra-High-Speed Chip-Chip Communications

Startup Claims to Speed Creation of Custom EDA Tools

Pleasant Surprises for Field Programmable Radio Frequency Users

Stop! Recycle Those Old Batteries!