基于FPGA的FIR低通滤波器设计总结报告

1.1数字滤波器简介

1.1.1 数字滤波的应用意义

由于计算机技术、集成工艺和材料工业的发展，滤波器的发展上了一个新台阶，朝着功耗小、精度高、体积小、功能全面、灵活性好、稳定可靠并且成本低廉等方向发展，目前数字滤波器、RC有源滤波器、开关电容滤波器和电荷转移器等各种滤波器的飞速发展。

在数字表示中，精度可以通过改变信号的字长来更好地控制，所以数字信号处理(Digital signal Processing，DSP)技术可以在放大信号的同时去除噪声和干扰，而在模拟信号中信号和噪声同时被放大，数字信号还可以不带误差地被存储和恢复、发送和接收、处理和操纵。信号处理的目的一般是对信号进行分析、变换、综合、估值与识别等。如何在较强的噪声背景下提取出真正的信号或信号的特征，并将其应用于工程实际是信号处理的首要任务。数字信号处理中一个非常重要且应用普遍的技术就是数字滤波。

所谓数字滤波器，是指输入、输出均为数字信号，通过一定运算关系改变输入信号所含频率成分的相对比例或者滤除某些频率成分的器件。达到提取和加强信号中的有用成份，消弱无用的干扰成份的目的。数字滤波器可广义地理解为一个信号选择系统。数字滤波器由加法器、乘法器、存储延迟单元等数字电路构成。与模拟滤波器相比，数字滤波器有精度高、稳定、可靠性高、体积小、重量轻、灵活性高、不要求阻抗匹配以及实现模拟滤波器无法实现的特殊滤波功能等优点，可以满足对幅度和相位的严格要求，还能降低开发费用，缩短研制到应用的时间。

数字滤波器的功能就是把输入序列通过一定的运算变换成输出序列。其实现可以采用两种方法：软件实现和硬件实现。软件实现即是在通用的微型计算机上用软件来实现。利用计算机的存储器、运算器和控制器把滤波器所要完成的运算编成程序通过计算机来执行，软件可由使用者自己编写，也可使用现成的。硬件实现即是设计专用的数字滤波硬件，目前主要采用单片机(Micro Controller Unit， MCU)、数字信号处理器(Digital Signal Processing，DSP)、ASIC和FPGA来实现。

一个数字滤波器可以用系统函数表示为：



直接由此公式可得出表示输入输出关系的常系数线性差分方程：



由式(2)可以看出，数字滤波器的功能就是把输入序列通过一定的运算变换成输出序列。

1.1.2 FIR滤波器简介

滤波器根据功能的不同可分为低通滤波器、高通滤波器、带通滤波器和带阻滤波器四种;根据设计方法的不同可分为切比雪夫滤波器和巴特沃斯滤波器;根据处理信号的不同可以分为数字滤波器和模拟滤波器;从单位脉冲响应上分类，可以分为无限脉冲响应（Infinite Impulse Filter, IIR）滤波器和有限脉冲响应（Finite Impulse Filter, FIR）滤波器两大类；根据设计选择的不同，可以分为时域设计法和频域设计法，如图所示。（本项目着重介绍FIR低通滤波器）



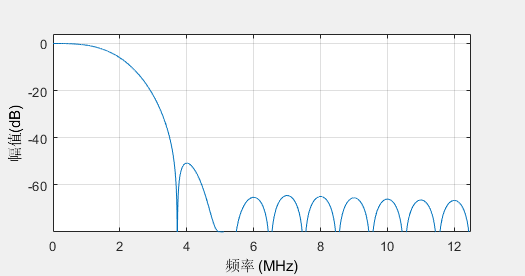
滤波器分类示意图

1.1.2.1 低通滤波器

滤波(Low-pass filter) 是对信号的一种过滤方式。

截止频率是当保持输入信号的幅度不变，改变频率使输出信号降至最大值的0.707倍，即用频响特性来表述即为-3dB点处即为截止频率，它是用来说明频率特性指标的一个特殊频率。

低通滤波器是容许低于截止频率的信号通过，但高于截止频率的信号不能通过，幅频响应曲线如图3所示。低通过滤是高通过滤的对立。低通过滤的概念存在于各种不同的领域，诸如电子电路，数据平滑，声学阻挡，图像模糊等领域经常会用到。最通用的就是巴特沃斯滤波器和切比雪夫滤波器。



低通滤波器的幅频响应

1.1.2.2 有限脉冲响应（FIR）滤波器

1、定义

FIR滤波器是有限长单位冲激响应滤波器，又称为非递归型滤波器，是数字信号处理系统中最基本的元件，它可以在保证任意幅频特性的同时具有严格的线性相频特性，同时其单位抽样响应是有限长的，因而滤波器是稳定的系统。因此，FIR滤波器在通信、图像处理、模式识别等领域都有着广泛的应用。

2、FIR滤波器的特点

·单位抽样响应h(n)是有限长序列。

·系统函数在有限z平面上没有极点存在。

·无输出到输入的反馈，一般为非递归型结构。

FIR的系统函数如公式(4)所示，该滤波器的主要结构主要有横截型（直接型）、级联型、频率抽样型



3、FIR滤波器工作原理

在进入FIR滤波器前，首先要将信号通过A/D器件进行模数转换，把模拟信号转化为数字信号；为了使信号处理能够不发生失真，信号的采样速度必须满足香农采样定理，一般取信号频率上限的4-5倍做为采样频率；一般可用速度较高的逐次逼进式A/D转换器，不论采用乘累加方法还是分布式算法设计FIR滤波器，滤波器输出的数据都是一串序列，要使它能直观地反应出来，还需经过数模转换，FIR滤波器的输出须外接D/A模块。



FIR滤波器工作原理

1.2基于FPGA的FIR低通滤波器的设计与实现

1.2.1性能指标

采样率：25MSPS

分辨率：8bit/12bit

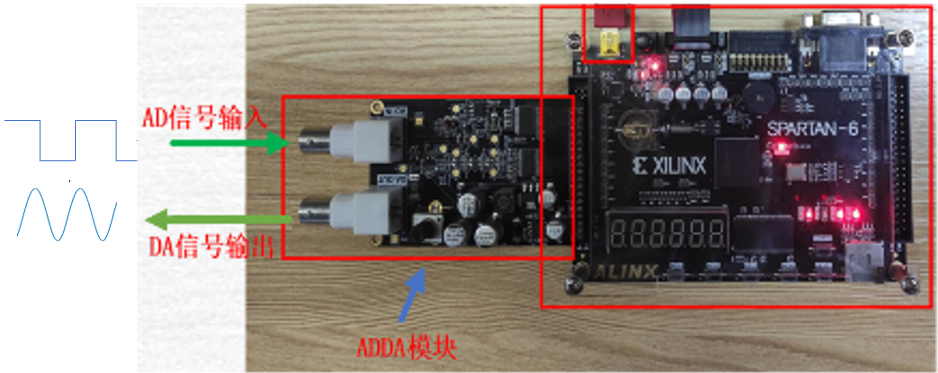
低通滤波截止频率:2MHz（或者1MHz、3MHz）

滤波器阶数：25阶

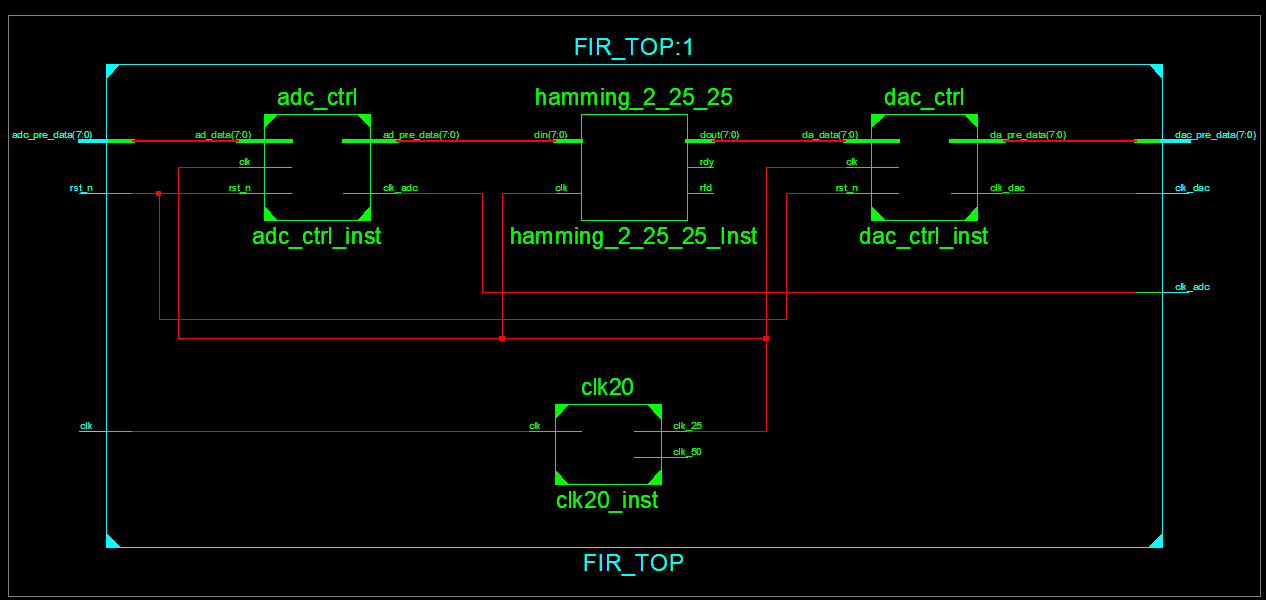
实现方法：Matlab, FPGA

验证方法:1MHz~2MHz方波经过滤波器后是否变形为接近正弦波

1.2.2整体框架



通过将方波信号从ADC模块输入，在FPGA中实现FIR低通滤波，最后通过DAC模块再将滤波后的波形输出，最后通过示波器或是ChipScope进行观察。



FPGA中简单分为ADC、DAC、滤波器、时钟分频四个模块；

ADC模块负责将模拟输入转换为数字量输入滤波器；

DAC将滤波器输出量转换为模拟量输出；

滤波器模块负责将Matlab产生的滤波器系数加以应用，实现对输入信号的滤波；

时钟分频模块主要保证产生一个统一时钟分别满足ADC采样，DAC输出，滤波器采样，滤波器时钟四个需求。

1.2.3 FIR滤波器Matlab仿真

1.2.3.1 FIR数字滤波器的设计方法

信号插值中所使用的滤波器可以选用FIR数字滤波器，利用其稳定性好、精度高、可以获得严格的线性相位的特点。根据信号内插的原理，所使用的FIR低通滤波器的归一化截止频率为1/*I*（其中*I*为插值倍数），通带内增益为*I*，滤波器阶数*N*根据设计精度进行合理选择。

FIR数字滤波器设计一般有两种方法：窗函数法和频率抽取设计法。窗函数这种设计方法对于窗口函数序列的形状和长度选择很重要。这种方法简单实用，可是其截止频率不易控制。而频率抽取设计法可以直接在频域进行设计，便于优化，但是其截止频率并不能够自由取值。二者各有其优缺点。

1、窗函数法

由滤波器的频率响应进行傅里叶反变换，可以得到滤波器的抽样响应：



由于是无限长序列且为非因果响应，是物理不可实现的。为此要寻找一个因果序列，在相应的误差准则下逼近。窗函数法设计的初衷是使设计的滤波器频率特性在频域均方误差最小意义下逼近期望频率响应，即



窗函数法就是用被称为窗函数的有限长加权序列（当*n*>*N*-1及*n*<0时，=0）来修正式(22)，则所得为：



求出FIR滤波器的频率响应，并检验是否满足设计要求，如不满足，则需重新设计。

设计FIR滤波器常用的窗函数有：矩形(Rectangle)窗函数、三角窗(Bartlett)函数、汉宁(Hanning)窗函数、海明(Hamming)窗函数、布莱克曼(Blackman)窗函数和凯塞(Kaiser)窗函数等。六种窗函数的旁瓣峰值衰减系数、过渡带、最小阻带衰减系数均有所不同，如表8.1所示。

表8.1 六种窗函数基本参数表[[[1]](#endnote-1)]

|  |  |  |  |
| --- | --- | --- | --- |
| 窗类型 | 旁瓣峰值衰减(dB) | 过渡带Δw | 最小阻带衰减(dB) |
| 矩形 | -13 | 4π/N | -21 |
| 三角 | -25 | 8π/N | -25 |
| 汉宁 | -31 | 8π/N | -44 |
| 海明 | -41 | 8π/N | -53 |
| 布莱克曼 | -57 | 12π/N | -74 |
| 凯撒b=7.865 | -57 | 10π/N | -80 |

窗形状决定最小阻带衰减，过渡带的宽度决定N的取值。所以选择窗函数的标准：使主瓣的宽度尽量窄，以获得较陡的过渡带；使能量尽量集中在主瓣中，从而减少肩峰和波纹，提高阻带衰减。

窗函数的特点：使理想特性不连续边沿加宽，形成一个过渡带，过渡带宽度（两个肩峰间的宽度）等于窗函数主瓣宽度；在过渡带两侧，幅频响应出现最大肩峰值；最大肩峰的两侧，形成起伏振荡，其振荡幅度取决于旁瓣的相对幅度，而振荡的多少，则取决于旁瓣的多少。

吉布斯现象：主瓣与旁瓣的相对比例只有窗函数的形状来决定。

2、频率采样法

频率采样法是从频域出发，将给定的理想频率响应加以等间隔采样，即：



然后以此作为实际FIR数字滤波器的频率特性的采样值，由通过IDFT可得有限长序列：



对式(25)做z变换，可得：



其傅里叶变换为：

 (25)

式中，为内插函数：



由内插函数可以看出，除各取样点上频域响应将严格与理想特性保持一致外，在取样点之间的响应由各取样点内插得到。因此，如果取样点之间的理想特性越平缓，则内插值就越接近理想，逼近就越好。相反，如果取样点之间的理想特性变化越积累，则内插值与理想值的误差就越大，因而在理想特性的每个不连续点附近会出现肩峰和起伏，不连续性越大，肩峰和起伏就越大。

本项目使用窗函数法。

1.2.3.2 数字滤波器系数生成方法

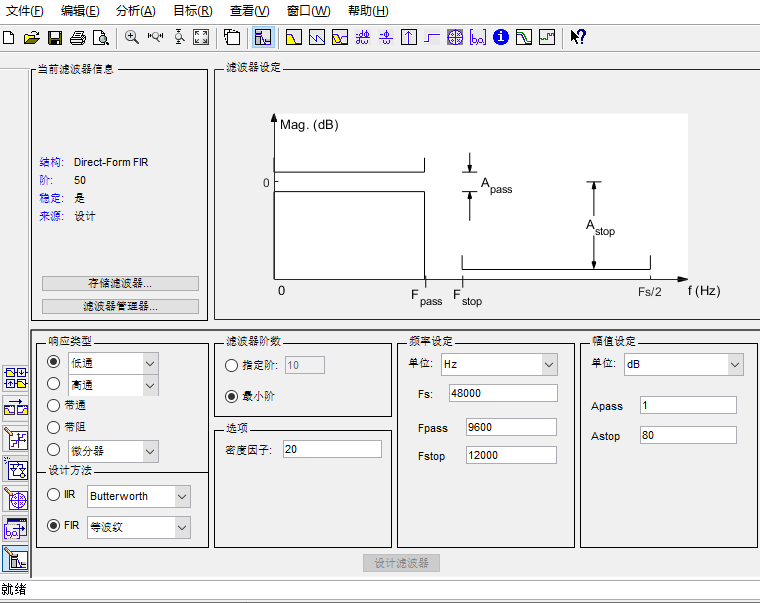
本项目推荐使用MATLAB的FDAtool设计得到。

按照实际需要确定滤波器的性能要求，比如确定所要设计的滤波器是低通、高通、带通还是带阻，滤波器阶数、截止频率是多少，阻带的衰减有多大，通带的波动范围是多少等。

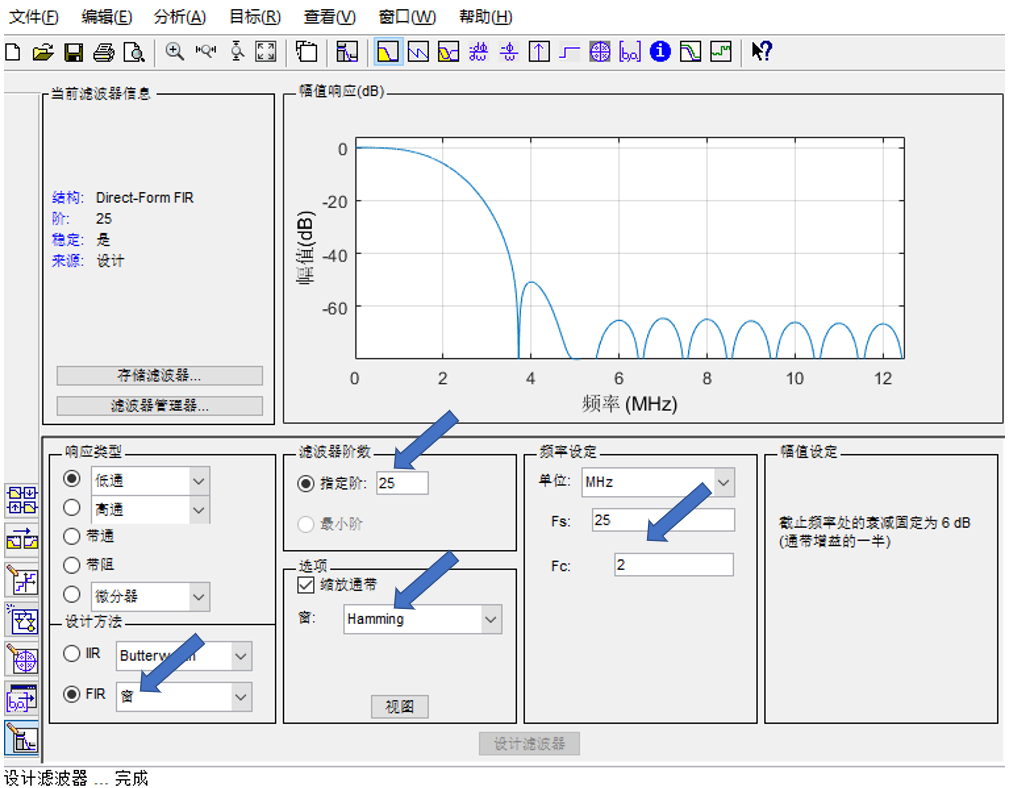
指标要求采样率为25MHz，截止频率设为2MHz,滤波器阶数设为25阶。

FDAtool设计

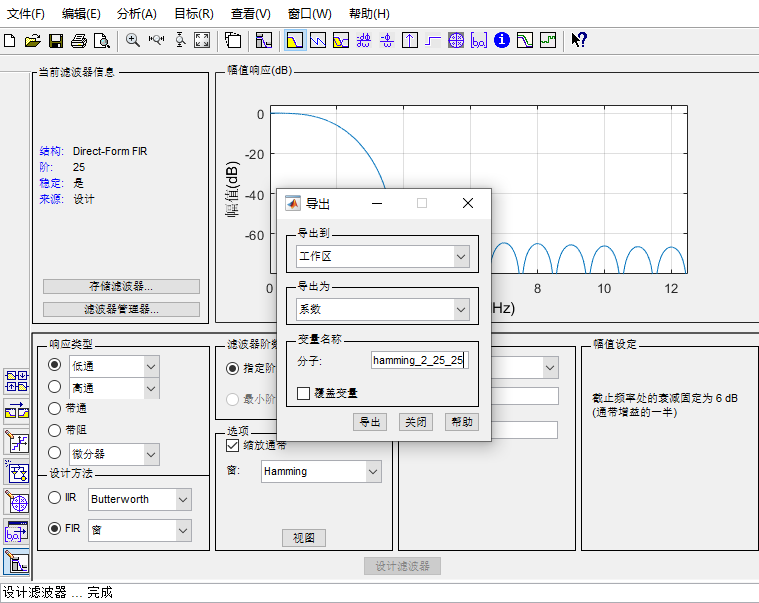
1. MATLAB命令输入fdatool，打开设计工具；



1. 设计数字FIR低通滤波器，根据需要设定阶数specify order，采样频率Fs=25MHz，输入信号频率Fc=2MHz；
2. 点击design filter，显示的是幅频响应magnitude response，可变化显示界面，选择滤波器系数filter coefficients，此时显示的系数是二阶级联情况下的系数，在Edit中选择转换convert to single section，Z系统下的系数，numerator是分子系数，Denominator是分母系数；



1. 保存系数。在文件下选择导出到工作区。



1.2.3.3 基于MATLAB完成FIR滤波器仿真验证

a. 确定采样频率、滤波器阶数N；

b. 确定通带截止频率、阻带截止频率及其归一化角频率；

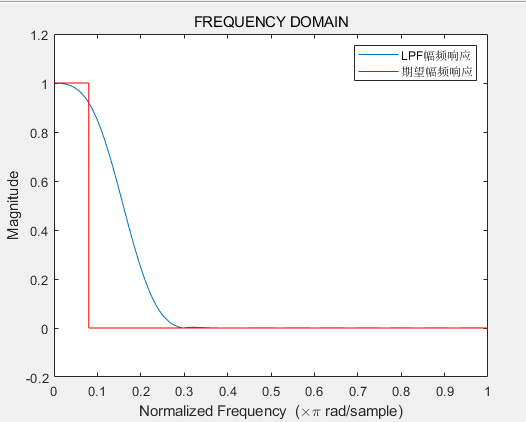
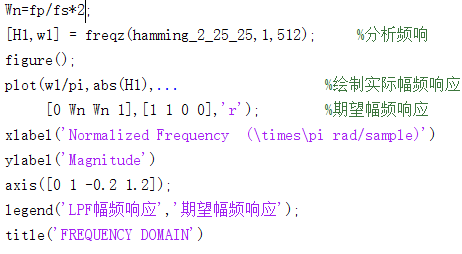
c. 根据给定条件选择合适的窗函数；

d. 基于上步得到滤波器对应的抽头系数；

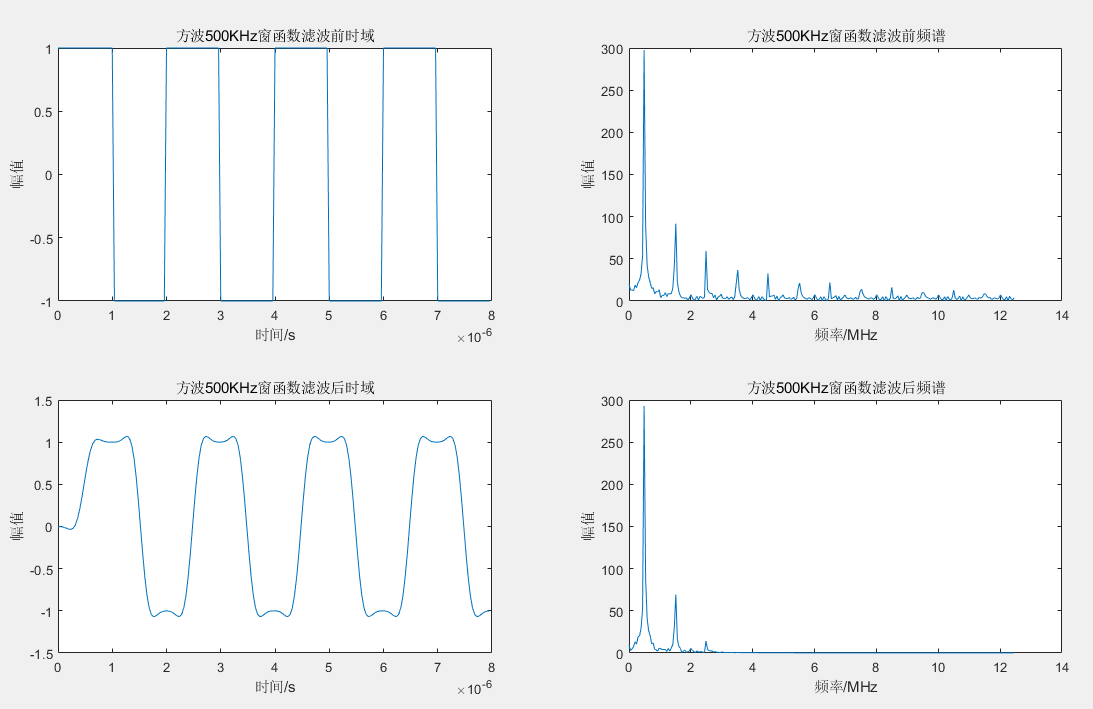
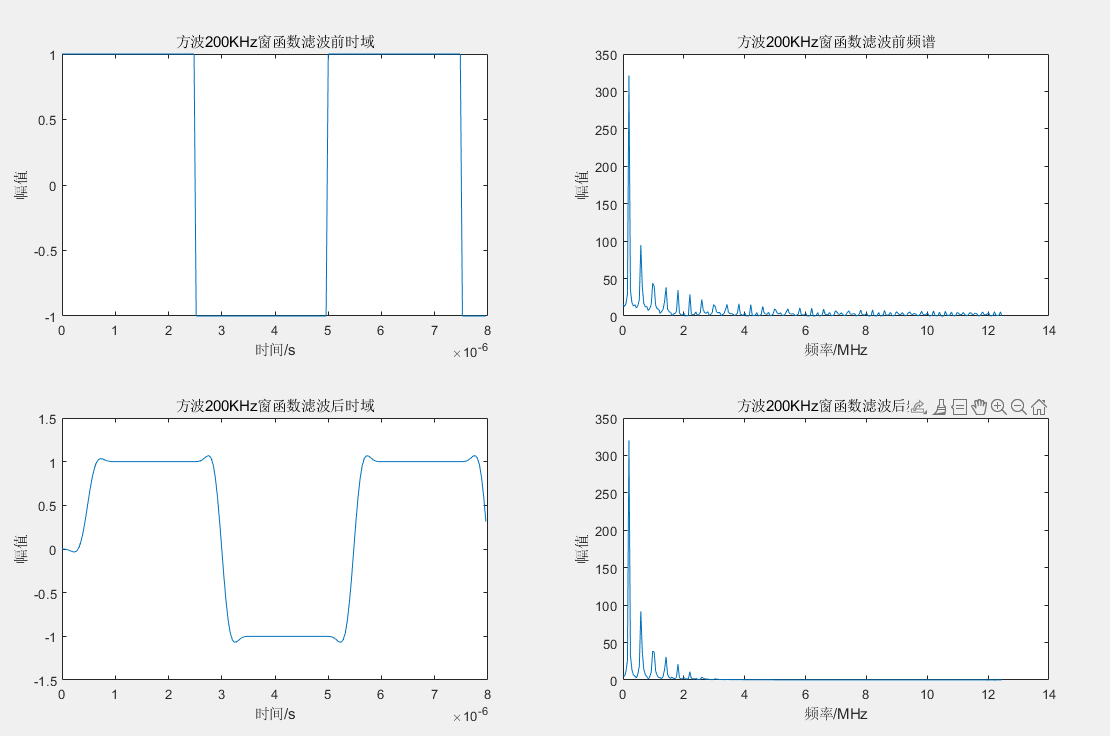
e. 利用freqz( )函数分析滤波器幅频响应；

f. 利用filter( )函数对需要滤波的信号进行滤波。

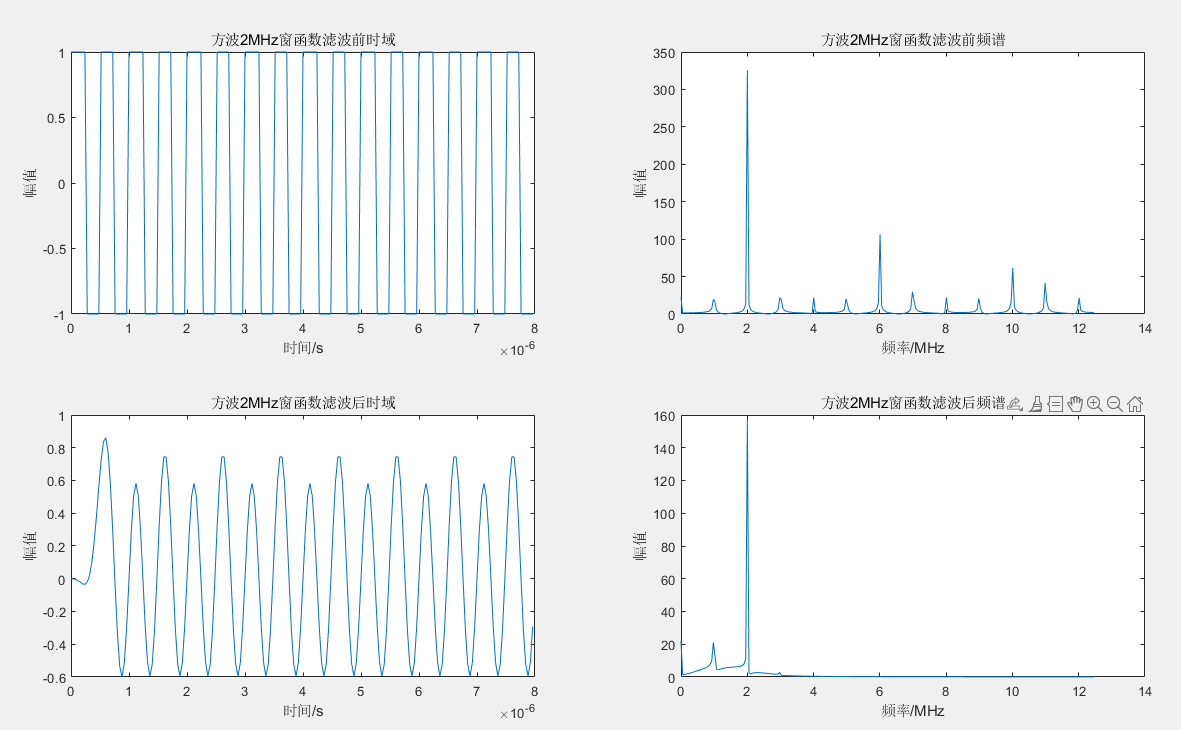
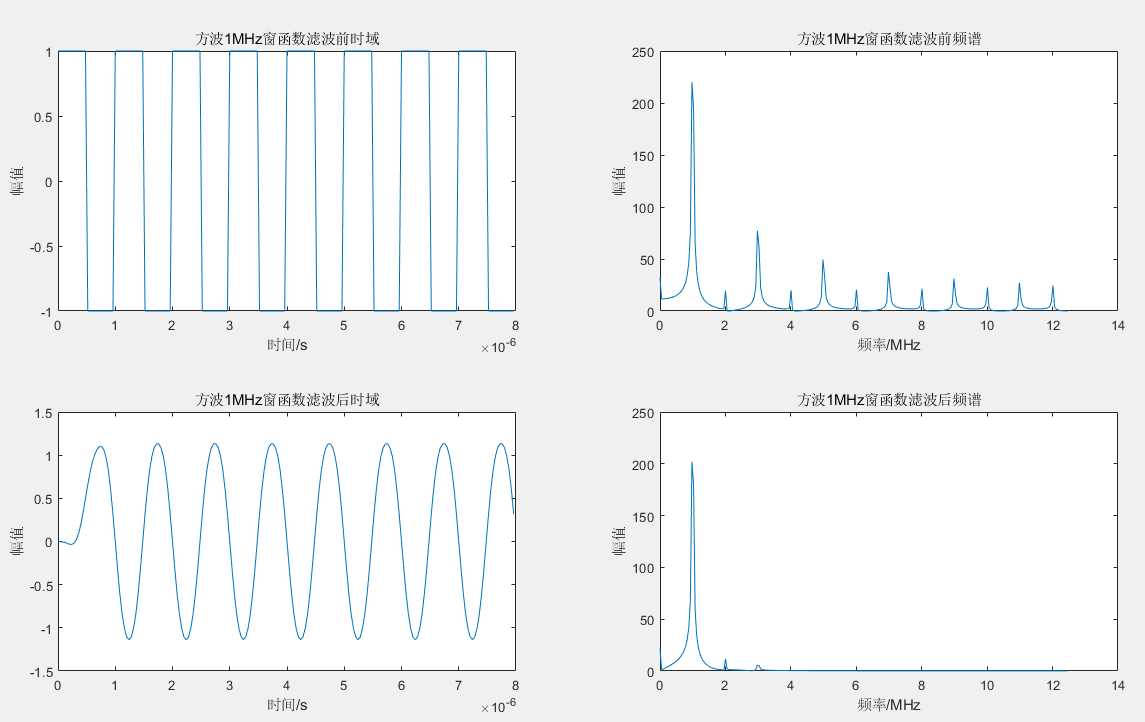
根据上述步骤，计算出相应的归一化频率，选择合适的窗函数和滤波函数，MATLAB代码大致如下，最后设计的结果如图的幅频响应曲线所示。



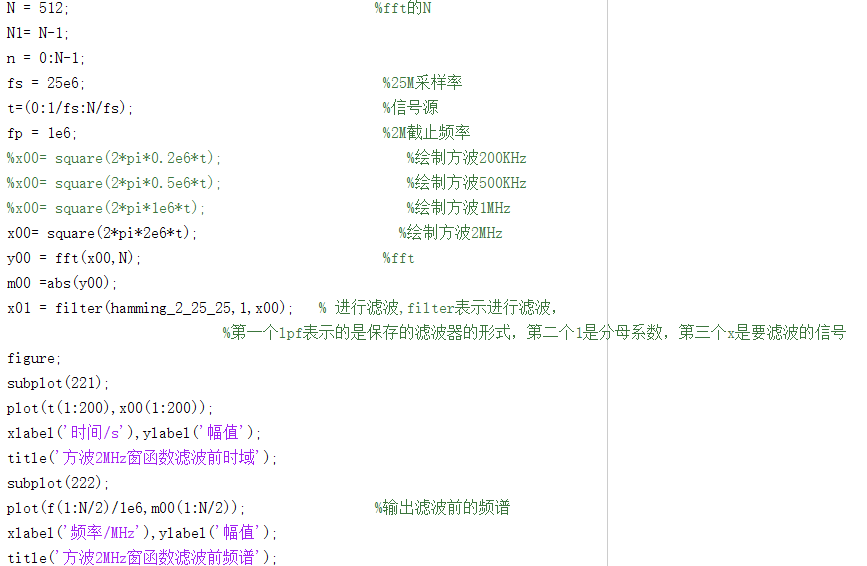
输入不同频率方波进行滤波：

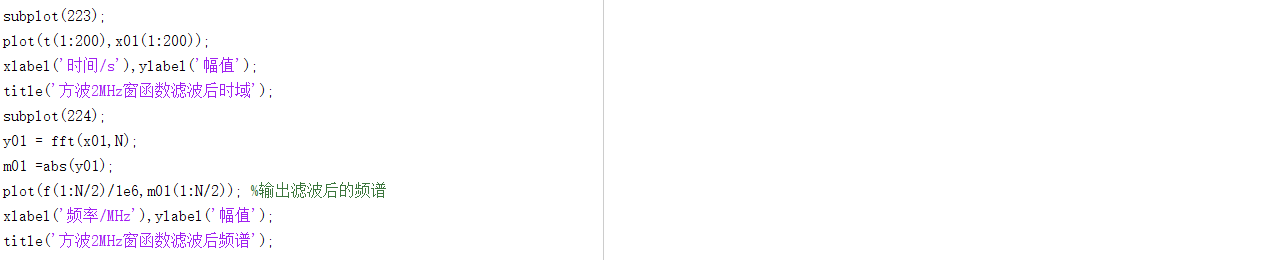


200KHz方波 500KHz方波



1MHz方波 2MHz方波



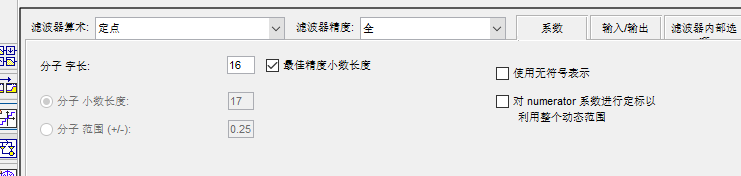
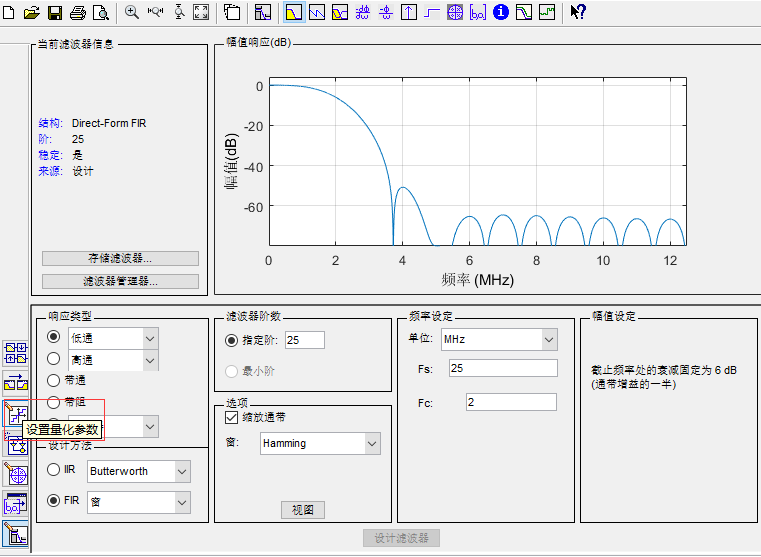


指标均满足设计要求，可以将抽头系数导入FPGA中。

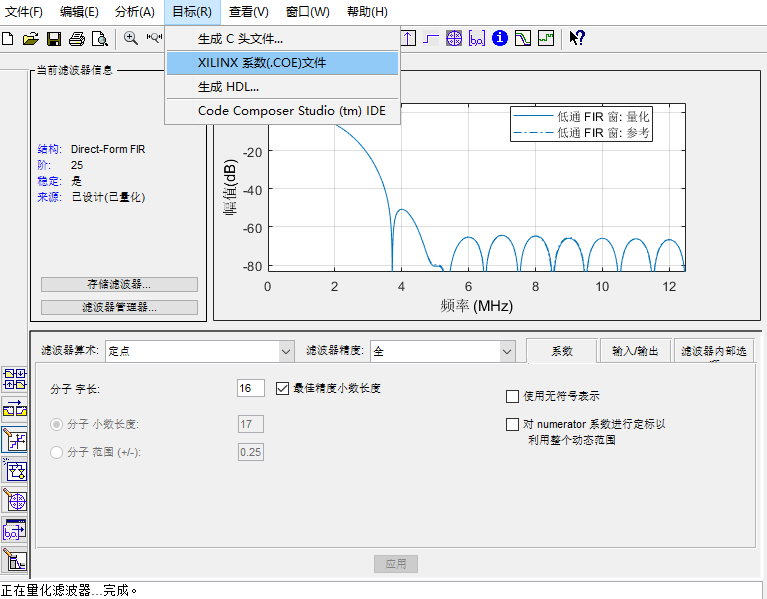
1.2.4 FIR滤波器在FPGA中的实现

1.2.4.1 数字滤波器系数导入IP核

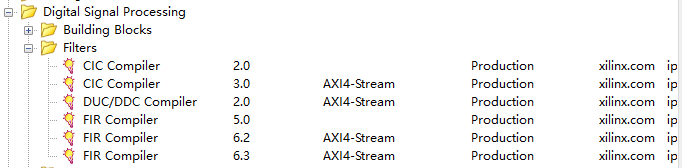
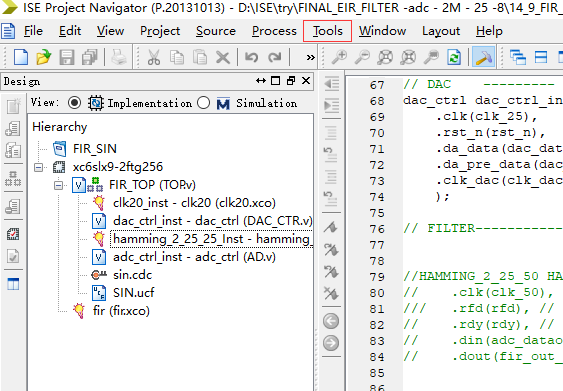
1）继续在FDAtool中点击**设置量化参数**，在滤波器算数处改为**定点**



2）在目标中选中生成**coe文件**，以便导入IP核中。

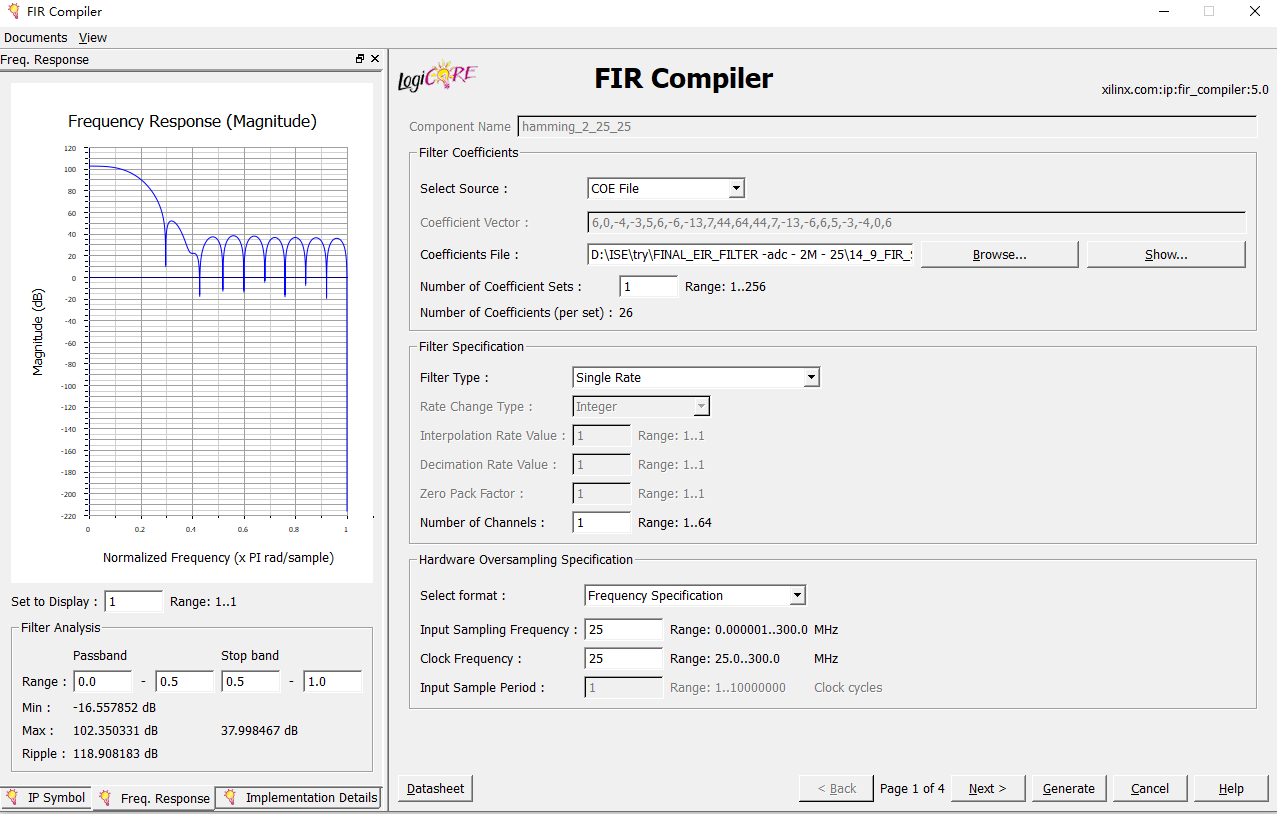


3）在新建的ISE工程下，点击tools🡪cores generator添加FIR Filter IP核，

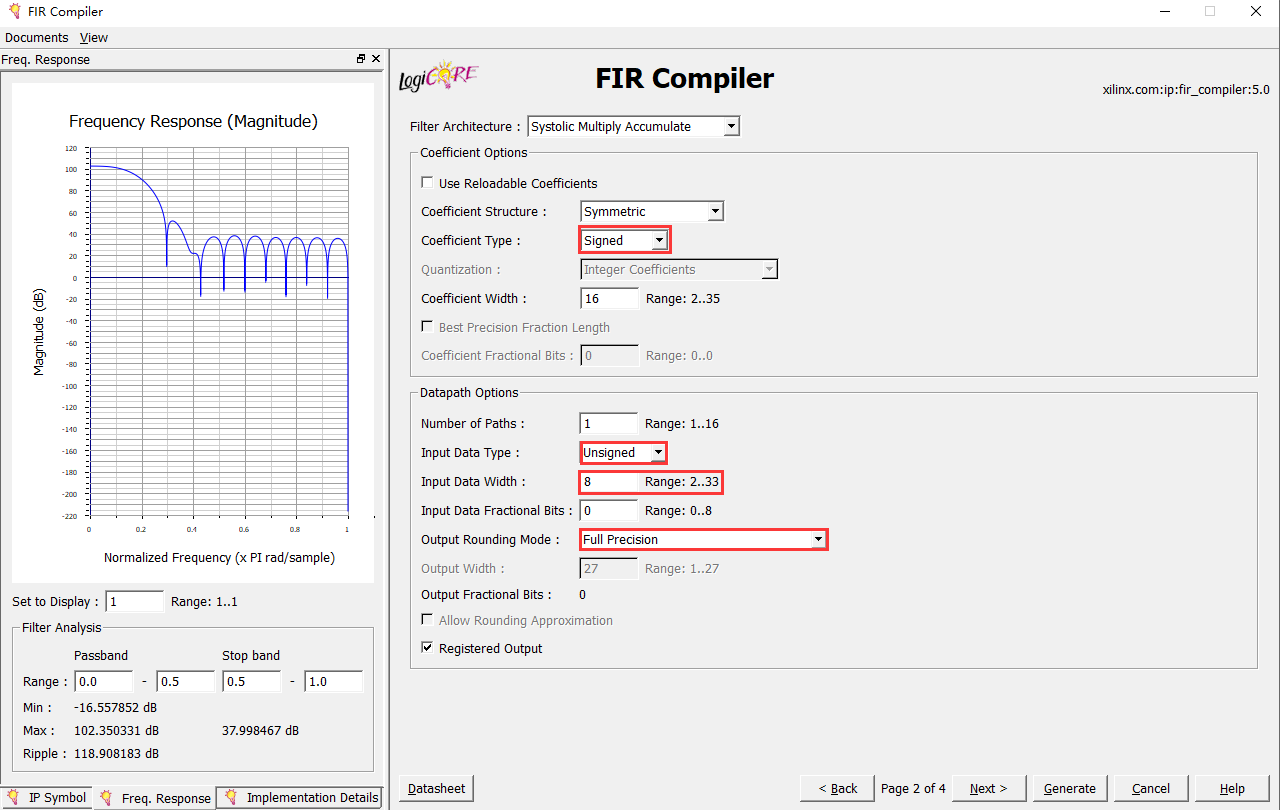


4）在第一页select source中选择coe文件，并导入刚才导出的文件；

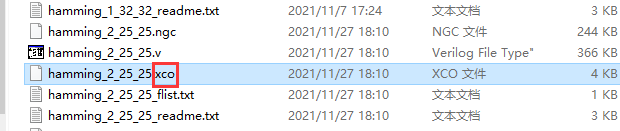
选择single rate 单通道；过采样设置选择频率，采样频率与时钟频率相同，均为25MHz（采样频率是之前在matlab中设置的fs，必须保持相同，但时钟频率可以与之相同，也可以用板载的时钟频率）



5）滤波器架构选择流水线直接型，**注意coe文件数据类型选择有符号数！！数据类型选择八位无符号数！！output选择全部输出，后边再做截断（这些选项至关重要）**



6）后面保持默认，直接生成IP就好

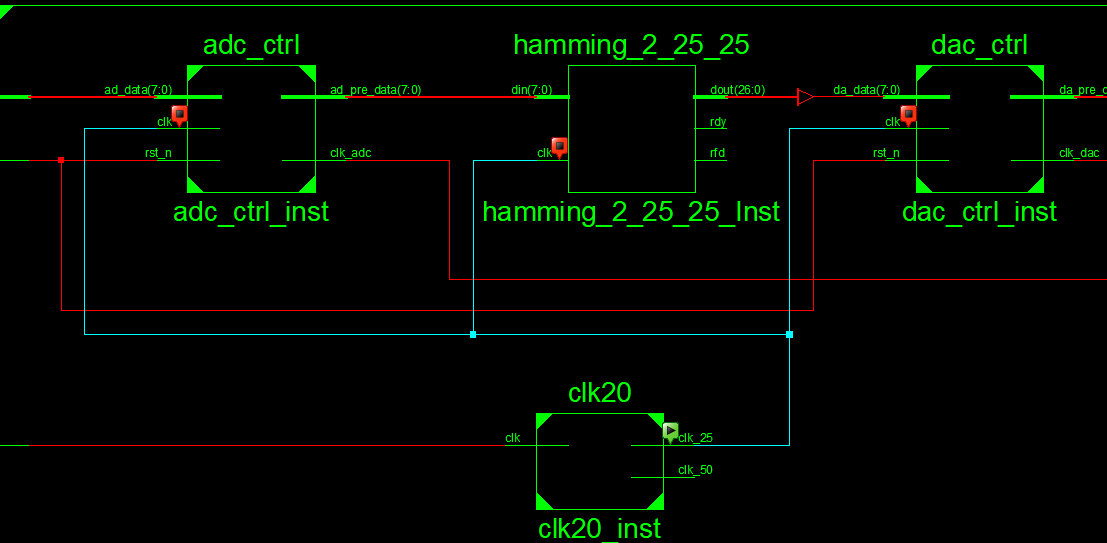
7）在工程中导入滤波器的.xco文件，IP核便顺利导入。

1.2.4.2 数字滤波器整体搭建

在顺利导入FIR IP核后，开始编写其他模块，例如时钟、ADC、DAC模块（附件），需要注意的是，滤波器生成好后会发现输出信号是一个27位长的信号，一般情况下都是取高八位作为输出，**但是！我们在截断的时候要空两位再进行截断，是因为如果按高八位截断的话，最高的两位输出为零，相当于波形幅值缩小到原来的1/4**



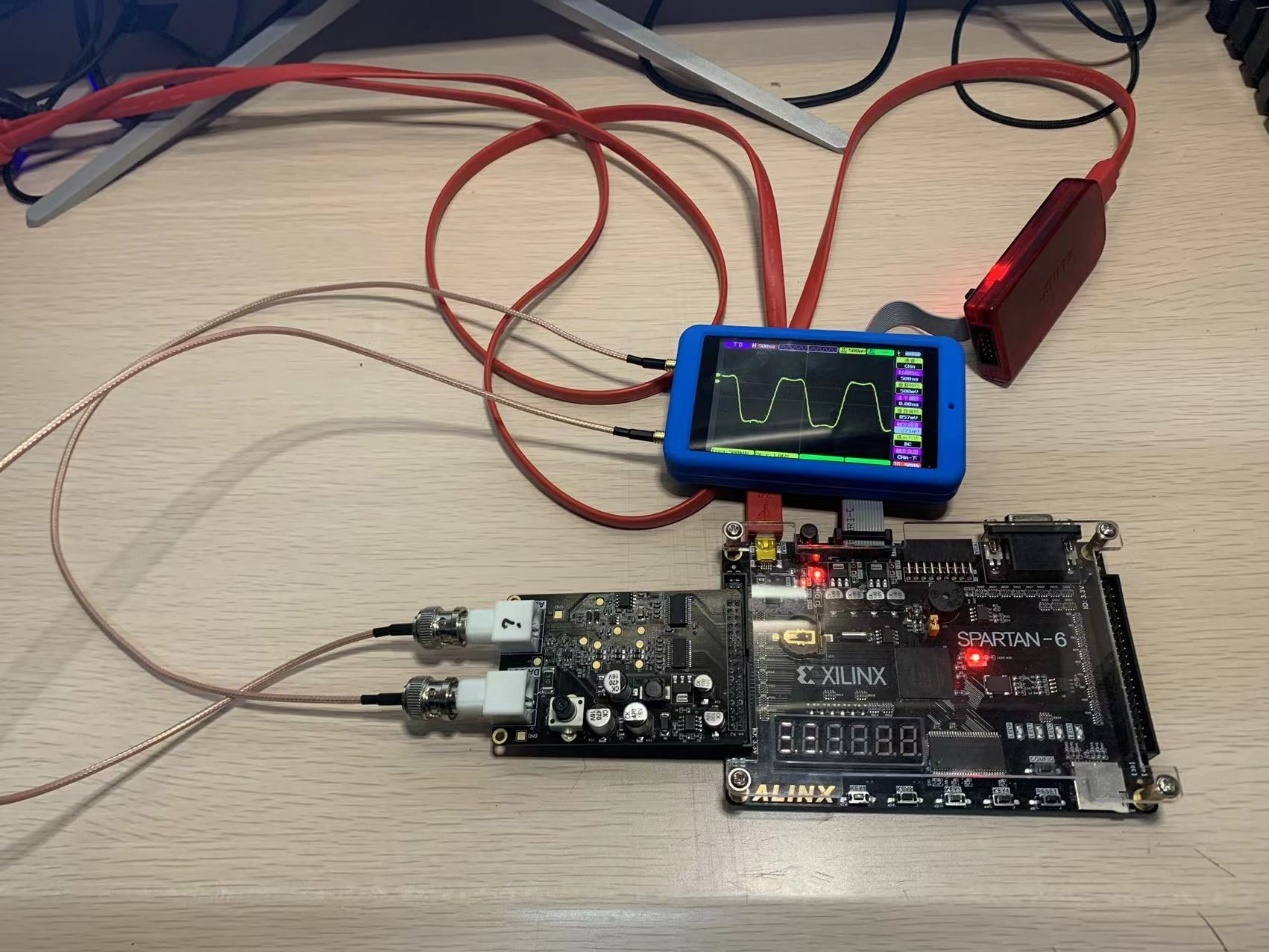
还有就是要统一时钟，均用25MHz;



最后将管脚约束起来，就可以输入实验了。

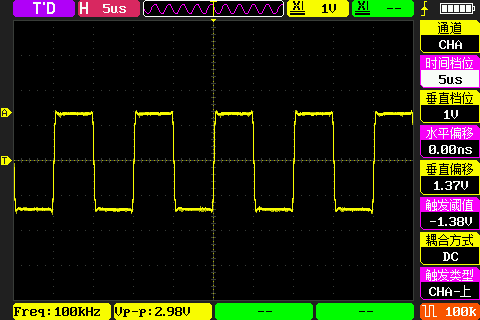
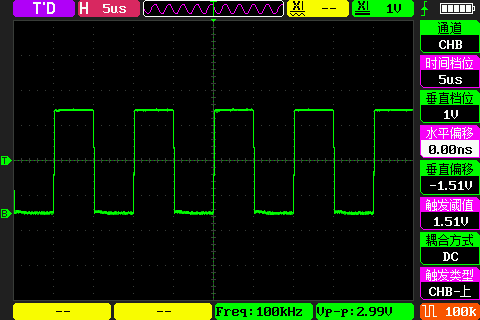
1.2.4.3 数字滤波器输入输出结果对比

1）将设备连接好后，开始输入测试。

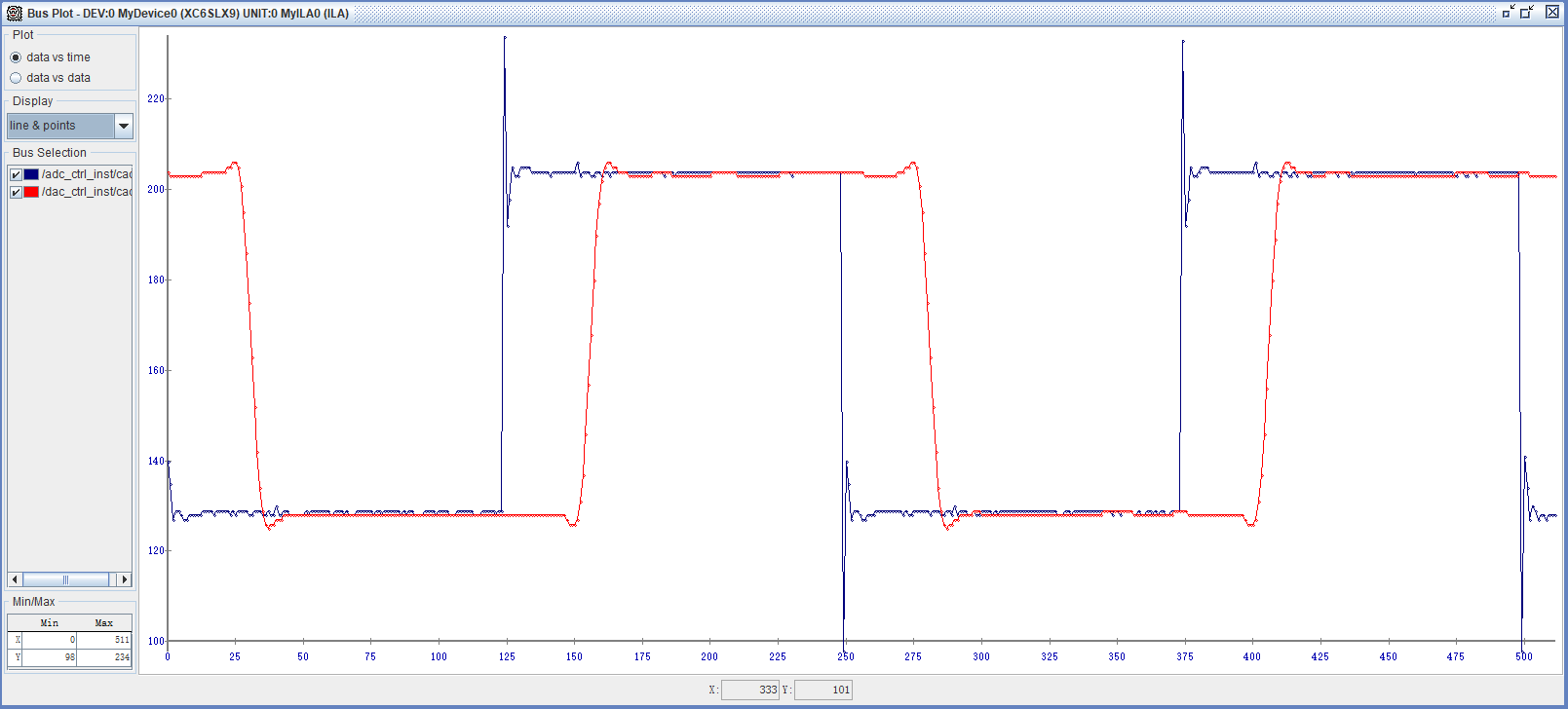


2）调整输入信号频率为100KHz

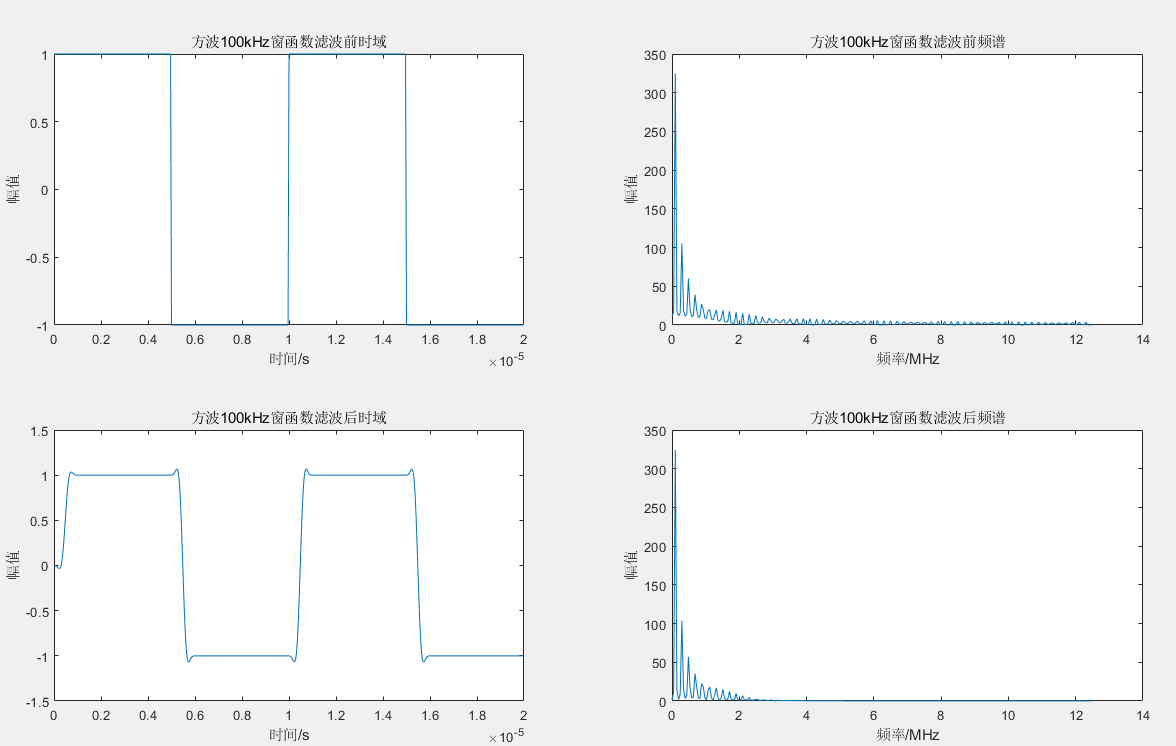
输入信号： 示波器获取的图像：



Chipscope获取波形（蓝色为滤波前ADC捕获，红色为滤波后DAC输出）：



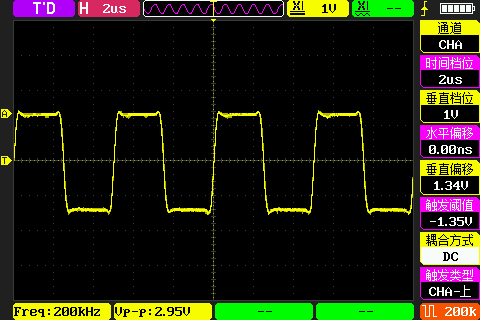
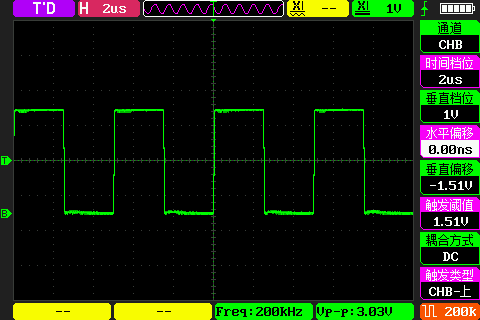
MATLAB仿真图像：



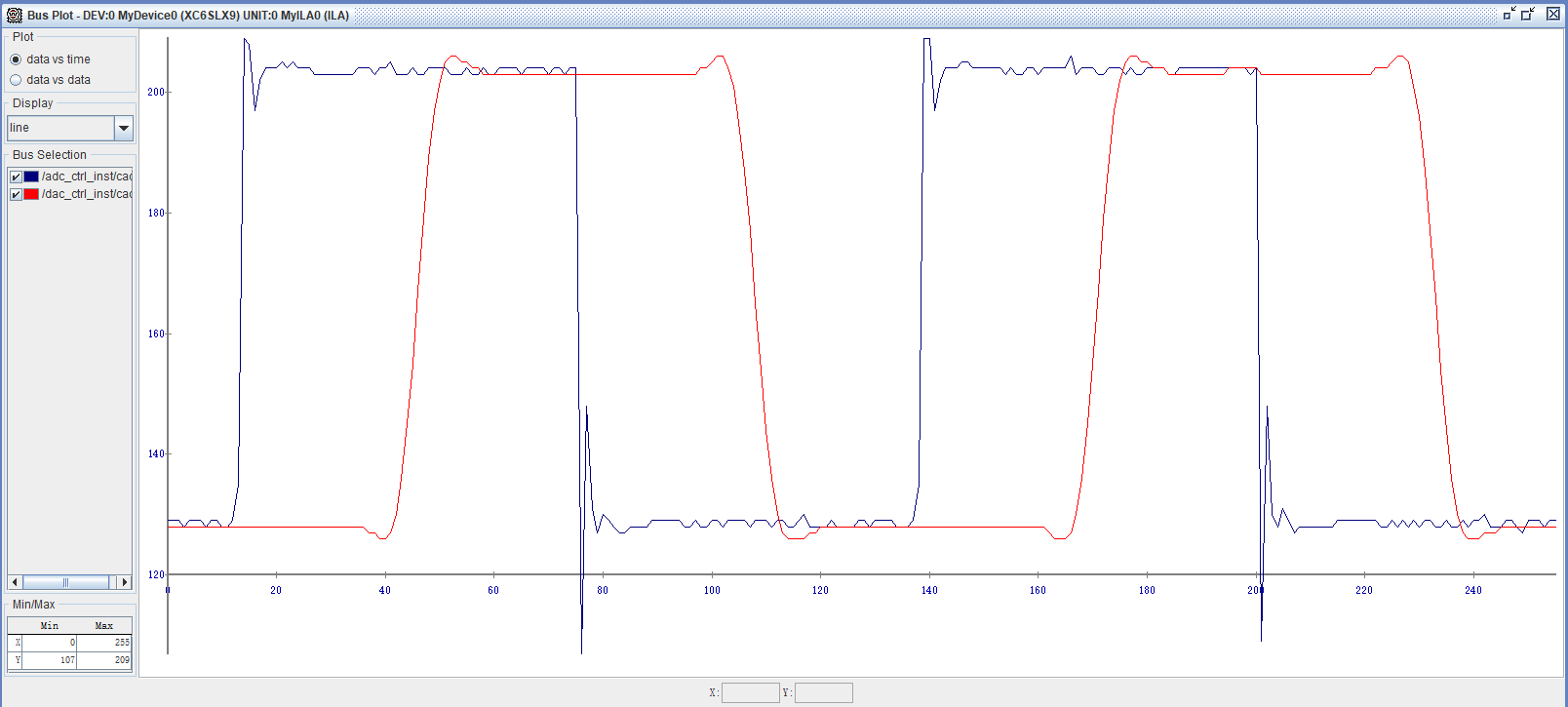
由波形可见，只是高次谐波分量在带外，波形只有一点点失真。

3）调整输入信号频率为200KHz

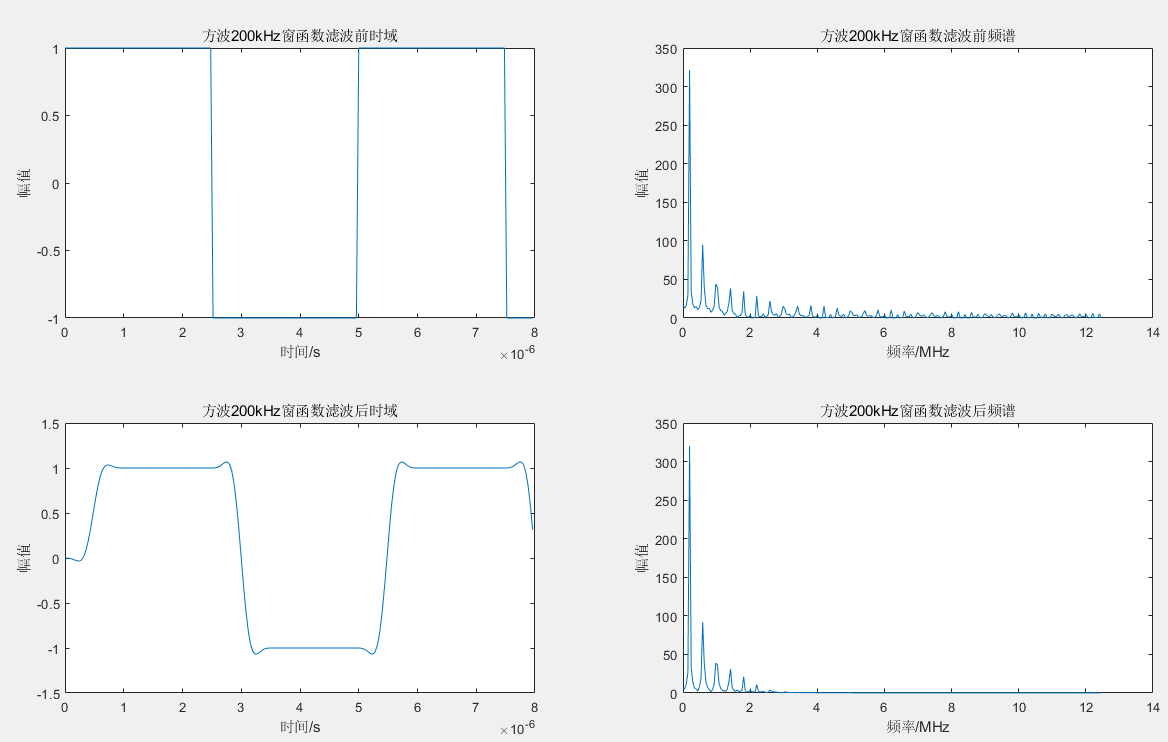
输入信号： 示波器获取的图像：



Chipscope获取波形（蓝色为滤波前ADC捕获，红色为滤波后DAC输出）：



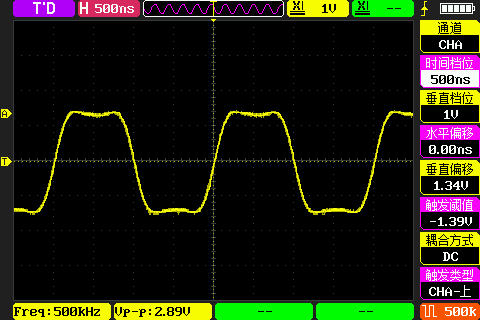
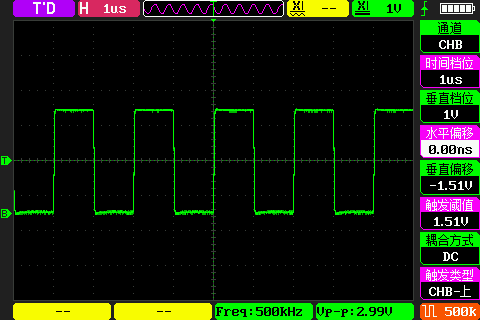
MATLAB仿真图像：



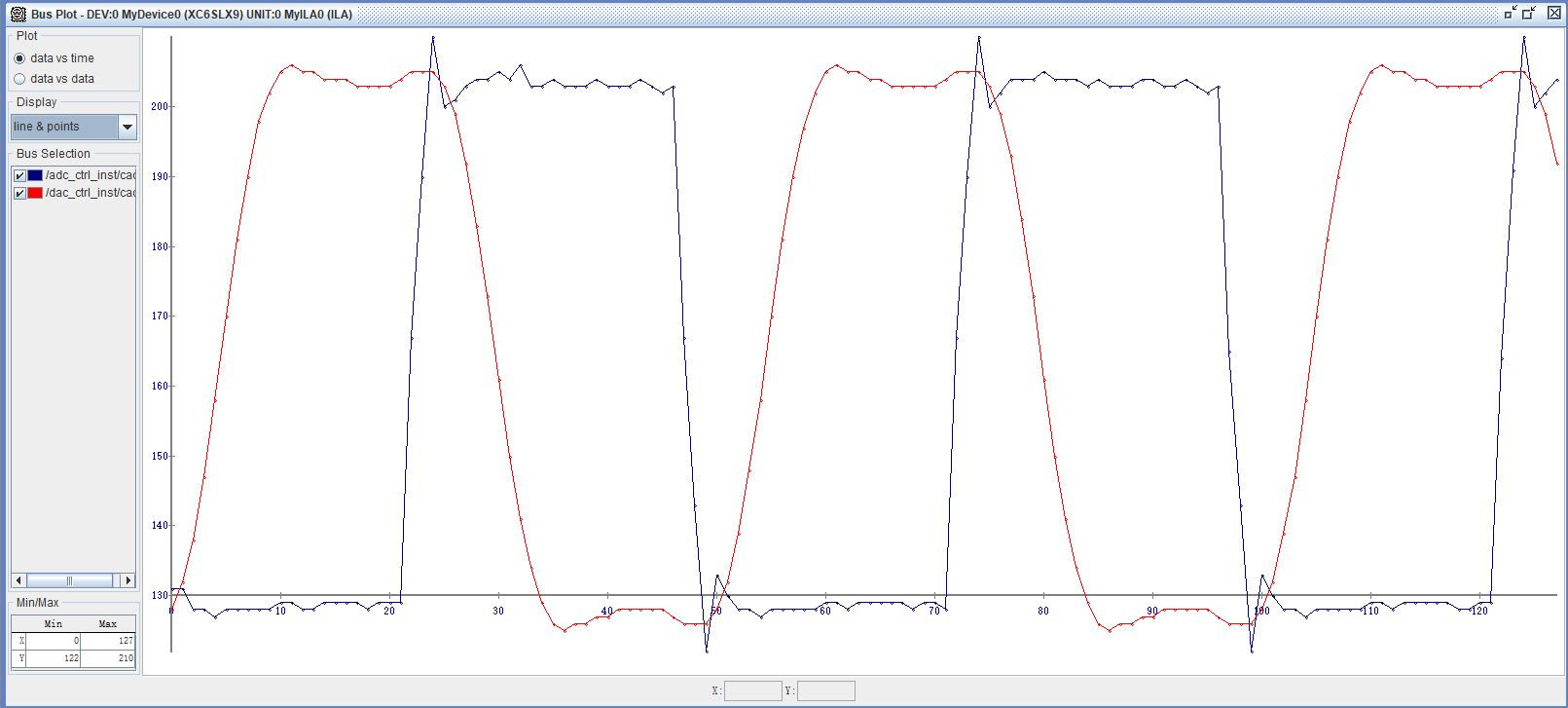
200K时，九次谐波以外的被滤过，波形失真逐渐加大。

4）调整输入信号频率为500KHz

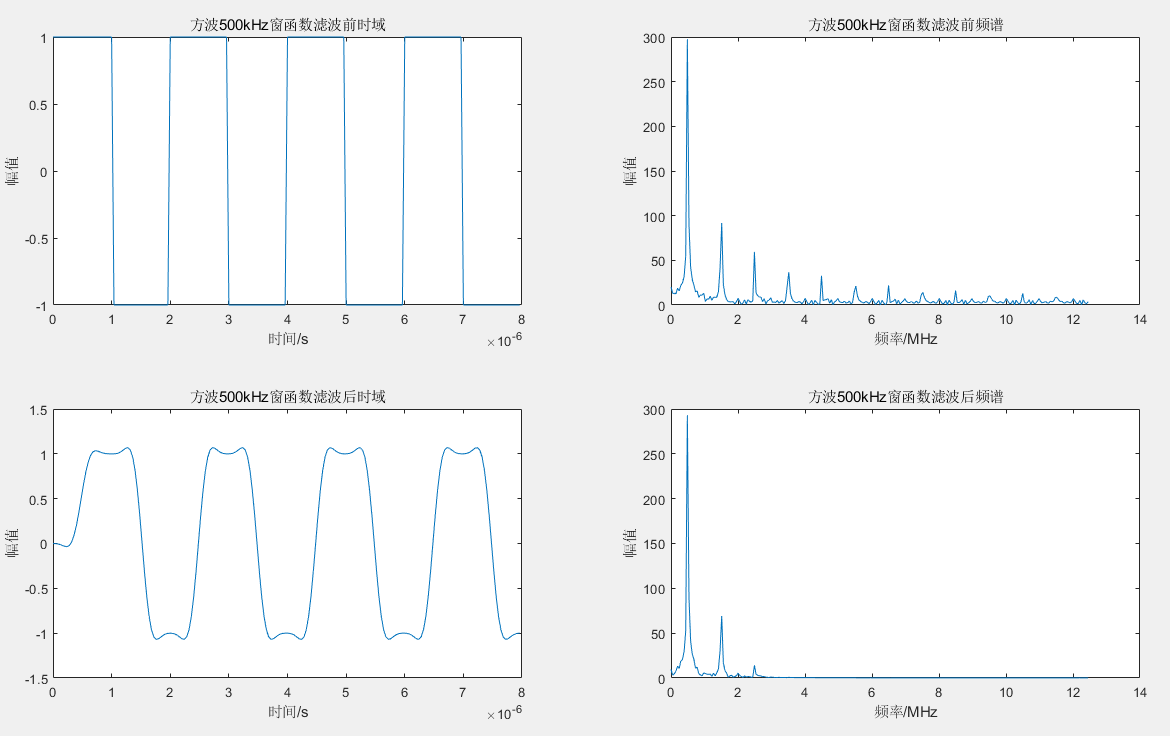
输入信号： 示波器获取的图像：



Chipscope获取波形（蓝色为滤波前ADC捕获，红色为滤波后DAC输出）：



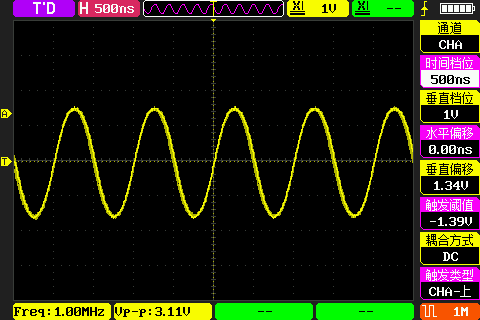
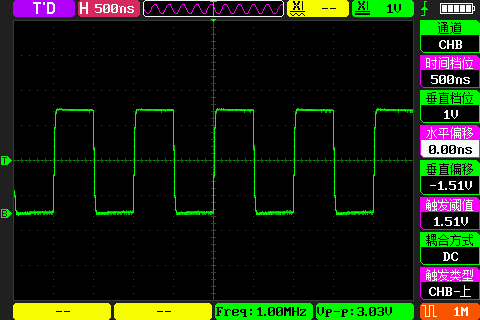
MATLAB仿真图像：



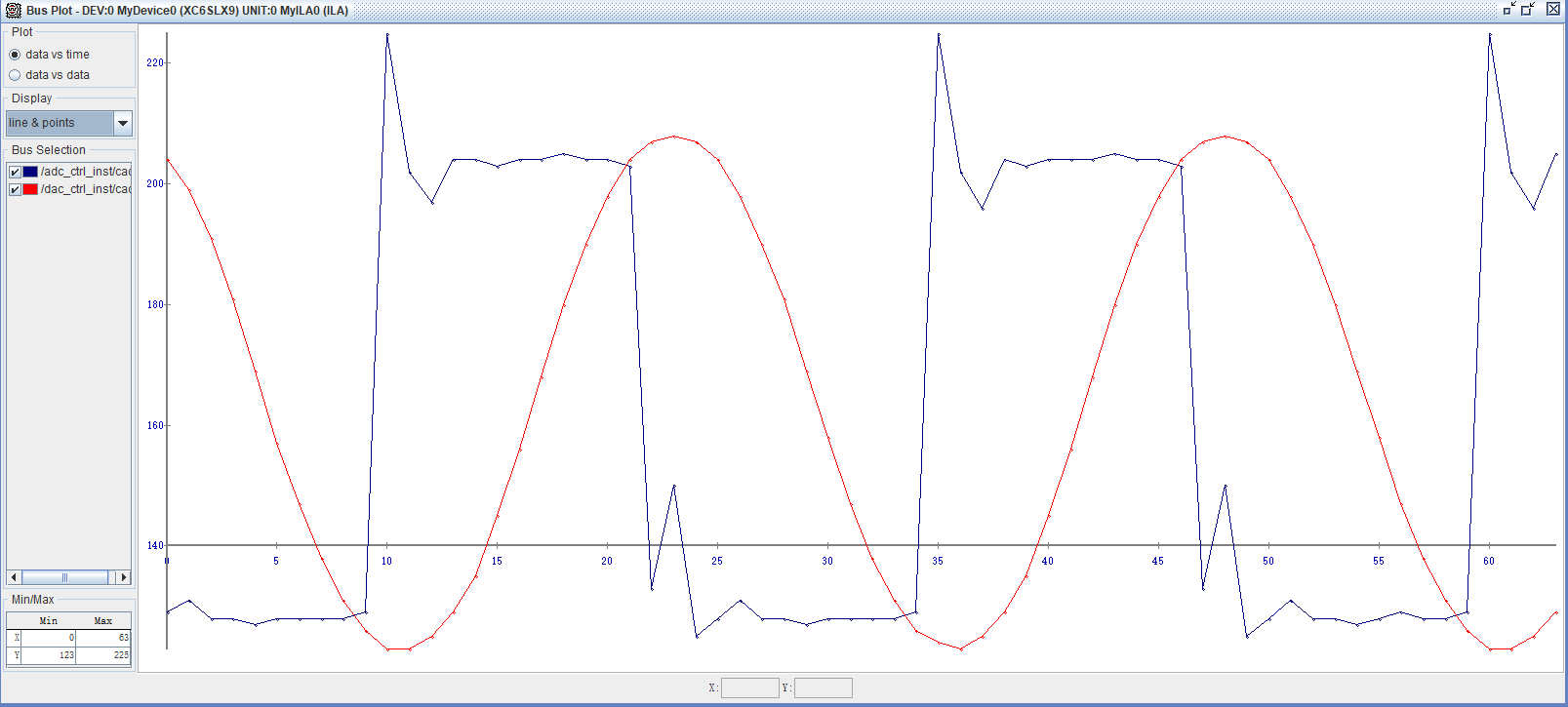
500k时，只有基波，和三次谐波在带内，波形失真严重。

5）调整输入信号频率为1MHz

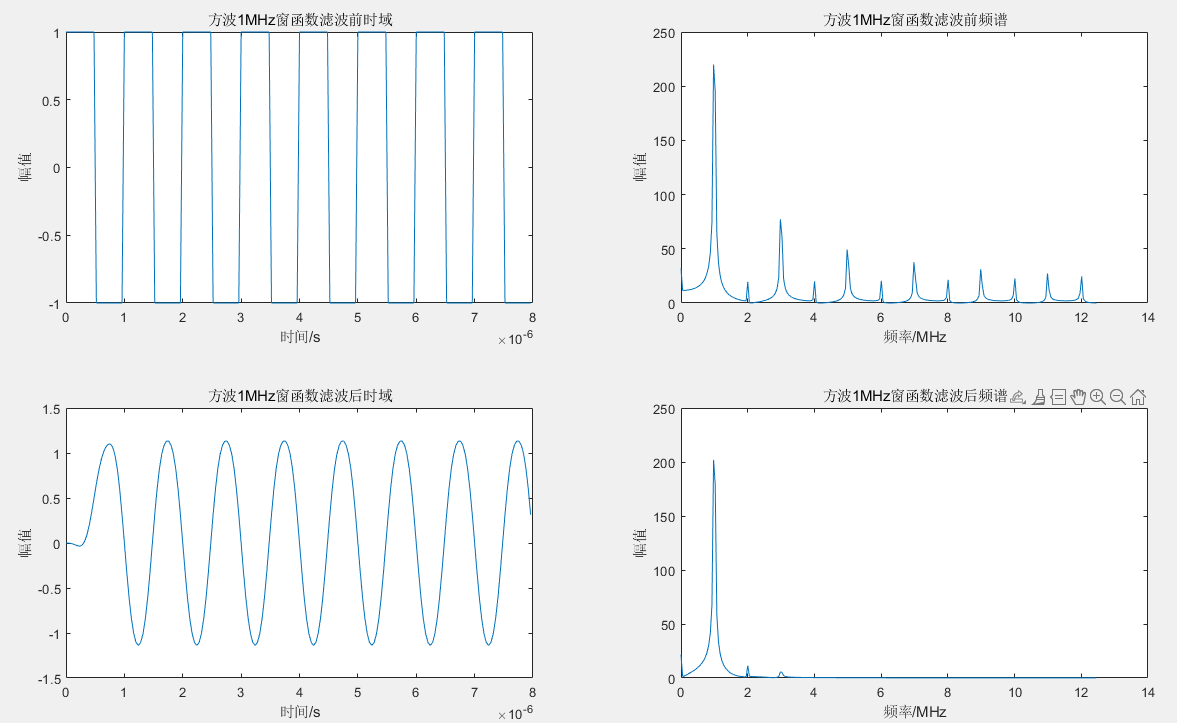
输入信号： 示波器获取的图像：



Chipscope获取波形（蓝色为滤波前ADC捕获，红色为滤波后DAC输出）：



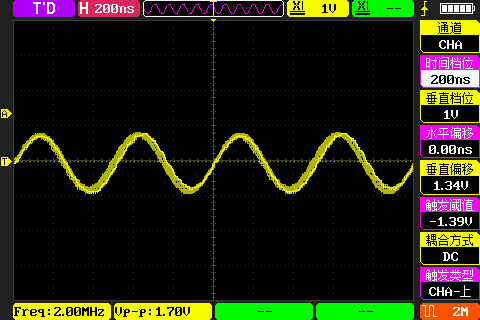
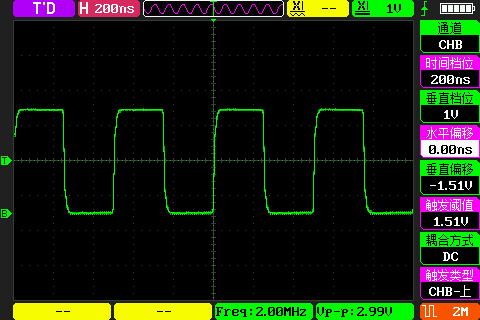
MATLAB仿真图像：



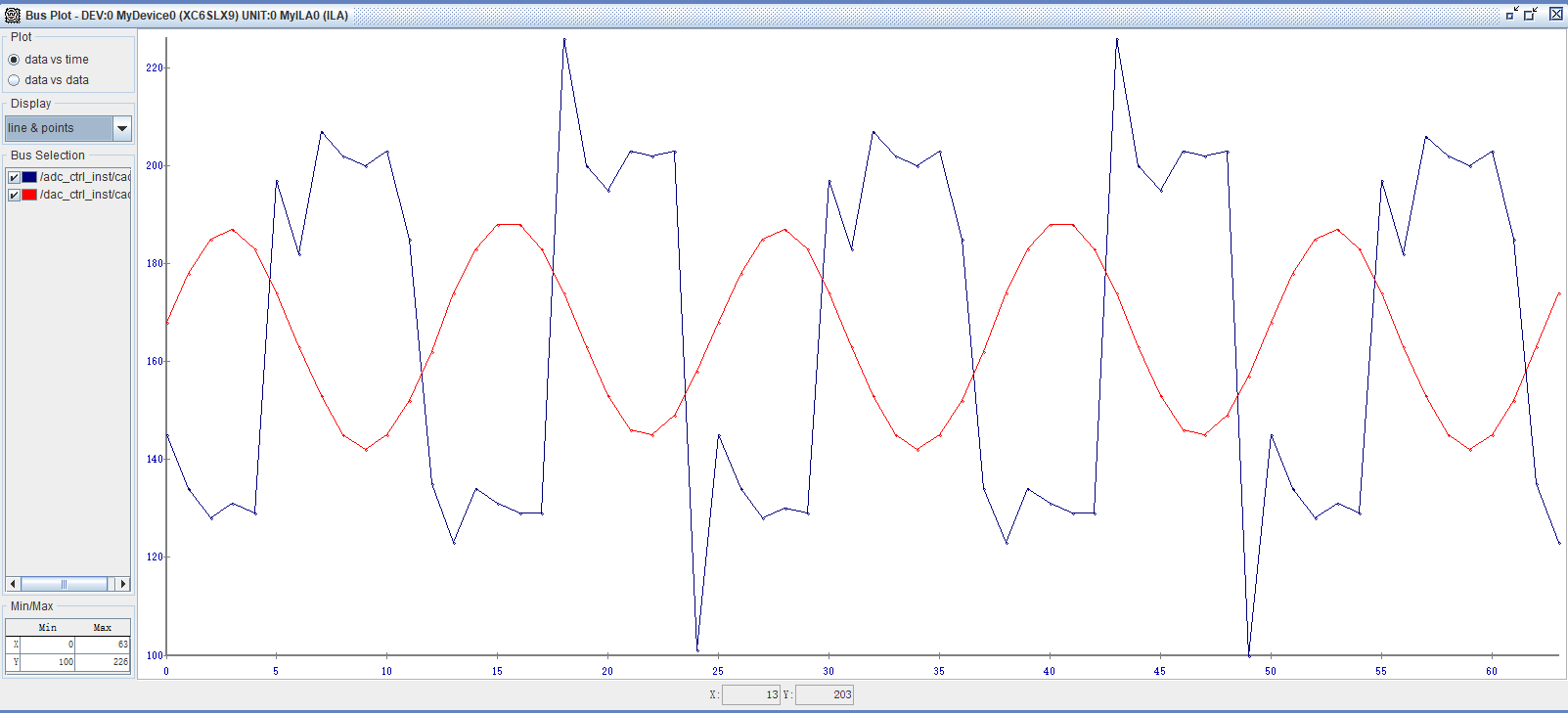
频率到了1MHz，除了基波，其他分量全部被抑制，因此得到了1MHz的正弦波

6）调整输入信号频率为2MHz

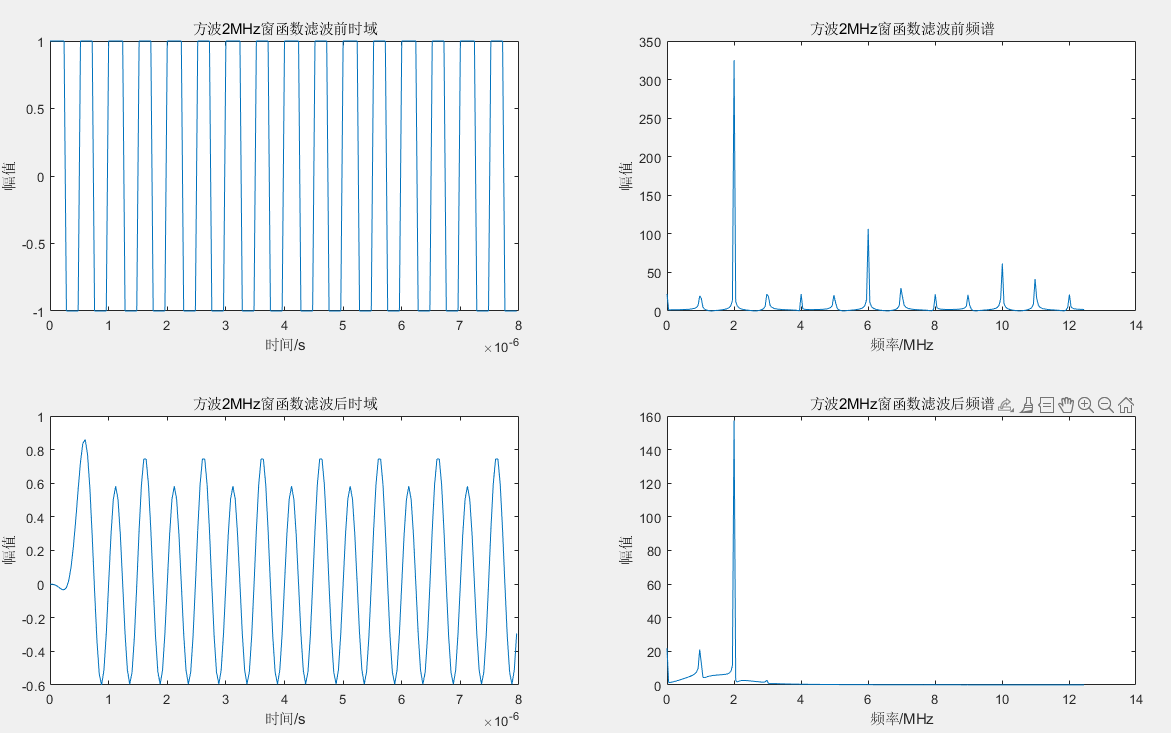
输入信号： 示波器获取的图像：



Chipscope获取波形（蓝色为滤波前ADC捕获，红色为滤波后DAC输出）：



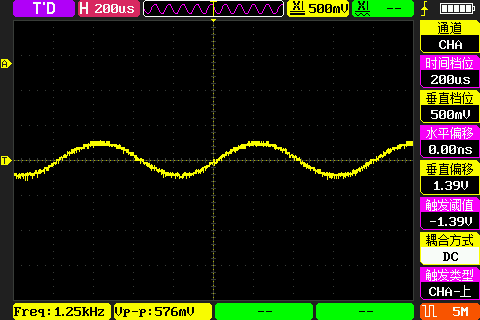
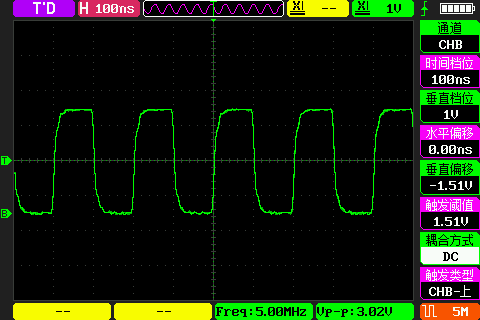
MATLAB仿真图像：



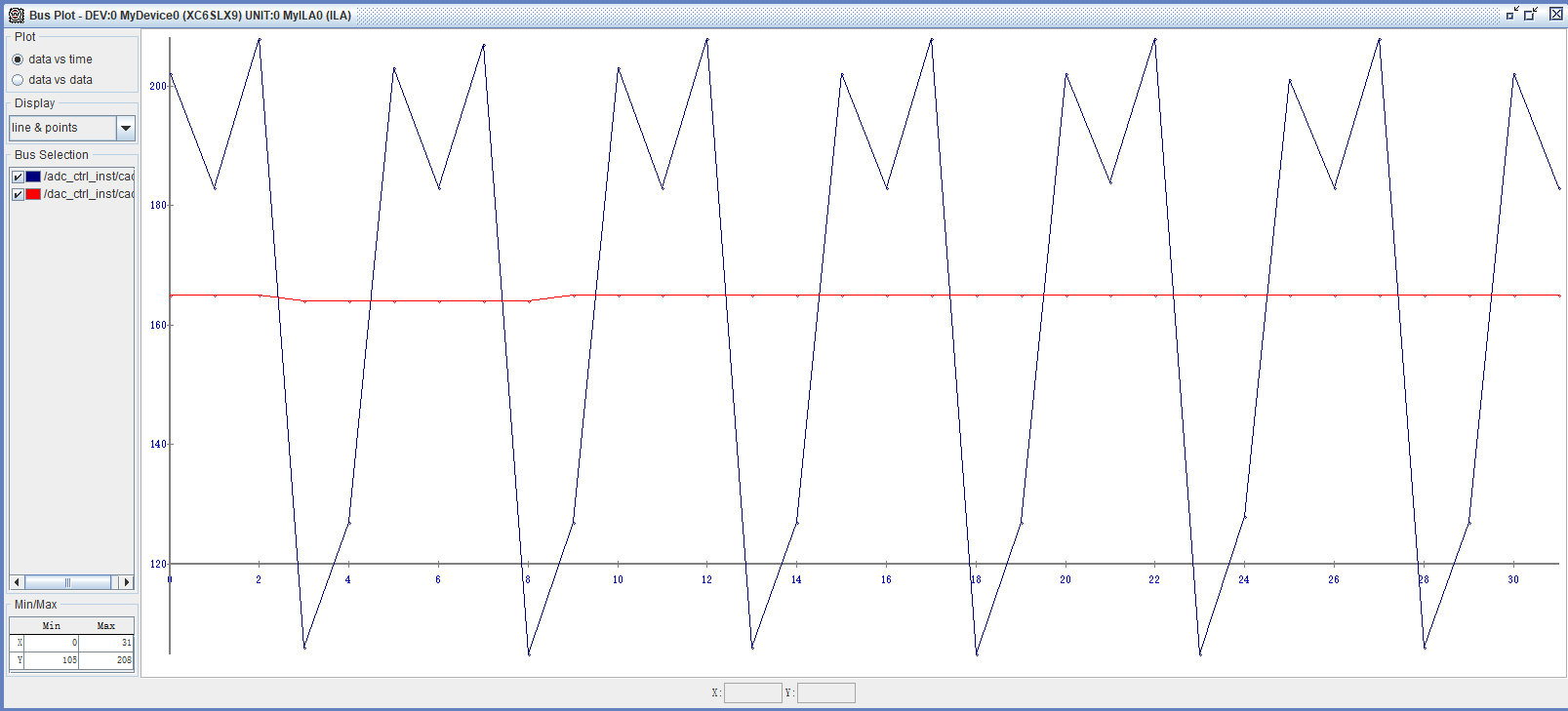
输入信号为2MHz时，达到了截止频率幅值有一定的衰减，达到了方波的0.707倍。

7）调整输入信号频率为5MHz

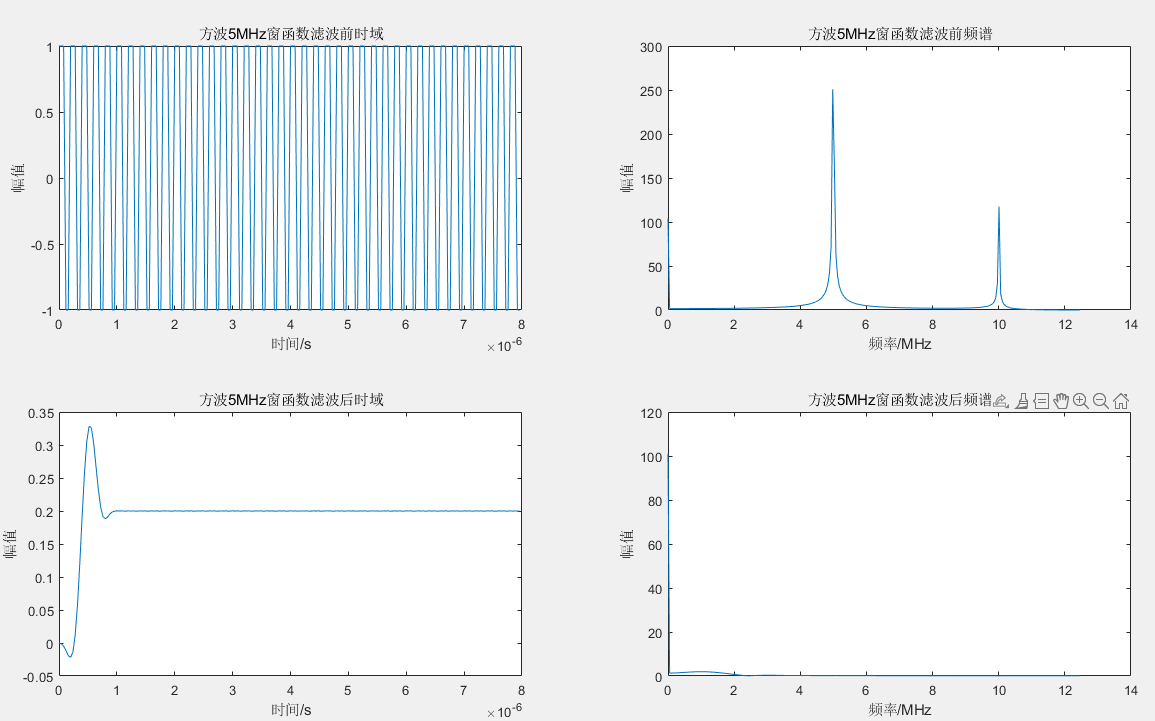
输入信号： 示波器获取的图像：



Chipscope获取波形（蓝色为滤波前ADC捕获，红色为滤波后DAC输出）：



MATLAB仿真图像：



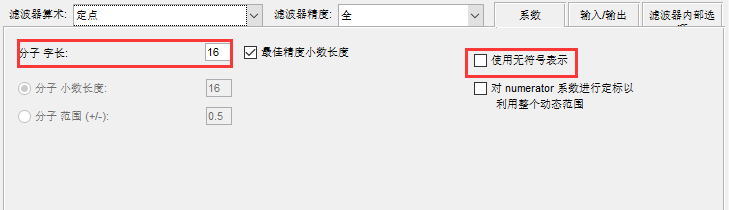
输入信号在5MHz时，所有频率分量全部在带外，只剩下很小的偏置。

1.3注意可能出现的问题

1.3.1 Matlab

1)在生成.coe文件时，要注意选择定点类型，这样才可以顺利导出抽头系数。

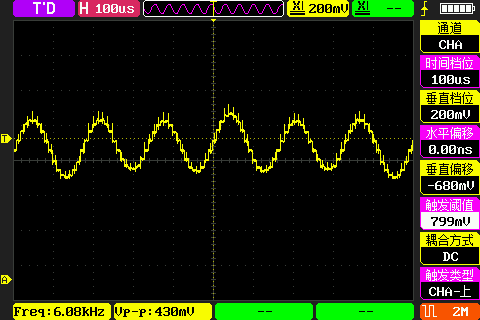
2）在选择量化时，字长和数种类保持默认，关键点是与后面FPGA中的设置保持一致，如果这里改，相应的在IP核设置时候也要改。



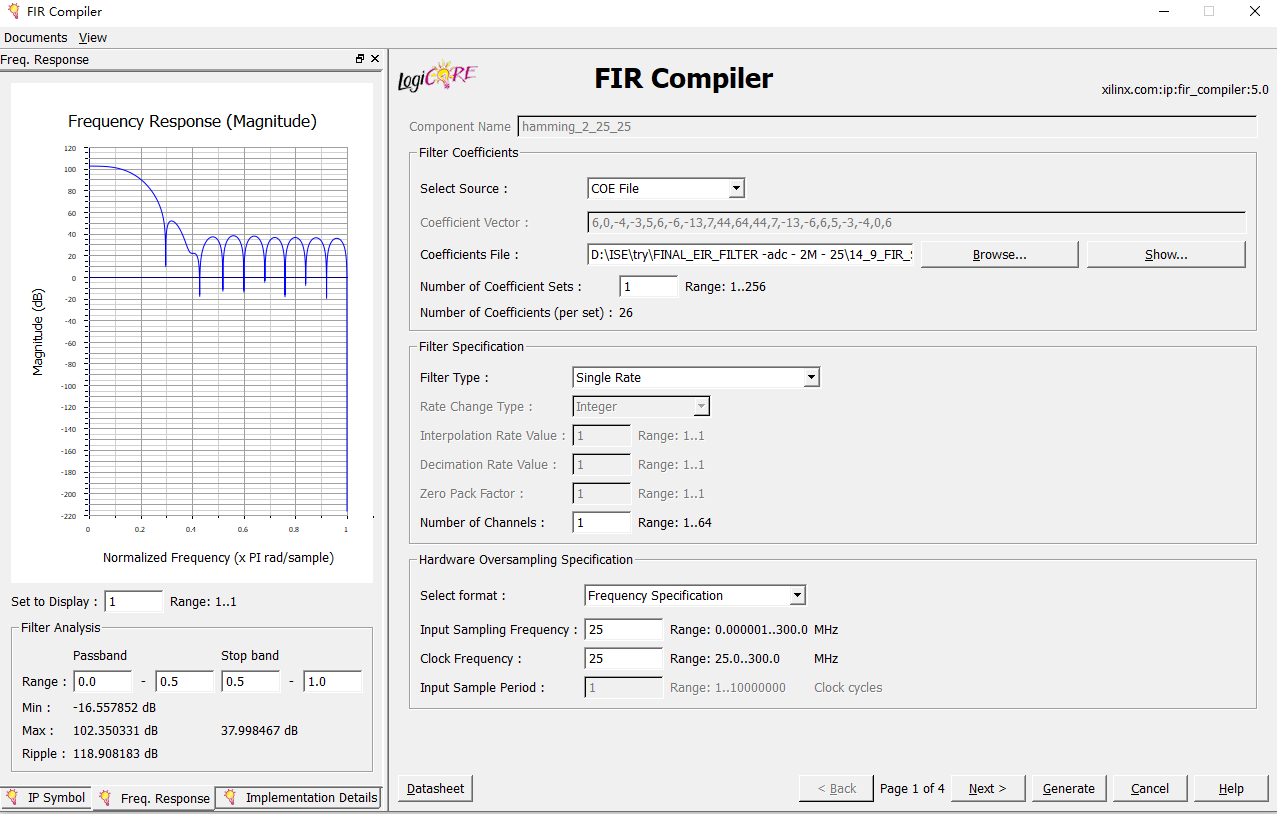
3）关于选择何种窗函数，在数字信号处理课堂上，会详细区分各种窗函数之间的区别，但在实际工程中，区别不太大，所以根据指标需要选择窗函数，本例中选择hamming窗。

1.3.2 FPGA

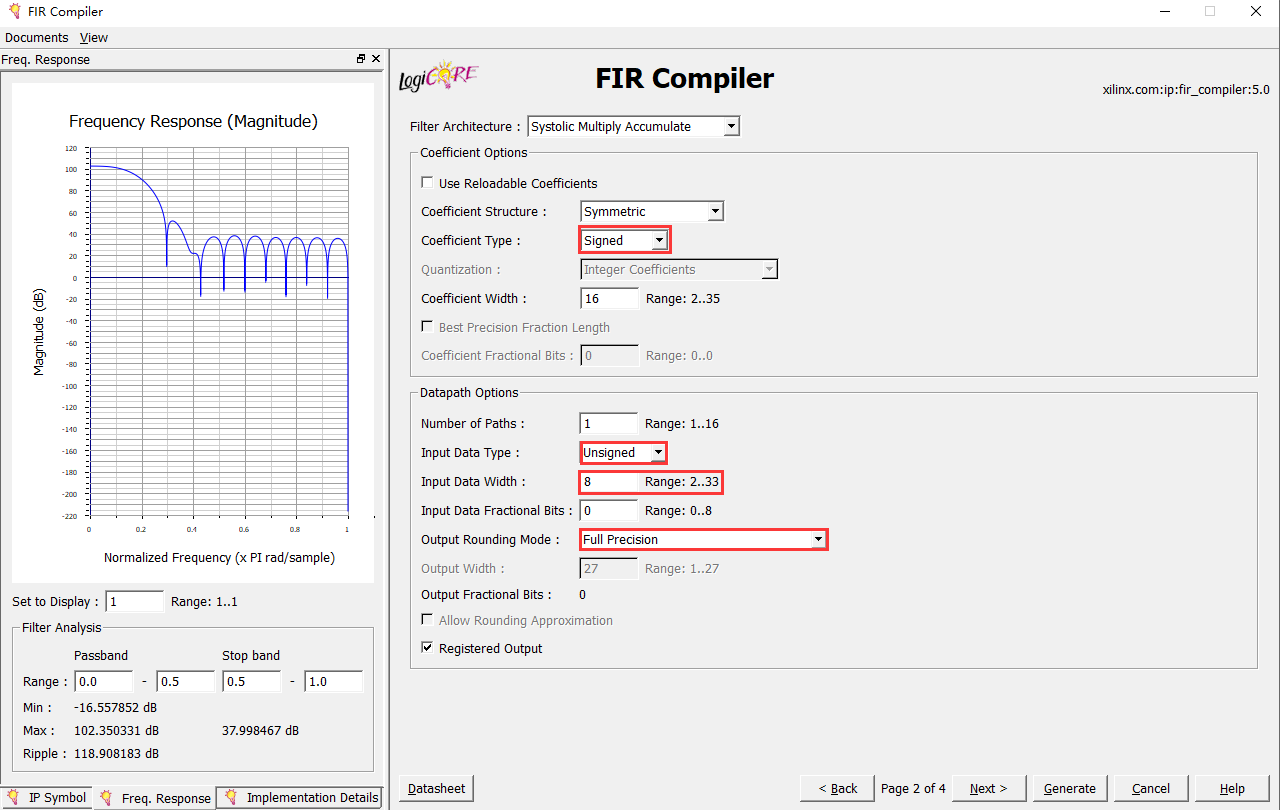
1）在生成IP核时，一定一定要注意时钟，要注意采样频率和matlab中的保持一致，且整个框架的DAC,ADC都要统一时钟，否则会出现下面的输出波形：



原因是如果时钟不严格约束，势必会丢掉采样点和输出点，意味着不能完全采样或输出，就会出现上图所示的阶梯型的波形。



2）在生成IP核时，一定要注意数的类型，要注意和matlab中的保持一致，否则将不会输出波形（这一点比较难发现）：



3）关于滤波器输出截断的问题：

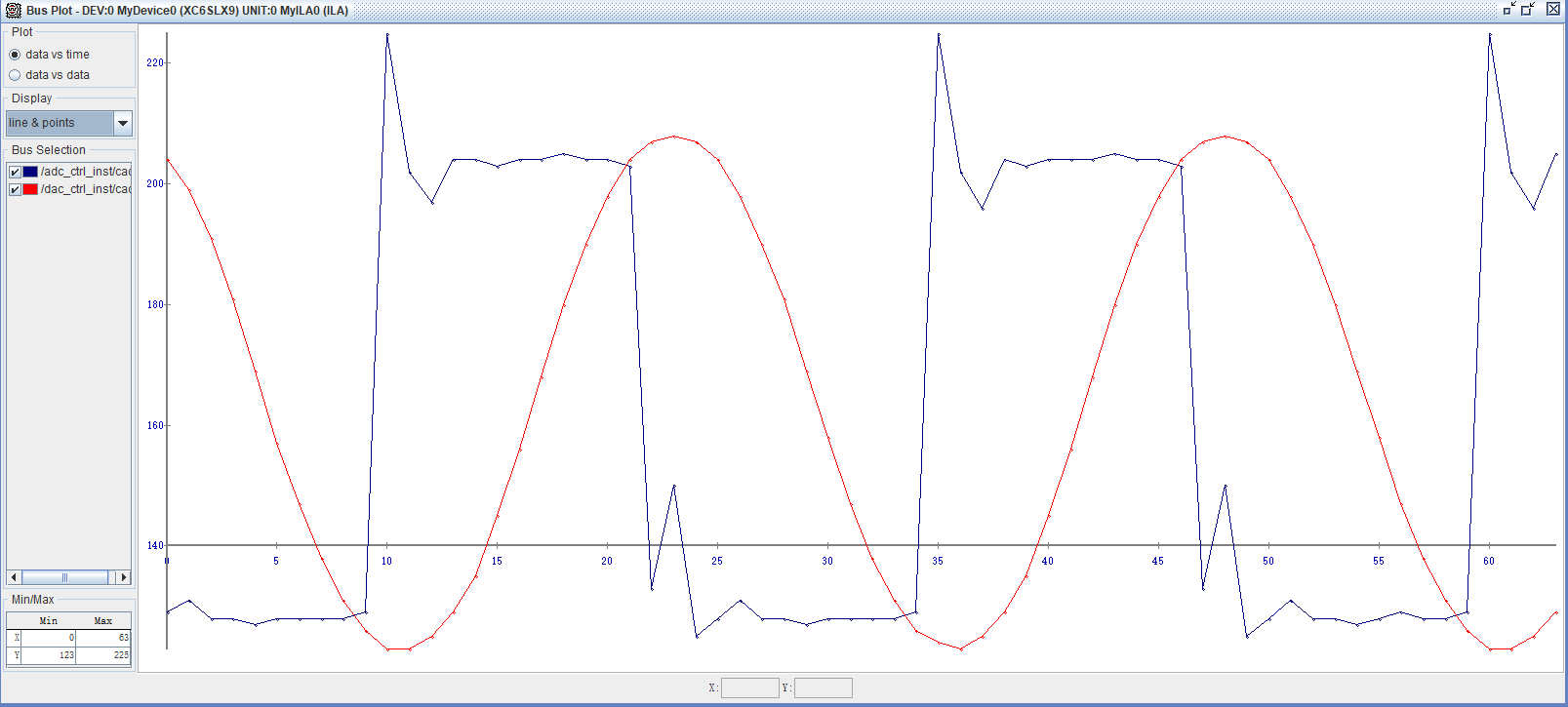
3.1）有必要全部输出吗？

有，因为如果在设置IP核时，选择了截断八位输出，系统直接默认高八位输出，这样前两位始终为零，使波形幅值四分。

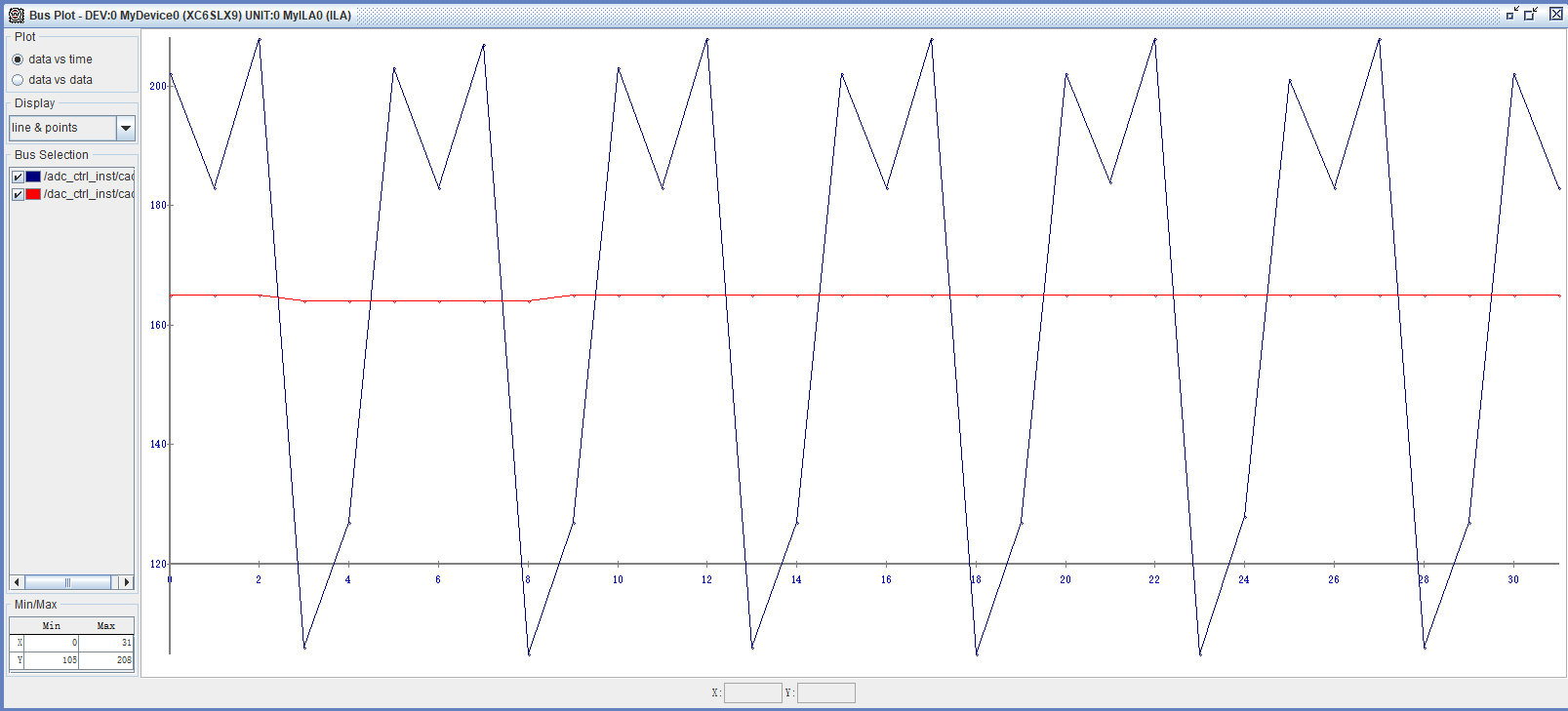
3.2）只能舍弃高两位吗？

这需要根据实际的输出波形图来观察，有时候会高三位均为零，所以建议下板子测试后再看如何截断。

4）有关chipscope采样波型很奇怪的解释：



上图是输入1MHz方波时的图像，之所以会产生如此不规整的方波，是因为在对1MHz采样时，用的采样频率是25MHz,也就意味着每个周期只能采到25个点，在chipscoe中显示时，点之间用直线连接，点少了，波形平滑度自然下降了，但根据奈奎斯特采样定理，每周期采两个点便可以获取波形的全部信息，所以可以看出输出波形也是正常的。此现象可以对照下图5MHz时，每周期采样5个点，方波非常失真。



1.4代码

1.4.1 Matlab

*N = 512; %fft的N*

*N1= N-1;*

*n = 0:N-1;*

*fs = 25e6; %25M采样率*

*t=(0:1/fs:N/fs); %信号源*

*fp = 1e6; %2M截止频率*

*x00= square(2\*pi\*0.1e6\*t); %绘制**方波100KHz*

*%x00= square(2\*pi\*0.2e6\*t); %绘制方波200KHz*

*%x00= square(2\*pi\*0.5e6\*t); %绘制方波500KHz*

*%x00= square(2\*pi\*1e6\*t); %绘制方波1MHz*

*%x00= square(2\*pi\*2e6\*t); %绘制方波2MHz*

*%x00= square(2\*pi\*5e6\*t); %绘制方波5MHz*

*y00 = fft(x00,N); %fft*

*m00 =abs(y00);*

*x01 = filter(hamming\_2\_25\_25,1,x00);*

*figure;*

*subplot(221);*

*plot(t(1:500),x00(1:500));*

*xlabel('时间/s'),ylabel('幅值');*

*title('方波100KHz窗函数滤波前时域’);*

*subplot(222);*

*plot(f(1:N/2)/1e6,m00(1:N/2));*

*xlabel('频率/MHz'),ylabel('幅值');*

*title(方波100KHz窗函数滤波前频谱’);*

*% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*

*subplot(223);*

*plot(t(1:500),x01(1:500));*

*xlabel('时间/s'),ylabel('幅值');*

*title('方波100KHz窗函数滤波后时域’);*

*subplot(224);*

*y01 = fft(x01,N);*

*m01 =abs(y01);*

*plot(f(1:N/2)/1e6,m01(1:N/2));*

*xlabel('频率/MHz'),ylabel('幅值');*

*title(方波100KHz窗函数滤波后频谱’);*

*Wn=fp/fs\*2;*

*[H1,w1] = freqz(hamming\_2\_25\_25,1,512); %分析频响*

*figure();*

*plot(w1/pi,abs(H1),... %绘制实际幅频响应*

*[0 Wn Wn 1],[1 1 0 0],'r'); %期望幅频响应*

*xlabel('Normalized Frequency (\times\pi rad/sample)')*

*ylabel('Magnitude')*

*axis([0 1 -0.2 1.2]);*

*legend('LPF幅频响应','期望幅频响应');*

*title('FREQUENCY DOMAIN')*

1.4.2 FPGA

1）ADC:

*module adc\_ctrl(*

*input clk,*

*input rst\_n,*

*input [7:0]ad\_data,*

*output [7:0]ad\_pre\_data,*

*output clk\_adc*

*);*

*assign clk\_adc = clk;*

*reg [7:0] cach;*

*always@(posedge clk or negedge rst\_n)*

*begin*

*if(rst\_n == 0)*

*cach <= 0;*

*else*

*cach <= ad\_data[7:0];*

*end*

*assign ad\_pre\_data = cach;*

*endmodule*

2）DAC:

*module dac\_ctrl(*

*input clk,*

*input rst\_n,*

*input [7:0] da\_data,*

*output [7:0] da\_pre\_data,*

*output clk\_dac*

*);*

*assign clk\_dac = clk ;*

*reg [7:0] cach;*

*always@(posedge clk or negedge rst\_n)*

*begin*

*if(rst\_n == 0)*

*cach <= 0;*

*else*

*cach <= da\_data[7:0];*

*end*

*assign da\_pre\_data = cach;*

*endmodule*

3）约束:

*NET "clk" LOC = T8;*

*##*

*NET rst\_n LOC = L3 | IOSTANDARD = "LVCMOS33"; ## reset pushbutton*

*##*

*########AD/DA Extended IO(J3)#####################*

*//DA*

*NET clk\_dac LOC = A5 | IOSTANDARD = "LVCMOS33" | SLEW=FAST; ## 13*

*NET dac\_pre\_data<0> LOC = A10 | IOSTANDARD = "LVCMOS33" | SLEW=FAST; ## 13*

*NET dac\_pre\_data<1> LOC = A9 | IOSTANDARD = "LVCMOS33" | SLEW=FAST; ## 12*

*NET dac\_pre\_data<2> LOC = C8 | IOSTANDARD = "LVCMOS33" | SLEW=FAST; ## 11*

*NET dac\_pre\_data<3> LOC = A8 | IOSTANDARD = "LVCMOS33" | SLEW=FAST; ## 10*

*NET dac\_pre\_data<4> LOC = B8 | IOSTANDARD = "LVCMOS33" | SLEW=FAST; ## 9*

*NET dac\_pre\_data<5> LOC = A7 | IOSTANDARD = "LVCMOS33" | SLEW=FAST; ## 8*

*NET dac\_pre\_data<6> LOC = A6 | IOSTANDARD = "LVCMOS33" | SLEW=FAST; ## 7*

*NET dac\_pre\_data<7> LOC = B6 | IOSTANDARD = "LVCMOS33" | SLEW=FAST; ## 6*

*NET clk\_adc LOC = C10 | IOSTANDARD = "LVCMOS33" | SLEW=FAST; ## 13*

*NET adc\_pre\_data<0> LOC = B15 | IOSTANDARD = "LVCMOS33" | SLEW=FAST; ## 13*

*NET adc\_pre\_data<1> LOC = B16 | IOSTANDARD = "LVCMOS33" | SLEW=FAST; ## 12*

*NET adc\_pre\_data<2> LOC = C15 | IOSTANDARD = "LVCMOS33" | SLEW=FAST; ## 11*

*NET adc\_pre\_data<3> LOC = C16 | IOSTANDARD = "LVCMOS33" | SLEW=FAST; ## 10*

*NET adc\_pre\_data<4> LOC = D16 | IOSTANDARD = "LVCMOS33" | SLEW=FAST; ## 9*

*NET adc\_pre\_data<5> LOC = E15 | IOSTANDARD = "LVCMOS33" | SLEW=FAST; ## 8*

*NET adc\_pre\_data<6> LOC = C9 | IOSTANDARD = "LVCMOS33" | SLEW=FAST; ## 7*

*NET adc\_pre\_data<7> LOC = E11 | IOSTANDARD = "LVCMOS33" | SLEW=FAST; ## 6*

*PIN "clk20\_inst/clkout1\_buf.O" CLOCK\_DEDICATED\_ROUTE = FALSE;*

*PIN "clk20\_inst/clkout2\_buf.O" CLOCK\_DEDICATED\_ROUTE = FALSE;*

*NET "clk" TNM\_NET = clk;*

*TIMESPEC TS\_clk = PERIOD "clk" 20 ns HIGH 50%;*

1. [↑](#endnote-ref-1)