Very Large FFT Multicore DSP Implementation Demonstration Guide

Overview

This demo software implements single precision floating point very large size FFT on Texas Instruments' latest multicore DSPs including C6678 and C6670. The software requires input data to be placed in the device's external memory. It distributes input data onto different DSP cores. Different DSP cores carry out the actual computations and place the output on the external memory. The software can be configured to use different number of cores to do the actual computation and can computer the FFT of the following sizes

- 16K
- 32K
- 64K
- 128K
- 256K
- 512K
- 1024K

The software can be run on the following EVM and simulators,

- C6678 EVM
- C6678 Functional Simulator
- C6670 EVM
- C6670 Functional Simulator

Requirements

The software requires Texas Instruments latest multicore SDK 2.0 (MCSDK 2.0). Particularly it requires the following software components from MCSDK 2.0.

- CCS 5
- DSP/BIOS 6.0
- IPC
- EDMA LLD

Software Design

The very large FFT implementation for multicore DSP is designed to achieve maximum performance by distributing the computation task onto multicores and by fully utilizing high performance computational power of DSP.

The basic decimation-in-time approach is used to formulate computing 1-D very large FFT into computing something similar to 2-D FFT computation. For very large N, it can be factored into N = N1*N2. If its very large 1-D input array is considered as a 2-D N1xN2 array (N1 rows, N2 columns), then the following steps can be taken to compute 1-D very large FFT.

1. Compute N2 FFTs of N1 size in column directions
2. Multiply twiddle factor
3. Store N2 FFTs of N1 size in row directions to form a N2xN1 2-D array
4. Compute N1 FFTs of N2 size in column direction
5. Store data in column direction to form a N2xN1 2-D array

The following paper describes the similar algorithm for multicore FFT implementation.

```
 "High-performance Parallel FFT algorithms for the Hitachi SR8000",
Daisuke Takahashi,
 Proceedings of The Fourth International Conference/Exhibition on High
Performance Computing in the Asia-Pacific Region, 2000, Issue Date:
14-17 May 2000, On page(s): 192 - 199 vol.1
```

In the actual computation, **N2/NUM_Of_CORES_FOR_COMPUTE** FFTs of size N1 in step 1. and **N1/NUM_of_CORES_FOR_COMPUTE** FFTs of size N2 in step 4. are computed on each core. Core0 is used as master core and the rest of the cores are used as slave cores. IPC software is used for inter processor communications. In addition to the FFT computations listed above, the core0 (master core) is also responsible for synchronizing all the cores.

The sequence of the main processings for the software thread on the master core (core0) and all the slave cores for computing an entire large size FFT is summarized as follows,

Software thread on core0

- FFT computation starts
- Core0 sends a command to all the slave cores informing each core to be in IDLE state
- Core0 waits for all the slave cores in IDLE state
- Core0 sends a command to all the slave cores informing each to start 1st iteration of processing
- Core0 starts its 1st iteration of processing
    1. Core0 fetches **N2/NUM_Of_CORES_FOR_COMPUTE** columns of its assigned data into L2 SRAM
    2. core0 computes **N2/NUM_Of_CORES_FOR_COMPUTE** FFTs of N1 size
    3. Multiply twiddle factors of each output
    4. Core0 stores **N2/NUM_Of_CORES_FOR_COMPUTE** FFTs of N1 size in row direction into an N2xN1 array in external memory (DDR3)
- Core0 waits for all the cores to complete their 1st iteration processing
- Core0 sends a command to all the slave cores to start 2nd iteration of processing
- Core0 starts its 2nd iteration of processing
    1. Core0 fetches **N1/NUM_Of_CORES_FOR_COMPUTE** columns of its assigned data into L2 SRAM
    2. core0 computes **N1/NUM_Of_CORES_FOR_COMPUTE** FFTs of N2 size
    3. Core0 stores **N1/NUM_Of_CORES_FOR_COMPUTE** FFTs of N1 size in row direction into an N2xN1 array in external memory (DDR3)
- Core0 waits for all the cores to complete their 2nd iteration processing
- FFT computation ends

Software thread on slave cores

- Each slave core waits for the command from master core (core0)
- Each slave core starts 1st iteration processing when receiving command from Core0 for starting 1st iteration processing
    1. Each slave core fetches **N2/NUM_Of_CORES_FOR_COMPUTE** columns of its assigned data into L2 SRAM
    2. Each core compute **N2/NUM_Of_CORES_FOR_COMPUTE** FFTs of N1 size
    3. Multiply twiddle factors of each output
    4. Each slave core stores **N2/NUM_Of_CORES_FOR_COMPUTE** FFTs of N1 size in row direction into an N2xN1 array in external memory (DDR3)
- Each slave core sends a message to core0 informing the completion of 1st iteration processing

- Each slave core waits for the command from master core (core0)
- Each slave core starts 2nd iteration processing when receiving command from Core0 for starting 2nd iteration processing
    1. Each slave core fetches **N1/NUM_Of_CORES_FOR_COMPUTE** columns of its assigned data into L2 SRAM
    2. Each slave core computes **N1/NUM_Of_CORES_FOR_COMPUTE** FFTs of N2 size
    3. Each slave core stores **N1/NUM_Of_CORES_FOR_COMPUTE** FFTs of N2 size in column direction into an N1xN2 array in external memory (DDR3)
- Each slave core sends a message to core0 informing the completion of 2nd iteration processing

On each core, depending on the sizes of N1 and N2, the total number of FFTs on each core, **N1/NUM_Of_CORES_FOR_COMPUTE** and/or **N2/NUM_Of_CORES_FOR_COMPUTE**, are divided into several smaller blocks in order to accomendate the limited available internal memory (L2 SRAM) on the device and each block size is 8 FFT.

In the actulal implementation, each block of data are prefetched by DMA from external memory into L2 SRAM and the FFT results are writen back to external memory by DDR. 16 DMA channels in total of EDMA instant0 are used for fethcing data. Two DMA channels are used by each core to transfer input and output samples between external memory (DDR3) and internal memory (L2 SRAM).

The following lists the memory utilization of the software for computing size N=N1*N2 FFT

External Memory (DDR3)

- input buffer: 1 complex single precision floating point arrays of size N
- Output buffer: 1 complex single precision floating point arrays of size N
- Temorary buffer: 1 complex single precision floating point arrays of size N

L2 SRAM

- 2 complex single precision floating point arrays of 16K size each
- 1 complex single precision floating point arrays of 8K size each
- 2 complex single precision floating point arrays of 1K size each
- 2 complex single precision floating point arrays of N2 size each (twiddle factors)
- 1 complex single precision of floating point array of size N1 (twiddle factors)

Build Instructions

The very large FFT demo software comes with pre-created project for C6678 or C6670 EVM. The following lists the steps to compile and build the project,

- Define a Windows System Environment variable, **TI_MCSDK_INSTALL_DIR**, and points the variable to the directory where TI MCSDK 2.0 locates.
- Import the project, vlfft_evmc6678l or vlfft_evmc6670l, into ccs5. The projects locates under the directory:

```
\demo\vlfft\
```

- To compile for C6678 EVM, open the file **vlfftconfig.h** under **\demo\vlfft\vlfftInc** and set the constant **EIGHT_CORE_DEVICE** to 1 and **FOUR_CORE_DEVICE** to 0.
- To compile for C6670 EVM, open the file **vlfftconfig.h** under **\demo\vlfft\vlfftInc** and set the constant **EIGHT_CORE_DEVICE** to 0 and **FOUR_CORE_DEVICE** to 1.
- To configure the size of the FFT, open the file **vlfftconfig.h** under **\demo\vlfft\vlfftInc** and set one of the following constant definitions to 1 and the rest to zero

```
VLFFT_16K
VLFFT_32K
VLFFT_64K
VLFFT_128K
VLFFT_256K
VLFFT_512K
VLFFT_1024K
```

- To configure the number of DSP cores to compute, open the file **vlfftconfig.h** under **...\demo\vlfft\vlfftInc** and change the constant definition **NUM_CORES_FOR_FFT_COMPUTE** to one of the following numbers
  - 4-core device: 1, 2, 4
  - 8-core device: 1, 2, 4, 8
- Set either Debug or Release active in ccs5
  - For Debug mode: the following 4 lines from line 92 - line 95 in file **vlfft_evmc6678l.cfg** under **..\demos\vlfft\evmc6678l** or **vlfft_evmc6670l.cfg** under **..\demos\vlfft\evmc6670l** should be disabled or commented out.

    var MessageQ = xdc.module('ti.sdo.ipc.MessageQ');

    var Notify = xdc.module('ti.sdo.ipc.Notify');

    Notify.SetupProxy = xdc.module('ti.sdo.ipc.family.c647x.NotifyCircSetup');

    MessageQ.SetupTransportProxy = xdc.module('ti.sdo.ipc.transports.TransportShmNotifySetup');
  - For Release mode: the following 4 lines from line 92 - line 95 in file **vlfft_evmc6678l.cfg** under **..\demos\vlfft\evmc6678l** or **vlfft_evmc6670l.cfg** under **..\demos\vlfft\evmc6670l** should be enabled.

    var MessageQ = xdc.module('ti.sdo.ipc.MessageQ');

    var Notify = xdc.module('ti.sdo.ipc.Notify');

    Notify.SetupProxy = xdc.module('ti.sdo.ipc.family.c647x.NotifyCircSetup');

    MessageQ.SetupTransportProxy = xdc.module('ti.sdo.ipc.transports.TransportShmNotifySetup');
- Build the project using the Build Project option under Build in ccs5

Run Instructions

- To run the code on C6678 functional simulator, load **vlfft_evmc6678l.out** from either **\vlfft\evmc6678l\\Debug** or **\vlfft\evmc6678l\Release** directory onto all the cores on the device. This is true regardless the number of cores is configured to compute the FFT. Run the code on all the cores.
- To run the code on C6678 EVM, initialize the PLL and DDR3 of the EVM using right GEL files. load **vlfft_evmc6678l.out** from either **\vlfft\evmc6678l\\Debug** or **\vlfft\evmc6678l\\Release** directory onto all the cores on the device. This is true regardless the number of cores is configured to compute the FFT. Run the code on all the cores.

Article Sources and Contributors

Very Large FFT Multicore DSP Implementation Demonstration Guide *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?oldid=103203 *Contributors*: A0214579