C6Accel Reference guide

Introduction

Why do I need C6Accel?

If you are an application developer on TI's SoCs using the DSP on the SoC for codec operation, C6accel can enable additional functionalities on the DSP. The codec operation doesn't always consume all the MHz of the DSP. C6Accel provides generic signal processing and math functionalities that can be leveraged by the developer to enable best use of the DSP in the application. If you are an application developers on TI's SoCs and you are not making use of the DSP today, the C6Accel provides ready to use, optimized DSP kernels that can be invoked from your ARM application. By making use of the DSP to do what it is best at (signal processing and math operations) you can make most efficient use of your SoC resources.

What is C6Accel?

C6Accel is a DSP algorithm that follows TI's specified algorithm interface called xDAIS. TI also provides a framework called Codec Engine that allows efficient execution of algorithms written using the xDAIS interface. All the codecs that TI provides follow the xDAIS algorithm interface. TI provides these codecs as part of the SW offering for the SoC and recommends the usage of Codec Engine framework. Codec engine framework provides APIs that allow ARM side application code to invoke execution of codecs on the DSP. Many application developers using the DSP for codec operation limit the DSP usage to codecs only. The ever growing performance of the DSP core and availability of specialized H/W accelerators imply that the DSP MHz may not be completely utilized by the codec operation only. Thus an application developer can develop a more compelling application by making use of the remaining MHz on the DSP. The C6Accel algorithm includes many core functions that are useful in various signal processing applications. These functions include digital signal processing routines such as FFTs and filters. These also include image processing routines such as edge detection and color space conversion. These also include math routines such as Logarithm, sqrt and trigonometric functions.

What Devices Does C6Accel Support?

The C6Accel project targets two-core heterogeneous SoC processors from Texas Instruments, specifically targeting ARM+DSP devices, which run Linux on the ARM, in particular:

- DM6467 (ARM926 and C64x+ fixed-point DSP)
- OMAP3530 (ARM Cortex A8 and C64x+ fixed-point DSP)
- OMAP-L137 (ARM926 and C6740 fixed- and floating-point DSP)
- OMAP-L138 (ARM926 and C6740 fixed- and floating-point DSP)

Note:: Target specific information can be found in the Release Notes.

What value does C6Accel provide to TI SoC developers?

The C6Accel is designed to solve some common problems associated with developing system-on-a-chip (SoC) applications. The most significant problems include:

- ARM SOC users want to have production quality TI DSP Library of functions on ARM that can allow an application to offload tasks to the DSP.These compute intense kernels run more efficiently on the DSP and hence allows application utilize DSP MIPS and save on ARM MIPS. C6Accel provides 100+ of these functions within an interface that is easy to call into from the ARM.
- ARM SOC users who are using codecs that run on the DSP would like to use the same interface to perform pre/post processing compute intensive tasks on the DSP. C6Accel allows these users to do more with the codec engine than just the encode and decoding of data by continuing to leverage the DSP as a blackbox.
- Programmers with a GPP (general-purpose processor) view don't want to have to learn to be DSP programmers. They don't want to have to worry about a DSP's complex memory management and DSP real-time issues. C6Accel abstracts these complexities by providing abstracted wrapper API calls that hides these complexities from an ARM user.
- Inorder to minimize codec engine overhead during multiple function calls through codec engine, C6Accel provides chaining of kernel API calls.
- C6Accel uses standard xdm codec engine complaint iUniversal interface and hence will provide easy porting of application across platforms.
- For market success, most applications need to support multiple codecs to handle the same type of media. For example, an application might need to support three or four audio formats. C6Accel provides an interface where the codec engine and universal handle can be shared with other codecs.
- For custom codec developers who want to add their own functionality, C6Accel provides an easy entry point to xdais their algorithm and package it in an codec engine compliant format. By following the xdais template of C6Accel, SOC user can package their algorithm with a iUniversal interface to the codec engine.

The C6Accel addresses these problems by providing a standard software architecture and interfaces for DSP algorithm execution.

The C6Accel is:

- **Easy-to-use.** Soc Application developers specify what algorithm needs to be run on the DSP using simple APIs.
- **Portable.** The APIs and algorithm are SoC target, platform independent.
- **Extensible and configurable.** New algorithms can be added by anyone, using C6Accel xdais algorithm as a template and refering to C6Accel doumentationAdding Kernels to C6Accel [1].

Features

- C6Accel provides optimized canned DSP software in a codec engine compliant format.
- Easy application interface codec.
- Simple callable APIs to DSP functionalities
- Fully compatible with most TI C6x devices
- Capabilty to chain function calls using single API call to the codec engine
- DSP kernel Benchmarks (cycle and code size) on different platforms that aid in evaluating performance that can be leveraged from the DSP and make informed decisions while developing applications.
- Asynchronous and synchronous calling modes. Asynchronous calling mode enables parallel processing on DSP and ARM.
- Ready to use DSP software reduces learning curve and decreases time to market for ARM applications.

Where does C6Accel fit in the SOC environment

The application code (or the middleware it uses) calls C6Accel using wrapper library APIs and/or codec engine compliant iUniversal APIs which are similar to the VISA APIs used to invoke Audio/Video/Speech codecs. These APIs then pass through the codec engine interface and invoke the C6Accel algorithm on the DSP which identifies the function call and executes the appropriate functionality on the DSP using the parameters passed from the application.

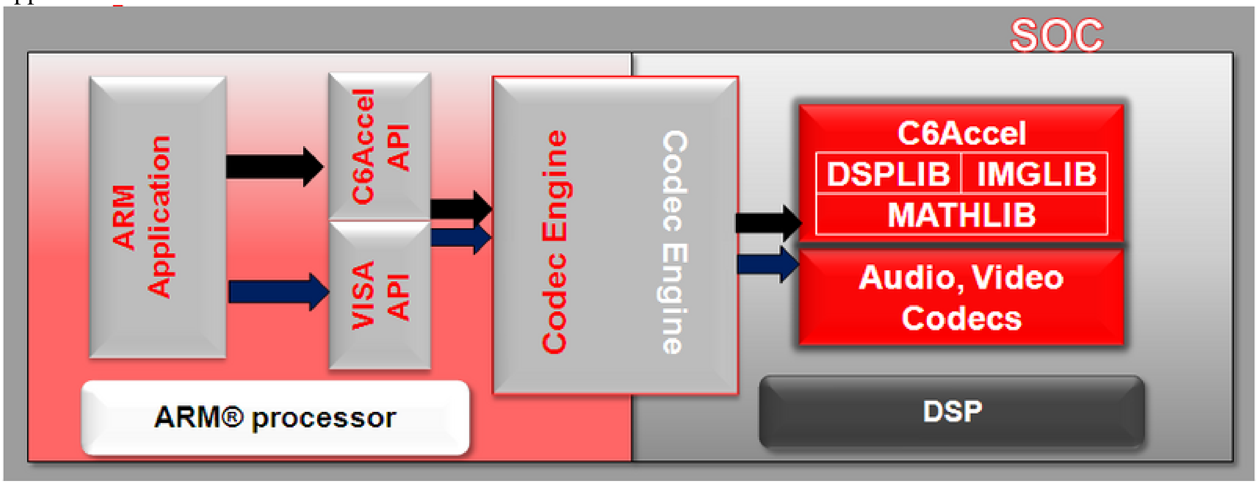

Figure: SoC View with C6Accel

Where Can I Get More Information?

- C6Accel FAQ [2]
- Creating_a_C6Accel_application [3]

Getting started with C6Accel

Where To Get It ?

- C6Accel version1.00.00.04 is part of the DSDK 4.00 and has been integrated into the codec server included in it. C6Accel is licensed under the TI TSPA License.
- Source releases and over view videos are available on[TBD].

Minimum system requirements in case the user is not using the SDK package

- Codec engine: Codec_engine_2_21 and higher
- XDCTools
- XDAIS
- LINUXUTILS
- CODEGEN tools
- DSP BIOS
- BIOSUTILS
- Code sourcery tools
- FRAMEWORK COMPONENTs
- EDMA LLD

Note: Please check release notes to find platform specific dependencies and version of dependencies used in testing C6Accel

Version of DSP Libraries linked to C6Accel

- C64X DSPLIB
- IMGLIB
- IQMATH
- C674x DSPLIB
- FAST MATH
- FAST RTS

How to Install C6Accel

- **Windows**

1. Download the setup file C6Accel-x.x-Setup.exe from the target content page
2. Install the package by running the setup. The setup installs the package along the default path C:\Program files\C6Accel however it is recommended to download the package in the SDK_INSTALL_DIR. If the user chooses to install the package along any other path, he is expected to set approriate paths in the xdcpaths.dat file of the SDK before rebuilding the SDK so that the build tools can find the C6Accel codec.

- **Linux**

1. Download the setup file C6Accel-x.x-Linux-x86-Install from the target content page into a temperory folder (eg /tmp).
2. To install the C6Accel package using the Linux installer,log in using a user account . The user account must have execute permission for the all the installation files. Switch user to “root” on the host Linux workstation and change directories to the temporary location where you have downloaded the bin files. Once you have changed the execute permissions you can go back to a normal user.

```
  host $ su root
  host $ cd /tmp
  host $ chmod +x *.bin
  host $ exit
```

3. Execute the C6Accel installer that you previously downloaded from the SDK target content download page.

```
 For example:
  host $ cd /tmp
  host $ ./C6Accel-x.x-Linux-x86-Install
```

Directory structure The C6Accel installation provides the directory structure below:

```
C6Accel_x_xx_xx_xx
|
+--soc: Example and test application code for C6Accel
|    |
|    +--packages: contains C6Accel codec and unit server
|    +--app: Sample test application to test and benchmark kernels in
C6Accel
|    +--c6accelw: contains C6Accel wrapper API lib
|
+--dsp: Xdais alg that creates the codec
|    |
|    +alg: Xdias algorithm to build C6accel
|    +libs: Version of the libs linked to C6accel
```

```
|
+--docs:  documentation for C6Accel
|
+--README.txt Top-level README file
```

Building the C6Accel Package and running the test application

Assumption: The Build step assumes that the linux kernel, cmem and dsplink module have been pre-built. If the are not built users are expected to build those dependencies before building the C6accel package.

1. Set paths to dependencies in the Rules.make file in the package. Ensure the C6ACCEL_INSTALL_DIR is set to the path where the C6accel package is installed

```
  C6ACCEL_INSTALL_DIR = USER_DEF_PATH
```

2. Build the package by using the command make command in the root directory of the package.

```
  make all
```

3. Set the EXEC_DIR in Rules. make Nd Install the test app in the work area on the filesystem of the target by executing

```
  make install
```

Note: Make install copies the testfiles, loadmodules_c6accel_*PLATFORM*.sh, codec server (.x64p or .x674), dsplinkk.ko and cmemk.ko along with the testapp c6accel_app to the filesystem

4. Power on the device, open Teraterm/Minicom to view the boot and the command propmt for the target. Change directory to working area on the filesystem.

```
  cd $(EXEC_DIR)
```

5. Load the cmem and dsplink modules by running the shell script loadmodules_c6accel_*PLATFORM*.sh

```
  For example: On OMAP3530 or DM3730
  ./loadmodules_c6accel_omap3530.sh
```

Note: The loadmodules script will vary as per the bootargs configured on the device. The default loadmodules corresponds to the bootargs described in the DVSDK 4.x software developers guide.

Note: Ensure that CMEM module was initialized with 20 pools of 0x4096 and heap memory and DSPLINK was created with the correct time stamp

6. Run the app by executing the following command on the command prompt on the target.

```
   ./c6accel_app
```

On a OMAP3 device you will observe the test results for fixed point kernels in C6Accel while on the OMAPL device you will observe test results for fixed as well as floating point kernels. The application also benchmarks all the kernel API calls in C6Accel and writes the benchmarks to a file benchmark.txt in the $(EXEC_DIR) path of the filesystem.

Using C6Accel

User classification based on Level of control

C6Accel package is designed for two class of ARM SoC users who want different level of control.

- **Basic User**: ARM SoC user who wants an abstracted view of the iUniversal and C6Accel implementation design while using C6Accel. For such users, the package has C6Accel wrapper API call library that abstracts the C6Accel design as well as the iUniversal codec engine interface from the ARM application developer
- **Advanced User**: ARM SoC user who has codec engine and iUniversal experience or one who does not mind learning it and some details of C6Accel implementation to extract maximum performance out of C6Accel.

The user experience for utilizing C6Accel for both class of users is as described in the section Interface C6Accel with the Application.

Interfacing the C6Accel with the application.

- To support Engines with remote codecs, a Codec Server must be created. The Codec Server integrates the various components

necessary to house the codecs (e.g. DSP/BIOS, Framework Components, link drivers, codecs, Codec Engine, etc.) and generates an executable. For any user defined application the user needs to integrate c6accel in the user defined codec server. For details of integrating c6accel into the codec server refer Codec Engine Server Integrator user guide [4]

The C6Accel package comes with a prebuilt unitserver that is built specific to the platform is utilized in the sample test app that can be found under soc/app.

To integrate a codec server and invoke a codec the application must contain

- A .cfg file that includes the codec server in the app along with configuration of the codec engine and the dsplink.Refere sample application in the package /soc/app

The configuration file .cfg includes the specific unit server that needs to be invoked in the application:

```
var demoEngine = Engine.createFromServer(
 "omap3530",
 "./omap3530.x64P",
 "ti.c6accel_unitservers.omap3530"
 );
```

Once the engine is configured in the application it can ..be invoked from the application.

Note: The Makefile for the application must include the C6Accel install directory $(C6ACCEL_INSTALL_DIR)

- App_main or files in which c6accel needs to be invoked the following steps need to be followed

1. Include codec engine header files

```
#include <ti/sdo/ce/Engine.h>
#include <ti/sdo/ce/CERuntime.h>
#include <ti/sdo/ce/osal/Memory.h>
```

Note: If the application uses dmai the Memory include can be replaced by a dmai include. These includes are necessary as C6Accel like all DSP codecs expects application developer to allocate contiguous memory for parameters the input and ouput buffers/vectors being passed.

2. Include C6accel application codec header file iC6accel_ti.h or the C6accel wrapper API file c6accelw.h

```
#include "../c6accelw/c6accelw.h"
```

OR

```
  #include "ti/c6accel/iC6accel_ti.h"
```

Note: The codec packge path must be set as include path

3. Declare a C6accel Handle

```
 C6accel_Handle hC6accel = NULL;
```

4. Define Engine Name (same as configured in the .cfg file) and alg name (default: c6accel) In this case

```
#define ENGINENAME "omap3530"
#define ALGNAME "c6accel"
```

5. Before creating a C6accel instance the user must ensure that the codec engine runtime initialization is performed. This can done using the codec engne API CE_Runtime_init() API before any of the C6accel APIs are used in the code.

```
  CE_Runtime_init() ;
```

6. Once the codec engine intialization is performed. The user can call into the C6accel_create API that willl generate the C6accel handle.

```
 hC6accel = C6accel_create(engineName, NULL,algName, NULL);
```

Refer to [ ] to find details of the create API call.

7. Once the C6accel_create is successfully invoked , basic user can make calls to kernels in the codec using API calls as shown in section C6Accel Wrapper Library Reference [5] and advanced users can utilize the chaining API feature as explained in section Chaining calls to kernels in a single API call to C6ACCEL codec [6] Note: All input and output buffer parameters used in these API calls need to be contiguous in memory.

8. Once the C6Accel functionality in the application is complete the user is expected to tear down the codec using C6accel_delete() API.

Note: C6Accel can work with heap as well as pool CMEM memory allocations.

Accessing floating point kernels in C6Accel

C6Accel contains fixed point as well as floating point kernels. However inorder to maitain portability between C64+ and C674x devices C6accel package is configured to provide access to just the fixed point kernel library. On C647x devices inorder to access the floating point kernels in C6Accel, the C6Accel package module contains a FLOAT Boolean flag which needs to be set inorder to access the floating point kernels.Default settings sets this FLOAT flag set to false.

Inorder to access floating point kernels add the following script to the .cfg file of the application.

```
   var C6ACCEL = xdc.useModule('ti.c6accel.ce.C6ACCEL');
   C6ACCEL.alg.FLOAT=true;
```

When the application builds the codec server with this float flag set, the C6accel package is directed to pick the .l674 library which contains the floating point kernels in addition to the fixed point kernels. A build error will be seen on a C64+ device if the application configuration tries to set this flag.

C6Accel wrapper library Reference

The C6Accel wrapper library is designed to abstract iuniversal and C6accel design considerations and provides an interface that appear like a simple function call within an application. The C6Accel wrapper library is available in source as well as object to be linked to the application. Refer package directory /soc/c6accelw to view wrapper library source.

Using C6Accel wrapper library in the application

Inorder to use C6Accel wrapper APIs in the application add the appropriate library from $(C6ACCEL_INSTALL_DIR)/c6accelw/lib in the Make file for the application:

For eg. On OMAP3530 platform add the following to the Makefile to include the c6accel wrapper library in the application.

```
 OBJFILES +=
$(C6ACCEL_INSTALL_DIR)/c6accelw/lib/c6accelw_omap3530.a470MV
```

C6accel_Handle

C6accel_Handle is a handle to the C6Accel Object. The C6Accel Object is defined as

```
 C6accel_Object{
                Engine_Handle hEngine;
                UNIVERSAL_Handle hUni;
                E_CALL_TYPE callType;
               }C6accel_Object;
```

The C6Accel Object carries the Engine Handle and the IUniversal Handle required for the current instance. E_CALL_TYPE is a custom defined datatype that can be take values as ASYNC or SYNC based on application requirements to make asynchronous or synchronous calls to the DSP.

C6accel_Handle C6accel_create(String engName, Engine_Handle hEngine,String algName, UNIVERSAL_Handle hUniversal)

Arguments:

- engName : Engine Name
- hEngine : Engine Handle
- algname : Algorithm name
- hUniversal: Universal Handle

Return: API returns C6Accel Handle if invoked successfully or NULL if create call failed

Description: This API returns a C6Accel Handle from the Engine Handle and universal handle passed from the application.

Note: Default C6Accel handle is configured to make synchronous calls to the DSP. To enable asynchronous calling of the DSP refer to [Making_Asynchronous_Calls_to_DSP_kernels_in_C6Accel]

Details:

Case	Engine Handle	Universal Handle	Action
Case 1	NULL	NULL	Creates engine handle from engine name and universal handle from algname and returns C6accel Handle
Case 2	Passed from app	Null	Creates universal handle and passes exiting engine handle to C6Accel object and return C6Accel HAndle
Case 3	passed from app	passed from app	Passes engine and universal handle to C6accel object and returns C6Accel handle
Case 4	NULL(No engine name passed)	**X**	Returns NULL
Case 5	passed from app	NULL (No algname)	Returns NULL

X : Don`t care

int C6accel_delete(hC6accel_handle)

Arguments:

- hC6Accel: C6Accel Handle
- Return: 1 when passed and 0 when failed

Description: This API tears down the C6accel instance by closing the codec engine and iUniversal interface using the C6accel Handle.

Error Codes

For all the C6Accel wrapper API kernel that call a funtionality on the DSP, the error codes returned from the API are documented below

- **_EOK (0):** No error occured
- **_EFAIL (-1) :** Error occured in invoking codec engine interface

This error is most likely occur only if the buffers/vectors being passed to the codec are not assigned contiguous memory. It can also occur if the application makes an asynchronous call to C6Accel when there is already a pending asynchronous call.

Specific fail mesages

- **_PARAMFAIL (-6) :** Error due to invalid parameters

This error is like to occuring when the parameters passed do not satisfy the parameter specifications of the underlying kernel. Check Wrapper API documentation of that specific kernel to know more about the range of permissible parameters.

- **_FXNIDFAIL (-7):**Error on function ID passed

This error is likely when the application passes a wrong function ID to the codec. This error is unlikely to occur while using the wrapper API calls as the passing of the function ID is done inside the wrapper code. Incase an advanced user comes across this error please verify if the function ID being passed is defined in the application codec interface header file iC6accel_ti.h.

Reference based on categories of kernels in C6Accel

The C6Accel organizes kernels into seven different functional categories: Digital Signal processing , Image Processing , Math, Analytics, Medical, Audio/Speech processing and Power/Control.
Note: The initial version of the C6Accel provides only kernels in the Digital signal processing, Image Processing and Math category.

The references for these functional categories have been furnished below:

- C6Accel Signal Processing kernels API Reference [7]
- C6Accel Image Processing kernel API Reference [8]
- C6Accel Math kernel API Reference [9]
- Analytics based kernels
- Medical based kernels
- Audio/Speech processing
- Power/Control based kernels

Advanced Features

Benchmarking and Performance of the kernels

These are the results from the benchmarking tests for kernels in C6Accel

Benchmarking results:

- Benchmarking on DM6467 [10]
- Benchmarking on OMAP3530 [11]
- Benchmarking on OMAPL137
- Benchmarking on OMAPL138 [12]

Related douments and links

- C6Accel FAQ [2]
- Wiki Design Doc
- Reference guides for all DSP libraries available on TI.com
- Download Link

References

[1] http://ap-fpdsp-swapps.dal.design.ti.com/index.php/Advanced_Features_of_C6Accel#Adding_a_new_kernel_to_C6Accel_and_integrating_with_codec_server
[2] http://ap-fpdsp-swapps.dal.design.ti.com/index.php/C6Accel_FAQ
[3] http://ap-fpdsp-swapps.dal.design.ti.com/index.php/OMAPL138_Software_Developers_Guide#Creating_a_C6Accel_application
[4] http://focus.ti.com/general/docs/litabsmultiplefilelist.tsp?literatureNumber=sprued5b
[5] http://ap-fpdsp-swapps.dal.design.ti.com/index.php/C6Accel_Reference_guide#C6Accel_wrapper_library_Reference
[6] http://ap-fpdsp-swapps.dal.design.ti.com/index.php/Advanced_Features_of_C6Accel#Chaining_calls_to_kernels_in_a_single_API_call_to_C6ACCEL_codec
[7] http://ap-fpdsp-swapps.dal.design.ti.com/index.php/Signal_Processing_Kernels_in_C6Accel
[8] http://ap-fpdsp-swapps.dal.design.ti.com/index.php/Image_Processing_Kernels_in_C6Accel
[9] http://ap-fpdsp-swapps.dal.design.ti.com/index.php/Math_Kernels_in_C6Accel
[10] http://ap-fpdsp-swapps.dal.design.ti.com/images/5/51/Benchmarking_6467.txt
[11] http://ap-fpdsp-swapps.dal.design.ti.com/images/0/0d/Benchmark_omap3530.txt
[12] http://ap-fpdsp-swapps.dal.design.ti.com/images/b/b8/Benchmarking_omapl138.txt

Article Sources and Contributors

C6Accel Reference guide *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?oldid=66189 *Contributors*: A0272049, GaganMaur

Image Sources, Licenses and Contributors

Image:C6Accel SoCview.bmp *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?title=File:C6Accel_SoCview.bmp *License*: unknown *Contributors*: A0272049