Simplifying DSP Development with C6EZ Tools
DSP Development made easier with C6EZ Tools

Seamlessly ports ARM code to DSP (ARM Developers)

Provides ARM access to ready-to-use DSP kernels (System Developers)

Graphical Software Development Tool for quick prototyping (DSP Developers)
C6EzFlo – For DSP developers
Target Audience for C6EZFlo

- **Target Developers**
  - Signal Processing Experts
  - People familiar with older generations of DSP Processors

- **Challenges**
  - Not expert software engineers
  - Unfamiliar with parallel VLIW architectures
  - Unfamiliar in C-level optimization
  - Unfamiliar with complex peripheral drivers

- **Strategy: C6EZFlo**
  - Drag-and-drop environment
    - For IO
    - For compute
  - Optimized underlying blocks
  - Well-structured easy to modify C output
  - Easy capability to create new blocks
C6EZFlo Overview

C6EZFlo’s graphical development tool creates a visual signal flow, making DSP prototyping easy, fast and cost-efficient.

Allows drag and drop functionality to connect i/o blocks to peripherals on the DSP without needing to know DSP code.

Generates optimized C code that is heavily commented and easily modifiable for further development.

Features broad and expandable block set to support a wide range of end applications.
C6EZFlo V2.0 Window View
Drawing a System in C6EZFlo

Blocks are self-contained algorithms

Connect blocks to implement a system

Configure system with individual block parameters

Special framework block controls global parameters

Integrated help pages for all blocks
Generating the Application

Draw a block diagram using the graphical tool

Click Generate Code button

New source files are automatically added to the project
Understanding the Generated Source

- **<app>_main.c**
  - **main**
  - Program entrypoint

- **<app>_threads.c**
  - **<thread>_TSK**
  - Processing loop(s)

- **<app>_blocks.c**
  - **<block>_create**
  - **<block>_init**
  - **<block>_proc**
  - **<block>_ctrl**
  - Optimized algorithm code

Supporting Files

- **<app>_h**
- **<app>_tcf**
- Library Files

includes

calls
# C6EZFlo Processing Blocks

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Now</th>
<th>2Q11</th>
<th>4Q11</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Foundation Signal Processing Software Algorithms</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Digital Signal Processing</td>
<td>12</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>Image Processing</td>
<td>7</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Math</td>
<td>49</td>
<td>49</td>
<td>49</td>
</tr>
<tr>
<td>Filter Package</td>
<td>7</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td><strong>System Design Components</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Signal Generation</td>
<td>6</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>System Design</td>
<td>9</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>Board I/O Connectivity</td>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td><strong>Application Specific Software Algorithms</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProAudio: Audio Processing Library</td>
<td>2</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Power and Energy</td>
<td></td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>Vision And Analytics</td>
<td></td>
<td></td>
<td>30</td>
</tr>
<tr>
<td><strong>Total Supported Functions</strong></td>
<td>110</td>
<td>157</td>
<td>252</td>
</tr>
</tbody>
</table>
C6EZFlo – Getting Started

Supported Devices
C674x, OMAP-L13x, DM643x, DM648, DM647, C6424, C6421, C6452

Availability
Downloadable at product page
Integrated into CCS v5 (April 2011)

Applicable Links
C6EZTools Wiki – www.ti.com/c6eztoolswiki
C6EZFlo Wiki – www.ti.com/c6ezflowiki
TI Product Page – www.ti.com/c6ezflo

Technical Support
Forums - http://e2e.ti.com/support/default.aspx

Feedback/Feature Request
Mailing list - C6EZFlo@list.ti.com
C6EzRun – For ARM Developers
What can a ARM developer do with the DSP?

- Run math intensive algorithms
- Leverage some DSP features, like floating-point computations
- Free up ARM MIPS for additional system features
- Save money by not moving to a more powerful & expensive ARM!
- Get true real-time response via DSP without sacrificing features of a high-level OS like Linux and Android
DSP Performance – Complex FFT

- FFT runs ~10x faster on DSP than on ARM.
- Small FFT size, overhead dominates, running on DSP does not provide advantage.
- Larger FFT size, overhead absorbed, running on DSP provides advantage.

SoC: ARM9 + Floating-Point DSP
CPU Frequency: 300MHz
Code & Data Location: External DDR2 Memory
Instruction and Data Cache: Enabled
Single-precision floating-point data buffers.
C6EZRun: Easy DSP Performance for the ARM Developer

Target Developers

• ARM Developers

Challenges

• Unfamiliar with
  • Heterogeneous programming environments
  • Graphical IDEs
  • IPC

Strategy: CEZ6run

• GCC-style build flow
• Automatically puts
  • Whole app
  • Or Selected functions
• On DSP
C6EZRun quickly ports ARM code to run on the DSP, making DSP development easy, fast and cost efficient.

- Compile C code for DSP execution right from the Linux shell.
- No modification to ARM application code required.
- Provides a familiar development environment with a GCC-like interface, simplifying the user/development experience.
C6EZRun Project

- Open Source Project, hosted on gforge.ti.com

- Intends to provide an abstracted mechanism for getting code running on the DSP

- Project Goals
  - Introduce DSP Development to ARM/Linux developer
  - Provide an abstracted mechanism for getting code running on the DSP
  - Start getting Linux and open-source community using the DSP
Leverage C6EZRun in 3 ways

- **Common Back-end Libraries**: Provide ARM and DSP software infrastructure to load and interact with the DSP
- **C6RunLib**: Partition an application between the ARM and DSP
- **C6RunApp**: Run an entire application on the DSP
Goal is to automate building an ARM-side library from user’s C code of critical functions.

User can call into that library in their app, and calls will be executed on the DSP.

Consists of automating creation of RPC framework, hiding low-level inter-core communication and other TI specific bits as possible.

Extract Critical Functions as a library using C6RunLib.
ARM-only Development

ARM+DSP Development with c6runlib
C6RunLib Example

$ c6runlib-cc -c -O2 -o dummy.o dummy.c

- Above converts C code containing critical functions to C6000 object file
- Also analyzes global C functions and generates ARM-side remote procedure call stubs

$ c6runlib-ar rcs dummy_dsp.lib dummy.o

- Add object file to library dummy_dsp.lib
- Underneath, the dummy.o object file is linked to a DSP executable and compiled into the framework
- Framework object file and stubs object file is archived into the lib
- ARM-side stubs resolve symbols for ARM application when built against the library
C6RunApp

Consists of two parts: Backend library builds and front end user interface

Backend libraries collate DSPLink, CMEM, DSP/BIOS, and custom code

Front end interface is a command-line cross compiler script (acts like GCC)

Entire application retargets to DSP, but with C I/O access to ARM/Linux
ARM-only Development

ARM+DSP Development with c6runapp

module1.c
module2.c
main.c

module1.c
module2.c
main.c

GCC Toolchain

ARM Executable

ARM Executable

Embedded DSP Executable

ARM+DSP Development with c6runapp

ARM-only Development
Example C6RunApp Usage

```
$ c6runapp-cc -o hello_world hello_world.c
```

- Compiles `hello_world.c` to C6000 object file, which is then linked into a DSP executable

- Executable is compiled into the ARM side framework, which is used to build an ARM-side executable called `hello_world`

```
$ c6runapp-cc -c -o file1.o file1.c
$ c6runapp-cc -c -o file2.o file2.c
$ c6runapp-cc -o myApp file1.o file2.o
```

- Individually compiles `file1.c` and `file2.c` to object files

- Links object files together to create application called `myApp`, with DSP executable image embedded inside it.
Notes on the C6EZRun tools

• Front end script wraps the TI C6000 Codegen tools (specifically cl6x)

• Usage syntax is primarily a GCC clone
  – Supports many GCC options and translates them to appropriate cl6x options
  – Many GCC optimization/warning options are silently ignored
    • They may have no direct analogue in the TI compiler
    • They may not apply
  – Unknown options passed directly to cl6x command-line
    • Allows specific usage of TI C6000 codegen tool
    • Can still perform optimizations and options that only apply to the C6000 codegen tools
C6EZRun – Making DSP Development EZ

Supported Devices
OMAP-L13x, C6A816x, CAA814x, DM814x, DM816x, DM3730, DM6467, OMAP3530,

Availability
Downloadable on product page
Standard part of Software Development Kit

Applicable Links
C6EZTools Wiki – www.ti.com/c6eztoolswiki
C6EZRun Wiki – www.ti.com/c6ezrunwiki
TI Product Page – www.ti.com/c6ezrun

Technical Support
Forums - http://e2e.ti.com/support/default.aspx

Feedback/Feature Request
Mailing list - C6EZRun@list.ti.com
C6EzAccel for System Developers
What can a System developer do on the DSP?

- Offload key signal-processing and math intensive algorithms.
- Leverage some DSP features, like floating-point computations.
- Free up ARM MIPS for additional system features.
- Save money by not moving to a more powerful & expensive ARM!

3 Ways to access the DSP from the ARM

- Call DSP directly over DSPLINK/SYLINK.
- Wrap the algorithm with IUniversal interface to Codec Engine.
- Use C6EZRun to cross compile and redirect code for DSP.
C6EZAccel: ARM access to Ready-to-Use DSP functions

Target Developers
- ARM SoC Developers

Challenges
- Unfamiliar with
  - Heterogeneous programming environments
  - Graphical IDEs
  - Inter-processor communication

Strategy: C6EZAccel
- Codec-style build flow
- Enables
  - Freeing MIPS on ARM
  - Pre/Post Processing
- Offloading tasks to DSP
C6EZAccel for System Developers

C6EZAccel provides ARM application access to production ready, performance tuned functions on the DSP.

- ARM side API with easy access to 100’s of optimized DSP kernels
- Asynchronous execution allows ARM/DSP parallel processing
- Includes key signal processing, image processing and math functions

C6EZAccel optimized xdais Algorithms for:
- Audio
- Medical
- Vision
- Power and Control
- Image processing
- Signal processing
Basic Overview of C6EZAccel with the Software Stack
C6EZAccel application

Consists of two parts: DSP server integrating the DSP algorithms and ARM side user interface

DSP server collating DSPLink, CMEM, DSP/BIOS, and DSP libraries

ARM user interface is a static library that abstracts the codec engine setting and ARM side cache management

Function is retargeted to DSP, with error handling managed on ARM/Linux using error codes
Leveraging C6EZAccel

Unit Server
- Provide ARM and DSP software infrastructure to load and interact with C6EZAccel algorithm on the DSP

Add to Codec Server
- Provide ARM and DSP software infrastructure to load and interact with C6EZAccel and A/V Multimedia codecs on the DSP

Add custom function on the DSP
- Run any algorithm on the DSP
C6EZAccel Application Usage

**User Experience in ARM Application**

### Headers and definitions

- `C6accel_Handle hC6accel = NULL`
- `#define ENGINENAME " <platfrom_name>"
- `#define ALGNAME "c6accel "
- `#include "soc/c6accelw/c6accelw "`

### Source code to call fxn1 and fxn2 on DSP

```c
CE_Runtime_init();
hC6accel = C6accel_create( );

Status = C6accel_fxn1(hC6accel,parameters1);
Status = C6accel_fxn2(hC6accel,parameters2);

C6accel_delete(hC6accel);
```

- Creates a C6accel handle.
- Calls fxn1 and fxn2 on the DSP.
- Deletes C6Accel handle.

### Steps

1. **Step 1:** Analyze source to determine what to move to the DSP
2. **Step 2:** Allocate all buffers to be passed to DSP using CMEM
3. **Step 3:** Offload function to the DSP by invoking C6EZAccel APIs or CE APIs
4. **Step 4:** Integrate DSP codec server with application using XDC configuration
5. **Step 5:** Compile all ARM code with the linker command file generated from Step 4
C6EAccel: Key Features

- **Asynchronous Calling**
  - Parallel processing on ARM and DSP
  - Frees ARM MIPS, allows feature addition
  - Maximum system level performance
  - Source to call fxn1 asynchronously on DSP

  ```
  C6Accel_setAsync(hC6);
  C6accel_fxn1(hC6accel,parameters1);
  < ARM code to run in parallel>
  C6accel_waitAsyncCall(hC6accel);
  ```

- **Chaining of APIs**
  - Averaging of Inter processor overhead
  - Improved Efficiency to make multiple calls
Performance benefits of offloading tasks to the DSP

C6EZAccel Overheads:
- Cache invalidation on ARM and the DSP
- Address translation from virtual to physical and vice versa
- DSP/SYSLINK overhead.
- Activating, processing, deactivating the C6EZAccel algorithm on the DSP

DSP provides higher gains on larger images/data

Performance of C6EZAccel vs Native ARM floating point FFT function
# C6EZAccel Supported Processing Blocks

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Now</th>
<th>2Q11</th>
<th>4Q11</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Foundation Signal Processing Software Algorithms</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Digital Signal Processing</td>
<td>32</td>
<td>45</td>
<td>60</td>
</tr>
<tr>
<td>Image Processing</td>
<td>40</td>
<td>50</td>
<td>60</td>
</tr>
<tr>
<td>Floating Point Math</td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Fixed Point Math</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>Filter Package</td>
<td></td>
<td></td>
<td>128</td>
</tr>
<tr>
<td><strong>Application Specific Software Algorithms</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vision And Analytics Library (VLIB)</td>
<td></td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>Open Source Computer Vision (OpenCV)</td>
<td></td>
<td>60</td>
<td>100</td>
</tr>
<tr>
<td>ProAudio: Audio Processing Library</td>
<td></td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Power and Energy</td>
<td></td>
<td></td>
<td>15</td>
</tr>
<tr>
<td><strong>Total Supported Functions</strong></td>
<td></td>
<td>134</td>
<td>269</td>
</tr>
</tbody>
</table>
C6EZAccel – Making DSP Development EZ

Supported Devices
OMAP-L13x, C6A816x, CAA814x, DM814x, DM816x, DM3730, DM6467, OMAP3530,

Availability
Downloadable on product page
Standard part of Software Development Kit

Applicable Links
C6EZTools Wiki – www.ti.com/c6eztoolswiki
C6EZAccel Wiki – www.ti.com/c6ezaccelwiki
TI Product Page – www.ti.com/c6ezaccel

Technical Support
Forums - http://e2e.ti.com/support/default.aspx

Feedback/Feature Request
Mailing list - C6EZAccel@list.ti.com

Supported Devices
OMAP-L13x, C6A816x, CAA814x, DM814x, DM816x, DM3730, DM6467, OMAP3530,

Availability
Downloadable on product page
Standard part of Software Development Kit

Applicable Links
C6EZTools Wiki – www.ti.com/c6eztoolswiki
C6EZAccel Wiki – www.ti.com/c6ezaccelwiki
TI Product Page – www.ti.com/c6ezaccel

Technical Support
Forums - http://e2e.ti.com/support/default.aspx

Feedback/Feature Request
Mailing list - C6EZAccel@list.ti.com
C6EZRun versus C6EZAccel

<table>
<thead>
<tr>
<th>C6EZRun</th>
<th>C6EZAccel</th>
</tr>
</thead>
<tbody>
<tr>
<td>• For ARM/Linux developer on DSP+ARM devices</td>
<td>• For users with some knowledge of DSP development</td>
</tr>
<tr>
<td>• For users with little to no knowledge of DSP development</td>
<td>• For users who want to use TI’s prebuilt, optimized DSP libraries</td>
</tr>
<tr>
<td>• For users who are bringing their own C source code</td>
<td>• For users who want to use other packaged components on the DSP</td>
</tr>
<tr>
<td>• For users who don’t need compatibility with other algorithms, codecs, etc.</td>
<td></td>
</tr>
</tbody>
</table>
Thank you!