

# Very Large Fast DFT (VL FFT) Implementation on KeyStone

Multicore Applications

# Outlines

- Basic Algorithm for parallelizing DFT
- Multi-core implementation of DFT
- Review Benchmark Performance

# Goals and Requirements

- **Goal:**
  - To implement very large floating point fast DFT on TI multicore devices: Shannon and Nyquist
- **Requirements:**
  - FFT sizes: 4K – 1M samples
  - Configurable to run on different number of cores: 1, 2, 4, 8
  - High performance

# Algorithm for Very Large DFT

- A generic discrete Fourier transform (DFT) is shown below,

$$y(k) = \sum_{n=0}^{N-1} x(n) e^{j \frac{2\pi}{N} k n} \quad k = 0, \dots, N-1$$

- Here  $N$  is the total size of DFT ,

# Algorithm for Very Large DFT

- For very large  $N$ , it can be factored into  $N = N1 * N2$  and with decimation-in-time, the DFT can be formulated as,

$$y(k) = \sum_{n1=0}^{N1-1} \sum_{n2=0}^{N2-1} x(n1 * N2 + n2) e^{j \frac{2\pi}{N1 * N2} k * (n1 * N2 + n2)}$$

# Algorithm for Very Large DFT

- The above DFT formula can be further simplified as:

$$y(k) = \sum_{n_2=0}^{N_2-1} \left( \sum_{n_1=0}^{N_1-1} x(n_1 * N_2 + n_2) e^{j \frac{2\pi}{N_1} k * n_1} \right) e^{j \frac{2\pi}{N_1 * N_2} k * n_2}$$

$$= \sum_{n_2=0}^{N_2-1} \left( DFT_{N_1}(k_1, n_2) e^{j \frac{2\pi}{N_1 * N_2} k_1 * n_2} \right) e^{j \frac{2\pi}{N_2} k_2 * n_2}$$

$$k_1 = 0, \dots, N_1 - 1,$$

$$k_2 = 0, \dots, N_2 - 1$$

# Algorithm for Very Large DFT

- Here  $DFT_{N1}(k1, n2)$  is defined as:

$$DFT_{N1}(n2, k1) = \sum_{n1=0}^{N1-1} x(n1 * N2 + n2) e^{j \frac{2\pi}{N1} k1 * n1}$$

*for every  $n2 \in [0, \dots, N2 - 1]$*

# Algorithm for Very Large DFT

- **A vary large DFT of size  $N=N1*N2$  can be computed in the following steps:**
  - 1) Formulate input into  $N1 \times N2$  matrix
  - 2) Matrix transpose:  $N1 \times N2 \rightarrow N2 \times N1$
  - 3) Compute  $N2$  FFTs and multiply twiddle factors. Each FFT is  $N1$  size.
  - 4) Matrix transpose:  $N2 \times N1 \rightarrow N1 \times N2$
  - 5) Compute  $N1$  FFTs. Each is  $N2$  size.
  - 6) Matrix transpose:  $N1 \times N2 \rightarrow N2 \times N1$



# Implementing VLFFT on Multiple Cores

- Two iterations of computations
- 1<sup>st</sup> iteration
  - N2 FFTs are distributed across all the cores.
  - Each core implements matrix transpose and computes **N2/numCores FFTs and multiplying twiddle factor.**
- 2<sup>nd</sup> iteration
  - N1 FFTs of N2 size are distributed across all the cores
  - Each core computes **N1/numCores FFTs and implements matrix transpose before and after FFT computation.**

# Data Buffers

- **DDR3: Three float complex arrays of size N**
  - Input buffer, output buffer, working buffer
- **L2 SRAM:**
  - Two ping-pong buffers, each buffer is the size of 16 FFT input/output
  - Some working buffer
  - Buffers for twiddle factors
    - Twiddle factors for N1 and N2 FFT
    - N2 global twiddle factors

# Global Twiddle Factors

- Global Twiddle Factors:

$$e^{j\frac{2\pi}{N1*N2}k1*n2} \quad n2 \in [0, \dots, N2-1] \quad k1 \in [0, \dots, N1-1]$$

- Total of  $N1*N2$  global twiddle factors are required.
- $N1$  are actually pre-computed and saved.

$$e^{j\frac{2\pi}{N1*N2}n2} \quad n2 \in [0, \dots, N2-1]$$

- The rest are computed during run time.

# DMA Scheme

- Each core has dedicated in/out DMA channels
- Each core configures and triggers its own DMA channels for input/output
- On each core, the processing is divided into blocks of 8 FFT each.
- For each block on every core
  - DMA transfer 8 lines of FFT input
  - DSP computes FFT/transpose
  - DMA transfers 8 lines of FFT output

# VLFFT Pseudo Code

## **VLFFT\_start:**

**1) Core0 sends message to each core to start 1st iteration processing.**

**2) Each core does the following,**

Wait message from core 0 to start,

numBlk = 0;

While( numBlk < totalBlk )

{

1) Trigger DMA to transfer (n+1)th blk from Input Buffer to L2 and to transfer (n-1)th blk output from L2 to Temp Buffer

2) Implement transpose, compute FFT, and multiply twiddle factors for nth blk

3) wait for DMA completion

4) numBlk++

}

Send a message to core 0

**3) Core0 waits for message from each core for completion of its own processing**

**4) After receiving all the messages from all the other cores, core0 sends message to each core to start 2nd iteration processing**

**5) Each core does the following,**

Wait message from core 0 to start,

numBlk = 0;

While( numBlk < totalBlk )

{

1) Trigger DMA to transfer (n+1)th blk from Temp Buffer to L2 and to transfer (n-1)th blk output from L2 to Output Buffer

2) Compute FFT and transpose for nth blk

3) wait for DMA completion

4) numBlk++

}

Send a message to core 0

**6) Core0 waits for message back from each core for completion its own processing**

## **VLFFT\_end:**

# Matrix Transpose

- The transpose is required for the following matrixes from each core:
  - $N1 \times 8 \rightarrow 8 \times N1$
  - $N2 \times 8 \rightarrow 8 \times N2$
  - $8 \times N2 \rightarrow N2 \times 8$
- DSP computes matrix transpose from L2 SRAM
  - DMA bring samples from DDR to L2 SRAM
  - DSP implements transpose for matrixes in L2 SRAM
  - 32K L1 Cache

# Major Kernels

- FFT: single precision floating point FFT from c66x DSPLIB
- Global twiddle factor compute and multiplication: 1 cycle per complex sample
- Transpose: 1 cycle per complex sample

# Major Software Tools

- DSP BIOS 6
- CSL for EDMA configuration
- IPC for inter-processor communication



# Conclusion

- DDR I/O bandwidth limits 8-core VLFFT performance on KeyStone
- Need better design for matrix transpose
- Different parallelization of DFT may produce better performance. Potentially reduce 1 transpose.