本文讲述的是 XN62Lxxx 的硬件滤波器的结构和使用方法。主要的内容是滤波器的结构，FIR（有限长单位冲激响应）滤波器配置和使用方法，IIR（无限长单位冲激响应）滤波器的配置和使用方法，以及 DMA 在滤波器上的使用。文中涉及到的 XN62Lxxx 芯片的其它有关内容请参考用户手册。

滤波器结构

无论是 FIR 还是 IIR 滤波器，它们的基本结构都是乘加。XN62Lxxx 硬件滤波器便是以乘加结构为基础，通过重复乘加和移位运算实现 1~16 阶滤波算法。

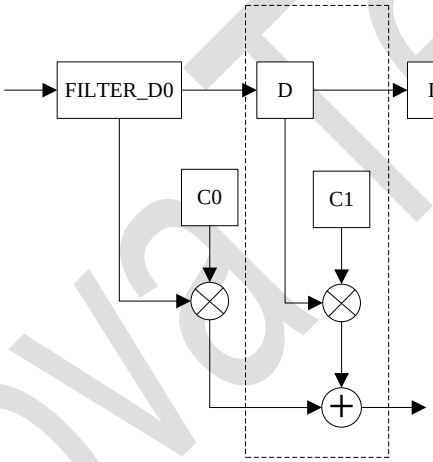


图 1 滤波器基本结构

在上图中，D 表示输入数据的延迟链（Delay chain），考虑到滤波器主要作用是对 AD 采样数据进行前期处理，所以输入数据和延迟链是 16 位的。

C0 和 C1 表示滤波器的参数，它们是 16 位的。在物理上它们和延迟链一样都是寄存器。由于不使用内存，所以可以显著加快滤波器计算速度，在每个时钟周期中完成一次乘加运算。

乘法器是 16x16=32 位的结构。

累加器的宽度为 36 位，可以充分保证滤波器的运算精度。

此外在滤波器的输出端还有一个钳位器，用户可以通过软件设置来决定输出位数，大于钳位值的输出将被置位成最大值。

下列寄存器控制滤波器（base address: 0x50070000）

FILTER_CR	R/W	0x280	Filter Control Register	0x0000 0000
FILTER_D	R/W	0x284	FILTER Data Register	0x0000 0000
FILTER_START	R/W	0x288	Filter Start Register	0x0000 0000
FILTER_RESULT	R/W	0x28C	Filter Result Register	0x0000 0000
FILTER_C0 ~ FILTER_C015	RO	0x290 ~ 0x2AC	Filter Coefficient 0~15 Register	0x0000 0000

1. 滤波器控制寄存器 (FILTER_CR, address 0x5007 0280)

位	名称	值	描述	初始值
15:0	SHF_DSRCSEL		定义各延迟点的输入来源	0000
		0	D[n] 的数据从上一个 D[n-1]移入	
		1	D[n] 的数据从滤波器的输出口移入	
19:16	FILTER_CYC		滤波器乘加次数设定. 0 表示做 1 次乘加后就输出 ~ 0xf 表示做 16 次乘加后输出. 如果配置成 FIR 滤波器, 这里应写入滤波器阶数-1。如果配置为 IIR 滤波器, 这里是分子项和分母项之和。	0
21:20	FILTER_MODE		滤波器工作模式	00
		00	软件模式：原始数据由软件写入，结果由软件取走。	
		01	半自动 DMA 模式：软件写入原始数据后，滤波器自动启动完成计算，随后数据由 DMA 自动取走，滤波器在计算完成后进入等待状态。半自动模式需要预先配置好 DMA。	
		10	全自动 DMA 模式：滤波器自动通过 DMA 取得数据，计算完成后再通过 DMA 将数据取走。xDSP 将自动重复以上步骤直到 XDSP0_DMA_info 里定义的次数全部完成.	
		11	保留	
23:22	-	-	保留.	0
24	FILTER_CLR		滤波器清零	0
		0	无效.	
		1	清除延迟线，累加器	
26:25	-	-	保留.	0
31:27	OUTPUT_SHIFT		输出移位控制. 滤波器按照设定右移一定位数并钳位。得到的数据将被写入 FILTER_RESULT 寄存器，如果是 IIR 滤波器的话还将进入反馈通路。	0

2.数据输入寄存器 (FILTER_D, address 0x5007 0284) bit description

位	名称	描述	初始值
15:0	FILTER_D	输入数据寄存器.	0

31:16	-	保留	0

3. 滤波启动寄存器 (FILTER_START, address 0x5007 0288) bit description

位	名称	描述	初始值
0	FILTER_START	滤波运算启动。写 1 启动，写 0 无效。	0
31:1	-	保留	0

4. 结果寄存器 (FILTER_RESULT, address 0x5007 028c) bit description

位	名称	描述	初始值
15:0	FILTER_RESULT	滤波运算结果.	0
16	FILTER_STAT	滤波运算器状态，该位只读。 0 忙 1 空闲	1
31:16	-	保留	0

5. 参数寄存器 (FILTER_Cn, address 0x5007 0290 ~ 0x5007 02ac) bit description

位	名称	描述	初始值
15:0	FILTER_C_{2n}	滤波参数 2n	0
31:16	FILTER_C_{2n+1}	滤波参数 2n+1	0

FIR 滤波器的配置和使用

FIR 滤波器的结构相对比较简单，只需配置 FILTER_CYC 位就能确定滤波器的阶数。

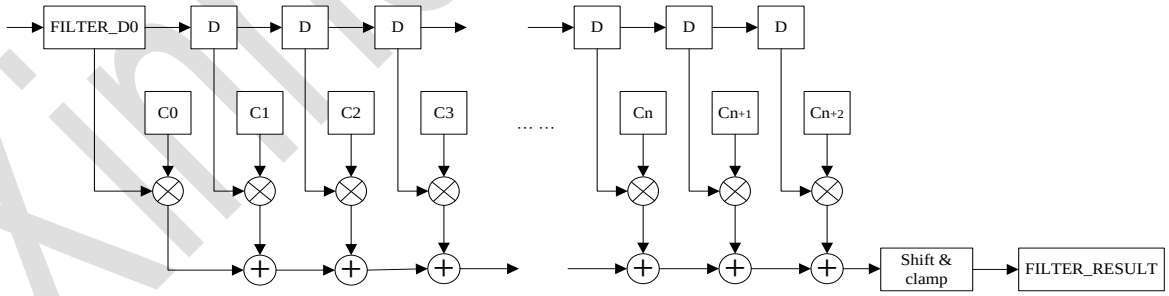


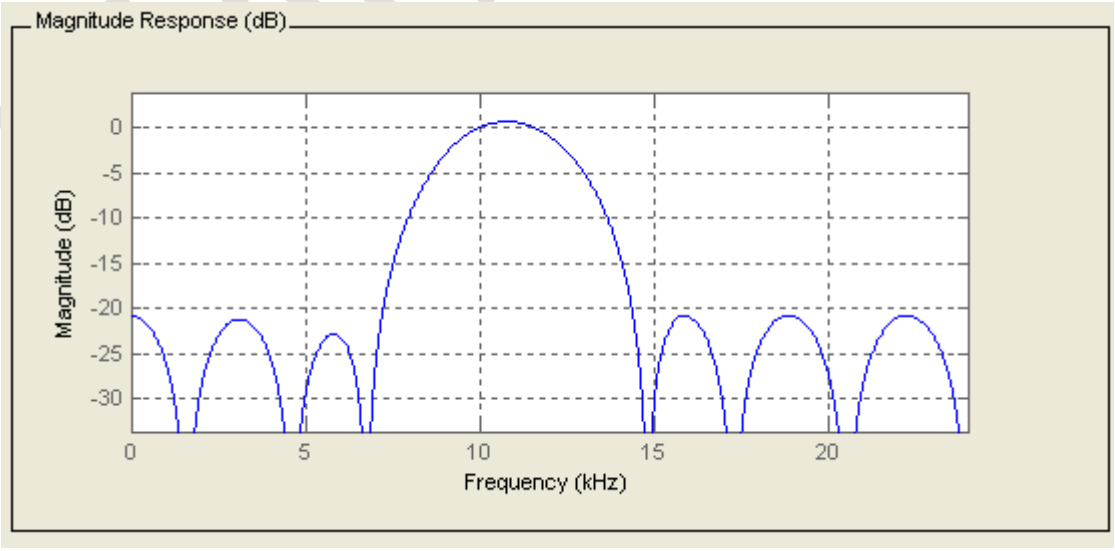
图 2 FIR 滤波器

例如，设计一个 FIR 带通滤波器，采样频率 48k，阻带 1 的频率 7.2k，通带 1 的频率 9.6k，阻带 2 的频率 14.4k，通带 2 的频率 12k。在 matlab 上可得到这样一组参数：

```
Numerator:

-0.01207850498523266

-0.094245527357469536
```

```
 0.010124500469595135

 0.12546040455577073

 0.034165372769331549

-0.16149871607240748

-0.098493055598291204

 0.15098231556468883

 0.15098231556468883

-0.098493055598291204

-0.16149871607240748

 0.034165372769331549

 0.12546040455577073

 0.010124500469595135

-0.094245527357469536

-0.01207850498523266
```

其频响特性如图：



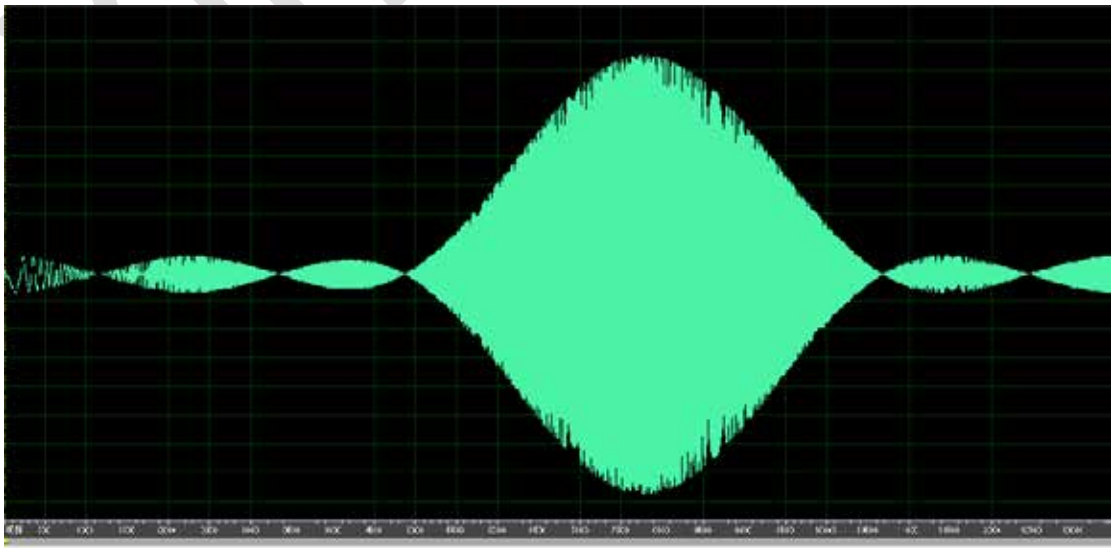
将其整型化（*1^15），得到

Fir_coef[]={-396，-3088，332，4111，1120，-5292，-3227，4947，4947，-3227，-5292，1120，4111， 332，-3088，-396};

于是程序可以写成这样：

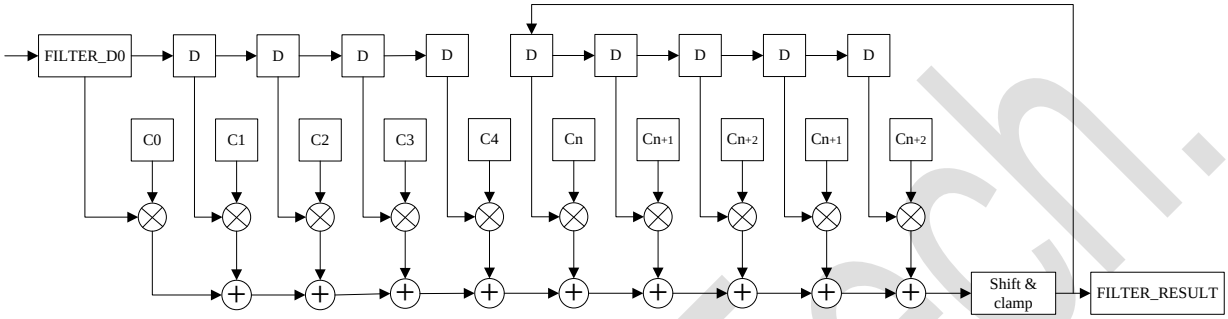
```
XDP_XDSP->FILTER_C01=(unsigned short)(fir_coef[0]&0xffff)+(fir_coef[1]<<16);

XDP_XDSP->FILTER_C23=(unsigned short)(fir_coef[2]&0xffff)+(fir_coef[3]<<16);

XDP_XDSP->FILTER_C45=(unsigned short)(fir_coef[4]&0xffff)+(fir_coef[5]<<16);

XDP_XDSP->FILTER_C67=(unsigned short)(fir_coef[6]&0xffff)+(fir_coef[7]<<16);

XDP_XDSP->FILTER_C89=(unsigned short)(fir_coef[8]&0xffff)+(fir_coef[9]<<16);

XDP_XDSP->FILTER_CAB=(unsigned short)(fir_coef[0xa]&0xffff)+(fir_coef[0xb]<<16);

XDP_XDSP->FILTER_CCD=(unsigned short)(fir_coef[0xc]&0xffff)+(fir_coef[0xd]<<16);

XDP_XDSP->FILTER_CEF=(unsigned short)(fir_coef[0xe]&0xffff)+(fir_coef[0xf]<<16);

XDP_XDSP->FILTER_CR = 1<<24;       //clear delay chain and accumulator

XDP_XDSP->FILTER_CR = 0;

XDP_XDSP->FILTER_CR=(15<<16)+(16<<27);
```

输入一个采样频率 48k，信号频率从 0 到 20k 的扫频信号，可以得到如图所示的输出信号。从图中可见高频和低频部分被充分抑制，通带部分的信号则得到保留。



IIR 滤波器的配置和使用

IIR 滤波器是一种有反馈的滤波器，可以理解为输出信号在某一点上进入延迟链。其结构如图：



设置 IIR 滤波器时，首先要设 SHF_DSRCSEL。它表示在延迟队列中从哪一级开始，shift in 的数据来自输出反馈。其次是设置 FILTER_CYC，它表示要做多少次乘加运算。例如，要计算一个二阶滤波器 Y=A0*X(n)+A1*X(n-1)+A2*X(n-2)-B0*Y(n-1)-B1*Y(n-2)。则 SHF_DSRCSEL=0x8(1<<3)；FILTER_CYC=4。

举例来说，使用一个隔直滤波器 $H(z)=\frac{1-z^{-1}}{1-0.9995z^{-1}}$，它的计算式是 Y=X-X(n-1)+0.9995*Y(n-1)。于是程序应该写成如下形式：

```
XDP_XDSP->FILTER_C01=0x0001ffff;

XDP_XDSP->FILTER_C23=0x7fef0000;

XDP_XDSP->FILTER_C45=0;

XDP_XDSP->FILTER_C67=0;

XDP_XDSP->FILTER_C89=0;

XDP_XDSP->FILTER_CAB=0;

XDP_XDSP->FILTER_CCD=0;

XDP_XDSP->FILTER_CEF=0;

XDP_XDSP->FILTER_CR = 1<<24;

XDP_XDSP->FILTER_CR = 0;

XDP_XDSP->FILTER_CR=(1<<2)+(2<<16)+(16<<27);
```

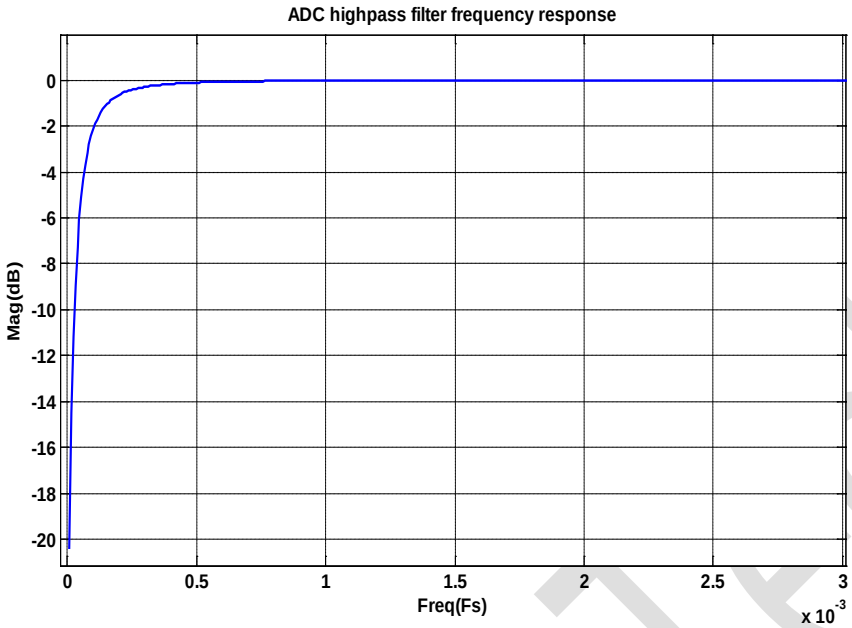


图 4. 隔直滤波器频响特性

使用 IIR 滤波器时要注意反馈通道的数据位数，一旦饱和的 sample 过多，会显著影响输出信号的质量。另外，由于参数和延迟链都是 16 位的，高阶的 IIR 滤波器配置得不合适也会显著影响信号质量。

DMA 与滤波器

滤波器通常会被用来大批量处理数据，甚至应用于需要较好的实时性的场合，比如对 ADC 采样数据进行平滑等处理。为此, XN62Lxxx 可以使用 DMA 来支持滤波器。有两种方式使用 DMA。一种是半自动模式, CPU 通过指令把数据写入 FILTER_D 寄存器，滤波结束后，滤波器通知 DMA 将结果放到指定位置。这种模式还可以是其它外设主动发起 DMA 请求，DMA 把数据送到 FILTER_D，完成计算后再由滤波器的 DMA 把结果送到指定地方。一种比较特色的用法是 ADC->滤波器->DAC。在某些特殊的应用场合下，它会起到很好的效果。

另一种是全自动模式，主要用于对内存中的批量数据进行处理。滤波器主动发起 DMA 请求，DMA 控制器将数据写入 FILTER_D，滤波计算结束后，滤波器再次发出请求，DMA 将结果取出，如此循环。

滤波器请求 DMA 写入数据的 DMA 通道号是 20，请求 DMA 取走结果的 DMA 通道号是 21。

例1， 半 DMA 模式

```
void test_halfDMA_FIR()

{

  char i;
```

```
    short dstbuf[10];

    XDP_DMA->CFG=1;       //enable controller

    XDP_DMA->CTRL_BASE_PTR =    (unsigned int)chn_ctrl_struct;

//源地址

DMA_XDSP1_info->SRC_END_PTR=(unsigned int)&XDP_XDSP->FILTER_RESULT;

//目的地址

DMA_XDSP1_info->DES_END_PTR=(unsigned int)&dstbuf[9];

//进行 10 次半字节 DMA 传输，源地址不变，目的地址按半字节递减

  DMA_XDSP1_info->CHANNEL_CTRL_DATA=(1<<CHNL_CFG_CYCLE_CTRL)                    \
     +(0<<CHNL_CFG_NEXT_USEBURST)+(10<<CHNL_CFG_TRANS)                          \
     +(0<<CHNL_CFG_R_POWER)+(1<<CHNL_CFG_SRC_SIZE)                              \
     +(3<<CHNL_CFG_SRC_INC)+(1<<CHNL_CFG_DST_SIZE) \
     +(1<<CHNL_CFG_DST_INC);


    XDP_DMA->CHNL_ENABLE_SET=0x00100000;

    XDP_XDSP->FILTER_C01 = xxxxxxxx;

    XDP_XDSP->FILTER_CR = (1<<16)+(1<<20);

    for(i=0;i<10;i++)

    {

       XDP_XDSP->FILTER_DR = 0x10+i;

       XDP_XDSP->FILTER_START=1;

       while((XDP_XDSP->FILTER_RESULT&0x10000)==0);

    }

    for(i=0;i<10;i++)
```

```
        printf(" %x\n",dstbuf[i]);
    }
```

例2,　　全自动模式

```
    void test_autoDMA_FIR()
    {
       char i;
       char        srcbuf[6]={0x10,0x20,0x30,0x40,0x50,0x60};
    //    short srcbuf[6]={0x100,0x200,0x300,0x400,0x500,0x600};

       short dstbuf[6];

         XDP_DMA->CFG=1;       //enable controller
       XDP_DMA->CTRL_BASE_PTR =    (unsigned int)chn_ctrl_struct;

       DMA_XDSP0_info->SRC_END_PTR=(unsigned int)&srcbuf[5];
       DMA_XDSP0_info->DES_END_PTR=(unsigned int)&XDP_XDSP->FILTER_DR;
        DMA_XDSP0_info->CHANNEL_CTRL_DATA=(1<<CHNL_CFG_CYCLE_CTRL)     \
           +(0<<CHNL_CFG_NEXT_USEBURST)+(5<<CHNL_CFG_TRANS)                                    \
           +(0<<CHNL_CFG_R_POWER)+(0<<CHNL_CFG_SRC_SIZE)                                       \
           +(0<<CHNL_CFG_SRC_INC)+(0<<CHNL_CFG_DST_SIZE)                    \
           +(3<<CHNL_CFG_DST_INC);

       DMA_XDSP1_info->SRC_END_PTR=(unsigned int)&XDP_XDSP->FILTER_RESULT;
       DMA_XDSP1_info->DES_END_PTR=(unsigned int)&dstbuf[5];
```

```
        DMA_XDSP1_info->CHANNEL_CTRL_DATA=(1<<CHNL_CFG_CYCLE_CTRL)     \
          +(0<<CHNL_CFG_NEXT_USEBURST)+(5<<CHNL_CFG_TRANS)              \
        +(0<<CHNL_CFG_R_POWER)+(1<<CHNL_CFG_SRC_SIZE)                                   \
        +(3<<CHNL_CFG_SRC_INC)+(1<<CHNL_CFG_DST_SIZE)                   \
        +(1<<CHNL_CFG_DST_INC);


    XDP_DMA->CHNL_ENABLE_SET=0x00300000;
    XDP_XDSP->FILTER_C01 = 0x10001;
    XDP_XDSP->FILTER_CR = (1<<16)+(2<<20);
    XDP_XDSP->FILTER_START = 1;


    while((XDP_XDSP->FILTER_RESULT&0x10000)==0);
    for(i=0;i<6;i++)
       printf(" %x\n",dstbuf[i]);
}
```