Comparison of FFT Generated by LegUp vs. Altera Megafunction

FFT Specifications

Comparison will be for:

- 16-bit input and output width
- 64-point FFT (length 64 input and output)

FFT Algorithm

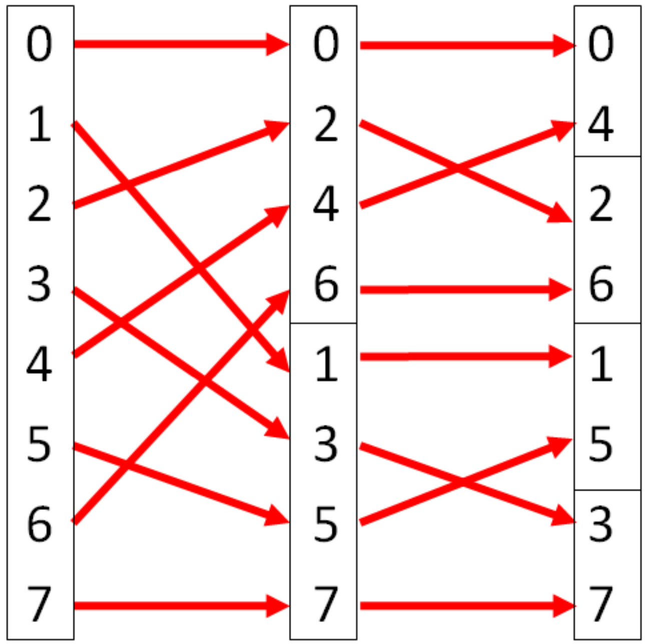
- Discrete Fourier Transform (DFT) algorithm is $O(N^2)$, where N = input size, X = input values, Y = output values

$$Y_k = \sum_{n=0}^{N-1} X_n \exp\left(\frac{-2\pi j}{N} kn\right), \quad k = 0, \ldots, N-1$$

- An alternative $O(N\log_2 N)$ algorithm exists (FFT) which takes advantage of symmetry. FFT contains 2 steps:
  1. Re-ordering the input data (time decimation)
  2. Recursively performing a computation known as the “butterfly operation”

Step 1. Time Decimation

Example with input Size 8 (Re-ordering data points 0-7)



With input size 4, the order would go from 0, 1, 2, 3 → **0, 2, 1, 3**

Step 2. Butterfly Calculation

- Example: input size 4. The $O(N^2)$ definition can be rearranged:

$$
\begin{aligned}
Y0 &= (x_0 + e^{-j2\pi(0)/2}\, x_2) + e^{-j2\pi(0)/4}\, (x_1 + e^{-j2\pi(0)/2}\, x_3) \\
Y1 &= (x_0 - e^{-j2\pi(0)/2}\, x_2) + e^{-j2\pi(1)/4}\, (x_1 - e^{-j2\pi(0)/2}\, x_3) \\
Y2 &= (x_0 + e^{-j2\pi(0)/2}\, x_2) - e^{-j2\pi(0)/4}\, (x_1 + e^{-j2\pi(0)/2}\, x_3) \\
Y3 &= (x_0 - e^{-j2\pi(0)/2} x_2) - e^{-j2\pi(1)/4}\, (x_1 - e^{-j2\pi(0)/2}\, x_3)
\end{aligned}
$$

Step 2. Butterfly Calculation

- Example: input size 4. The O(N^2) definition can be rearranged:

$$
\begin{aligned}
Y0 &= (x_0 + e^{-j2\pi(0)/2}\, x_2) + e^{-j2\pi(0)/4} (x_1 + e^{-j2\pi(0)/2}\, x_3) \\
Y1 &= (x_0 - e^{-j2\pi(0)/2}\, x_2) + e^{-j2\pi(1)/4} (x_1 - e^{-j2\pi(0)/2}\, x_3) \\
Y2 &= (x_0 + e^{-j2\pi(0)/2}\, x_2) - e^{-j2\pi(0)/4} (x_1 + e^{-j2\pi(0)/2}\, x_3) \\
Y3 &= (x_0 - e^{-j2\pi(0)/2}\, x_2) - e^{-j2\pi(1)/4} (x_1 - e^{-j2\pi(0)/2}\, x_3)
\end{aligned}
$$

Butterfly 1 Butterfly 2

Butterfly Stage 1

Step 2. Butterfly Calculation

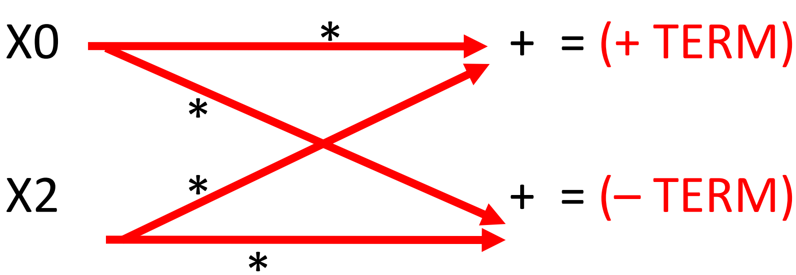
- Example: input size 4. The $O(N^2)$ definition can be rearranged:

$$
\begin{aligned}
Y0 &= (x_0 + e^{-j2\pi(0)/2}\, x_2) + e^{-j2\pi(0)/4}(x_1 + e^{-j2\pi(0)/2}\, x_3) \\
Y1 &= (x_0 - e^{-j2\pi(0)/2}\, x_2) + e^{-j2\pi(1)/4}(x_1 - e^{-j2\pi(0)/2}\, x_3) \\
Y2 &= (x_0 + e^{-j2\pi(0)/2}\, x_2) - e^{-j2\pi(0)/4}(x_1 + e^{-j2\pi(0)/2}\, x_3) \\
Y3 &= (x_0 - e^{-j2\pi(0)/2}\, x_2) - e^{-j2\pi(1)/4}(x_1 - e^{-j2\pi(0)/2}\, x_3)
\end{aligned}
$$

Butterfly 1 Butterfly 2

Butterfly Stage 1

Butterfly operation:



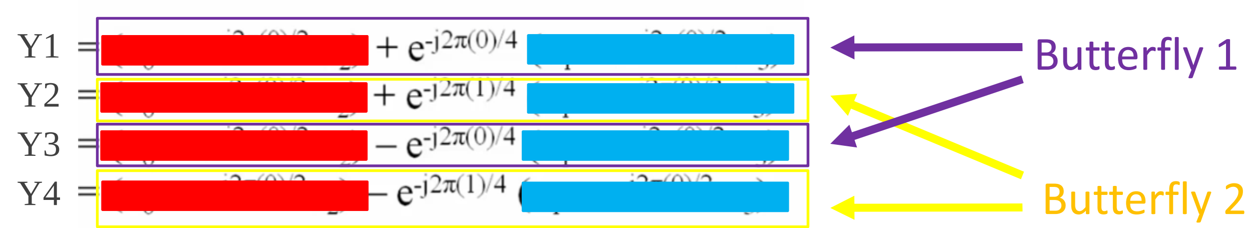
Step 2. Butterfly Calculation

- Example: input size 4. The $O(N^2)$ definition can be rearranged:

Butterfly Stage 2

Step 2. Butterfly Calculation

- Example: input size 4. The $O(N^2)$ definition can be rearranged:



Butterfly Stage 2

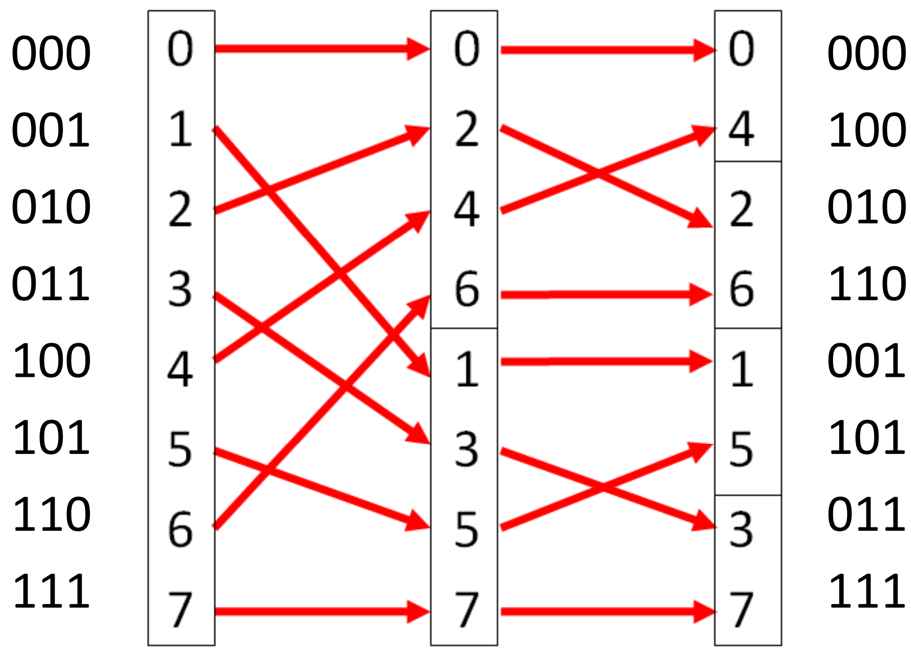
Stage 2 is the final stage. 2 Butterflies per stage, 2 computations per butterfly, and 2 stages, → Total number of calculations = 4 log 4 = **8**

C Implementation for LegUp

- Fixed point, sin lookup table, values stored as integers

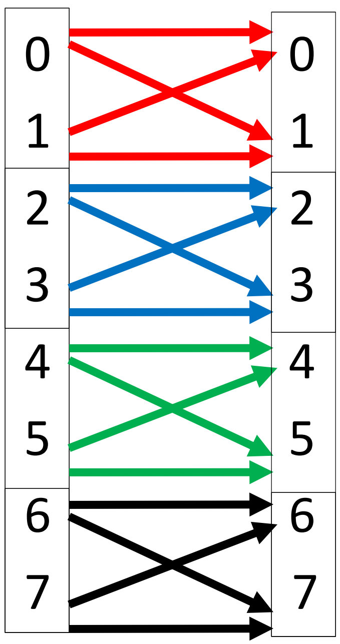
C Implementation for LegUp

- Fixed point, sin lookup table, values stored as integers
- Time decimation performed by reversing bits:



C Implementation for LegUp

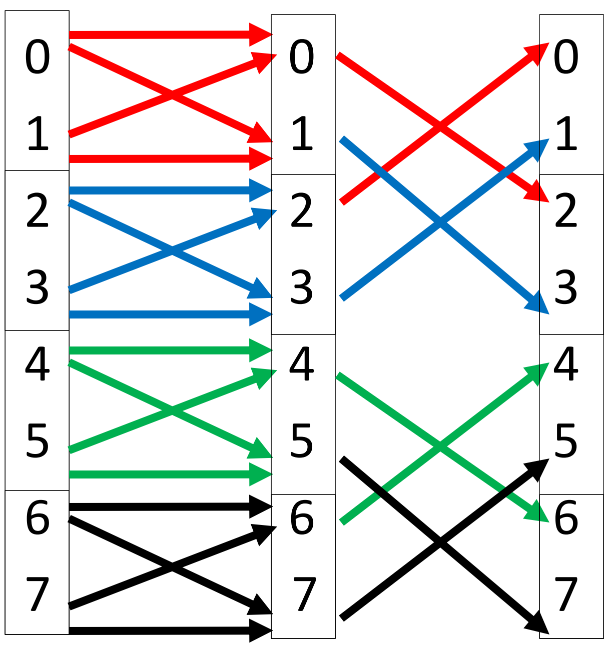
- Since LegUp currently does not support recursion, butterfly algorithm was implemented iteratively, e.g. with input size 8:



Stage 1/3

C Implementation for LegUp

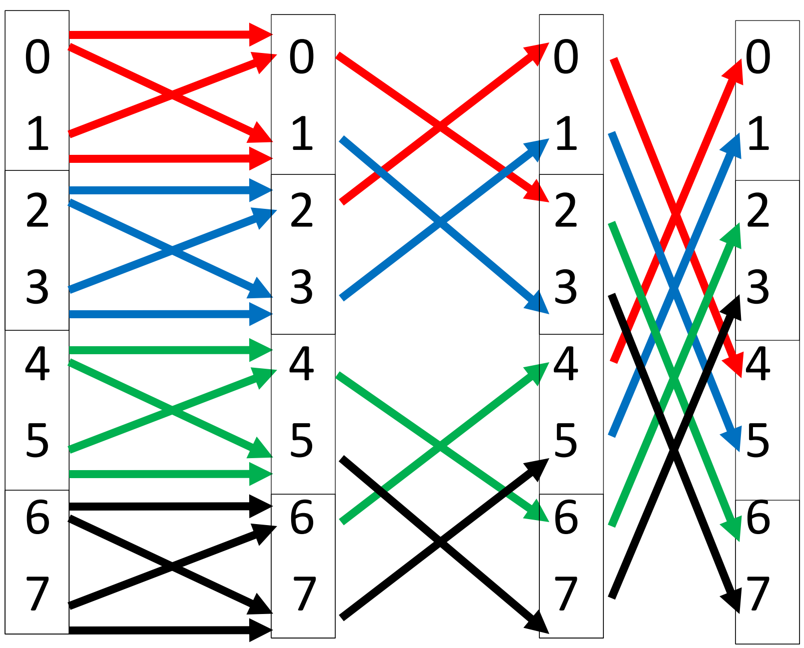
- Since LegUp currently does not support recursion, butterfly algorithm was implemented iteratively, e.g. with input size 8:



Stage 2/3

C Implementation for LegUp

- Since LegUp currently does not support recursion, butterfly algorithm was implemented iteratively, e.g. with input size 8:



Stage 3/3

Note that the butterflies in each stage are independent, hence in hardware should be performed in parallel

C Implementation for LegUp

- Pattern is true for all sizes (for 64 point FFT, 6 stages)
  - First stage: Butterfly (0,1), (2,3), (4,5) . . . (62,63)
  - Second stage: Butterfly (0,2), (1,3), (4,6), (5, 7) . . . (61, 63)
  - Third stage: (0,4), (1,5), (2,6), (3,7), (8,12). . . (59,63)
  - Fourth stage: (0,8), (1,9) . . .
  - Fifth stage: Skips by 16
  - Sixth stage: Skips by 32: (0,32), (1,33)… (31,63)

- Implemented in C using for loops, but then LegUp does not take advantage of parallel butterflies

Modifications to Initial C

- However since FFT size is known…
  - Time decimation ordering can be stored rather than calculated
  - Only required sin values can be stored, no need for large lookup

- In fact, *all* parts of the algorithm can be stored, computations are always the same for a given input size (only input values change)

Manual Verilog Implementation

Finite state machine with 9 states:

State 1 - Accept 1 input per clock cycle and store in shift register

- 64 Clock Cycles

State 2 - Time decimation, re-order input data

- 1 Clock Cycle since order is known

States 3-8 are the 6 stages of computation, during each 32 butterflies are performed in parallel

State 9 - Output one per clock cycle, then return to state 1

Altera Performance vs. Input Size

FFT Size	Input Cycles	Cycles between last input and first output	Total Cycles
32	32	66	130 (32+66+32)
64	64	100	228 (64+100+64)
128	128	175	431 (128+175+128)
256	256	304	812 (256+300+256)
512	512	572	1596 (512+572+512)

Performance Comparison (64-point FFT)

	Altera	Manual Verilog (verison 2)	LegUp, fft.c (inline on)	LegUp, fft.c (inline off)
LE LUTs Dedicated Logic Registers	4,449 3,601 3,836	27,785 25,566 15,460	2,421 2,207 977	2,372 2,136 1,072
Total Clock Cycles (first input to last output)	228	210 (=64+1+1+ 20*4+64)	3800 (including return sum)	10,010 (including return sum)
Fmax Critical path delay	198	138	88 (inline on)	100 (inline off)
Memory Bits	9,948	26,112	6,128	6,128
9-bit Multiplier Elements	24	256 (EP2C70F672C6)	10	4

For all performance results physical synthesis was on, and the default minimum clock frequency was constrained to 1000MHz (and not met).