Maxeler Apps
Fast Fourier Transformation

Jan 2016

Fast Fourier Transformation (FFT)

- Computes the discrete Fourier transform
  - Converts signal from its original domain (e. g., time, space) into the frequency domain
- Widely used in engineering, science and mathematics
- Definition:

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

Cooley-Tukey FFT algorithm

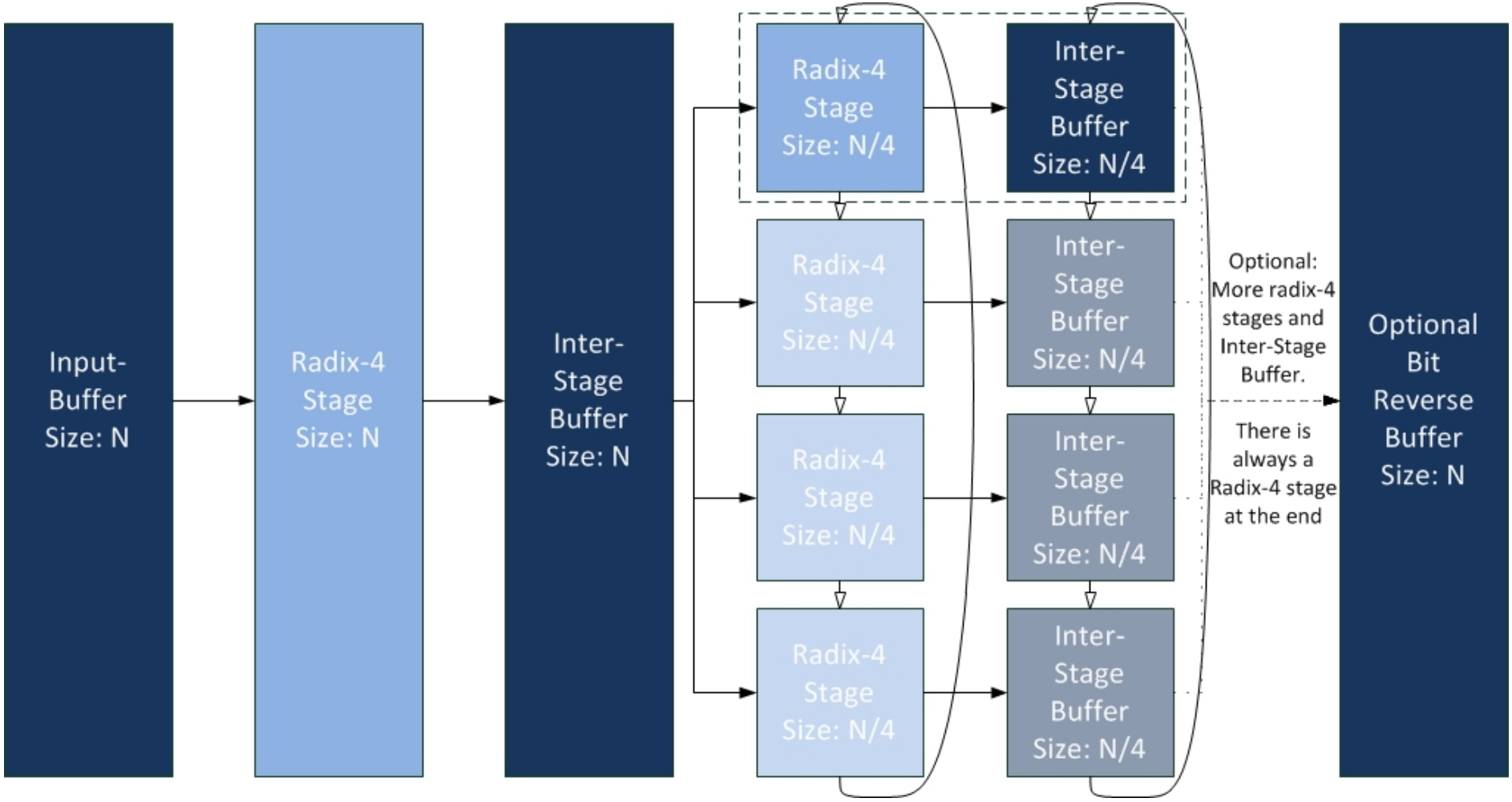
- Most common algorithm to calculate the FFT
- Reduces the required computation by using a divide and conquer method
  - Instead of calculating the FFT on all values calculate two FFTs on a subset
- Example: Radix-2 decimation-in-time (DIT) FFT

$$X_k = \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N}(2m)k} + \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N}(2m+1)k}$$

General Design Decisions

- To facilitate FFT usage in many application areas, minimizing resource utilization is crucial
- Experience shows FMEM as the bottleneck
- We use a decimation in frequency Radix-4 FFT
  - This allows us to reduce the buffer size after each stage by a factor of 4
    - In the case of Radix-2 twice as many and twice as large buffers are needed (on avg)
  - Also the latency gets significantly reduced

General Architecture



The four Radix-4, N/4 stages and the corresponding stage buffers are not executed in parallel. Actually they use the same hardware.

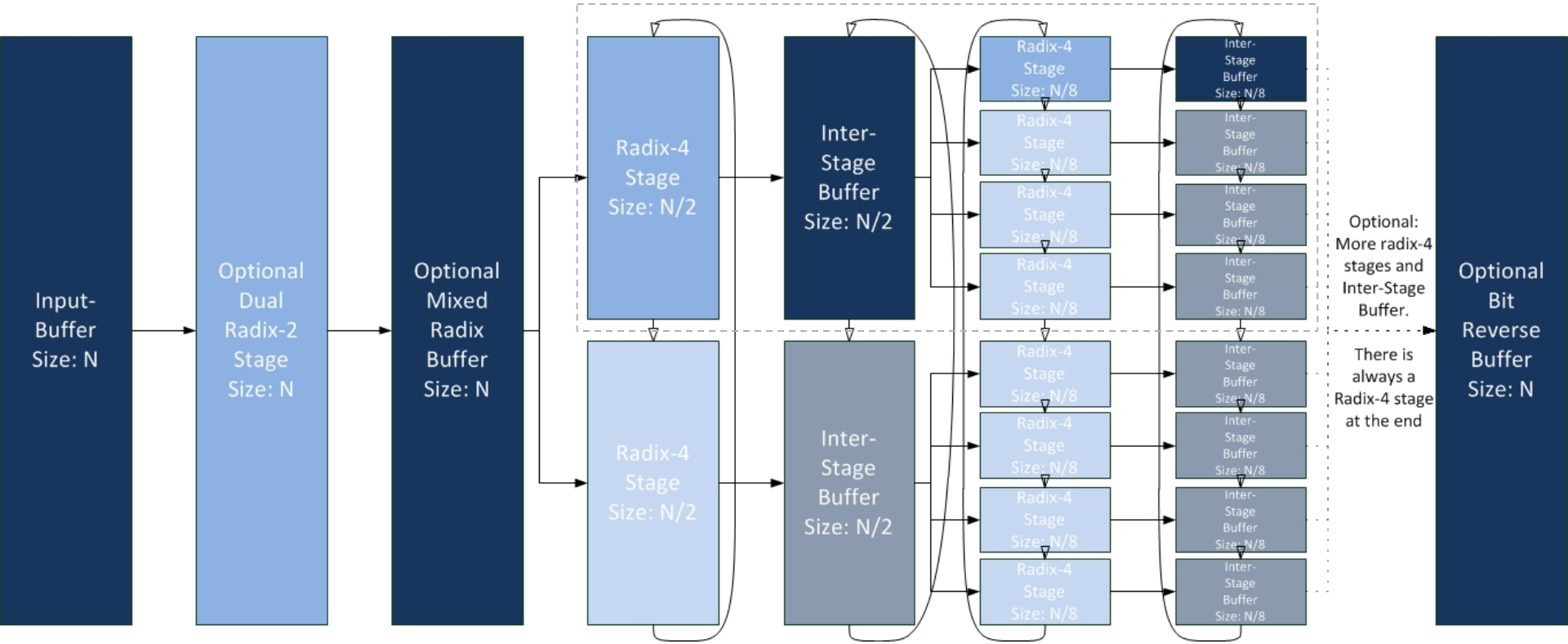
General Architecture – Explanation

- Algorithm:

1. Reorder the input buffer data
2. Apply the first Radix-4 butterfly stage
3. Reorder the data in an inter-stage buffer
4. Apply again the Radix-4 butterfly
5. Repeat 3 and 4 for $\log_4(\textit{FFT size})$ butterfly stages

- The above can only calculate power-of-4 point FFTs

General Architecture – Additional Radix-2 Stage



General Architecture – Additional Radix-2 Stage (Explanation)

- In order to also calculate power-of-2 point FFTs we add an additional Radix-2 stage at the beginning
- This change maintains the overall advantages of the Radix-4 FFT as stated earlier
- The additional full size buffer needed by the Radix-2 FFT introduces significant area overhead
- Hence power-of-4 sizes are should be preferred

General Architecture – Additional Notes

- Four samples per cycle are processed
  - This nicely fits to the Radix-4 butterfly since we can perform exactly 1 butterfly calculation per cycle
    - Or two Radix-2 butterflies per cycle
  - It is also good for optimizing PCIe utilization if we work with single precision floating point (16 bytes allow optimal use)
- Interfaces to calculate multiple FFTs in parallel are provided
  - This allows more efficient packing of data into memory as well as re-usage of control logic and twiddle factors
    - Cheaper as compared to calculating multiple FFTs separately
  - However, keep in mind that this complicates scheduling
    - Data has to be ready and be consumed at the same time for all FFTs

Radix-4 Butterfly

- Decimation in frequency Radix-4 DFT can be:

$$X(4k) = \sum_{n=0}^{N/4-1}\left[x(n) + x\left(n+\frac{N}{4}\right) + x\left(n+\frac{N}{2}\right) + x\left(n+\frac{3N}{4}\right)\right]W_N^0 W_{N/4}^{kn}$$

$$X(4k+1) = \sum_{n=0}^{N/4-1}\left[x(n) - ix\left(n+\frac{N}{4}\right) - x\left(n+\frac{N}{2}\right) + ix\left(n+\frac{3N}{4}\right)\right]W_N^n W_{N/4}^{kn}$$

$$X(4k+2) = \sum_{n=0}^{N/4-1}\left[x(n) - x\left(n+\frac{N}{4}\right) + x\left(n+\frac{N}{2}\right) - x\left(n+\frac{3N}{4}\right)\right]W_N^{2n} W_{N/4}^{kn}$$

$$X(4k+3) = \sum_{n=0}^{N/4-1}\left[x(n) + ix\left(n+\frac{N}{4}\right) - x\left(n+\frac{N}{2}\right) - ix\left(n+\frac{3N}{4}\right)\right]W_N^{3n} W_{N/4}^{kn}$$

- The butterfly itself only consists of a few additions and multiplications by *i*
- The more challenging part is the twiddle factor multiplication

Twiddle Factors

- The twiddle factors can be calculated as:

$$W_N^{kn} = e^{-i2\pi\frac{kn}{N}}$$

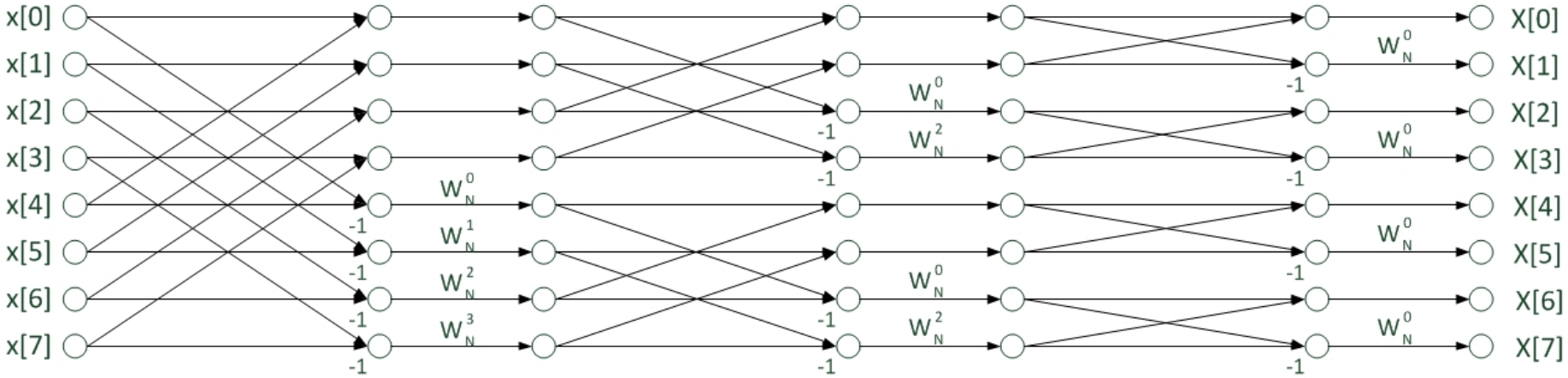
- For each Radix-4 stage the twiddle factors have to be calculated and stored separately since N changes
- The twiddle factors get stored in an ROM on the DFE
  - In order to save space we exploit the symmetries of the twiddle factor calculation and only store the first quadrant
  - Twiddle factors can be reused for multiple FFTs
  - Addressing of the ROMs between cycles is done using *simple counters*

Buffering

- FMEM is usually the scarcest resource
- The Buffers consume most FMEM
- The Buffers do not only store the data between stages but also reorder it
- This means that if you want to read and write in a trivial order (e. g., write linear and read in the required order) double buffering is a must
  - –This doubles the amount of needed FMEM memory
  - –To address the above, we need to write and read from the same address in order to use only a single buffer
    - So we developed complex addressing schemes for every buffer to avoid double buffering while providing the same functionality

Input Buffer

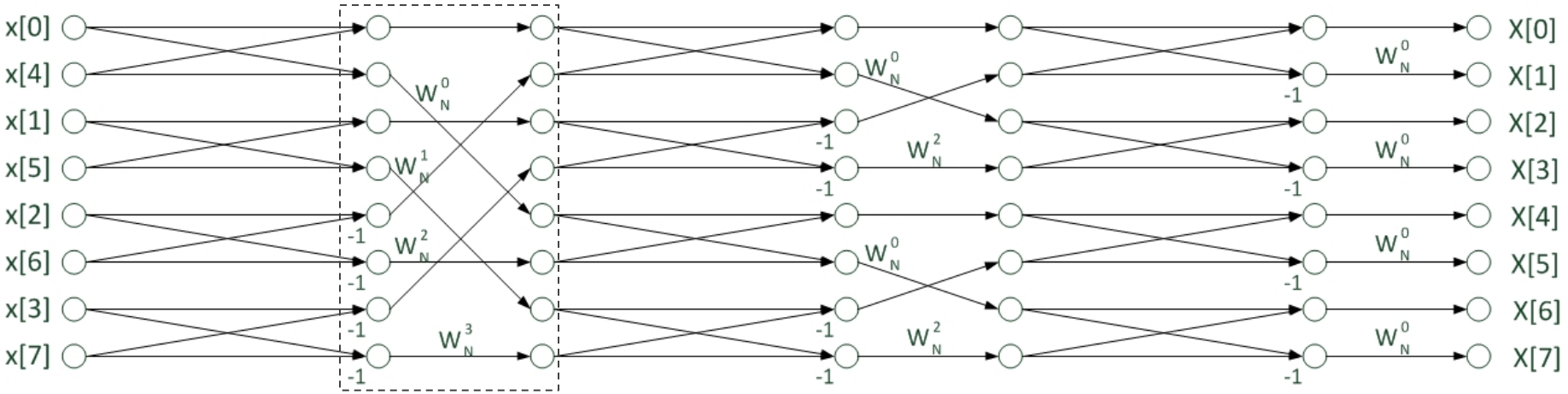
- Why we need the input buffer?
  - –Division in frequency FFTs consume data in linear order
- Let consider butterfly diagram for a Radix-2 8-point FFT



- The first butterfly for example needs the input values of 0 and 4 and produces values with the indices 0 and 4
  - – In case we calculate one butterfly on each cycle
    - We need to read and write in a different order

Input Buffer

- Below is the butterfly diagram where the butterflies are untangled



- The highlighted box shows the data reordering between the butterflies
- This enables the computation of one butterfly at every cycle
  - –Please mind the complex access pattern
- The same principles apply to Radix-4

2D FFT

- In order to calculate a 2D FFT we first calculate a 1D FFT on each row and then a 1D FFT on each column
- So we only need a 2D transpose
- Currently only the single buffered version is implemented for the special case when width and height are equal
- In case the 2D FFT has different sizes in different dimensions the 2D transpose is double buffered

3D FFT

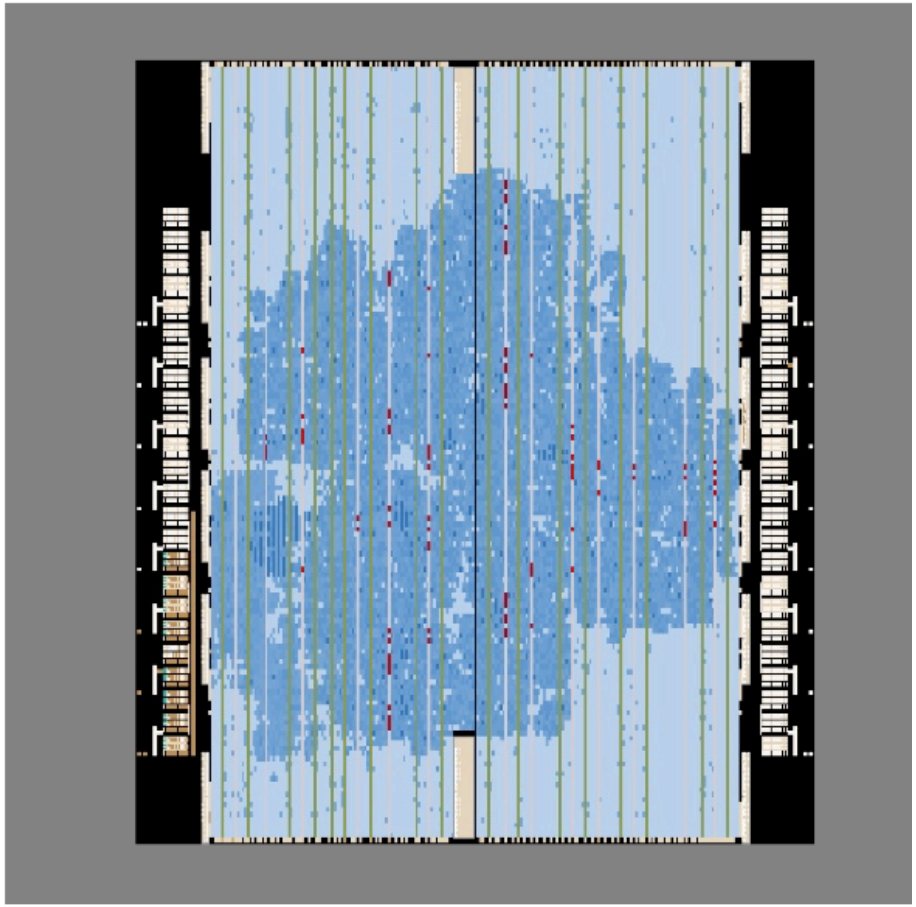
- For 3D FFT we compute 1D FFT in each direction
- 3D transpose is emulated using 2D transpose passes
  - The only downside to this is that for now in almost all cases the transposes are double buffered
  - For larger sizes (e.g., above 32x32x32) 3D FFT should be changed to use LMEM

Realization on Maia

- 32 bit floating point complex arithmetic
- 2D FFT of Size 512x512

- Kernel frequency = 200 MHz
- Logic utilization = 38%
- Multiplier utilization = 4.79%
- FMEM utilization = 100%

- The throughput is 800M values per second



2D FFT floorplan

MaxPower

- The source code of the FFT is accessible at: github.com/maxeler/maxpower
  - Additional explanation may be found there
  - The above information is also available upon request