

U2 Implementierung RC-Tiefpassfunktion

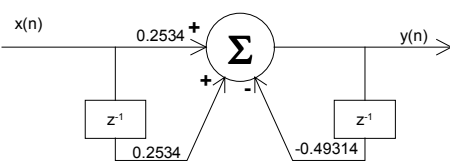
Umfeld

Im Rahmen eines Praxisbeispiels wurde die analoge Übertragungsfunktion $G(s)$ eines RC-Tiefpass 1. Ordnung mit $f_c=1\text{kHz}$ mittels Bilinearer z -Transformation in die frequenzdiskrete Übertragungsfunktion $G(z)$ übergeführt und als IIR-Filter realisiert werden:

$$G(s) = \frac{6283.185}{s + 6283.185} \quad (\text{analog}) \quad f_s = 9.6\text{kHz}$$

$$G(s) = \frac{6517.52}{s + 6517.52} \quad (\text{analog, prewarped})$$

$$G(z) = G(s) \Big|_{s=\frac{2}{T_s} \frac{z-1}{z+1}} = \frac{0.2534(z+1)}{z-0.49314} = \frac{0.2534 + 0.2534z^{-1}}{1 - 0.49314z^{-1}}$$



IIR-Filter 1. Ordnung in direkter Form.

Das Filter auf dem DSP56002 EVM implementiert und untersucht werden. Dazu ist die Übertragungsfunktion als IIR-Filter in direkter Form, d.h. mit zwei Verzögerungsgliedern, zu codieren und zu testen.

Aufgaben

1. Aufsetzen eines neuen, leeren C-Projektes (z.B. Uebung2_RC-Tiefpass, keine Leerschläge erlaubt) auf der Grundlage des Prototypen-Files. Kommentieren und Anpassen des Prototypen.
2. Implementieren der Übertragungsfunktion durch Definition der Filterfunktion, die die eigentliche Übertragungsfunktion aufnimmt:

```

_fract filterRC(_fract x)
{
    static _fract z1;    // Zeitverzoeigerung(en) definieren
    ....
    return ...    // Ausgangssignal y retournieren
}

```

Hinweise:

Benutzen Sie zur Rechnung den Datentyp `fract`. Er verkörpert den Festkommatyp des DSP56002 Signalprozessors mit dem Wertebereich $[-1,1)$. Zu beachten ist, dass der Codec mit 9.6kHz Samplerate initialisiert wird. Dazu ist die symbolische Konstante `SAMPLE_RATE_9` aus `CodecConst.h` zu benutzen.

3. Austesten mit Musik, evtl. ausmessen des Amplitudenganges mit NF-Voltmeter.

Als Hilfe erhalten Sie eine mögliche Lösung, jedoch mit globalen Variablen realisiert:

```
void filterRC(void)
< static _fract xz1,yz1; // Zeitverzoeagerte Signale um eine Periode fuer x und y

y=0.2534 * x + 0.2534 * xz1 + 0.49314 * yz1; // G(z)
xz1= x; // x -> z^-1
yz1= y; // y -> z^-1
}

int main(void)
/* Hauptprogramm, Daten von Codec holen und je nach Brueckenstellung gefiltert oder direkt ausgeben.
*/
< PCTL_I=0x261009; // PLL des DSP setzen
PBC = 0; // Port B als Ein/Ausgaenge konfigurieren
PBDDR = 0x0100; // PB0..PB7,PB9..PB14 als Eingang definieren. PB8 als Ausgang.
PBD=0x0; // PB8 auf 0. PB8 dient als GND fuer Bruecke zu PB14(HACK)
// Codec Initialisieren:9.6kHz Sample Rate, Stereo, mit HP Filter, 16 Bit Datenwerte...
codecInit(NO_PREAMP | HI_PASS_FILT | SAMP_RATE_9 | STEREO+DATA_16 , // CTRL_WD_12
IMMED_3STATE |ITAL2_SELECI | BITS_64 | CODEC_MASTER, // CTRL_WD_34
0x000000, // CTRL_WD_56
0x000000 // CTRL_WD_78
);

while(1)
< while( SSISR.B.IFS); //Warten auf frame sync Transition
while(!SSISR.B.IFS)
x = *xData.d.i.audio_left;

if (PBD & 0x4000) // PB14-PB8 keine Bruecke?
y = x; // Audio direkt passieren lassen
else
filterRC(); // Audio gefiltert ausgeben

txData.d.i.audio_left=y;
txData.d.i.audio_right=<_fract>0.0; // Rechter Kanal nichts
txData.d.i.output_setting=TONE_OUTPUT;
txData.d.i.input_setting=TONE_INPUT;
}
return 0;
}

r:\dsp\c56\bin\c56.exe rc-tiefpass.c -fmb6f18a.tmp
55C1163F-5F52-4352-8D91-0B310B310B31
total errors: 0, warnings: 1
r:\dsp\c56\bin\as56.exe rc-tiefpass.src -fmb6f18b.tmp
r:\dsp\c56\bin\cc56.exe -o rc-tiefpass.out -fmb6f18c.tmp
r:\dsp\c56\bin\lc56.exe -o rc-tiefpass.abs rc-tiefpass.out -fmb6f18d.tmp
TASKING program builder v2.0 r8 SN00084745-006 (c) 1997 TASKING, Inc.
```