Aufsetzen und Debuggen eines CCSv5.5-Projektes für D.module.C6713 DSP mit CODEC D.module.ADDA16

1 Programmstart CCSv5 und Erstellung eines Projektes

Starten Sie CCS 5 durch Doppelklick auf das zugehörige Icon auf dem Desktop.

Wenn beim Start nicht die CCS Edit-Ansicht (vgl. Abb.1) geöffnet ist, öffnen Sie diese über ***Window → Open Perspective → Other…*** und Doppelklick auf ***CCS Edit*** (Abb. 2).

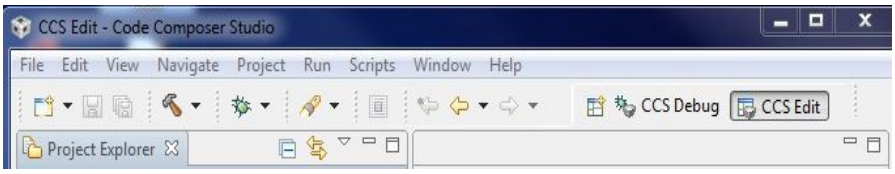

Abb. 1: CCS Edit - Ansicht

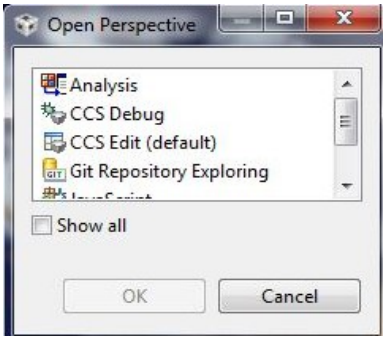

Abb. 2: Dialog zum Öffnen von Ansichten

Erstellen Sie ein neues Projekt in *D:\ti_work5* über ***Project → New CCS Project***. Übernehmen Sie dazu die Einstellungen aus Abb. 3, passen Sie den Projektnamen für Ihr Projekt sinnvoll an, hier *First-project_dmod6713_adda16* und klicken Sie am Ende auf *Finish*.

Der gleichnamige Ordner *First-project_dmod6713_adda16* wird dabei automatisch erstellt. Drei mit einem vorangestellten Punkt benannte Projektdateien in diesem Verzeichnis speichern alle Projekt- und Buildinformationen im ASCII-Format (vergleichbar mit Windows INI-Dateien).

Kopieren Sie folgende Dateien aus
D:\ti_work5\dmod-c6713_dmod-adda16\...
…\Lab_support_dmod-c6713_dmod-adda16
in Ihr Projektverzeichnis:

1. ***dmod-c6713_dmod-adda16_test.c (C-Source Code)***
2. ***dmod-c6713_dmod-adda16.cmd (Link command file)***
3. ***send_str.c***
4. ***adda16.h***
5. ***bios.h***

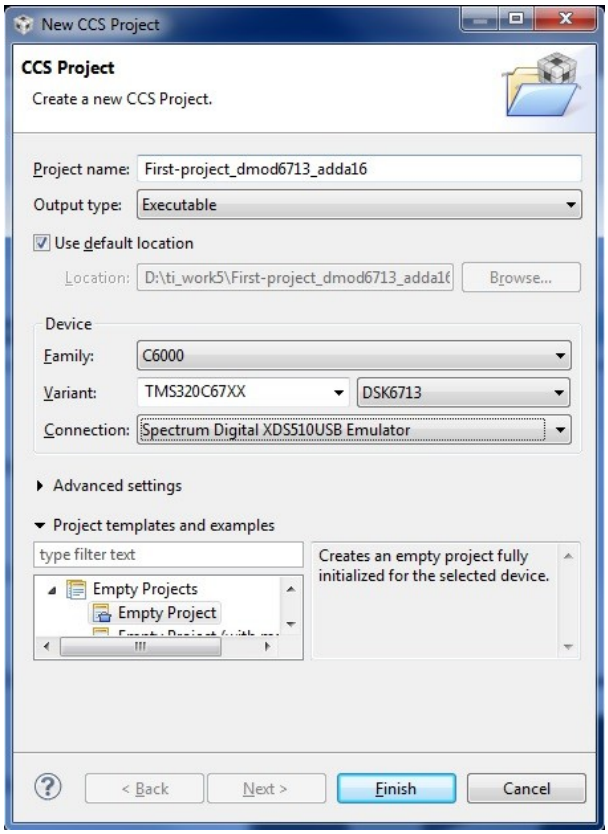

Abb. 3: Create a new CCS Project

Spätestens nach einem Aktualisieren der Ansicht mit <F5> sehen Sie die Dateien bereits in Ihrem CCS Projekt Explorer.

Nun müssen noch zwei **Build Options** eingestellt werden, dazu rechter Mouseclick in der CCS Edit Perspective auf das Projekt, *Properties* wählen, unter *Build → C6000 Compiler → Advanced Options → Runtime Model Options* als „Data access model“ auswählen: „far“ (siehe Abb.4 a).

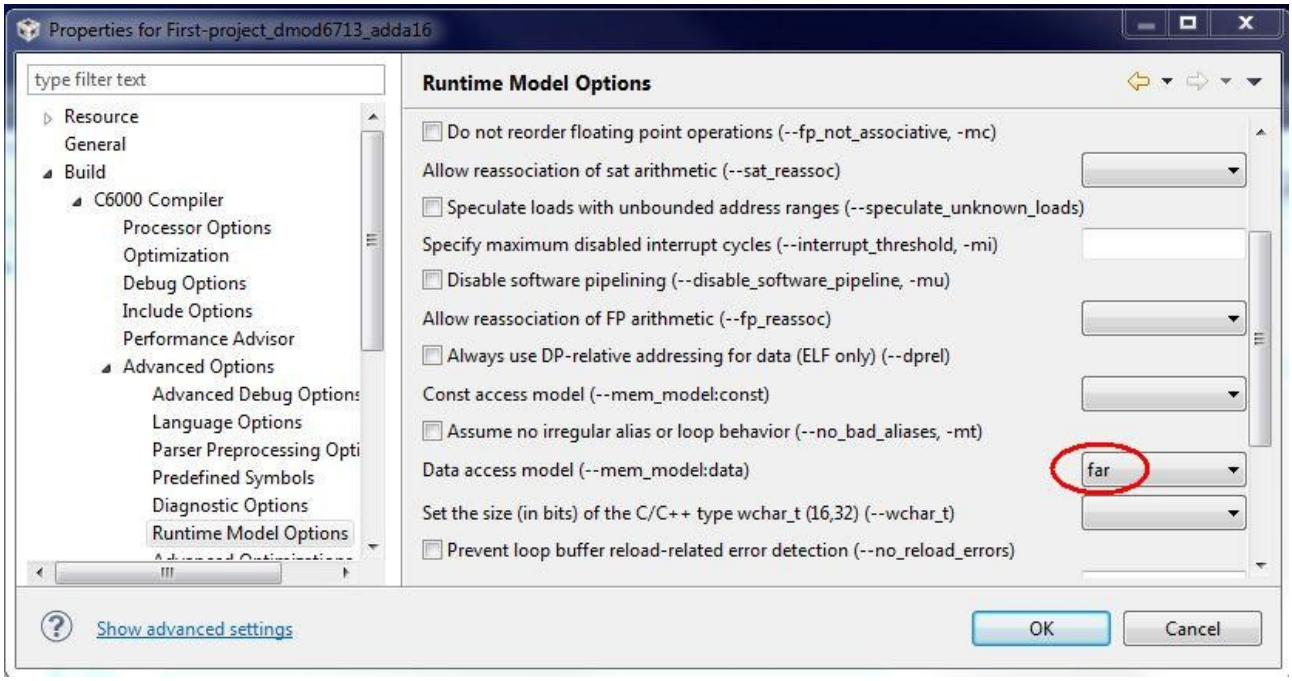

Abb. 4 a: Build Options

Als Zweites muss die richtige Prozessor-Version eingetragen werden. Tragen Sie dazu unter *Build → C6000 Compiler → Processor Options* in dem Feld neben *Target processor version* ein: „6710“ (siehe Abb. 4 b).

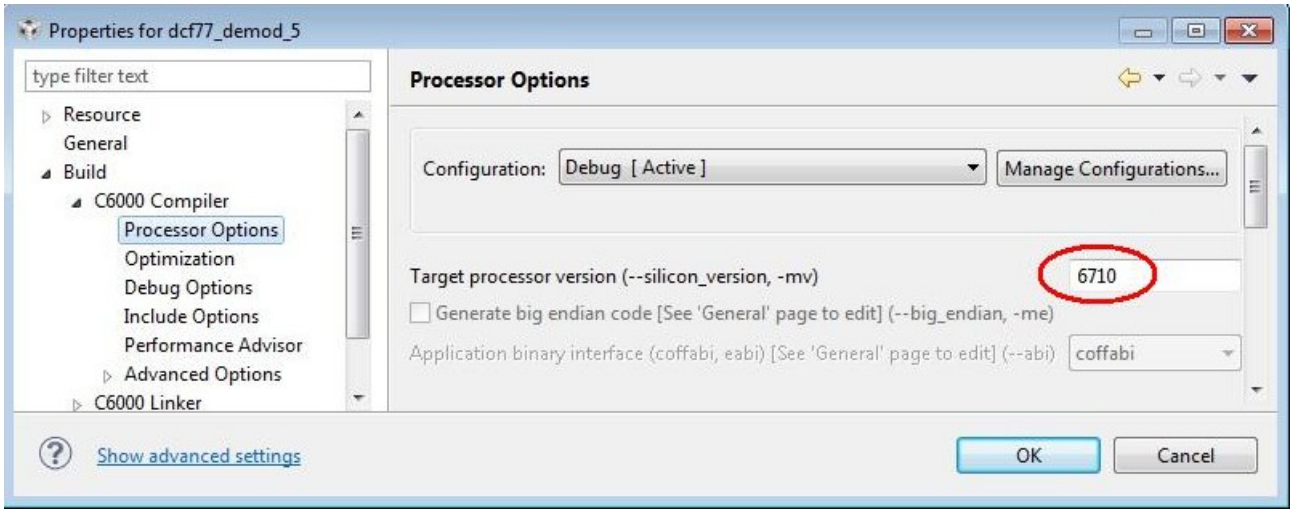

Abb. 4 b: Build Options

2 Kompilieren, Linken und Laden des Projektes

Schließen Sie den Aufbau ***D.SignT – DSP6713 + ADDA16*** per USB über den unter dem Board befestigten ***JTAG Adapter XDS510Plus*** an.
Rufen Sie nun ***Run → Debug*** (bzw. **<F11>**) auf.
Auf diesen Befehl hin

- werden die eingebundenen Dateien **kompiliert**,
- **gelinkt**, wobei der Linker die Datei *dmod-c6713_dmod-adda16.cmd* auswertet
- und eine **ausführbare Datei wird erstellt**.
- Die Zielhardware wird initialisiert und **verbunden**,
- die ausführbare Datei ..\ *First-project_dmod6713_adda16\Debug\ First-project_dmod6713_adda16.out* wird auf das Board **geladen.**
- Das Programm wird **gestartet** und **hält hinter der Zeile `main()`** an.

Das Fenster „Problems“ sollte keine Fehler oder Warnungen enthalten bzw. noch überhaupt nicht geöffnet worden sein.

Sie befinden sich jetzt automatisch in der Debug-Ansicht (Debug Perspective, in Abb. 5 rot umkreist). In die ursprüngliche CCS Edit-Ansicht kommen Sie über die Schaltflächen rechts daneben.

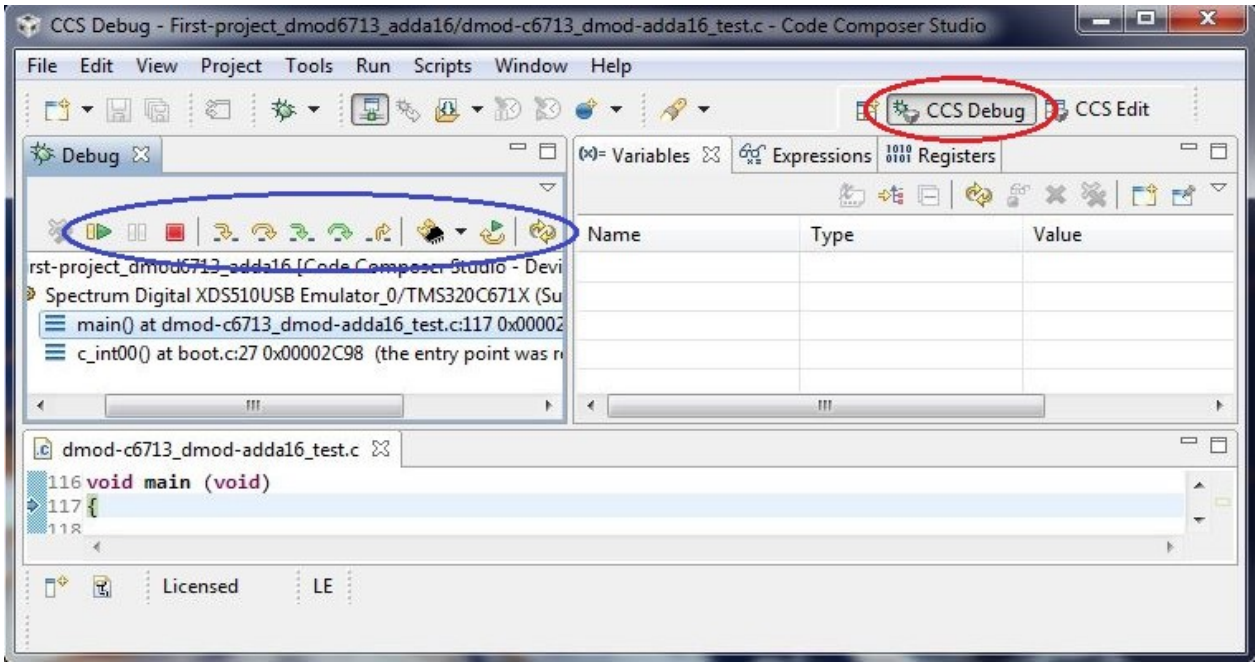

Abb. 5: Ansicht nach Debug → Run

Mit ***Run → Resume*** oder **<F8>** oder dem entsprechenden Button (Abb. 5, grüner Pfleil links in blau umkreistem Bereich) lassen Sie das Programm nun laufen. Ausgaben erscheinen im Fenster „Console“.
Über *View → Debug* öffnen Sie das Fenster mit den Steuerelementen (Run, Resume, Stop, Single Steps) wieder, wenn Sie es geschlossen haben.

3 Beispielprogramm dmod-c6713_dmod-adda16_test.c mit Signal-Ein- und ausgabe

Mit Hilfe eines einfachen Demoprogramms *dmod-c6713_dmod-adda16_test.c* wird der DSP so programmiert, dass ein über den Audio-CODEC eingelesenes Audiosignal direkt wieder ausgegeben wird.

Die max. Abtastrate des CODECs ADDA16 beträgt 500 kHz. Es können diverse Abtastraten eingestellt werden.

Externer Takt: Mit einem Generator kann jeder beliebige (externe) Takt eingestellt werden, dazu wird ein Rechtecksignal (0..3,3 V) auf den Eingang EXT_CLK gegeben. FSAMP muss im Programm auf 0 gesetzt werden über:
ADDA16->fs = 0; statt ADDA16->fs = ADDA16_FS(FSAMP);

```
/* common sampling frequencies */
// added by US fs=4 MHz/(REG_WERT+1) (ADDA 16 spec. pp. 9)
// Bsp: fs = 500 kHz
// REG_WERT=7, denn 4e6/(7+1) = 500 kHz !!
#define ADDA16_FS500    0x0007          #define ADDA16_FS250    0x000F
#define ADDA16_FS200    0x0013          #define ADDA16_FS160    0x0018
#define ADDA16_FS120    0x001F          #define ADDA16_FS100    0x0027
#define ADDA16_FS80     0x0031          #define ADDA16_FS67     0x003B
#define ADDA16_FS50     0x004F          #define ADDA16_FS33     0x0077
#define ADDA16_FS25     0x009F          #define ADDA16_FS20     0x00C7
#define ADDA16_FS16     0x00F9          #define ADDA16_FSEXT    0x0000

/* this macro can be used for any integer sampling frequency from 15625 Hz to 250 kHz */
#define ADDA16_FS(x)    ((4000000/(int)x)-1)
```

Die Kanäle werden einzeln vom ADC gelesen und können in 16-bit `short int` Variablen gespeichert werden:

```
inL = (ADDA16_READ_ADC(0));
inR = (ADDA16_READ_ADC(1));
```

Das Schreiben dieser Variablen zum DAC erfolgt über folgende Zeilen:

```
// Abfrage beim ersten DAC nicht nötig:
//   while (!(ADDA16->cfg & ADDA16_DAC_READY) );

    ADDA16_WRITE_DAC(0, inL);
    while (!(ADDA16->cfg & ADDA16_DAC_READY) );
        ADDA16_WRITE_DAC(1, inR);
```

4 Der CCS Debugger

Mit ***Run → Restart*** wird das Programm erneut gestartet und hält vor `main()` an.
Run → Resume führt das Programm aus, mit den Single Step-Funktionen wird das Programm schrittweise ausgeführt.
Mit ***Run → Step Over*** bzw. <F6> können Sie das Programm zeilenweise ausführen lassen.
Die einzelnen Funktionen des ***Run*** Menüs sind in Abbildung 6 aufgelistet.

Abb. 6: Debug–Funktionen (im Menü Run)

Mit einem Restart werden die internen DSP-Register nicht verändert. Möchten Sie auch die internen CPU- und Peripherie-Register reinitialisieren, führen Sie zunächst (vor erneutem Laden über *Run → Debug*) ***Run →Reset→Reset CPU*** aus.

CCS bietet weitere Debug-Möglichkeiten, die hier Schritt für Schritt erklärt werden. Unter anderem können Sie:

- Breakpoints im Sourcecode setzen und wieder entfernen
- das Programm in Einzelschritten ausführen
- sich gemischt Source- und Assembler-Code anzeigen lassen
- Variablen überwachen
- den Inhalt der DSP-Register (DSP-core registers, peripheral registers) anzeigen
- den Inhalt beliebiger Speicherplätze anzeigen.

Setzen Sie einen Breakpoint vor die Zeile, vor der Sie das Programm anhalten möchten, indem Sie den Mousezeiger vor der Zeilenummer platzieren und doppelklicken (alternativ: Zeile markieren, rechte Mousetaste, *Breakpoint→Breakpoint*). Ein blauer Punkt zeigt den gesetzten Breakpoint an. In Abb. 7 ist das Programm bis zum Breakpoint gelaufen. Die Anzeige aller Breakpoints öffnet sich mit ***View → Breakpoints.***

Mit der rechten Mousetaste auf den Breakpoint, dann *Breakpoint Properties* öffnet ein neues Fenster. Unter ***Action*** können Sie nun festlegen, ob das Programm beim Haltepunkt stehen bleiben soll (*Remain Halted*) oder z.B. über *Update View* oder *Refresh All Windows* immer wieder bis zu diesem Punkt laufen und Fenster aktualisieren soll. Diese Anwendung ist bei Verwendung graphischer Displays nützlich!

Sie können nun eine beliebige Anzahl an Breakpoints in jedem C- oder Assembler-Code setzen. Sie erinnern sich: die aktuelle Position ist mit dem blauen Pfeil markiert. So können Sie das Programm stückweise ausführen. Einzelne Breakpoints können durch Doppelklicken des blauen Punktes gelöscht werden; *Toggle Breakpoint* im Menü, das die rechte Mousetaste öffnet, setzt oder löscht den Breakpoint, *Disable/Enable Breakpoint* in dem Menü vereinfacht es, immer dieselbe Zeile für einen Breakpoint zu wählen.

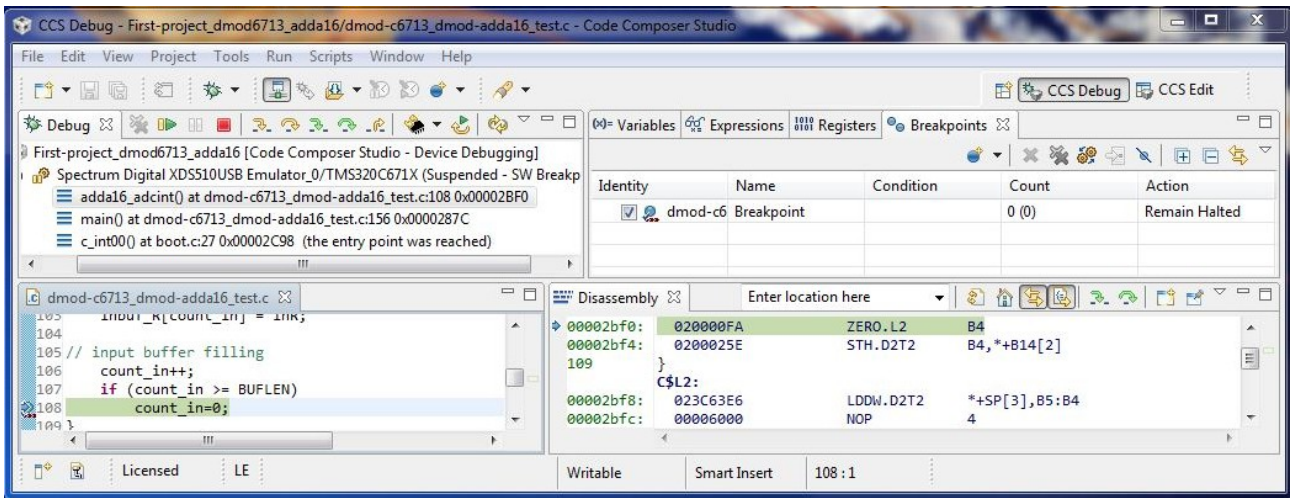

Abb. 7: Halt am Breakpoint

Sie sehen außerdem in Abb. 7 im rechten Bereich unten das Disassembly-Fenster (***View → Disassembly***). Darin können Sie die einzelnen Schritte beim Assembly Single Step Befehl überwachen. Ein oder mehrere (bei paralleler Bearbeitung mehrerer Statements) blaue Pfeile zeigen auch hier die aktuelle Position des program counter.

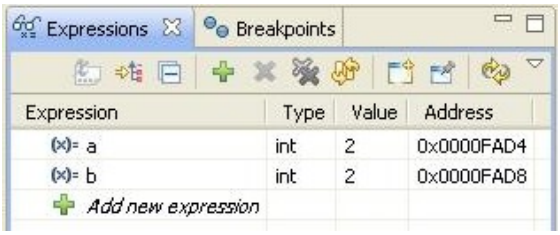

Abb. 8: Variablen überwachen

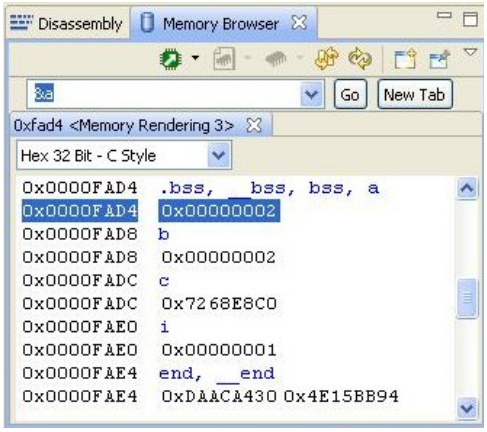

Abb. 9: Speicher ansehen im Memory Browser

Überwachen Sie eine Variable, indem Sie diese markieren, rechter Mouseklick, <*Add Watch Expression…*>. Alternativ über *View → Expressions*, dort hinzufügen, vgl. Abb. 8.
Sie können sich den Inhalt von Variablen auch anzeigen lassen, ohne diese ins Expressions-Window zu übernehmen: Gehen Sie einfach mit dem Mousezeiger über die Variable, ihr Inhalt (bzw. die Startadresse, falls es sich um eine Funktion handelt) erscheint auf dem Bildschirm.

Speicherinhalte können Sie sich über ***View → Memory Browser*** (vgl. Abb. 9) ansehen. Im sich öffnenden **Memory Browser** tragen Sie dazu im Adressfeld die Adresse ein, in Abbildung 8 z.B. „&a“, klicken auf <Go> und wählen das Datenformat für die Anzeige aus, hier „Hex 32 Bit – C Style“.

Die **Register** können mit ***View → Registers*** überwacht werden, das ein neues Fenster öffnet.

Innerhalb dieses Fensters kann unter **Core Registers (interne CPU-Register)** der Inhalt der 32 A und B Register und des „program counters“ PC überwacht und verändert werden.
Aktuell veränderte Registerinhalte sind hervorgehoben, die Inhalte können über Klick auf den Registerinhalt geändert werden.

In diesem Fenster haben Sie u. a. ebenfalls Zugriff auf die **Peripherie Register** (für Timer, Interrupt und EMIF) sowie die Register der seriellen Schnittstelle McBSP, über die der CODEC angesteuert wird.

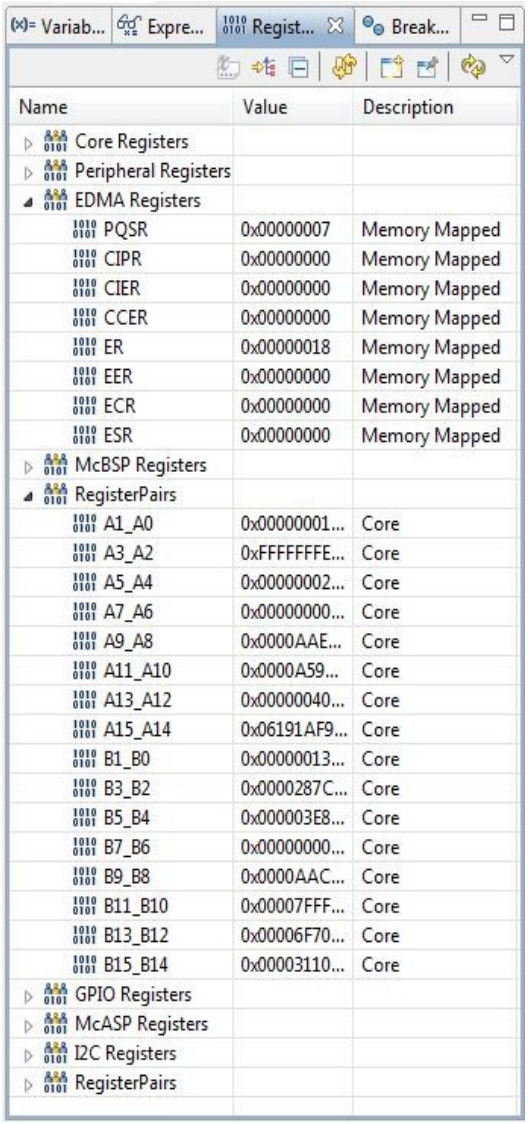

Abb.10: View → Registers

Hinweis zur Anordnung der Fenster in CCS:
Ein Klick mit der rechten Mousetaste auf den Reiter z.B. des Registerfenster und die Auswahl von *Detached* entkoppelt ein derartiges Fenster, so dass es beliebig verschoben und angeordnet werden kann, das Entfernen des Häkchens vor *Detached* koppelt das Fenster wieder in die CCS Oberfläche ein.
Dort kann es mit der Mouse ganz einfach in andere Bereiche gezogen werden.