
1 AD9767 高速 DAC 的 DDS 信号发生器设计

工程源码	-- -----DDS_AD9767 (优化 125M 带仿真)
相关视频课程	
	如果您手头的硬件不支持本实验，您可以学习本实验的理论内容，也可以跳过本节内容，继续后续内容的学习。

章节导读

本节将介绍基于 ACM9767 模块的 DDS 数字信号发生器原理及设计方法, 通过本节课程的学习，读者能够对基于高速 DA 转换器的数字信号发生器设计有更深入的理解。

1.1 DDS 概述

DDS(Direct Digital Synthesizer)即数字合成器, 是近年来发展起来的一种新的频率合成技术,其主要优点是相对带宽很大, 频率转换时间极短（可小于 20 ns), 频率分辨率很高, 全数字化结构便于集成, 输出相位连续可调, 且频率、相位和幅度均可实现程控。随着 FPGA 技术的不断发展, 该技术得到愈加成熟的应用。借助 FPGA 平台开发的高性能的 DDS 发生器与基于 DDS 芯片设计的信号发生器相比, 具有成本更低, 操作更加灵活, 而且还能根据要求在线更新配置等诸多优点。同时, 整个系统开发流程更趋于软件化、自定义化, 开发周期更短, 调试过程更加简单。

正是由于其便捷的频率、相位以及幅度的数控优势, 基于 FPGA 控制的 DDS 信号发生器有其广泛的应用前景。

1.2 ACM9767 模块概述

提到基于 FPGA 的 ACM9767 模块波形信号发生器设计, 就不得不先对 ACM9767 模块进行简单的介绍。

ACM9767 模块使用的是 ADI 公司 AD9767 型 DAC 芯片。该芯片支持 I、Q 输出模式（该模式常用于数字通信领域）。输出形式为差分电流输出, 输出电流满量程范围为可设置为 2~20mA。芯片本身自带 1.2V 的参考电压, 无需外部提供参考源。

继承了 AD9767 芯片的优异性能, ACM9767 模块是一款高性能高速双通道 DAC 模块。模块具有单电源 5V 供电输入, 双通道数字转模拟信号输出, 每个通道数据分辨率为 14 位, 输出电压范围为±5V, 且转换速率高达 125Msps,

非常适合诸如信号发生器、数字调制通信系统的开发等应用。

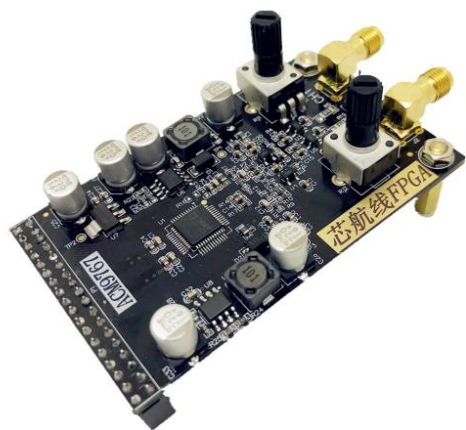


图 1-1 ACM9767 模块样图

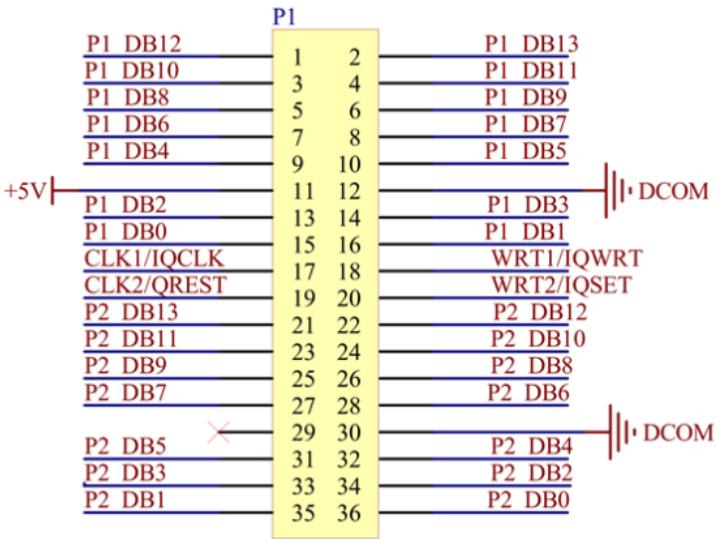


图 1-2 ACM9767 模块插接管脚信号含义

模块对用户提供一个 36 针的排母接口，可直接插接到市面上各种常见的 FPGA 开发板（Terasic 公司 DE2、DE1、DE0，芯路恒科技（小梅哥 FPGA）所有的 FPGA 开发板，包含 ACZ702、ACX720、AC620、AC6102、AC601、AC609 等）上进行使用，而无需使用任何转接线转接。

本次系统设计将在小梅哥团队出品的 ACZ702 开发板上进行。在使用时，将 ACM9767 模块插接到 ACZ702 开发板的 40pin 通用 GPIO 接口上，即可实现 FPGA 和 ACM9767 模块的数字接口互通。由于在 ACM9767 模块模拟量输出峰值范围内，输出的模拟量电压值可以由输入的数字量信号完全控制，这样，就可以通过 FPGA 完全控制数字量的输出值，实现满足预定指标的模拟信号波形输出。

基于此，借助 ACM9767 模块和 ACZ702 开发板，我们可以实现不同频率、幅度的正弦波、三角波、方波信号的输出。

考虑到设计的应用场景，我们还可以在设计中加入 VIO 核，实时控制波形类型以及波形频率、幅度的变化。

接下来，我们将从典型的 DDS 信号发生器信号发生流程图，着手以上设计功能的实现。

1.3 系统原理及整体设计

1.3.1 DDS 信号发生器的系统设计原理

介绍完 DDS 信号发生的应用背景和 ACM9767 的模块硬件性能，接下来我们来共同分析和探讨基于 FPGA 的 DDS 信号发生器的实现过程。

DDS 是如何实现控制发生信号的呢？如果想生成一个质量较好的 DDS 发生信号，无非是对波形、频率、相位这几个关键要素进行控制和搭配组合。下面是其中一种常用的 DDS 控制信号发生器的实现方案原理图。

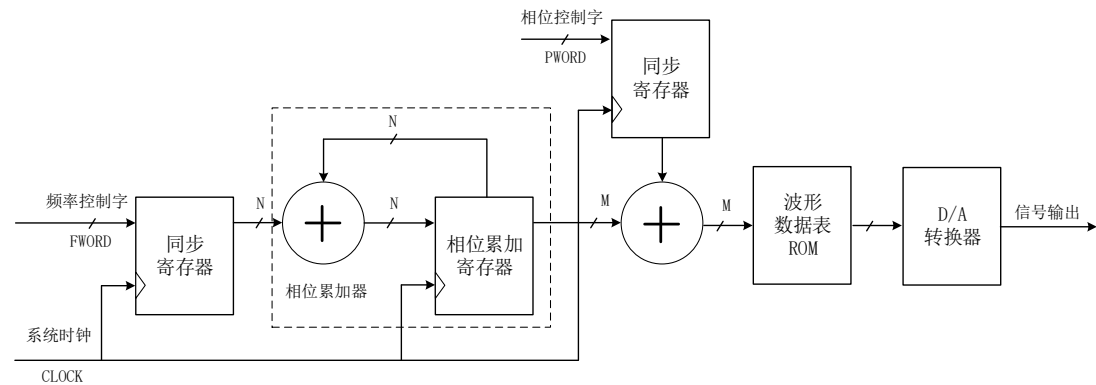


图 1-3 DDS 控制信号发生原理图

由图可以看出，DDS 主要由相位累加器、相位调制器、波形数据表以及 D/A 转换器构成。其中相位累加器由 N 位加法器与 N 位寄存器构成。每个时钟周期的时钟上升沿，加法器就将频率控制字与累加寄存器输出的相位数据相加，相加的结果又反馈至累加寄存器的数据输入端，以使加法器在下一个时钟脉冲的作用下继续与频率控制字相加。这样，相位累加器在时钟作用下，不断对频率控制字进行线性相位累加。即在每一个时钟脉冲输入时，相位累加器便把频率控制字累加一次。

相位累加器输出的数据就是合成信号的相位。相位累加器的溢出频率，就是 DDS 输出的信号频率，相位累加器输出的数据，作为波形存储器的相位采样地址，这样就可以把存储在波形存储器里的波形采样值经查表找出，完成相位到幅度的转换。

波形存储器的输出数据送到 D/A 转换器，由 D/A 转换器将数字信号转换成模拟信号输出。

DDS 信号流程示意图如下图所示：

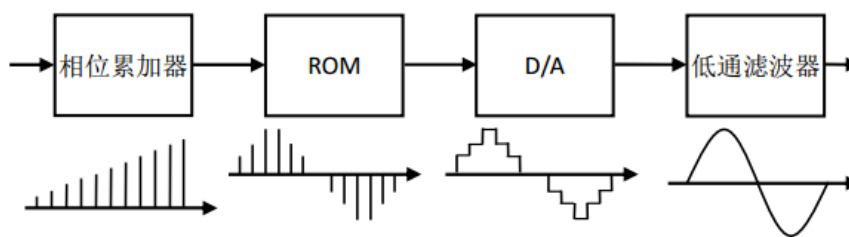


图 1-4 DDS 信号流程示意图

这里相位累加器位数为 N 位 (N 的取值范围实际应用中一般为 24~32)，相当于把正弦信号在相位上的精度定义为 N 位，所以其分辨率为 $1/2^N$ 。

若 DDS 的时钟频率为 F_{clk} ，频率控制字 f_{word} 为 1，则输出频率为 $F_{\text{out}} = \frac{F_{\text{clk}}}{2^N}$ ，这个频率相当于“基频”。若 f_{word} 为 B ，则输出频率为 $F_{\text{out}} = B \times \frac{F_{\text{clk}}}{2^N}$ 。

因此理论上由以上三个参数就可以得出任意的 f_o 输出频率。且可得出频率分辨率由时钟频率和累加器的位数决定的结论。当参考时钟频率越高，累加器位数越高，输出频率分辨率就越高。

从上式分析可得，当系统输入时钟频率 F_{clk} 不变时，输出信号频率由频率控制字 B 所决定，由上式可得： $B = 2^N \times \frac{F_{\text{out}}}{F_{\text{clk}}}$ 。其中 B 为频率字且只能取整数。为了合理控制 ROM 的容量，此处选取 ROM 查询的地址时，可以采用截断式，即只取 32 位累加器的高 M 位。这里相位寄存器输出的位数一般取 10~16 位。

1.3.2 DDS 信号发生器原理讲解举例

以上通过理论计算加数据变换的形式对 DDS 原理进行了较为严谨的公式推导解释，但是如果从采样点选取的角度分析，DDS 究竟是怎么实现频率和相位的控制的呢？以下通过一个简化的实例来描述 DDS 实现频率和相位控制的过程。

如下图为一个完整周期的正弦信号的波形，总共有 33 个采样点，其中第 1 点和第 33 点的值相同，第 33 点为下一个周期的起始点，因此，实际一个周期为 32 个采样点 (1~32)。

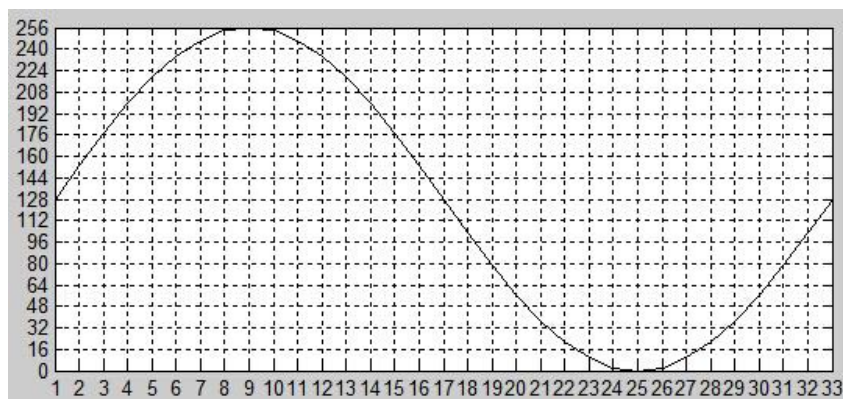


图 1-5 完整周期的正弦信号波形示例

例如：当使用 FPGA 控制 DAC 输出一个周期的正弦信号时，每 1ms 输出一个数值。如果每个点都输出，则总共输出这一个完整的周期信号需要输出 32 个点，因此输出一个完整的信号需要 32ms，可知输出信号的频率为 $1000/32\text{Hz}$ 。

如果需要用这一组数据来输出一个 $2 * (1000/32)\text{Hz}$ 的正弦信号，因为输出信号频率为 $2 * (1000/32)\text{Hz}$ ，那么输出一个完整的周期的正弦波所需要的时间为 $32/2$ ，即 16ms。为了保证输出信号的周期为 16ms，我们需要对我们的输出策略进行更改，上面输出周期为 32ms 的信号时，我们采用的为逐点输出的方式，以 32 个点来输出一个完整的正弦信号，而我们 FPGA 控制 DAC 输出信号的频率固定为 1ms。因此，我们要输出周期为 16ms 的信号，只能输出 16 个点来表示一个完整的周期。我们就选择以每隔一个点输出一个数据的方式来输出即可。我们可以选择输出 (1、3、5、7……29、31) 这些点，因为采用这些点，我们还是能够组成一个完整周期的正弦信号，而输出时间缩短为一半，即频率提高了一倍。最终结果如下图所示：

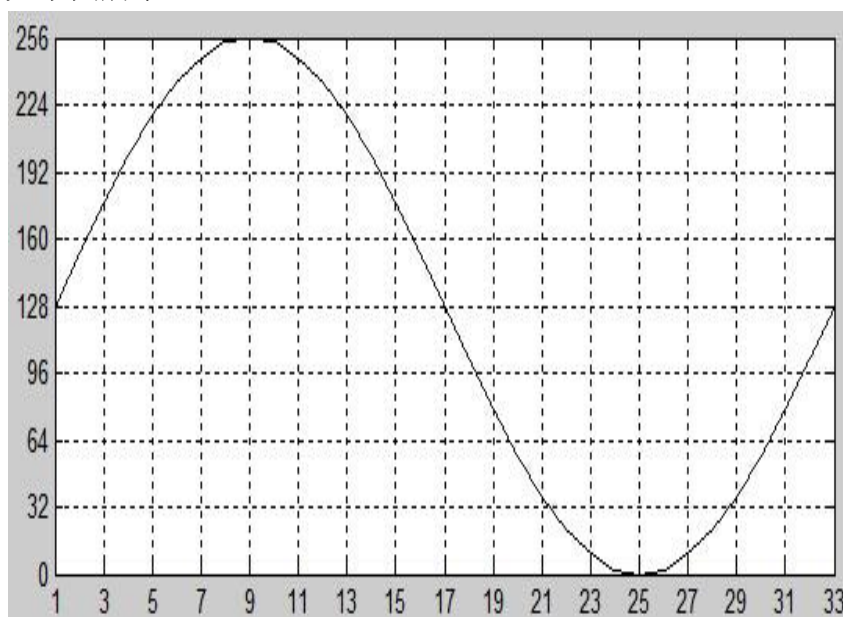


图 1-6 频率提高 1 倍后的正弦输出信号

如果需要使用该组波形数据输出频率为 $(1/2) * (1000/32)$ Hz 的信号，即周期为 64ms，则只需要以此组数据为基础，每 2ms 输出一个数据即可，例如第 1ms 和第 2ms 输出第一个点，第 3ms 和第 4ms 输出第二个点，以此类推，第 63ms 和第 64ms 输出第 32 个点，即可实现周期加倍，即频率减半的效果。

概括来说，一个周期的离散点数值表格一定，如果想提高输出频率，则采用跳过若干点位（1,3,5.....）的方法，重构原波形；如果想降低输出频率，则采用拉长每一个离散点的输出时间(1,1,2,2,3,3.....)，来完成输出波形不变，而扩大时钟周期的输出效果。时钟周期扩大，则频率降低。

对于相位的调整，则更加简单。只需要在每个取样点的序号上加上一个偏移量，便可实现相位的控制。例如，上述一个周期 32 个点均匀的将 360 度等分，上面默认的是第 1ms 时输出第一个点的数据，假如我们现在在第 1ms 时从第 9 个点开始输出，则将相位左移了 90 度，这就是控制相位的原理。

在本次设计中，对于一个周期的离散点数值表格，我们实现 DDS 输出时，将横坐标上的数据作为 ROM 的地址，纵坐标上的数据作为 ROM 的输出，那么指定不同的地址就可实现对应值的输出。而我们 DDS 输出控制频率和相位，归结到底就是控制 ROM 的读出地址。

在理解了 DDS 的实现原理之后，接下来便可以开始本次系统的设计实现了。

1.3.3 信号发生控制方案设计框图

介绍完 DDS 信号发生的原理，并进行了举例后，接下来将进行 DDS 信号发生器的整体设计。

根据以上原理分析，基于 ACZ702 开发板，我们可以作如下架构设计：

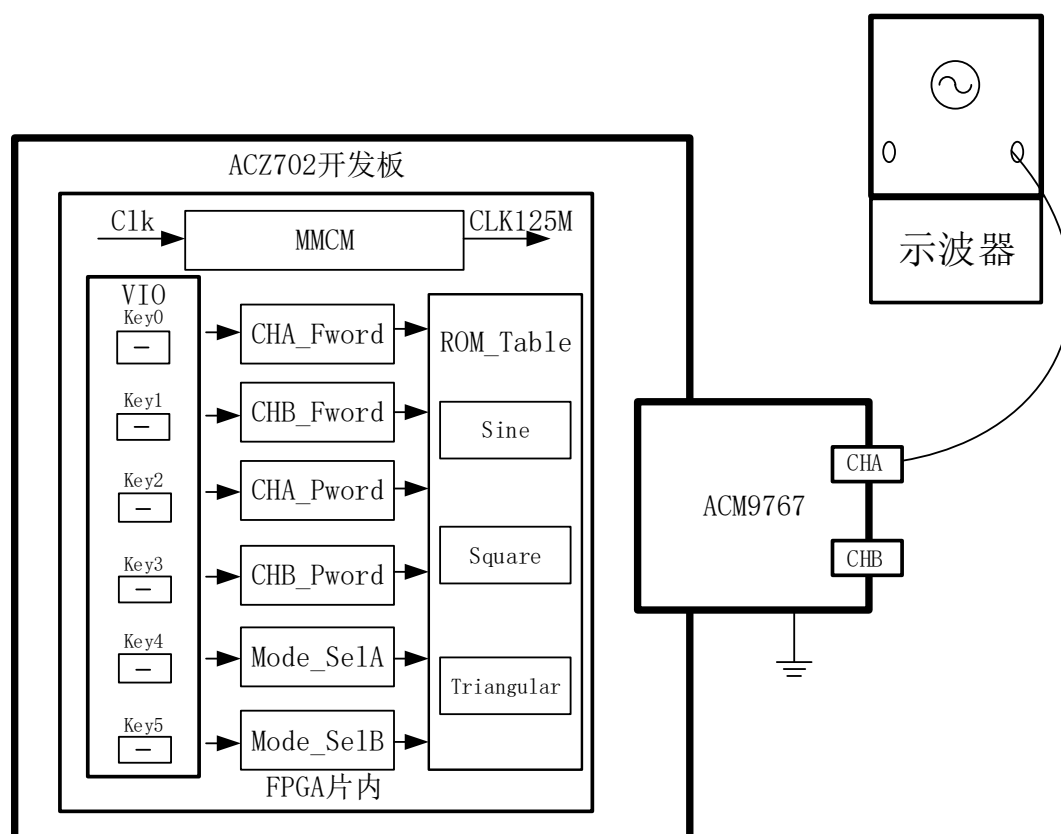


图 1-7 基于 ACZ702 开发板的 ACM9767 模块系统框图

为了验证频率和相位可以在一段区间内实现变化这一特性，设计中通过 VIO 核输出多个控制信号控制档位切换的方式，设定几个可以跳变切换的典型值以验证频率相位的实时变化效果。为了方便操作，我们可以将这些输出信号设定为按钮触发模式。通过点击按钮按输出高脉冲的方式，设定几个可以跳变切换的典型值以验证频率相位的实时变化效果。实现上来看，我们在设计时可以先按如下规律实现设计：

1. 程序上电后给出默认频率和相位初始值
2. 每当按下一次按键扩大频率 10 倍，
3. 每当按下一次按键，相位跳变一个特定的角度
4. 每当按下一次按键，切换一次波形类型

其中，key0、key1、key2、key3、key4、key5 分别作为通道 A，通道 B 的频率、相位、波形类型控制字的模拟切换按钮。

明确了频率和相位控制字切换的设计方案后，还需要明确波形信号的存储和读取策略。为了便于波形数据取用的统一管理，对于 3 种不同类型的波形数据存储，我们可以为每个信号发生通道开辟 3 个 ROM 空间。这三个 ROM 空间分别

存储正弦波、方波和三角波的离散信号值作信号发生时查表使用。

在设计之初,设计人员也许有过这样的思考:我能否只开辟一块 ROM 空间,然后输出哪种波形,就装载哪种波形的 ROM 表?这样做可以节省 FPGA 片上的 ROM 空间资源,让更多 ROM 空间资源预留给其他需要进行缓存的设计单元。

但是,这样做会产生如下问题:熟悉 FPGA 的 ROM IP 核的用户都知道,切换 ROM 表的装载文件,即 coe 文件,不是一件容易的事情,每切换一次 coe 文件,都需要对 ROM IP 核进行 coe 文件的重新装载,并重新编译 IP 核和工程。这样看来,要想让已经完成编译并下载到 FPGA 开发板内的工程进行实时的发生信号波形切换,就变成了难以完成的任务。所以,从波形实时切换和选择的角度,只有并行开辟三块 ROM 空间并加载 coe 文件,然后同时读取波形数据,并在最终输出端进行输出波形数据选择,才能实现信号的实时切换功能。

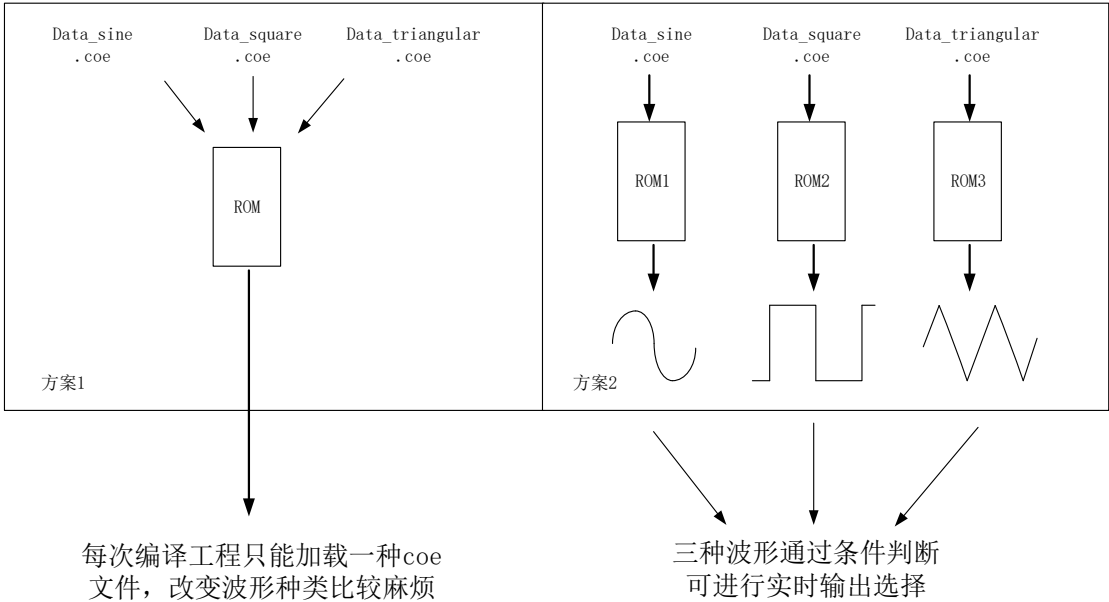


图 1-8 方案 1 和方案 2 模块架构对比分析

通过以上分析比较发现,使用方案 2 比使用方案 1 虽然占用更多的 ROM 资源,但输出波形的实时响应性能更优秀,所以在 ROM 资源充沛的情况下,优先选用方案 2。

1.4 系统各模块设计

在有了系统总体框架思路后,接下来即可着手对模块各个击破。经过上述分析,我们在本次设计中需要利用 VIO 控制输出波形,同时,我们需要配置锁相环模块、信号离散值 ROM 表、以及对 DDS 驱动模块进行设计。

1.4.1 锁相环模块

由于 AD9767 的工作时钟频率为 125MHz，所以这里我们需要利用 FPGA 内部的锁相环为 ACM9767 模块以及其 FPGA 配套的驱动模块提供相同的工作频率。这样，在数据交互时，能确保时钟域相同。

锁相环的配置方法我们也讲过多次，我们也就不再进一步介绍。这里，我们给出配置界面供参考即可。

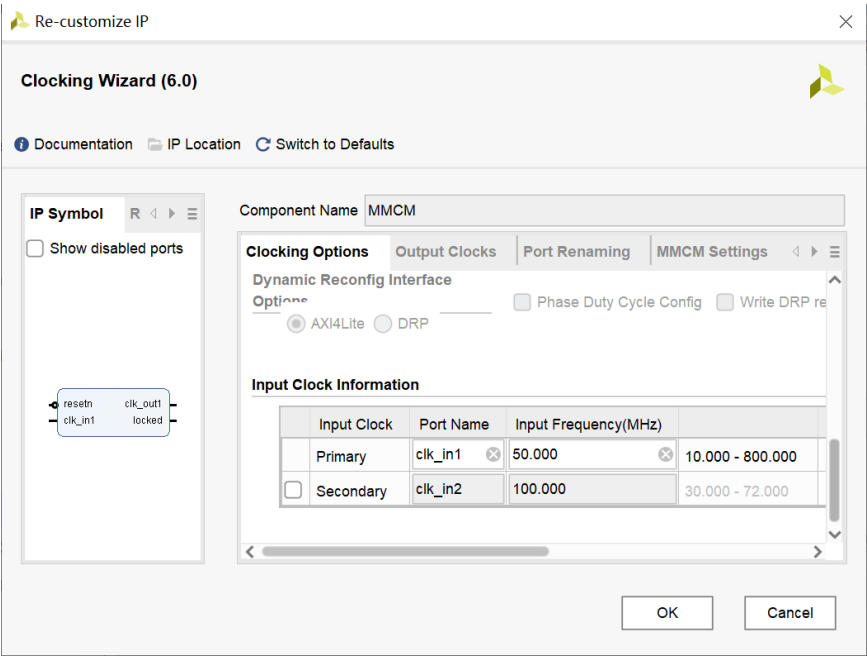


图 1-9 锁相环配置 1

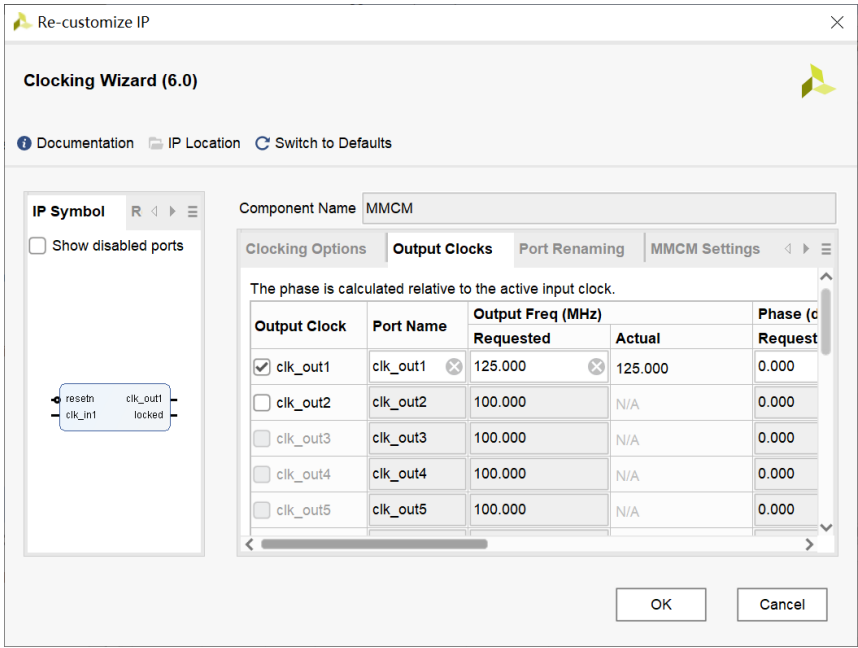


图 1-10 锁相环配置 2

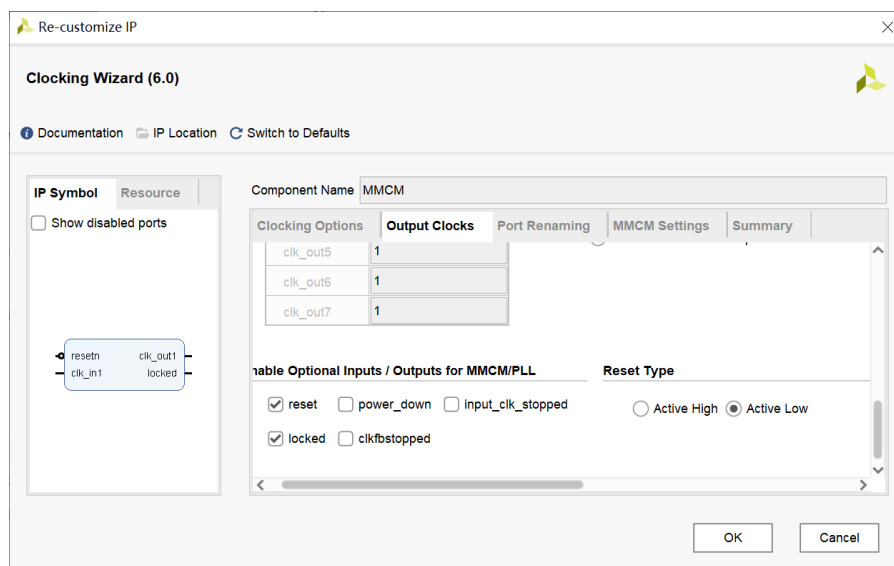


图 1-11 锁相环配置 3

完成配置后，即可利用锁相环将 50M 主时钟频率，通过分频倍频而获得我们需要的 125MHz 时钟。

1.4.2 波形数据存储单元

为了能产生相应的波形，我们需要借助 ROM 来分别存储正弦波、方波、三角波的波形数据。其中单端口的 ROM 主要配置数据如下图所示，其初始化文件选为已经生成的相应波形的 coe 文件。此处 coe 位宽为 14，深度为 4096。

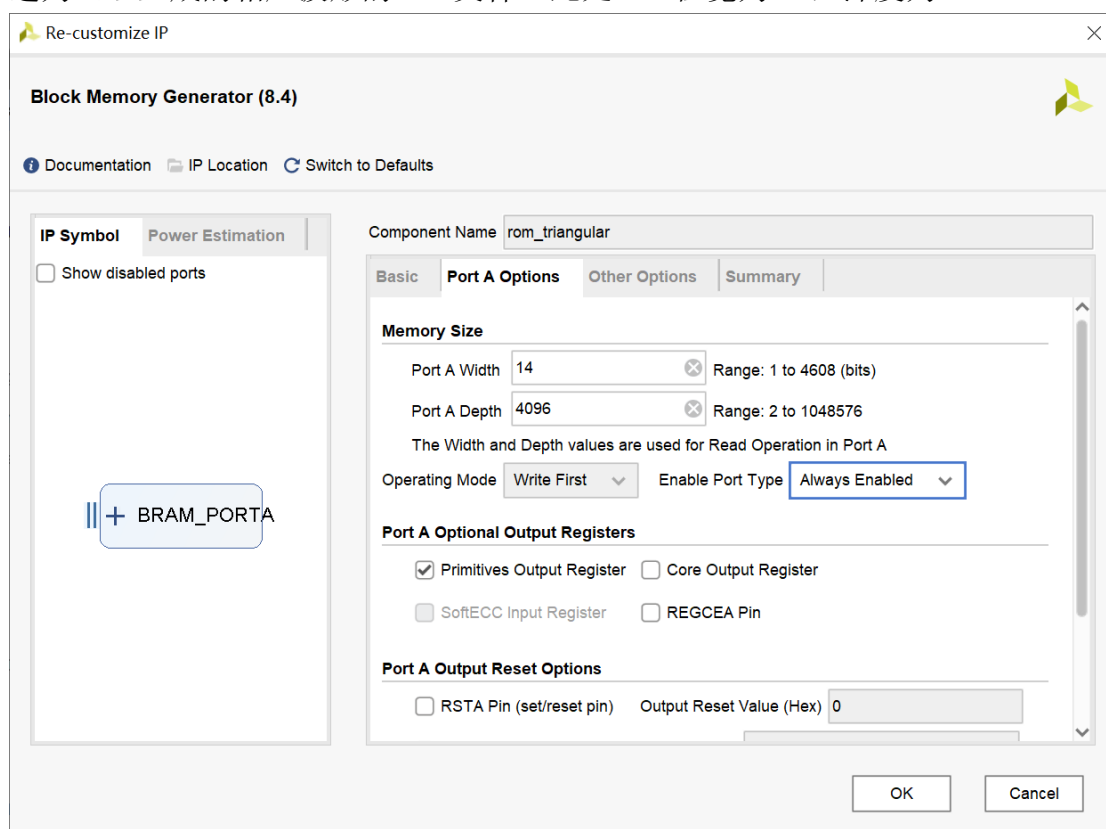


图 1-12 波形数据存储 ROM 配置表

关于生成 coe 文件需要注意的是，由于本次设计使用的是 Xilinx 出品的芯片，所以在使用 Mif 精灵生成该文件时，应选择 Xilinx。

同时，由于 AD 芯片的数字位宽为 14 位，所以 maxi 中数据改为 16383（即 $2^{14} - 1$ ）可以确保 FPGA 输出的二进制值，都能反映到输出模拟量实时输出值的变化之中，其余设置如下图所示：

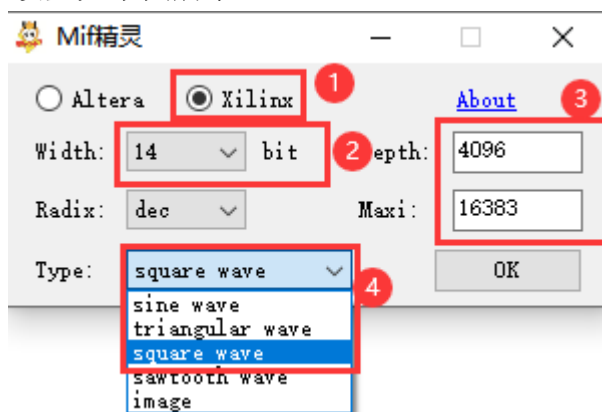


图 1-13 使用 Mif 精灵生成波形信号参考图示

1.4.3 ACM9767 驱动模块设计

在本设计中参考时钟 F_{clk} 频率为 125 MHz，相位累加器位数 N 取 32 位，频率控制字位数 M 取 12 位。经过以上的分析，可以得出 DDS 模块的端口示意图如图 1-14 所示。

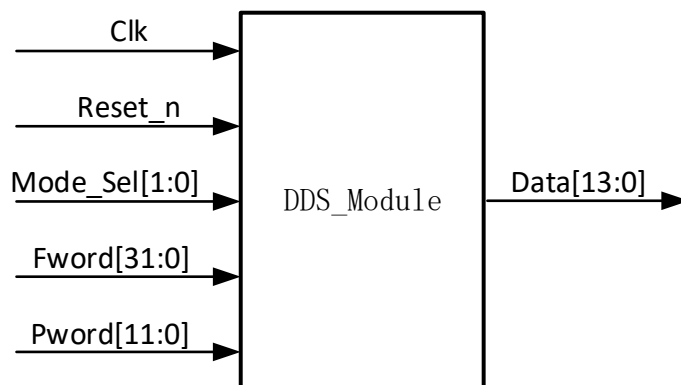


图 1-14 ACM9767 驱动模块端口

其中，每个端口的功能描述如下表所示：

表 1-1 ACM9767 驱动模块端口列表

端口名称	I/O	功能描述
Clk	I	为本模块的工作时钟，频率为锁相环输出的 125MHz
Rst_n	I	控制器复位，低电平复位
Model_Sel	I	波形选择
FWord	I	频率控制字

PWord	I	相位控制字
Data	O	数据输出

为研究简便，我们以双通道信号发生器的其中一个通道作为研究对象，而另一个通道与之完全相同。实际在 FPGA 驱动 ACM9767 工作过程中，FPGA 设计上只需重复例化驱动模块一次，在管脚绑定时只需按双通道对应的数字信号驱动管脚驱动外部 ACM9767 模块即可。

接下来，开始驱动模块的代码设计讲解。

为了便于对频率和相位控制字的后期处理，在驱动模块之中必须先对频率和相位值进行缓存。其代码如下：

```
//频率控制字同步寄存器
reg [31:0]Fword_r;
always@(posedge Clk)
    Fword_r <= Fword;

//相位控制字同步寄存器
reg [11:0]Pword_r;
always@(posedge Clk)
    Pword_r <= Pword;
```

实现波形数据的还原，其实质就是按频率控制字每个时钟周期进行累加，进而决定下一个时钟周期应该偏移多少个周期切分的单元，从而作为下一次还原的数据。

```
//相位累加器
reg [31:0]Freq_ACC;
always@(posedge Clk or negedge Reset_n)
    if(!Reset_n)
        Freq_ACC <= 0;
    else
        Freq_ACC <= Fword_r + Freq_ACC;
```

需要注意的是，在该模块设计中我们直接截取 32 位累加器结果中的高 12 位作为 ROM 的查询地址，这样，查询地址也是 4096 个点，和 ROM 表保存的单周期的离散点存储位深是一致的。这样产生的误差虽然会对频谱纯度有影响，但是对波形的精度的影响是可以忽略的。

```
//波形数据表地址
wire [11:0]Rom_Addr;
assign Rom_Addr = Freq_ACC[31:20] + Pword_r;
```

由于本设计有两个输出通道，所以将该模块例化两次，以实现双通道的输出。

本工程设计共创建了 3 种典型的发生信号模型，如果希望输出哪一种波形，则进行对应的模式选择即可。从这个角度来看，每个通道的 3 块 ROM 空间，实际上都在同时读取数据和输出数据，至于最终呈现在 FPGA 输出端的到底是何

种波形的数据，则由模式选择信号 Mode_Sel 决定。

```
always@(*)
    case(Mode_Sel)
        0:Data = Data_sine;
        1:Data = Data_square;
        2:Data = Data_triangular;
        3:Data = 8192;
    endcase
```

这样，完成上述设计后，我们以一个通道为例的 ACM9767 驱动模块设计，就完成了。

将上面介绍的各个子模块和 IP 核进行例化，就可以得到工程的顶层设计。

1.4.4 工程顶层模块设计

在工程顶层中，我们需要对前面设计的各个子模块例化，并添加 VIO 核，设计控制逻辑，根据 VIO 的输出信号，完成控制字的累加计数切换和对应控制值的选择。

关于子模块的例化，这里不多赘述，主要讲解如何配置并添加 VIO 核以及如何设计控制逻辑。

首先在 IP Catalog 中搜索并找到 VIO(Virtual Input/Output)，双击打开配置。根据前面的分析，我们需要 6 个输出控制信号，用来分别控制 A、B 两个通道中输出波形的频率、相位、波形类型。因此，VIO 的输出端口需要设置为 6，每个端口的位宽为 1，如图 1-15 和图 1-16 所示：

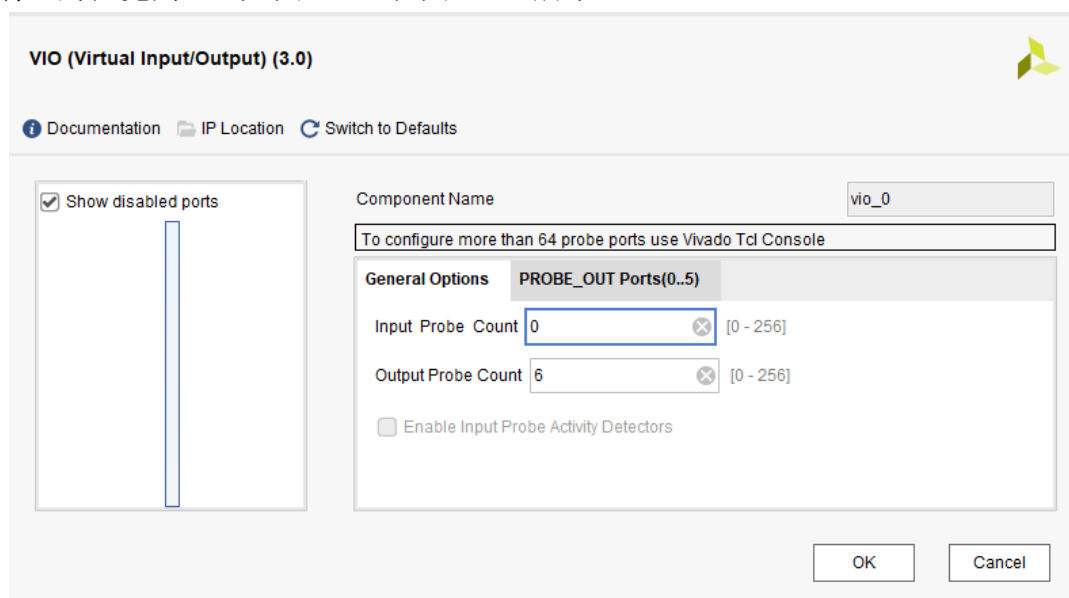


图 1-15 配置 VIO 端口

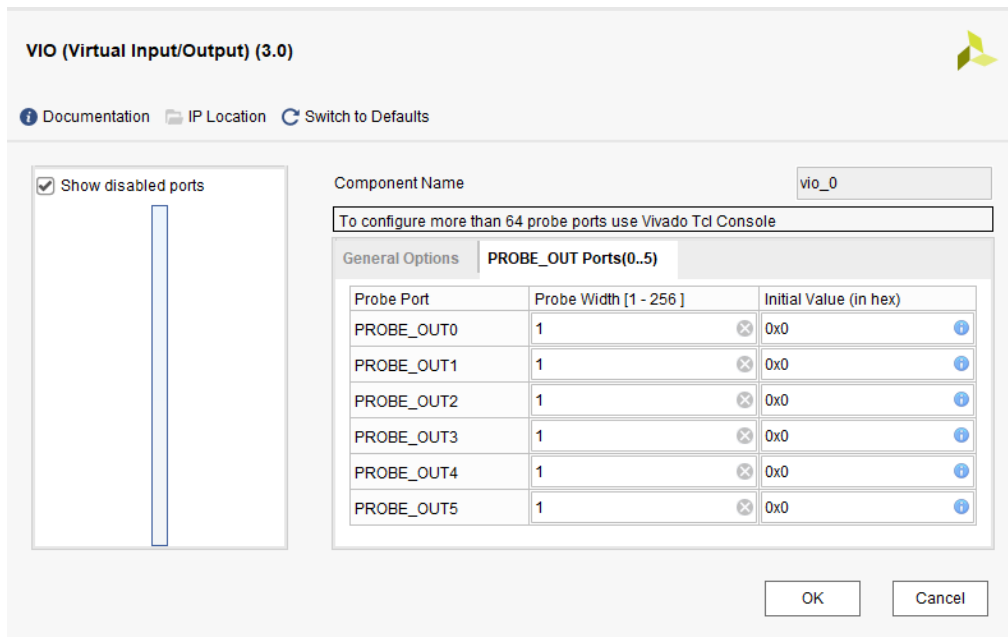


图 1-16 配置 VIO 端口位宽

配置完成后在顶层中例化 VIO 核，为了确保用户每次通过 VIO 核输出的数据只会被作用一次，我们需要对信号寄存两拍用以判断上升沿。该上升沿判断技巧原理在前面章节中有过讲解，这里不再赘述，相关代码如下：

```

wire CHA_Fword_flag;
wire CHB_Fword_flag;
wire CHA_Pword_flag;
wire CHB_Pword_flag;
wire Mode_SelA_flag;
wire Mode_SelB_flag;

vio_0 vio (
    .clk(CLK125M),                //input wire clk
    .probe_out0(CH_A_Fword_flag), //output wire [0:0] probe_out0
    .probe_out1(CHB_Fword_flag), //output wire [0:0] probe_out1
    .probe_out2(CH_A_Pword_flag), //output wire [0:0] probe_out2
    .probe_out3(CHB_Pword_flag), //output wire [0:0] probe_out3
    .probe_out4(Mode_SelA_flag), //output wire [0:0] probe_out4
    .probe_out5(Mode_SelB_flag) //output wire [0:0] probe_out5
);

//对 flag 信号打两拍，用于上升沿检测
reg CHA_Fword_flag_reg0, CHA_Fword_flag_reg1;
reg CHB_Fword_flag_reg0, CHB_Fword_flag_reg1;
reg CHA_Pword_flag_reg0, CHA_Pword_flag_reg1;
reg CHB_Pword_flag_reg0, CHB_Pword_flag_reg1;
reg Mode_SelA_flag_reg0, Mode_SelA_flag_reg1;

```

```
reg Mode_SelB_flag_reg0,Mode_SelB_flag_reg1;

always@(posedge CLK125M or negedge Reset_n)
if(!Reset_n)begin
    CHA_Fword_flag_reg0 <= 0;
    CHA_Fword_flag_reg1 <= 0;
end
else begin
    CHA_Fword_flag_reg0 <= CHA_Fword_flag;
    CHA_Fword_flag_reg1 <= CHA_Fword_flag_reg0;
end

always@(posedge CLK125M or negedge Reset_n)
if(!Reset_n)begin
    CHB_Fword_flag_reg0 <= 0;
    CHB_Fword_flag_reg1 <= 0;
end
else begin
    CHB_Fword_flag_reg0 <= CHB_Fword_flag;
    CHB_Fword_flag_reg1 <= CHB_Fword_flag_reg0;
end

always@(posedge CLK125M or negedge Reset_n)
if(!Reset_n)begin
    CHA_Pword_flag_reg0 <= 0;
    CHA_Pword_flag_reg1 <= 0;
end
else begin
    CHA_Pword_flag_reg0 <= CHA_Pword_flag;
    CHA_Pword_flag_reg1 <= CHA_Pword_flag_reg0;
end

always@(posedge CLK125M or negedge Reset_n)
if(!Reset_n)begin
    CHB_Pword_flag_reg0 <= 0;
    CHB_Pword_flag_reg1 <= 0;
end
else begin
    CHB_Pword_flag_reg0 <= CHB_Pword_flag;
    CHB_Pword_flag_reg1 <= CHB_Pword_flag_reg0;
end

always@(posedge CLK125M or negedge Reset_n)
if(!Reset_n)begin
```

```

        Mode_SelA_flag_reg0 <= 0;
        Mode_SelA_flag_reg1 <= 0;
    end
    else begin
        Mode_SelA_flag_reg0 <= Mode_SelA_flag;
        Mode_SelA_flag_reg1 <= Mode_SelA_flag_reg0;
    end

    always@(posedge CLK125M or negedge Reset_n)
    if(!Reset_n)begin
        Mode_SelB_flag_reg0 <= 0;
        Mode_SelB_flag_reg1 <= 0;
    end
    else begin
        Mode_SelB_flag_reg0 <= Mode_SelB_flag;
        Mode_SelB_flag_reg1 <= Mode_SelB_flag_reg0;
    end

    wire CHA_Fword_posedge;
    wire CHB_Fword_posedge;
    wire CHA_Pword_posedge;
    wire CHB_Pword_posedge;
    wire Mode_SelA_posedge;
    wire Mode_SelB_posedge;
    //上升沿检测
    assign CHA_Fword_posedge = (!CHA_Fword_flag_reg1) &
CHA_Fword_flag_reg0;
    assign CHB_Fword_posedge = (!CHB_Fword_flag_reg1) &
CHB_Fword_flag_reg0;
    assign CHA_Pword_posedge = (!CHA_Pword_flag_reg1) &
CHA_Pword_flag_reg0;
    assign CHB_Pword_posedge = (!CHB_Pword_flag_reg1) &
CHB_Pword_flag_reg0;
    assign Mode_SelA_posedge = (!Mode_SelA_flag_reg1) &
Mode_SelA_flag_reg0;
    assign Mode_SelB_posedge = (!Mode_SelB_flag_reg1) &
Mode_SelB_flag_reg0;

```

在得到对应控制信号的上升沿标志信号后，我们就可以据此控制对应通道频率、相位、波形类型控制字递增变化。代码如下：

```

    always@(posedge CLK125M or negedge Reset_n)
    if(!Reset_n)
        CHA_Fword_Sel <= 0; //板级验证时启用
    else if(CHB_Fword_posedge)
        CHA_Fword_Sel <= CHA_Fword_Sel + 1'd1;

```



```

else
    CHA_Fword_Sel <= CHA_Fword_Sel;

always@(posedge CLK125M or negedge Reset_n)
if(!Reset_n)
    CHB_Fword_Sel <= 0; //板级验证时启用
else if(CHB_Fword_posedge)
    CHB_Fword_Sel <= CHB_Fword_Sel + 1'd1;
else
    CHB_Fword_Sel <= CHB_Fword_Sel;

always@(posedge CLK125M or negedge Reset_n)
if(!Reset_n)
    CHA_Pword_Sel <= 0; //板级验证时使用
else if(CHB_Pword_posedge)
    CHA_Pword_Sel <= CHA_Pword_Sel + 1'd1;
else
    CHA_Pword_Sel <= CHA_Pword_Sel;

always@(posedge CLK125M or negedge Reset_n)
if(!Reset_n)
    CHB_Pword_Sel <= 0;
else if(CHB_Pword_posedge)
    CHB_Pword_Sel <= CHB_Pword_Sel + 1'd1;
else
    CHB_Pword_Sel <= CHB_Pword_Sel;

always@(posedge CLK125M or negedge Reset_n)
if(!Reset_n)
    Mode_SelA <= 0;
else if(Mode_SelA_posedge)
    Mode_SelA <= Mode_SelA + 1'd1;
else
    Mode_SelA <= Mode_SelA;

always@(posedge CLK125M or negedge Reset_n)
if(!Reset_n)
    Mode_SelB <= 0;
else if(Mode_SelB_posedge)
    Mode_SelB <= Mode_SelB + 1'd1;
else
    Mode_SelB <= Mode_SelB;

```

我们在前面讲解设计原理分析的内容时，已经讲解了频率控制字的理论推导计算方法。

/*如果把周期完整的一个波形等分成 2 的 32 次方份，那么如果希望 1 秒钟输出一个完成的周期，每一拍递进多少份？*/

```
always@(*)
case(CHB_Fword_Sel)
0:FwordA = 34; //2**32 / 125000000;    34.35
1:FwordA = 344; //2**32 / 125000000;
2:FwordA = 3436; //2**32 / 12500000;
3:FwordA = 34360; //2**32 / 1250000;
4:FwordA = 343597; //2**32 / 125000;
5:FwordA = 3435974; //2**32 / 12500;
6:FwordA = 34359738; //2**32 / 1250;
7:FwordA = 343597384; //2**32 / 125;
endcase

always@(*)
case(CHB_Fword_Sel)
0:FwordB = 34; //2**32 / 125000000;    34.35
1:FwordB = 344; //2**32 / 125000000;
2:FwordB = 3436; //2**32 / 12500000;
3:FwordB = 34360; //2**32 / 1250000;
4:FwordB = 343597; //2**32 / 125000;
5:FwordB = 3435974; //2**32 / 12500;
6:FwordB = 34359738; //2**32 / 1250;
7:FwordB = 343597384; //2**32 / 125;
endcase
```

为了尽可能将波形细分，我们将单周期波形细分的份数，设定为 2 的 32 次方即 4,294,967,296 份。而时钟频率是 125MHz，则说明每一个时钟周期上升沿到来，得递进累加（4,294,967,296/125000000）份数，即算出得到一个基准的频率控制字。以此，如果频率控制字扩大 10 倍，则相同时间内出现的波形数量，也扩大 10 倍，以此，从代码的角度，就形成了以上频率控制字的查找表。

```
always@(*)
case(CHB_Pword_Sel)
0:PwordA = 0;    //0
1:PwordA = 341;  //30
2:PwordA = 683;  //60
3:PwordA = 1024; //90
4:PwordA = 1707; //150
5:PwordA = 2048; //180
6:PwordA = 3072; //270
7:PwordA = 3641; //320
endcase

always@(*)
```

```
case(CHB_Pword_Sel)
    0:PwordB = 0;    //0
    1:PwordB = 341;  //30
    2:PwordB = 683;  //60
    3:PwordB = 1024;  //90
    4:PwordB = 1707;  //150
    5:PwordB = 2048;  //180
    6:PwordB = 3072;  //270
    7:PwordB = 3641;  //320
endcase
```

相对而言，相位控制字的设计更加简单。我们单周期波形的存储深度为 4096，将 4096 等分后与 360 度角度等分对应，即可得到每个等分角度的变化值。

到这里我们也就完成了设计的顶层例化和对控制字值变化的逻辑设计，设计中 VIO 输出的控制信号对应的功能如下表：

表 1-2 VIO 输出信号功能表

按键/vio 虚拟按键	功能描述
key0 (CHA_Fword_flag)	调节 ACM9767 通道 1 的输出频率，可依次在 1Hz，10Hz，100Hz，1kHz，10kHz，100kHz，1MHz，10MHz 频率中切换
key1 (CHB_Fword_flag)	调节 ACM9767 通道 2 的输出频率，可依次在 1Hz，10Hz，100Hz，1kHz，10kHz，100kHz，1MHz，10MHz 频率中切换
key2 (CHA_Pword_flag)	调节通道 1 输出信号相位，可依次在 0°，30°，60°，90°，150°，180°，270°，320° 中切换
key3 (CHB_Pword_flag)	调节通道 2 输出信号相位，可依次在 0°，30°，60°，90°，150°，180°，270°，320° 中切换
key4 (Mode_SelA_flag)	控制通道 1 输出的波形，依次在正弦波、方波、三角波、直线中切换
key5 (Mode_SelB_flag)	控制通道 2 输出的波形，依次在正弦波、方波、三角波、直线中切换

接下来我们就可以基于该表设计本次仿真测试文件了。

1.5 激励创建及仿真测试

1.5.1 激励信号设计

由于默认的频率控制字给出的控制频率是 1Hz，这就表明如果想看到一个周期的完整波形，就需要让 VIVADO 仿真绘制出 1 秒的完整波形。而对于 VIVADO 软件来说，绘制 1 秒钟的仿真波形，对于这个涉及到 ROM 数据读取的设计工程来说，是非常缓慢的。

针对这个问题，我们在仿真时可以改变频率控制字的默认初始值，达到加快仿真的目的。经过计算，我们认为仿真以 100us 为一个周期是既便于观察，又能让 VIVADO 仿真迅速出图的一个合理值。

基于此，我们对顶层中通道 A、B 频率控制字的初始值进行修改，通过使用条件编译的方式，控制在板级验证和仿真验证时的初始值。代码如下：

```
`define sim

always@(posedge CLK125M or negedge Reset_n)
if(!Reset_n)
`ifdef sim
    CHA_Fword_Sel <= 4; //仿真时启用
`else
    CHA_Fword_Sel <= 0; //板级验证时启用
`endif
else if(CHA_Fword_posedge)
    CHA_Fword_Sel <= CHA_Fword_Sel + 1'd1;
else
    CHA_Fword_Sel <= CHA_Fword_Sel;

always@(posedge CLK125M or negedge Reset_n)
if(!Reset_n)
`ifdef sim
    CHB_Fword_Sel <= 4; //仿真时启用
`else
    CHB_Fword_Sel <= 0; //板级验证时启用
`endif
else if(CHB_Fword_posedge)
    CHB_Fword_Sel <= CHB_Fword_Sel + 1'd1;
else
    CHB_Fword_Sel <= CHB_Fword_Sel;
```

根据不同的应用场合，我们只需在顶层中选择是否屏蔽 sim 定义语句，我就能控制两个通道的频率。

为了确定设计确实能够控制输出波形类型，这里我们还可以通过条件编译的方式对通道 B 输出波形类型控制字的初值修改。这样我们就能看到 A、B 两个通道输出不同类型的波形数据。代码如下：

```
always@(posedge CLK125M or negedge Reset_n)
if(!Reset_n)
`ifdef sim
    Mode_SelB <= 1;
`else
    Mode_SelB <= 0;
`endif
else if(Mode_SelB_posedge)
    Mode_SelB <= Mode_SelB + 1'd1;
else
```

```
Mode_SelB <= Mode_SelB;
```

经过对代码的化繁为简，经过对设计的仿真优化，我们只需在仿真文件中进行时钟和复位信号的描述即可。仿真 tb 文件可以作如下设计：

```
`timescale 1ns / 1ps

module DDS_AD9767_tb;

    reg Clk;
    reg Reset_n;
    wire [13:0]DataA;
    wire [13:0]DataB;
    wire ClkA;
    wire WRTA;
    wire ClkB;
    wire WRTB;

    DDS_AD9767 DDS_AD9767(
        .Clk(Clk),
        .Reset_n(Reset_n),
        .DataA(DataA),
        .ClkA(ClkA),
        .WRTA(WRTA),
        .DataB(DataB),
        .WRTB(WRTB),
        .ClkB(ClkB)
    );

    initial Clk = 1;
    always #10 Clk = ~Clk;

    initial begin
        Reset_n = 0;
        #201;
        Reset_n = 1;
        #2000000;
        $stop;
    end
endmodule
```

1.5.2 仿真结果分析

根据以上仿真代码，可以得到如下仿真波形，这里我们对波形进行了分组，同时我们对关键信号进行监视，以有利于观察波形数据输出结果。

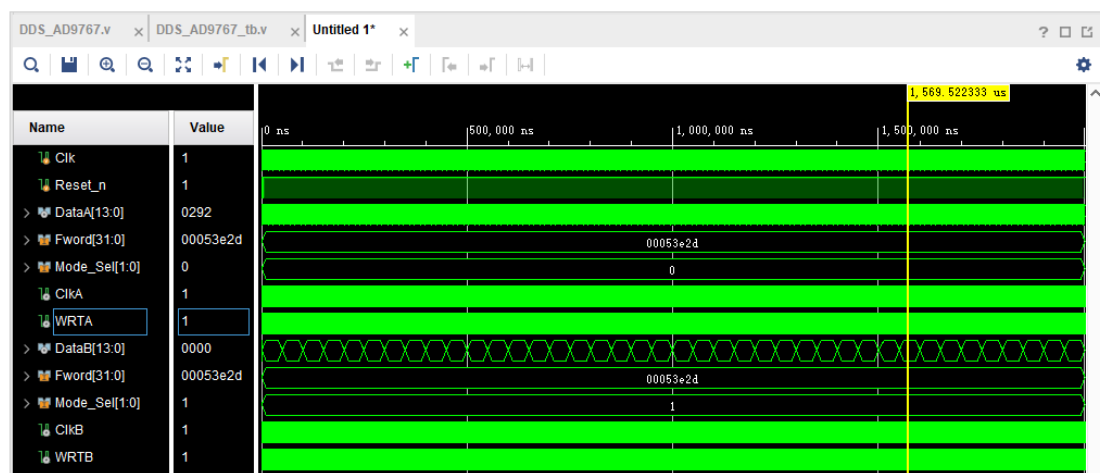


图 1-17 仿真结果波形总览

默认来说，软件给出的输出结果都是数字量，而对于本实验来说，则是让仿真软件输出模拟量波形会更加直观展现我们的设计意图。VIVADO 仿真工具具备将数字量转换成模拟量的能力。这里，我们只需要鼠标右键点击信号名称，然后切换波形输出类型即可。

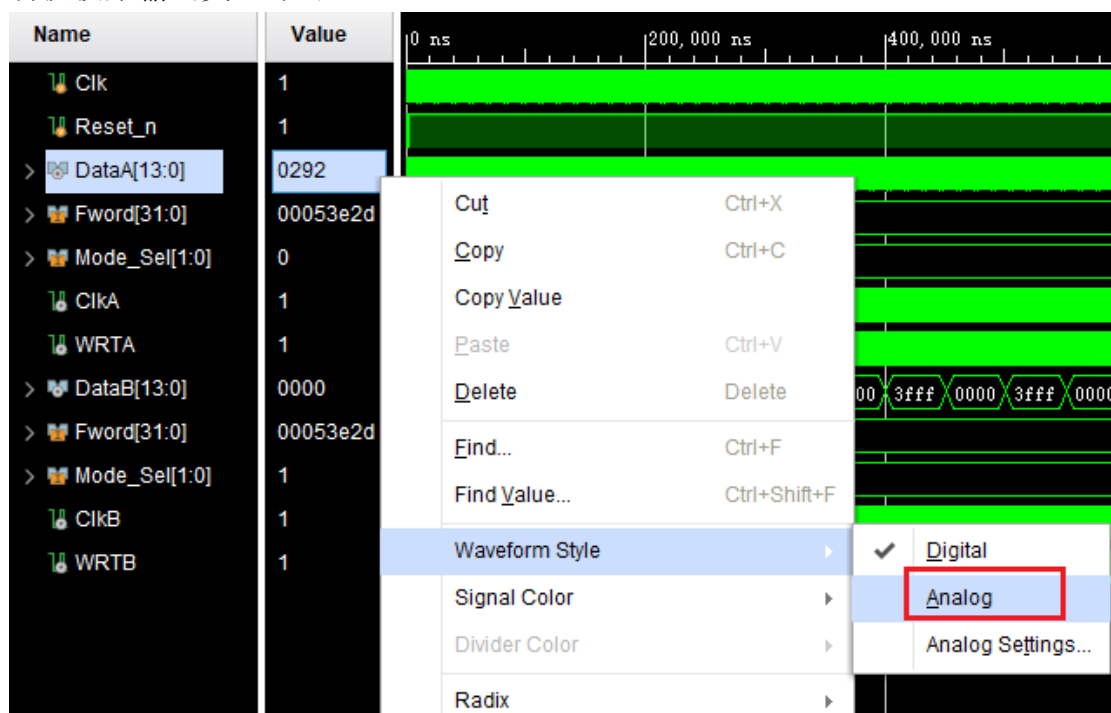


图 1-18 波形的数字量输出转换成模拟量输出

对通道 A，经过转换后，可以看到，输出了周期为 0.1ms 的正弦波，也正符合我们的设计预期。

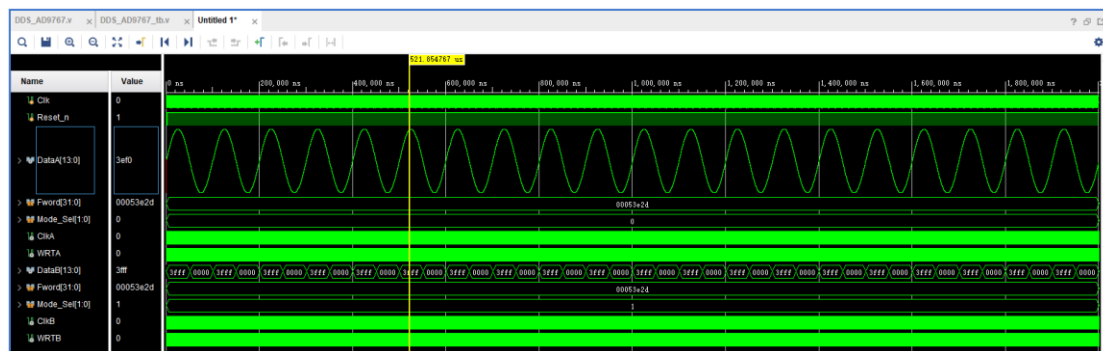


图 1-19 通道 A 的正弦波输出效果

对于通道 B，我们从图 1-19 输出数值直接观察的角度，就可以很容易的判断出是方波信号。因为通道 B 的输出数据，呈 3fff 和 0000 的交替变化。特别是有了上方通道 A 的正弦波波形作为参考，更是能够判断出通道 A 和通道 B 的输出波形类型虽然不同，但是周期相同。即通道 B 输出了周期为 0.1ms 的方波信号。

那么，实际经过模拟量波形变换，设置后，通道 B 能否如设想一致，输出方波信号呢？按图 1-18 的操作方法，我们对通道 B 进行再次设置却得到的是三角波波形如下：

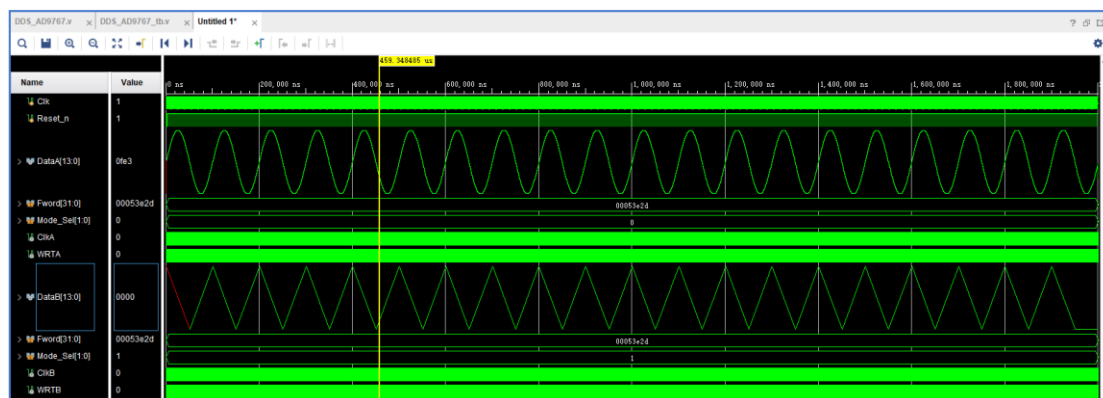


图 1-20 通道 B 进行模拟输出设置后的实际输出波形 1

经过前面数字量结果的肯定，同时经过分析和查阅资料，我们肯定了工程设计的正确性。而出现这种问题的根本原因在于软件将波形从数字量显示模式切换为模拟量显示模式后，会对离散点绘制的模拟量波形进行插值填充，以保证波形的美观和平滑过渡。但是在本工程的方波信号绘制过程中，软件的自动插值并不是我们期望看到的结果，它直接将我们已经确认的方波信号，绘制成为了三角波。

针对这个问题，我们可以通过对输出模拟量进行进一步设置。

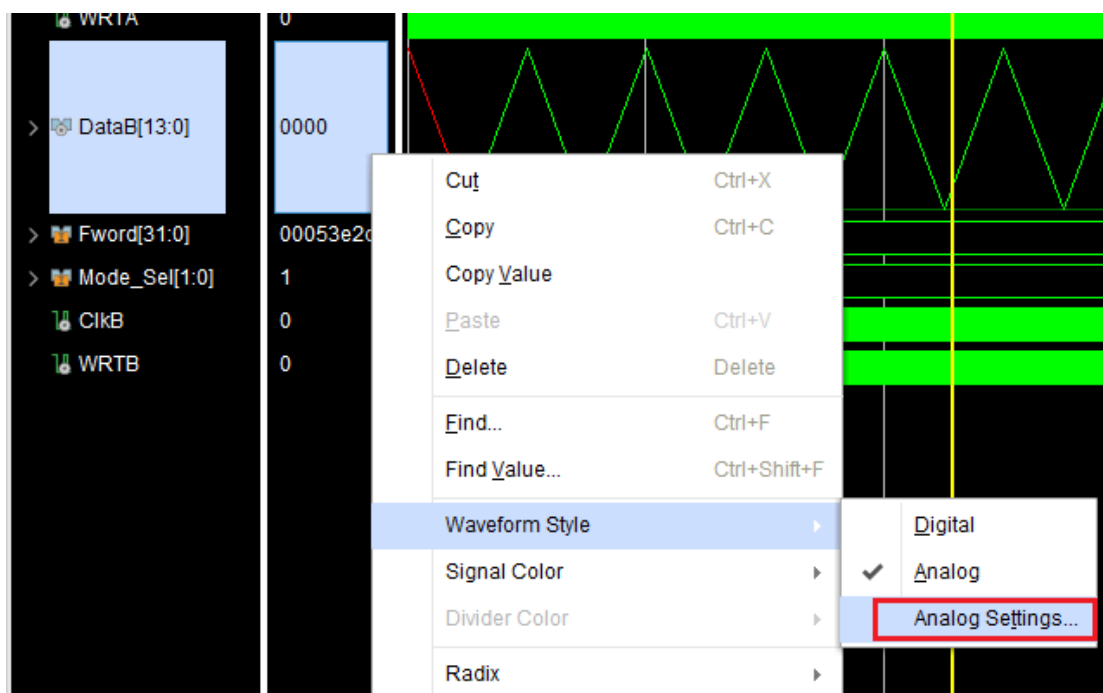


图 1-21 输出模拟量设置

点击 Analog Settings 下方的输出模拟量设置选项卡。

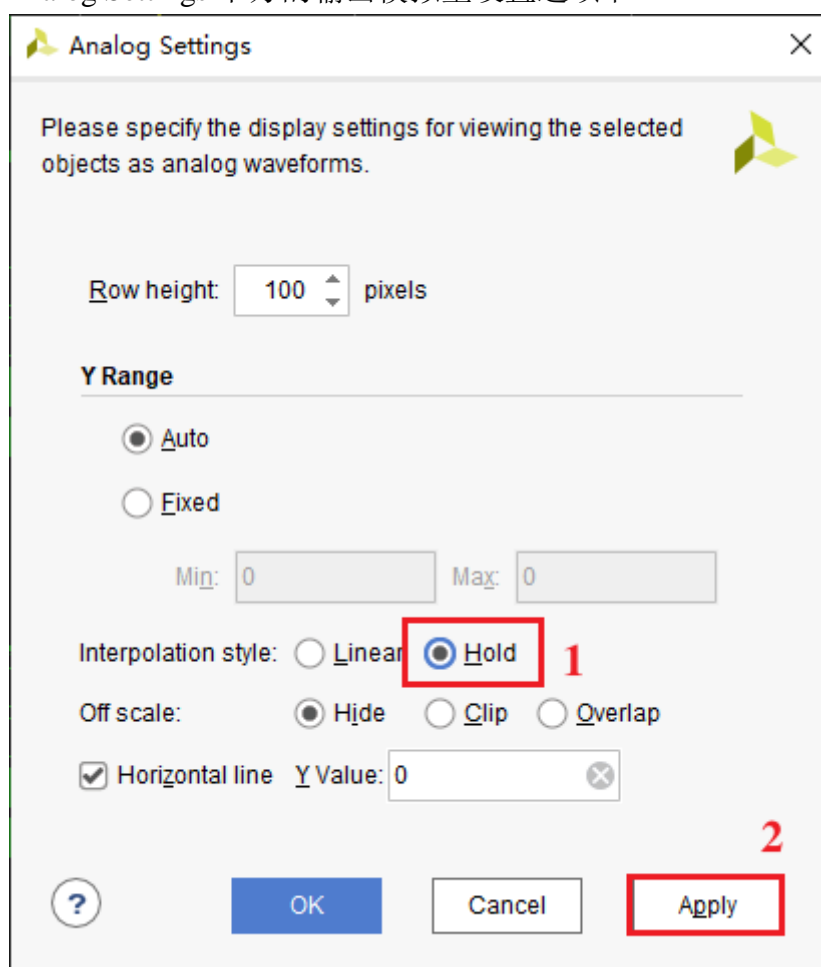


图 1-22 仿真设计去除插值的设置方法

这样，就可以在模拟量输出波形模式下，输出真实反映数字量变化情况的方波信号了。

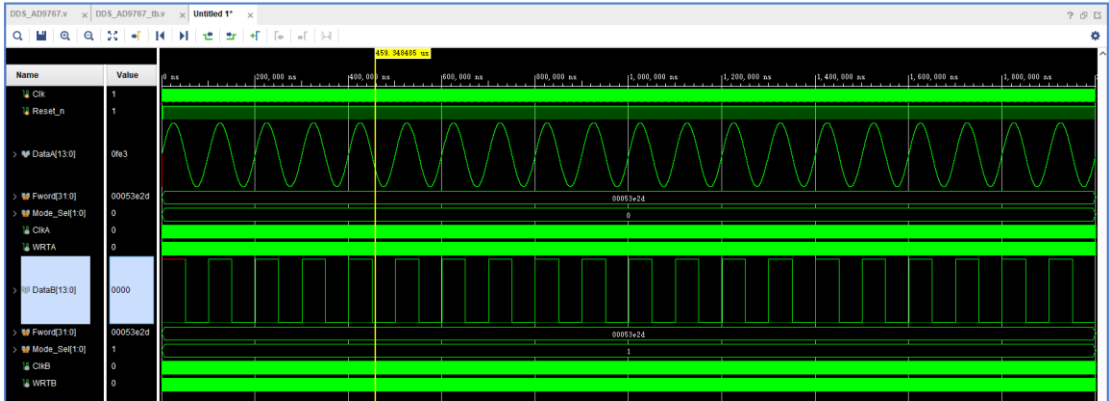


图 1-23 设置不进行插值后的输出波形

到这里，我们通过仿真验证了信号发生频率设置，信号发生波形的模式设置是符合设计需求的。

1.5.3 改变相位初始值的设定

接下来，我们改变通道 A 的正弦波信号发生相位，观察正弦波信号是否能够受控，发生信号相位偏移。

```
always@(posedge CLK125M or negedge Reset_n)
if(!Reset_n)
`ifdef sim
    CHA_Pword_Sel <= 3; //仿真时使用
`else
    CHA_Pword_Sel <= 0; //板级验证时使用
`endif
else if(CHA_Pword_posedge)
    CHA_Pword_Sel <= CHA_Pword_Sel + 1'd1;
else
    CHA_Pword_Sel <= CHA_Pword_Sel;
```

这里，我们将 CHA_Pword_Sel 的仿真默认值修改为 3，然后再次启动仿真。得到如下输出波形。

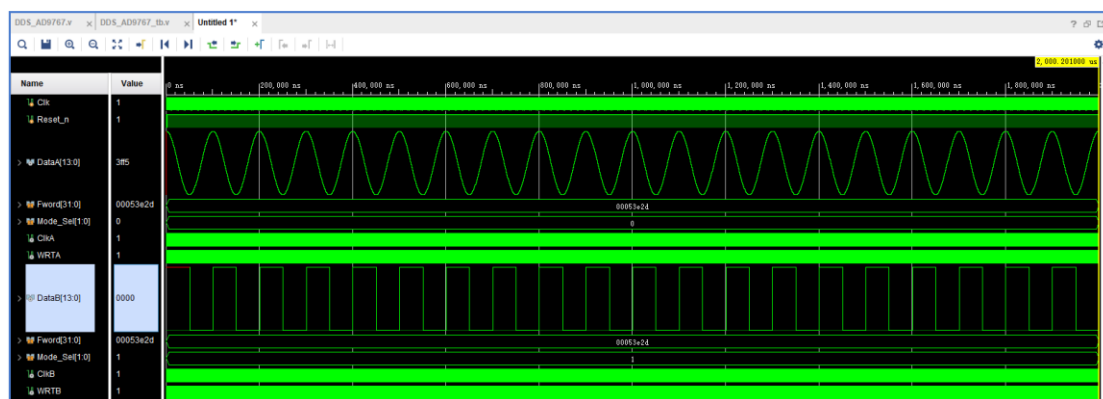


图 1-24 通道 A 经过相位调整后的输出波形

经过调整后，对比图 1-23 和图 1-24 发现通道 A 相位确实相对于通道 B 进行了 90 度调整。这样，也验证了相位调整功能的设计符合预期。

经过以上分析，本工程的各设计功能也全部验证完成。完成仿真分析后，我们需记得将专门为仿真而定制修改的代码还原。

1.6 板级验证

1.6.1 实验目标

本实验的板级验证环节，主要验证以下三个目标：

1. 能否正确的烧写和下载程序。
2. 下载并配置好输出波形后，能否在示波器上观察到期望的波形。
3. 通过 VIO 改变对应控制字的值，示波器显示的波形能否发生变化。

1.6.2 系统所需硬件

系统所需硬件如下：

1. ACZ702 开发板 x1
2. Type-c 下载线 x1
3. ACM9767 模块 x1
4. 示波器 x1
5. Dc 供电线 x1（可选）

1.6.3 硬件连接

本次系统硬件方面连接十分简单，只需给 ACZ702 开发板上连接好电源线并

将 ACM9767 模块连接到 ACZ702 的拓展接口上即可。ACM9767 模块的 CH1 与 CH2 通道则会按照用户的设置输出相应的波形，将通道与示波器相连即可看到产生的波形。当所有连接工作完成后将开发板的电源开关拨到对应供电侧，本次系统设计开发板连接方法如下图所示：

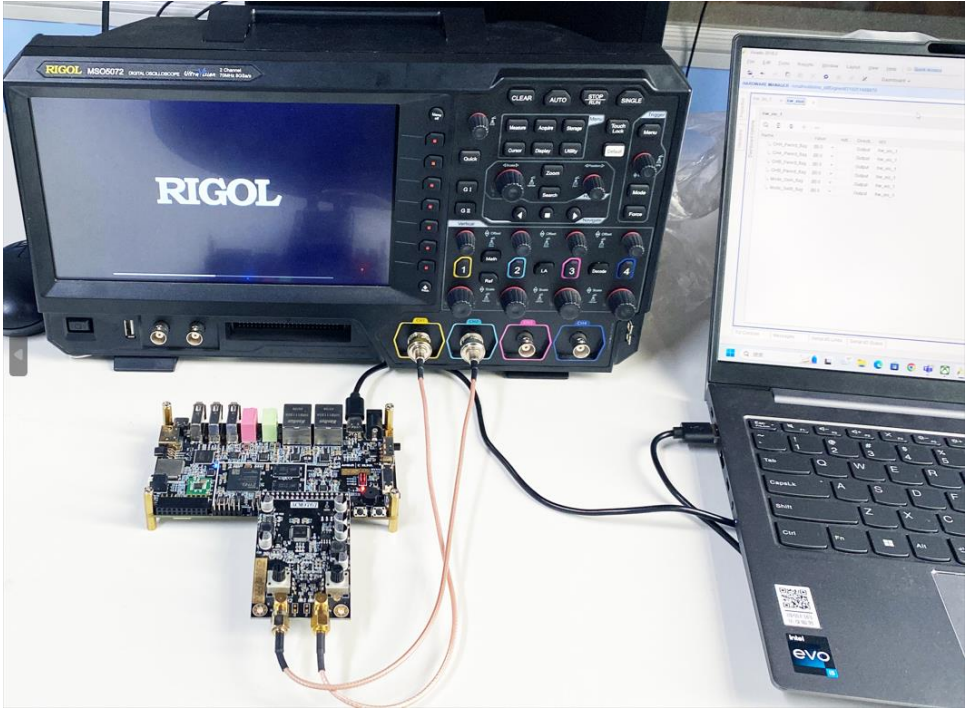


图 1-25 ACM9767 信号发生器实验硬件连接图

为了方便识别器件的 PCB 针脚，在 PCB 设计中，焊盘上 1 号管脚通常会被设计为方形。在进行硬件连接时，ACM9767 的 1 号引脚应该与 ACZ702 开发板 40pin 拓展接口的 1 号引脚相对应，即我们俗称的方口对方口。

1.6.4 管脚绑定

本次设计 ACM9767 模块需要连接到 ACZ70 开发板的 40pin 拓展接口上，对应的管脚绑定如表 1-3 所示：

表 1-3 引脚分配表

Pin Name	Signal Name	Pin NO.	Pin Name	Signal Name	Pin NO.
clk1	CLKA	G19	clk2	CLKB	G18
wrt1	WRTA	G20	wrt2	WRTB	H20
DA1_13	DataA[13]	K14	DA2_13	DataB[13]	H16
DA1_12	DataA[12]	L15	DA2_12	DataB[12]	J20
DA1_11	DataA[11]	G14	DA2_11	DataB[11]	J19
DA1_10	DataA[10]	J14	DA2_10	DataB[10]	H18
DA1_9	DataA[9]	F16	DA2_9	DataB[9]	K19
DA1_8	DataA[8]	H15	DA2_8	DataB[8]	J18
DA1_7	DataA[7]	D18	DA2_7	DataB[7]	H17
DA1_6	DataA[6]	E17	DA2_6	DataB[6]	K18

DA1_5	DataA[5]	D19		DA2_5	DataB[5]	J16
DA1_4	DataA[4]	E18		DA2_4	DataB[4]	K16
DA1_3	DataA[3]	F17		DA2_3	DataB[3]	L20
DA1_2	DataA[2]	D20		DA2_2	DataB[2]	L19
DA1_1	DataA[1]	G17		DA2_1	DataB[1]	M20
DA1_0	DataA[0]	E19		DA2_0	DataB[0]	M19
FPGA_GCLK1	Clk	U18		FPGA_KEY0	Reset_n	F20

分配完管脚后还需约束管脚对应脚电平为 LVCMOS 33，随后保存设计并生成比特流，准备将设计下载到开发板中进行验证。

1.6.5 下载与验证

连接好硬件后将开发板上的拨码开关拨到 ON 侧，将我们提供好的工程源码下载至开发板中即可从示波器中看到相应的波形。

下载完成后在示波器上显示的默认波形如下图所示：

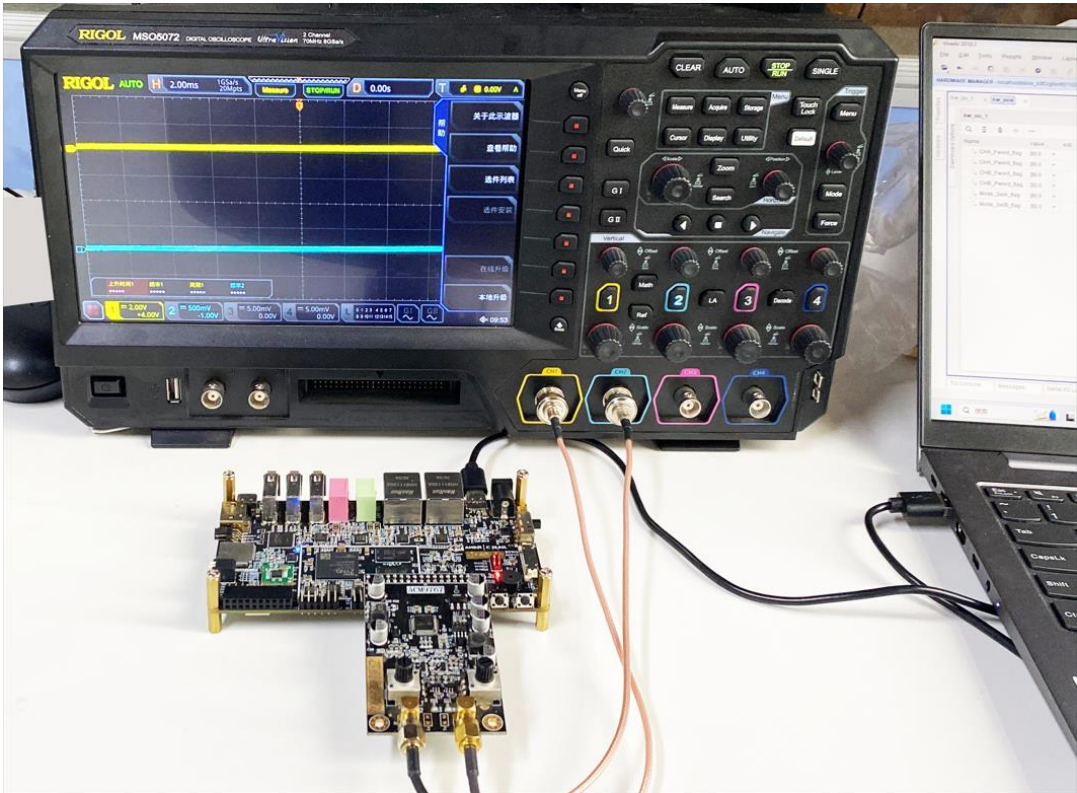


图 1-26 输出信号波形

由于我们默认的基准波形为 1Hz 的正弦波，而示波器对于如此低频的信号很难识别观察，所以屏幕暂时显示一条横线也是正常现象。

接下来我们通过 VIO 输出不同控制信号来调整波形的类型、频率和相位，分别如下图：

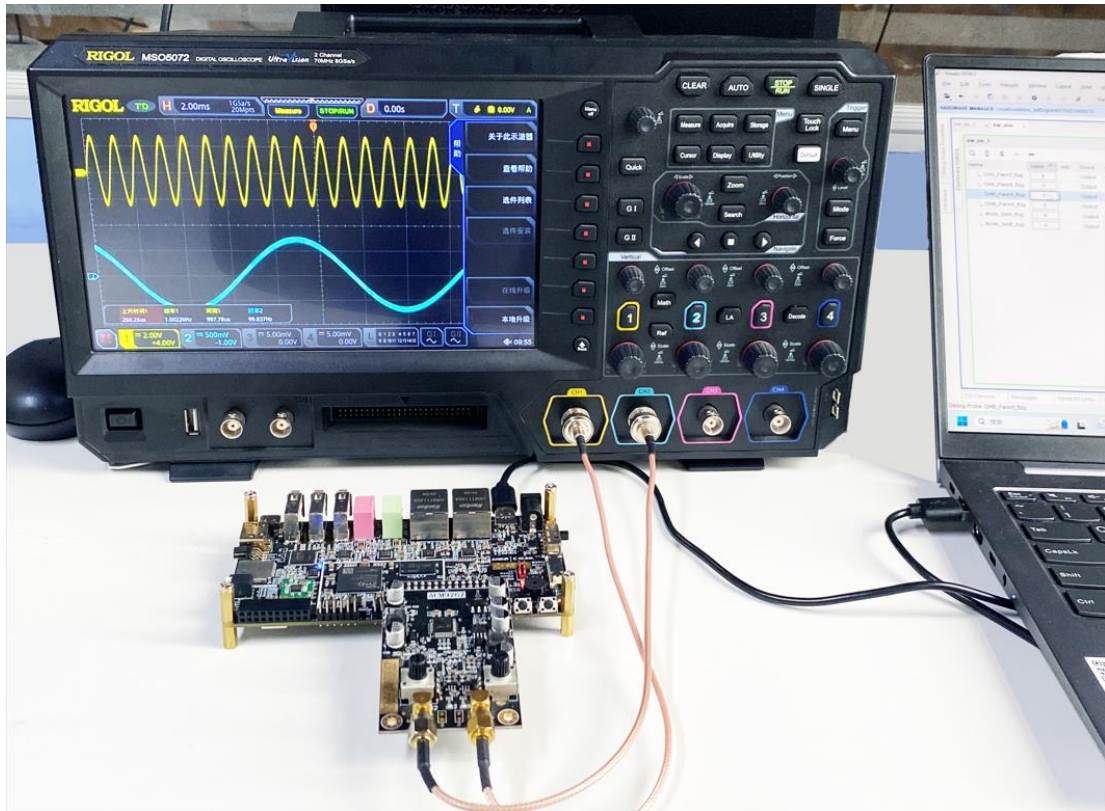


图 1-27 通过 Fword 调节波形频率

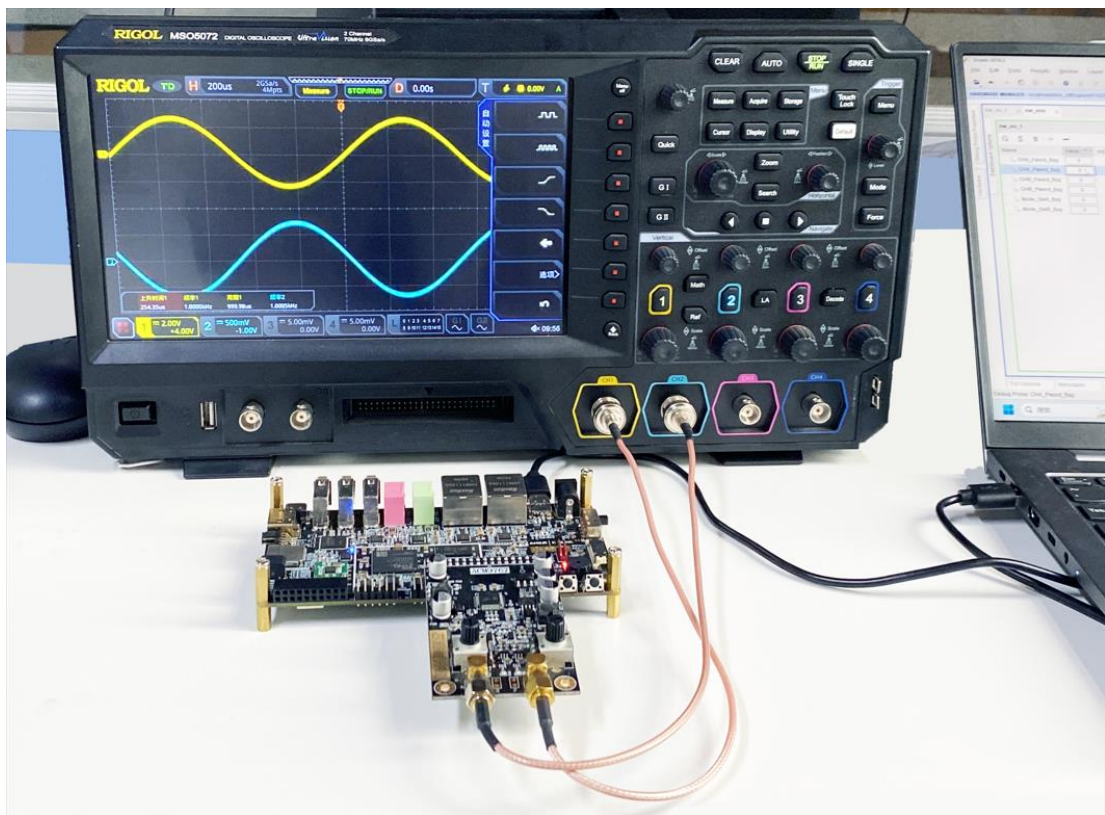


图 1-28 通过 Pword 调节波形相对相位

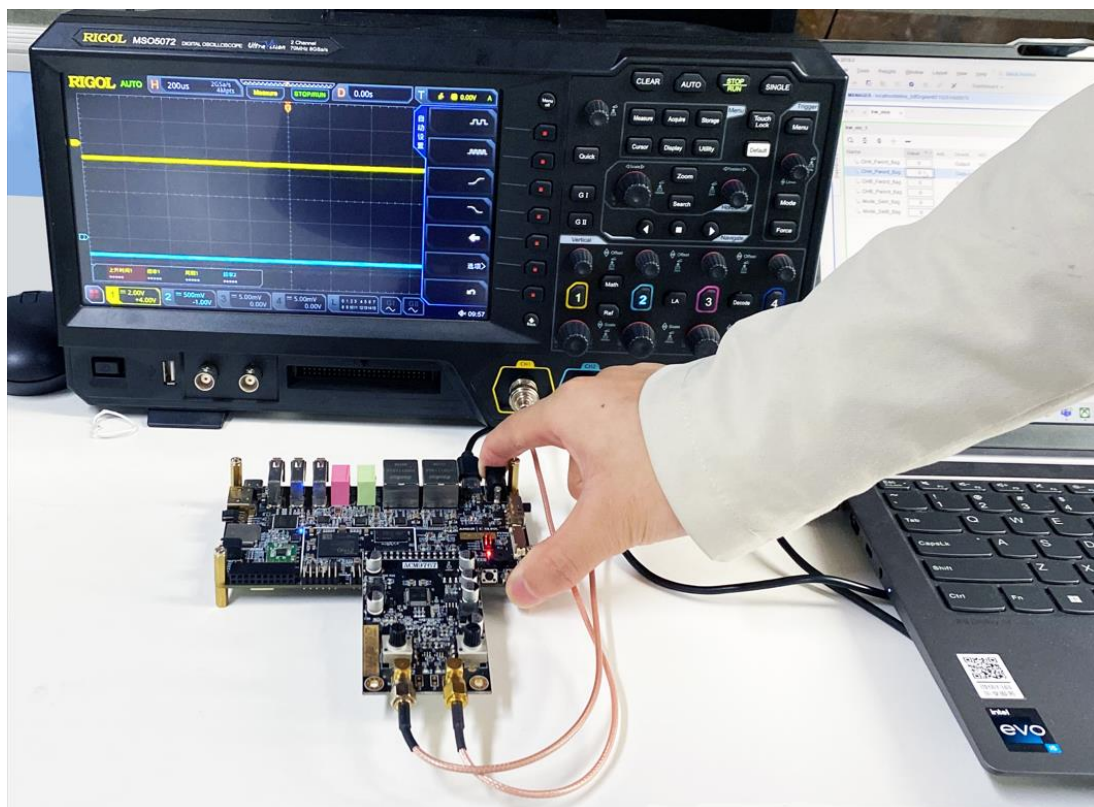


图 1-29 按键 S2 复位

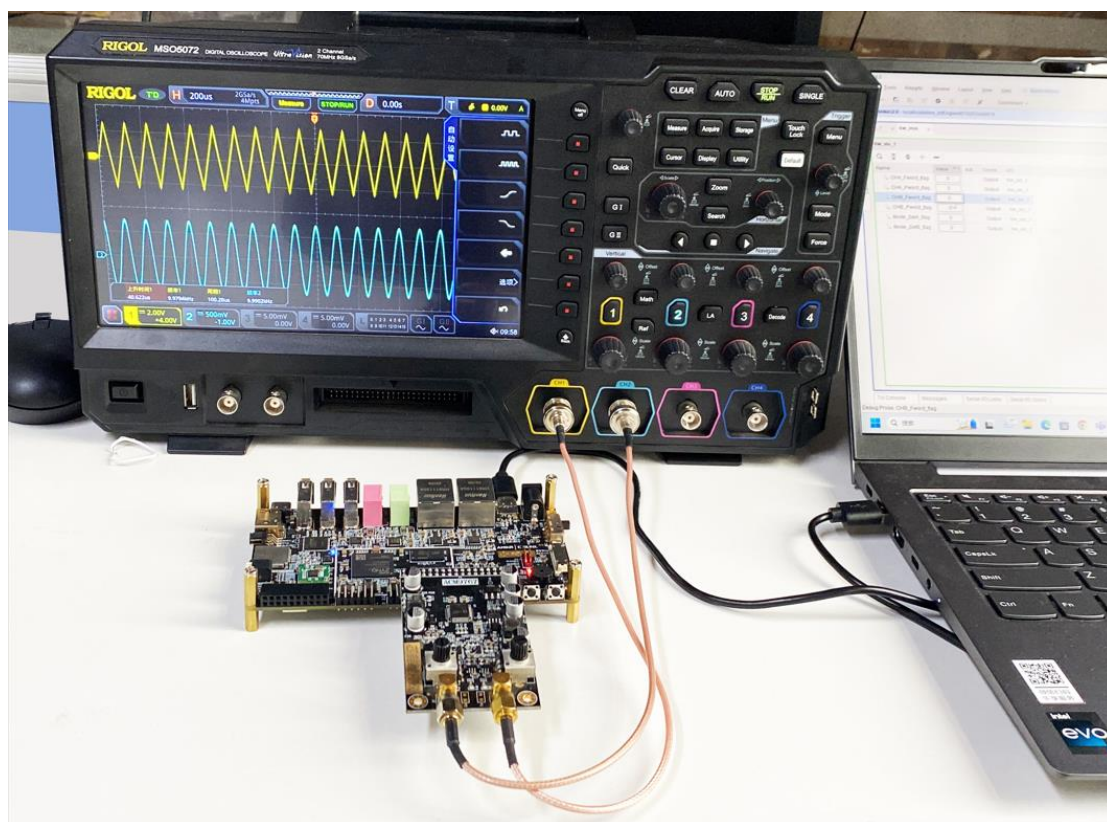


图 1-30 通过 Mode_Sel 切换输出波形

可以看到整个系统设计输出的波形是可调节的，通过 vio 核控制，能顺利

实现对三种波形的切换以及频率，相位的调整。

至此，设计板级验证成功，通过 VIO 核，我们可以根据自己需求对输出波形调节控制。

1.7 常见问题与思考总结

1. 由于频率控制和相位控制是通过按键进行的切换，而按键按下的信号需要进行消抖后才能被其他模块使用。实际使用时，只有当通道 A 和通道 B 的相位控制字切换完成并稳定后，再进行发生信号的输出。通过对信号发生模块的复位（即 DDS_Module 的复位信号）进行控制，可以较好的实现该需求。

2. 输出波形的类型，是通过 Mode_Sel 信号实现的。为了实现输出波形的类型可选择，必须为每个通道开辟 3 个 rom 空间，用于存放代表 3 种不同类型的波形数值。

3. 使用 VIVADO 仿真时，VIVADO 会将离散绘图数值进行插值处理以实现波形的平滑过渡，而通过对 VIVADO 的模拟量绘图设定，可以调整波形输出时插值处理与否。

4. 本实验以 ACZ702 开发板搭载 ACM9767 模块作为开发模型，通过 VIO 核控制波形的类型、频率和相位的档位切换。而真正在实现以此实验为基础的信号发生器开发时，可以考虑进行无极调节的硬件设计。借助带滑动变阻器的旋钮，频率和相位的控制将有更加丰富的选择，也更接近于一个真实的信号发生器。