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。

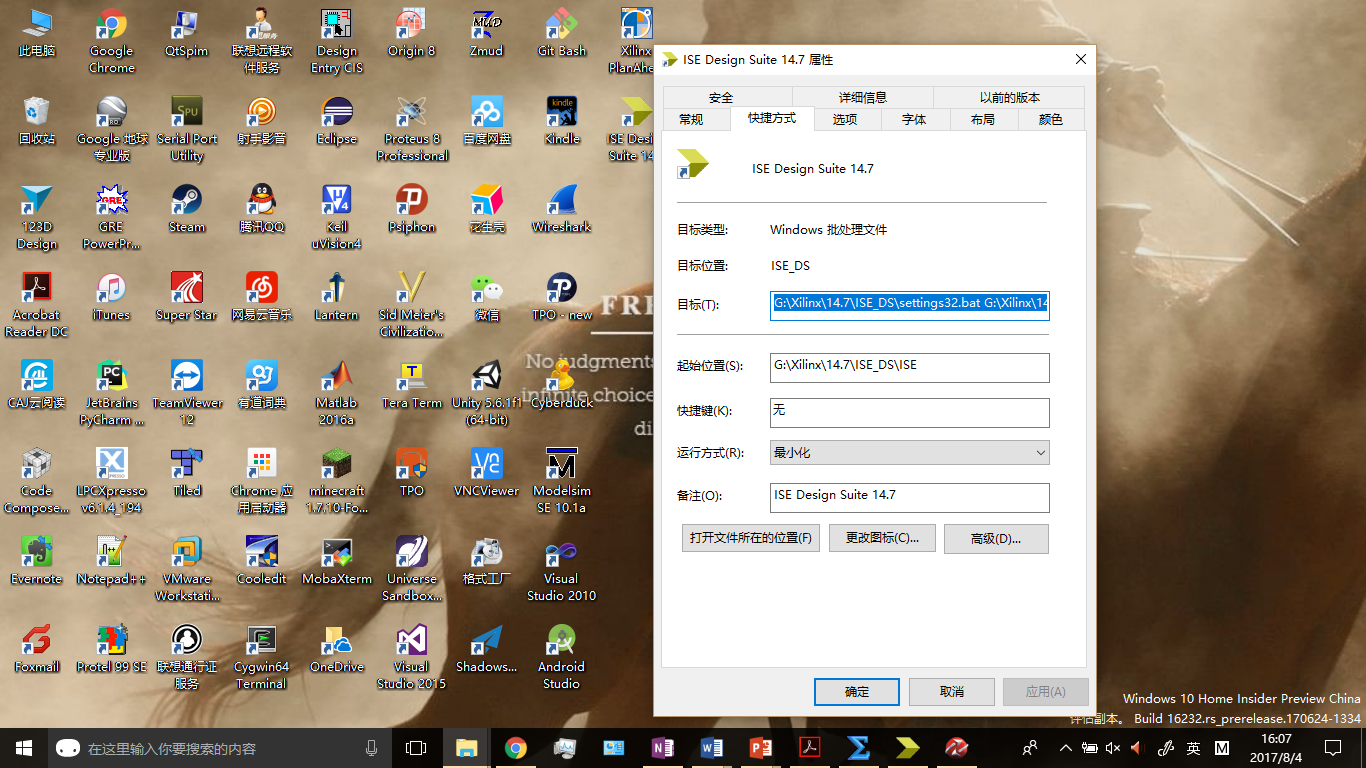
# 1 准备开发环境

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。

一 》 ECDK 
0 
0 ， 3 \ × 山 n 对 1 7 SE DS \ ED ℃ 上 in \ nt6d 乱 
囗 
过 
囗 
过 
Setting XlLlWk_EDK=G:\Xi1inx\14. 7\lSE_DS\EDK 
全 部 消 
士 反 向 〕 西 
史 记 录 
trl+E 〕 
． nt64 ． 月 
萨 0 下 下 园 0 一 0 一 0 国 思 
32 KB 
R0101 子 ． 纟 召 
41 KB 
DM 《 接 囗 一 通 信 百 科 
xps xe 一 椠 纟 充 钅 《 吴 
FPG 众 在 高 性 能 计 算 方 面 的 
2015 ． 08 一 29 
无 法 续 执 行 代 码 。 新 安 程 可 能 会 
于 拢 不 到 QtC 。 re.dl 《 
] 2015 ． 08 一 30 
蚓 关 此 河 题 。 
会 ] 2015 ． 08 一 21 
8R0101 充 固 ， 牛 」 钦 ， 牛 ， f 发 
” ] 2017 ． 08 [ 还 
xdsgen.exe 
2013 / 10 / 146 ： 2 彐 
48 KB 
划 bf 。 e 艹 
应 程 
2013 / 10 / 14 023 
25 KB 
[Exmd.exe 
应 程 
2013 / 10 / 145 ： 24 
32 KB 
应 程 
2013 / 10 / 146 ： 2 彐 
24 KB 
0 xps archive 
应 程 
2013 / 10 / 14 023 
go KB 
应 程 
0 三 
2013 / 10 / 14 023 
17 KB 
． OneDmve 
应 程 
2013 / 10 / 14 023 
257 KB 
[ i? e 艹 
应 程 
2013 / 10 / 14 023 
32 KB 
应 程 
2013 / 10 / 14 023 
32 KB 
zynq flash.exe 
应 程 
2013 / 10 / 14024 
32 KB 
4 彐 个 顼 目 远 中 1 个 顼 目 24 ℃ KB 
1 1 ： 14 
0 吨 
[ 囗 ] 
在 这 里 输 入 你 要 的 丙 容 
2017 / 8 / 4 

因此拆卸并重新安装Xilinx Design Suite 14.7。

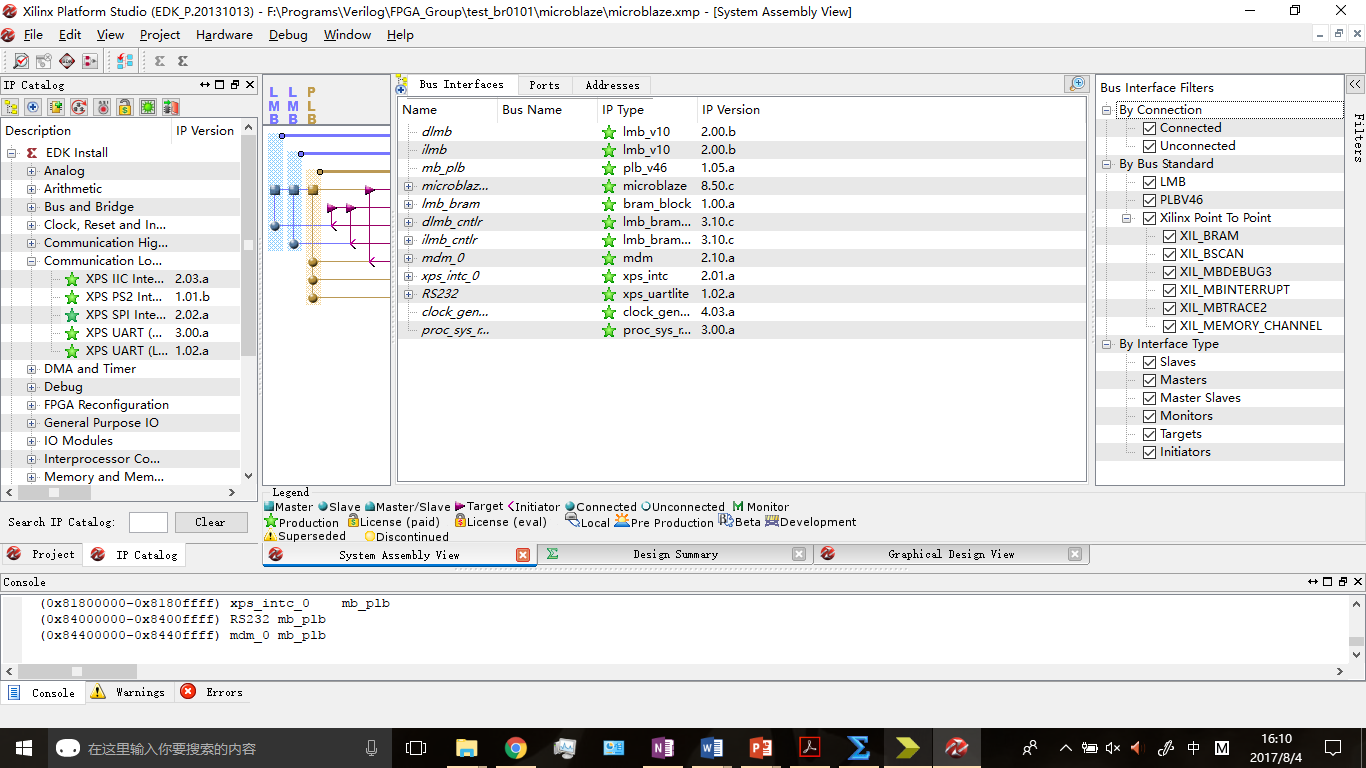
后来发现好像不是安装的问题，而是因为64位的版本有问题。



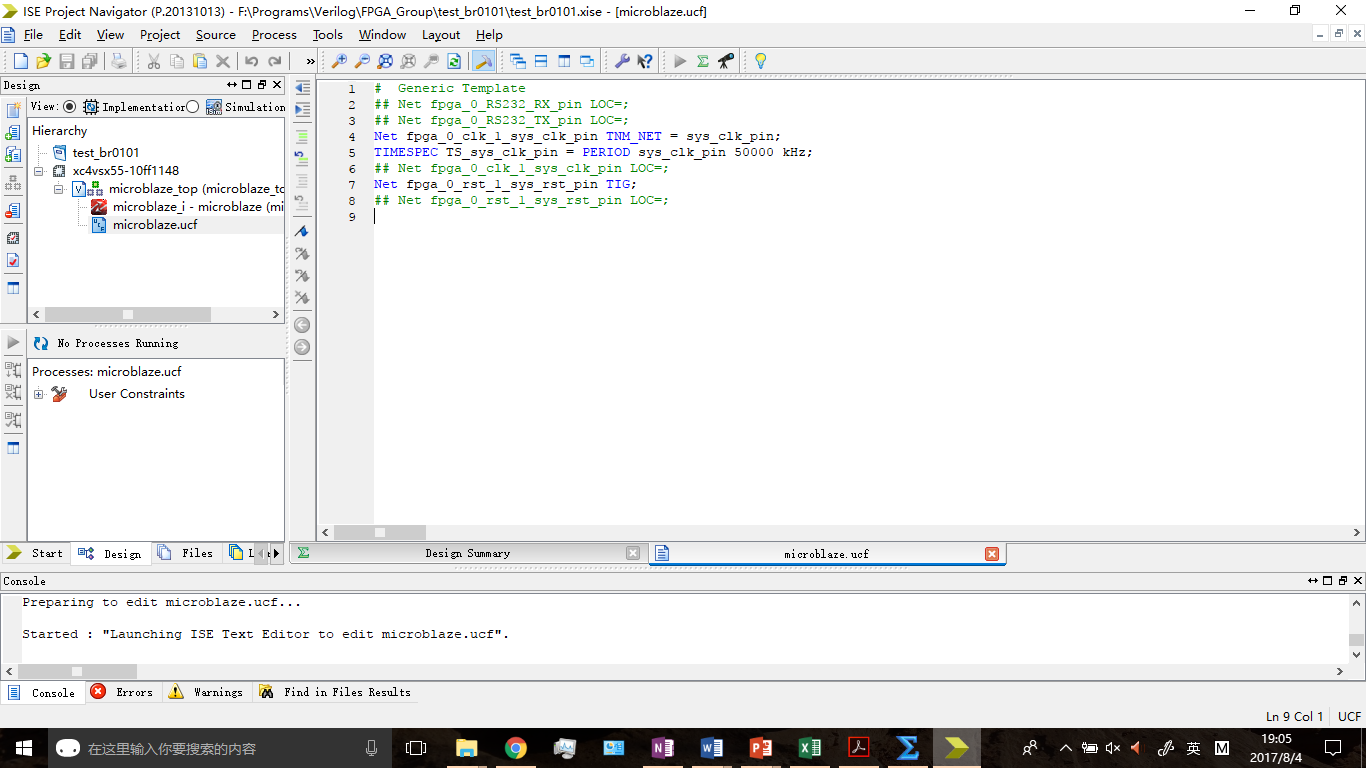
将ISE Design Suite 14.7的启动方式改为32位模式启动，命令如下：

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

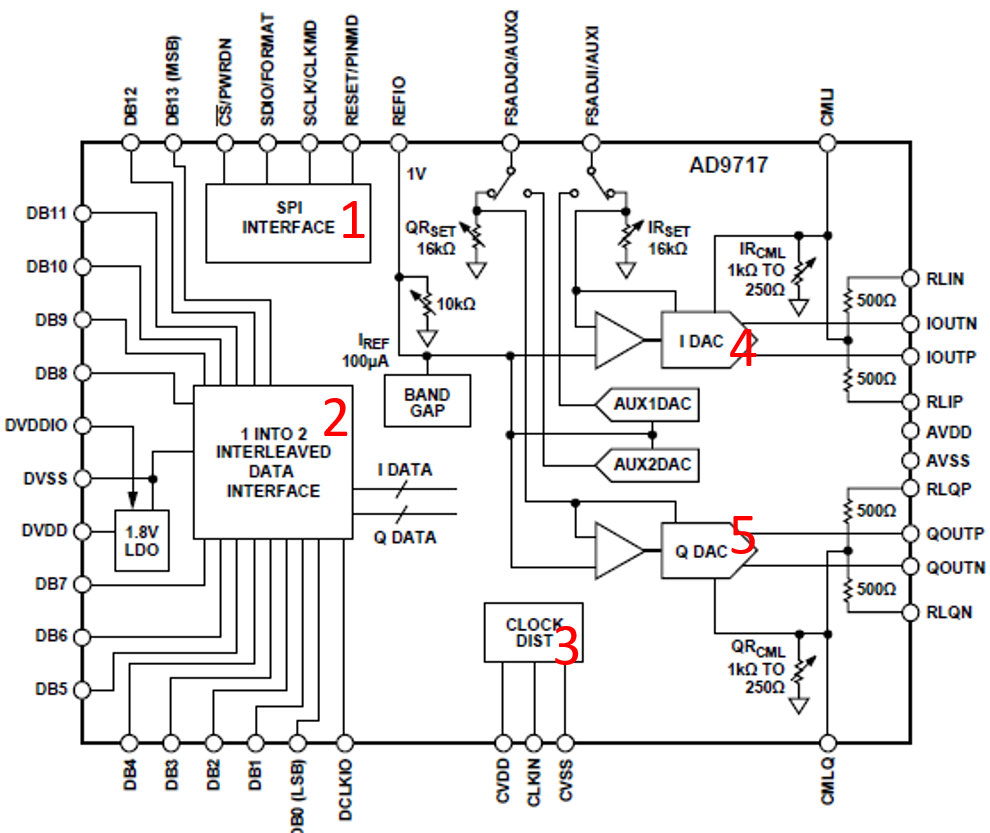
## 2.1 功能调研

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 |  |
| 中间4个bit | DB4 – DB1 | 15 |  |
| 最后的LSB | DB0 | 1 |  |

### 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

参考电流由参考控制放大器和参考电压共同决定，参考控制放大器受FSADJx引脚上的电阻控制。满标输出电流与参考电流的关系为：



## 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。

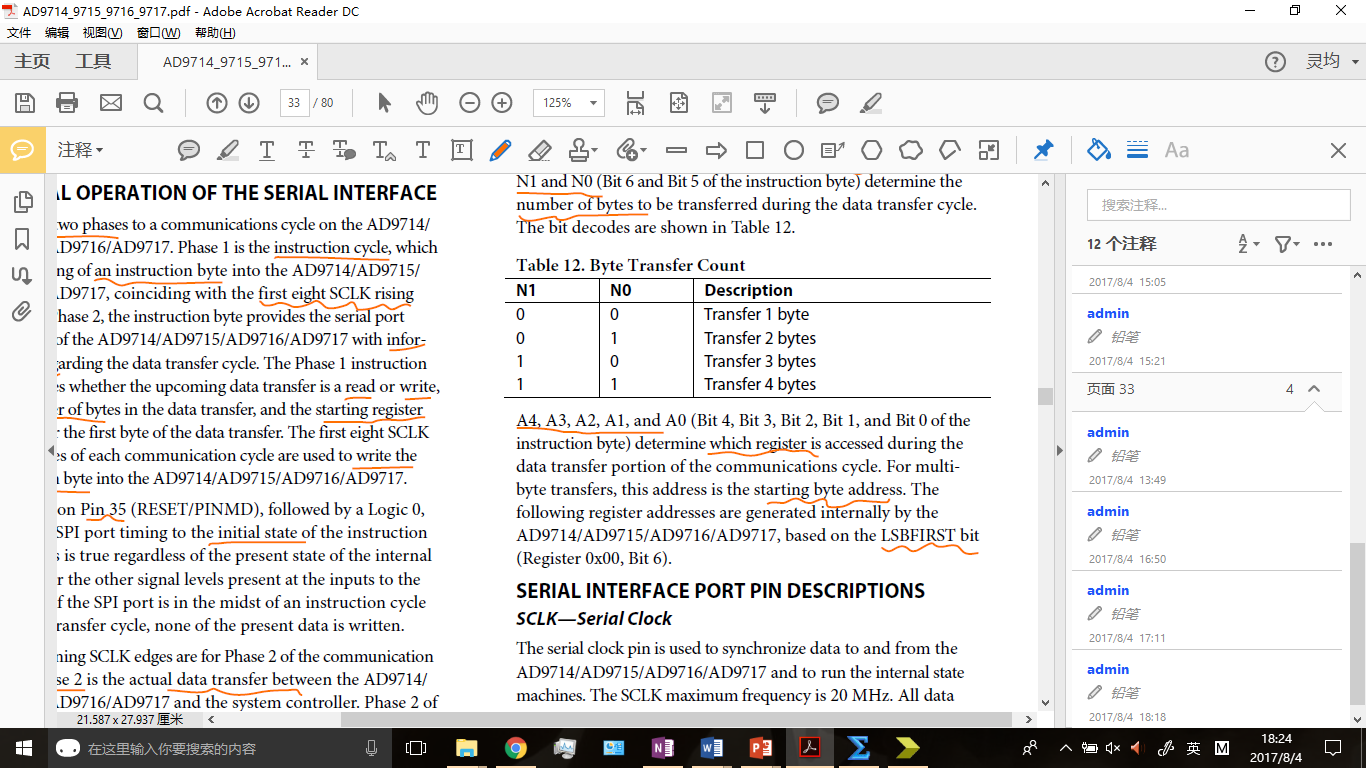
#### 信息传输阶段

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：



其中字节数控制位的具体含义如下所示：



当传输多个字节时，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

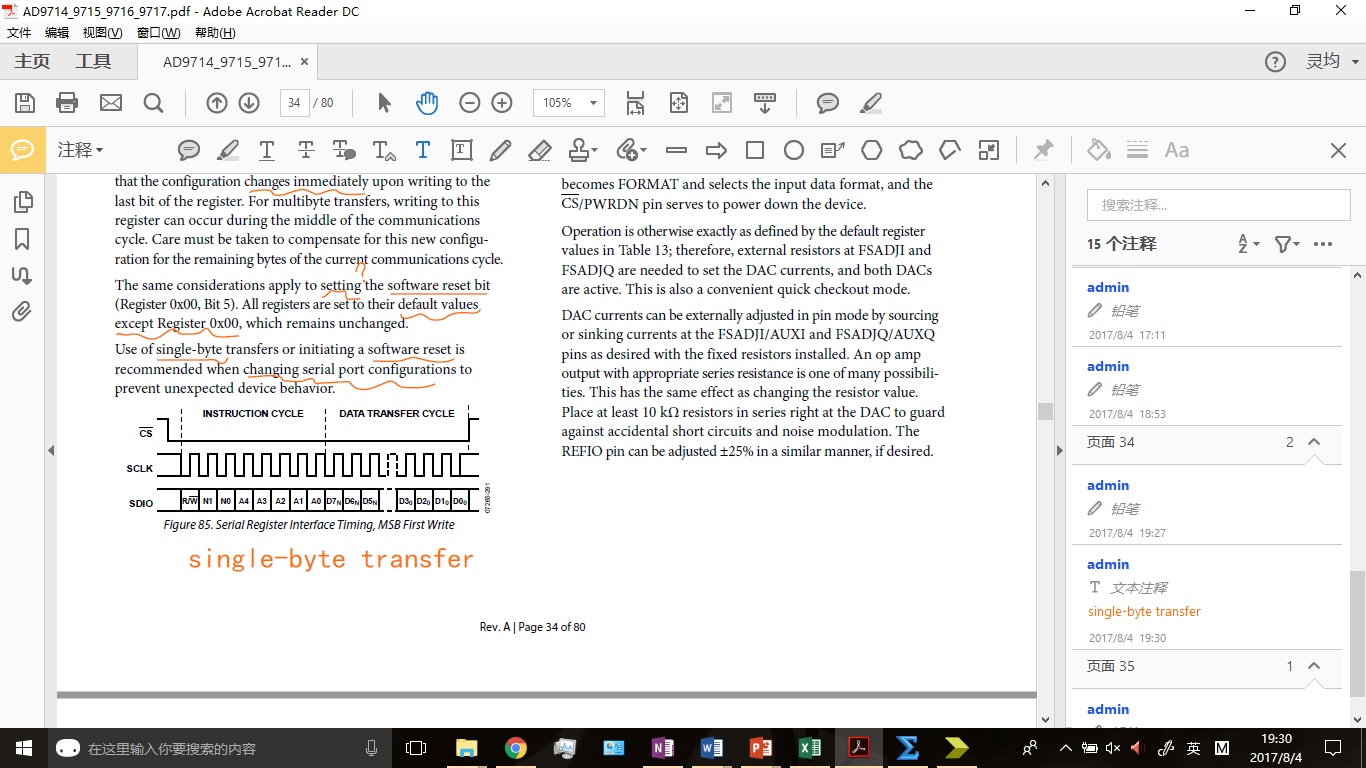
|  |  |  |
| --- | --- | --- |
| 信号 | 作用 | 描述 |
| SCLK | 串行时钟 | 用于同步数据，最高20MHz。输入数据上升沿有效，输出数据下降沿有效 |
|  | 片选信号 | 低电平选中，开始一个通信循环，在整个通信循环中需要保持为低电平 |
| SDIO | 串行数据I/O | 双向数据收发口 |

### 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（？）。

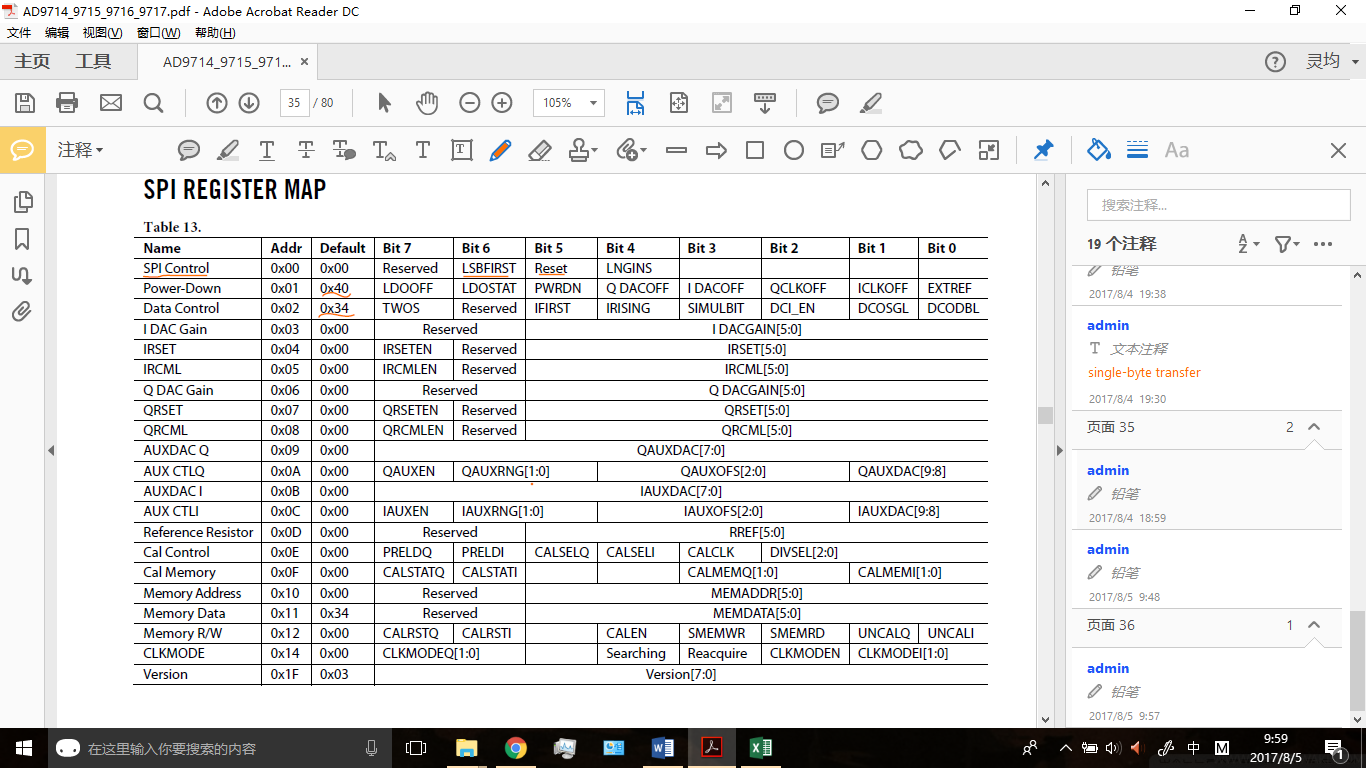
|  |  |
| --- | --- |
| 引脚 | 功能 |
| RESET/PINMD | 保持高，进入引脚模式 |
| SCLK/CLKMD | 提供时钟模式控制 |
| SDIO/FORMAT | 选择输入数据格式 |
| /PWRDN | 控制掉电 |

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

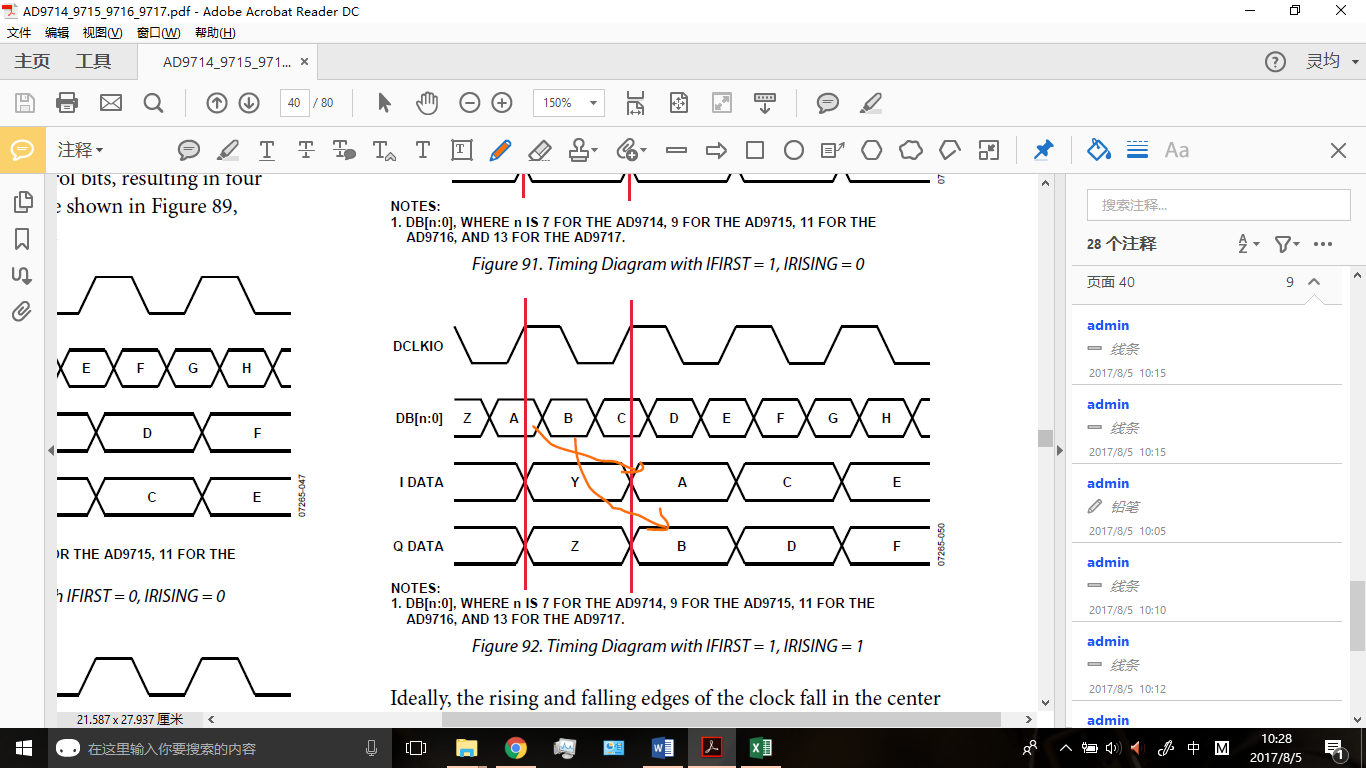
完整的寄存器格式如下图所示：



## 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。



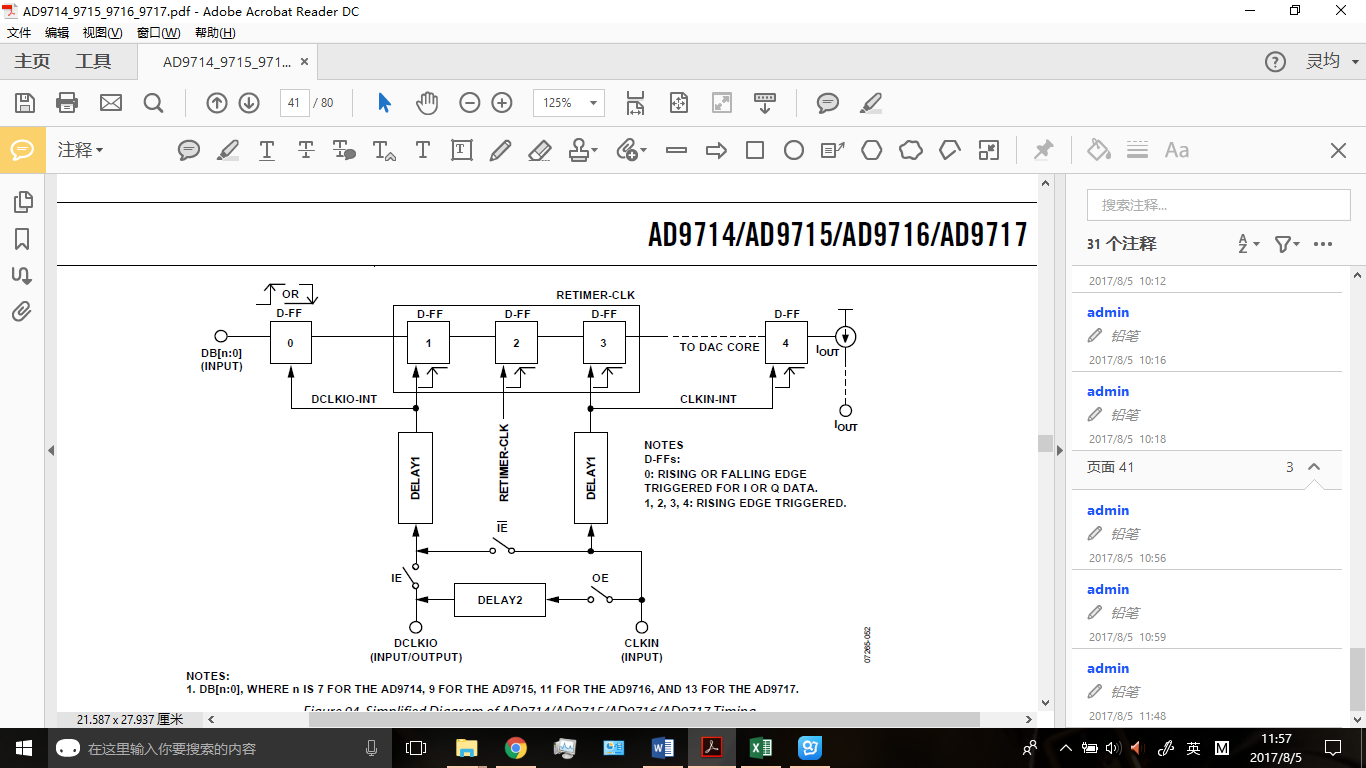
在数据对中I DAC的数据优先（Q DAC的数据滞后），在DCLKIO的上升沿进行 I DAC的数据锁存（在DCLKIO的下降沿进行Q DAC的数据锁存）。

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。

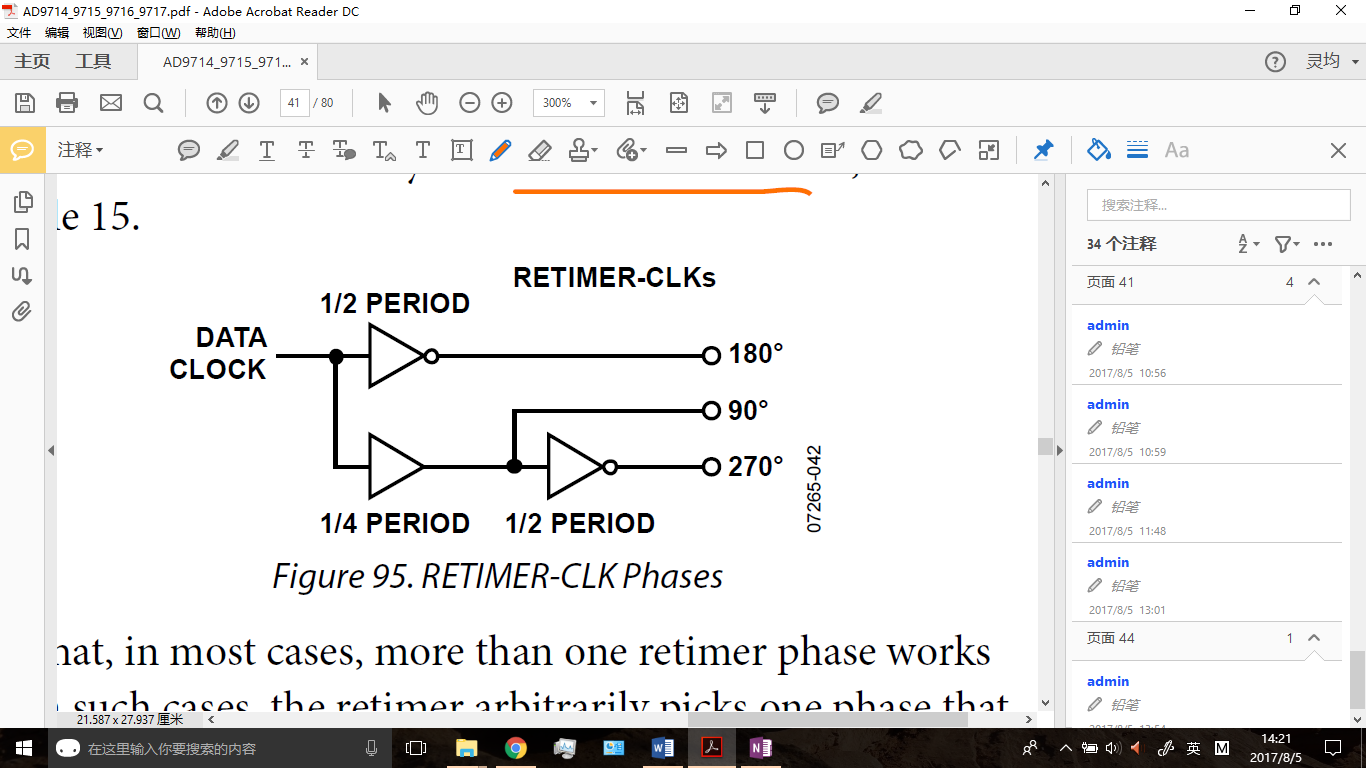
在默认的引脚模式和SPI设置中，IE为高电平（关闭）OE为低电平（打开）。这种设置在RESET/PINMODE（管脚35）被置为高电平时生效。在这种模式下，用户需要同时提供DCLKIO和CLKIN。在引脚模式下，也推荐DCLKIO和CLKIN采用相同的相位，使得DAC能够正常工作。用户也可以访问SPI，将控制位DCI\_EN（寄存器0x02，Bit 2）置低，使得CLKIN也用作DCLKIO。

DCOSGL和DCODBL可以控制CLKIN从DCLKIO输出。

#### 重定时器

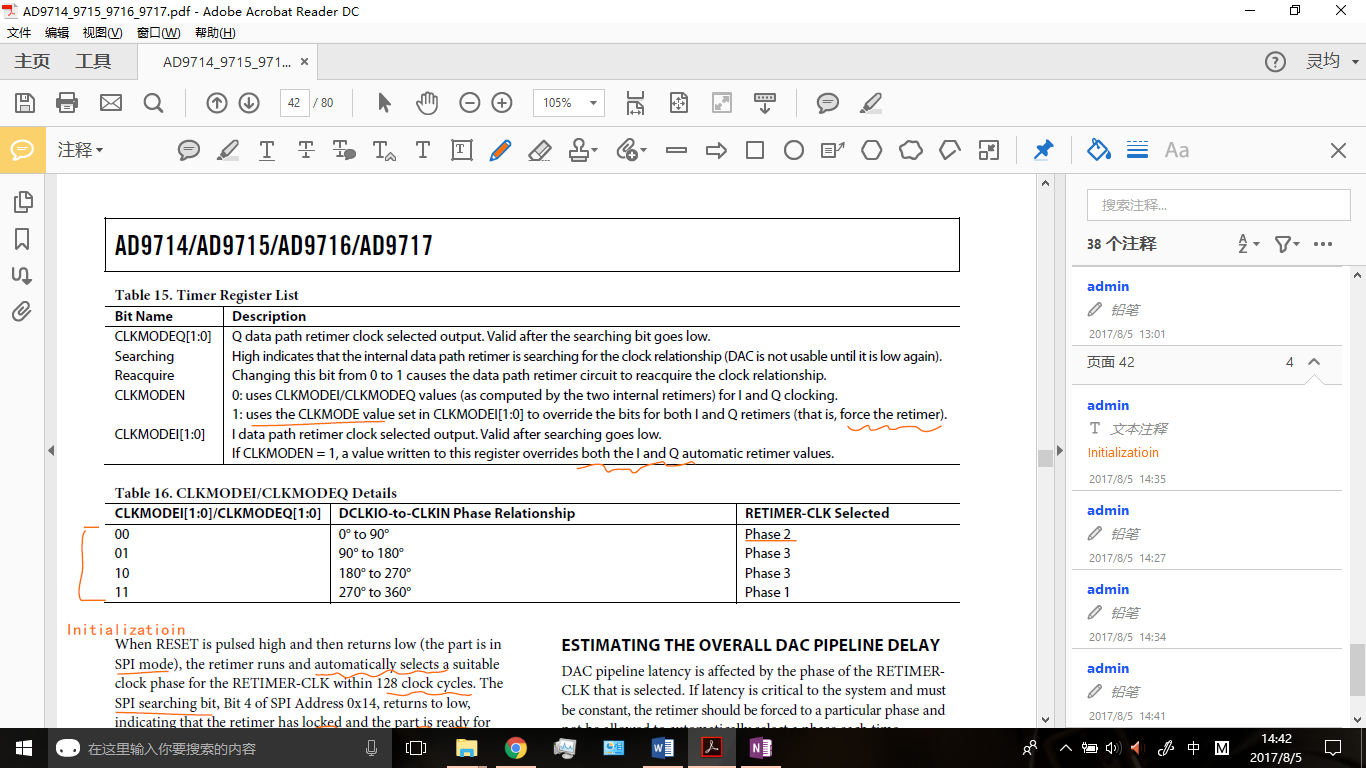
AD9715有内部重定时电路，可以比较CLKIN-INT时钟和DCLKIO-INT时钟，并基于它们的相位关系选择一个重定时时钟，从而实现从用于片上输入接口的DCLKIO时钟到用于DAC模拟核心的CLKIN时钟的安全转换。

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：



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。

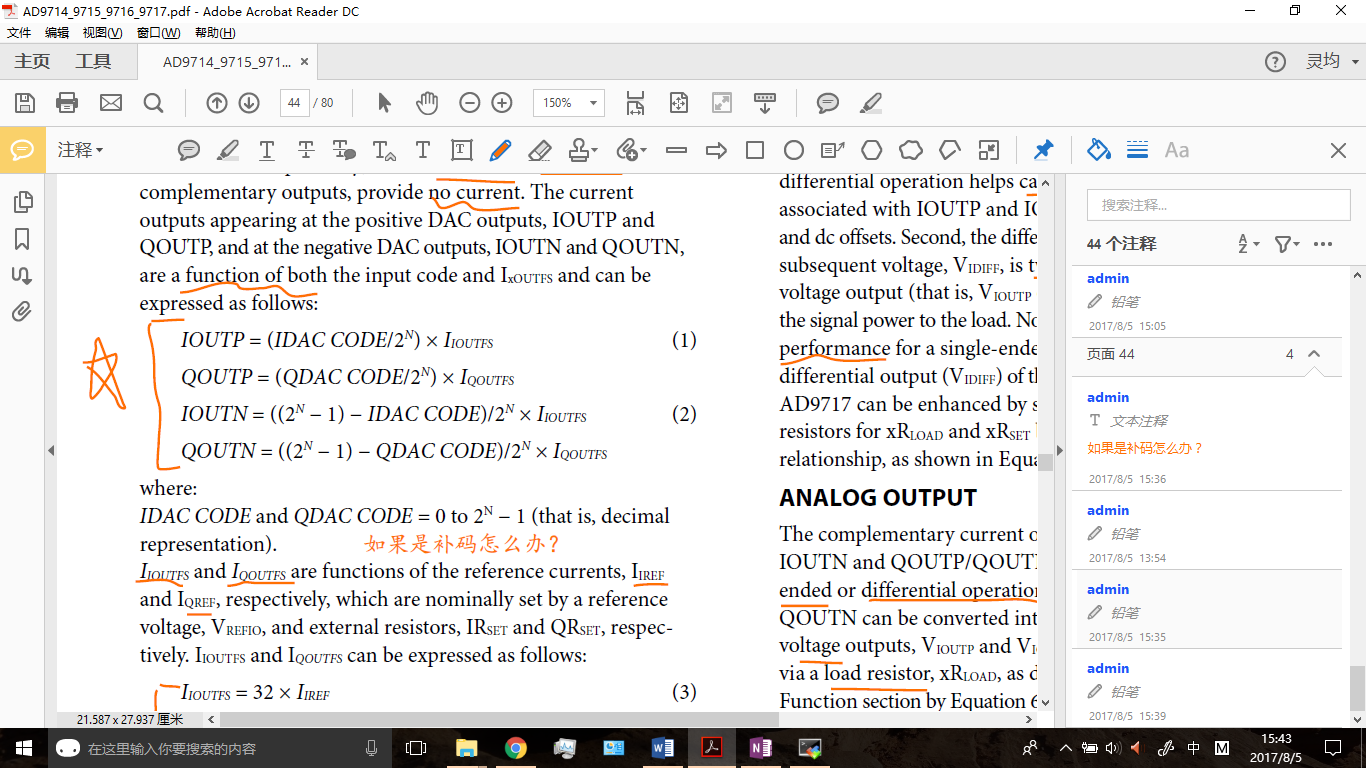
引脚模式下，用户最好把CLKIN和DCLKIO连接到一起。不过器件也通过复用SPI引脚提供了一些可编程的功能。略。

### 2.6.2 总DAC流水线延时的估计

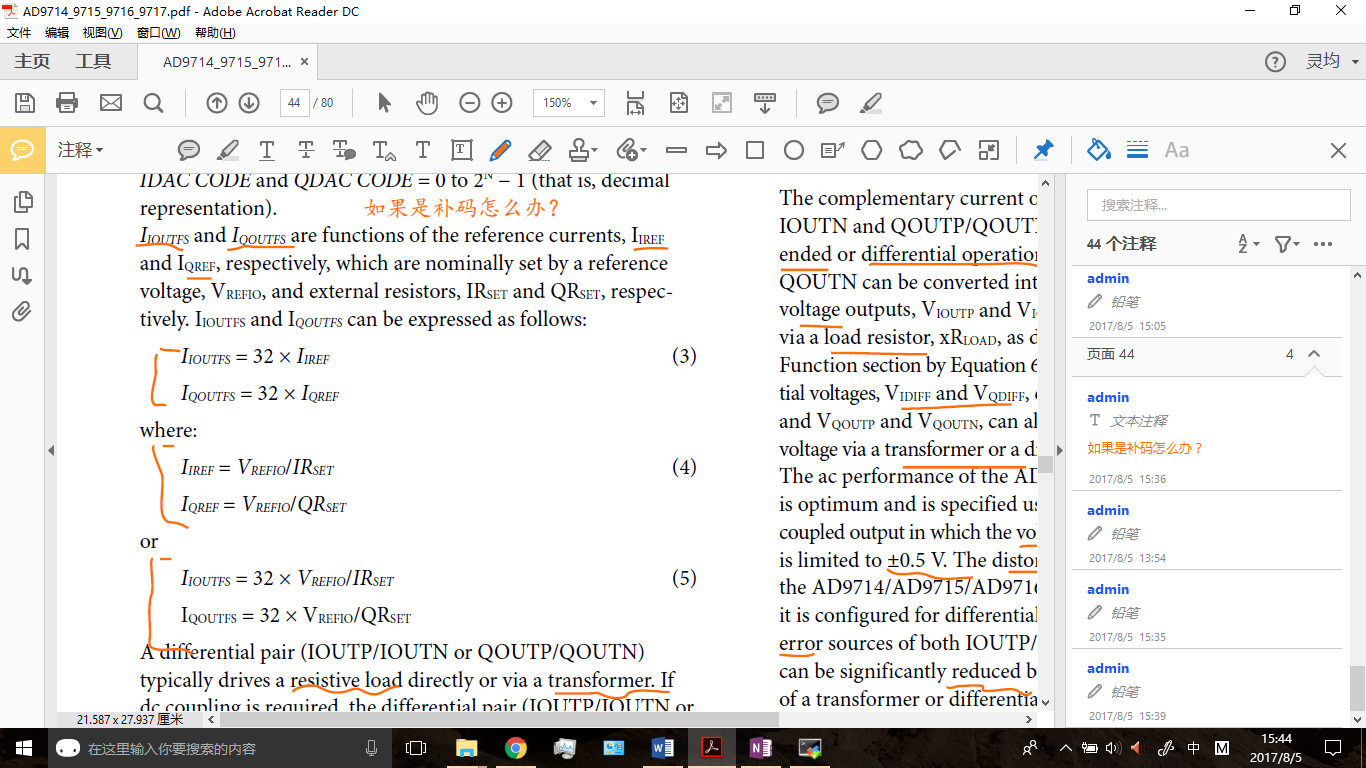
略。

### 2.6.3 DAC传输函数

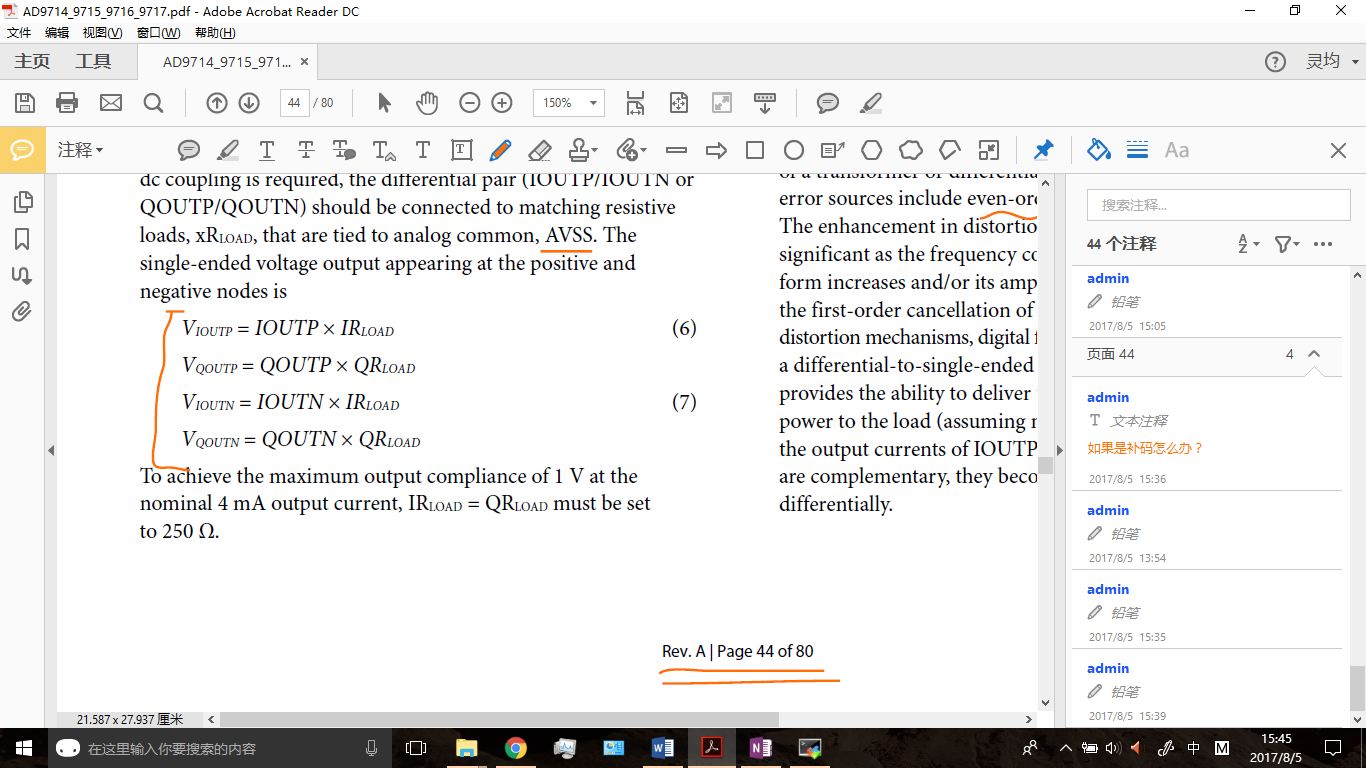
DAC输出电流与输入代码的关系如下：



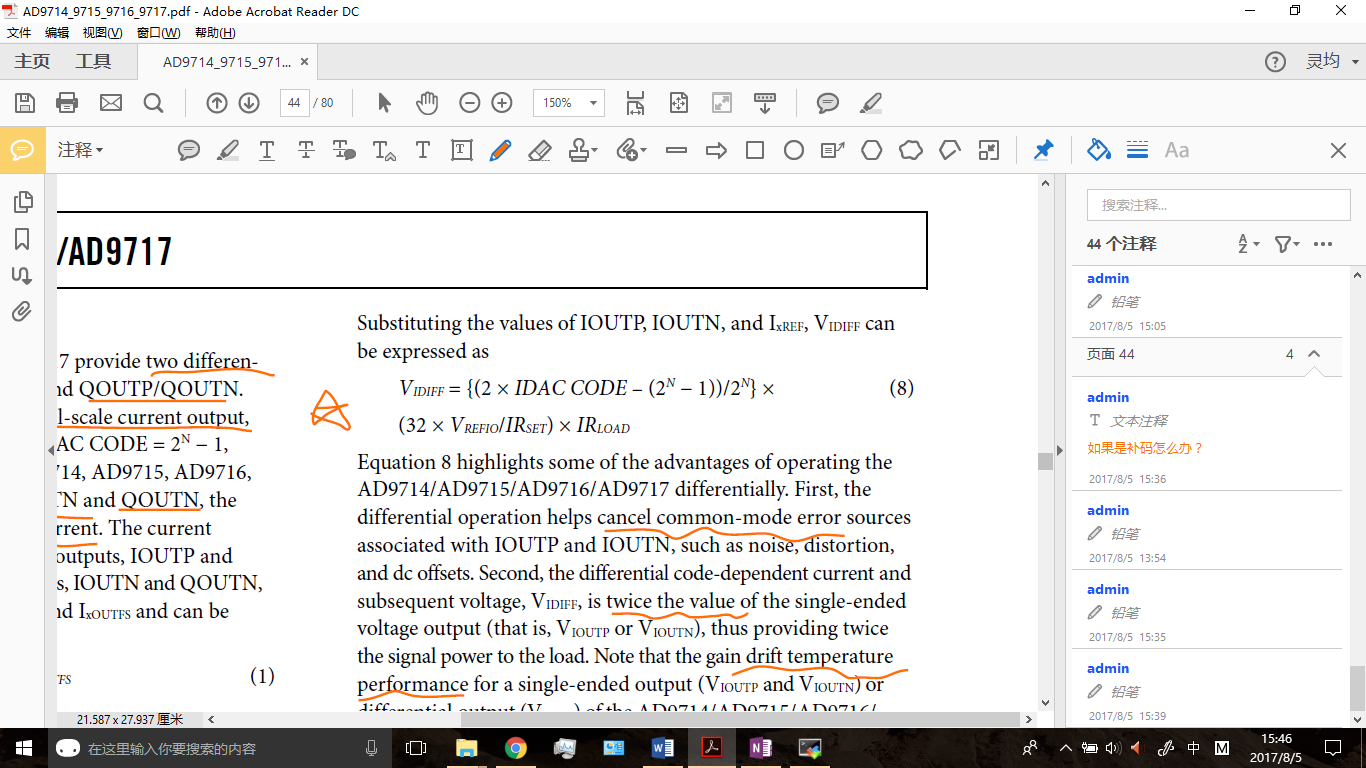
满标电流与参考电压、电阻的关系如下：



单端输出电压与输出电流的关系如下：



差分输出电压的表达式如下：



## 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 |
| /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

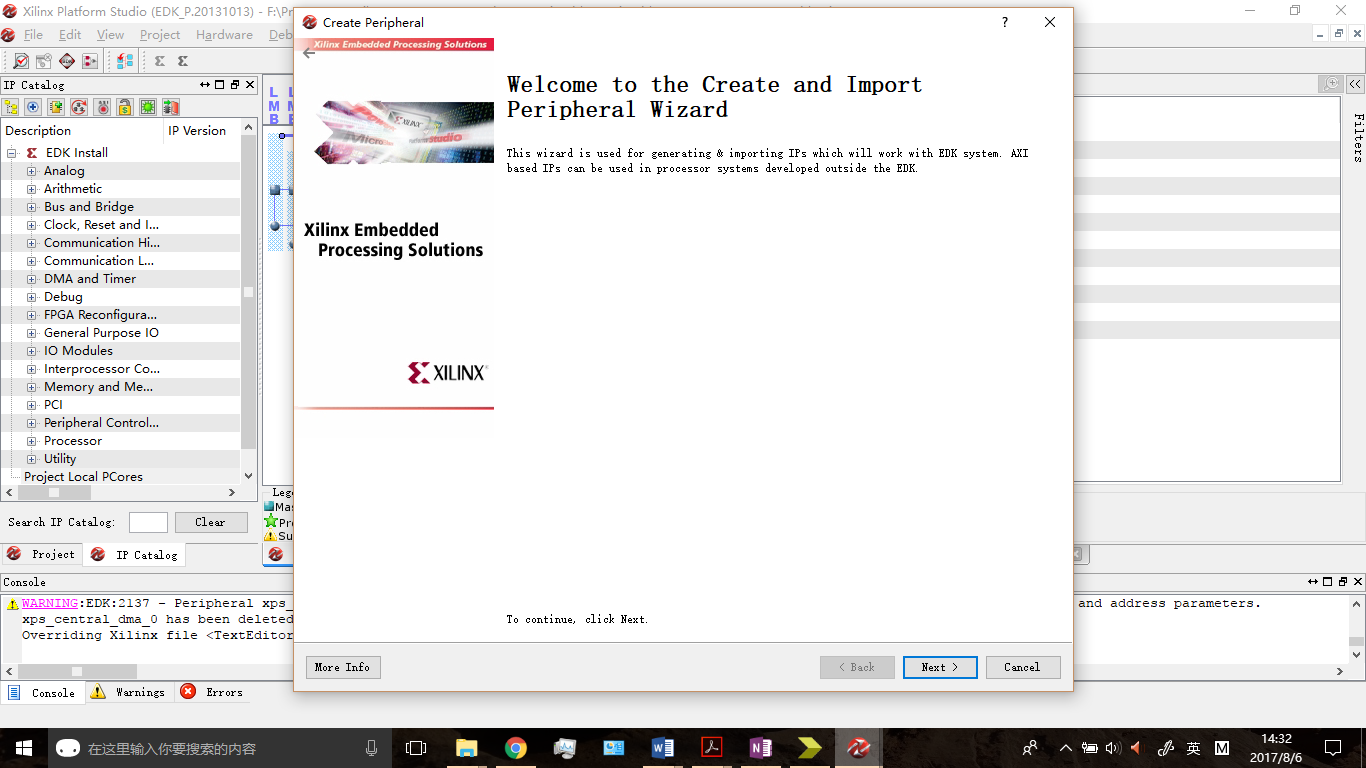
## 3.1 固件设计的设想

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。

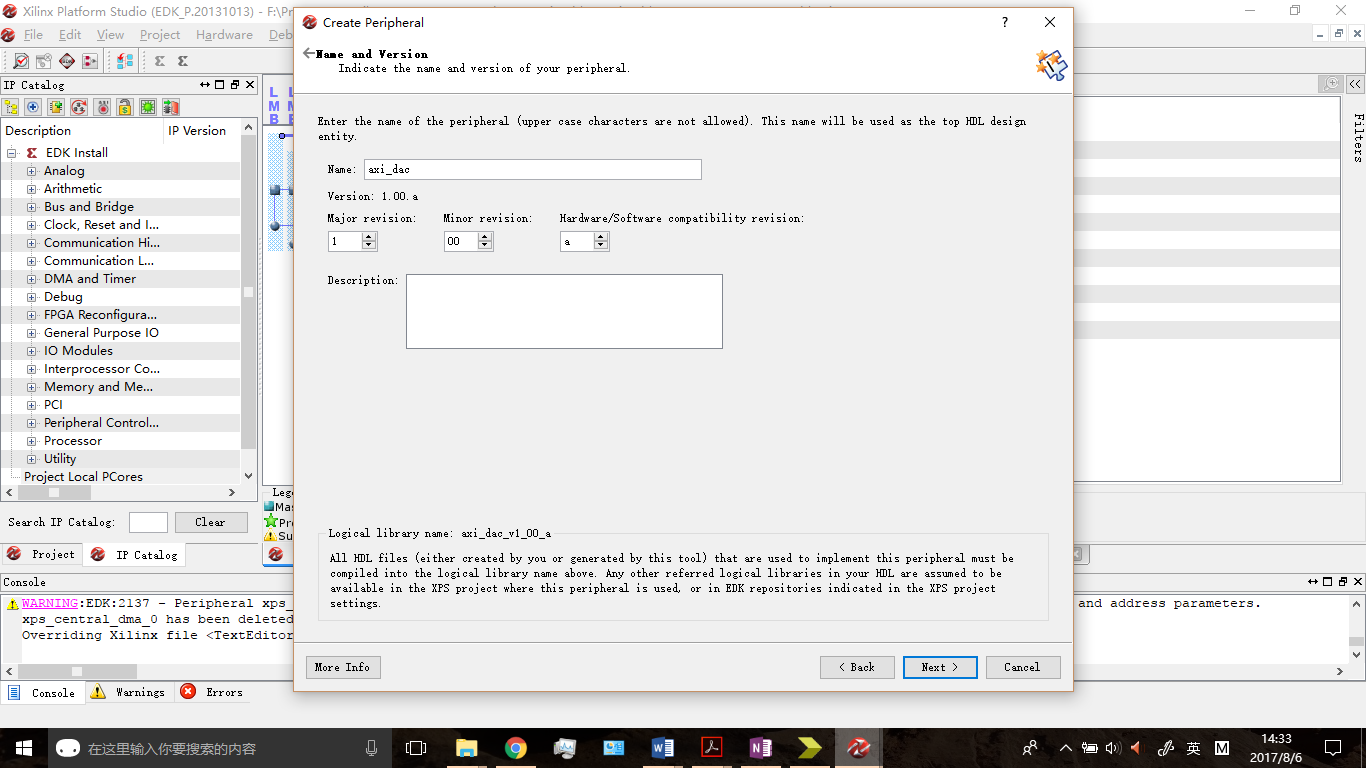
简单的行为级描述如下：MicroBlaze核向固件的数据寄存器中写一个数据，固件就将这个数据传递给AD9715，产生相应的模拟信号。

## 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。



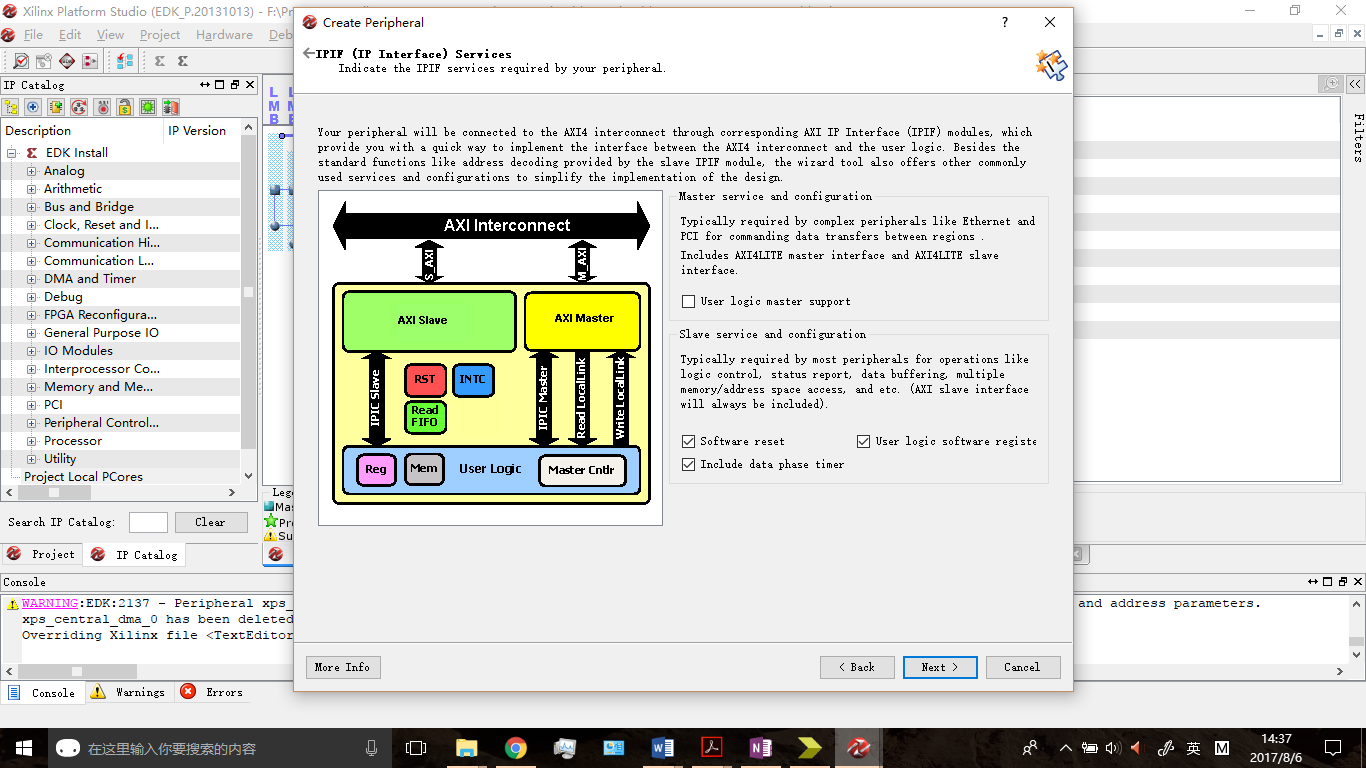
尝试创建一个名为axi\_dac的外设。



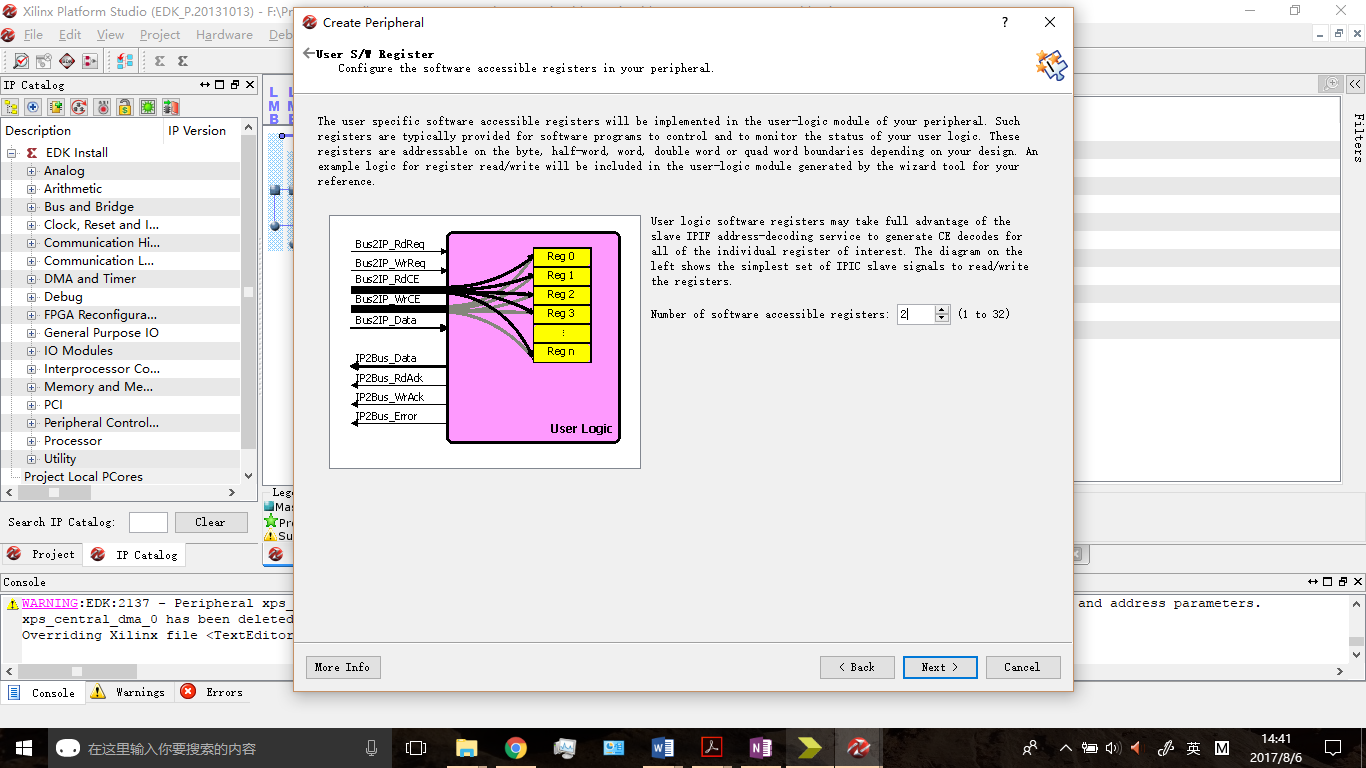
选择较为简单的AXI4-Lite总线。



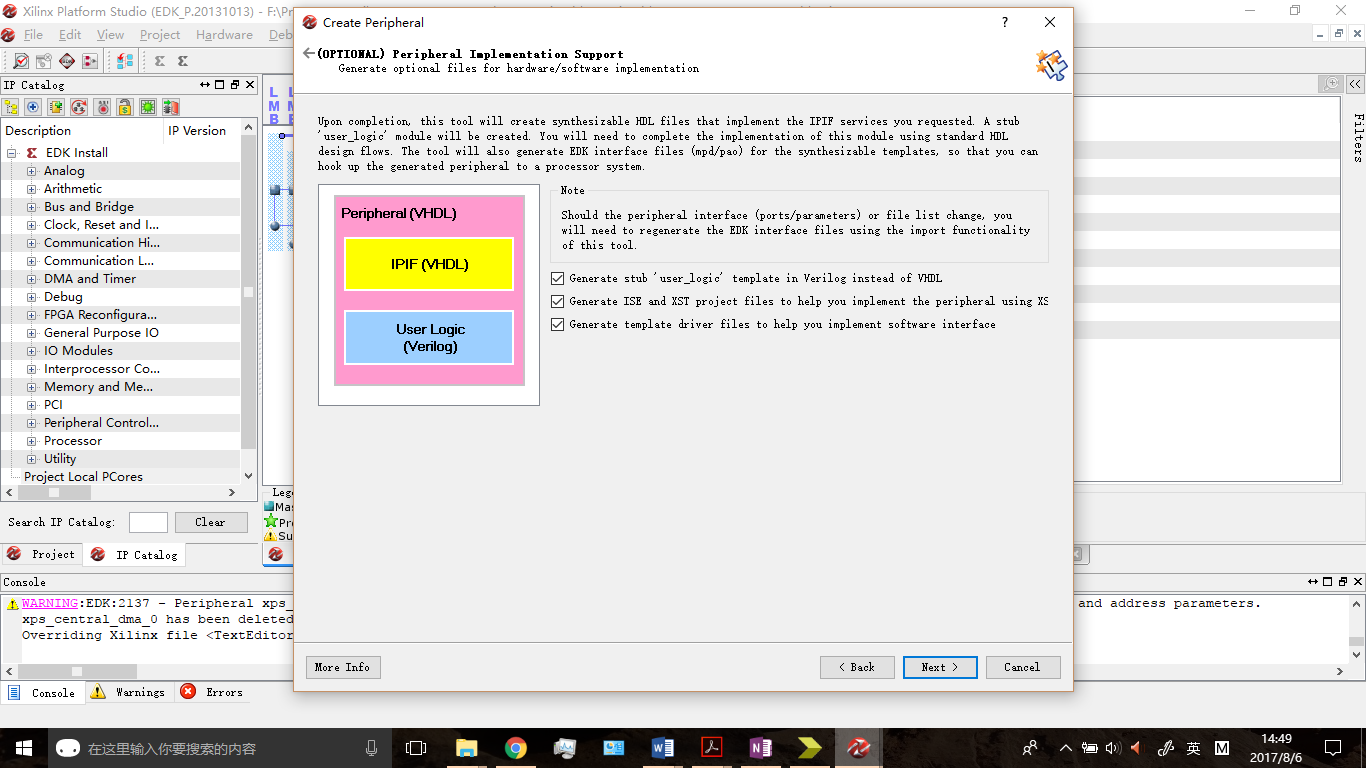
启用软件复位、逻辑软件寄存器和数据相位时钟。



用2个寄存器实现用户对接口的控制。



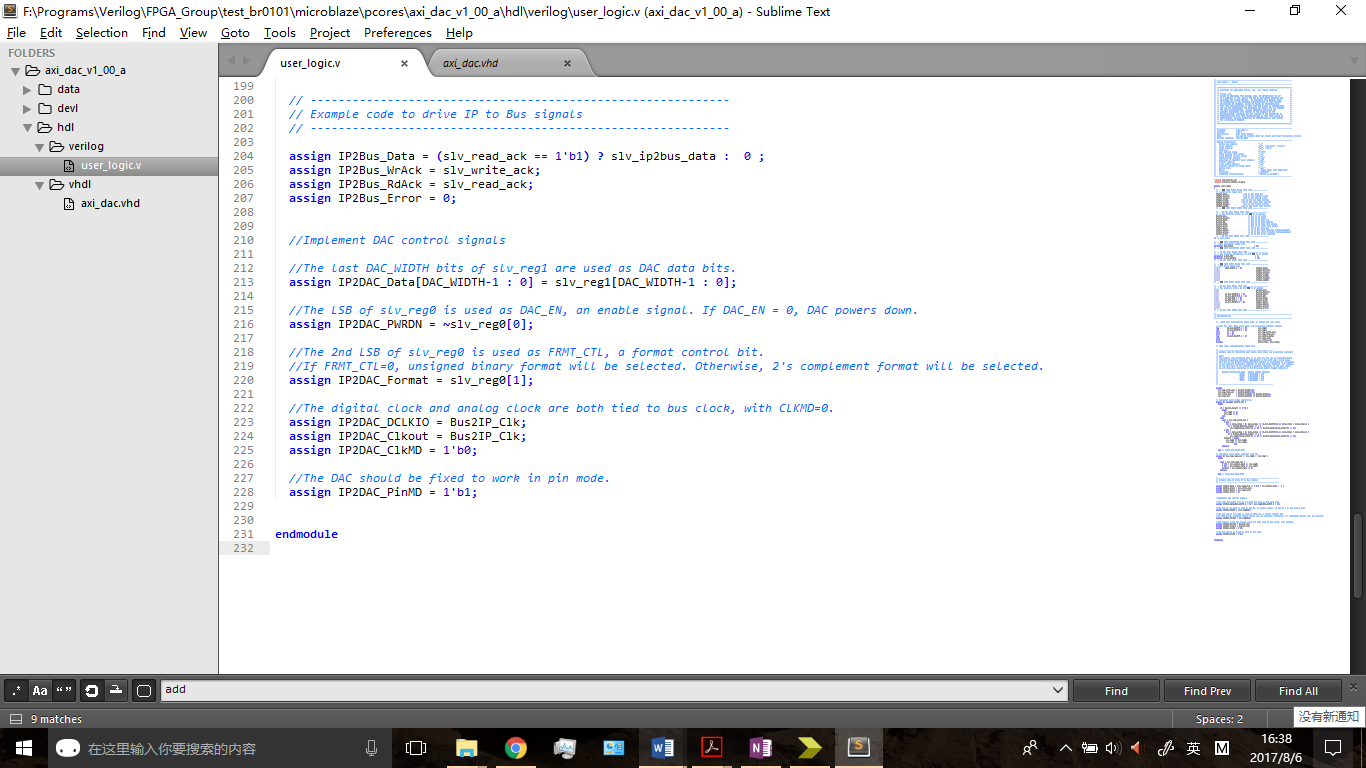
采用Verilog描述用户逻辑，生成项目和驱动文件。



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：



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：

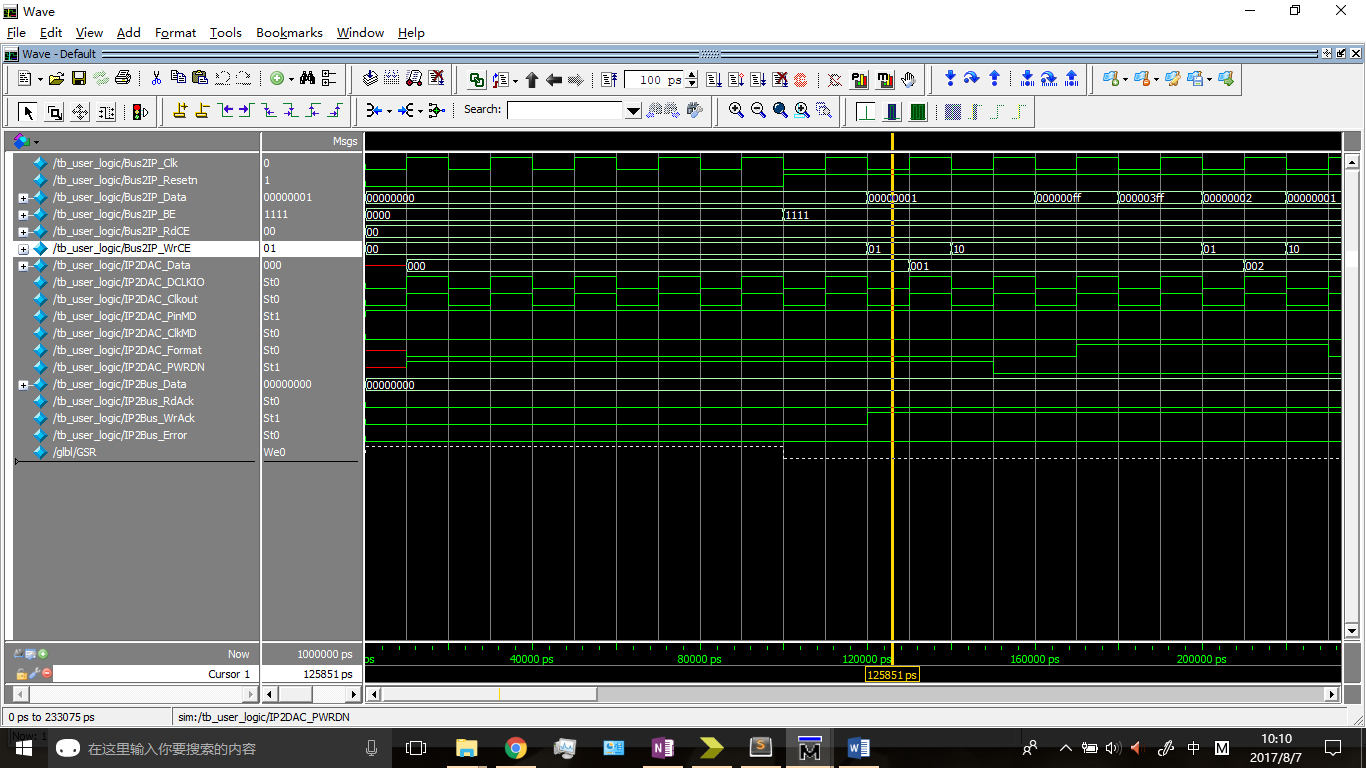


## 3.4 DAC接口固件的仿真

### 3.4.1 用户逻辑（user\_logic）部分单独仿真

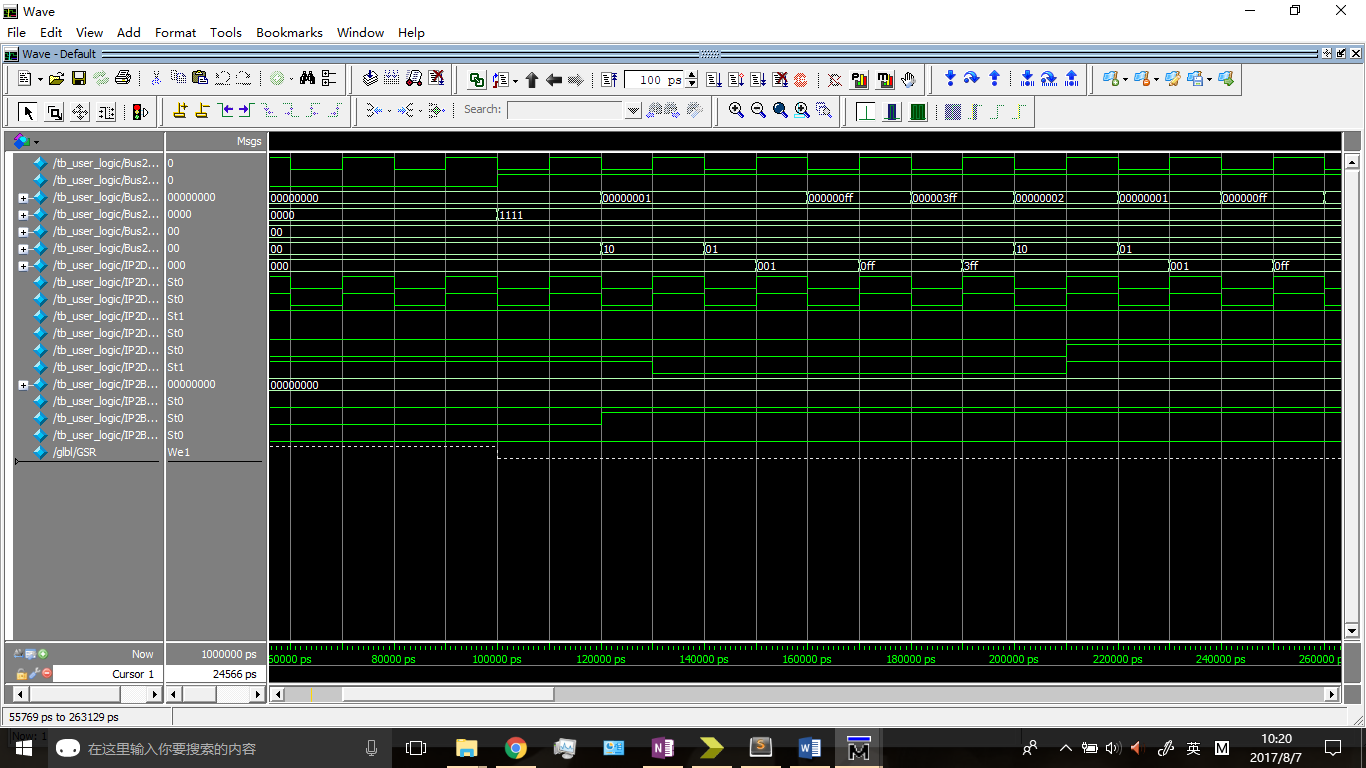
先对用户逻辑（user\_logic）部分进行单独仿真，测试寄存器读写和DAC控制信号产生功能是否正确。因为不涉及到太多总线信号，所以testbench写起来较为容易。

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

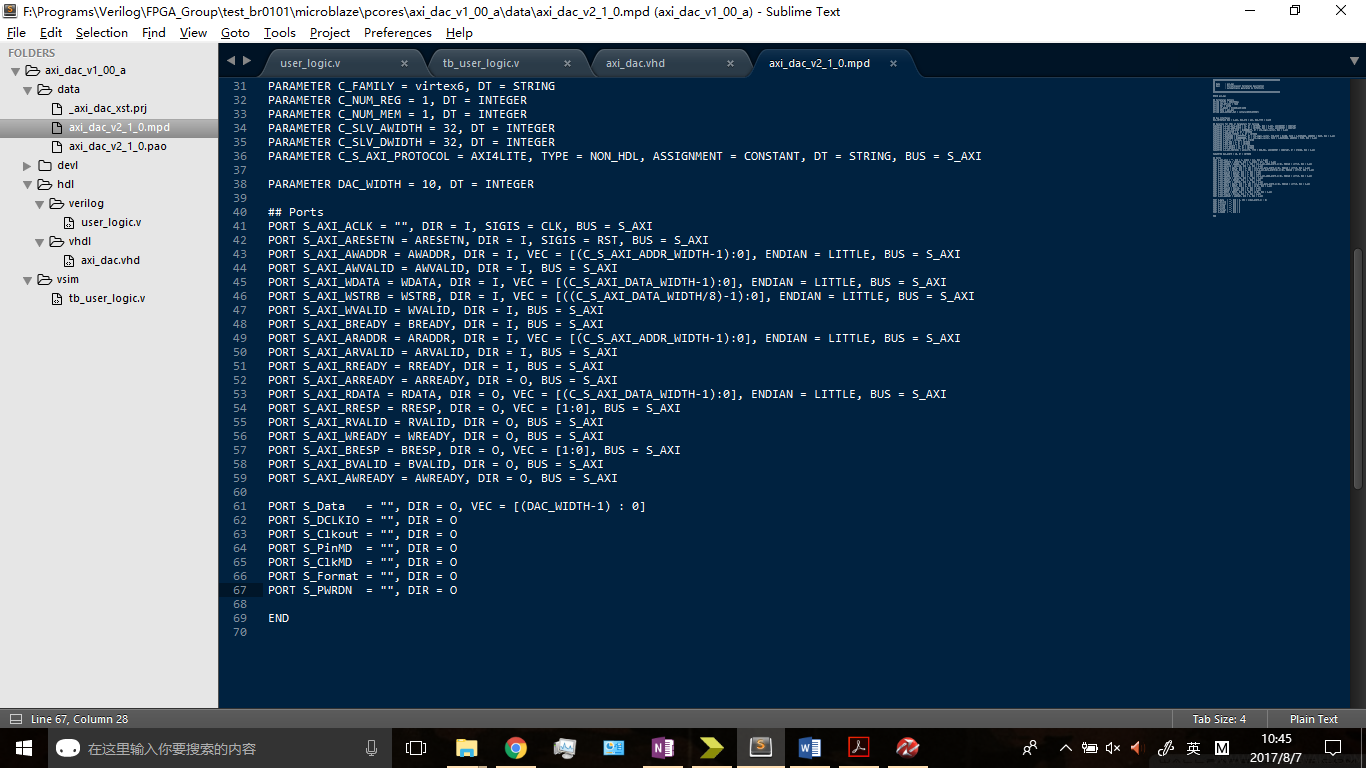
在将固件添加到MicroBlaze系统中之后，还需要用二进制激励文件对它进行仿真。这个步骤放到后面的章节中。

## 3.5 DAC接口固件的添加

### 3.5.1 接口固件输出端口的声明

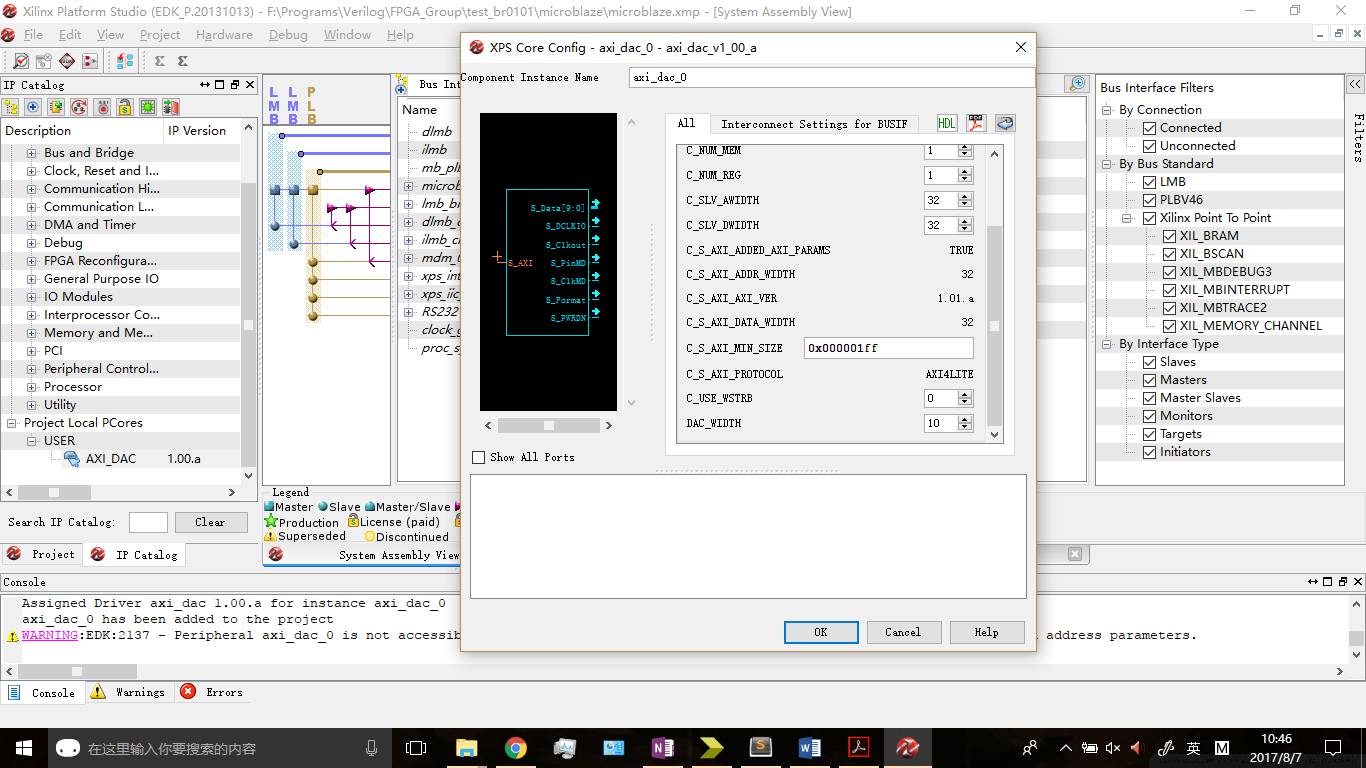
因为这个固件需要通过几个输出端口发出信号控制片外的DAC，所以好像还需要在外设的说明文件中添加上这些端口的信息。

在axi\_dac\_v2\_1\_0.mpd中，添加如下信息：

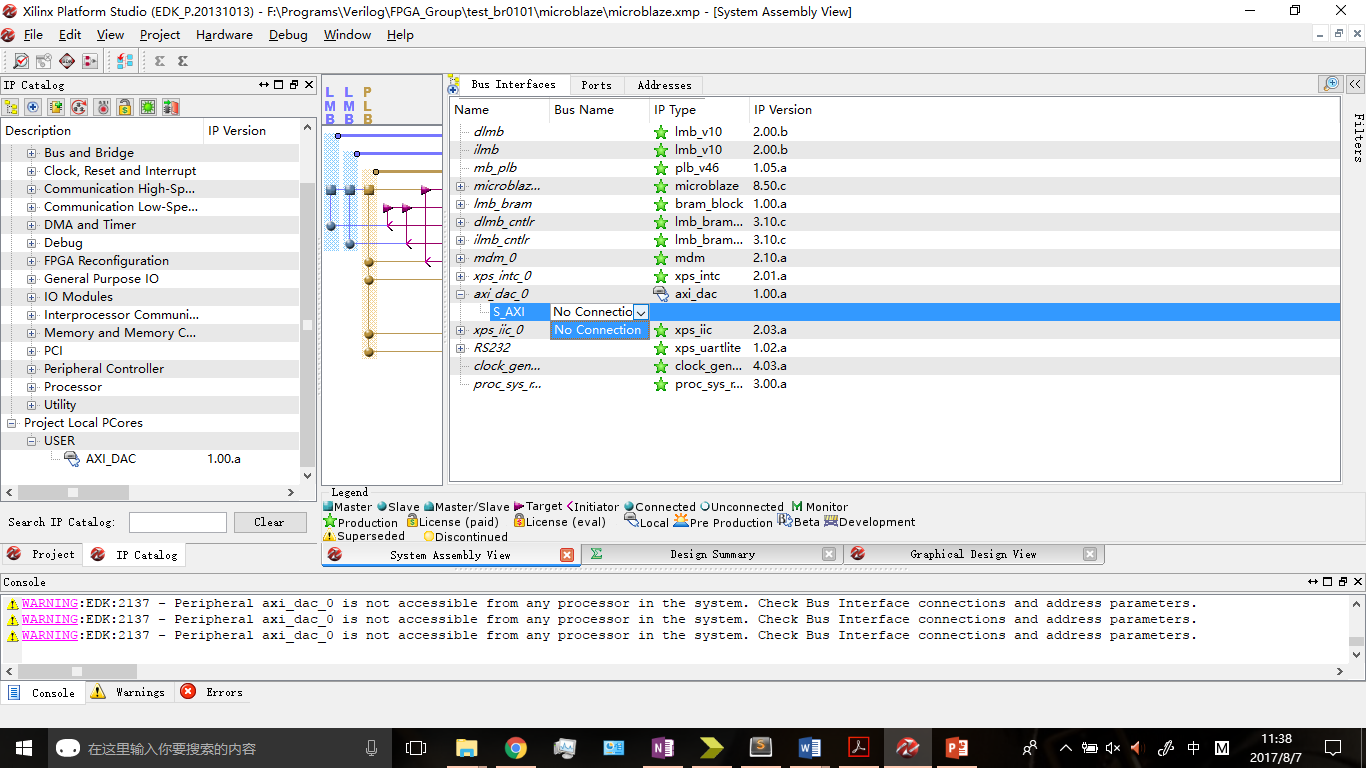


### 3.5.2 将IP核添加到系统中

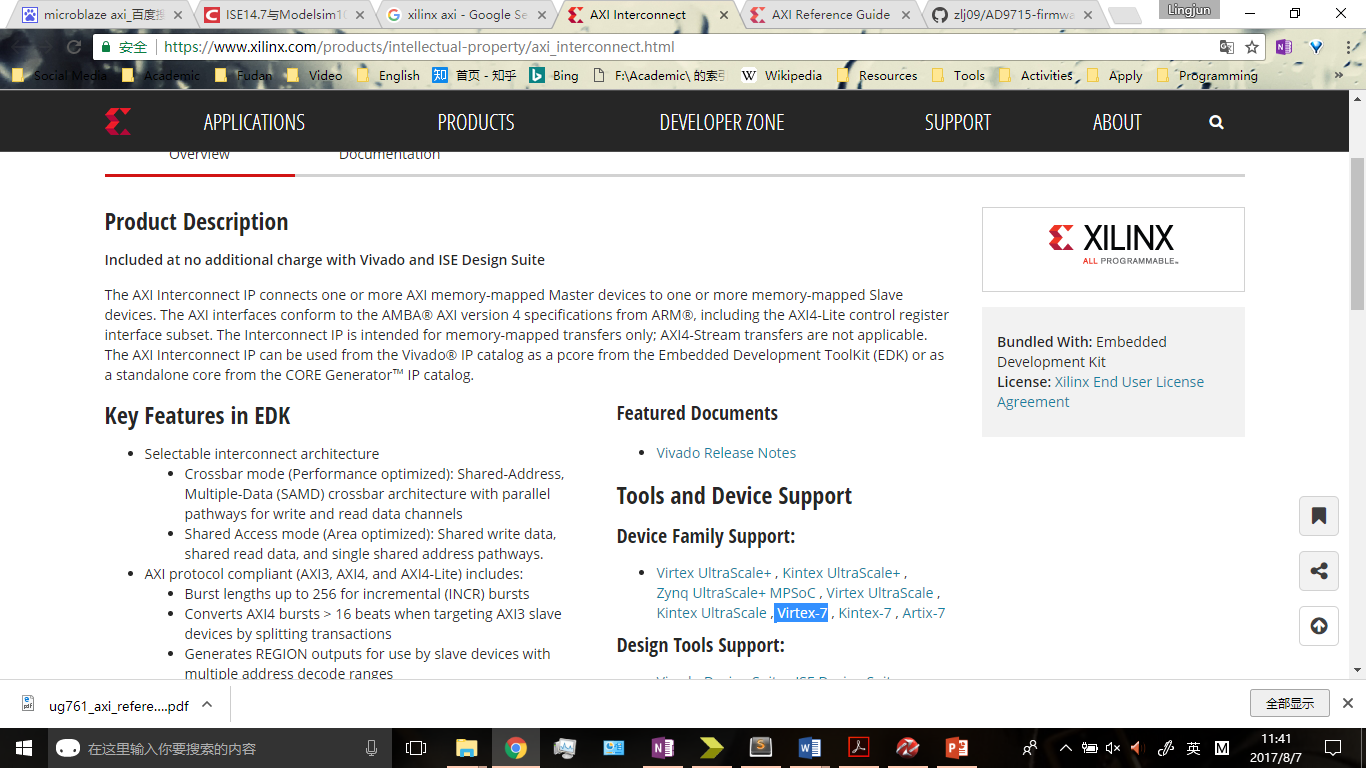
在XPS中重新扫描IP核之后，尝试将aix\_dac添加到系统中。



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



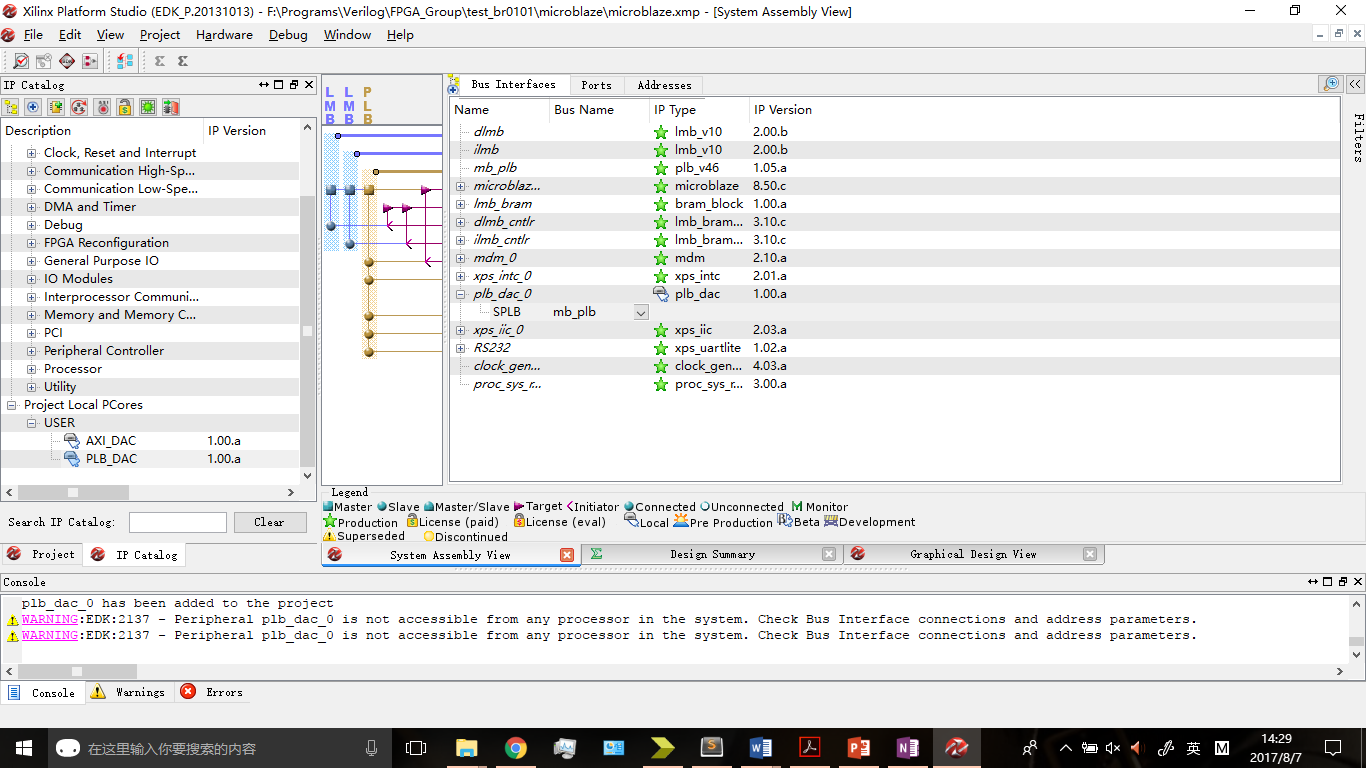
查阅官网并咨询王星学长后才发现，Vertex-4不支持AXI总线。



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。

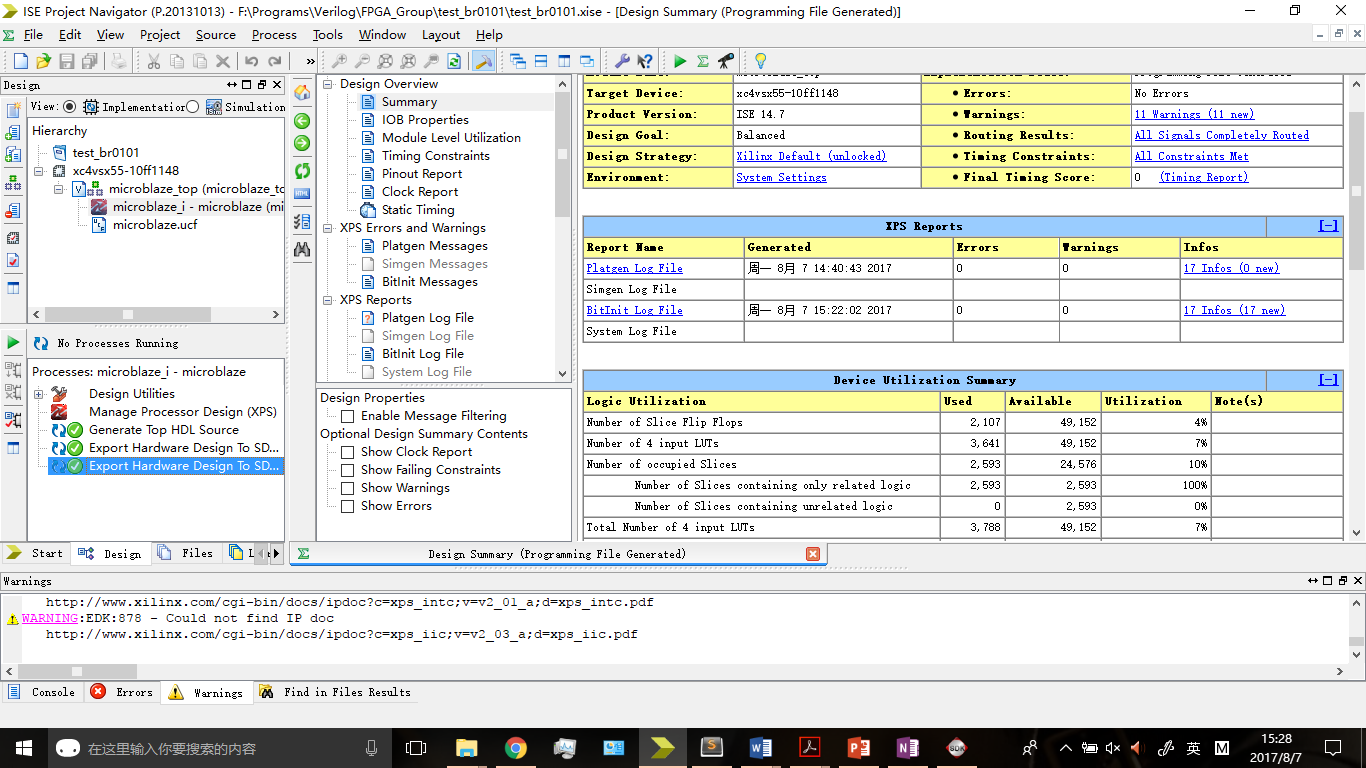
最终可以将plb\_dac添加到系统中，并连接到PLB总线上。



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。

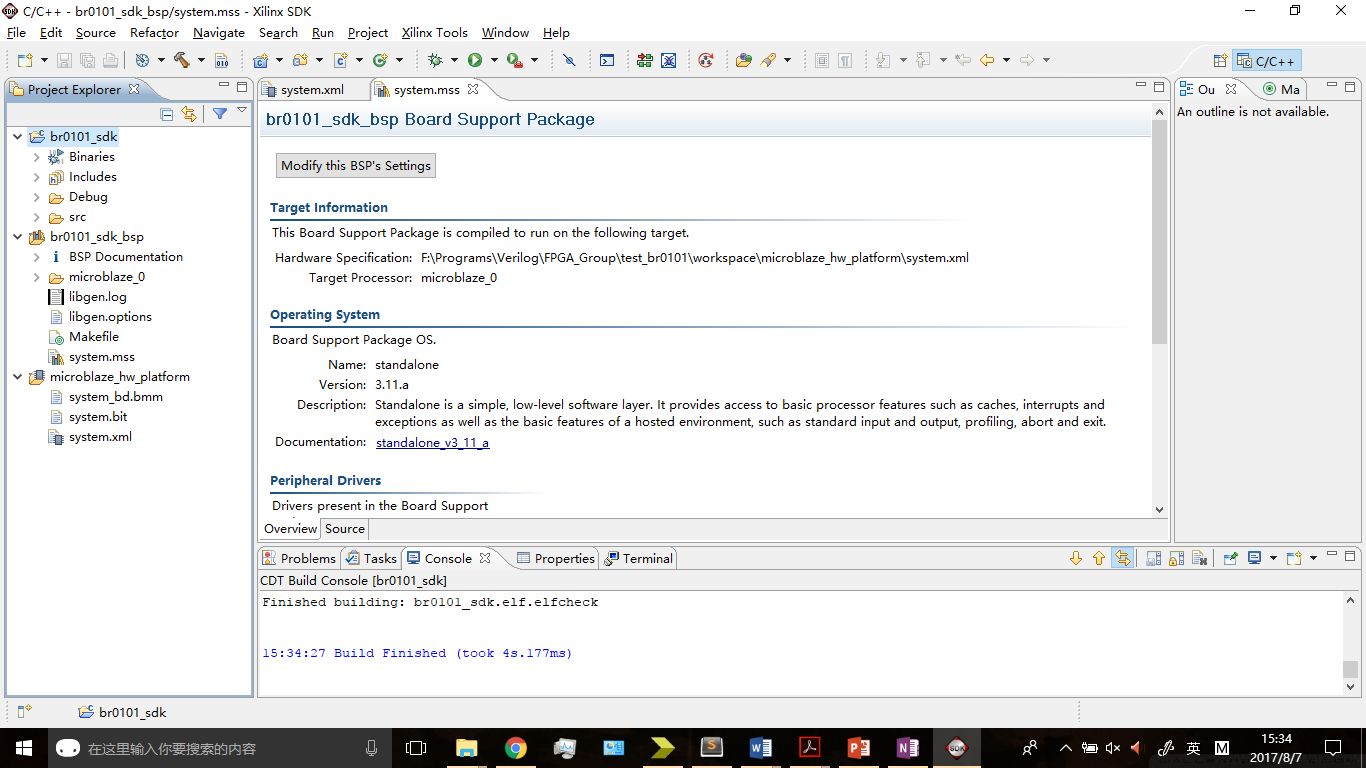
### 3.5.3 综合、生成位流并导出

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。



## 3.6 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。

