Application Report
SPRAC87–April 2017

Processor SDK RTOS Audio Benchmark Starter Kit

Rahul Prabhu

ABSTRACT

The TI TMS320C6000™ Digital Signal Processors (DSPs) have many architectural advantages that make them ideal for computation-intensive real-time applications that are commonly used in audio processing application. This application notes describes Audio Benchmark Starterkit software that is intended to provide an easy and quick way to benchmark key audio functions on C66x and C674x DSP devices using Processor SDK RTOS. This package is intended for users who are new to the TI DSP development environment and provides an easy path to compare core audio benchmarks to other implementations.

Contents

List of Figures

List of Tables

Trademarks

TMS320C6000, Code Composer Studio are trademarks of Texas Instruments.
All other trademarks are the property of their respective owners.

1 Introduction

The Audio Benchmark Starter Kit is intended to provide an easy and quick way to benchmark key audio functions on TMS320C66x and TMS320C674x DSP devices. This package is intended for users who are new to the TI DSP development environment and provides an easy path to compare core audio benchmarks to other implementations. For the purposes of benchmarking, the following signal processing functions were selected:

- Complex fast fourier transform (FFT)
- Real block FIR filters with 128 samples, 16 coefficients
- Cascaded biquad (2 channels, 3 stages) IIR filter for 128 Samples

Software Features

- Benchmark applications for core signal processing functions
- Makefile and Code Composer Studio (CCS) project scripts to build applications
- SD card bootable binaries [Supported on SoCs that support SD boot]

1.1 Directory Structure

The Audio Benchmark Starter Kit is located in the Processor SDK RTOS release under the directory path:

```
<SDK_INSTALL_PATH>\processor_sdk_rtos_<soc>_x_xx_xx_xx\demos\audio-benchmark-kit
```

The directory structure for the Audio Benchmark Starter Kit is shown in Figure 1.

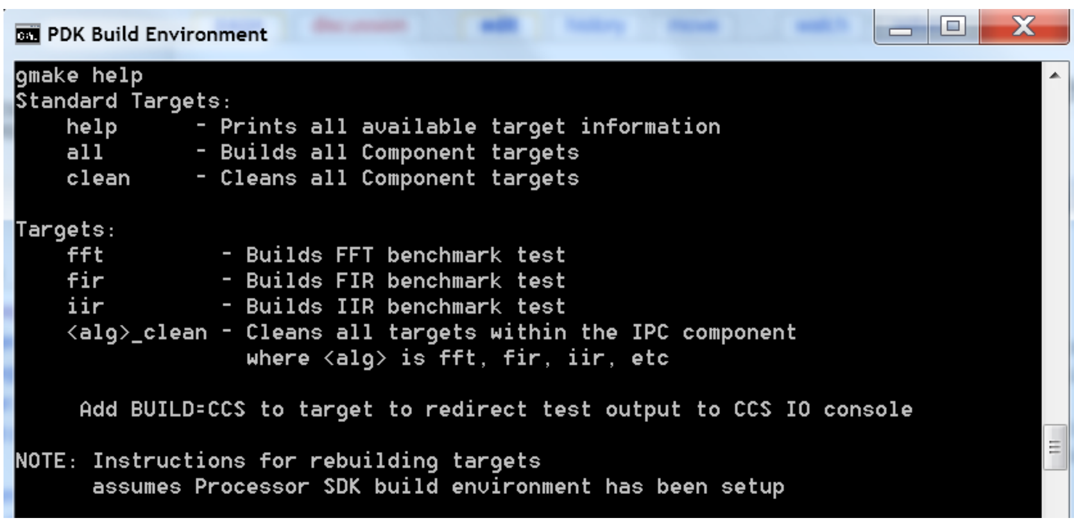


Figure 1. Benchmark Starter kit Directory Structure

Detailed description of the directory structure is provided below:

- pre-built-binaries - Directory contains pre-built out files to run the benchmarks
- bootimages: SD card boot files to run the benchmarks using SD boot
- docs - Directory contains ReadMe, Quick Start Guide and the software manifest for the package.
- scripts - Directory contains .txt script files that are used by BenchmarkProjectCreate script to create CCS projects.
- src
  - common - Contains linker command file and logging functions used by all benchmark tests.
  - singlePrecision_FFT - Source files for benchmark app for FFT
  - singlePrecision_FIR - Source files for benchmark app for FIR
  - singlePrecision_IIR - Source files for benchmark app for IIR

1.2 Software Dependencies

- Processor SDK RTOS v3.3 and later
- Code Composer Studio IDE Environment v7 and later

NOTE: To download the correct version of Code Composer Studio™ (CCS), see the Release Notes corresponding to the Processor SDK RTOS version that you have installed.

1.3 Supported Hardware

Platforms supported in the Processor SDK RTOS 3.3 and later:

- K2G Evaluation Module (EVM)
- AM572x GP EVM
- AM571x IDK

Platforms supported in the Processor SDK RTOS 4.0 and later:

- OMAPL138/C6748 LCDK
- K2H EVM rev 1.1 and later
- K2E EVMs rev 1.0.2.0 and later
- C6678 EVM
- C6657 EVM

1.4 Quick Start With How-To Video

For an easy experience to build and run the benchmark tests, a short "How To" video was created. This video demonstrates how the Benchmark starter kit can be built and run the C66x DSP on the K2G EVM, which can be checked out from the link provided below:

Demonstration of Audio Benchmark Starter Kit demo on 66AK2G02 GP EVM (https://training.ti.com/66ak2gx-gp-evm-audio-benchmark-starter-kit-demo)

2 How to Build the Benchmarks

The benchmark starter kit is designed to build with makefiles as well as with the CCS IDE environment, both of which require developers to setup the Processor SDK RTOS development environment. Either approach can be used based on familiarity with the chosen build environment.

2.1 Using Makefile

1. Setup the Processor SDK RTOS build environment.

   Developers are required to setup the Processor SDK RTOS build environment as described in the Processor SDK RTOS Setup Environment wiki page.

2. Invoke make from the root directory.
   The make file in the root directory of the audio-starter kit can be used to build the entire package. To build the benchmark examples:

   (a) cd <PROC_SDK_INSTALL_PATH>/demos/audio-benchmark-kit

   (b) make all

NOTE: The build picks up the system-on-chip (SoC) information from the SDK setup. Also, in the make environment, the benchmark application is built to send benchmark logs to the Universal Asynchronous Receiver/Transmitter (UART) console so there is no dependency on the CCS IDE environment.

For Other supported options, type:

For Windows:

```
gmake help
```

For Linux :

```
gmake help
```

All available options are provided below:

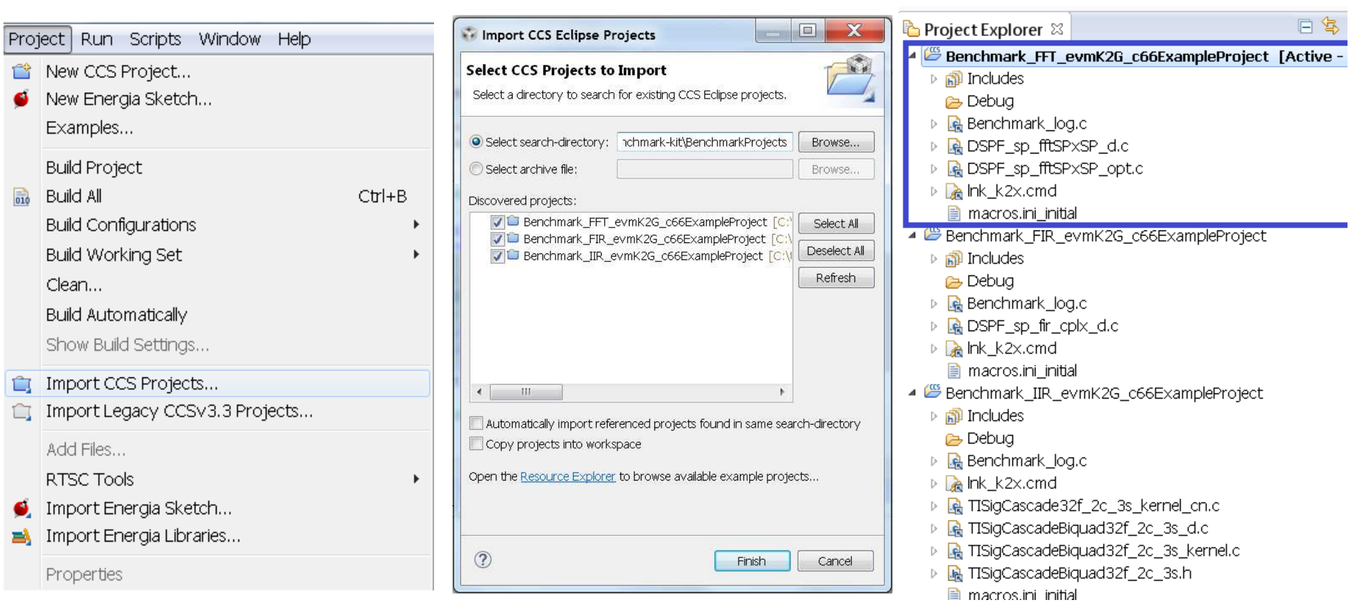


Figure 2. Using Makefile

2.2 Using CCS Projects

The Audio Benchmark Starter Kit does not provide pre-canned CCS projects because it is difficult to set up projects to be portable across various developer build environments. To create CCS projects with the benchmarks, developers are required to run the BenchmarkProjectCreate script provided in the root directory of the starter kit.

1. Setup Processor SDK RTOS build environment.

   Developers are required to setup the Processor SDK RTOS build environment as described in the Processor SDK RTOS Setup Environment wiki page.

   NOTE: If CCS or Processor SDK RTOS is installed under a Custom path, a reference should be included to the setup instructions, described under Setup Environment.

2.
3. Run BenchmarkProjectCreate script to generate CCS projects.
   To generate the CCS projects:

   (a) cd $PROC_SDK_INSTALL_PATH/demos/audio-benchmark-kit

   (b) BenchmarkProjectCreate [Options]

   The Project create script can be run using the following syntax:

   ```
   BenchmarkProjectCreate.bat <soc> <board> <all>
   ```

   Description of arguments:

   - SoC - K2G (Default) / K2H/ K2E/ C6678/ C6657/ AM572X/ AM571x/ OMAPL138
   - board - all (Default) / <SoC supported EVMs>
   - module - all / (FFT / FIR / IIR)

   ```
   Example:
    a) BenchmarkProjectCreate.bat
                 - Creates all module projects for the K2G soc for evmK2G platform
    b) BenchmarkProjectCreate.bat AM572x
                 - Creates all module projects for AM572x soc for evmAM572x and idkAM572x platform
    c) BenchmarkProjectCreate.bat C6657 evmC6657
                 - Creates all modules for C6657 DSP for evmC6657 platform
    d) BenchmarkProjectCreate.bat K2H evmK2H FFT
                 - Creates FFT module project for K2H soc for evmK2H
   ```

4. Import the generated CCS projects into the CCS workspace.

   Launch CCS and import the CCS project using Project → Import Existing CCS Project and browse to the audio-benchmark-kit folder .

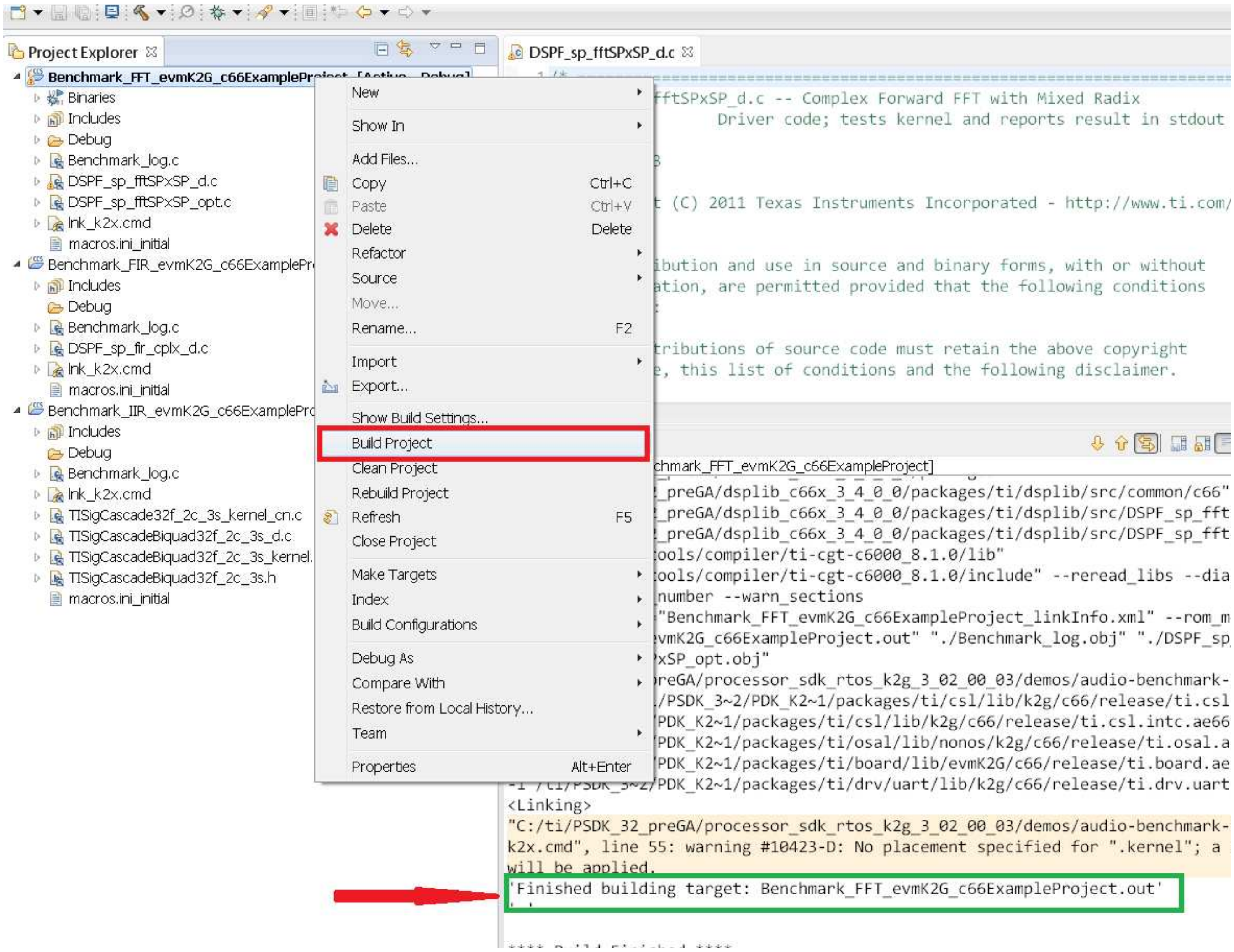
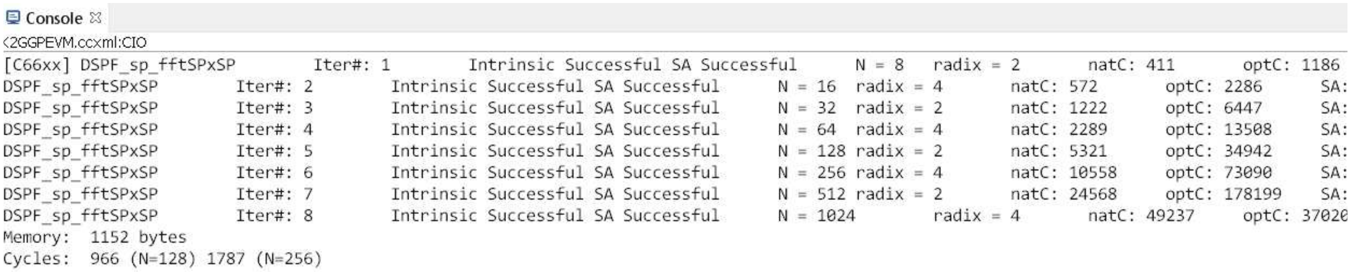


Figure 3. Using CCS Projects

5. Build Imported CCS Benchmark projects.
   Right click on the Benchmark Project File and Build Project as shown in Figure 4.



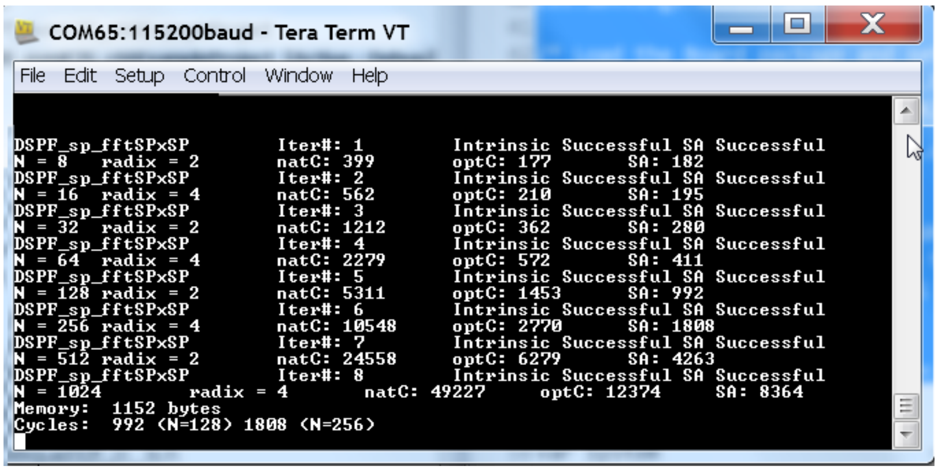


Figure 4. Benchmark Project File and Build Project

3 How to Run the Benchmarks

The benchmark examples can be run by loading the built out files with an emulator using the CCS Debug functionality or the examples can be run on the DSP by creating SD card bootable images using out files.

3.1 Using CCS

1. Connect the emulator and UART to the hardware.
   (a) For more information, see the Hardware Setup Guide wiki page and connect the onboard or external emulator to the hardware and Host machine with CCS installed.
   (b) Connect the UART cable from the EVM to the Host machine and configure the Serial console with the following settings:
      (i) Baud Rate: 115200
      (ii) Data Bits: 8
      (iii) Parity: None
      (iv) Flow Control: Off
2. Create Target configuration and connect to the DSP.
   To connect to the SoC, developers need to create a Target configuration by following the procedure described in the Create_Target_Configuration_File_for_EVM wiki page
   (a) K2G GP EVM CCS Setup wiki page
   (b) AM572x GP EVM CCS Setup wiki page

NOTE: For more information, see the device-specific Hardware User Guide; setup the boot switches to No boot if available.

---

3. Load and run the Benchmark applications on the DSP.
   (a) Load the out file using Run → Load → Load Program and browse to the output binary.
   (b) After loading the out file, run the benchmark app by Pressing F8 or Run → Resume.

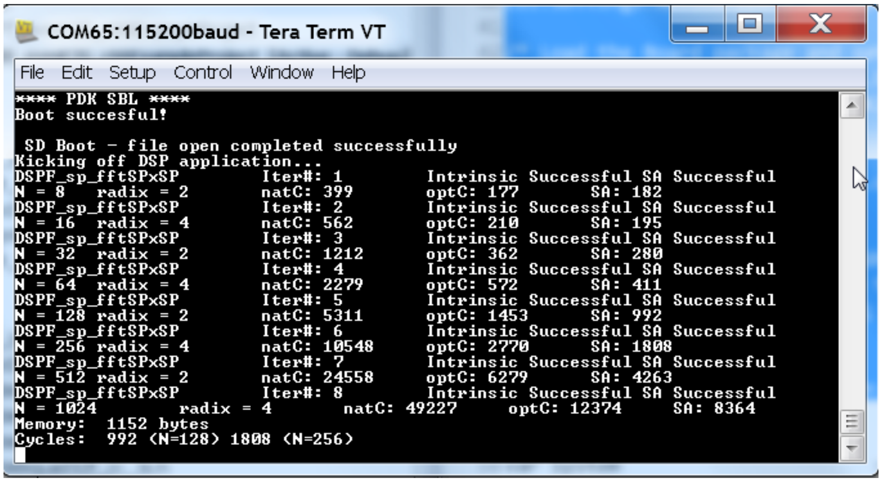


Figure 5. Missing Title

3.2 Using SD Card (supported only on AM57xx and K2G)

1. Run Create SD script to generate SD bootable binaries.

   Create an SD card using the procedure described in the Creating SD card in Windows and the Create SD card in Linux wiki pages.

   Copy the "MLO" and "Singleprecision_<Module>_app" to the boot partition on the SD card.
2. Boot the Benchmark app by configuring SD boot on the EVM.
   (a) Configure the boot switches on the evaluation hardware to SD boot.
   (b) Insert the SD card in the microSD or SD card slot on the board.
   (c) Connect the UART on the hardware to the Host and configure the host to Baud Rate= 115200, Data Bits= 8 , Parity= None, Flow Control= Off.
   (d) Power on the EVM to view the output on the Serial console on the host.

Figure 6. Benchmark App Output on UART Console

4 Benchmark Starter Kit Implementation

4.1 Signal Processing Functions Used in Starter Kit

The Audio Benchmark Starterkit uses the following three algorithms to provide a starting point to benchmark signal processing functions used in audio applications:

- Single Precision FFT
- Single Precision FIR Filtering
- Single Precision Multichannel IIR filtering

Theses algorithms are explained in details in the following sections.

4.1.1 Single Precision FFT: DSPF_sp_fftSPxSP (Mixed Radix Forward FFT)

The audio benchmark kit uses the FFT implementation (DSPF_sp_fftSPxSP) from the TI DSP Library. The DSPF_sp_fftSPxSP kernel calculates the discrete Fourier transform of complex input array `ptr_x` using a mixed radix FFT algorithm. The result is stored in complex output array `ptr_y` in normal order. Each complex array contains real and imaginary values at even and odd indices, respectively. DSPF_sp_fftSPxSP kernel is implemented in assembly to maximize performance, but a natural C implementation is also provided. The demonstration app for this kernel includes the required bit reversal coefficients, `brev`, and additional code to calculate the twiddle factor coefficients, `ptr_w`.

NOTE:

- For implementation details of this FFT computation, see the documentation provided in Section 7.
- For real input sequences, efficient FFT Implementation is described on the Efficient_FFT_Computation_of_Real_Input wiki page.

4.1.2 Single Precision FIR: DSPF_sp_fir_cplx (Complex FIR filter)

The audio benchmark kit uses the FFT implementation (DSPF_sp_fftSPxSP) from the TI DSP Library. The DSPF_sp_fir_cplx kernel performs complex FIR filtering on complex input array x with complex coefficient array h. The result is stored in complex output array y. For each complex array, real and imaginary elements are respectively stored at even and odd index locations.

The API reference and the implementation details can found in the TI DSPLIB documentation included in the Processor SDK.

4.1.3 Single Precision IIR : tisigCascadeBiquadSP_2c_3s_kernel (Cascade biquad filter for multichannel input)

The Cascade biquad filtering function in the Audio Benchmark Starter Kit is an improved biquad infinite impulse response filter Patent US20160112033 Pending. The new filter structure modifies the feedback path in the filter, resulting in a significant reduction in execution cycles. One of the most-used digital filter forms is the biquad. A biquad is a second order (two poles and two zeros) Infinite Impulse Response (IIR) filter. It is a high enough order to be useful on its own. And, because of the coefficient sensitivities in higher order filters, the biquad is often used as the basic building block for more complex filters. For instance, a biquad low-pass filter has a cutoff slope of 12 dB/octave, useful for tone controls. If a 24 dB/octave filter is needed, you can cascade two biquads that have less coefficient sensitivity problems than a single fourth-order design.

For implementation details, see the USTO link.

API Reference:

```
int tisigCascadeBiquad32f_2c_3skernel(CascadeBiquad_FilParam *pParam)
```

where CascadeBiquad_FilParam is defined as:

```
CascadeBiquad_FilParam {
     float *restrict pin1;    // Input Data Channel 1
     float *restrict pin2;    // Input Data Channel 2
     float  *restrict pOut1;  // Output Data Channel 1
     float  *restrict pOut2;  // Output Data Channel 1
     float  *restrict pCoef;  // Filter Coefficients a, b for 3 stages
     float  *restrict pVar0;  // Filter Variables d0, d1 for 3 stages channel 0
     float  *restrict pVar1;  // Filter Variables d0, d1 for 3 stages channel 1
     int      sampleCount;    // Number of samples
}  CascadeBiquad_FilParam;
```

4.2 Memory Placement of Instruction and Data

The best performance of the DSP can be obtained by placing all the data and instructions in L2 SRAM. For information on how the instructions and data can be placed in DSP internal L2 memory, see the linker command files include in the src/common folder.

NOTE: In application use cases where audio data needs to be placed in on-chip shared memory (OCMC or MSMC) and DDR memory, it is recommended to move data from external memory to L2 for processing using EDMA or enable DSP cache using CSL to optimize performance.

4.3 Compiler Optimization Flags

All the projects in the Audio Benchmark Starter Kit are built using the TMS320C6000™ compiler with -o3 optimization that allows the source code to be compiled with the highest compiler optimization settings. To modify the compiler options, see the compiler Build settings in the Makefiles or go to Build Settings in the CCS Project settings.

NOTE:

- For more Details on the recommended C6000 Compiler options, see C6000_Compiler:_Recommended_Compiler_Options wiki page.
- C6000 compiler documentation: TMS320C6000 Optimizing Compiler v8.1.x User's Guide

---

4.4 SoC Integration and Optimization

4.4.1 Configuring Device Clocks

Every SoC with TI DSP requires users to enable the DSP clocks by setting up the PLL and or enabling the DSP through Power Sleep Controller or Power and Control (PRCM) module. The way the clocks are set up differs depending on the environment setup:

- Development environment with emulator: In this case, the SoC clocks are setup using GEL files that are added to the target configuration file.
- Application boot from boot media: If you are booting the application from a boot media like Secure Data Memory Card/Dual-Core Processor MultiMedia Card (SD/MMC) or flash device, the ROM bootloader or a secondary level bootloader performs the clock configuration. For the audio starter kit, this initialization is done using the board library that is linked to the secondary bootloader and the benchmark tests.

NOTE: If the clocks are not configured, the DSP runs at speeds of the input clock rather than at the device speed grade. If the clocks are not configured correctly, the benchmarks runs much slower than anticipated, but the cycle count shows the same.

4.4.2 Benchmarking Using DSP TSCH/TSCL Registers

For C66x+ and C674x members of the C6000 family, there is a pair of registers that together provide a 64-bit clock value: TSCL and TSCH, You can create your own clock function to take advantage of these registers by adding this function to your program and it will override the clock function from the library.

The Bench mark test application, use the following functions to capture cycle count using the TSCH and TSCL registers:

```
/* ---------------------------------------------------------------- */
/* Initialize timer for clock */
TSCL= 0,TSCH=0;
/* Compute the overhead of calling _itoll(TSCH, TSCL) twice to get timing info */
/* ---------------------------------------------------------------- */
t_start = _itoll(TSCH, TSCL);
t_stop = _itoll(TSCH, TSCL);
t_overhead = t_stop - t_start;
```

```
 t_start = _itoll(TSCH, TSCL);
<Algorithm to be bechmarked>
t_stop = _itoll(TSCH, TSCL);
t_measured = (t_stop - t_start) - t_overhead;
```

4.4.3 Benchmark Logging

The audio benchmarks demonstrates two ways to log benchmark numbers. One approach that can be used when code is loaded and run from CCS is to use standard printf messages from the standard IO RTS libraries. The other approach is to use UART-based logging that can send the benchmark logs to serial console on the host at the baud rate of 115.2 kbps.

All of the benchmark test applications include a *Benchmark_log.h* and *Benchmark_log.c* file, which is used to log messages based on the definition of macro IO_CONSOLE. If IO_CONSOLE is defined, the output is directed to the CCS console. If it is not defined, the logs are sent to the UART console.

IO_CONSOLE is not defined in makefiles and scripts that build binaries to boot from the SD card. Therefore, the benchmark logs are directed to the UART serial console. In the CCS projects, the IO_CONSOLE macro is defined so that the output can be observed on the CCS console.

4.4.4 Cache Configuration for Code/Data Sections in SRAM/DDR

The best performance of the DSP can be obtained by placing all of the data and instructions in the L2 SRAM. If the developer application use cases places audio data in on-chip shared memory (OCMC or MSMC) and DDR memory, then it is necessary to enable L1 and L2 cache using CSL API.

To enable and utilize cache in the application, see the *csl_cacheAux.h* file in the *pdk_<soc>_x_x_x/packages/ti/csl* folder in the SDK, and link the CSL library for the SoC into the application code.

5 Benchmark Results

Table 1. Algorithm/DSP Architecture

Algorithm\DSP Architecture	C66x DSP	C674x DSP
Single Precision FFT (256 samples)	1808 cycles	2314 cycles
Single Precision FIR (128 samples, 16 coefficients)	2652 cycles	4465 cycles
Single Precision IIR (1k samples from 2 channel with 3 stage cascade biquad)	8258 cycles	12381 cycles

6 Support

For questions, feature requests and bug reports, use the TI E2E Forums provided below:

- Multicore DSP Forums
- Single core DSP Forums
- Sitara Forums

7 References

- *Introduction to TMS320C6000 DSP Optimization*
- *TI DSP Benchmarking*
- *Optimizing Loops on the C66x DSP*
- *TI's New C66x Fixed- and Floating-Point DSP Core Conquers the 'Need for Speed' White Paper*
- *Efficient Fixed- and Floating-Point Code Execution on the TMS320C674x Core Delivers Faster Code Development and Reduces System Cost With Improved Performance*
- TMS320C6000 Optimizing Compiler v8.1.x User's Guide

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265