Signalverarbeitungs-Praktikum: DSP – PC Kommunikation mit dem Texas Instruments TMS320C6x EVM*

Ralf Herz, Hannes Hartenstein
Institut für Informatik
Albert-Ludwigs-Universität Freiburg
Am Flughafen 17
79110 Freiburg, Germany

Juni 1999

Zusammenfassung

Dieser Artikel beschreibt die grundlegende Funktionsweise der Kommunikation zwischen einem Texas Instruments TMS320C6201 DSP EVM und einem Personal Computer unter Windows NT. Als Beispiel dazu hierzu dient die Implementierung eines Spektrum-Analysators, der Tonsignale über den Audioport des EVM aufnimmt und diese Fourier-transformiert; die Transformierte wird dann zur Visualisierung an den PC übertragen. Außerdem wird ein Sprachkodierer nach dem G.723-Standard auf den DSP portiert.

1 TMS320C6201 Evaluation Module

Das Texas Instruments TMS320C6201 EVM (Evaluation Module) ist eine PCI-Steckkarte, auf welcher neben dem TMS320C6201 DSP noch verschiedene andere Peripherieeinheiten untergebracht sind. Die Karte kann sowohl zur Entwicklung von Host-PC unterstützenden Routinen als auch für eigenständige DSP-Anwendungen verwendet werden. Hier wird nur der Einsatz in Verbindung mit einem PC betrachtet.

*Technical Report 123, Institut für Informatik, Universität Freiburg, Juni 1999. Diese Arbeit wurde unterstützt durch das BMBF Mathematik-Programm im Rahmen des Projekts "Adaptive Verfahren zur Bild- und Bildsequenzkodierung".

1.1 DSP TMS320C6201

Der Digitale Signalprozessor (DSP) TMS320C6201 ist eines der jüngsten Mitglieder der Texas Instruments DSP-Familie. Die 32-bit Architektur ist speziell auf die Echtzeit-Signalverarbeitung ausgelegt. Durch seine hohe Rechenleistung von maximal 1600 MIPS bei 200 MHz eignet er sich auch für sehr komplexe Anwendungen. Die sehr hohe Rechenleistung, die theoretisch möglich ist, erreicht der DSP durch acht integrierte funktionelle Einheiten, durch die je Prozessortakt acht Operationen gleichzeitig ausgeführt werden können. Die Rechenleistung kann in der Regel in der Praxis nicht voll ausgenutzt werden, da in jeder Funktionseinheit nur bestimmte Operationen ausgeführt werden können und sich nicht jeder Algorithmus optimal an diese Struktur anpassen läßt.
Der TMS320C6201 ist als Fixed-Point DSP in der Lage, 8-, 16- und 32-bit Integer-Daten zu verarbeiten. Innerhalb der Recheneinheiten kann bei einigen Operationen mit 40-bit Genauigkeit gerechnet werden. Eine Einführung in das Thema Fixed-Point-Berechnungen findet man in [4]. Zur Verarbeitung von Fließkommazahlen kann eine Software-Emulation [2] eingesetzt werden, welche die üblichen Operationen für die Datentypen `float` und `double` enthält.
Die Taktfrequenz bei dem von uns verwendeten DSP/EVM liegt bei maximal 166 MHz. Ein zum TMS320C6201 kompatibler Prozessor mit zusätzlicher in Hardware implementierter Fließkommazahlen-Verarbeitung ist unter der Bezeichnung TMS320C6701 verfügbar.

1.2 Peripherie

Das EVM verfügt bereits über verschiedene Peripheriekomponenten, zu denen sich über einen flexiblen Erweiterungsbus anwendungsspezifische Komponenten wie Speichermodule, I/O-Einheiten oder auch weitere DSPs hinzufügen lassen. Die folgende Liste gibt einen kurzen Überblick über die vorhandenen Komponenten.

- **Speicher:**

  Das EVM beinhaltet 256 KB sehr schnelles SBSRAM, sowie zwei Speicherbänke mit jeweils 4 MB SDRAM. Außerdem kann weiterer externer Speicher über das *Expansion Memory Interface (EMIF)* hinzugefügt werden.

- **PCI Interface:**

  Diese Schnittstelle wird dazu verwendet, den DSP vom PC aus über das *Host Port Interface (HPI)* zu steuern sowie Datentransfers zum und vom Host-Speicher durchzuführen. Über den DMA-Controller kann sowohl der PC auf den EVM-Speicher als auch der DSP auf den Host-Speicher direkt zugreifen.

- **Audio Interface:**

  Die Audioeinheit kann 16-bit Samples bei Raten von 5.5 kHz bis 48 kHz in Stereo verarbeiten. Es stehen ein Mikrofon- und ein Line-Eingang sowie ein Line-Ausgang zur Verfügung.

- **Interruptverwaltung:**

  Die Peripherie hat 16 verschiedene Interruptquellen. Da der DSP nur 12 Interruptleitungen zur Verfügung hat, kann man mit dem Interrupt Selector diejenigen Interrupts auswählen, welche in der Anwendung benötigt werden.

- **Universal-Timer:**

  Die zwei Timer des DSP können als Zeitmesser, Taktgeneratoren, Ereigniszähler oder Interruptquellen verwendet werden.

- **Status LED:**

  Es steht eine extern am Slotblech und eine intern auf der Platine angebrachte LED zur freien Verfügung. Damit können beispielsweise bestimmte Betriebszustände signalisiert werden.

1.3 WWW-Resourcen

Auf den Webseiten von Texas Instruments erhält man viele Informationen über die TMS320C6x-Familie. Eine Einstiegsseite hierfür ist:
`http://www.ti.com/sc/c6x.`
Den Supportbereich mit Online-Handbüchern, Neuigkeiten und vielen weiteren Informationen erreicht man direkt unter:
`http://www.ti.com/sc/docs/dsps/hotline/wizsup62xx.htm.`

1.4 Programmierung des DSP in C

Natürlich besteht die Möglichkeit, den DSP nativ in Assembler zu programmieren. Allerdings ist zur effizienten Nutzung der Möglichkeiten des DSP eine genaue Kenntnis der komplexen internen Strukturen nötig, die eine simultane Ausführung von bis zu 8 Operationen erlauben. Einfacher erfolgt die Entwicklung mit dem C-Compiler von Texas Instruments, da dieser den

Programmcode speziell für die C6x-Familie optimiert und dem Entwickler die Flexibilität der verbreiteten Programmiersprache C bietet.

1.4.1 Compilierung

Die Compilierung erfolgt mit dem Cross-Compiler `cl6x` auf der PC-Plattform. Der Compiler ist ein Kommandozeilen-Programm, dessen Syntax sehr ähnlich zu der von C-Compilern unter UNIX ist. Neben den üblichen Compileroptionen wie `-g` zur Debugsymbolerzeugung und `-o2|3` zur Optimierung stehen noch weitere Optionen zur Nutzung der speziellen Eigenschaften des DSP zur Verfügung. Diese werden im Online-Handbuch des C-Compilers genauer beschrieben. Interessant zur Analyse des durch den Compiler erzeugten Assembler-Codes sind die Optionen `-k` und `-ss`. Hierdurch wird der erzeugte Assembler-Code in eine Datei ausgegeben und die zugrundeliegenden C-Codezeilen werden als Kommentare an den entsprechenden Stellen eingefügt. Das anschließend notwendige Linken der einzelnen Programmteile und der DSP Runtime Bibliothek zum fertigen DSP-Programm kann direkt beim Compilieren durch den Parameter `-z` und anschließende Link-Optionen erfolgen oder durch expliziten Aufruf des Linkers `lnk6x`.
Im Vergleich zu gewöhnlichen Computern unterscheidet sich die Speicherverwaltung von DSPs in einigen Punkten. Bei fast allen Computern wird der gesamte Speicher als ein großer, zusammenhängender Speicherbereich für Programm und Daten betrachtet, der gelesen und beschrieben werden kann. Im Gegensatz dazu sind bei den meisten DSPs getrennte Speicherbereiche für Programm, Konstanten und variable Daten vorhanden. Das Programm und die Konstanten werden in einem ROM abgelegt, während die variablen Daten im RAM gespeichert werden. Zusätzlich zu diesen Bereichen kommt beim TMS320C6201 EVM noch eine Anzahl von Memory-Mapped I/O Registern. Diese verschiedenen Bereiche werden durch zusätzliche Logikbausteine so angeordnet, daß sie für den DSP in festgelegten Adressbereichen, innerhalb des virtuellen 32-bit Adressraumes, ansprechbar sind.
Das benutzte EVM besitzt zwei verschiedene Memory-Mappings, zwischen denen man wählen kann. Beim Linken muß man darauf achten, welches der beiden möglichen Memory-Mappings man verwendet. Die Zuordnung von Programmcode, Variablen und I/O-Ports zu physikalischen Adressen erfolgt mit einer Linker-Steuerungsdatei, welche beim Linken angegeben wird (siehe Abbildung 1). Der erste Teil der Datei definiert die physikalisch vorhandenen Speicherbereiche und ihre Position im Adressraum. Die zweite Hälfte definiert die Aufteilung der bei der Compilierung oder Assemblierung erzeugten logischen Segmente in die vorhandenen Speicherbereiche.
Die zwei Bereiche `INT_PROG_MEM` und `SBSRAM_PROG_MEM` stehen für den Pro-

```
/* linker command file for MAP 1 */

MEMORY
{
  INT_PROG_MEM (RX)    : origin = 0x00000000 length = 0x00010000
  SBSRAM_PROG_MEM (RX) : origin = 0x00400000 length = 0x00020000
  SBSRAM_DATA_MEM (RW) : origin = 0x00420000 length = 0x00020000
  SDRAM0_DATA_MEM (RW) : origin = 0x02000000 length = 0x00400000
  SDRAM1_DATA_MEM (RW) : origin = 0x03000000 length = 0x00400000
  INT_DATA_MEM (RW)    : origin = 0x80000000 length = 0x00010000
}

SECTIONS
{
  .vec:          load = 0x00000000
  .text:         load = SBSRAM_PROG_MEM
  .const:        load = INT_DATA_MEM
  .bss:          load = INT_DATA_MEM
  .data:         load = INT_DATA_MEM
  .cinit         load = INT_DATA_MEM
  .pinit         load = INT_DATA_MEM
  .stack         load = INT_DATA_MEM
  .far           load = INT_DATA_MEM
  .sysmem        load = SDRAM0_DATA_MEM
  .cio           load = INT_DATA_MEM
}
```

Abbildung 1: Die für den Spektrum-Analysator verwendete Linker-Steuerungsdatei

grammcode zur Verfügung. Für die Daten stehen die restlichen vier Bereiche bereit. Diese unterscheiden sich in der Größe und Zugriffsgeschwindigkeit. Häufig benötigte Variablen und kleine Lookup-Tabellen legt man deshalb im schnellen internen Speicher ab, während man größeren Strukturen (Datenpuffer, etc.) einen langsameren Speicher zuordnet.
Die Einstellung, mit der das Programm gelinkt wurde, muß auch später bei der Programmausführung und im Debugger verwendet werden. Da ein Großteil der Peripherie zum Datenaustausch Memory-Mapped I/O verwendet, muß bei Zugriff auf diese Speicherbereiche bereits dem Compiler bekannt sein, welches Mapping verwendet wird.

1.4.2 Programmausführung

Der Linker erzeugt Dateien mit der Endung `.out`, welche den nativen Programmcode enthalten. Zur Ausführung auf dem DSP muß dieser über das PCI-Interface in den DSP-Speicher kopiert werden. Dazu hat man drei verschiedene Möglichkeiten:

- **Programmladeroutine:**

  Mit dem Programm `evm6xldr` kann ein DSP-Programm auf das EVM geladen und dort gestartet werden. Die Laderoutine kann anschließend vom DSP zum Laden von Daten aus einer Eingabedatei und Speichern in einer Ausgabedatei genutzt werden.

  Der Software-Simulator `load6x` kann man verwenden um den Zeitbedarf kleinerer Codeteile zu messen. Das EVM wird dann auf dem PC emuliert und der Programmcode in dieser Umgebung ausgeführt.

- **Debugger:**

  Mit dem graphischen Windows-Debugger `evm6x` kann das Programm auf das EVM geladen werden und danach schrittweise unter Kontrolle des Debuggers ausgeführt werden. Die Programmverfolgung kann im Sourcecode (mit eingeschalteter Debugsymbolgenerierung) sowie dem nativen Assemblercode erfolgen. Es können Prozessorregister, Variablen und Speicherbereiche überwacht werden. Außerdem ist ein Profiler zur Ermittlung der Laufzeit der einzelnen Routinen enthalten.

  Beim Programmaufruf ist zu beachten, daß man als Parameter das Board angeben muß. Für das erste Board im System lautet der Aufruf `evm6x -n cpu_a`. Außerdem muß auch dem Debugger das verwendete Memory-Mapping mitgeteilt werden, da der Zugriff auf nicht vorhandene Speicherbereiche oder der Versuch schreibgeschützte Register zu beschreiben ein undefiniertes Verhalten zur Folge hat.

- **Host-Programm:**

  Die Host-PC Support Software stellt Routinen zur Verfügung, um das EVM aus eigenen Anwendungen heraus zu steuern. Dabei wird normalerweise zuerst eine Verbindung zum EVM-Board aufgebaut und noch auf dem DSP laufende Programme werden angehalten. Dann wird der neue Programmcode überspielt und gestartet. Anschließend können mit auf DSP- und Hostseite vorhandenen Routinen Daten ausgetauscht werden.

2 Beispiel 1: Spektrum-Analysator

Der implementierte Spektrum-Analysator dient zur Visualisierung der Kurzzeit Frequenzspektren von Audiosignalen. Hierzu wird ein analoges Audiosignal durch den auf dem EVM vorhandenen Audio-Codec digitalisiert und in einem Ping-Pong-Puffer im DSP-Speicher abgelegt. Das Ping-Pong- oder auch Doppelpuffer-Schema ermöglicht dabei eine unterbrechungsfreie Audioaufnahme, indem die aktuellen Daten in einem von zwei identischen Puffern gesammelt werden, während die Daten des jeweils anderen Puffers verarbeitet werden können. Ist der aktuelle Aufnahmepuffer voll, so wird die Aufnahme im anderen Puffer ohne Unterbrechung fortgesetzt und eine Nachricht erzeugt, die dem Programm mitteilt, daß wieder neue Daten vorhanden sind. Die Bearbeitung der Daten muß natürlich jeweils vor der Umschaltung der Aufnahme auf den Puffer mit den gerade bearbeiteten Daten abgeschlossen sein.
Die Kurzzeit-Spektren werden auf Frames von $N = 256$ Samples berechnet, was bei einer Abtastfrequenz von 8 kHz einer Zeit von 32 $msec$ entspricht. Jeder Frame wird zuerst durch ein Hamming-Fenster skaliert. Je zwei Frames werden als reelle und imaginäre Teile eines komplexen Signals aufgefaßt, mit einer komplexen Fourier-Transformation transformiert und anschließend zu zwei Spektren separiert. Diese werden dann zusammen mit den zugrundeliegenden Originalsamples an den PC übertragen.
Die PC-Anwendung besteht aus einem Dialogfenster, welches hauptsächlich aus den Bereichen für die graphische Anzeige des Spektrums und der Originalsamples aufgebaut ist. Dazu kommt ein Regler zur Skalierung der Spektrenanzeige und Kontrollelemente zum Abrufen der letzten 3.2 sec (100 Spektren je 256 Samples).

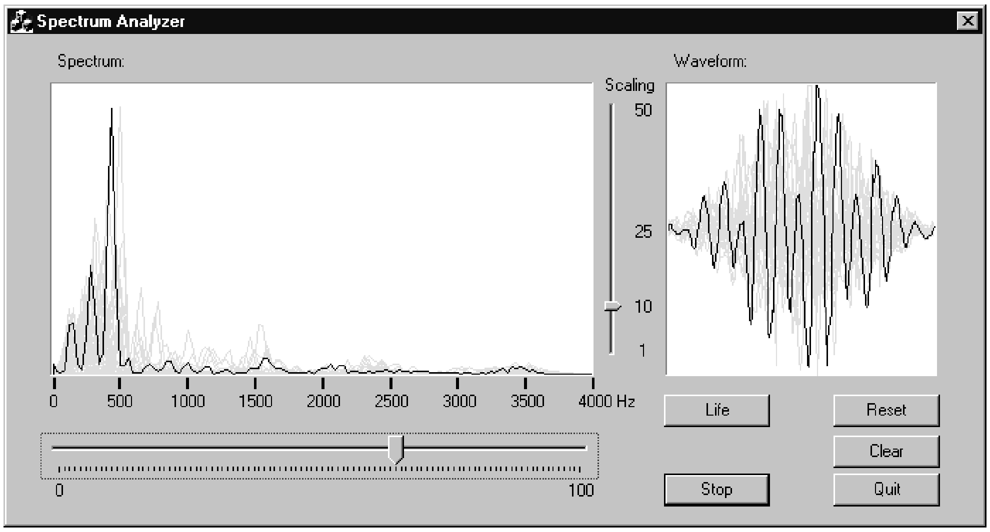

Abbildung 2: Screenshot des Spektrum-Analysators

2.1 DSP-Seite

In diesem Abschnitt wird der Programmablauf auf dem DSP und die verwendeten Routinen erläutert. Eine Beschreibung der Funktion und Parameter der einzelnen Routinen findet man in [3].
Im Programmcode für den DSP wird mit folgenden Routinen das EVM und ein *Multichannel Buffered Serial Port (McBSP)* zur Nutzung der Audiokomponenten initialisiert:

```
evm_init();
mcbsp_drv_init();
```

Danach folgt die Reservierung des Speichers für die verschiedenen Puffer mit der `malloc` Routine. Als Speicherbereich hierfür wird der System-Heap verwendet, welcher in der Linker-Steuerungsdatei durch `.sysmem` bezeichnet wird.
Anschließend werden die Audio-Schnittstellen-Parameter eingestellt und als Audioquelle der Line-Eingang gewählt:

```
memset(&mcbspConfig,0,sizeof(mcbspConfig));
mcbspConfig.loopback                 = FALSE;
mcbspConfig.tx.update                = FALSE;
mcbspConfig.rx.update                = TRUE;
mcbspConfig.rx.clock_polarity        = CLKR_POL_FALLING;
```

```
mcbspConfig.rx.frame_sync_polarity= FSYNC_POL_HIGH;
mcbspConfig.rx.clock_mode         = CLK_MODE_EXT;
mcbspConfig.rx.frame_sync_mode    = FSYNC_MODE_EXT;
mcbspConfig.rx.phase_mode         = SINGLE_PHASE;
mcbspConfig.rx.frame_length1      = 0;
mcbspConfig.rx.word_length1       = WORD_LENGTH_32;
mcbspConfig.rx.frame_ignore       = FRAME_IGNORE;
mcbspConfig.rx.data_delay         = DATA_DELAY0;
mcbsp_config(dev,&mcbspConfig);

codec_init();
codec_change_sample_rate(sampleRate, TRUE);
codec_adc_control(LEFT,0.0,FALSE,LINE_SEL);
codec_adc_control(RIGHT,0.0,FALSE,LINE_SEL);
```

Es folgt die Initialisierung der PCI-FIFO-Schnittstelle

```
pci_driver_init();
pciFifoDev = pci_fifo_open();
```

und die Berechnung der in der FFT und für das Hamming-Fenster benötigten konstanten Koeffizienten.
Zuletzt wird die kontinuierliche, asynchrone Aufnahme gestartet. Die Routine `mcbsp_cont_async_receive` veranlaßt den DMA-Controller des EVM dazu, jeweils 512 Samples in den freien Ping-Pong-Puffer zu schreiben und im Anschluß daran immer eine Callback-Funktion aufzurufen:

```
status= mcbsp_cont_async_receive(dev, pBuffer1, pBuffer2,
                  numBytes, FALSE, NULL, AsyncRxCallback);
```

In der Callback-Funktion wird dann für jeweils zwei Frames die Skalierung mit dem Hamming-Fenster durchgeführt, beide Frames gleichzeitig fouriertransformiert und anschließend das Resultat zusammen mit den Originalsamples durch den FIFO-Mechanismus der PCI-Schnittstelle zum PC gesendet:

```
pci_fifo_async_send(pciFifoDev, SENDbuff, numBytes,
                    PciAsyncTxCallback);
```

2.1.1 FFT-Routine

Die Skalierung mit dem Hamming-Fenster dient der Reduzierung von Störungen im Frequenzspektrum. Sie erfolgt durch Multiplikation jedes der $N = 256$ Samples mit dem entsprechenden Koeffizienten des Hamming-Fensters. Dabei wurden folgenden Koeffizienten benutzt:

$$w(k) = 0.54 - 0.46 \cos\left(\frac{2\pi k}{N-1}\right) \quad 0 \leq k < N. \tag{1}$$

Die verwendete Routine zur schnellen Fouriertransformation von komplexwertigen Signalen (jeweils 16 Bit für den reellen und den imaginären Teil) wurde von Texas Instruments als Benchmark veröffentlicht. Sie ist speziell für die C6x-Familie optimiert. Da der Algorithmus die Daten auf jeder Ebene der FFT um den Faktor 2 skaliert, werden die Eingabesamples vor Ausführung der Transformation so verkleinert, daß keine Überläufe bei den Berechnungen auftreten können.

2.1.2 Datenübertragung

Zur Anbindung des EVM an den Computer wurde der häufig für PC-Erweiterungskarten eingesetzte PCI-Bus verwendet. Er besitzt eine Breite von $32bit$ und eine maximale Übertragungsrate von 132 MB/sec. Die in der Praxis erreichten Raten liegen jedoch um einiges tiefer, da diese Bandbreite auch auf andere an den PCI-Bus angeschlossenen Geräte, wie beispielsweise den Video-Conntroller, aufgeteilt wird.
Zur Datenübertragung über den PCI-Bus stehen mehrere Möglichkeiten zur Verfügung.

- **Messages:**

  Die PCI-Spezifikation sieht für jede Buskomponente vier 32-bit Nachrichtenregister vor, welche von anderen Komponenten beschrieben und gelesen werden können. Dieser Mechanismus findet vor allem dann Verwendung, wenn nur kurze Nachrichten zwischen den Komponenten ausgetauscht werden müssen.

- **PCI-FIFO:**

  Dieser kleine *First In First Out* Zwischenspeicher kann zur Datenübertragung von einer Komponente fortlaufend beschrieben werden und vom Empfänger wieder ausgelesen werden. Die Synchronisierung, das heißt die Blockierung weiterer Übertragungen bei vollem Zwischenspeicher und das Warten auf eintreffende Daten, wird von der Hardware überwacht.

Diese Übertragungsmethode wird auch hier verwendet, da die Übertragungsgeschwindigkeit von ca. 6 MB/sec ausreichend ist und einfache Routinen zum Umgang mit dem FIFO-Speicher bereitstehen.

- **DMA-Übertragungen**

  Viele PCI-Komponenten, auch das EVM, besitzen die Fähigkeit, Daten per DMA zu anderen Komponenten zu übertragen. Die Übertragung erfolgt dabei im Hintergrund, ohne Beteiligung der CPU bzw. des DSP. Zur Initiierung wird ein DMA-Controller entsprechend programmiert und die Übertragung gestartet.

2.2 Host-Seite

Die Grundlage auf der Host-Seite bildet eine *Microsoft Foundation Classes (MFC)* Dialog-Anwendung, welche die übertragenen Spektren in ihrem Fenster anzeigt. Dabei wird der Haupt-Thread der Anwendung für Benutzereingaben und die Verarbeitung von Systemnachrichten verwendet, während für das Empfangen und die Ausgabe der Spektren ein eigener Thread erzeugt wird.
Nach dem Programmstart werden zuerst die Komponenten des Dialogs initialisiert. Danach wird der Ausgabe-Thread gestartet. Hier wird zuerst eine Verbindung zum EVM aufgebaut, der DSP angehalten und das Board zurückgesetzt. Anschließend wird der oben beschriebene DSP-Programmcode mit einer Laderoutine überspielt und die Ausführung auf dem DSP gestartet.

```
HANDLE hBd    = NULL; /* Board device handle             */
short  iBd    = 0;    /* Board index                     */
LPVOID hHpi   = NULL; /* HPI interface handle            */
char coffNam[]= "progcode.out"; /* COFF file name        */

...

hBd = evm6x_open( iBd, bExcl );
evm6x_reset_board(hBd);
evm6x_reset_dsp(hBd,mode);
hHpi = evm6x_hpi_open(hBd);
evm6x_init_emif(hBd, hHpi);
WriteWord2Mem( hHpi, 0x01840070, 0x00000010);
evm6x_coff_load(hBd,hHpi,coffNam,bVerbose,bClr,bDump);
evm6x_hpi_close(hHpi);
evm6x_unreset_dsp(hBd);
```

Zur Datenübertragung über den PCI-Bus stehen auch auf PC-Seite einfache Routinen zur Verfügung. In der Schleife des Ausgabe-Threads werden mit dem folgenden Funktionsaufruf jeweils die Daten für einen Frame über die PCI-Schnittstelle in einen Puffer eingelesen.

```
evm6x_read(hBd, pciBuffer, &Length);
```

Die Absolutbeträge der Frequenzspektren sowie die zugrundeliegenden Audiodaten werden anschließend graphisch dargestellt.
Stoppt der Benutzer die Anzeige, um die letzten Spektren zu analysieren, so werden weiterhin Daten über den PCI-Bus empfangen. Dies verhindert ein zu langes Blockieren des DSP: für den benutzten Ping-Pong-Mechanismus muss gewährleistet sein, daß die Callback-Routine rechtzeitig vor einem erneuten Aufruf der Routine (mit dem anderen gefüllten Puffer) beendet wird.

3 Beispiel 2: G723-Coder

Als weiteres Beispiel einer typischen Anwendung für DSPs wurde der von der ITU spezifizierte G723-Coder[1] auf dem DSP implementiert. Hierbei handelt es sich um ein verbreitetes Verfahren zum Kodieren von Sprachsignalen in Telefonqualität mit einer Datenrate von 6.3 bzw. 5.3 $kbit/s$. Neben dem schriftlichen Standard ist von der ITU auch der C-Sourcecode einer Fixed-Point Implementierung des Coders erhältlich. Dieser enthält auch eine Menge von Testsequenzen mit denen die bitgenaue Kodierung überprüft werden kann. Der Code wurde in der Version 5.1 als Grundlage für die Anpassung an den TI C62x DSP verwendet.
Der Referenzcode ist sehr allgemein gehalten und verzichtet auf die Verwendung von plattformspezifischen Eigenheiten. Das erleichtert den Einsatz auf vielen unterschiedlichen Plattformen ohne gravierende Änderungen in der Struktur des Codes. Bei der Anpassung an den TI C62x DSP ergaben sich jedoch einige typische Schwierigkeiten, die im folgenden genauer erläutert werden.

3.1 Datenübertragung

Im Referenzcode erfolgt die Ein- und Ausgabe der Samples bzw. des erzeugten Codes über Dateien. Da der DSP jedoch keinen direkten Dateizugriff hat, erfolgt der Datenaustausch wie auch beim Spektrum-Analysator durch PCI-Datenübertragungen. Um die Testsequenzen zur bitgenauen Überprüfung verwenden zu können, erfolgt auch die Eingabe der Sprachsignale über den PCI-Bus vom Host, statt direkt über den Audio-Codec des EVM.

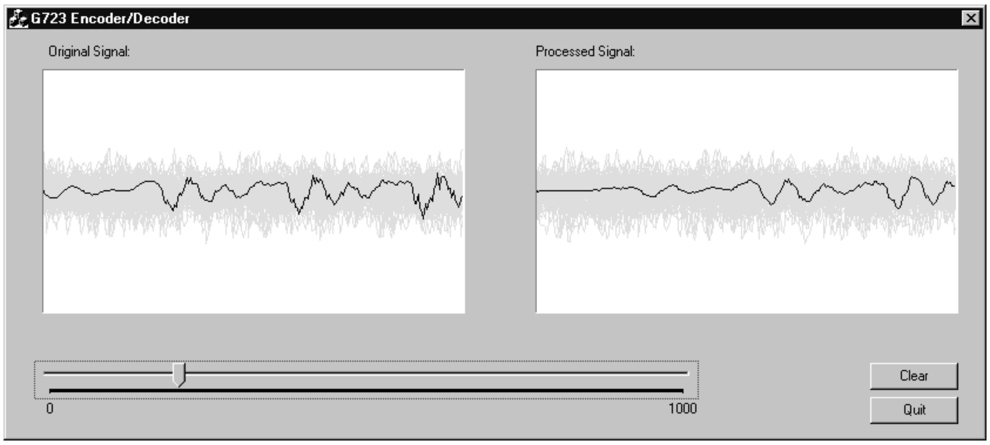

Abbildung 3: Screenshot des G723-Coders

Als kleiner Unterschied zum Spektrum-Analysator werden die Daten synchron vom und zum Host übertragen. Das bedeutet, daß bei jeder Übertragung der weitere Programmablauf bis zur Beendigung des Datenaustausches angehalten wird. Auf der einen Seite vereinfacht sich die Speicherverwaltung dadurch beträchtlich, da keine Puffer zur Übertragung von Daten im Hintergrund verwaltet werden müssen. Auf der anderen Seite kommt der Zeitaufwand der Übertragung zum reinen Kodierungszeitaufwand hinzu.

3.2 Mathematische Operationen

Übersetzt man den Referenzcode nur mit den Änderungen zur Ein- und Ausgabe von Daten, so benötigt der DSP etwa 20 mal länger zum Kodieren und anschließenden Dekodieren als es für den Echtzeitbetrieb erforderlich ist. Dies ist im wesentlichen auf die allgemeine Implementierung zurückzuführen, welche für jede mathematische Operation einen Funktionsaufruf ausführt. In den einzelnen Unterfunktionen werden dann teilweise recht komplexe Operationen durchgeführt, um auf jeder Hardware exakt die gleichen Ergebnisse zu erhalten.
Für viele der Operationen unterstützt der C62x C-Compiler sogenannte *Intrinsics*, die für die jeweilige Operation den passenden DSP-Befehl einsetzen. Als gutes Beispiel dient die Operation *Saturated Add*, welche zwei Werte addiert und das Ergebnis bei Überlauf auf das Maximum bzw. Minimum begrenzt.

```
int sadd(int a, int b) {
  int result;
```

```
    result = a + b;
    if (((a ^ b) & 0x80000000) == 0) {
      if ((result ^ a) & 0x80000000)
        result = (a < 0) ? 0x80000000 : 0x7fffffff;
    }
    return result;
  }
```

Der Befehlssatz des TI C62x DSP enthält genau diese Operation als Grundoperation. Aus dem C-Code erfolgt der Aufruf durch das Intrinsic `_sadd(a, b)`.
Um die Funktionsaufrufe zu vermeiden, welche auf dem DSP sehr zeitaufwendig sind, ersetzt man die Funktionsdefinitionen wo möglich durch `#define`-Anweisungen. Dadurch werden die Operationen ohne manuelle Änderung jedes Aufrufs im Quellcode direkt ersetzt. Die *Multiply and Add* Operation wird beispielsweise durch folgende effiziente Anweisung definiert:

```
#define mac(a,b,c) (_sadd(a,_smpy(b,c)))
```

In einem von TI erhältlichen Designers Notebook[5] finden sich für viele der im Kodierer verwendeten Operationen für den TI C62x optimierte Alternativen.

3.3 Speicherzuweisung

Wie bereits oben erwähnt besitzt das EVM mehrere verschiedene Speicherbereiche für Programm und Daten, welche mit unterschiedlichen Geschwindigkeiten arbeiten. Durch den Umfang des compilierten Programmcodes von insgesamt ca. $90KB$ reicht der interne Programmspeicher mit $64KB$ nicht mehr aus, wodurch dem Programm standardmäßig der externe Programmspeicher zugewiesen wird. Allerdings ergaben Versuche, daß Programmcode im externen Speicher ca. um den Faktor drei langsamer ausgeführt wird als im internen Speicher. Deshalb wurden mit dem Profiler die zeitaufwendigsten Teile des Algorithmus gesucht und diese dem internen Programmspeicher zugewiesen. Die restlichen Teile wie Initialisierungen und Code zur Datenübertragung wurden im externen Programmspeicher belassen.
Da der Compiler normalerweise alle Programmteile in die Section `.text` einordnet, muß ihm mitgeteilt werden, wenn er davon abweichen soll. Das geschieht durch eine `#pragma`-Anweisung für jede Funktion, die nicht zur Standardsection gehören soll. Angegeben wird der Funktionsname und die

Section, zu welchem die Funktion gehören soll. Ein Beispiel für die Funktion `Comp_Lpc`:

```
#pragma CODE_SECTION(Comp_Lpc, ".intprog")
```

Außerdem müssen die Funktionsaufrufe zwischen den beiden Codesections als `far` deklariert werden, da die Speicherbereiche zu weit auseinander liegen. Eine Möglichkeit wäre alle Funktionen, die aus dem jeweils anderen Segment aufgerufen werden, einzeln als `far` zu deklarieren. Dadurch erhält man zwar den kleinsten und schnellsten Code, allerdings muß man bei jeder Verlagerung einer Funktion in die jeweils andere Section überprüfen, aus welchen Teilen die Funktion aufgerufen wird und welche weiteren Funktionen daraus aufgerufen werden. Eine einfachere Möglichkeit ist die Änderung der globalen Voreinstellung für Funktionsaufrufe auf `far`. Dies erreicht man mit dem Compilerschalter `-ml1`.
Dem Linker muß dann noch mitgeteilt werden, in welche Speicherbereiche die beiden Sections `.text` und `.intprog` gehören. Dazu werden die folgenden Zeile zur Linkersteuerungsdatei im Sections-Abschnitt hinzugefügt.

```
.text          load = SBSRAM_PROG_MEM
.intprog       load = INT_PROG_MEM
```

Die Daten belegen insgesamt nicht mehr als $64KB$ und können deshalb alle im internen Datenspeicherbereich `INT_DATA_MEM` plaziert werden.

3.4 Hostanwendung

Im Hostprogramm werden die Sprachdaten aus einer Datei eingelesen und über den PCI-Bus an den DSP übertragen. Dort werden die Daten nach dem G723-Standard kodiert, dekodiert und wieder zum Host zurückübertragen. Das Originalsignal und das durch den DSP kodierte und dekodierte Signal werden dann zum Vergleich in zwei Fenstern nebeneinander graphisch dargestellt.
Bei Aktivierung aller Optionen, wie dem im Standard vorgesehenen Hochpassfilter und dem Postfilter sowie der graphischen Ausgabe, ist der mit obigen Maßnahmen optimierte Code um den Faktor $1,5$ schneller als zum Echtzeitbetrieb nötig wäre. Verzichtet man auf die graphische Ausgabe, so erreicht er den Faktor 2. Würde man die Datenübertragung auf dem EVM als Hintergrund-Prozess implementieren, so wäre mit den oben genannten Techniken ungefähr der Faktor 3 realisierbar.

Literatur

[1] *Dual Rate Speech Coder for Multimedia Communications Transmitting at 5.3 and 6.3 kbit/s*. ITU, March 1996. Standard Number: G.723.1.

[2] *TMS320C6000 Optimizing C Compiler User's Guide*. Texas Instruments, July 1998. Literature Number: SPRU187D.

[3] *TMS320C6x Evaliation Module Reference Guide*. Texas Instruments, March 1998. Literature Number: SPRU269.

[4] Fixed-point arithmetic - an introduction. http://www.shadow.net/ yates/fp.htm.

[5] R. Scales. *Designer's Notebook: ETSI Math Operations in C for the 'C62xx*. Texas Instruments.