Praktikumsprotokoll Mikrorechentechnik II – „Grundlagen der digitalen Signalverarbeitung“

Gruppe 50:
Marcel Junige, Fabian Kurz
Martin Laabs, Lars Lindenmüller

02. Mai 2005

Inhaltsverzeichnis

1 Aufgabenstellung

Aufgabe des Praktikums war die Implementierung eines Funktionsgenerators und eines nichtrekursiven FIR Filters auf der Basis eines ADSP-21020 DSP Chips.

1.1 Aufgabe 1 - Funktionsgenerator

Der zu implementierende Funktionsgenerator soll an den beiden Ausgängen `OUT A` und `OUT B` der AD7769 D/A-Wandler je nach Betriebsart entweder ein Sinus- und ein Rechtecksignal oder eine Rampenfunktion und ein Dreieckimpuls mit einer vorgegebenen Periodendauer T und Amplitude max erzeugen, wie in Abbildung 1.1 dargestellt.

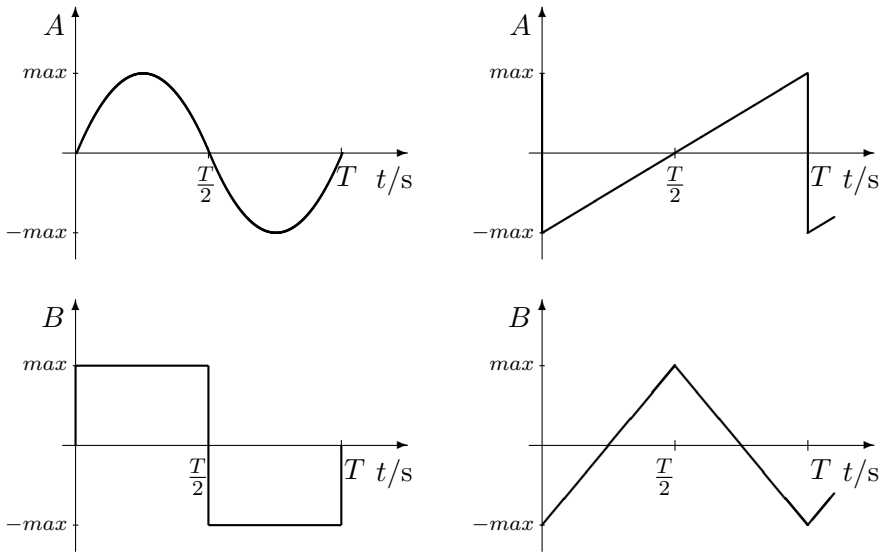

Abbildung 1: Ausgabefunktionen des Funktionsgenerators. Links Sinus/Rechteck, rechts Rampe/Dreieck

1.2 Aufgabe 2 - Nichtrekursives Digitalfilter

Ein nichtrekursives Digitalfilter (FIR) mit folgenden Differenzengleichungen ist zu realisieren:

$$y_1(n) = x(n) \tag{1}$$

$$y_2(n) = 0{,}5\,[x(n) + x(n-i)] \tag{2}$$

y_1 und y_2 sind die Ausgangssignale des Systems, wobei y_1 lediglich der Kontrolle dient. $y_2(n)$ ist das Ausgangssignal des nichtrekursiven Filters, dessen Aufbau in Abbildung 2 verdeutlicht wird. Es wird der Mittelwert des aktuellen Abtastwertes $x(n)$ und einem früheren Abtastwert

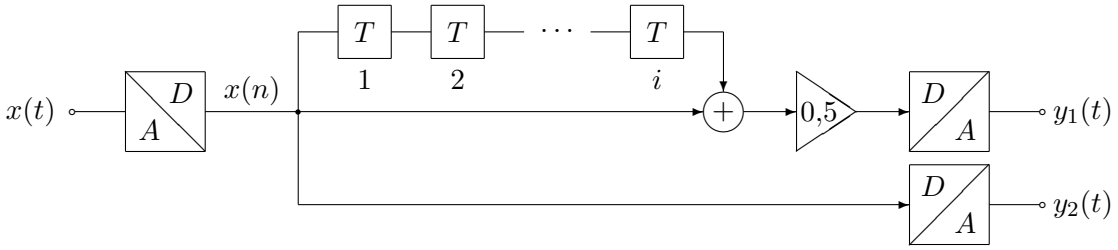

Abbildung 2: Blockschaltbild des FIR Filters

$x(n-i)$ gebildet. Bei Sinussignalen ergibt sich eine frequenzabhängige Phasenverschiebung, die zu einer Verstärkung oder Auslöschung des Signales führen kann.

2 Entwurf und Implementierung

2.1 Grundlagen des verwendenten DSP Systems

Zum Verständnis der Implementierung wird kurz auf das DSP System eingegangen.

2.1.1 Ablaufsteuerung

Die Ablaufsteuerung der zu implementierenden Programme geschieht über den Interrupt 2, welcher über einen Taster ausgelöst werden kann.

Bei der Auslösung wird eine Funktion aufgerufen, in der die globale Variable `choice` im Bereich $1 \ldots 3$ umlaufen lassen, was für die 3 verschiedenen Betriebszustände des Funktionsgenerators bzw. des Filters steht. Außerdem wird die globale Variable `Run` auf 0 gesetzt.

Im Hauptprogramm läuft nach der Initialisierung eine unendliche Schleife (`while (1)`), in der mit einem `switch`-Konstrukt abhängig von `choice` eine Funktion aufgerufen wird. Vorher wird `Run` jedoch auf 1 gesetzt.

Die aufgerufene Funktion ihrerseits enthält eine `while`-Schleife mit der Bedingung `Run`, läuft also so lange bis `Run` wieder durch Auslösen des Interrupt 2 zu Null wird.

2.1.2 Timing

Das Timing in den eigentlichen DSP Funktionen wird durch Aufruf von `idle()` realisiert. Diese Funktion wartet bis zum Ablauf des aktuellen `TimerCycle`, welcher eine Länge von $\frac{f_{\text{Takt}}}{\text{SampleRate}}$ Takten hat. Somit können in einer Schleife erst sämtliche Berechnungen ausgeführt werden, was weniger Takte als einen `TimerCycle` dauern muss, und dann durch Aufruf von `idle()` sichergestellt werden, daß die nächste Berechnung erst in der nächsten Sampleperiode durchgeführt werden.

2.1.3 Datenumwandlung mit FloatToByte

Die Ausgabe der Daten aus den DSP-Funktionen geschieht durch direktes Schreiben in den Speicherbereich der D/A-Konverter. Da diese allerdings im konkreten Falle eine Auflösung von 8 Bit aufweisen, und somit einen Integerwert im Bereich $0 \ldots 255$ erwarten, muss der in den Funktionen berechnete Wert, der zwischen $-max \ldots max$ liegt umgewandelt werden.

Per Definition muss $max \leq 1$ sein, also liegt ein umzuwandelnder Wert maximal im Bereich von $-1 \ldots 1$. Um diesen Bereich nach $0 \ldots 255$ zu transformieren, muss um 1 verschoben, also 1 addiert werden um in den Bereich $0 \ldots 2$ zu gelangen, und dieser Wert auf $0 \ldots 255$ skaliert werden, also eine Multiplikation mit 127,5. Der gesuchte Wert muss eine ganzzahl sein, daher wird das Ergebnis als `int` gecastet. Die Funktion `FloatToByte` sieht also folgendermaßen aus:

Quelltext

```
int FloatToByte(float Fl) { return (int) ((Fl + 1) * 127.5); }
```

2.1.4 Lesen der Daten vom A/D-Wandler

Beim Einlesen der Werte vom A/D-Wandler erhält man eine 32 Bit Festkommazahl, in deren 8 MSB sich der Wandlerwert im Bereich von $0 \ldots 255$ befindet. Um diesen in die zum Rechnen gebrauchte Zweierkomplementdarstellung umzuwandeln, sind folgende Operationen von Nöten:

```
InPortA = InPortA >> 24;
```

Der Inhalt von `InPortA` wird um 24 Bits nach rechts verschoben.

```
InportA = ((~InPortA) & 0xFFFFFF80) | (InPortA & 0x7F);
```

Umwandlung in Zweierkomplementdarstellung.

2.2 Funktionsgenerator

Die Implementierung des Funktionsgenerators erfolgt wie im vorgegenenen Quelltext in zwei Funktionen `sine` und `slopetriangle`. Diese wurden jeweils in externen Dateien gespeichert, so daß diese implementierungsunabhängig mit einer festgelegten Schnittstelle aus dem Hauptprogramm aufgerufen werden können.

2.2.1 Sinus- und Rechteckgenerator

Die Berechnung des Sinus- und Rechtecksignals geschieht in der Funktion `void sine()`. Es gibt eine Variable `SampleNr`, die in einer Sekunde von 1 bis `SampleRate`, im konkreten Fall 25 000 zählt. Dies geschieht dadurch, daß nach der Ausgabe jedes Wertes die Funktion `idle()` aufgerufen wird, die auf den nächsten Timer-Interrupt wartet, der die Sample-Rate bestimmt. Erreicht der Wert der Wert von `SampleNr` den Wert `SampleRate`, wird ersterer wieder auf 1 zurückgesetzt.

Für jeden dieser Werte von `SampleNr` muß nun der entsprechende Wert der Sinusfunktion bei der Frequenz f (gegeben durch die Variable `int f`) und mit der Amplitude max (`float max`, $0 \leq max \leq 1$) errechnet werden. Die Formel lässt sich einfach aufstellen:

$$f(x) = max \cdot \sin\left(2\pi \cdot f \cdot \frac{x}{R}\right)$$

wobei x `SampleNr` und R `SampleRate` entsprechen.

Die Rechteckfunktion lässt sich schliesslich einfach aus der Sinusfunktion berechnen: Wenn diese einen negativen Wert annimmt, also der mit `FloatToByte` umgewandelte Wert unter 127 liegt, ist der Funktionswert $-max$, ansonsten ist er max.

Quelltext

```
void sine(int f, float max) {
    long int SampleNr = 1;  /* SampleNr, zaehlt in einer Sekunde bis 25k */
    while (Run) {
        /* Berechnung des Sinus-Signals mit f bei SampleRate */
        OutPortA = FloatToByte (max * sinf(2*pi*f*SampleNr/SampleRate));

        /* Wenn Sinuswert negativ, OutPortB = -max, sonst max */
        OutPortB = (OutPortA<127) ? FloatToByte(-1.0*max) : FloatToByte(max);

        /* Naechstes Sample */
        SampleNr++;

        /* Verhindern, dass es einen Ueberlauf der Sample-Variable gibt */
        SampleNr %= SampleRate;

        /* Der Prozessor hat diese Befehle deutlich eher berechnet als das
         * naechste Sample ausgegeben werden muss. Daher wird gewartet
         * bis der naechste Timer-Interrupt ausgeloest wird. */
        idle();

    } /* Ende der while(Run)-Schleife */
} /* Ende von sine() */
```

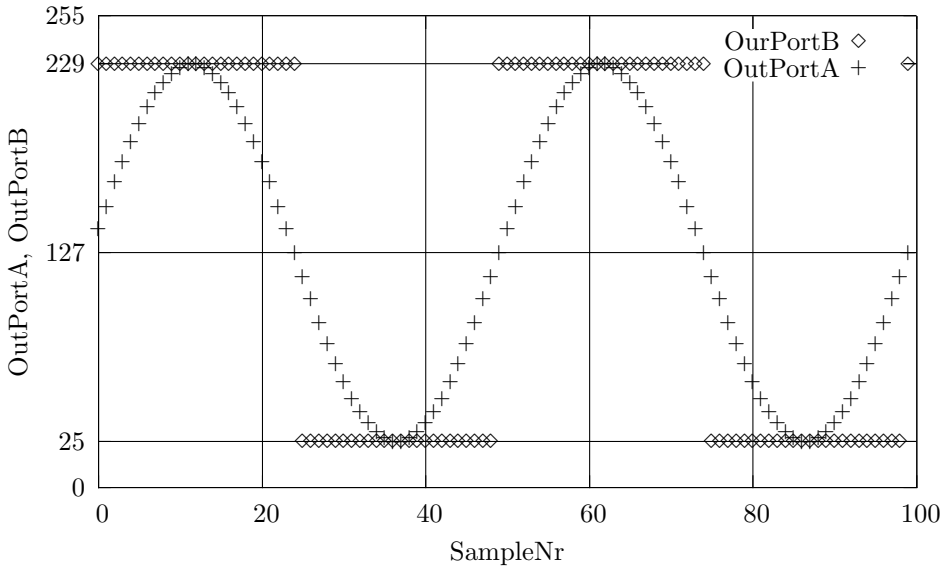

Abbildung 3: Die ersten 100 Werte der Ausgabe der Funktion `sine(500,0.8)`

Da zur Vorbereitung des Versuches kein DSP-Board zur Verfügung stand, wurde das Programm leicht abgewandelt und mit dem GNU gcc getestet. Die Ausgabe wurde mit Hilfe von GNUplot visualisiert und diente zur Überprüfung der Richtigkeit der gelieferten Werte. Abbildung 3 zeigt die ersten 100 Werte der Ausgabe von `sine(500,0.8)`, die den Erwartungen entsprechen.

2.2.2 Rampe- und Dreiecksgenerator

Die Funktion zur Berechnung des Rampen- und Dreieckssignales ist der Funktion zur Generierung des Sinus- und Rechtecksignals sehr ähnlich: Auch hier gibt es eine Laufvariable `SampleNr`, die in einer Periode von 1 bis `SampleRate` läuft. Für jeden Wert dieser Variable wird der Wert der Rampenfunktion nun anhand folgender Formel berechnet:

$$f(x) = -max + \frac{2 \cdot max \cdot f \cdot \left(x \ \% \ \frac{R}{f}\right)}{R}$$

Wobei x `SampleNr`, R `SampleRate` entspricht. Diese Funktion liefert eine verschobene Gerade der Steigung $\frac{2 \cdot max \cdot f}{R}$, die sich periodisch alle $\frac{R}{f}$ wiederholt. Die Dreiecksfunktion berechnet sich nun aus dieser Funktion durch

$$g(x) = max - |2 \cdot f(x)|\,,$$

wie sich schnell durch grafische Konstruktion anhand der Rampenfunktion ermitteln ließ.

Quelltext

```
void slopetriangle(int f, float max) {
    long int SampleNr = 1;    /* Samplenummer, zaehlt in einer s bis 25k */
    float Slope;              /* Hilfsvariable fuer Rampenfunktion       */
    while (Run) {
        /* Berechnung des Rampensignals als Gerade mit Modulo */
        Slope = -1*max +((2*max*f*(SampleNr%(SampleRate/f)))/SampleRate);

        OutPortA = FloatToByte(Slope);
```

```
        /* Berechnung der Dreiecksfunktion aus der Rampenfunktion       */
        OutPortB = FloatToByte(max - fabsf(2*Slope));

        /* Naechstes Sample                                             */
        SampleNr++;

        /* Verhindern, dass es einen Ueberlauf der Sample-Variable gibt */
        SampleNr %= SampleRate;

        /* Der Prozessor hat diese Befehle deutlich eher berechnet als das
         * naechste Sample ausgegeben werden muss. Daher wird gewartet
         * bis der naechste Timer-Interrupt ausgeloest wird. */
        idle();
    } /* Ende der while(Run)-Schleife */
} /* Ende von slopetriangle() */
```

Auch die Funktion von `slopetriangle()` wurde anhand von GNUplot überprüft und verifiziert. Im Praktikum funktionierten die Funktionen exakt wie erwartet.

2.3 FIR Filter

Das zu implementierende FIR-Filter wurde mit Hilfe eines Ringpuffers realisiert. Dieser besteht aus einem Array `Queue` aus maximal 20 Integerwerten und zwei „Zeigern“ auf die Positionen n (`p`) und $n-1$ (`p2`) des Ringpuffers. Es handelt sich hierbei jedoch nicht um echte Zeiger, die auf eine Speicheradresse verweisen sondern um Integerzahlen, die die Position des betreffenden Elements im Array angeben.
In der `while`-Schleife wird nun

- ein Wert vom AD-Wandler eingelesen, in 8 Bit Zweierkomplementdarstellung umgewandelt und an der Position `p` im Array gespeichert,
- der Eingangswert ungeändert an den Ausgang A ausgegeben,
- die Position des Zeigers `p2` berechnet,
- der Durchschnitt aus dem Eingangssignal und dem ältesten Wert des Ringpuffers (entspricht `Queue[p2]`) gebildet und am Ausgang B ausgegeben,
- die Position von `p` für den nächsten Durchgang berechnet.

Quelltext

```
void FilterFIR (int Grad) {
    int p=0;            /* Pseudozeiger auf aktives Element              */
    int p2=0;           /* Pseudozeiger auf n-i                          */
    int Queue[20];      /* Array der Laenge Grad zum Speichern alter Werte */
    int i;

    for (i=0;i<=Grad;i++){            /* Initialisierung der Queue       */
            Queue[i] = 0;             /* mit Nullen                      */
    }
```

```
    while (Run) {
        /* Neuen Wert vom Input einlesen und in
         * Zweierkomplementdarstellung bringen    */
        InPortA = InPortA >> 24;
        InPortA = ((~InPortA) & 0xFFFFFF80) | (InPortA & 0x7F);

        Queue[p] = InPortA;            /* neuen Wert ins Array schreiben */

        OutPortA = InPortA;            /* Ungeaenderter Wert fuer A      */
        OutPortA = OutPortA + 127;
        OutPortA = OutPortA << 24;

        /* p2 = Position des Elements n-1, also immer das naechste Element,
         * bzw. bei p=9 das 0. Element */

        (p == (Grad)) ? (p2 = 0) : (p2 = p+1);

        OutPortB = (InPortA + Queue[p2])/2;    /* Addition */

        OutPortB = OutPortB + 127;
        OutPortB = OutPortB << 24;

        p++;                           /* 'Pointer' erhoehen */
        p %= (Grad+1);                 /* Ueberlauf verhindern */

        idle();
    }
}
```

Im Gegensatz zur ersten Teilaufgabe funktionierte das FIR-Filter nicht auf anhieb, obwohl auch dessen Funktion durch eine kleine Testroutine vor dem Praktikum auf korrekte Funktion überprüft wurde. Es stellte sich nach kurzer Suche heraus, daß der Aufruf der `idle()`-Funktion fehlte, bzw. auskommentiert war. Nachdem dies behoben war, funktionierte das Filter wie gefordert.

Am Oszilloskop war die doppelte Amplitude bei einer Phasenverschiebung von $G \cdot 2\pi$ und die Auslöschung des Signales bei einer Phasenverschiebung von $\pi + G \cdot 2\pi$ (G: Grad) deutlich zu beobachten. Bei der konkreten Implementierung mit $G = 10$ und $G = 20$ ergaben sich für die Auslöschung Frequenzen von respektive 500 Hz und 1000 Hz und für die Addition von 1000 Hz und 2000 Hz.

3 Diskussion

Die Lösung der Praktikumsaufgaben stellte dank der genauen Aufgabenstellung und der vorbereiteten Quelltexte keine große Hürde dar. Bis auf eine vergessene Auskommentierung in der Funktion zur Implementierung des FIR-Filters funktionierte alles auf Anhieb.

Bei der Implementierung des Funktionsgenerators hätte man die Laufvariable `SampleNr` nicht von 1...`SampleRate` sondern nur bis `SampleRate/f` (entsprechend einer Periode) laufen lassen können; somit erspart man sich beim Rampengenerator die umständliche Modulo-Berechnung und gewinnt etwas an Geschwindigkeit.