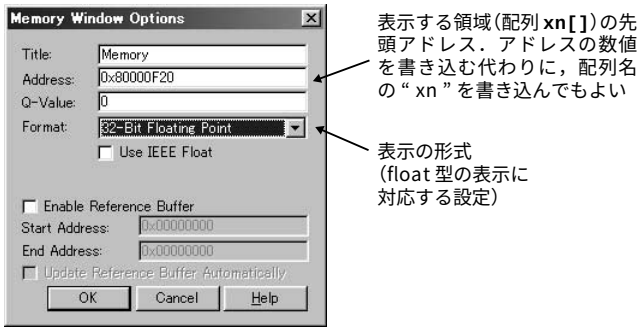
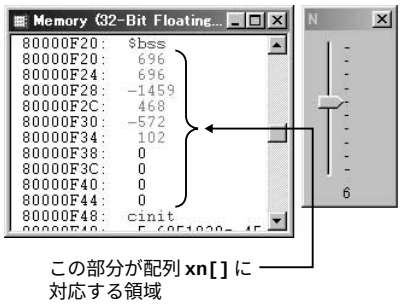
〔図4.7〕メモリの内容を表示する際のオプション設定のための画面



〔図4.8〕メモリの内容の表示とそのときのスライダのつまみの位置



示しています．**リスト4.1**のプログラムでは，配列**xn[ ]**の大きさは10として宣言していますが，スライダのつまみは6に設定されています．そのため，**0x80000F20**～**0x80000F34**の領域だけが使われ，残りの**0x80000F38**～**0x80000F44**の領域は使われておらず，0になっているようすがわかります．なお，**xn[0]**，**xn[1]**に対応する領域のデータがどちらも同じ696になっているのは，この状態が配列の内容を移動した後，まだ新しいデータが入力される前の状態に対応しているからです．

4.2 FIRフィルタ

4.2.1 FIRフィルタとは

FIRフィルタのFIRとは，finite impulse responseの略です．つまり，インパルス応答の継続時間が有限なフィルタです．言い換えると，インパルス応答が有限時間内に0になるフィルタです．インパルス応答とは，入力に式(4.9)で表されるインパルス信号$\delta[n]$を入力したときの出力です．

$$\delta[n]=\begin{cases}1, & n=0\\ 0, & n\neq 0\end{cases} \tag{4.9}$$

式(4.2)で表される移動平均処理について，式(4.9)の入力に対して出力を計算すると次のようになります．

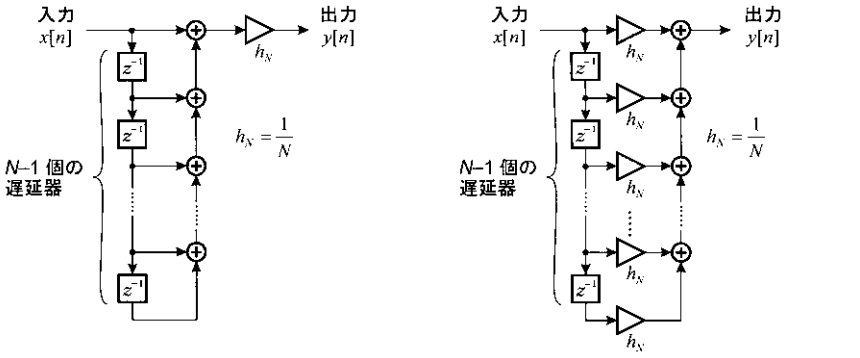
$$y[n]=\begin{cases}1/5, & n=0,1,\text{L},4\\ 0, & \text{それ以外}\end{cases} \tag{4.10}$$

この式からわかるように，$n\geq 5$で出力は0になるので，移動平均処理はFIRフィルタの一種と言うことができます．どのようなFIRフィルタかと言うと，移動平均の振幅特性を表している**図4.4**，**図4.5**から，移動平均による処理は低域通過フィルタとみなせることがわかります．しかし，低域通過フィルタとしての特性はあまりよいとはいえません．

4.2.2 直接形FIRフィルタ

ところで，N点移動平均のブロック図は**図4.9**(**a**)から**図4.9**(**b**)のように書き換えても，結果として同じ処理を行うことができます．**図4.9**(**b**)では，乗算器の係数がどれも同じh_Nという値になっていますが，必ずしも同じ値でなくてもかまいません．そのようなものが，一般のFIRフィルタということになりま

〔図4.9〕
移動平均のブロック図の変形



（**a**）原型（加算の後にh_Nによる乗算を行う）　（**b**）変形したもの（加算の前にh_Nによる乗算を行う）

す．ブロック図で描くと**図4.10**のようになり，その入出力の関係は，次の差分方程式で記述することができます．

$$y[n] = \sum_{m=0}^{M} h_m x[n-m] \tag{4.11}$$

この**図4.10**のタイプのFIRフィルタは，直接形（direct form）FIRフィルタと呼ばれています．このブロック図で実現されるFIRフィルタの振幅特性は，h_m（$m = 0, 1, \cdots, M$）により決定されます．

式（4.11）に対応する伝達関数$H(z)$は，次のようになります．

$$H(z) = \sum_{m=0}^{M} h_m z^{-m} \tag{4.12}$$

4.2.3　転置形FIRフィルタ

図4.10のブロック図で表されるFIRフィルタに対し，“転置（transpose）”という操作を行っても，式（4.11）の差分方程式と同じ処理ができます．転置の操作は，次のように行います．

（1）入力と出力を交換する
（2）信号の流れをすべて逆転する
（3）加算器と分岐点を交換する

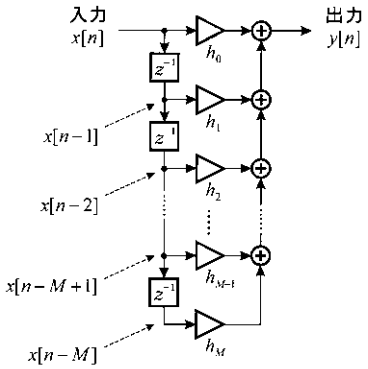
以上の操作を行って得られる転置形（transposed direct form）のFIRフィルタのブロック図は，**図4.11**のようになります．このブロック図に対応する差分方程式は，**図4.11**で示す$u_m[n]$，$m = 0, 1, \cdots, M$を使って，次のような連立の差分方程式で表すことができます．

$$\begin{cases} u_M[n] = h_M x[n] \\ u_{M-1}[n] = h_{M-1} x[n] + u_M[n-1] \\ \quad \vdots \\ u_1[n] = h_1 x[n] + u_2[n-1] \\ u_0[n] = h_0 x[n] + u_1[n-1] \\ y[n] = u_0[n] \end{cases} \tag{4.13}$$

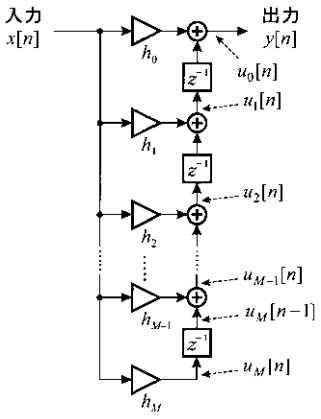
この式から，$u_m[n]$，$m = 0, 1, \cdots, M$を消去すると式（4.11）が得られます．

図4.11からわかるように，転置形のFIRフィルタでは係数と入力データとの乗算を行い，その積と遅

〔図4.10〕
直接形FIRフィルタのブロック図



〔図4.11〕
転置形FIRフィルタのブロック図



延器のデータとの加算を行い，その結果は次の遅延器に書き込むという処理になります．したがって，直接形の場合には必要だったデータの移動が，転置形では不要になります．

なお，式(4.13)に対応する伝達関数$H(z)$は，次のように書くことができます．

$$H(z)=h_0+\left(h_1+\left(h_2+\mathrm{L}\ +\left(h_{M-1}+h_Mz^{-1}\right)z^{-1}\mathrm{L}\ \right)z^{-1}\right)z^{-1} \tag{4.14}$$

この式は，多項式の計算でよく使われるホーナー(Hornor)法を式(4.12)に適用して変形した形になっています．したがって，式(4.14)と式(4.12)は，見かけ上は異なって見えますが同じものです．

その他，FIRフィルタのタイプとしては，縦続形(cascade form)，格子形(lattice form)などがありますが，本書では扱いません．

4.3 FIRフィルタのプログラミング（浮動小数点演算を利用）

4.3.1 直接形FIRフィルタのプログラム

(1) 設計

ディジタル・フィルタの場合，設計とはフィルタの係数を求めることです．ここでは**付録C.2**に示すFIRフィルタ設計プログラム(本書付属のCD-ROMに収録されている)でフィルタの係数〔式(4.11)のh_m〕を求めます．

設計の際に与えたパラメータを**表4.1**に示します．また，設計されたフィルタの振幅特性を**図4.12**に示します．

(2) プログラム作成

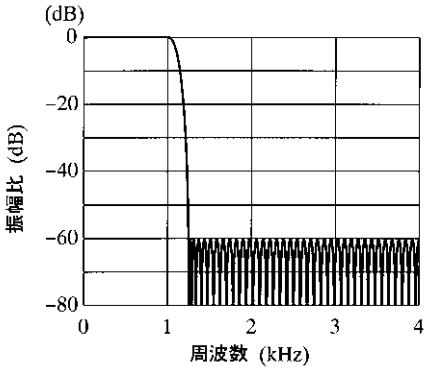
プログラム(**FIR100_LPF.c**)を**リスト4.3**に示します．配列**hn[ORDER+1]**に格納されているのがフィルタの係数です．入力信号の取り扱い方は，**4.1.5　*N*点移動平均のプログラム**の項で説明したように，入力信号を通常の配列に格納し，新しい信号が入力されるたびに，古い信号を移動していく方法を採用しています．

インクルード・ファイル**DSK_poll.c**，リセットベクタ，リンカ・コマンド・ファイルについては**4.1.5　*N*点移動平均のプログラム**の項で説明しているとおりです．ビルドの際のオプションは**3.2.2**

〔表4.1〕FIRフィルタを設計した際に与えたパラメータ

種類	通過域 / 阻止域型	
次数	100次	
標本化周波数	8kHz	
	帯域1	帯域2
下側帯域端周波数	0.0kHz	1.25kHz
上側帯域端周波数	1.0kHz	4kHz
ゲイン	1	0
重み	1	2

〔図4.12〕作成するFIRフィルタの振幅特性



〔リスト4.3〕直接形FIRフィルタのプログラム(FIR100_LPF.c)

```
/*************************************/
/*  FIR low pass filter, Direct form  */
/*      cutoff freq.  1 kHz           */
/*      order         100             */
/*************************************/
#include "DSK_polling.c"
#define ORDER 100

float xn[ORDER+1];
const float hn[ORDER+1] = {
     8.48509091e-06, -8.31609328e-04, -6.67256698e-04, -3.36995227e-04,
     3.29552972e-04,  9.00744435e-04,  8.73906254e-04,  9.79587163e-05,
    -9.86905519e-04, -1.55275509e-03, -9.83779166e-04,  5.45358331e-04,
     2.01087198e-03,  2.18703622e-03,  6.21431000e-04, -1.80715048e-03,
    -3.31113710e-03, -2.46482732e-03,  5.47591800e-04,  3.74173842e-03,
     4.57792551e-03,  1.91140409e-03, -2.80439593e-03, -6.18506087e-03,
    -5.27052985e-03,  1.68385595e-05,  6.27191885e-03,  8.68716172e-03,
     4.63309534e-03, -3.85018859e-03, -1.08274006e-02, -1.04732907e-02,
    -1.71024253e-03,  1.00692148e-02,  1.60674887e-02,  1.03854897e-02,
    -4.77635447e-03, -1.92655655e-02, -2.13474592e-02, -6.56526899e-03,
     1.71991140e-02,  3.30369250e-02,  2.58962235e-02, -5.41508820e-03,
    -4.34663544e-02, -5.93493168e-02, -2.89796892e-02,  5.06828324e-02,
     1.55816282e-01,  2.45108508e-01,  2.80071298e-01,  2.45108508e-01,
     1.55816282e-01,  5.06828324e-02, -2.89796892e-02, -5.93493168e-02,
    -4.34663544e-02, -5.41508820e-03,  2.58962235e-02,  3.30369250e-02,
     1.71991140e-02, -6.56526899e-03, -2.13474592e-02, -1.92655655e-02,
    -4.77635447e-03,  1.03854897e-02,  1.60674887e-02,  1.00692148e-02,
    -1.71024253e-03, -1.04732907e-02, -1.08274006e-02, -3.85018859e-03,
     4.63309534e-03,  8.68716172e-03,  6.27191885e-03,  1.68385595e-05,
    -5.27052985e-03, -6.18506087e-03, -2.80439593e-03,  1.91140409e-03,
     4.57792551e-03,  3.74173842e-03,  5.47591800e-04, -2.46482732e-03,
    -3.31113710e-03, -1.80715048e-03,  6.21431000e-04,  2.18703622e-03,
     2.01087198e-03,  5.45358331e-04, -9.83779166e-04, -1.55275509e-03,
    -9.86905519e-04,  9.79587163e-05,  8.73906254e-04,  9.00744435e-04,
     3.29552972e-04, -3.36995227e-04, -6.67256698e-04, -8.31609328e-04,
     8.48509091e-06};
```

〔リスト4.3〕直接形FIRフィルタのプログラム（つづき）

```
void main()
{
    short yn;
    int k;
    float acc;

    InitDSK(cach_bk4);                  /* Initialize DSK          */

    for (k=0; k<=ORDER; k++) xn[k] = 0.0;

    while(1)                            /* Endless loop            */
    {
        xn[0] = (float)McBSP0ReadRdy(); /* input                   */

        acc = 0.0;
        for (k=0; k<=ORDER; k++)        /* multiply and accumulate */
            acc = acc + hn[k]*xn[k];
        yn = (short)acc & 0xFFFE;
        for (k=ORDER; k>0; k--)         /* shift input samples     */
            xn[k] = xn[k-1];

        McBSP0Write(yn);                /* output                  */
    }
}
```

ビルドの際のオプションでの説明と同じです．これらは，基本的に以降で紹介するプログラムでも同じなので，以降は省略します．

(3) 実行のようす

ここで作成したFIRフィルタを実行させ，正弦波を入力したときの波形を，**写真4.1**（**a**），（**b**），（**c**），（**d**）に示します．各写真で，上の波形が入力信号，下の波形が出力信号です．これらの写真の中で，オシロスコープの縦軸の設定は，（**c**）の下の波形は100mV/div，（**d**）の下の波形は10mV/divとしています．一方，他はすべて1V/divとしています．

これらの波形の写真からわかるように，1kHzより高い周波数では出力信号の振幅が入力信号の振幅より小さくなっているので，作成したフィルタが低域通過フィルタとして働いていることを確認できます．

(4) 実行時間の測定

プログラムの実行時間は，Code Composer Studioで簡単に測定することができます．その手順を次に示します．

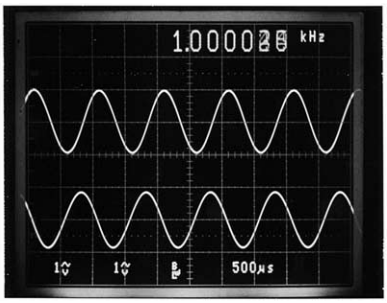
① 実行コードをロードする．

② メニューバーで［Profiler］を選択し，［Enable Clock］をクリックし，✔マークを付ける（**図4.13**）．

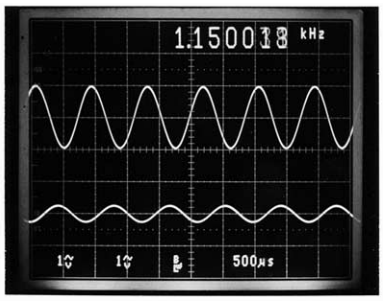
③ メニューバーで［Profiler］を選択し，［View Statics］をクリックすると，“Profile Statics”ウィンドウが開く．このウィンドウの大きさを調整し，見やすいようにする．

④ アセンブリ言語のレベルで測定点を指定したい場合は，**図4.14**のCのソースコードが表示されている

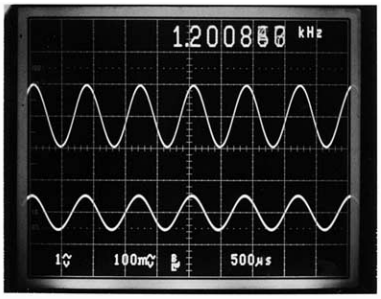
〔写真4.1〕
FIRフィルタを実行したようす（上：入力信号，出力信号）



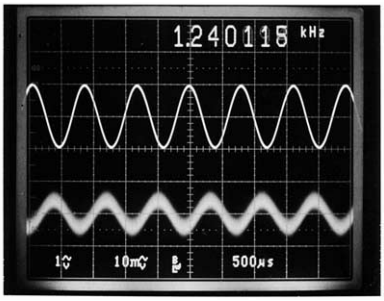
（**a**）1.00kHz(上：1V/div，下：1V/div)



（**b**）1.15kHz(上：1V/div，下：1V/div)



（**c**）1.20kHz(上：1V/div，下：100mV/div)

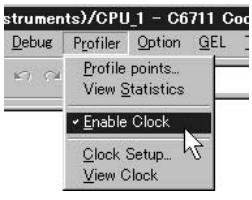


（**d**）1.24kHz(上：1V/div，下：10mV/div)

ウィンドウの上にカーソルを移動し，右クリックし，さらに[Mixed Mode]をクリックする．そうすると，C言語のソースコードの間にアセンブリ言語のコードも表示される．

⑤ カーソルを測定の開始点に移動し，  をクリックし，プロファイル・ポイントを設定する〔**図4.15**（**a**)〕．さらに，カーソルを測定の終了点に移動し，  をクリックし，プロファイル・ポイントを設定する〔**図4.15**(**b**)〕．これらの行は緑色に強調表示される．

〔図4.13〕
Enable Clockに✓マークを付けた状態



〔図4.14〕
Cのソース・コードと対応するアセンブリ命令を同時に表示するための手順