



Cellot Inc

Cellot

FPCL

Field Programmable Cell Logic

From Idea to Product & ASIC

**Architecture for
16 bits Complex, 1024 points FFT
Utilizing Standard Cells / Full
Custom FPCL Device**

This document is an introduction to Cellot and is not an offer to sell, or a solicitation to buy an equity position in Cellot. The information, data, drawings and the like contained in this document are proprietary to Cellot. The disclosure by Cellot of information contained herein does not constitute any license or authorization to use or disclose information, ideas or concepts presented. No part of this document may be disclosed or used for any purpose other than the review and consideration of information by the recipient. Information in this document is subject to change without notice and does not represent a commitment on the part of Cellot Inc.



Summary

Several FFT implementations have been investigated for different size and different lengths, for speed optimization and for size optimization which shows the great advantage of the FPCL technology. Below is the result for an 1024 points, 16 bit complex (16 bits real, 16 bits imaginary) FFT implementation. Investigating some algorithms, it has been found that the Cooley-Tukey radix-4 algorithm is best for speed, while the Cooley-Tukey radix-2 algorithm is best for size. The latest could implement the whole FFT in few cells, but would take long time for the transform. If time-sharing is executed, dozens of thousands of transforms could be performed, but the time for any transform would be counted in seconds.

The implementation presented below has been designed for speed optimization therefore the radix-4 algorithm has been chosen.

Following is a "straightforward" implementation results although we may implement the FFT in less real-estate (same transform time) or faster (same real-estate).

Based upon the Radix 4, Cooley-Tukey Algorithm and no accuracy compromise a sample of the indicative results for the 1M Bytes chip is presented below. For other chip sizes the Transfer Time is directly linked to path delay.

FFT TRANSFORM TIME (Micro-Seconds)	NUMBER OF K Bytes used	REMARKS
4.0	450	FPCL Standard Cells Implementation. One butterfly.
2.5	450	FPCL Full Custom Implementation. One butterfly.
24.5	80	FPCL Standard Cells Implementation.
15	80	FPCL Full Custom Implementation.
16.4	20	Using 4 Hardwired Multiplier-Accumulators and Standard Cells.
10.2	20	Using 4 Hardwired Multiplier-Accumulators and Full Custom Cells.

We can achieve transform times better than 2 microseconds (the best result known today for reprogrammable devices). For example, using 24 multiplier-accumulators and 150 K Bytes or 900 K Bytes (utilizing core only) Full Custom will lead to 1.25 microseconds transform time. The transform time is divided by two each time the real-estate is duplicated.

There is room to achieve better results, which have not yet been investigated:

- Compromising accuracy,
- Using 16 bits Floating Point,
- Optimizing the indicated implementation,
- Investigating other algorithms possibly better suited for FPCL (WFFT),
- Using LNS (Logarithmic Number System).



The Preferred Architecture

This document presents the architecture for an 1024 points, 16 bit complex (16 bits real, 16 bits imaginary) FFT implementation. Investigating some algorithms, it has been found that the Cooley-Tukey radix-4 algorithm is best for speed, while the Cooley-Tukey radix-2 algorithm is best for size. The implementation presented below has been designed for speed optimization therefore the radix-4 algorithm has been chosen.

The FFT radix-4 concept is explained in 'Appendix a'. The architecture of the radix 4 is such that it has 1024 equivalent blocks, each block is called butterfly. The throughput of one or more butterflies can be implemented in the FPCL in one clock. On the other hand – in one clock only part of a butterfly could be performed.

Appendix a shows the picture of the 1024 equivalent butterflies, each is built out of 4 parts. In a matter of fact each of these butterflies is built out of 8 parts where 6 of them are equivalent and the other two are equivalent as well.

Consider one butterfly is implemented – the transform is being done by re-using the butterfly block 1024 times, therefore the transform is completed in 1024 clocks. Therefore, if the FPCL is implemented in standard cells, the speed would be about 4 Micro Sec but if the FPCL is implemented in full custom – the speed would be about 2.5 Micro Sec. 'Appendix b' explains the difference between standard cells / Full Custom FPCL implementations. Accordingly, if two butterflies are implemented the speed would be 2 Micro Sec in standard cells and 1.25 Micro Sec in full custom.

As mentioned above, the Radix 2 algorithm was found as the best algorithm for size. It could implement the whole FFT in few cells, but would take long time for the transform. If time-sharing is executed, dozens of thousands of transforms could be performed, but the time for any transform would be counted in seconds. This document does not present this concept.

Some algorithms have not been investigated, although they might result in better performance:

- The WFFT (Winograd's Fourier Transform),
- Using the LNS (Logarithmic Number System) to implement any algorithm utilizing Logarithm
- Other

Our proposed implementation is a "straightforward" and is not the best possible one. Actually we may implement the FFT in less real-estate (same transform time), but it is too cumbersome to implement and explain at this point.

The FFT Accuracy

In the multiplications, the most significant 18 bits of the 34 bits available have been used. This means that the whole 34 bits multiplication is implemented. If an error with the value of the 19-th bit (LSB) is permitted the implementation results in less real-state (note that the difference between 15 and 16 is at the LSB value although all the first 5 bits are different).



The Die Usage

The following die usage numbers (of K bits) are according to the butterfly as shown in 'appendix a'.

For $\frac{1}{4}$ butterfly that have multipliers:

- Two 4 input (16 bits each) adders are needed. Each adder is implemented in 63 K bits and both n 126 K bits.
- Four 2 inputs (one 18 bits and one 16 bits) multipliers are needed. Each multiplier is implemented in 232 K bits. Note that there is no need to use two 18 bits input multiplier as the coefficient is 16 bits. The result for multipliers is 928 K bits.
- Two 2 inputs (18 bits each) adders are needed. Each adder is implemented in 38 K bits. If 16 bits only are used, each adder is implemented in 32 K bits. Therefore for both adders 76 K bits are considered.

For $\frac{1}{4}$ butterfly that do the sum only:

- Two 4 input (16 bits each) adders are needed. Each adder is implemented in 63 K bits. Some consideration should be done to use the same hardware for the two adders (only 63 K bits) as the same adder can operate twice at the "pipe-line" duration of the multiplier. Nevertheless, for the present, it is considered twice therefore 126 K bits are considered.

General purpose hardware:

- To hold the 1024 (real and imaginary) results, 32 K Bits for 16 bits are needed.
- For spare and the simple timing - 36 K Bits are estimated.

To implement one butterfly - sum the above. The result would be 3584 K bits \pm 2% which are 448 \pm 2% K Bytes. As clearly seen, most of the cells usage is for multipliers. There are 12 multipliers in one butterfly. The results of these multipliers are accumulated. Therefore, if 12 multiply - accumulators are added to the FPCL device, less than 75 K Bytes are needed to implement the FFT in time of 1024 clocks.

Time Calculation

The implemented butterfly (see appendix 'a') can input the new signals at each clock. To calculate one column, 256 butterflies are needed. To calculate the whole FFT, 4 columns are needed. Therefore, to calculate the whole FFT 1024 calculations of one butterfly are needed that means 1024 clocks, and few more for the pipeline delay.

If faster transform is needed, the hardware could be doubled. For example: the '1M Byte' device can implement two butterflies (using no extra hardware such as multiply - accumulators) and its clock rate is 4 nSec (standard cells) and 2.5 nSec (full custom). In such device the conversion time would be 2 Micro Sec and 1.25 Micro Sec accordingly. The '2.5 M Byte' device can implement five butterflies (using no extra hardware such as multiply - accumulators) and its clock rate is 5 nSec (standard cells) and 3.125 nSec (full custom). In such device the conversion time would be 1 Micro Sec and 0.625 Micro Sec accordingly.



Different Configuration for Time - Space

Implementing 1/8 of the formula with around 72 K Bytes will lead to 8 times the duration that means, the transformation will be accomplished in 8192 clocks which are equivalent to ~33 Micro sec in standard cells or ~20 Micro Sec in full custom. Nevertheless, adding ~8 K Bytes to implement the first part of the butterfly will lead to 6 times only. That means ~80 K Bytes will accomplish the transform in 6144 clocks which are 24.5 Micro Sec in standard cells or 15 Micro Sec in full custom.

If more space is needed – use Radix 2 implementation.

As mentioned above: doubling the hardware size - doubles the speed of the transform. Nevertheless, if the device of 2.5 M Bytes device is used to implement 5 butterflies for the best speed, the clock (of such device) is 5 nSec in standard cell and not 4 nSec so the speed would be in this case about 1 Micro Sec, and in full custom around 0.625 nSec, and not as expected from linear calculation.

On the other hand, if a smaller device is used to implement the 80K Bytes, e.g. one of the smallest FPCL devices, its typical speed is 400 MHz (2.5 nSec and not 4 nSec) in standard cells therefore the transform time would be 15 Micro Sec, and in full custom it is estimated to be less than 9 Micro seconds.

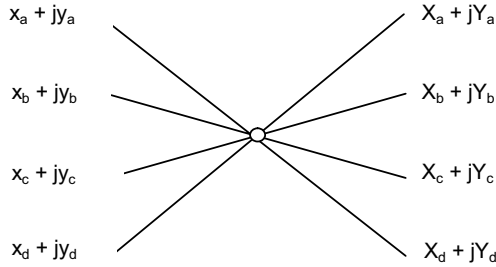
Additional information

1. The presented implementation may be realized utilizing smaller and larger FPCL chips as only the base FPCL cells are in use.
2. If 2 hardwired embedded multiplier-Accumulators are added to the FPCL device throughout the cross-connect, the usage (of the 80 K Byte configuration) will be around 16 K Bytes (same conversion time of 6144 clocks). Additional 2 multiplier-accumulators and doubling the hardware (the 16 K Bytes) can reduce the transform time by half. This process may be continued, e.g. if 12 multiplier-accumulators are available – use the full butterfly implementation (duration of 1024 clock) utilizing less than 75 K Bytes.
3. Implementing the FFT using a “Mantissa – Exponent” presentation (floating) and if Mantissa + Exponent = 16 bits, will result with a significant hardware saving (or be a lot faster).



Appendix a: The Cooley-Tukey radix-4 Implementation

One butterfly is calculated according to the following formula:



In the following, 'C' stands for the real part of the coefficient - cos(...), 'S' for the imaginary part - sin(...).

$$X_a = x_a + x_b + x_c + x_d$$

$$Y_a = y_a + y_b + y_c + y_d$$

$$X_b = (x_a + y_b - x_c + y_d)^x C_b + (y_a - x_b - y_c + x_d)^x S_b$$

$$Y_b = (y_a - x_b - y_c + x_d)^x C_b + (x_a + y_b - x_c + y_d)^x (-S_b)$$

$$X_c = (x_a - x_b + x_c - x_d)^x C_c + (y_a - y_b + y_c - y_d)^x S_c$$

$$Y_c = (y_a - y_b + y_c - y_d)^x C_c + (x_a - x_b + x_c - x_d)^x (-S_c)$$

$$X_d = (x_a - y_b - x_c + y_d)^x C_d + (y_a + x_b - y_c - x_d)^x S_d$$

$$Y_d = (y_a + x_b - y_c - x_d)^x C_d + (x_a - y_b - x_c + y_d)^x (-S_d)$$

For 1024 points, the result of such butterfly is routed into three more butterflies to have 4 DFT outputs. As 4 DFT outputs are generated at one construction, to have the full DFT results the upper line is to be implemented 256 times.

Any two of the upper lines could be implemented as the following. The result for X_b and for Y_b is shown for example.

The total number of cells for the following configuration is:

63	K bits	(~8 K Bytes)	for 4 inputs adder, each input is 16 bits
232	K bits	(29 K Bytes)	for multiplier
38	K bits	(~5 K Bytes)	for 2 inputs, 18 bits each adder

126	K bits	(~16 K Bytes)	for 2 adders
928	K bits	116 K Bytes	for 4 multipliers
76	K bits	9.5 K Bytes	for 2 adders

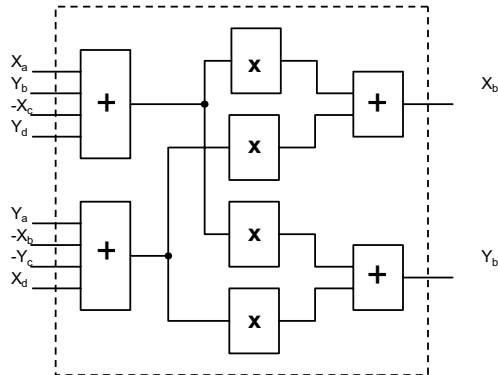
Total number of cell for the Radix 4 butterfly:

3453 K bits **432 K Bytes**

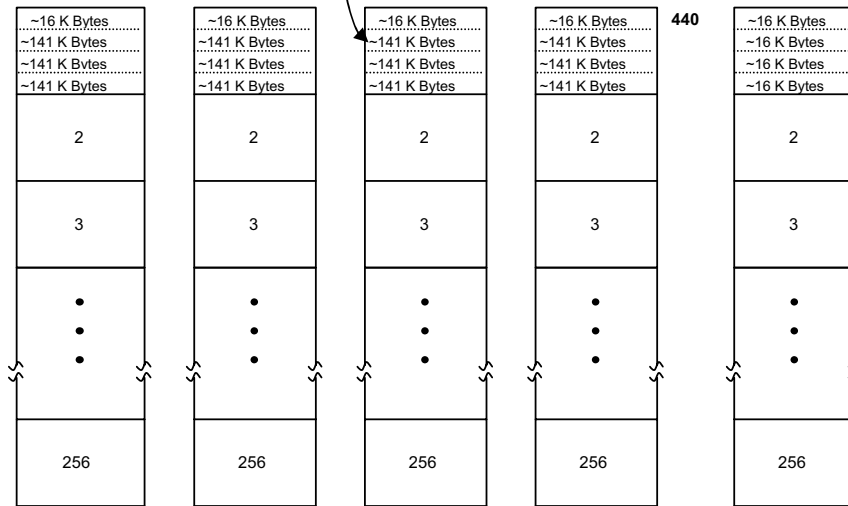


1/4 of One radix 4 butterfly:

The inputs to the multipliers that define the coefficient is not shown



Total number of cell for the Radix 4 FFT:



Once the result of each formula (done by the adder or by any of the three equivalent hardware shown above) each result is written into a memory to be called after the whole 256 iteration occur. An exception is the last column which its coefficient are all 1 therefore practically there is no multiplication: After the first four butterflies of the previous column is calculated, the results are routed into the last column adders and calculated along with the calculation of the previous column butterflies to save and calculated along with the calculation of the previous column butterflies to save transform time.



Appendix b: Full Custom Implementation

For full details, please read the patent publication.

The idea is to separate the chip into smaller devices. A 64K Bytes device is proposed here. As the amount of connections needed for connecting any point to any point in the device is very small, a 3 level matrix will be used. Each device is implemented in a closed small area. Integrating the two parameters: smaller area and smaller delay in the cross connect, results in increased speed.

To connect the devices without degrading performance (compare to a non separated chip) another three level matrix is required. Therefore, the overall delay – compared to a non separated chip – is increased, as to connect any point in the device to any other point 5 levels only are needed, and in the separated device – 6 levels.

Below there is a quote from a preliminary work done on the full custom implementation:

Distance between opposite corners of SRAMS = 6,000 microns.

Routing delay = $6 * 0.15 = 0.9$ ns

SRAM delay = 1.3ns output delay + 0.45ns setup = 1.75ns

Mux delay = 3 stages * 0.65ns = 1.95nSec

Total delay = $0.9 + 1.75 + 1.95 = 4.6$ nSec

Slow frequency = $1000 / 4.6 = 217$ MHz

Typical frequency = $217 * 2 = 435$ MHz

Fast frequency = $217 * 2.8 = 609$ MHz

For reference, the 1028 SRAMS version:

Distance between opposite corners of SRAMS = 20,000 microns

Routing delay = $20 * 0.15 = 3$ ns

SRAM delay = 1.3ns output delay + 0.45ns setup = 1.75ns

Mux delay = 5 stages * 0.65ns = 3.25ns

Total delay = $1.75 + 3.25 + 3 = 8$ ns

Slow frequency = $1000 / 8 = 125$ MHz

Typical frequency = $125 * 2 = 250$ MH

Fast frequency = $125 * 2.8 = 350$ MHz

As seen, the frequency in the internal device is doubled.

Please note that only the original frequency and the doubled frequency are permitted, therefore none of the FPCL features is compromised in this implementation including the emulator which retains the One-To-One feature. It is designed in such a way that its simulation speed decreases only to the parts for which the clock is doubled! As the work on the doubled speed device is not completed, it is safe to assume that the speed will be 2.5 nSec rather than the above speed (2.3 nSec).