ANALOG DEVICES

AN-543
应用笔记

One Technology Way • P.O. Box 9106 • Norwood, MA 02062-9106 • 781/329-4700 • World Wide Web Site: http://www.analog.com

利用ADSP-2181 DSP和AD9850直接数字频率合成器产生高质量、全数字RF频率调制

作者：Dean R. Becker

本文将描述针对音频输入信号的一种高质量全数字FM RF发生器的实现方式。FM RF信号输出位于广播FM频带，可能是单声道或立体声。

自艾德温•H•阿姆斯特朗(Edwin H. Armstrong)少校(也称为FM之父)发明调频无线电及其在1936年发表经典调频论文以来，我们就有了FM调制器。他是将固定中心频率(根据独立于调制信号频率的恒定偏移系数而变化)这一基本概念撰写成文的第一人。FM调频逐渐受到广泛推崇，因为自然界产生的噪声表现出AM特性。我们将发现，受偏移系数相关改比的影响，FM信号的信噪比超过AM信号。众所周知，相对于AM广播，宽频FM广播频带拥有出众的音质和卓越的抗扰度。

模拟FM调制器一般采用调谐电路中搭载变容二极管的振荡器。该变容二极管的电压因调制信号而异，而调制信号的电容将发生变化，从而改变振荡器调谐电路的谐振点，结果产生频率偏移。如前所述，不变中心频率和偏移系数是高品质FM的基本要素。为了获得必要的稳定性，研究者们做了大量工作以改进基本振荡器电路。尽管可以增加复杂电路以自动补偿电源和温度变化，但随着元件的老化，此类电路仍需要定期进行再校准。

数字信号处理(DSP)器件领域的最新研究成果使构建基于全数字电路的高品质FM调制方案成为可能。这种调制器不需要定期进行再校准，不会随温度或电源变化而漂移，而且由于不存在模拟调节(除输入模拟调制信号的电平以外)，很容易再现。这种电路的核心是直接数字频率合成器(DDS)，如AD9850完整DDS(C-DDS)器件。AD9850包括一个数字相位累加器、一个相位/幅度转换器和一个数模转换器。结果在相位累加器中形成线性斜坡，其频率取决于相位累加器的输入值。该斜坡被相位转幅度转换器映射到采样正弦信号上。在此基础上，采样数字信号进行数模转换，并通过一个重构滤波器进行滤波处理，以形成模拟波形。DDS的所有功能(除模拟重构滤波器以外)均包含于AD9850芯片之中。采样信号以125 MHz采样速率生成，支持高频RF输出。在AD9850面市之前，市场上根本不存在极速型低成本C-DDS器件。有关C-DDS器件的工作原理和规格参数的更多详情，请参阅AD9850数据手册的工作原理部分。

硬件实现方式

FM RF发生器的实现过程需要用到ADI EZ-KIT Lite 16位DSP开发板、一些I/O解码电路和DDS评估板——AD9850-FSPCB，如图1所示。

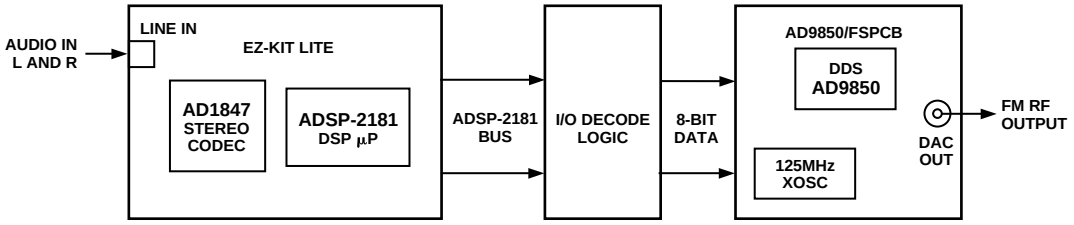

图1. FM RF发生器实施功能框图

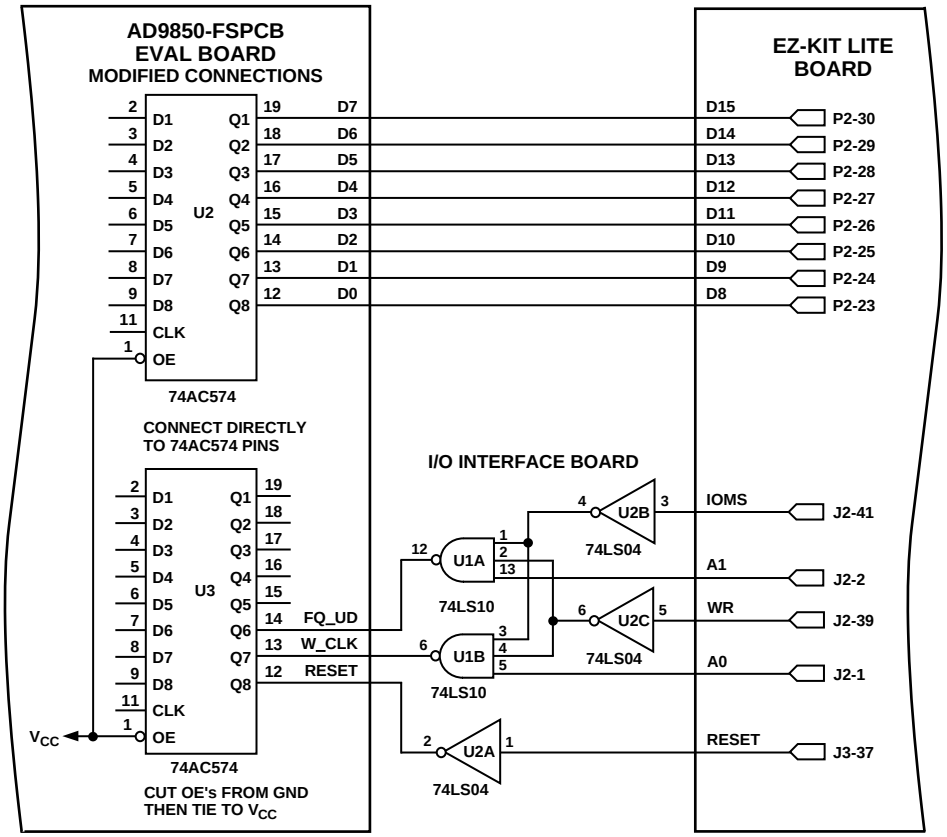

图2. I/O解码逻辑和布线原理图

EZ-KIT Lite包含一款ADI ADSP-2181 KS-133 16位DSP，搭载16K × 24程序存储器RAM和16K × 16数据存储器RAM。

有关硬件文档随EZ-KIT Lite开发板提供，AD9850 C-DDS原理图见AD9850数据手册。图2显示了I/O解码逻辑和布线。

AD9850-FSPCB评估板旨在与PC并口接口。该评估板采用并口连接器，配有并口缓存。将其连接至外置DSP的一种简便办法是禁用缓存，如原理图所示：先断开输出使能与PCB地平面的连接，然后将其连接至V_{CC}。在禁用缓存的情况下，EZ-KIT Lite搭载的ADSP-2181可以直接驱动AD9850数据线路。原理图中的I/O接口对I/O地址1进行解码，产生W_CLK(写时钟)信号；同时对I/O地位2进行解码，产生FQ_UD(频率更新)信号。这些控制信号用于将连续频率字节写入AD9850，然后将整个32位频率字分别选通写入操作寄存器。RESET用于初始化AD9850。有关加载C-DDS的更多信息，请参阅AD9850数据手册。

DSP固件实现方式

这里将描述两个程序。第一个针对单声道FM调制器(fm_xmit.dsp)，第二个针对立体声(fmStereo.dsp)。两个程序都是独立的单模块，可利用EZ-KIT Lite监控器通过串口载入EZ-KIT Lite。两者的开头都是编解码器变量初始化。唯一差别在于，fm_xmit的编解码器采样速率为48 kHz，FmStereo则为44.1 kHz。(说明请参看FmStereo程序)。该部分(初始化代码、中断矢量、DSP初始化和编解码器初始化)复制自EZ-KIT Lite自带的示例程序。图3仅列出了部分内容，描述了编解码器的每个输入采样发生的情况。首先，左右输入采样被除以2，以确保二者相加时不产生溢出。然后，用前面得到的和乘以偏差值，再对所得乘积加上一个中心频率值。以上均基于32位双精度，以确保C-DDS的精度。最后，32位频率字被分解成4个字节，并被发送至AD9850 C-DDS。

```
.const      w_clk=      0x0001;     { 9850 write clock address }
.const      fq_ud=      0x0002;     { 9850 frequency update address }

input_samples:
ena sec_reg;            { use shadow register bank }
mr0 = dm (rx_buf + 1);    { Left data in }
mr1 = dm (rx_buf + 2);    { Right data in }
{ loopback inputs to outputs }
sr = ashift mr0 by -1 (hi); { scale left input by 0.5 }
af = pass sr1;          { setup for add }
sr = ashift mr1 by -1 (hi); { scale right input by 0.5 }
ar = sr1 + af;          { add left and right together }

dm (tx_buf + 1) = ar;     { Left data out }
dm (tx_buf + 2) = ar;     { Right data out }
{ same output on both channels }

mx1 = 0x004e;                       { MSW of peak deviation value }
mx0 = 0xa4a8;                       { LSW of peak deviation value }
my1 = ar;                   { L+R codec value (MSW only, LSW=0) }
{ 32 bit mpy with one of the LSWs=0 }
mr = mx0 * my1 (us);      { codec MSW * deviation LSW }
mr0 = mr1;                  { shift right by 16 to align with next }
mr1 = mr2;
mr = mr + mx1 * my1 (ss); { mpy MSWs and add previous product }
ax1 = 0x4710;                       { MSW of DDS center frequency }
ax0 = 0xcb29;                       { LSW of DDS center frequency }
ay1 = mr1;              { MSW of modulated deviation }
ay0 = mr0;              { LSW of modulated deviation }
ar = ax0 +ay0;                      { add LSWs first }
mr0 = ar, ar=ax1 + ay1 + c; { save result LSW and get sum of MSWs }

sr0 = 0;                { DDS phase number = 0 }
io(w_clk) = sr0;        { output first byte to DDS }
sr = lshift ar by -8 (hi); { move MSB of MSW to align with D8-D15 }
io(w_clk) = sr1;        { output second byte to DDS, MSB of MSW }
nop;
io(w_clk) = ar;         { output third byte to DDS, LSB of MSW }
sr = lshift mr0 by -8 (hi); { move MSB of LSW to align with D8-D15 }
io(w_clk) = sr1;        { output forth byte to DDS, MSB of LSW }
nop;
io(w_clk) = mr0;        { output fifth byte to DDS, LSB of LSW }
nop;
io(fq_ud) = sr0;        { output latch pulse, data is irrelevant }

rti;
```

图3. 单声道FM代码的部分内容

该信号可通过常规的FM收音机(单声道)接收到，频率为90.3 MHz。C-DDS的f_S为125 MHz，因此，编程频率为34.7 MHz时，其第一混叠频率为90.3 MHz。在采样sin(x)/x曲线的影响下，第一混叠频率将衰减9.4 dB，但仍有充足的信号来驱动FM收音机。(混叠这一概念详见AD9850数据手册中的图5。)请注意，不能在C-DDS输出端放置奈奎斯特截止低通滤波器，否则会阻断第一混叠。须采用88 MHz至108 MHz的带通滤波器来过滤掉基频和其他混叠。

34.7 MHz下，C-DDS中心频率的32位值为4710CB29H。75 kHz峰值偏差等于00275254H的C-DDS偏移。当作偏差值的数字为该值的两倍，以补偿输入总和被除以2以避免溢出这一事实。这些常数通过以下等式求得：

$$值 = f/(125\ MHz/2E32) = 34.3597 \times f\ (f_S = 125\ MHz)$$

发射中心频率值和偏差值可以更改，也可存储在数据存储器中，程序中可以当作变量使用。然后可以根据需要通过I/O开关或串口进行动态更改。

将I/O发送至AD9850时，通过数个等待状态来补偿解码逻辑。AD9850 I/O之间至少需要一个时钟周期，可通过必要的指令或nop命令实现。DDS的第一个字节为相位字节，始终为0。其后的频率字节依次按照MSB到LSB的顺序输出。

立体声实现方式

实际上，FM单声道的实现非常简单。然而，构建FM立体声信号却稍显复杂。首先，不用更改或添加硬件。对于立体声，先构建三种信号并求和，然后利用其对DDS进行FM调制。第一个信号与单声道条件下相同，即左右声道之和。第二个信号为19 kHz的导频音，其电平为最大信号的9%。第三个信号是左声道减去右声道之差，经过双边带抑制，并以38 kHz 音进行载波调制处理。该38 kHz音在19 kHz导频音的每次零交越时均须有一个上升沿。

这些信号的多路复合产物的频率可能扩展至53 kHz(假设输入信号上升至15 kHz)。显然，如果保留Fm_xmit中使用的48 kHz采样速率，则复合音将与自身相混叠。我们选择了44.1 kHz的编解码器采样速率，并插入了两个间距相等的采样。最后得到的采样速率为132.3 kHz。两个“额外”采样通过ADSP-2181定时器和一些合理代码创建。该采样速率为创建19 kHz和38 kHz音的数控振荡器(NCO)提供几近精准的16位相位输入值。图4所示汇编代码为立体声FM发生器的部分代码。其中省略了单声道FM发生器的初始化代码和冗余代码。

```
ax0 = 0;
dm(TSCALE) = ax0;          { no pre-scalar }
ax0 = 251;
dm(TPERIOD) = ax0;         { auto load for third interrupt }
dm(TCOUNT) = ax0;

ax0 = 0;
dm(nco_19k) = ax0;          { Set accumulators. Phase of the 38 kHz NCO }
dm(nco_38k) = ax0;          { must be such that the 38 kHz has a rising }
                            { edge on every zero crossing transition of }
                            { the 19 kHz pilot tone. }
reset fl1;

{-------------------------------------------------------------
 -
 -  wait for interrupt and loop forever
 -
 -------------------------------------------------------------}

main: idle;    { wait for an interrupt }
call dds_out; { output previous sample to DDS }
                            { putting this here gives a constant update }
                            { position for the DDS with no jitter }
ax0 = dm(sample_flag); { get the flag that was set during intr }
ar = tstbit 0 of ax0;       { 1=codec intr, 0=timer intr }
if eq jump timer_intr;      { skip the codec part if 0 }

input_samples:
{ IMPORTANT!... If you change the number of instructions up to the }
{ ena timer, you need to adjust the following TCOUNT accordingly. }
ax0 = 188;      { to provide the next intr @ 44.1kHz x 3 }
dm(TCOUNT) = ax0;          { this takes previous instructions and }
                            { interrupt latency into account }
ena timer;                  { the next two intr's are gen by timer }
ax0 = dm(rx_buf + 1); { Left data in }
dm(left) = ax0;
dm(tx_buf + 1) = ax0; { Left out loopback}
ax0 = dm(rx_buf + 2); { Right data in }
dm(right) = ax0;
dm(tx_buf + 2) = ax0; { Right out loopback}
ax0 = 3;                    { first of three interrupts in group }
dm(sample_number) = ax0;   { initialize count }

timer_intr:
ax0 = dm(sample_number);   { 2 timer interrupts after codec }
ar = ax0 - 1;       { decrement count }
dm(sample_number) = ar;    { save count for 3rd interrupt in group }
if ne jump ti1;             { exec on second interrupt of group }
dis timer;                  { after third interrupt, wait for codec }

                { 19 kHz generator }
ti1:            ax0 = 0x24c4;               { NCO phase increment for 19 kHz }
                ay0 = dm(nco_19k);     { get 19 kHz NCO phase to update }
                ar = ax0 + ay0;             { add phase step for 19 kHz }
                dm(nco_19k) = ar;           { put accumulation back into NCO }
                ax0 = ar;                   { prep for sine routine }
                call sin;                   { get sine }
                dm(sin_19k) = ar;           { save for later }

                { 38 kHz generator }
                ax0 = 0x4988;               { NCO phase increment for 38 kHz }
                ay0 = dm(nco_38k);    { get 38 kHz NCO phase to update }
                ar = ax0 + ay0;             { add phase step for 38 kHz }
                dm(nco_38k) = ar;           { put accumulation back into NCO }
                ax0 = ar;                   { prep for sine routine }
                call sin;                   { get sine }
                dm(sin_38k) = ar;           { save for later }

                { generate the L+R and L-R signals }
    ax0 = dm (left);    { Left data in }
    ay0 = dm (right);   { Right data in }
                ar = ax0 + ay0;
                if av set fl1;      { input overdrive indicator }
                dm(LplusR) = ar;
                ar = ax0 - ay0;
                if av set fl1;      { input overdrive indicator }
                dm(LminusR) = ar;           { L-R remains in ar }

                { generate the multiplex modulating signal }
                mr2 = 0;
                mr0 = 0;
                mr1 = dm(LplusR);                       { baseband signal in mr }
                my0 = dm(sin_38k);          { 38 kHz carrier }
                mr = mr + ar * my0 (ss);   { DSB L-R with 38 kHz }
                mx0 = dm(sin_19k);          { 19 kHz carrier }
                my0 = 0x0b85;               { factor to get 9% of 19 kHz carrier }
                mr = mr + mx0 * my0 (ss);  { add in 9% of 19 khz pilot tone }
                dm(dds_samp) = mr0;         { save result LSW for next iteration }
                dm(dds_samp+1) = mr1; { save result MSW for next iteration }
                if mv set fl1;     { input overdrive indicator }
                                    { reset by pressing interrupt (IRQE) }
                jump main;      { loop forever }


{-------------------------------------------------------------
 -
 -  The following outputs the previously calculated sample to the DDS
 -  Execution time is 31+(6*W), with W=3; =49
 -
 -------------------------------------------------------------  ------}
dds_out:
                { create 32 bit deviation value }
                mx1 = 0x0027;                           { MSW of peak deviation value }
                mx0 = 0x5254;                           { LSW  of peak deviation value }
                my1 = dm(dds_samp+1);   { L+R, (L-R)@38K, & 19k }
                my0 = dm(dds_samp);      { 32 bit mpy }
                mr = mx0 * my0 (uu);       { mpx LSW * deviation LSW }
                mr0 = mr1;    { shift right by 16 to align with next }
                mr1 = mr2;
                mr = mr + mx1 * my0 (su);{ mpx LSW * deviation MSW }
                mr = mr + mx0 * my1 (us);{ mpx MSW * deviation LSW }
                mr0 = mr1;    { shift right by 16 to align with next }
                mr1 = mr2;
                mr = mr + mx1 * my1 (ss);{ mpy MSWs and add previous product}
                { add the 32 bit center frequency }
                ax1 = 0x4710;               { MSW of DDS center frequency }
                ax0 = 0xcb29;               { LSW of DDS center frequency }
                ay1 = mr1;                  { MSW of modulated deviation }
                ay0 = mr0;                  { LSW of modulated deviation }
                ar = ax0 +ay0;              { add LSWs first }
                mr0 = ar, ar=ax1 + ay1 + c; { save result LSW and get sum of MSWs }

                { output to the DDS}
                sr0 = 0;                    { DDS phase number = 0 }
                io(w_clk) = sr0;            { output first byte to DDS }
                sr = lshift ar by -8 (hi);{ move MSB of MSW to align with D8-D15 }
                io(w_clk) = sr1;            { output second byte to DDS, MSB of MSW }
                nop;
                io(w_clk) = ar;             { output third byte to DDS, LSB of MSW }
                sr = lshift mr0 by -8 (hi); { move MSB of LSW to align with D8-D15 }
                io(w_clk) = sr1;            { output forth byte to DDS, MSB of LSW }
                nop;
                io(w_clk) = mr0;            { output fifth byte to DDS, LSB of LSW }
                nop;
                io(fq_ud) = sr0;            { output latch pulse, data is irrelevant }
    rts;
```

图4. 立体声FM发生器的部分汇编代码

代码开头，ADSP-2181内部定时器被设置为不采用预分频器，并在上一个定时器中断后经过251个时钟周期后中断。结果在每个编解码器中断之间设置了两个定时器中断。详见后文。针对内部产生的19kHz和38 kHz载波的固件NCO也被初始化，以使其相位正确，详见前文说明。然后，程序进入每个中断执行一次的主循环。该循环首先调用DDS输出程序。通过在中断后的同一点调用DDS输出程序，结果不会在输出信号上产生抖动。该子程序与fm_xmit中的程序相似，只不过运行于多路复用发生器产生的全32位采样。在fm_xmit程序中，输入采样仅为16位。偏差值0x00275254相当于75 kHz。DDS输出程序其余部分的工作方式与fm_xmit相同。

在此基础上，程序通过一个在中断程序中设定的标志(sample_flag)来确定中断是由编解码器还是定时器所致。如果中断来自编解码器，则定时器将被初始化并开始创建下一个中断。该过程在计数188之后执行，以补偿在中断之后执行的指令。结果使中断相隔251个周期。请注意，该定时器值的选择对创建等距交错中断至关重要。此时，从其DM位置检索左右输入信号(输出信号则从编解码器中检索，以进行完整性检查)，采样数设为3，以启动中断交错过程。

接下来，程序进入一般管理函数，以创建定时器中断。Sample_number被减至2。下一次通过该程序时(由第一个定时器中断导致)，sample_number将被减至1，定时器自动载入。第三次通过时，sample_number将被减至0，定时器关闭。然后，从来自编解码器的下一个中断重新开始循环。

后面的代码由所有中断执行。首先通过增量19 kHz NCO创建19 kHz载波，然后利用正弦程序(来自ADI应用手册第一卷)创建正弦载波。该程序利用一个功率系列来逼近NCO相位角的正弦。然后以相同方式产生38 kHz载波，只是其NCO相位增量相当于19 kHz发生器的2倍。之后算出左右声道信号的和与差，并进行溢出检查。若存在溢出，则EZ-KIT Lite板上的红色LED指示灯点亮。必须按下中断按钮，才能手动关掉该指示灯。

下一步产生多路复用信号。首先，再次调用左右信号之和，并与左右之差乘以38 kHz载波之积相加。该积为差动信号形成约38 kHz的DSB调制。该和再与19 kHz信号乘以常数0x0b85之积相加，结果得到9%的19 kHz参考载波。然后将多路复用采样存储起来，以在下次中断后输出，循环返回空闲状态，等待下一个中断。

这个程序是多速率处理的示例。显示了ADSP-2181强大的功能和AD9850巨大的灵活性。即使有了FM立体声多路复用信号必需的额外代码，仍然存在提升空间。其中一种可能是在立体声多路复用信号上添加SCA多路复用信号，就如众多FM广播所做的那样。为此，必须进一步提高采样速率，并以类似于19 kHz和38 kHz载波的方式创建SCA的载波。

动态性能

图5显示了AD9850直接DAC输出的杂散性能。首先，实验电路板原型上未设置屏蔽，因此有可能拾取噪声。首先需要注意的是，125 MHz以及该频率一半情况下似乎存在时钟馈通。当工作频率接近40 MHz时，AD9850的预期杂散为–45 dBc。从图5可以看出，情况正是如此。

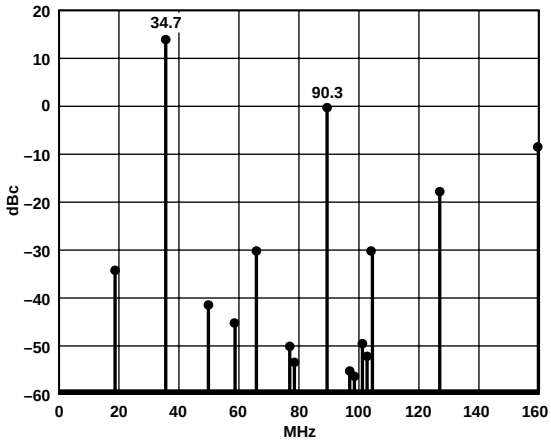

图5. 典型DDS输出杂散性能

请注意，90.3 MHz下的第一混叠约比基频30.4 MHz约低12 dB。该值接近于从前面提到的(sin x)/x曲线计算而得的值。相对于90.3 MHz信号，杂散仅低30 dB，但杂散信号至少相隔15 MHz，并可通过常规方式进行滤波处理。研究发现（图中未显示），从90.3 MHz开始存在众多增量为48 kHz的杂散，范围为±300 kHz。相对于90.3 MHz信号，其值为–55 dB。从该范围之外到始于90.3 MHz的第一个杂散(相距7 MHz)，所有能量均比90.3 MHz信号低60 dB。

作为参考，1982年版的《无线电工程师参考数据》在第30-6页中表示，相对于带外辐射，FM广播站的可接受性能为距离载波120 kHz至240 kHz，辐射为–25 dBc。从240 kHz到600 kHz，该值为–35 dBc，超过600 kHz，则为–80 dBc或43 + 10 log P，其中，P为发射机功率(单位：W)。鉴于上述指标，似乎常规的集总元件滤波即可消除AD9850输出信号中的噪声，使其达到极致广播质量。另一种可能是采用SAW滤波器；鉴于AD9850提供的高电平输出，可以对SAW中的损耗进行补偿，而不影响SNR。