QNX® Aviage Multimedia Suite 1.2.0
QNX Codec Engine for OMAP3530

The QNX Codec Engine for OMAP3530 is a codec engine created to support QNX multimedia applications on OMAP3530 (Beagle) platforms, providing accelerated decoding for AAC, MP3 and WMA audio media formats.

It is delivered as binaries and as source. The binaries provide a working example for demonstration in a pre-defined configuration. The source can be used to modify the configuration provided in the binary delivery. This technote includes the following sections:

- Directory structure
- Install and set up the QNX Codec Engine
- Build the QNX Codec Engine

Directory Structure

The Codec Engine directory includes the following four package repositories (directories):

- **`codec_engine_2_22`** — the primary Codec Engine package, which also includes the Codec Engine tools package, documentation and release notes:
  - The QNX OSAL implementation code can be found in **`codec_engine_2_22/packages/ti/sdo/ce/osal/qnx`**
  - The QNX IPC implementation stub is in **`codec_engine_2_22/packages/ti/sdo/ce/ipc`**

  In addition, to allow use of the QNX OSAL and IPC, some RTSC script changes have been made in **`codec_engine_2_22/packages/ti/sdo/ce`**.
- **`framework_components_2_22`** — a supporting package repository (with some supporting libraries). Currently, we only use the trace module from this repository, and no changes were made to its contents. However, configuration and **`Make`** file changes to the build are required; see “Building the Codec Engine primary library” for details.
- **`omap3530_dvsdk_combos_3_16`** — the TI- provided decoder server and codecs. All codecs can be found under **`packages/ti/sdo/codecs`**, while the server is under **`packages/ti/sdo/servers`**. No changes were made to this directory. However, configuration and **`Make`** file changes to the build are required; see “Building the Codec Engine primary library” for details.
- **`qnx_packages`** — the main QNX-ported packages (libraries) directory. It includes the following subdirectories:

- **gnu/targets/arm** — the specification for building a target; it specifies the QNX compiler tool chain, and is used by all package builds
- **ti/sdo/ce/ipc/dsplink** — the QNX implementation of the layer between **dsplink** and the Codec Engine
- **sdo/qnxutils/cmem** — the QNX CMEM implementation
- **ti/bios/power** — the QNX implementation of the power module
- **dsplink/dsplink160** — the package that specifies the QNX-ported **dsplink** client-side library
- **codec_engine** — the QNX shim layer of the Codec Engine API. Building this package will produce a **libcodecengine.so** library your application can use. This package specifies the decoder server to be used; in the current implementation, this is the decodeCombo server from the **omap3530_dvsdk_combos_3_16** package.

Install and set up the QNX Codec Engine

The binaries for the QNX Codec Engine for OMAP3530 can be downloaded from the QNX download site, at **http://www.qnx.com/download/**.

This technote section describes:

- Components list
- Codec Engine Installation and configuration
- Configuring **io-media** for the Codec Engine

Components list

This section lists the components that are included in the pre-built QNX Codec Engine for OMAP3530 (binaries delivery). Components identified as such are from Texas Instruments (TI); all others are from QNX, using the TI Codec Engine library.

Codec Engine components

The Codec Engine components include:

- **arm/so.le/libcodecengine.so.1** — library for the ARM side of the Codec Engine
- **arm/so.le/libcodecengineS.a** — library for the ARM side of the Codec Engine
- **dsp-bins/decodeCombo.x64P** — the pre-built DSP image containing the DSP side of the Codec Engine, plus watermarked decoders, provided by Texas Instruments (TI).
- **dsp-bins/dsplink160** — the QNX port of **DSPLink161**

- **ce_loader** — a QNX utility used to load the DSP image and stay resident for the duration of the multimedia session. Without this utility, the **ce_audio_decoder** filter loads the DSP as each new track is started and unload the DSP when the track is terminated, causing longer wait times between tracks.

MME components

The MME components included in the pre-built Codec Engine are the following **io-media** filters for audio decoding on the OMAP3530 platform:

- **ce_audio_decoder** — the audio filter

This filter is located under **lib/mmedia/filters/decoders/**.

Board Support Package

The QNX Codec Engine for OMAP3530 needs the TI OMAP 3530 EVM and Beagle Board Support Package (BSP), which can be downloaded from QNX download site, at **http://www.qnx.com/download/**.

Installation and configuration

To configure and run the Codec Engine on an OMAP3530 Beagle target, complete the following tasks in sequence:

1. Open the build file for the Beagle BSP (**beagle.build**) and reserve memory for **dsp**, **dsplink** and **cmem** by changing the startup command line the in the build file, as follows:

   ```
   startup-omap3530 -L 0x87E00000,0x200000 -x 0x85800000,0x02600000 -v -p
   ```

2. Copy the decoder filters to the multimedia DLL directory on the target:

   ```
   # cp ce_audio_decoder.so path/armle/lib/dll/mmedia
   ```

3. In your MME **boot.sh** script, copy the DSP image to **/tmp**:

   ```
   # cp decodeCombo.x64P /tmp
   ```

4. Launch **dsplink** and **ce_loader**:

   ```
   # dsplink160 -q
   # ce_loader
   ```

5. When these applications have been started, you can start **io-media** and the other MME components.

For more information about **io-media**, see the *MME Utilities Reference*.

Configuring io-media

After you have installed and configured the Codec Engine, you need to:

- Configure **io-media** to use the DSP decoder
- Configure **io-media** to keep DLLs resident

Configure io-media to use the DSP decoder

To configure **io-media** to use the DSP decoder, change the **io-media** configuration file as shown below.

In the **mmf_graphbuilder** module configuration section:

```
module-options {
                module = "mmf_graphbuilder"
```

change:

```
format {
        url = "*.mov"
        parser = "mp4_parser"
        decoder = "qnx_raac_decoder"
}
format {
        url = "*.mp[a123]"
        parser = "mpega_parser"
        decoder = "xing_mpega_decoder"
        # You can set MMF graph-level parameters here:
        # graphparam {
        #       name = "foo"
        #       value = "bar"
        # }
}
format {
        url = "*.mp4"
        parser = "mp4_parser"
        decoder = "qnx_raac_decoder"
}
format {
        url = "*.m4a"
        parser = "mp4_parser"
        decoder = "qnx_raac_decoder"
}
format {
        url = "*.aac"
        parser = "aac_parser"
        decoder = "qnx_raac_decoder"
}
format {
        url = "*.wma"
        parser = "wma9_parser"
        decoder = "wma9_decoder"
}
```

to:

```
format {
        url = "*.mov"
        parser = "mp4_parser"
        decoder = "ce_audio_decoder"
}
format {
        url = "*.mp[a123]"
        parser = "mpega_parser"
        decoder = "ce_audio_decoder"
        # You can set MMF graph-level parameters here:
        # graphparam {
        #       name = "foo"
        #       value = "bar"
        # }
}
format {
        url = "*.mp4"
        parser = "mp4_parser"
        decoder = "ce_audio_decoder"
}
format {
        url = "*.m4a"
        parser = "mp4_parser"
        decoder = "ce_audio_decoder"
}
format {
        url = "*.aac"
        parser = "aac_parser"
        decoder = "ce_audio_decoder"
}
format {
        url = "*.wma"
        parser = "wma9_parser"
        decoder = "ce_audio_decoder"
}
```

Configure `io-media` to keep DLLs resident

To significantly reduce the time between track changes, you should also configure **`io-media`** to keep the **`ce_audio_decoder`** DLLs resident.

In the **`mmf`** module configuration section:

```
module-options {
                module = "mmf"
```

Add the following to keep the DLLs resident:

```
keepdll {
          name = "ce_audio_decoder"
          optional = yes
       }
```

Build the QNX Codec Engine

This section presents a brief description of how to build and port the Codec Engine Version 2.22 for an OMAP3530 platform. The ported Codec Engine uses the QNX-ported DSP/BIOS Link (**dsplink** version 1.61) as the transport between the GPP and the DSP.

- Prerequisites
- Building the Codec Engine
- Customizing the codec for the Codec Server
- Setting up and customizing the memory map

To modify the Codec Engine source you will need to obtain some proprietary tools, such as the TI Code Composer Studio; TI codecs, and relevant licences. Please contact QNX for more information.

Prerequisites

Before building the Codec Engine, you need to have a correctly built QNX-ported **dsplink** (1.61), which is available from QNX in a separate package.

To build the QNX-ported **dsplink** you need:

- a QNX compiler tool chain
- QNX 6.4.*n* on Windows or Linux
- the Texas Instruments XDC tools, version 3.10.2 or more recent; see
- the Texas Instruments DSP-side compiler, **cg6x** 6.0.15 or more recent
- the Texas Instruments BIOS **bios** 5.33.03 or more recent

For more information, or to obtain the Texas Instruments tools, see the Texas Instruments site at:

http://focus.ti.com/dsp/docs/dspsupportaut.tsp?familyId=44&sectionId=3&tabId=416&toolTypeId=30

Building the Codec Engine

Building the Codec Engine for QNX requires three tasks:

1 Build the Codec Engine library, including the primary library and the framework component.

2 Build the codecs and codec servers.

3 Build the QNX packages, including the CMEM, power, IPC **dsplink**, and the shim layer (the final **libcodecengine.so**).

Detailed instructions for building the component parts of the Codec Engine are provided below. As a general rule:

- if a package includes a **Makefile**, do: **make clean**, then **make**
- if there is no **Makefile** in the package, do: **xdc clean**, then **xdc**

Recommended build order

The first time you build the Codec Engine, you should perform the following tasks and build the packages in the following sequence:

1 Set up the environment variables.

2 Build **dsplink**.

3 Build the target package.

4 Build the Codec Engine primary library.

5 Build the framework component package.

6 Build the **dsplink** client library package.

7 Build the power package.

8 Build the **ipc/dsplink** package.

9 Build the CMEM package.

10 Build the codecs and codec servers.

11 Build the shim layer (**libcodecengine.so**).

- If a package is already built, you can skip building it and move on to the next package.
- If a package build fails because another package on which it depends is missing, simply build the required package, then rebuild the package that failed.
- After the first build of the all the packages, you can modify any package, build it, then rebuild the shim layer to generate a new **libcodecengine.so**.

Set up the environment variables

To set up the environment variables for the Codec Engine:

1. In the Codec Engine root directory, edit **`setbuildenv.sh`** to make sure the correct tool paths are set.
2. Export the **XDCPATH** environment variable so that the XDC tool will find the QNX packages, instead of using the default Linux packages. The order of the directories is very important, as it determines what gets over-written, and what gets kept.
3. After you have finished editing the script, finishing setting up the environment variable by enter the following on the command-line:

```
$ . ./setbuildenv.sh
```

Build `dsplink`

To build the QNX-ported **`dsplink`**:

1. Build **`dsplink`**.
2. Copy the client-side library **`libdsplink160-client.a`** to the following directory: **`qnx_packages/dsplink/dsplink160/arm/a.le.clent/`**.

Build the target package

You don't need to change anything to build the target package; simply:

1. Use **`xdc clean`**, then **`xdc`** to build the package.

However, if you want to change the compiler, you can change the **`GCArmv5T.XDC`** file, which can be found at:

```
$(rootDir)/bin/arm-unknown-nto-qnx6.4.0-ar
```

Build the Codec Engine primary library

To build the Codec Engine primary library, you need to:

1. Set the paths and environment variables.
2. Build the primary packages.

Set the paths and the environment variables

To set the paths and the environment variables:

1. In the **`codec_engine_2_22/packages`** directory, edit the following files to ensure that they set the appropriate paths and environment variables:

- **config.bld**
- **user.bld**
- **xdcpaths.mak**

If you are not building the DSP side of the image or changing the compiler version, you should not have to change anything in the files listed above. Nevertheless, it is a good idea to review their content to confirm that nothing needs to be changed.

Building the DSP side of the image

A standard build does not require a build of the DSP side of the image (codecs and servers). However, if you want to build the DSP side of the image, you need to edit the **user.bld** file. For example:

```
"DSP":    [{doBuild: true, // DSP builds (DSP servers)
 // Specify the "root directory" for the compiler tools.
 // NOTE: make sure the directory you specify has a "bin" subdirectory
 cgtoolsRootDir: "C:/CCStudio_v3.3/cg6x_6_0_15",
```

Build the primary packages

To build the primary packages for the Codec Engine base library:

1 In the **codec_engine_2_22/packages/ti/sdo/ce** directory, go into the subdirectories for each package and use **xdc clean**, then **xdc** to build each package. You must build at least the following packages:

- **alg**
- **audio**
- **audio1**
- **global**
- **image**
- **image1**
- **ipc/qnx**
- **ipc**
- **node**
- **osal/qnx**
- **osal**
- **universal**
- **utils/xdm**

2 When you have finished building all the primary packages:

2a Return to the **codec_engine_2_22/packages/ti/sdo/ce** directory.

2b Use **xdc clean**, then **xdc** to build this package.

- You need some familiarity with the Codec Engine and the TI tools to complete this task.
- The packages you build depends on the codecs you will use.

Build the framework component package

Currently, the codec engine only needs the trace module from this package. To build this module with supporting libraries:

1. In the **framework_components_2_22/packages/ti/sdo/utils/trace** directory, check the **config.bld** file to see if you need to change any paths or environment variables. If you do need to change a path or environment variable, follow the procedure as you did to build the Codec Engine primary library.
2. Use **xdc clean**, then **xdc** to build the package.

Build the dsplink client library package

The **dsplink** client library package tells the build tool where the **dsplink** client library and header files are located.

To build the **dsplink** client library package:

1. In the **qnx_packages/dsplink/dsplink160** directory, check the **config.bld** file to see if you need to change any paths or environment variables. If you do need to change a path or environment variable, follow the procedure as you did to build the Codec Engine primary library.
2. Use **xdc clean**, then **xdc** to build the package.

Build the power package

Currently, there is nothing you need to do for the power package. Power control for the OMAP3530 platform is done in **dsplink**, and the separate power package provided is a dummy implementation of the power interface, already built.

Build the ipc/dsplink package

The **ipc/dsplink** package is the QNX implementation of the layer between **dsplink** and the Codec Engine, providing the glue layer between these two components.

To build the **ipc/dsplink** package:

1. In the **qnx_packages/ti/sdo/ce/ipc/dsplink** directory, check the **config.bld** and **user.bld** files to see if you need to change any paths or environment variables.

If you do need to change a path or environment variable, follow the procedure as you did to build the Codec Engine primary library.

2 Use **make clean**, then **make** to build the package.

Building the CMEM package

The CMEM package is the QNX CMEM implementation. To build the CMEM package:

1 In the **qnx_packages/qnx/sdo/qnxutils/cmem** directory, edit the **rules.make** files, if you want to change the compiler version or comnpiler options.

2 Use **make clean**, then **make** to build the package.

- Please refer to the Linux CMEM documents from TI for more information about the CMEM package.
- The current QNX CMEM implementation only supports single application process access.

CMEM reserved memory

The default memory that needs to be reserved for CMEM is defined in **cmem_module.c**, as follows:

```
#define BLOCK_0_ADDR  0x85800000 //start physical address of the CMEM
#define BLOCK_0_SIZE 0x1000000  //size of the CMEM.
```

And the default pool block allocation is as follows:

```
/*default pool block allocation*/
static int pools[NBLOCKS][MAX_POOLS][2] = {
{
    {20,4096}, //number of blocks and block size
    {8,131072},
    {5,1048576},
    {1,1429440},
    {1,256000},
    {1,3600000},
    {5,829440},
},
};
```

Customizing CMEM parameters

To customize the CMEM parameters, use the environment variable **CMEM_PARAMETERS** by specifying, as follows:

- the start of the CMEM physical address (hexadecimal)
- the size, in bytes, of CMEM (hexadecimal)
- the specifics of each pool allocation, where the first number is the number of pools and the second number is the size, in bytes, of each pool; for example, `20,4096` means 20 pools of 4096 bytes each.

For example:

```
export CMEM_PARAMETERS=0x85800000,0x10000000,20,4096,8,131072,5,1048576,1,1429440,1,256000,1,3600000,5,829440
```

If you customize the CMEM parameters, you must export the **CMEM_PARAMETERS** environment variable before you start the Codec Engine related components.

Build the codecs and codec servers

To build the codecs and codec servers:

1. In the `omap3530_dvsdk_combos_3_16` directory, edit:
   - the `rules.make` file information to ensure that the correct path are set
   - the `config.bld` file information to ensure that the correct paths are set for `C64P.rootDir` and `GCArmv5T.rootDir`
2. Save the file, and from the command line do a `make clean`, then a `make` to generate the decoder server dsp image.

The generated decoder server DSP image will be: `decodeCombo.x64P`, in the `omap3530_dvsdk_combos_3_16/packages/ti/sdo/servers/decode/` directory.

You need to copy this binary to the target, and specify it when rebuilding `libcodecengine.so`.

Building production and evaluation servers

If you have a non-watermarked version of the package from TI, before building you can edit the `XDCARGS` option in the `Makefile` for building the server to make either a production or an evaluation server:

- `XDCARGS="eval"` — build an evaluation server
- `XDCARGS="prod"` — build a production server

Build the shim layer (`libcodecengine.so`)

The shim layer (**`libcodecengine.so`**) is the library you need to integrate the Codec Engine into the filter. Building the shim layer is the final step in building the codec engine.

- You must have successfully built all the other packages before you build the shim layer.
- If modify or build any other package, you must *rebuild the shim layer*.

To build the shim layer:

1. If you want to change the compiler version, in the **`qnx_packages/codec_engine/nto/arm/so.le`** directory, edit the **`user.bld`** file to make the change.
2. In the **`qnx_packages/codec_engine`** directory, use **`make clean`**, then **`make install`** to generate **`libcodecengine.so`**, the codec engine.

Changing the default DSP image

By default, the shim layer looks for the DSP image on the running target system at **`/tmp/decodeCombo.x64P`**.

To change this default, modify the **`cebuild.cfg`** file in the **`qnx_packages/codec_engine/nto/arm/so.le`** directory as follows:

```
var myEngine = Engine.createFromServer(
//creating audio dec server
    "audiodec",
    "./decodeCombo.x64P",
    "ti.sdo.servers.decode"
);

myEngine.server = "/tmp/decodeCombo.x64P"
```

where:

- **`"audiodec"`** is the codec engine name
- **`"./decodeCombo.x64P"`** is the relative path image to the **`ti.sdo.servers.decode`** package
- **`myEngine.server = "/tmp/decodeCombo.x64P"`** is the path where the Codec Engine looks for the image on the running target; you can change this location according to your needs

Customizing the codec for the codec server

The codecs server is like a codec container that includes all the codecs you want to have in the generate DSP image. To customize the image, you can:

1. In the **omap3530_dvsdk_combos_3_16/packages/ti/sdo/servers/decode/** directory, modify the **decode.cfg** file, as required to enable or disable specific codecs.
2. Rebuild the codec server.
3. Rebuild **libcodecengine.so**.

Setting up and customizing the memory map

If you need to shrink memory usage, or move the memory to a different location, you can change the memory map.

To change the memory allocation, you need to change the memory map. The memory map is in the **decode.tcf** file in the **combos_3_16/packages/ti/sdo/servers/decode** directory.

For instructions on how to set up and customize the memory map, refer to **decode_combo_datasheet.pdf** which can be found in the **omap3530_dvsdk_combos_3_16/packages/ti/sdo/servers/decode/docs** directory. This document includes all the information you will need about the memory map used in the codec server.

The codec server's memory regions are used by either the GPP or the DSP, or are shared by both the GPP and the DSP. For more information about CMEM, refer to the "CMEM Overview" on the *Texas Instruments Developer Wiki* at **http://wiki.davincidsp.com/index.php?title=CMEM_Overview**.

You should also check with Texas Instruments to find out how much memory each codec requires.

Reserving memory in the BSP startup

You need to reserve the memory region used by **dsplink** and the DSP codec server in the BSP startup, in order to prevent the kernel from allocating the required memory region for another use.

To reserve the memory, specify command-line arguments in the BSP startup; for example:

```
startup-omap3530 -L 0x87E00000,0x200000 -x 0x85800000,0x02600000
```

where:

- the **-L** option reserves for **dsplink** a memory region of 0x200000 bytes, with a start address of 0x87E00000

- the **`-x`** option reserves for the codec engine a memory region of 0x2600000 bytes with a start address of 0x85800000; this memory region includes the CMEM, and other memory specified in the **`decode.tcf`** file

---

- The startup above is the default startup; you can modify this startup to suit your needs.
- You can customize all the above parameters, but it is best *not* to leave gaps between the allocated memory regions. You should also make sure the startup commands and the **`decode.tcf`** file are consistent.

Published under license by:

QNX Software Systems International Corporation
175 Terence Matthews Crescent
Kanata, Ontario
K2M 1W8
Canada
Voice: +1 613 591-0931
Fax: +1 613 591-3579
Email: **info@qnx.com**
Web: **http://www.qnx.com/**