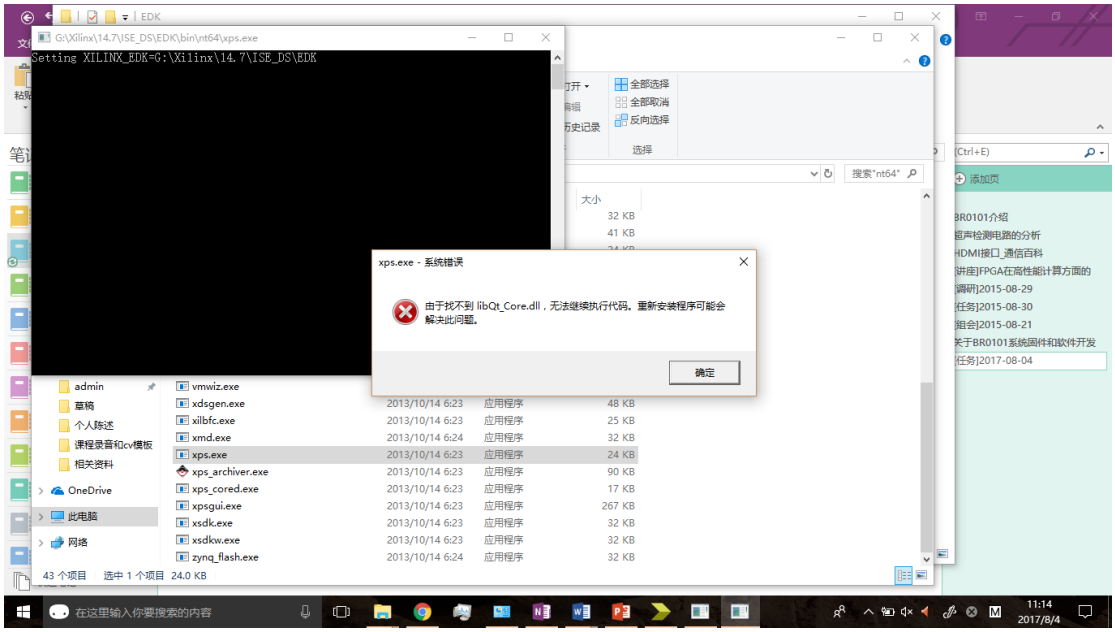
BR0101 固件开发——模数转换器（DAC）

我们尝试进行 BR0101 的固件开发，首先从信号通道区的模数转换器（DAC）AD9175 开始。

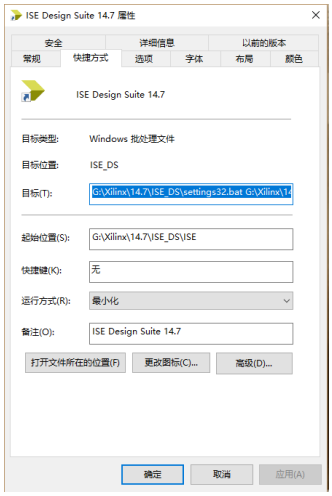
1 准备开发环境

BR0101 固件开发是基于 MicroBlaze 软核的，需要用到 Xilinx EDK 开发套件。我之前安装过 Xilinx Design Suite 14.7，但是运行 XPS 工具时会出现缺少.dll 的错误，如下图所示。



因此拆卸并重新安装 Xilinx Design Suite 14.7。

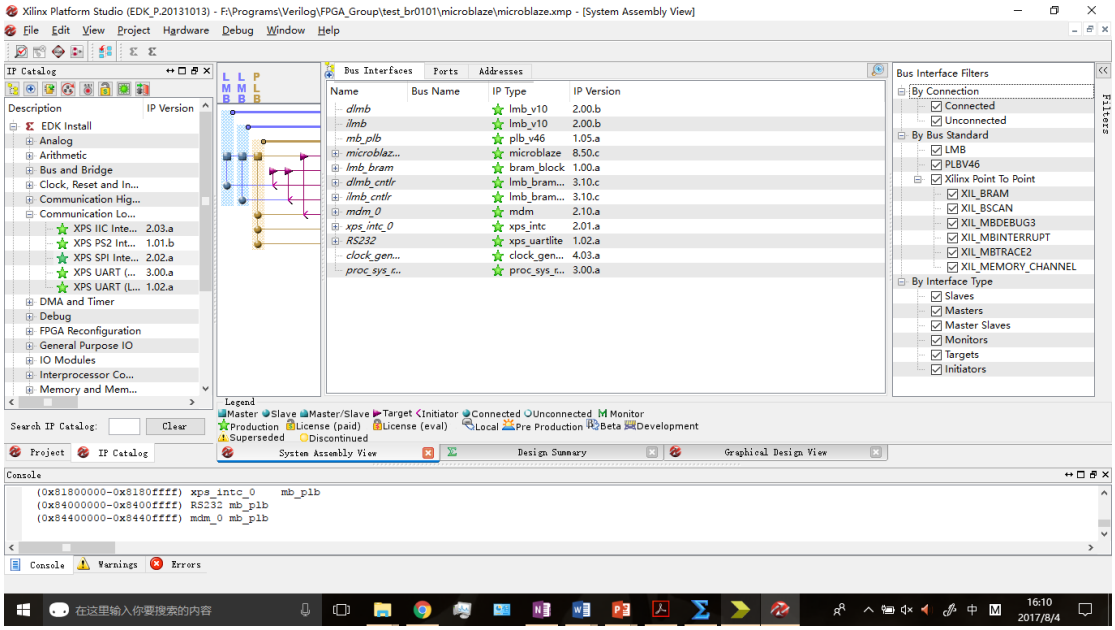
后来发现好像不是安装的问题，而是因为 64 位的版本有问题。



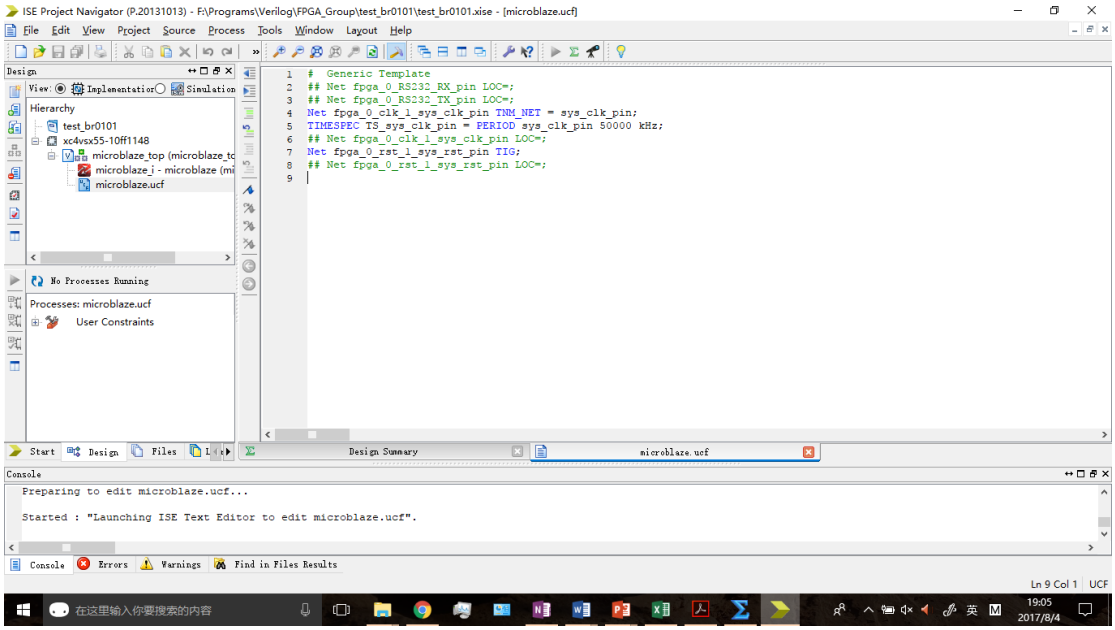
将 ISE Design Suite 14.7 的启动方式改为 32 位模式启动，命令如下：

G:\Xilinx\14.7\ISE_DS\settings32.bat G:\Xilinx\14.7\ISE_DS\ISE\bin\nt\ise.exe

Xilinx XPS 工具可以正常运行，完成 MicroBlaze 的配置：



但是 XPS 生成的.ucf 文件（microblaze/data/microblaze.ucf）文件似乎有些问题。



2 模数转换器 AD9715 的调研

2.1 功能调研

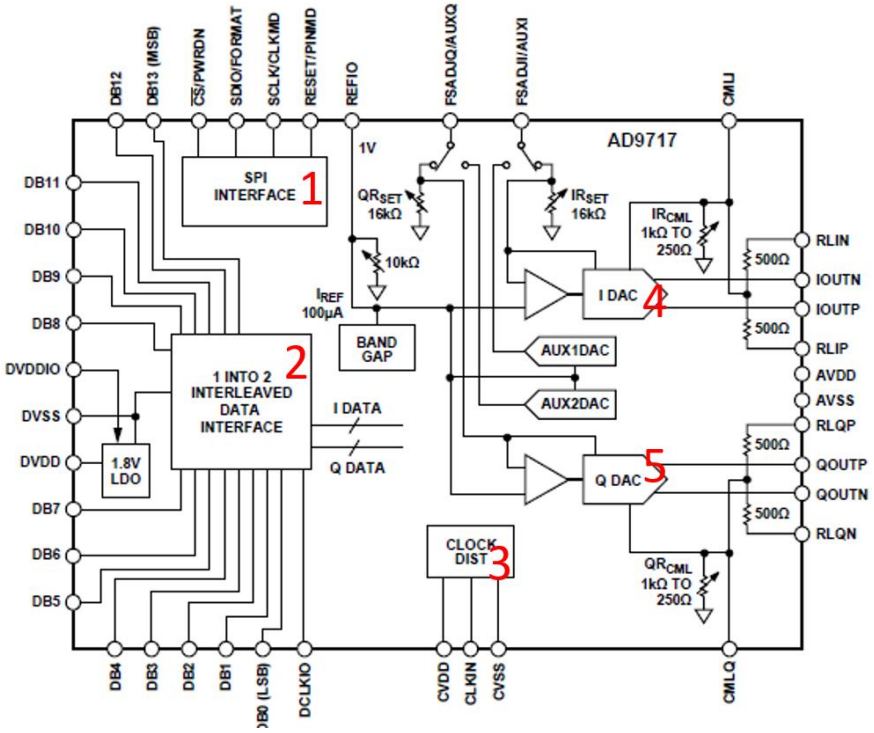
AD9715 是 ADI 公司推出的一款低功耗 10 位模数转换器，它的特点是：

- 低功耗：单电源供电，电压可以在 1.8 到 3.3V；供电电压 1.8V，采样率 125MSPS 时功耗 35mW；睡眠状态供电电压 3.3V 时功耗小于 3mW。
- CMOS 时钟输入：可以接受高速、单端时钟输入，支持 125MSPS 的采样率。

- 与其他组件容易接合：输出从 0V 到 1.2V 可调。

2.2 功能模块

AD9715 内部功能单元结构如下图所示：



其中，重要的功能模块有：

1. SPI 接口：可以方便地与微控制器、微处理器进行通信
2. 交叉存取数据接口：接收并处理数字信号，似乎与 DDR2 有关，有待进一步研究
3. 时钟分配器：接收片外输入的时钟信号

2.3 工作原理

2.3.1 模拟电流的产生

模拟输出电流 I_{xOUTFS} 由电流阵列叠加产生，数字信号和模拟电流的对应关系如下表：

数字信号范围	对应管脚	单位模拟电流数量	单位模拟电流大小
前 5 个 MSB	DB9 – DB5	31	$I_{MSB-Current}$
中间 4 个 bit	DB4 – DB1	15	$\frac{1}{16} I_{MSB-Current}$
最后的 LSB	DB0	1	$\frac{1}{32} I_{MSB-Current}$

2.3.2 差分电流输出

由 PMOS 差分电流开关控制。

2.3.3 电源配置

模拟电源输入（AVDD）和数字电源输入（DVDDIO）分开。 数字部分需要 1.8V 的电源供电，可以通过 DVDDIO 输入高于 1.8V 的电压再在片上进行转换，也 可以通过 DVDD 直接输入 1.8V 的供电电压。

2.3.4 参考电流

参考电流 I_{xREF} 由参考控制放大器和参考电压 V_{REFIO} 共同决定，参考控制放大器受 FSADJx 引脚上的 xR_{SET} 电阻控制。满标输出电流与参考电流的关系为：

$$I_{xOUTFS} = 32 \times I_{xREF}$$

2.4 接口

AD9715 利用 SPI 接口进行通信，具有以下特点：

- 可以访问所有配置寄存器
- 一次可以传送单个或多个字节
- 可以采用 MSB 优先或 LSB 优先的模式

2.4.1 通用操作

一次通信循环包括两个阶段：信息传输阶段和数据传输阶段。

- **复位**

在 35 号管脚（RESET/PINMD）上，先置逻辑 1 再置逻辑 1 可以使通信循环复位。

- **信息传输阶段**

在 SCLK 信号的前 8 个上升沿，AD9715 先读取一个字节的指令，这一个字节的指令决定了 AD9715 在数据传输阶段的行为，是读还是写，传送多少字节的数据，以及开始的寄存器地址。

指令字节的具体格式如下所示：

MSB							LSB
DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
R/ $\overline{W}$	N1	N0	A4	A3	A2	A1	A0
读写控制位	字节数控制位		起始地址控制位				

其中字节数控制位的具体含义如下所示：

Table 12. Byte Transfer Count

N1	N0	Description
0	0	Transfer 1 byte
0	1	Transfer 2 bytes
1	0	Transfer 3 bytes
1	1	Transfer 4 bytes

当传输多个字节时，$\overline{A4A3A2A1A0}$ 决定了起始寄存器的地址，之后的寄存器地址由 AD9715 内部产生，根据 LSB 优先控制位 LSBFIRST 决定。

- **数据传输阶段**

数据传输阶段完成了 AD9715 和系统控制器之间的实际数据传输，可以一次传输 1 个、2 个、3 个或 4 个字节。推荐一次传输多个字节的方式。在写寄存器时，每传输完一个字节的数据相应寄存器的值立即改变。

2.4.2 串行通信管脚描述

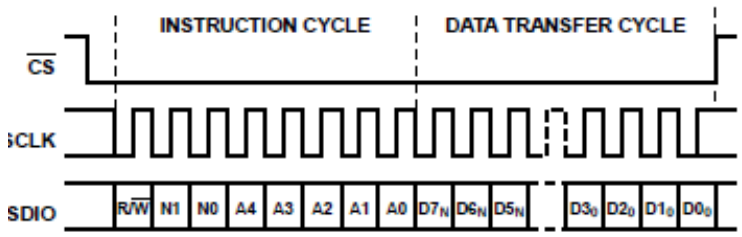
信号	作用	描述
SCLK	串行时钟	用于同步数据，最高 20MHz。输入数据上升沿有效，输出数据下降沿有效
$\overline{CS}$	片选信号	低电平选中，开始一个通信循环，在整个通信循环中需要保持为低电平
SDIO	串行数据 I/O	双向数据收发口

2.4.3 MSB/LSB 优先传输

AD9715 的串行口提供两种传输模式：MSB/LSB 优先传输，由 LSBFIRST 控制位（寄存器 0x00，Bit 6）控制。

模式	控制位（LSBFIRST）	位传输方向	字节传输方向	字节地址
MSB 优先（默认）	0	从 MSB 到 LSB	从高地址到低地址	递减
LSB 优先	1	从 LSB 到 MSB	从低地址到高地址	递增

MSB 优先，单字节模式写的时序如下图所示：



2.4.4 串行口操作

串行口配置受寄存器 0x00 控制，向 0x00 写一个完整的字节会使得配置立即改变。

软件重置位（寄存器 0x00，Bit 5）置 1（？）会使除了寄存器 0x00 之外的所有寄存器恢复默认值，0x00 保持不变。

建议改变串口配置时采用单字节传输或软件复位。

2.4.5 引脚模式

不需要写寄存器，也可以通过引脚来控制 AD9715（？）。

引脚	功能
RESET/PINMD	保持高，进入引脚模式
SCLK/CLKMD	提供时钟模式控制
SDIO/FORMAT	选择输入数据格式
$\overline{CS}$/PWRDN	控制掉电

外部电阻需要连接到 FSADJI 和 FSADJQ 来设置 DAC 电流，两个 DAC 都工作。

也可以通过在 FSADJI 和 FSADJQ 灌电流或拉电流来调整 DAC 电流，可以用运放来实现，和调整电阻的值效果相同。要在 DAC 右边串接至少 10 千欧的电阻来防止短路和噪声调制。REFIO 管脚也可以用相似的方式在 25%的范围内调整。

2.5 寄存器

完整的寄存器格式如下图所示：

Table 13.

Name	Addr	Default	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SPI Control	0x00	0x00	Reserved	LSBFIRST	Reset	LNGINS				
Power-Down	0x01	0x40	LDOOFF	LDOSTAT	PWRDN	Q DACOFF	I DACOFF	QCLKOFF	ICLKOFF	EXTREF
Data Control	0x02	0x34	TWOS	Reserved	IFIRST	IRISING	SIMULBIT	DCI_EN	DCOSGL	DCODBL
I DAC Gain	0x03	0x00	Reserved		I DACGAIN[5:0]					
IRSET	0x04	0x00	IRSETEN	Reserved	IRSET[5:0]					
IRCML	0x05	0x00	IRCMLEN	Reserved	IRCML[5:0]					
Q DAC Gain	0x06	0x00	Reserved		Q DACGAIN[5:0]					
QRSET	0x07	0x00	QRSETEN	Reserved	QRSET[5:0]					
QRCML	0x08	0x00	QRCMLEN	Reserved	QRCML[5:0]					
AUXDAC Q	0x09	0x00	QAUXDAC[7:0]							
AUX CTLQ	0x0A	0x00	QAUXEN	QAUXRNG[1:0]		QAUXOFS[2:0]			QAUXDAC[9:8]	
AUXDAC I	0x0B	0x00	IAUXDAC[7:0]							
AUX CTLI	0x0C	0x00	IAUXEN	IAUXRNG[1:0]		IAUXOFS[2:0]			IAUXDAC[9:8]	
Reference Resistor	0x0D	0x00	Reserved		RREF[5:0]					
Cal Control	0x0E	0x00	PRELDQ	PRELDI	CALSELQ	CALSELI	CALCLK	DIVSEL[2:0]		
Cal Memory	0x0F	0x00	CALSTATQ	CALSTATI			CALMEMQ[1:0]		CALMEMI[1:0]	
Memory Address	0x10	0x00	Reserved		MEMADDR[5:0]					
Memory Data	0x11	0x34	Reserved		MEMDATA[5:0]					
Memory R/W	0x12	0x00	CALRSTQ	CALRSTI		CALEN	SMEMWR	SMEMRD	UNCALQ	UNCALI
CLKMODE	0x14	0x00	CLKMODEQ[1:0]			Searching	Reacquire	CLKMODEN	CLKMODEI[1:0]	
Version	0x1F	0x03	Version[7:0]							

2.6 数字接口操作

DAC 的数字信号由一个单并行总线（DB[n：0]）提供，对于 AD9715，n 等于 9，还需要一个限定时钟（DCLKIO）。数字信号以 DDR 的格式传递到片上，最大速率为

250MSPS，时钟 125MHz。数据对的顺序和采样边沿的选择由数据控制位 IFIRST 和 IDATA 来决定，所以有四种可能的时序。

其中默认的时序（IFIRST = 1, IRISING = 1）如下图所示。

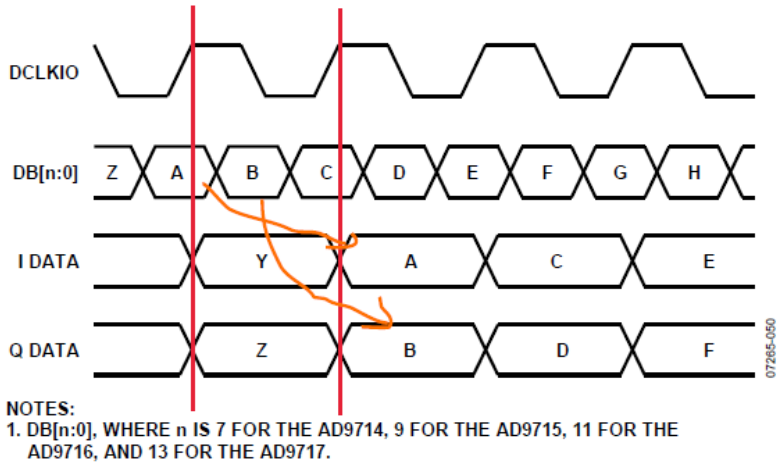


Figure 92. Timing Diagram with IFIRST = 1, IRISING = 1

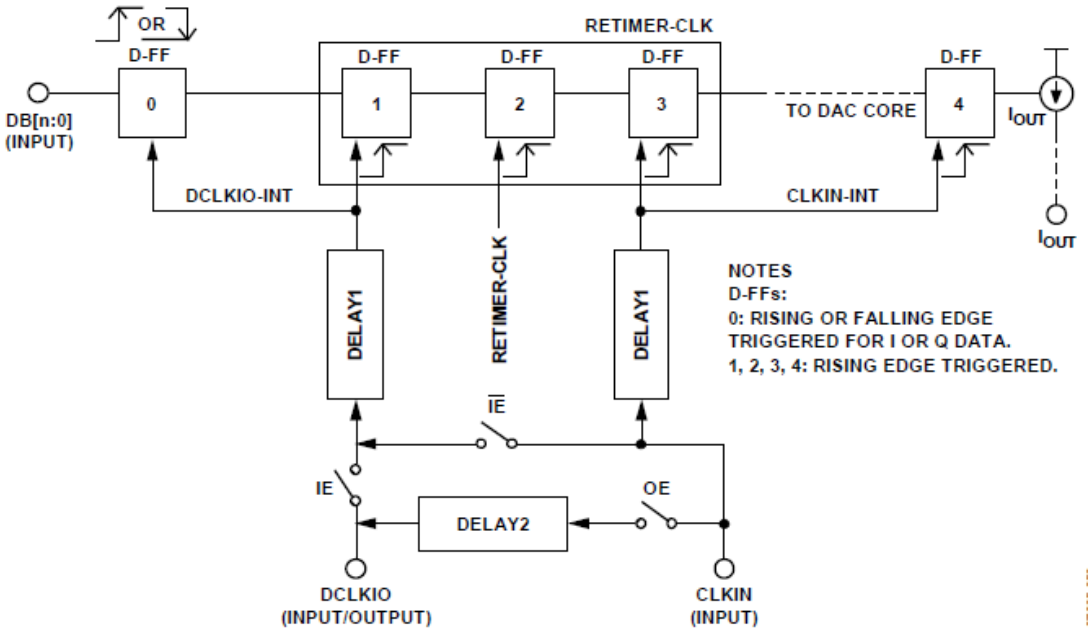
在数据对中 I DAC 的数据优先（Q DAC 的数据滞后），在 DCLKIO 的上升沿进行 I DAC 的数据锁存（在 DCLKIO 的下降沿进行 Q DAC 的数据锁存）。

时序上对信号的 setup time 和 hold time 都有要求，具体参考 Datasheet Table 2。

数据格式可以是无符号二进制数也可以是二进制补码，由 TWOS 数据控制位控制。

2.6.1 数字信号锁存和重定时模块

AD9715 的时钟单元结构如下所示：



AD9715 有两个时钟输入，DCLKIO 和 CLKIN。CLKIN 是模拟时钟影响 DAC 的性能；DCLKIO 是数字时钟，需要和输入数字信号有一种固定的关系，来保证数据能够正确地被锁存。

DDR 数据的传输模式由 IRISING 和 IFIRST 控制，上一节已经说过了。

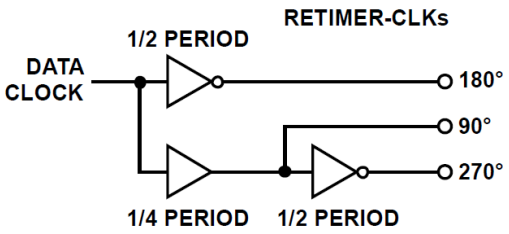
在默认的引脚模式和 SPI 设置中，IE 为高电平（关闭）OE 为低电平（打开）。这种设置在 RESET/PINMODE（管脚 35）被置为高电平时生效。在这种模式下，用户需要同时提供 DCLKIO 和 CLKIN。在引脚模式下，也推荐 DCLKIO 和 CLKIN 采用相同的相位，使得 DAC 能够正常工作。用户也可以访问 SPI，将控制位 DCI_EN（寄存器 0x02，Bit 2）置低，使得 CLKIN 也用作 DCLKIO。

DCOSGL 和 DCODBL 可以控制 CLKIN 从 DCLKIO 输出。

- **重定时器**

AD9715 有内部重定时电路，可以比较 CLKIN-INT 时钟和 DCLKIO-INT 时钟，并基于它们的相位关系选择一个重定时时钟，从而实现从用于片上输入接口的 DCLKIO 时钟到用于 DAC 模拟核心的 CLKIN 时钟的安全转换。

重定时器会选择下面三个相位之一，受 CLKMODE 控制位控制。



注意，很多情况下多个重定时相位都能使 DAC 正常工作，在这种情况下，重定时器不一定能够选到最好、最安全的相位。如果用户知道 CLKIN 和 DCLKIO 之间的相位关系，那么可以将 CLKMODEN 置为 1 强制重定时器固定为这个相位。

重定时器相关的寄存器格式和定义如下表所示：

Table 15. Timer Register List

Bit Name	Description
CLKMODEQ[1:0]	Q data path retimer clock selected output. Valid after the searching bit goes low.
Searching	High indicates that the internal data path retimer is searching for the clock relationship (DAC is not usable until it is low again).
Reacquire	Changing this bit from 0 to 1 causes the data path retimer circuit to reacquire the clock relationship.
CLKMODEN	0: uses CLKMODEI/CLKMODEQ values (as computed by the two internal retimers) for I and Q clocking. 1: uses the CLKMODE value set in CLKMODEI[1:0] to override the bits for both I and Q retimers (that is, force the retimer).
CLKMODEI[1:0]	I data path retimer clock selected output. Valid after searching goes low. If CLKMODEN = 1, a value written to this register overrides both the I and Q automatic retimer values.

Table 16. CLKMODEI/CLKMODEQ Details

CLKMODEI[1:0]/CLKMODEQ[1:0]	DCLKIO-to-CLKIN Phase Relationship	RETIMER-CLK Selected
00	0° to 90°	Phase 2
01	90° to 180°	Phase 3
10	180° to 270°	Phase 3
11	270° to 360°	Phase 1

当 RESET 引脚被拉高又被拉低，器件会进入 SPI 模式，重定时器会开始运行并在 128 个时钟周期之内自动选择一个合适的时钟相位。SPI 搜索标识位（寄存器 0x14，Bit

4）会被置低，表示重定时器被锁定，器件已经可以使用（相当于一个初始化过程）。重获取控制位（寄存器 0x14，Bit 3）可以用来在任何时候重新启动 I 和 Q 重定时器的相位检测。可以从寄存器 0x14 的 CLKMODEQ[1:0]和 CLKMODEI[1:0]读取重定时器内部相位检测器选取的值。

SPI 模式下强制选取相位的方法前面已经说过了。

引脚模式下，用户最好把 CLKIN 和 DCLKIO 连接到一起。不过器件也通过复用 SPI 引脚提供了一些可编程的功能。略。

2.6.2 总 DAC 流水线延时的估计

略。

2.6.3 DAC 传输函数

DAC 输出电流与输入代码的关系如下：

$$IOUTP = (IDAC\ CODE/2^N) \times I_{IOUTFS}$$
$$QOUTP = (QDAC\ CODE/2^N) \times I_{QOUTFS}$$
$$IOUTN = ((2^N - 1) - IDAC\ CODE)/2^N \times I_{IOUTFS}$$
$$QOUTN = ((2^N - 1) - QDAC\ CODE)/2^N \times I_{QOUTFS}$$

满标电流与参考电压、电阻的关系如下：

$$I_{IOUTFS} = 32 \times I_{IREF}$$
$$I_{QOUTFS} = 32 \times I_{QREF}$$

re:

$$I_{IREF} = V_{REFIO}/IR_{SET}$$
$$I_{QREF} = V_{REFIO}/QR_{SET}$$

$$I_{IOUTFS} = 32 \times V_{REFIO}/IR_{SET}$$
$$I_{QOUTFS} = 32 \times V_{REFIO}/QR_{SET}$$

单端输出电压与输出电流的关系如下：

$$V_{IOUTP} = IOUTP \times IR_{LOAD}$$
$$V_{QOUTP} = QOUTP \times QR_{LOAD}$$
$$V_{IOUTN} = IOUTN \times IR_{LOAD}$$
$$V_{QOUTN} = QOUTN \times QR_{LOAD}$$

差分输出电压的表达式如下：

$$V_{IDIFF} = \{(2 \times IDAC\ CODE - (2^N - 1))/2^N\} \times (32 \times V_{REFIO}/IR_{SET}) \times IR_{LOAD}$$

2.7 重要引脚定义

AD9715 的重要引脚（特别是与固件设计相关的引脚）定义如下：

符号	类型	描述
DB[9:0]	Input	数字输入
DCLKIO	I/O	数据输入/输出时钟，用于限定输入数据
CLKIN	Input	LVCMOS 电平的采样时钟输入
IOUTP	Output	I DAC 的电流输出，当所有数据位为 1 时输出满标电流
IOUTN	Output	I DAC 的互补电流输出，当所有数据位为 0 时输出满标电流
RESET/PINMD	Input	决定器件操作模式，低电平进入 SPI 模式，先产生高电平脉冲再拉低复位所有 SPI 寄存器；高电平进入引脚模式
SCLK/CLKMD	Input	功能 1：串口时钟，在 SPI 模式作为串口时钟输入； 功能 2：时钟模式，在引脚模式决定了内部重定时时钟的相位
SDIO/FORMAT	I/O	功能 1：串口输入/输出，在 SPI 模式作为串口的双向数据线； 功能 2：格式引脚，在引脚模式决定输入数据的格式，逻辑低采用无符号二进制格式，逻辑高采用二进制补码格式
$\overline{CS}$/PWRDN	Input	功能 1：片选信号，在 SPI 模式作为低有效的片选； 功能 2：掉电控制信号，在引脚模式高电平使得全器件掉电，除了 SPI 接口

3 固件开发的尝试

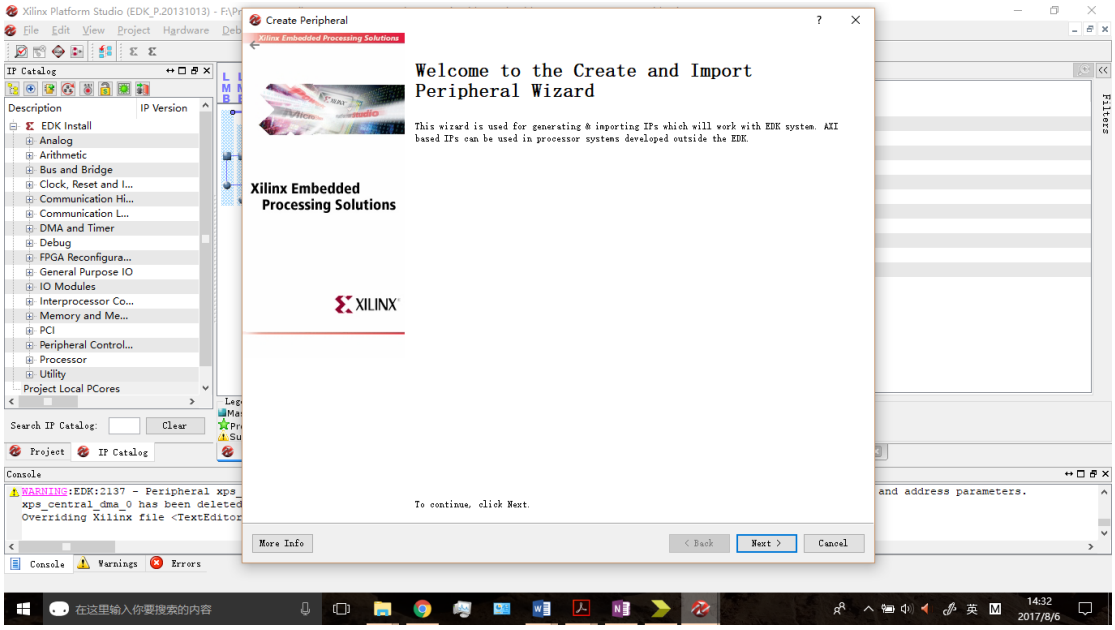
3.1 固件设计的设想

先实现最简单的 DAC 控制功能。固件作为自定义 IP 核添加到 MicroBlaze 的外设当中去，提供几个寄存器用来传递数据和控制 DAC。固件通过上述一些信号和 AD9715 相连，实现最基本的数字信号转换成模拟信号的功能。为了简单起见，采用 Pin Mode，不用 SPI。

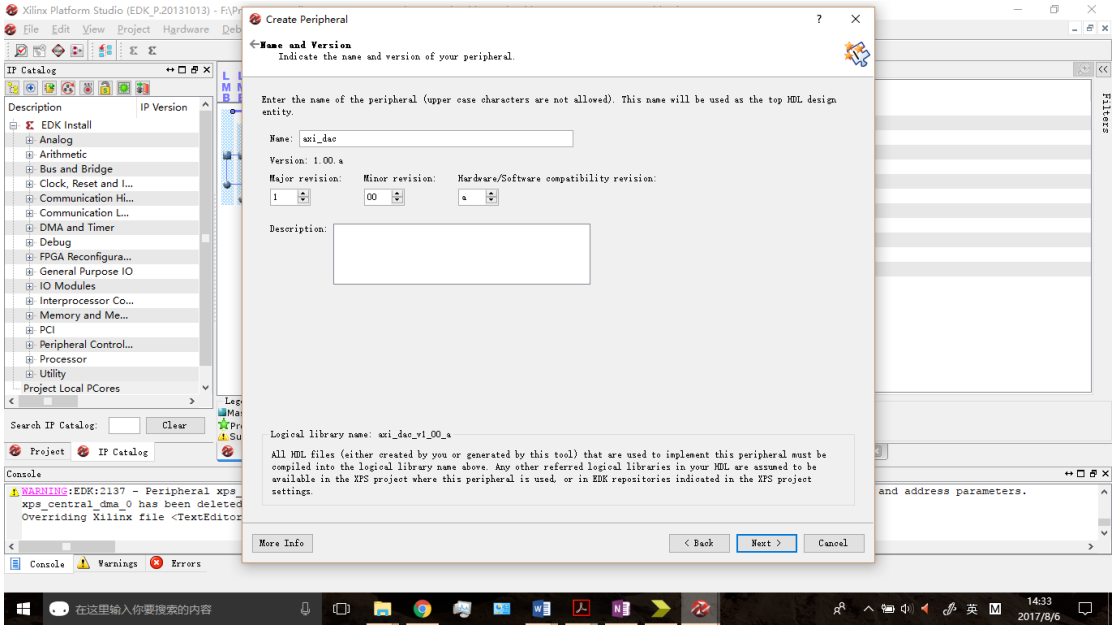
简单的行为级描述如下：MicroBlaze 核向固件的数据寄存器中写一个数据，固件就将这个数据传递给 AD9715，产生相应的模拟信号。

3.2 自定义 IP 核的生成

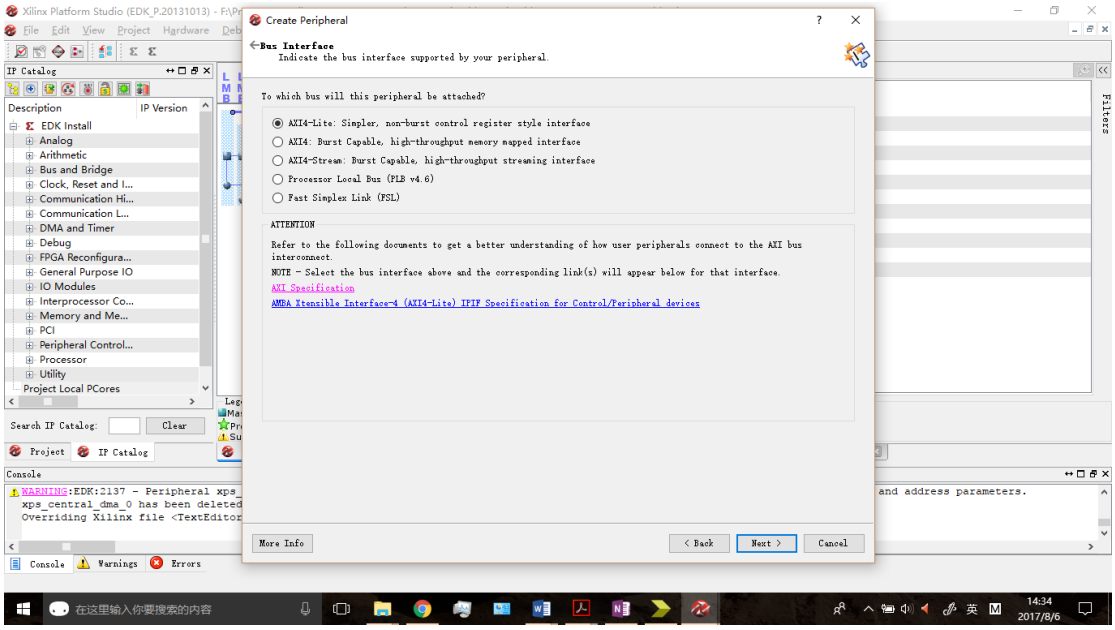
尝试在 XPS 中自定义 IP 核，点击 Hardware -> Create or Import Peripheral…，打开新建和导入外设向导。



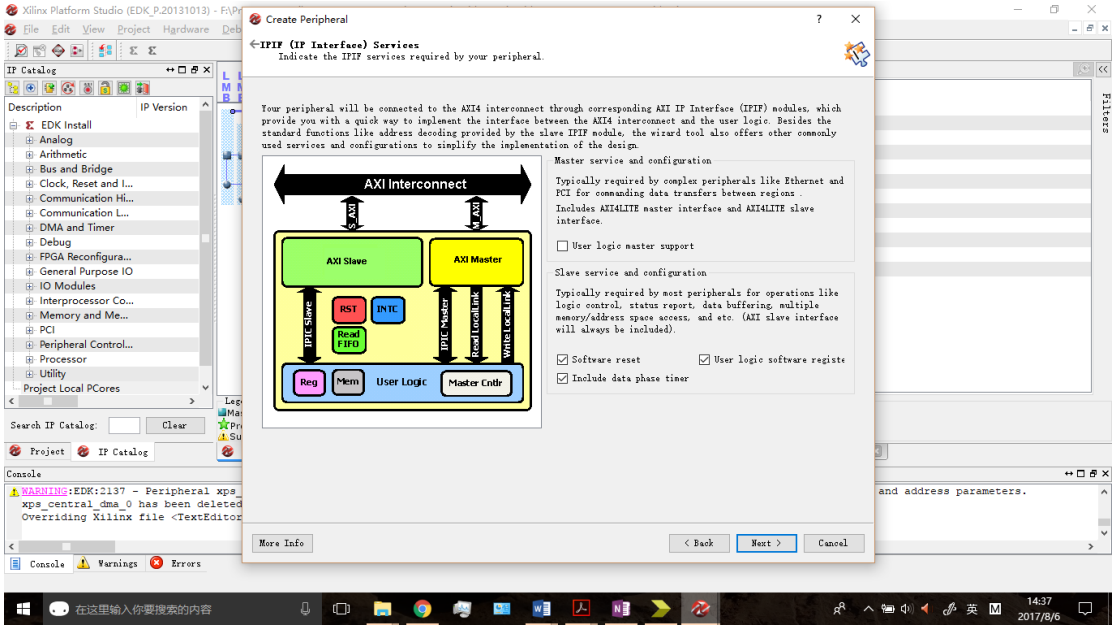
尝试创建一个名为 axi_dac 的外设。



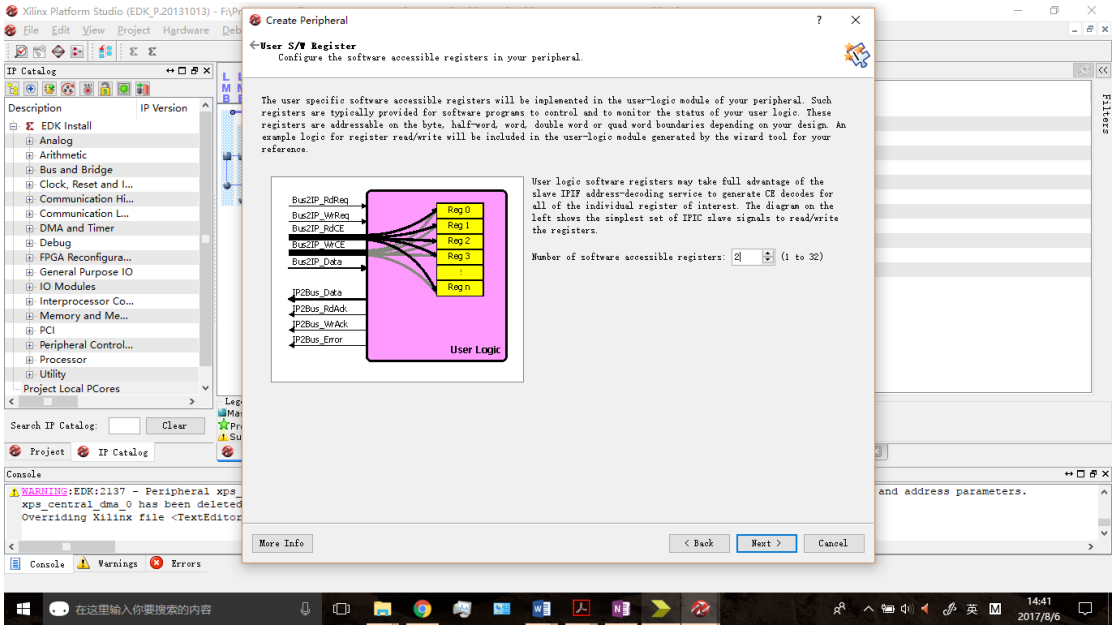
选择较为简单的 AXI4-Lite 总线。



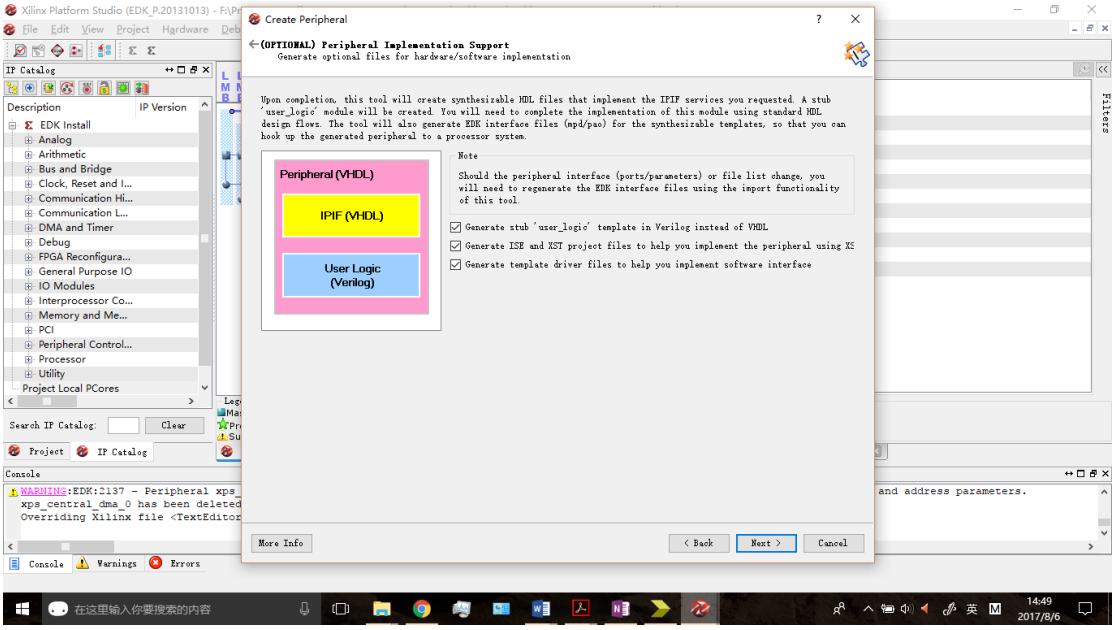
启用软件复位、逻辑软件寄存器和数据相位时钟。



用 2 个寄存器实现用户对接口的控制。



采用 Verilog 描述用户逻辑，生成项目和驱动文件。



完成后，向导会自动生成一系列与自定义 IP 核有关的文件，包括 IP 核的描述、用户逻辑 Verilog 模块的模板和驱动文件的示例等，还有一个项目文件。用 ISE Project Navigator 打开这个项目文件，就可以在这里进行 IP 核的开发。

3.3 DAC 接口固件的实现

在 user_logic.v 中完成 DAC 接口固件的实现。最初尝试的具体逻辑实现部分如下：

```
//Implement DAC control signals

//The last DAC_WIDTH bits of slv_reg1 are used as DAC data bits.
assign IP2DAC_Data[DAC_WIDTH-1 : 0] = slv_reg1[DAC_WIDTH-1 : 0];

//The LSB of slv_reg0 is used as DAC_EN, an enable signal. If DAC_EN = 0, DAC powers down.
assign IP2DAC_PWRDN = ~slv_reg0[0];

//The 2nd LSB of slv_reg0 is used as FRMT_CTL, a format control bit.
//If FRMT_CTL=0, unsigned binary format will be selected. Otherwise, 2's complement format will be selected.
assign IP2DAC_Format = slv_reg0[1];

//The digital clock and analog clock are both tied to bus clock, with CLKMD=0.
assign IP2DAC_DCLKIO = Bus2IP_Clk;
assign IP2DAC_Clkout = Bus2IP_Clk;
assign IP2DAC_ClkMD = 1'b0;

//The DAC should be fixed to work in pin mode.
assign IP2DAC_PinMD = 1'b1;
```

好像还需要在顶层模块中声明这些端口。顶层模块在 axi_dac.vhd 中，是用 VHDL 写的，修改起来较为麻烦。

接口修改完成后，综合通过。

初步设计的寄存器控制格式如下：

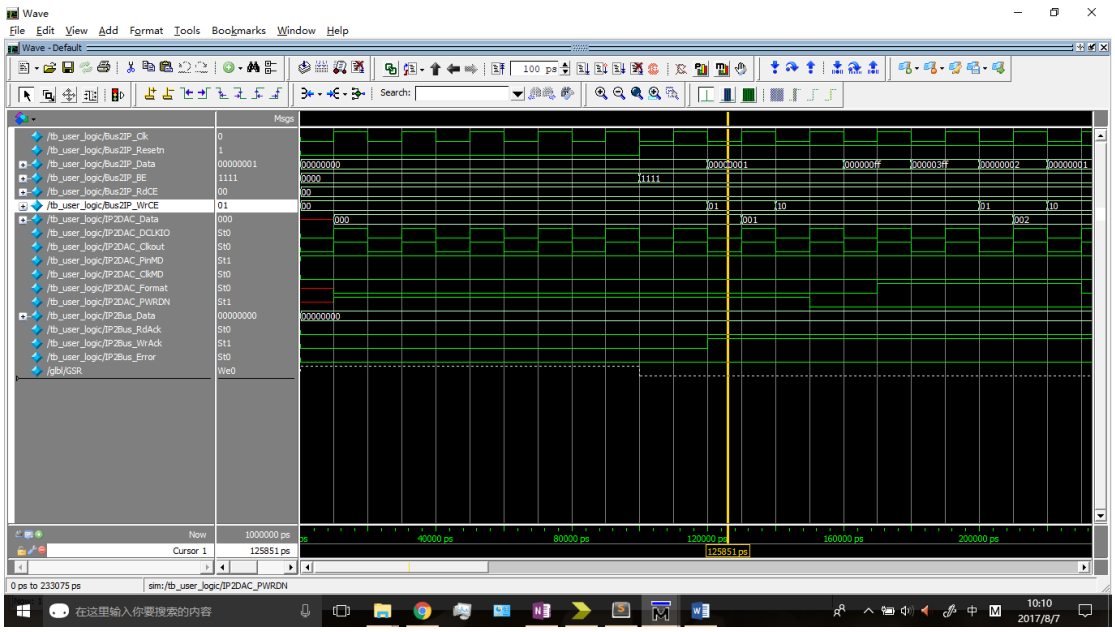
	MSB		LSB
reg0 DAC_CTL DAC控制寄存器	b31 - b2	b1	b0
	N/A	FRMT_CTL	DAC_EN
	备用	格式控制位，0表示无符号二进制数，1表示二进制补码，默认为0	DAC使能控制位，为0时DAC掉电，为1时正常工作
reg1 DAC_DATA DAC数据寄存器	b31-b10		b9-b0
	N/A		DAC_DATABITS
	备用		DAC数据位

3.4 DAC 接口固件的仿真

3.4.1 用户逻辑（user_logic）部分单独仿真

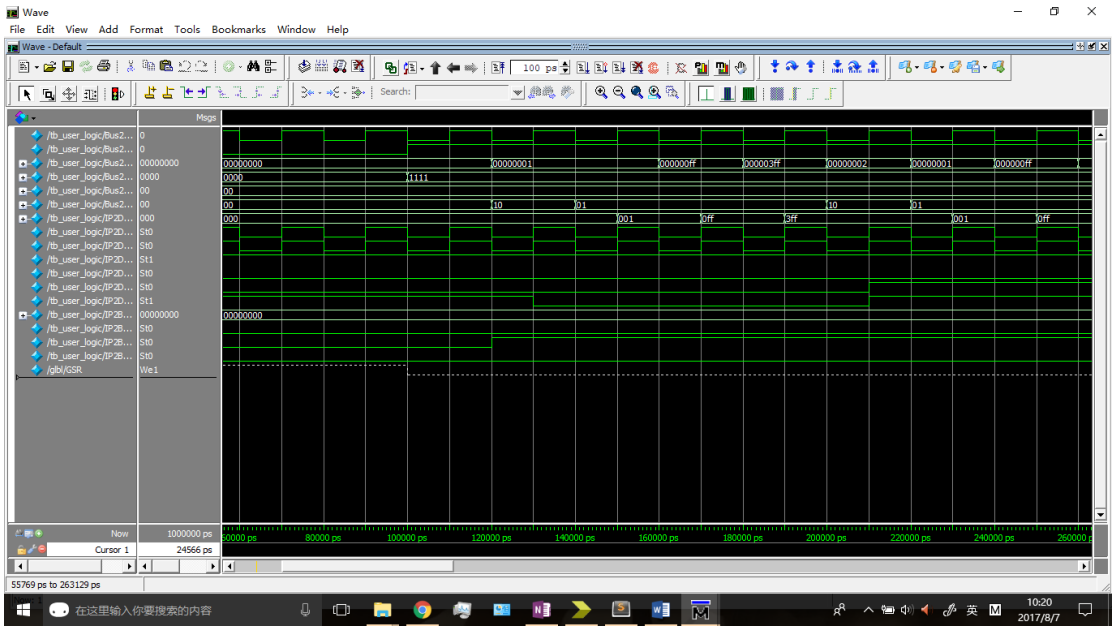
先对用户逻辑（user_logic）部分进行单独仿真，测试寄存器读写和 DAC 控制信号产生功能是否正确。因为不涉及到太多总线信号，所以 testbench 写起来较为容易。

在仿真过程中，也发现了一些问题。



如 100ns 处，本应向寄存器 0（reg0, DAC_CTL）中写 0x0000_0001 产生使能信号使 DAC 从掉电状态恢复到正常工作状态，结果却发现 0x0000_0001 写到了寄存器 1(reg1, DAC_DATA)中产生了 001 的数字信号 。检查代码发现，XPS 生成的用户逻辑模板中，写选通信号 2'b01 对应的是寄存器 1，而 2'b10 对应的是寄存器 0，比较容易混淆。

修改写选通信号的格式后，仿真的结果符合了预期。



3.4.2 固件在 MicroBlaze 系统中的仿真

在将固件添加到 MicroBlaze 系统中之后，还需要用二进制激励文件对它进行仿真。这个步骤放到后面的章节中。

3.5 DAC 接口固件的添加

3.5.1 接口固件输出端口的声明

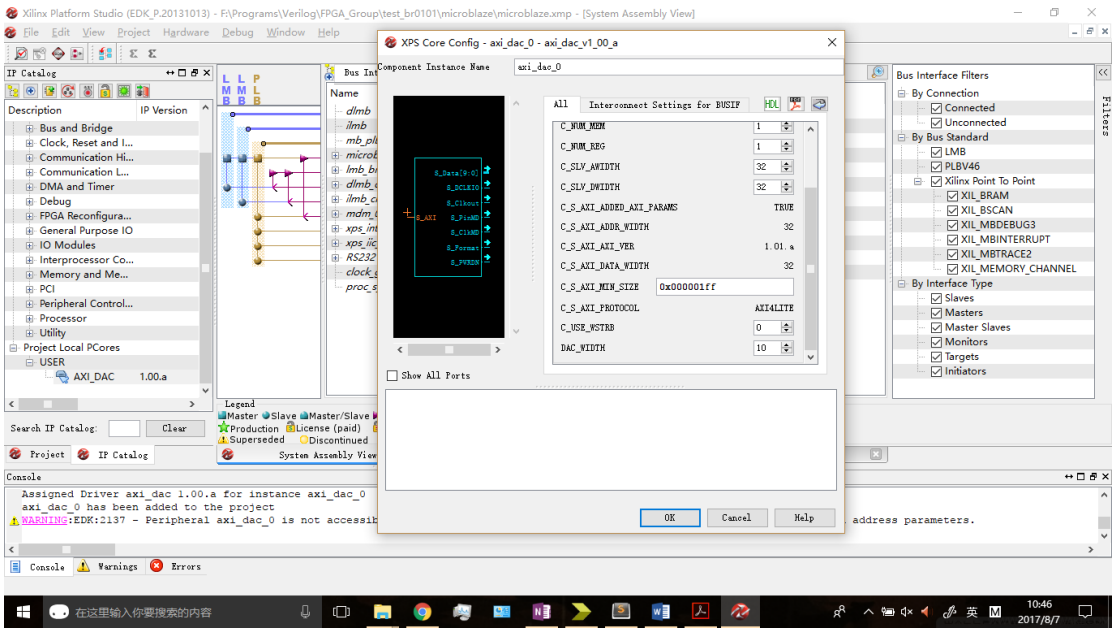
因为这个固件需要通过几个输出端口发出信号控制片外的 DAC，所以好像还需要在外设的说明文件中添加上这些端口的信息。

在 axi_dac_v2_1_0.mpd 中，添加如下信息：

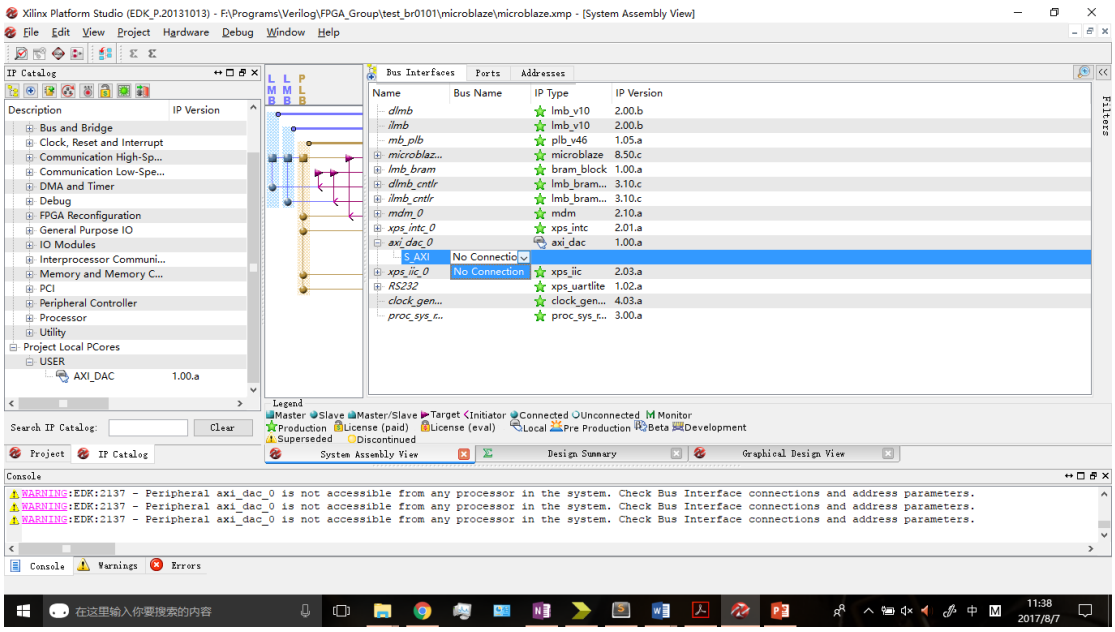
```
PORT S_Data    = "", DIR = O, VEC = [(DAC_WIDTH-1) : 0]
PORT S_DCLKIO = "", DIR = O
PORT S_Clkout = "", DIR = O
PORT S_PinMD   = "", DIR = O
PORT S_ClkMD   = "", DIR = O
PORT S_Format = "", DIR = O
PORT S_PWRDN   = "", DIR = O
```

3.5.2 将 IP 核添加到系统中

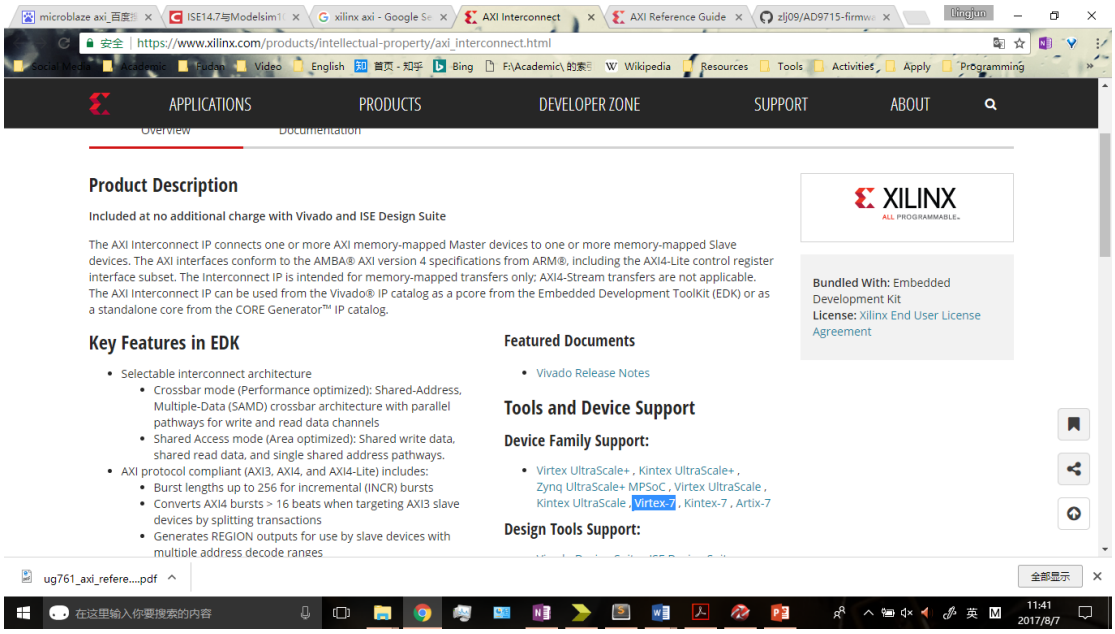
在 XPS 中重新扫描 IP 核之后，尝试将 aix_dac 添加到系统中。



结果发现添加好的 IP 核无法连接到 MicroBlaze 上，因为没有 AXI 总线。



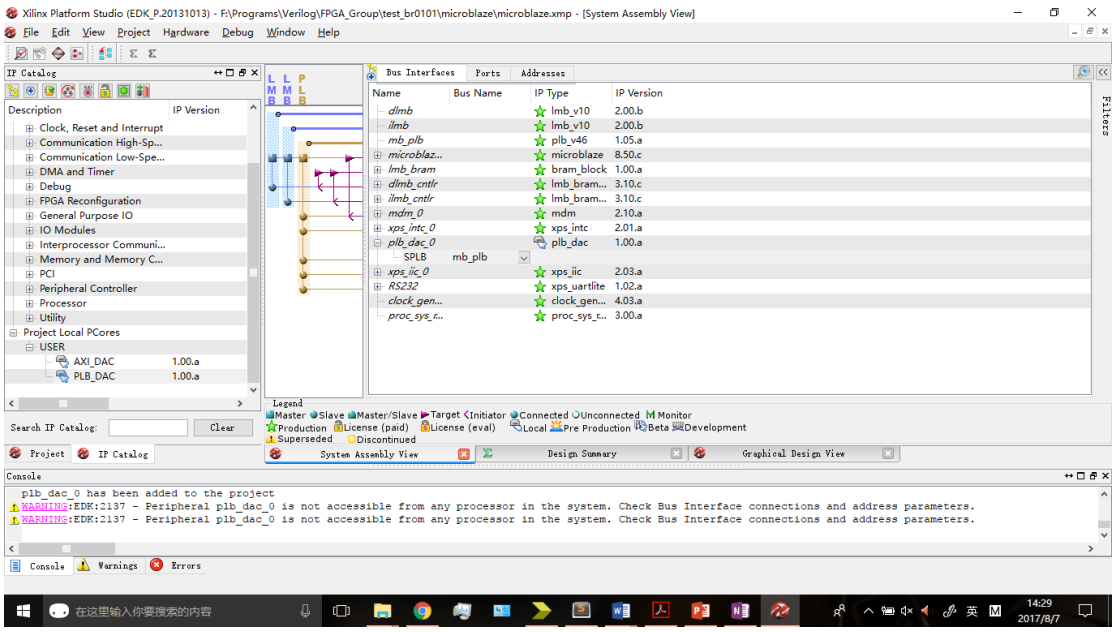
查阅官网并咨询王星学长后才发现，Vertex-4 不支持 AXI 总线。



所以只能尝试换成 PLB 总线，把上述流程再来一遍。

需要注意的是，PLB 总线的大小端规定似乎与 AXI 总线相反，很多接口信号向量的方向需要调整，仿真中也容易出现很多问题。

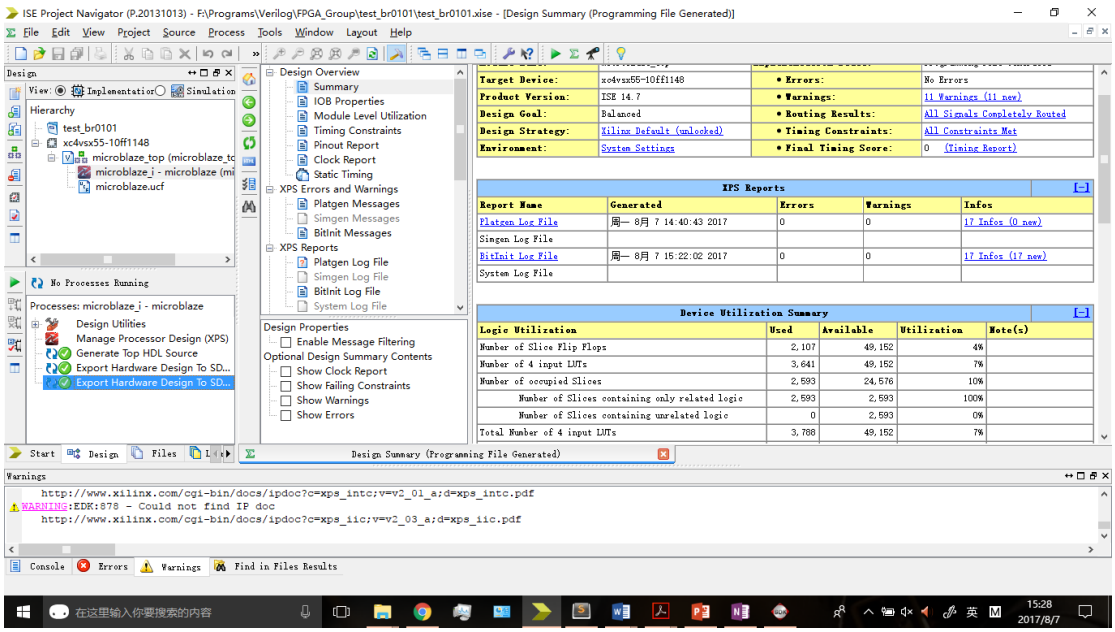
最终可以将 plb_dac 添加到系统中，并连接到 PLB 总线上。



再经过声明外部端口、分配地址等步骤，就可以将 IP 核完整地添加到系统中。

3.5.3 综合、生成位流并导出

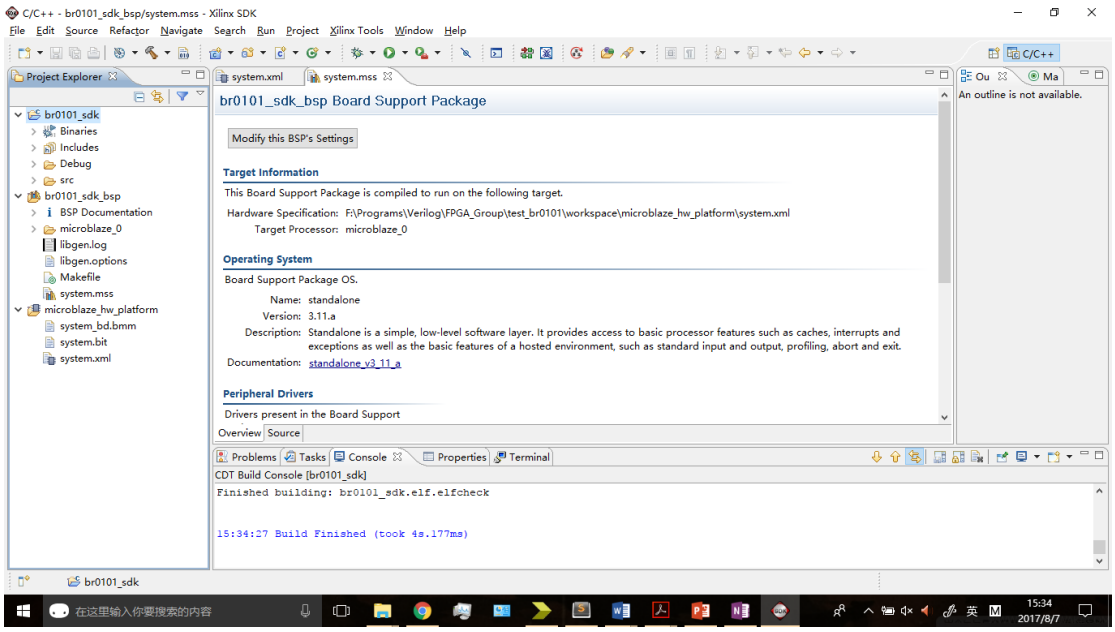
由于暂时没有拿到 BR0101 的原理图，所以还没有办法将.ucf 用户约束文件补充完整，先用临时的.ucf 综合、生成位流，并导出到 SDK 中。



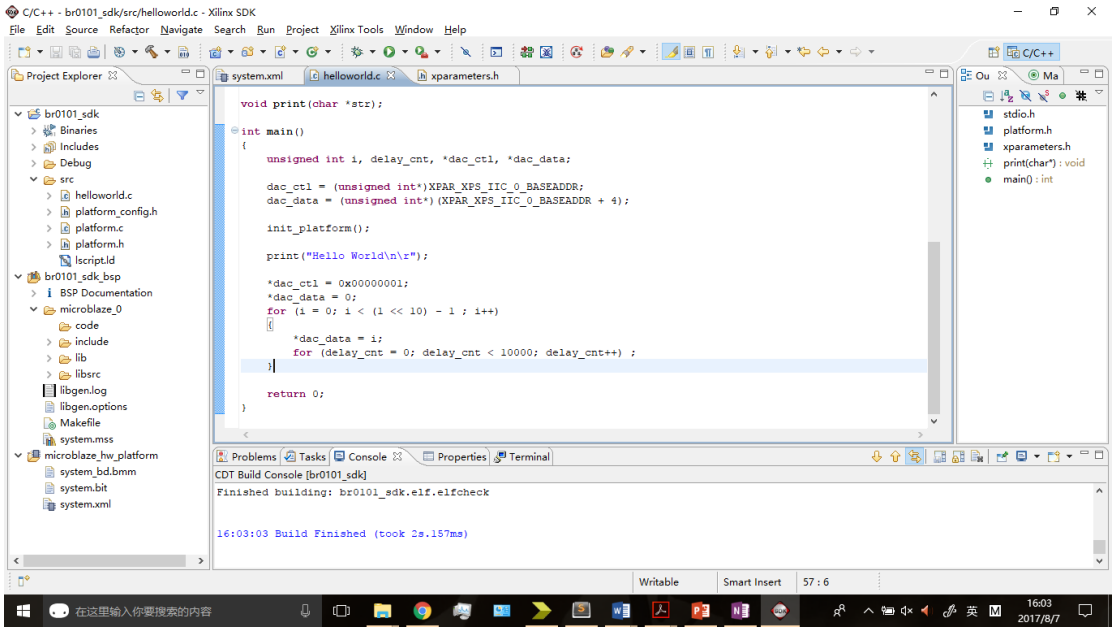
3.5.4 补充用户约束文件

3.6 固件驱动设计与测试的尝试

在 SDK 中新建应用工程。



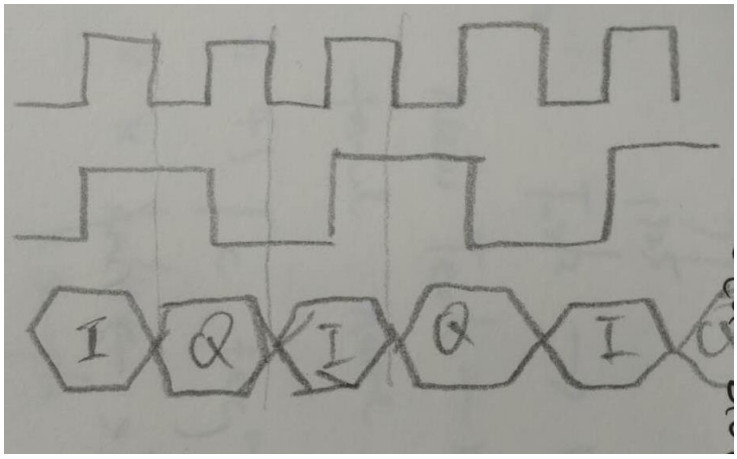
完成了简单的测试程序。



3.7 DAC 接口固件的改进

3.7.1 DAC 双通道信号的支持

之前没有注意到 AD9715 的电路中引出了 I DAC 和 Q DAC 两个通道，所以在接口固件中没有提供对双通道交叉读写的支持。



默认状态下，I DAC 与 Q DAC 数据交叉读写的关系如上图所示。为了实现双通道交叉读写的功能，我们需要对时钟进行分频，并在数据输出端口加上多路选择器。对 Verilog 代码的修改如下：

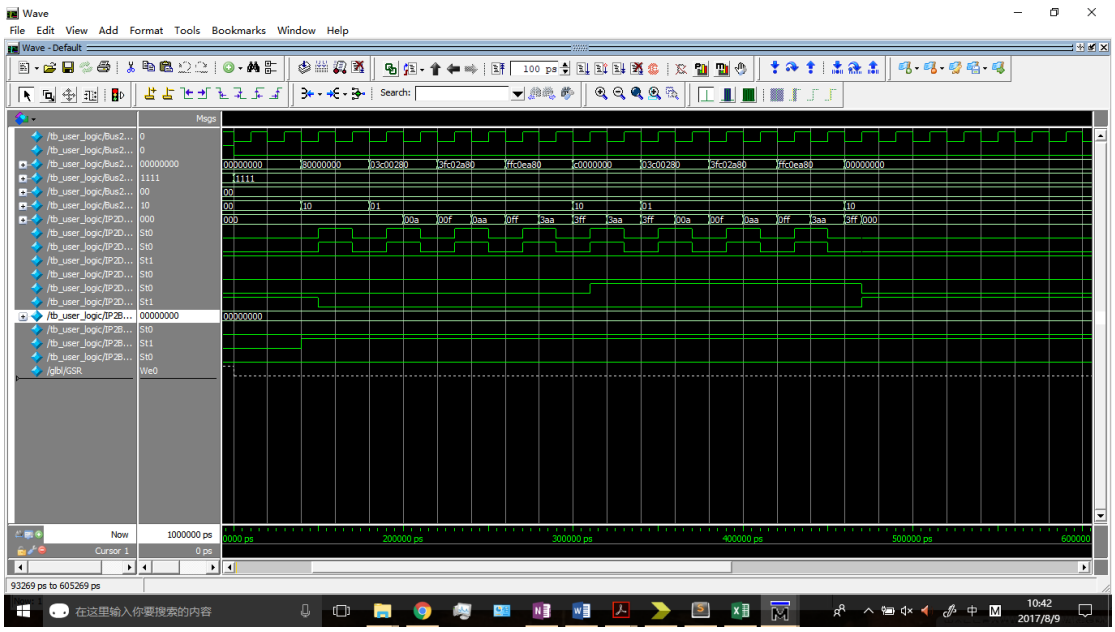
```
// generate DAC clock here, with a frequency divider imposed on the bus clock
always @(posedge Bus2IP_Clk)
  begin
    if (Bus2IP_Reset == 1)
      dac_clk_reg <= 1'b0;
    else
      dac_clk_reg <= ~dac_clk_reg;
  end

always @(negedge Bus2IP_Clk)
  begin
    if (Bus2IP_Reset == 1)
      dac_data_reg <= 10'b0;
    else begin
      dac_data_reg <= (dac_clk_reg) ? (slv_reg1[16: 25]) : (slv_reg1[0 : 9]);
    end
  end
```

修改之后，DAC 接口固件的寄存器格式发生了变化，具体如下所示：

	MSB			LSB
reg0 DAC_CTL DAC控制寄存器	b31 - b2	b1	b0	
	N/A	FRMT_CTL	DAC_EN	
	备用	格式控制位，0表示无符号二进制数，1表示二进制补码，默认为0	DAC使能控制位，为0时DAC掉电，为1时正常工作	
reg1 DAC_DATA DAC数据寄存器	b31-b26	b25-b16	b15-b10	b9-b0
	N/A	DAC_Q_DATA	N/A	DAC_I_DATA
	备用	Q DAC数据位	备用	I DAC数据位

对修改后的固件进行仿真，结果如下所示：



可见，修改后的固件能够将 Q DAC 与 I DAC 的数据分离开，实现双通道的交叉读写。

之后还需要重新进行固件导入、位流更新和驱动改写。

4 模拟电路的分析与讨论

详见 AD9715 相关模拟电路的分析和讨论.docx。