Precis

Poster

Agenda

An Efficient Architecture for Ultra Long FFTs in FPGAs and ASICs

Tom Dillon
Dillon Engineering, Inc.

This presentation outlines an architecture for efficient Ultra Long FFTs for use in FPGAs and ASICs. Analysis of accuracy, performance, cost and power consumption are presented.

FFTs are at the heart of many real time signal processing applications and Ultra Long FFTs are quite often used for frequency analysis and communications applications. As the processing requirements increase, the use of FPGAs and ASICs become the logical choice for implementing real time FFTs.

This presentation describes and efficient framework for implementing the Cooley-Tukey algorithm for Ultra Long FFTs using minimal external memory. Typically for lengths over 16K the memory resources of the FPGA or ASIC are exhausted and external memory is required. The architecture is implemented using two shorter length FFTs (lengths N_1 and N_2) to calculate an FFT of length $N = N_1 \times N_2$. This architecture is optimized for continuous data FFTs, minimizing the external memory requirements and offering flexibility so that it can be used for many different applications.

The $(N_1 \times N_2)$ -point FFT can be computed as

$$X[k_1 N_2 + k_2] = \sum_{n_1=0}^{N_1-1} [e^{-j\frac{2\pi n_1 k_2}{N}} (\sum_{n_2=0}^{N_2-1} x[n_2 N_1 + n_1] e^{-j\frac{2\pi n_2 k_2}{N_2}})] e^{-j\frac{2\pi n_1 k_1}{N_1}} .$$

Computing this for $0 \leqslant k_1 \leqslant N_1 - 1$ and $0 \leqslant k_2 \leqslant N_2 - 1$ results in $X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi nk}{N}}$ for $0 \leqslant k \leqslant N-1,$ as desired. This leads to the following high-level architecture:

The input data is re-ordered by performing the equivalent of a matrix transposition. The next step is to compute N_1 FFTs, each of length $N_2,$ followed by the second matrix transposition. The re-ordered data set is multiplied by the twiddle factors, and N_2 FFTs, each of length $N_1,$ are computed. The final step is to perform the third matrix transposition so the output data is in the correct order.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **01 FEB 2005**	2. REPORT TYPE **N/A**	3. DATES COVERED **-**

4. TITLE AND SUBTITLE
An Efficient Architecture for Ultra Long FFTs in FPGAs and ASICs

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Dillon Engineering, Inc.

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR’S ACRONYM(S)

11. SPONSOR/MONITOR’S REPORT NUMBER(S)

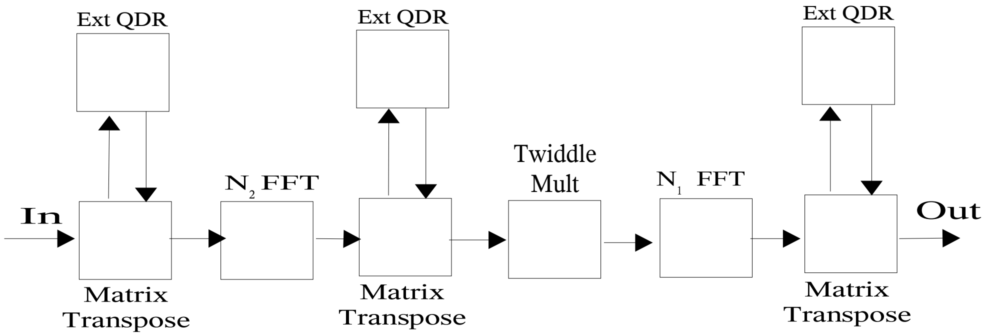
12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM00001742, HPEC-7 Volume 1, Proceedings of the Eighth Annual High Performance Embedded Computing (HPEC) Workshops, 28-30 September 2004 Volume 1., The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**UU**	**17**	



The external memory requirements can be reduced by using QDR synchronous SRAM and an addressing sequence to allow a single bank of memory to be used for each matrix transposition. This presentation describes details of this addressing sequence. SRAM is a requirement because data needs to be read and written on every clock cycle and the addresses are usually not consecutive. QDR allows writing and reading different locations simultaneously, thereby removing the requirement for two banks of memory at each matrix transposition.

The potential data growth in longer length FFTs makes numerical analysis a necessity. A finite word length analysis will be presented for both fixed and floating point FFTs which will show that either floating point or fairly wide fixed point FFTs are required to maintain the precision required for most applications. These wider word lengths affect the memory architecture because wider word lengths require more memory bandwidth for the matrix transpositions. The trade-offs between word length requirements and memory architecture are discussed in the presentation.

This architecture can also function as 2D FFT by simply bypassing the twiddle multiply and removing the first Matrix Transpose.

A variable length FFT engine can be built form the same architecture by using variable length N_1 and N_2 FFTs and modifying the Matrix Transpose blocks. Often a run time length selection is desired so that the resolution can be adjusted.

Accuracy, component cost, and power consumption data will be presented for a system implemented in a single FPGA and three QDR SRAM ICs computing 512K FFTs on continuous data at 200MSPS.

An Efficient Architecture for Ultra Long FFTs in FPGAs and ASICs

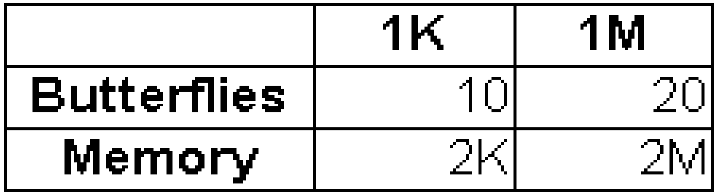
- Architecture optimized for Fast Ultra Long FFTs
- Parallel FFT structure reduces external memory bandwidth requirements
- Lengths from 32K to 256M
- Optimized for continuous data FFTs
- Architecture reduces the algorithm to two smaller manageable FFT engines

Ultra Long FFTs

- An FFT length that exceeds the internal memory requirements of the FPGA or ASIC
- **System cost can be reduced in moderate length FFTs in designs where the FPGA/ASIC size is driven by the memory requirements.**
- **This architecture puts most of the storage for the FFT off chip in relatively inexpensive SRAM, reducing the system cost.**
- **Ultra Long FFTs have a similar structure to 2D FFTs**
- **Cooley-Tukey algorithm**
- **Minimizes external memory IC count and bandwidth**

What Ultra Long FFTs Need

The following shows the execution unit (logic) and memory requirement for continuous data FFTs of two lengths:



	1K	1M
Butterflies	10	20
Memory	2K	2M

- **The logic requirements for a 1M FFT are only double a 1K FFT, while the memory requirements are 1000 times.**
- **Logic for 1M FFT easily fits into large FPGA**
- **Memory requirements exceed what is available even in a large FPGA**

Computing N = N_1 x N_2

The N_1 x N_2 FFT can be computed as:

$$X[k_1 N_2 + k_2] = \sum_{n_1=0}^{N_1-1} [e^{-j\frac{2\pi n_1 k_2}{N}} (\sum_{n_2=0}^{N_2-1} x[n_2 N_1 + n_1] e^{-j\frac{2\pi n_2 k_2}{N_2}})] e^{-j\frac{2\pi n_1 k_1}{N_1}}$$

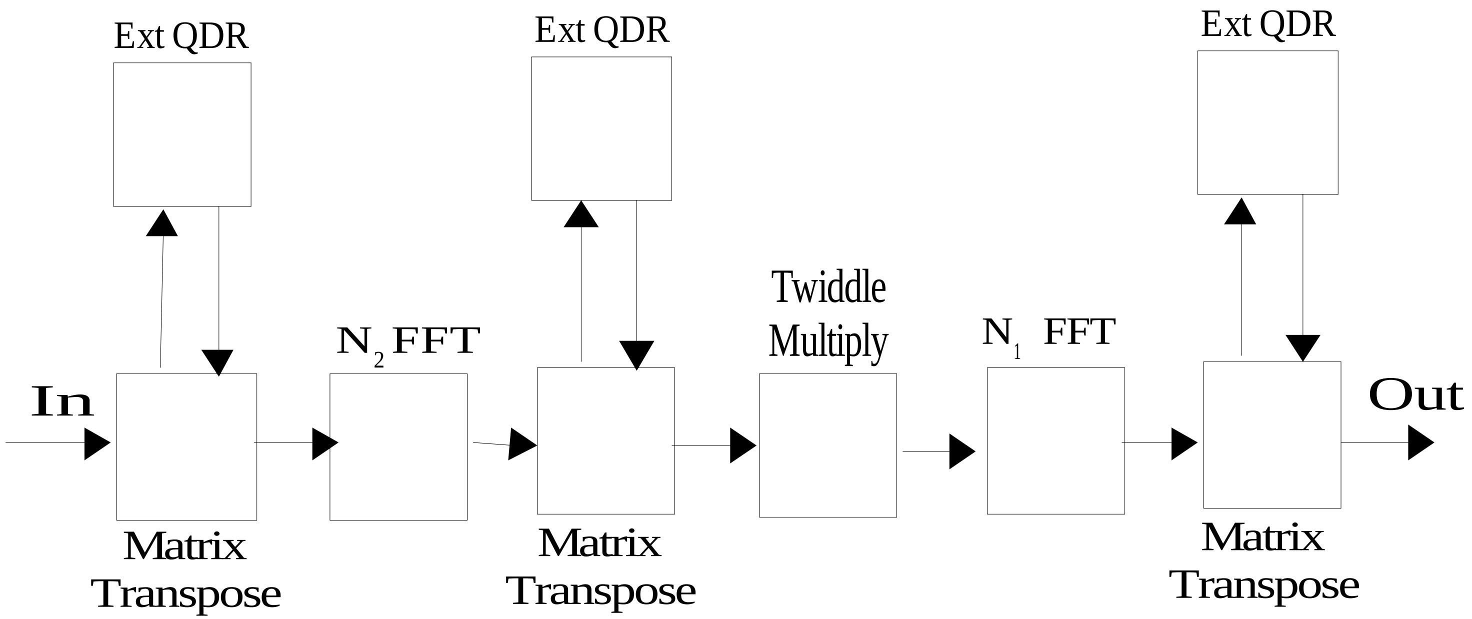
Computing this for:

$0 \leqslant k_1 \leqslant N_1 - 1$ and $0 \leqslant k_2 \leqslant N_2 - 1$

Results in:

$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi nk}{N}}$ for $0 \leqslant k \leqslant N-1,$ as desired

$N = N_1 \times N_2$ Architecture



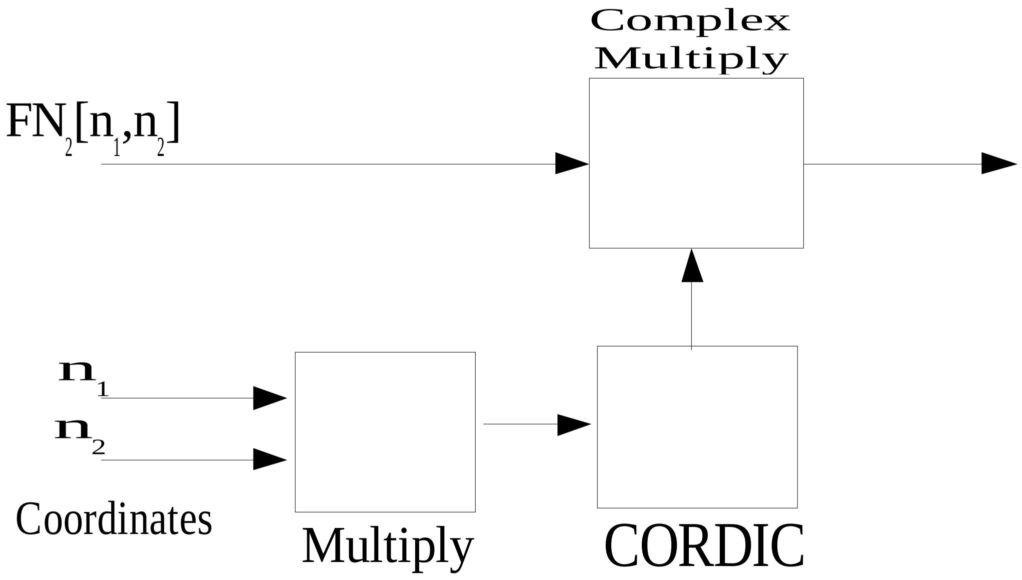
- Three banks of external QDR Memory (single copy each)
- Two continuous data FFTs (N_1, N_2) inside FPGA
- Twiddle Multiply provides vector rotation between N_2 and N_1 FFTs.
- Final matrix transpose for normal order output.

QDR SRAM

- Simultaneous read/writes (separate address/data bus) allow single bank of memory per memory transpose.
- DDR Style I/O so dual clock edge transfer with FPGA results in narrower data path.
- Single copy can be kept at each stage while maintaining continuous data flow.
- Special address sequence employed so data isn't overwritten in continuous data application. Reduce IC count.
- QDR with Virtex II Pro I/O up to 150MHz (read/write)
- QDR II with Virtex II Pro I/O up to 200MHz (read/write)

CORDIC For Twiddle Factors Generation

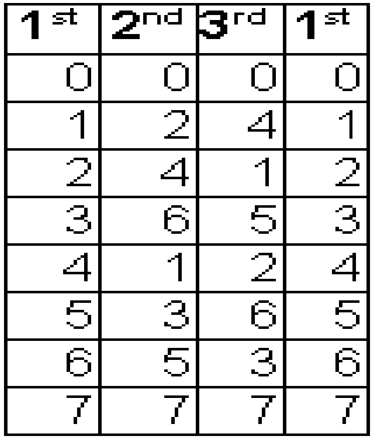
- **Almost N/2 twiddle factors required.**
- **Very large ROM for FPGA or ASIC.**
- **CORDIC a natural fit, use coordinate product as input.**



- **CORDIC produces the sin/cos terms for angle input.**

Matrix Transpose Address Sequence

- Allows single copy for each matrix transpose.
- Operates on continuous data, one point read/written per clock cycle.
- Reduces memory IC count.
- Simple logic for sequence generation.
- Works for square or rectangular matrices.
- Sequence repeats after $\log_2(N)$ sets.
- Write always follows read.
- Simple $N = N_1 \times N_2 = 8$ example:



1st	2nd	3rd	1st
0	0	0	0
1	2	4	1
2	4	1	2
3	6	5	3
4	1	2	4
5	3	6	5
6	5	3	6
7	7	7	7

- First and last matrix transpose go left to right in table, second right to left.

Fixed vs. Floating Point

- Numbers in radix-2 FFT can grow by $\log_2(N)$, or 1 bit per butterfly rank.
- A 1M FFT can have 20 bits of growth. With 16 bit inputs results would be 36 bits.
- Scaling always required in fixed point versions.
- Fixed point scaling should be limited to every to every other rank, so 10 times for a 1M FFT producing 26 bit results from 16 bit input.
- Floating point FFT maintains precision without overflowing.
- Floating Point FFT uses approximately 8 times the logic of a similar precision fixed point version.

Virtex II Pro Performance – 512K FFT

- 80MHz Continuous Data
- 1K FFT Engine – 4 butterflies
- 512 FFT Engine – 4 butterflies
- FFT Engines at 160MHz
- QDR memory at 80MHz
- Real 14 bit input, complex 24 bit output
- Virtex II Pro – Device Usage
  - Slices - 12,500
  - BlockRAM - 144
  - MULT18x18 – 88
- Fits in XC2VP40

Other Uses of Architecture

- 2D FFT – Remove first matrix transpose and twiddle multiply.
- Variable Length – Use variable length FFTs and dynamic matrix transpose blocks.
- Mixed Radix FFTs – Substitute other than radix-2 for 2nd FFT.
- Performance increases easy with parallel input radix-2 FFTs and multiple paths to SRAM.

Other Dillon Engineering Resources

- ParaCore Architect (parameterized core builder)
- DSP Algorithms
  - Mixed radix FFTs
  - 2D FFTs for image processing
  - Fixed or floating-point FFTs
  - Floating point math library
  - AES Cryptography
- System level DSP
  - OFDM Transceivers
  - Radar Processing on single FPGA
  - Image Compression/Processing
- Hardware/Software SOC
  - High speed Ethernet Appliances
  - Linux Based SOC in FPGA
  - MicroBlaze application

An Efficient Architecture for Ultra Long FFTs in FPGAs and ASICs

- Architecture optimized for Fast Ultra Long FFTs
- Parallel FFT structure reduces external memory bandwidth requirements
- Lengths from 32K to 256M
- Optimized for continuous data FFTs
- Architecture reduces the algorithm to two smaller manageable FFT engines
- Key Features
  - Uses 2 short manageable FFT engines ($N = N_1 \times N_2$)
  - QDR SRAM, reduce IC count, simultaneous read/write
  - CORDIC to generate rotation twiddle factors
  - Matrix transpose address sequence
  - Structure similar to 2D FFT or mixed radix FFT

Computing N = N$_1$ x N$_2$

The N$_1$ x N$_2$ FFT can be computed as:

$$X[k_1 N_2 + k_2] = \sum_{n_1=0}^{N_1-1} [e^{-j\frac{2\pi n_1 k_2}{N}} (\sum_{n_2=0}^{N_2-1} x[n_2 N_1 + n_1] e^{-j\frac{2\pi n_2 k_2}{N_2}})] e^{-j\frac{2\pi n_1 k_1}{N_1}}$$

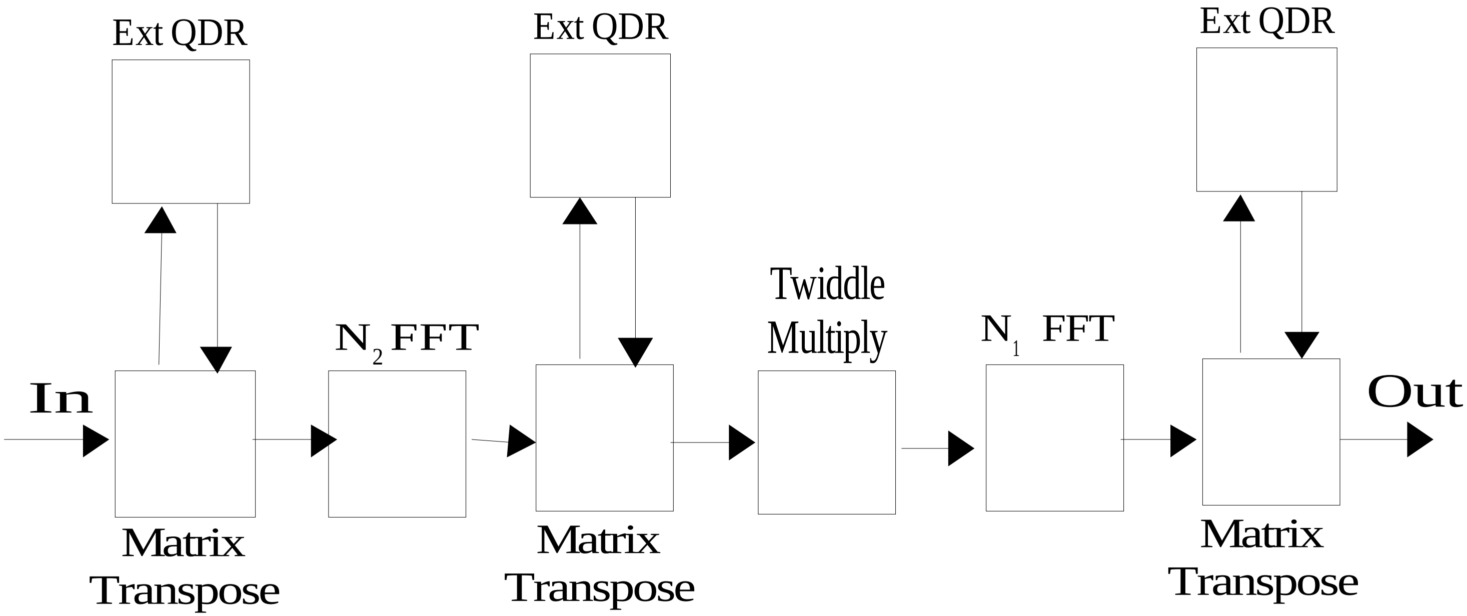
Computing this for:

$0 \leq k_1 \leq N_1 - 1$ and $0 \leq k_2 \leq N_2 - 1$

Results in:

$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi nk}{N}}$ for $0 \leq k \leq N-1,$ as desired

N = N_1 x N_2 Architecture



- Three banks of external QDR Memory (single copy each)
- Two continuous data FFTs (N_1, N_2) inside FPGA
- Twiddle Multiply provides vector rotation between N_2 and N_1 FFTs.
- Final matrix transpose for normal order output.