A 128-Tap Complex FIR Filter Processing 20 Giga-Samples/s in a Single FPGA

Florent de Dinechin, Honoré Takeugming
LIP (CNRS/INRIA/ENS-Lyon/UCBL)
Université de Lyon
Email: florent.de.dinechin@ens-lyon.fr

Jean-Marc Tanguy
Alcatel-Lucent France
Email:jean-marc.tanguy@alcatel-lucent.com

***Abstract*—To enable 40Gb/s data transmission over optical fibres using QPSK modulation, the first step of the receiver signal-processing pipeline is a 128-tap FIR filter that compensates the chromatic dispersion due to the medium. We present an implementation of this FIR filter in the largest Stratix-IV GX device that is able to process 20 giga-samples per second, where each sample is a complex number with 5+5 bits resolution. This FFT-based architecture processes 128 complex samples per cycles at a frequency of 156MHz. The FFT and inverse FFT pipelines use ad-hoc memory-based constant multipliers well suited to the FPGA features, while the multiplications in the Fourier domain use the FPGA embedded DSP blocks. This FPGA is thus able to perform more than 2 tera-operations per second. The precision of the intermediate signals is chosen to ensure that the error of the output signal with respect to the Matlab reference is never more than one least significant bit.**

I. Introduction

The TCHATER project aims at demonstrating a coherent terminal operating at 40Gb/s using real-time digital signal processing (DSP) and efficient polarization division multiplexing [1]. The terminal will benefit to next-generation high information- spectral density optical networks, while offering straightforward compatibility with current 10Gbit/s networks.

Fig. 2 describes the main tasks to perform, and the board-level architecture under design. This article surveys the first DSP step of this terminal, a large and high-bandwidth finite impulse response (FIR) filter whose task is to compensate the chromatic dispersion (CD) of the fiber for one polarization. This is the box labelled *Chromatic dispersion compensation* on Fig. 2.

Without detailing the application at large, the constraints for this step to enable 40Gb/s transmission are as follows. For each of the two polarizations, the optical signal is sampled at 20GHz with a resolution of 5 bits for each of the imaginary and real parts (there is a factor 2 oversampling here). The input bandwidth to each of the two first parallel FPGAs must therefore be (5+5) bits at 20 GHz, or 200 Gb/s. The analog-to-digital converters (ADC) demultiplex this bandwidth by a factor four, enabling data transmission over high-speed serial links operating at 5GHz. We therefore need 40 such links on each FPGA, which maps the capability of commercially available high-end FPGAs. Two parallel FPGAs consume this data, and produces an equivalent output bandwidth, which is sent through standard I/O pins to a third FPGA performing the rest of the DSP pipeline.

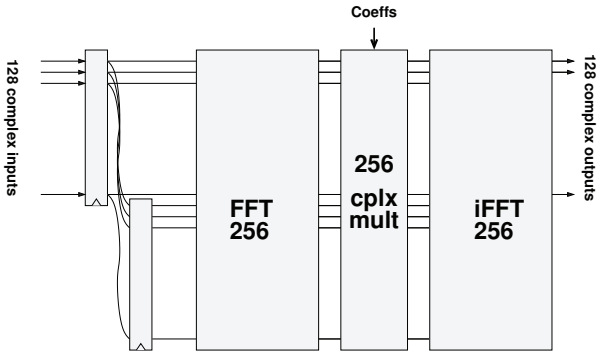


Fig. 1: FFT-based FIR implementation

The main application constraints are actually on the input/output. FPGAs providing enough I/O bandwidth also provide massive amounts of processing power, which is exploited in this paper to implement the main DSP task of each of these input FPGAs, a Finite Impulse Response (FIR) filter of at least 100 taps.

To our knowledge, none of the commercially available FIR implementations offers the required performance, even one fifth of it. Fortunately, we need very low resolution since the signals are sampled on 5 bits. Still, we shall need more than 5-bit accuracy in intermediate computations to ensure that the output signal is not turned into noise due to the accumulation of tens of rounding errors in the processing.

FPGAs may only compute at a frequency much lower than the 5GHz of data input: we aim at 5GHz/32=156.25MHz. Therefore, the first task of the FPGA is to demultiplex the data to this lower frequency. This is achieved using a combination of hardwired SerDes (serializer-deserializer) blocks and soft logic. At this point, we have at each 156MHz cycle a vector of 128 complex samples.

II. An FFT-based FIR

As we now have at each cycle a vector of consecutive samples that arrives in parallel, it is natural to use the FFT to perform the FIR in the frequency domain, with a pipeline depicted by Fig. 1.

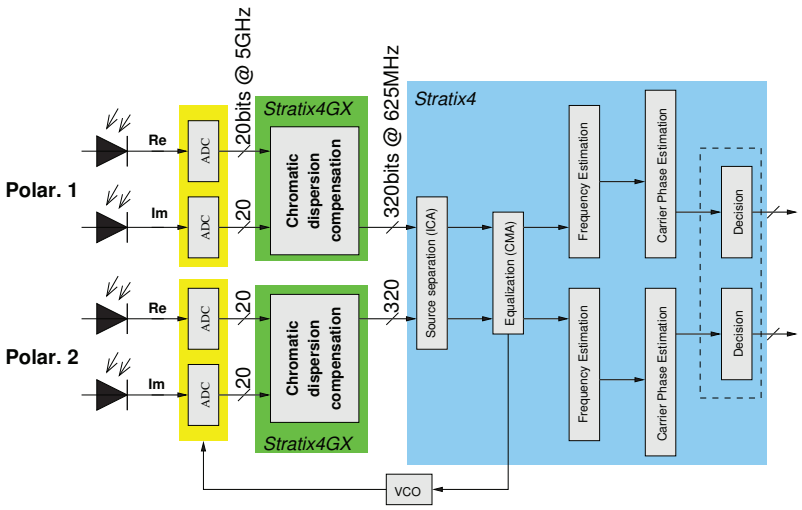


Fig. 2: TCHATER pipeline overview

TABLE I: Features of the Stratix IV EP4SGX530 relevant to this project [2]

high-speed serial links	40
standard IO ports	904
Arithmetic/Logic Modules	212480
6-input LUTs	414960
1-bit registers	414960
DSP blocks	1024 9x9, or 256 complex 18x18 multipliers
M9k blocks (9 Kbits)	1,280
M144k blocks (144 Kbits)	64

A. Arithmetic matching

An FFT-based FIR also happens to perfectly match the resources available in the FPGA, summed up in Table I.

Specifically, the application requires that the coefficients of the FIR may be changed, typically to adapt to commutations of optical fibers. For a 128-tap FIR, we therefore need 256 complex multipliers, by filter coefficients which will be held in registers. This perfectly matches the hardwired DSP blocks in the largest StratixIV GX.

All the other multiplications, in an FFT-based FIR, are multiplications by constant values (the roots of unity), and we now describe possible implementations of these, using the remaining FPGA resources: arithmetic and logic modules (ALMs), and embedded memories (M9K for 9Kbit memories).

B. Defining the precisions used along the datapath

The pipeline inputs and outputs samples with a resolution of 5 bits, and performs tens of operations on them. Obviously, we need to use an intermediate precision larger than 5 bits if we want any accuracy in the results. This section discusses this issue.

First consider the FFT. A 256-point FFT is needed for a 128-tap FIR filter. We chose a radix-4 FFT consisting of 4 butterfly stages, each stage composed of a row of complex multipliers by some $e^{\frac{2\pi kj}{2^n}}$, and two rows of complex additions. The first row of constant multipliers actually only multiply by 1 or -1. The following rows multiply by $e^{\frac{2\pi kj}{16}}$ then $e^{\frac{2\pi kj}{64}}$ then $e^{\frac{2\pi kj}{256}}$. We have to ensure that every computation is meaningful, in particular that we take into account even the results of the multiplications by the smallest constants (e.g. $sin(\pi/256) \approx /0.0245$).

As we start with 5-bit signals and end with 18-bit hard multipliers, a solution that minimizes both rounding errors and resource consumption is to let the datapath width grow, avoiding in particular any rounding in addition. Fig. 3 shows the sizes in bits of the intermediate signals in this case. The notation $p.q$ describes a fixed-point format with p bits in the integer part and q bits in the fraction part. The following details how we came to the formats on this figure.

Let us first consider the range of the data (which defines the number p of integer bits in the fixed-point format).

- Each constant multipliers produces a result of the same order of magnitude as its input, in other words p is the same before and after a multiplier. Although there is a scalar addition in the implementation of a complex multiplier, this addition should never overflows in the case of multiplications by roots of unity, since they do not increase the module. Actually, this assertion may be false in the rare case of extremal values combined with roundoff errors away from zero. However, this situation is avoided a-priori in our application, by setting the ADC gains so that extremal values are not used. Another option would have been to use saturated arithmetic, but at a much higher cost.
- However, we have to keep the overflow bit of each complex addition, wich means that p grows.

We arrive at $p = 9$ at the end of the FFT. As this data is input to DSP-based complex multipliers that have 18-bit resolution, we must have $q \leq 9$ so that $p + q \leq 18$. The next

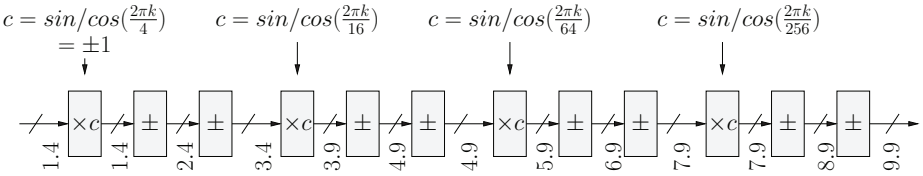


Fig. 3: Fixed-point precisions in the FFT. All the operations shown are complex operations.

design choice is to try $q = 9$, then retrofit this $q = 9$ to all the FFT datapath: this will entail that all the additions are exact, thus minimizing rounding error. The two last constant multiplications have identical input and output format. The first multiplication also, as it is exact (multiplications by $1, j, -1$ or $-j$). The precision $q = 9$ is actually introduced by the second constant multiplication.

Combined with the ad-hoc constant multiplication techniques of next section, this design choice ensures very high accuracy while keeping resource consumption within the range of the FPGA.

After multiplication by the filter coefficient using DSP blocs, we have to compute an iFFT that will ultimately output the data with 5-bit resolution. In this iFFT, we currently use constant k-bit precision for all the operations. Only the final result is rounded back to 1.4 format. The value of k is the largest possible such that the design fits the target FPGA and runs at the target frequency of 156MHz. Currently, $k = 14$. As Fig. 4, right shows, for this value of k, the accuracy of the whole pipelined, measured by simulation, is very good (error always smaller than one unit in the last place, or 1/32). A value of $k = 18$ would provide perfect accuracy (Fig. 4, left). This better design actually fits the FPGA, but we were so far unable to have it run at the target frequency.

As the application is latency-insensitive, the design is pipelined with two pipeline levels per constant multiplication and one per addition, for a total of 20 cycles for the FFT or iFFT.

Let us now review the implementation of the constant multipliers used in the FFT and iFFT pipelines.

III. Ad-hoc constant multiplier design

The multiplication of a complex constant $a + ib$ by a complex number $x+iy$ is equal to $(ax+by)+i(bx-ay)$. We use, for different sizes, four variations on the idea of tabulating constant multiplication. In all this section, we focus on the four products ax, by, bx and ay. The two additions of a complex product are implemented the standard way.

A. Simple tables

For 6-bit (or less) products, we can use 64-entry tables adressed by input data on 6 bits, well matched to the Stratix ALM structure [2, Fig. 2.7] used as dual 6-input look-up table (see Fig. 5). In this case, we need two ALMs per output bit.

Another option is to use M9K memories configured as dual-port $2^9 \times 18$ (see Fig. 6). Here, each 18-bit table entry holds the concatenation of ax (on 9 bits) and bx (on 9 bits), x being the address.

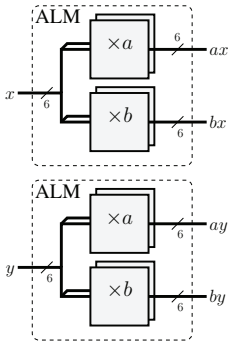


Fig. 5: Tabulating a complex constant multiplication in ALM

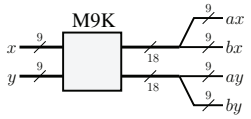


Fig. 6: Tabulating a complex constant multiplication in M9K

In each case, the data from each table is used twice, so these solutions are quite resource efficient: one could claim, for instance, that one M9K of Fig. 6 computes 4 9-bit products at 300MHz, so the correponding cumulated peak performance for the whole FPGA is 1280×4×300M = 1.5 TOp/s, where the Op is a 9-bit multiplication with a real constant.

One strength of this approach is that the accuracy is better than using a multiplier, since the result stored in the table is the correct rounding of the product by the real number $sin(\frac{2\pi k}{2^s}j)$. Using a multiplier, we would have to first round the real constant to some finite precision value, then to round the product, leading to a combination of two rounding errors.This good accuracy is all the more important as these techniques are used for small precisions.

B. Variations on the KCM algorithm

The two other multiplier techniques used are variations of the KCM idea [3], [4] adapted to fixed-point product. For instance, a 18-bit x input is decomposed into two 9-bit numbers $x_1 + 2^{-9}x_0$, and the product ax is equal to $ax_1 + 2^{-9}ax_0$, tabulated in two tables, ax_1 and ax_0. For an

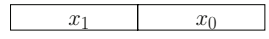
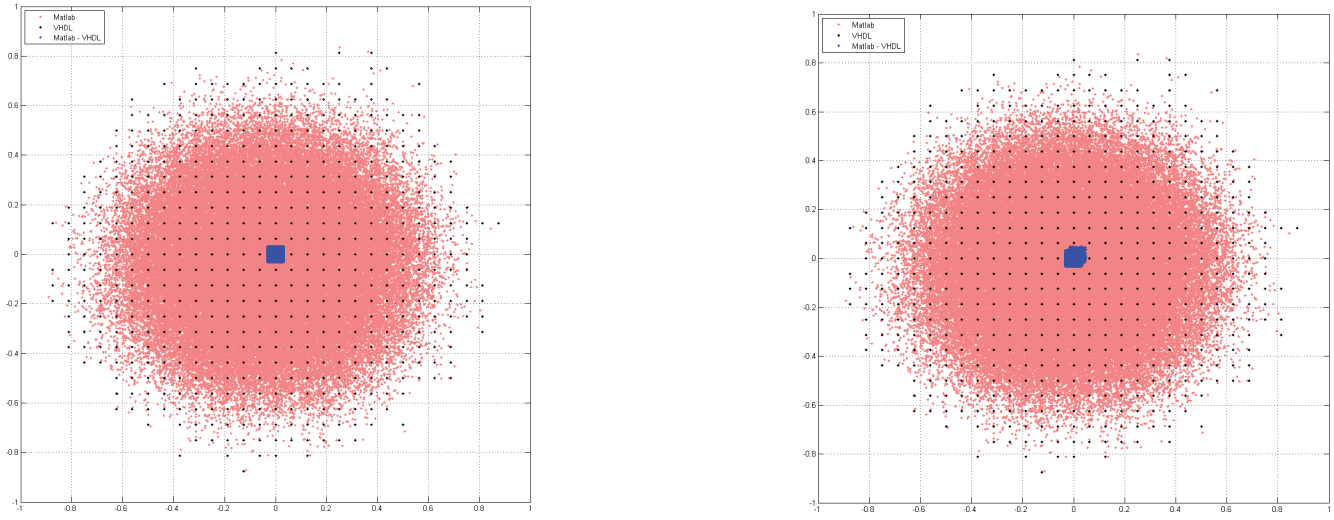


Fig. 7: Splitting a $2k$-bit number in two $k - bit$ chunks



(a) Inverse FFT computed on 18 bits

(b) Inverse FFT computed on 14 bits

Fig. 4: Plots of the result computed by our implementation (darker dots with 5-bits resolution), against the results computed in double-precision by Matlab (lighter dots). The dark square in the center is the plot of the difference between the two. In both cases this design is always last-bit accurate with respect to the Matlab result. On this limited simulation, the 18-bit implementation is always as accurate as rounding the Matlab result to 5 bits.

output precision of 18 bits, we tabulate ax_1 on 18 bits (this consumes two M9K), but we need only tabulate ax_0 on 9 bits (one M9K) since it is scaled down by 2^{-9} with respect to ax_1. If both tables contain correctly rounded product, the sum is computed with a accuracy of 1 unit in the last place, which is still good (and equivalent to the truncation of an exact multiplication). Remark that this decomposition is compatible with Fig. 6, so one 18-bit constant complex multiplication consumes three M9K used as per Fig. 6.

The 1280 M9K of the target FPGA (see Table I) allow us to implement 426 such multiplications. They are used for almost two multiplier columns of the inverse FFT. The other multiplications of the iFFT use the same idea, but splitting the input x into 3 6-bit chunks that are tabulated in ALMs. The multiplications of the FFT also all use ALMs.

IV. Results and future work

This design, along with the deserialisation logic and a smaller 4-tap interpolation filter compensating the difference in optical delays in the incoming fibers, consumes 100% of the DSP resources, 100% of the M9K resources, and 92% of the logic resources. The pipeline depth of the FIR is 20+3+20 cycles, and it runs at slightly more than 156MHz. It is last-bit accurate with respect to a double-precision Matlab computation, as Fig. 4 shows.

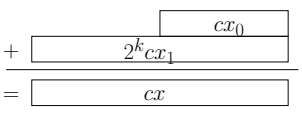


Fig. 8: KCM-like multiplication of a fixed-point number x by a real constant

The main issue with this design is that its natural floorplan (Fig. 1) poorly matches the physical structure of the target FPGAs. For instance, data is input on both sides of the chips, and the physical DSP blocks are grouped in several columns spread over the chip. This leads to long wires and makes the placement and routing difficult for the tools – synthesis takes several days. Logic partitionning helps a little, but we couldn't find a sensible partitionning of the logical design that could match a partition of the phyical chip.

Current work mostly consists in building the experimentation board for the TCHATER project, and completing the programming of the remaining FPGA (on the right of Figure 2).

In the longer term, we hope to build on this experience to investigate a more automated approach to the design of this type of pipelined FFT operators, possibly in the FloPoCo project (www.ens-lyon.fr/LIP/Arenaire/Ware/FloPoCo/). FloPoCo already incorporates multipliers of a real constant by a fixed-point number.

References

[1] J. Renaudier, "Coherent-based systems for high capacity wdm transmissions," in *Optical Fiber communication/National Fiber Optic Engineers Conference*, 2008.
[2] *Stratix-IV Device Handbook*, Altera Corporation, 2008.
[3] K. Chapman, "Fast integer multipliers fit in FPGAs (EDN 1993 design idea winner)," *EDN magazine*, May 1994.
[4] *Implementing Multipliers in FPGA Devices*, Altera Corporation, 2004.