BR0101固件开发——模数转换器AD9715

（第二周）

我们继续进行BR0101的固件开发，按照计划，进行AD9715固件的仿真和软硬件协同的调试。

# 1 MicroBlaze嵌入式系统的仿真调试

在完成了最初的固件开发尝试后，我们需要将自定义的固件挂载到MicroBlaze系统上，对系统进行整体的仿真和调试。调试流程大致如下图所示：

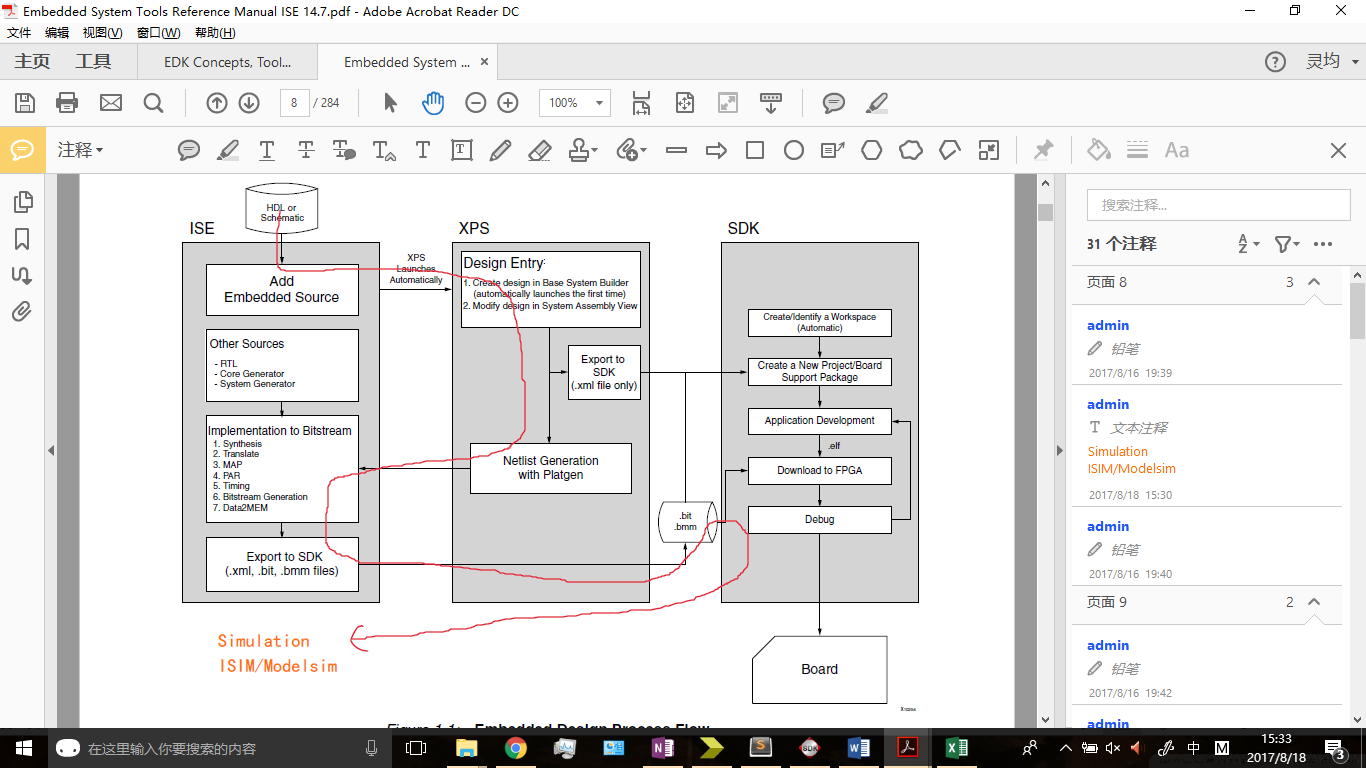


图 1 Xilinx 嵌入式系统设计、开发、测试流程[1]

## 仿真调试方法的调研

因为固件的仿真调试需要包含在MicroBlaze系统中，而且牵涉到软硬件协同调试的问题，和普通的FPGA应用仿真调试方法有所不同。我们针对这些方面进行了调研。

对于MicroBlaze以及其他嵌入式系统的测试和验证，Xilinx其实提供了两种方案：一种是硬件仿真验证，另一种是软件调试验证。具体区别如下：

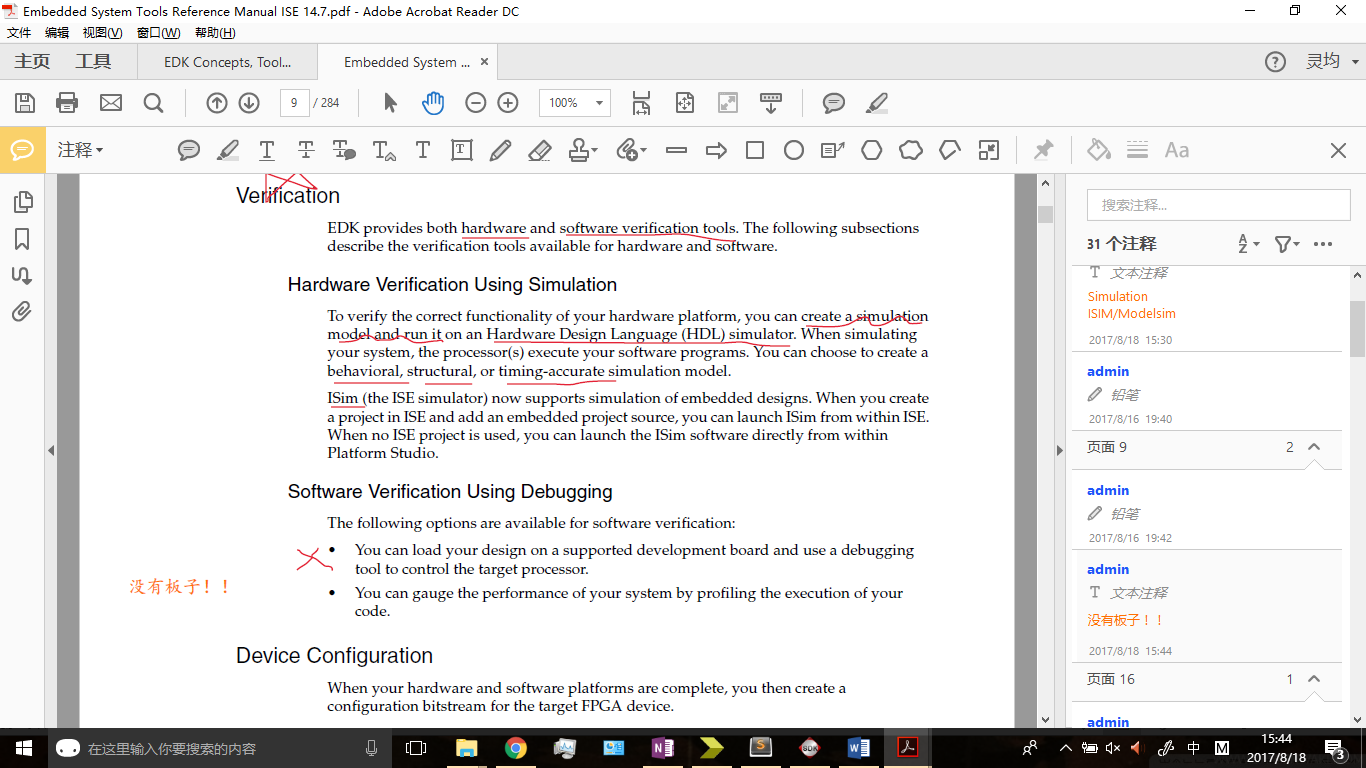


图 2 硬件仿真模型与软件调试模型的区别[1]

简单来说，硬件仿真验证还是采用传统的HDL仿真器，将MicroBlaze等软核作为硬件的一部分，将程序生成的可执行文件转换为BRAM中的信号，自动生成Testbench对系统进行测试和验证，类似传统的FPGA测试验证方法。软件调试验证则是以MicroBlaze软核为核心，以程序指令为单位，通过XMD或GDB等调试器控制程序执行，验证系统和程序的正确性，与嵌入式软件的测试验证方法相似。

此外，对于嵌入式系统的硬件仿真验证模型还可以分为三种：行为模型、结构模型和时序模型。其中，行为模型的生成方法如下所示：

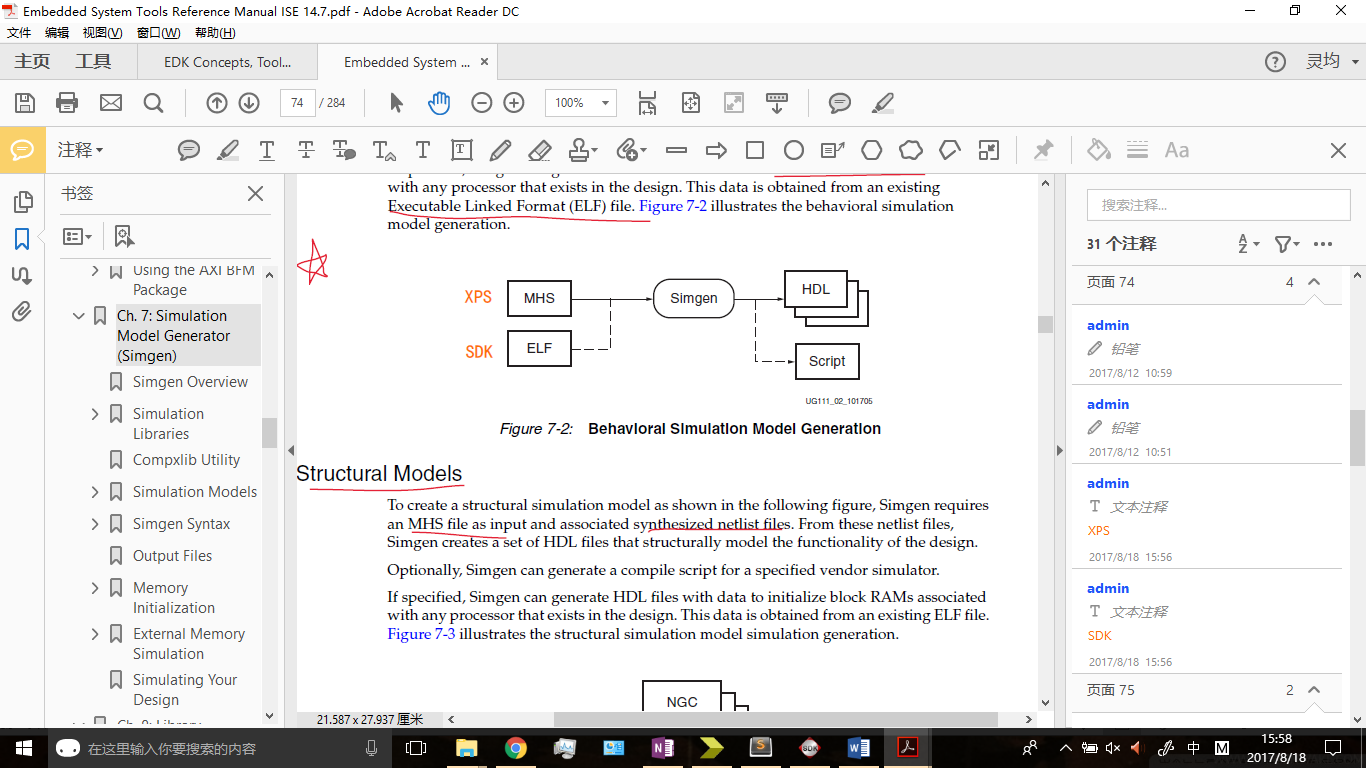


图 3 行为模型的生成[1]

XPS产生的微处理器硬件描述文件（MHS）包含了硬件平台的信息，SDK编译产生的可执行链接文件（ELF）包括软件部分的程序。Xilinx提供了Simgen工具来将这些文件综合起来生成仿真工具可以读取的HDL语言和脚本。行为模型可以验证软硬件部分基本功能的正确性，但不包含综合后信息和时序信息，不过可以基本满足我们的需求。

## 软件调试验证的尝试

我们先尝试了软件调试验证的方法。Xilinx提供了微处理器调试器（XMD）对软件程序进行调试。XMD有JTAG、串口、TCP Socket等多种接口，可以和实际的或远程的微处理器纪念性调试，但是由于我们暂时没有拿到开发板和芯片，只能通过GDB接口对仿真的处理器进行调试。

在SDK的菜单Xilinx Tools – XMD Console中可以打开XMD工具：

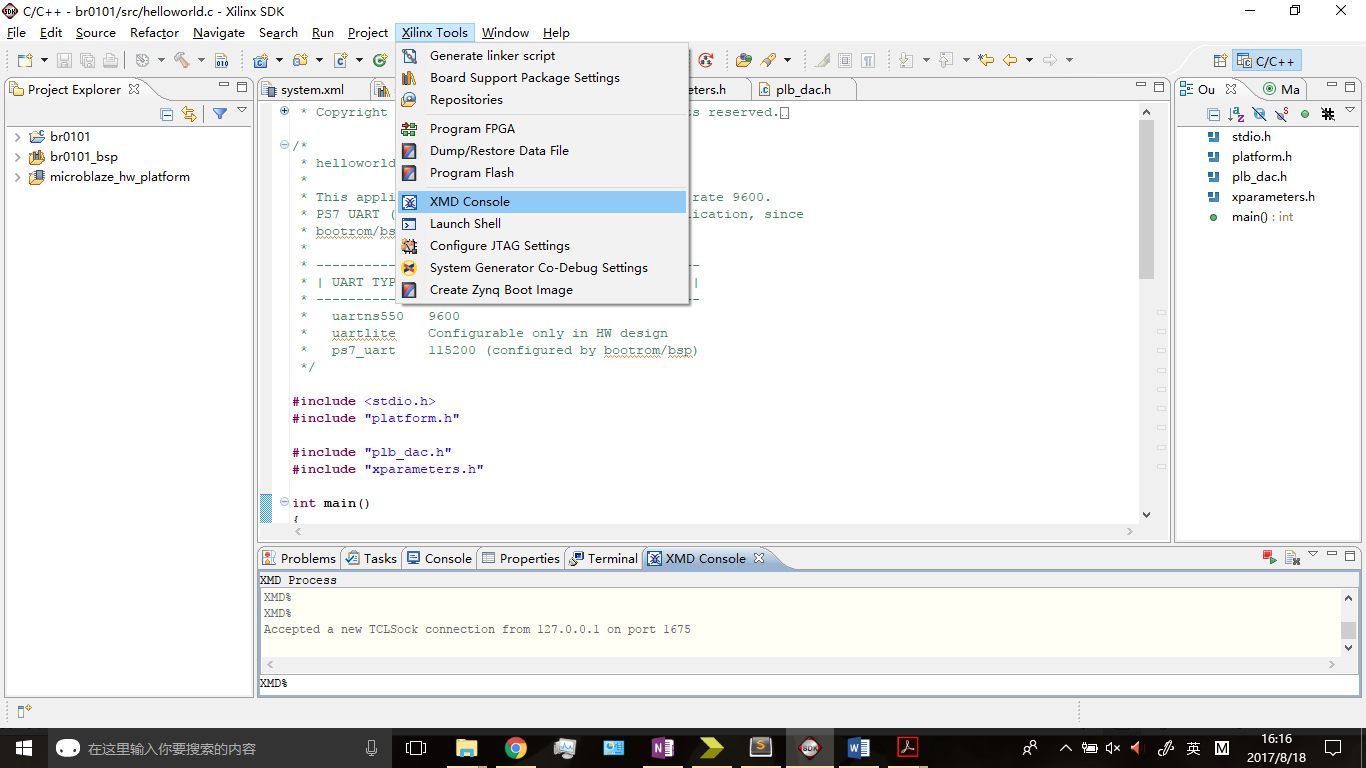


图 4 启动XMD调试器

输入下列命令可以连接到仿真的MicroBlaze处理器并下载位流：

connect mb sim

dow workspace/br0101/Debug/br0101.elf

还可以通过con、mrd、mwr等命令来控制指令执行和读写内存。但是读写内存的值经常不对，不知道是为什么。

XMD还可以和GDB联合进行调试。在XMD Console中找到ELF文件的位置，然后执行下列语句：

start mb-gdb br0101.elf

就可以启动一个GDB调试窗口：

o 
gäJ 
*-5LEt.äfit 
- C orrpl ement to certain set corm-rands 
43 
69 
l_mset - 
set 
set 
set 
set 
(gdb) 
Nichijou 
FPGA 
Further 
Developme.. 
Exchange 
Program 
FDUROP 
Ideas 
Games 
1 ne 
Digital Signal 
Processing 
• Computer 
Network I 
Japanese 1 
Political 
- Cancel environment variable VAR for the program 
environment - 
substi tute-path Llsage: l_mset substi tute-path [FROM] 
- Llnset target description specific variables 
tdesc 
tdesc filename Llnset the file to read for an target description 
#include <stdio. h) 
#include platform. h 
void print (char *str) 
int 
rn_unb er 
main ( ) 
init_platform() 
print ( 
return 
47 out 
Hello 
of range, 
46 
lines. 
2:41 
2:41 
2:41 
2:41 
2:41 
2:41 
2:41 
2:41 
2:41 
2:41 
2:41 
2:41 
2:41 
2:42 
2:41 
2:41 
.. /src/helloworld. c has 
wc .exe 
which.exe 
who.exe 
Windows8 OS 
LENOVO 
112 
whoa mi .exe 
2CC5/4/21 2:41 
2008/8/10 22:23 
2005/4/21 2:42 
2005/4/21 2:41 
2005/4/21 2:41 
go KB 
86 KB 
24 KB 
62 KB 
77 KB 
42 KB 
18 KB 
26 KB 
23 KB 
33 KB 
28 KB 
32 KB 
24 KB 
32 KB 
35 KB 
177 KB 
98 KB 
23 KB 
72 KB 
24 KB 
23 KB 
bin " 
2017/8/16 

图 5 XMD与GDB联合调试

在GDB中也可以通过list、stp、print等一系列指令控制程序的执行。

不过，软件调试方法还是比较适合有实际的处理器或者软件部分比较复杂的情况，对于我们固件开发的测试验证，我觉得还是硬件仿真验证方法比较重要。

## 硬件仿真验证的过程

硬件仿真验证的流程比较复杂，下面会做一些具体讨论。

### 硬件仿真验证流程的调研

Xilinx为嵌入式开发、测试和验证提供了一套完整的工具链，包括XPS和SDK两大部分，还有一系列小工具如Platgen、Simgen、XMD、Bitinit等。不过这些工具有的集成在XPS和SDK的流程中，有的需要从XPS和SDK的菜单里调用。

我们常用的Project Navigator开发环境实际上只是一个简单的界面，它提供了一套基本的流程和与XPS、SDK等工具之间的接口。实际上，它是用一系列命令（Tcl脚本？）和很多文件（HDL、MHS、ELF等）调用工具完成整套流程的衔接。不过这些工具之间有时候耦合得不太好，比如SDK重新编译并更新ELF之后，在Project Navigator中用ISIM进行仿真的时候并不会自动调用Simgen重新生成仿真模型，这就导致仿真波形反映的还是上次编译的ELF得到的结果（也有可能是我操作的姿势不对）。有时候必须要手动调用这些工具，一种方法是在Tcl控制台中输入命令。在Project Navigator的菜单View – Panels – Tcl Console打开Tcl控制台：

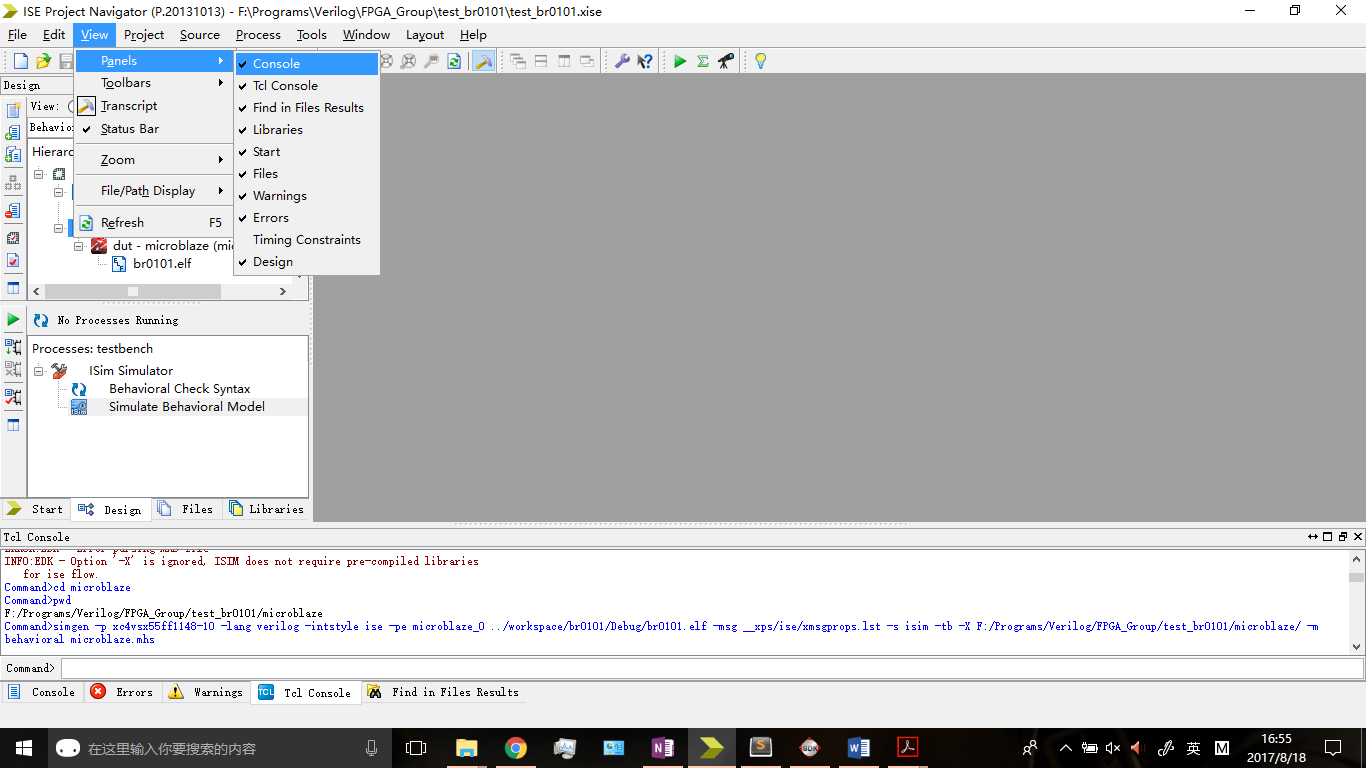


图 6 打开Tcl控制台

在Tcl Console中输入以下命令可以手动调用Simgen生成行为仿真验证模型：

cd microblaze

simgen -p xc4vsx55ff1148-10 -lang verilog -intstyle ise -pe microblaze\_0 ../workspace/br0101/Debug/br0101.elf -msg \_\_xps/ise/xmsgprops.lst -s isim -tb -X F:/Programs/Verilog/FPGA\_Group/test\_br0101/microblaze/ -m behavioral microblaze.mhs

这些命令可以在Simgen Log File的开头找到。

除了Project Navigator之外，Xilinx还提供了PlanAhead工具作为另外一种开发环境。PlanAhead进行嵌入式开发的界面大致如下所示：

brOIOI_planahead - 
File Edit Flow 
Tools Window Layout View' Help 
Open 
Settings 
Generating 
- PlanAhead 14.7 
o j e ct Snuy 
Flo. 
project Settings 
8 
Add 
IP Catalog 
Simulation 
RIL Analys 
Symthes1S 
Symthes1S Settings 
Symthes1S 
Open S >mth 
esized 
Impl on 
Impl 
Open 
Debug 
Settings 
ChipScope 
iMPAcr 
project — ImOIOI_p1anahea 
Waiting I cl 
o a z e _t op 
(miuoblaze_ 
(miuoblazæ :mp) 
(1) 
project Settings 
project 
ImOIOI_p1 a 
product 
project put: 
--10 
(S) 
(S 
C) 
Messages 
um ngs 
Y miuoblaze_i — 
AE ELF (1) 
Hi u archy IP 
I at e s 
aze 
Launch Simulator 
simulation mo 
x c4v sxSSff114S—IO 
x c4v sxSSff1148—IO 
Rep 
Synthe s 
Status: 
sub—desi 
x c4v 148 
Ready 
Status: 
Put: 
Str at e 
um ngs 
nt at I on 
Ready 
xc4vsxssff1148-10 
ISE Defaults 
the 
C ons tr s 
constrs I 
constrs I 
Back g 
m atlon 1 
Str at e 
Re ady 
(MHz) 
symth_l 
imp 1 _ 
console 
Defaults 
(ISE 
ISE Defaults 
(xsr 
14) 
14) 
Not 
Not 
st uted 
st u ted 
simulation models for 
gn 'microblaze 
the XPS sub—de s so 
2017/8/17 

图 7 PlanAhead开发环境

不过PlanAhead也容易出现各种问题，网上教程还少。其实最好的办法还是自己写Tcl脚本管理所有的流程和工具，不过难度比较高。据说Xilinx的新开发工具Vivado完美地解决了这些问题，希望以后可以学习一下。

### 硬件仿真验证流程图

参考Xilinx关于嵌入式系统开发的教程[2] ，我们得到基于Project Navigator进行嵌入式系统硬件仿真验证的大致流程图如下：



图 8 基于Project Navigator进行嵌入式系统硬件仿真流程图

### 硬件仿真验证的具体过程和遇到的问题

上述硬件仿真的流程是我们在查阅资料和实际尝试的过程中一点点总结出来的，也可能有很多不完善的地方。在具体实践的过程中，我们也遇到了很多大大小小的问题。

#### ISE与ModelSim联合仿真的问题

一开始我们准备采用ISE与ModelSim联合仿真，这就需要预编译Xilinx的IP核，并将预编译库导入ModelSim中。之前用ISE 10.1的时候我导入过一次，但是ISE 14.7还是需要重新导入。

采用32位的预编译库向导compxlibgui.exe，编译Vertex-4系列的所有相关库，注意要取消勾选Exclude Superseded选项，否则编译会报错。

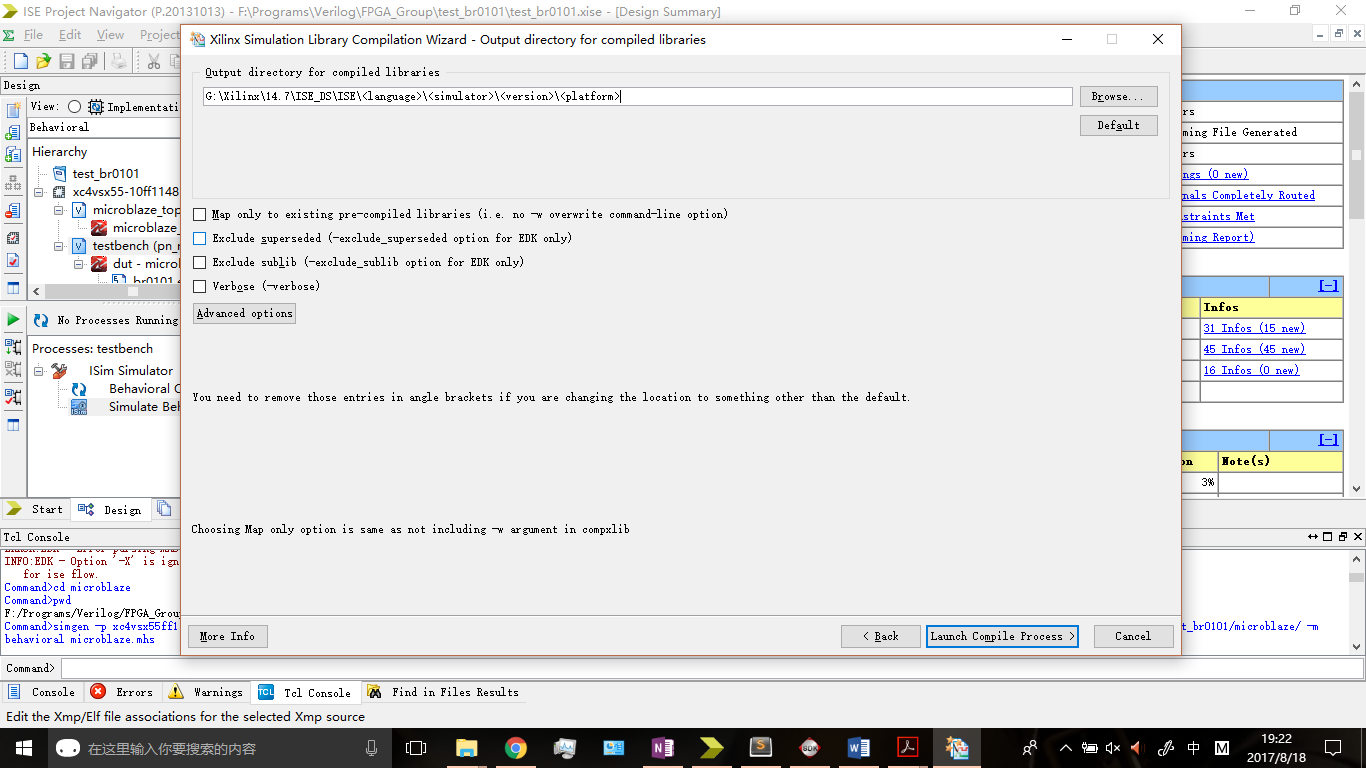


图 9 预编译库向导编译选项

编译完成后将默认路径下生成的modelsim.ini文件覆盖到ModelSim根目录，就完成了导入。

然而仿真过程中却发现，还是有一个MicroBlaze相关的库没有导入成功：

ISE Project Navigator (9.20131013) - brOIOI\test brOIOI xise - 
File Edit View Project Source Process Tools Window Layout Help 
Behavioral 
console 
in Files Results 
[pn 
. PI b dac O 
. PI b dac O 
S Format pin ( PI b dac O S Format pin ) , 
S PWRDN pin ( plb dac O S PWRDN pin ) 
H ierarchy 
test brOIOI 
xc4vsx55-IOff1148 
microblaze_top (microblaze_top.v) 
microblaze_i - 
microblaze (mic 
testbench (pn microblaze_testben•—— 
dut - microblaze (microblaze.x 
Run Failed: Simulation 
Processes: testbench 
ModelSim Simulator 
M Simu late Behavioral Model 
The fil 
70 
O 
O 
// Clock generator for fpga O clk I sys clk pin 
initial 
begin 
fpga O clk I sys clk pin = I 'bO; 
forever # (fpga O clk sys clk pin PERIOD/ 2. 00) 
fpga O clk sys clk pin 
fpga O clk sys clk pin; 
end 
// Reset Generator for fpga 0 1st I sys 1st pin 
initial 
begin 
fpga O ISC I sys ISC pin = I 'bO; 
# (fpga 0 1st sys 1st pin LENGTH) fpga O rsC I sys rsC pin 
end 
— —fpga O rsC I sys rsC pin; 
START 
User: 
USER CODE (Do not remove this line) 
Put your stimulus here. Ccde in this 
section will not be overwritten. 
types pkg.vhd is distributed by Xilinx encrypted and w - 
not be read by any simpaIatOI. 
Please use compxlib to setup the EDK 
ilea libraries andprovide Che pach Co Chem using the —E sw 
[simulation/ behavioral/ microblaze setup. do] ' 
make : 
ERROR : EDK 
Console 
W ngs 
Find 
Ln 76 cal 8 
2017/8/14 
Verilog 

图 10 ModelSim仿真报错

上网查阅资料后发现，有人也遇到了这种问题，好像是因为XPS没有正确地指定库的位置。最后我们没能找到较好的解决方法，只能采用ISIM代替ModelSim作为仿真器。

#### SDK自定义IP驱动导入的问题

在软件开发过程中我们要给自定义IP写驱动程序，并把它导入到SDK中使用。在创建自定义IP时XPS也为我们生成了驱动程序的模板，只需要对它进行简单的修改即可。然而在导入SDK时却出现了一些问题。

按照官方教程，只需要在SDK的菜单Xilinx Tools – Repositories中，将包含驱动的文件夹添加到本地库中即可：

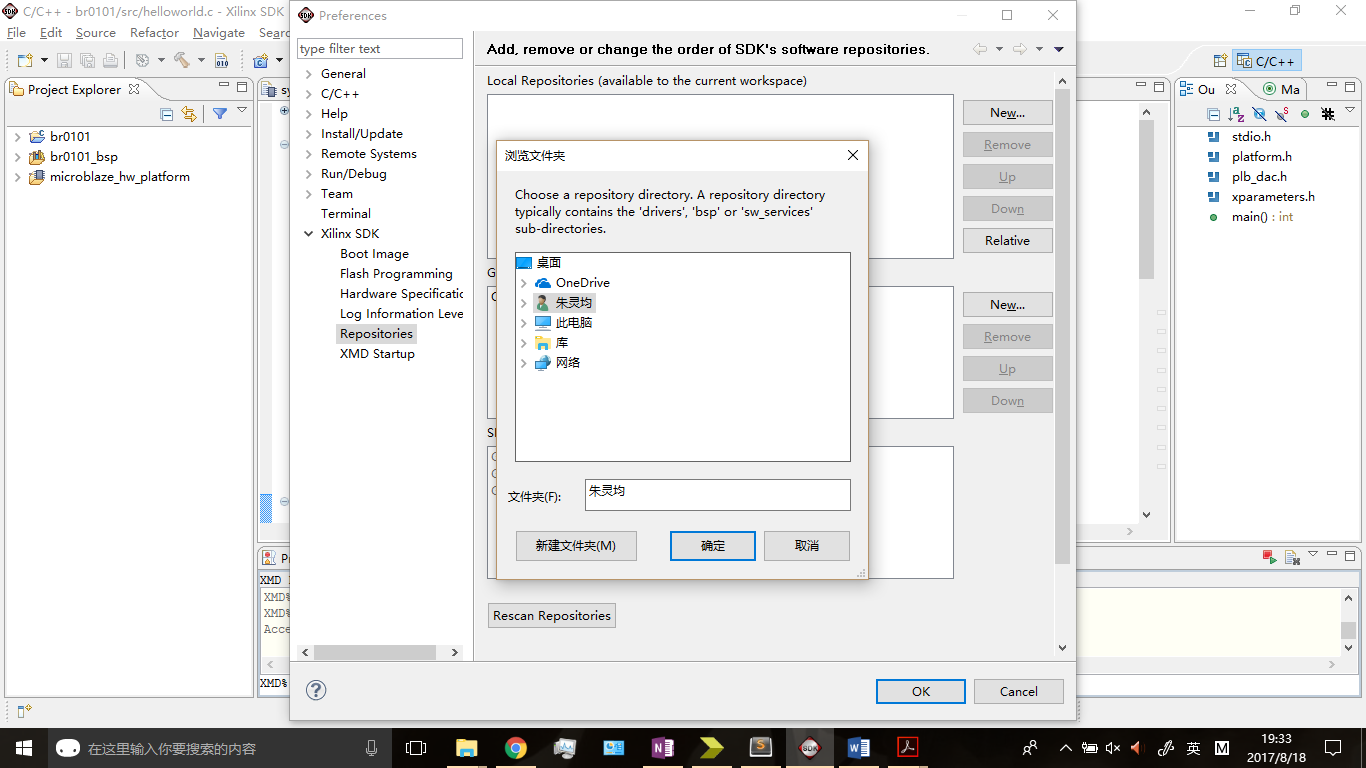


图 11 在SDK中添加自定义IP驱动库

然而这样做常常不起效果。查阅网上论坛的资料后发现，这是SDK的一个bug，必须要将驱动文件夹drivers放到C:\edk\_user\_repository\MyProcessorIPLib，并添加为Global Repositories，SDK才能找到。不仅如此，当前的项目还不能立即检测到这个库，Rescan Repositories也没用。必须要把这个SDK项目删掉，重新创建一个，才能正确找到自定义IP核的驱动。

#### ELF文件与仿真模型更新的问题

前面一节也提到这个ELF文件与仿真模型不同步的问题。一开始我们没有搞明白Simgen的作用，为了强制更新仿真模型，我们只要修改了程序就必须重新在XPS中更改硬件设计，生成网表，更新顶层HDL文件，重新导出到SDK，将整个流程再跑一遍，浪费了很多时间。

#### 软硬件结合出现的问题

按照上述流程，我们对整个包含DAC固件、软硬件结合的MicroBlaze嵌入式系统进行了仿真，第一次仿真得到的波形如下所示：

File Edit View Simulation 
and Processes 
Instances 
'Sim (9.20131013) - [Default.wcfg] 
es Results 
h Results 
Window La'ß•ut Help 
Objects 
Simulation Objects E 
aun c h 
Name 
Instance and Process Name 
std logic_1164 
attributes 
std logic_misc 
std logic_arith 
std logic_signed 
nu meric std 
text i o 
vital timing 
vital_primitives 
std logic_u nsigned 
Imb bram if funcs 
proc common_pkg 
ipif pkg 
family support 
'"0 m pone nts 
G vpkg 
VI types 
Instanc••• 
Console 
Default flow is: r 
test k 
std 
attril 
std 
std 
std 
texti 
vital 
vital 
fami 
vpkc 
VI ty 
Object Name 
fpga O RS23.. 
plb dacOS 
plb dacOS 
plb dacOS 
plb_dac O S 
plb dacOS 
plb dacOS 
plb dacOS 
fpga_O clk_u 
fpga O rst I 
fpga_o RS23„ 
fpga O clk_ 
fpga O rst I 
Value 
0000000000 
20000 
320000 
run IOOOns 
O 
O 
seuc 
V 
14 fpga o_RS232 
plb dacOS1 
plb dacOSI 
plb dacOSI 
plb dac O S 
plb dacOSI 
plb dacOSI 
fpga O clk_l 
fpgaOrst I _ 
fpga O RS232 
fpga O clk_l_ 
fpga O rst I 
Top level port 
plb dac C 
01b dac C 
Value 
0000000000 
20000 _ 0000 
320000 _ 000 
x 
Default wcEg 
w _ top; run IOOOns o 
Setting up 
# run all 
Console 
1 gna_l tracing 
Find 
Sim Time: fs 
2017/8/16 

图 12 MicroBlaze嵌入式系统第一次仿真波形图

可以看到DAC相关的信号PWRDN、DATA等完全没有变化。为了找到问题的原因，我们做了很多尝试：改变读写寄存器的方式、改变延时的时间、删除UART传输程序等等，但都没有起到效果。

最后我们发现，这还是因为PLB总线的比特方向颠倒导致的。之前固件单独仿真的时候我们就遇到了这个问题，但是认识得不深入，以为软硬件协同的时候可以自然解决，结果出现了很大的问题。

在生成的PLB固件模板中，信号都是以wire [0 : WIDTH -1] signal;的格式声明的，与我们常见的位宽声明方向相反。这导致信号整个颠倒过来，signal[0]是MSB，而signal[31]是LSB。不知道这是否与PLB总线的Big Endian特性有关。

在搞清楚问题之后，修改起来还是较为简单的，只要在固件开发的时候把signal[0]当成MSB就可以。但是要注意后面添加约束文件（.ucf）的时候，也要保证信号相符，比如DAC的并行数据端口DB[9]就得接FPGA输出信号S\_DATA[0]。

# 硬件仿真验证的结果

最后仿真验证的结果如下所示：

Float (9.20131013) - [Default.wcfg*] 
File Edit View Simulation Window 
z 
Layout 
Name 
fpga O RS232 TX pin 
plb dac O S_CIkMD_pir 
plb dac O S_CIkout pin 
plb dac O s_DCLK10 pi 
plb_dac O S_Data pinCC 
plb dac O s_PWRDN p 
plb dac O S_PinMD_pir 
fpga O clk_l sys clk_pi 
fpga O rst I sys rst pir 
fpga O RS232 RX pin 
fpga O clk_l sys clk_pi 
fpga O rst I sys rst pir 
Top level ports 
Value 
20000 _ 000000 
320000 _ 000000 
x 
1000110100 
Help 
17,850 n 
234 
100•• 
345 
110•• 
17,900 n 
234 
100•• 
17, ns 
77 
000 ns 
Oso 
234 
100••• 
18, ISO n 
345 
R e —1 aun c h 
18, 200 ns 
234 
100•• 
300 ns 
345 
345 
18, 100 
234 
100••• 
345 
234 
16 
16 
16 
16 
plb dac O S PWRDI 
plb dac O S Format 
plb daco S clkMD 
plb dacos PinMD 
plb dac O S Clkout 
plb daco S DCLKI( 
plb_dac O S_Data p 
110•• 
100•• 
234 
100•• 
345 
110•• 
234 
100•• 
345 
110•• 
fpga O rst I sys rst 
fpga O clk_l sys Cll 
16 fpga O RS232 TX pi 
Default wcEg* 
Sim Time: fs 
2017/8/18 

图 13 MicroBlaze嵌入式系统仿真验证结果

可以实现DAC的使能控制、格式控制、交叉读写控制等功能，与固件单独仿真的结果也相符：

Float (9.20131013) - [Default.wcfg*] 
Edit View Simulation Window 
Value 
Name 
IP2DAC 
284 
IP2DAC 
IP2DAC 
IP2DAC 
IP2DAC 
IP2DAC 
IP2DAC 
IP2aus_l 
oooooooc 
IP28us 
IP28us ' 
IP28us 
aus21P 
aus21P 
aus21P 
1234234E 
aus21Pl f 
aus21P 
aus21P 
La 'ß•ut 
000 
Help 
234 
11 
000 
700 ns 
Re—I aunch 
950 n 
234 
750 ns 
234 
000 
234 
000 
000 
234 
000 
234 
xl: 988737 ns 
De Eau] E 
000 ns 
234 345 
Sim Time: ps 
2017/8/18 

图 14 固件单独仿真结果

# 2 AD9715测试程序的设计

我们希望设计一个测试程序，对AD9715的基本特性和固件的功能进行测试。

## 2.1 测试程序的初步设计

为了进行上板调试，我们先初步设计了一个简单的测试程序。它的功能就是简单地写控制寄存器，使能AD9715，再向数据寄存器中循环写入从0到的数据，使AD9715的I通道产生锯齿波。

## 2.2 测试程序的改进

为了实现多种波形产生、频率改变、通道选择等功能，我们计划对测试程序进行改进，通过UART实现一个简单的UI，对AD9715进行控制。这就需要用到UART的中断读写等功能，还需要进一步研究。

# 3 AD9715上板调试实验

在王老师的指导下我们进行了AD9715上板调试实验，在实验过程中也发现了不少问题和需要改进的地方。

## 3.1 问题1：无法进行位流下载

我们最初尝试进行位流下载的时候发现，我的电脑无法正常连接到Xilinx下载器。虽然设备管理器中能够找到Programmable Device下的Xilinx Cable，但在XPS的FPGA Programming向导中每次尝试下载都会失败。如果在iMPACT工具中搜索器件，也无法找到下载器。推测可能是驱动安装的问题。

之后我们将工程拷贝到实验室的测试机上，再进行位流下载，就没有再出现什么问题。

## 3.2 问题2：UART乱码

我们最先进行的是UART通信部分的调试。测试程序的开头会通过UART输出“Hello World”，但是一开始我们却只能收到一串乱码。我们意识到可能是波特率配置有错误，但是硬件和串口调试助手都设置的是波特率9600，无校验位，1位停止位。

后来我们发现是在XPS的MicroBlaze配置中，我们将系统时钟设为了默认的50MHz，而板上实际输入的时钟信号是100MHz，这就导致UART模块产生的波特率出错。只要在顶层模块中加入一个时钟分频器，对输入时钟进行二分频，再作为系统时钟连接到MicroBlaze软核上，就可以解决这个问题。

不过，这个时钟分频的代码和之前复位产生器的代码都很奇怪，因为没有复位信号。在仿真过程中会出现一些问题，但是在实际板子上确实可以用。

## 3.3 问题3：UCF文件配置有问题

我们最初将管脚约束添加在microblaze/data文件夹下自动生成的microblaze.ucf中，但是项目没有正确地把这个文件包含在内，导致管脚约束出现了一些问题。运行PlanAhead – I/O Pin Planning就发现信号没有约束到正确的管脚上。于是我们删除了原来的UCF文件，在项目根目录下重新添加了一个UCF文件，将管脚约束写在里面，就解决了这个问题。

## 3.4 问题4：输出信号无法快速变化

在实验中我们发现了一个奇怪的现象：如果传送给DAC的数字信号变化稍快（快于2ms/点），系统就无法输出正确的模拟信号。这个问题暂时还没有得到解决。

第二次调试中，我们发现可能的原因是FSADJx管脚上的电压被拉低了。

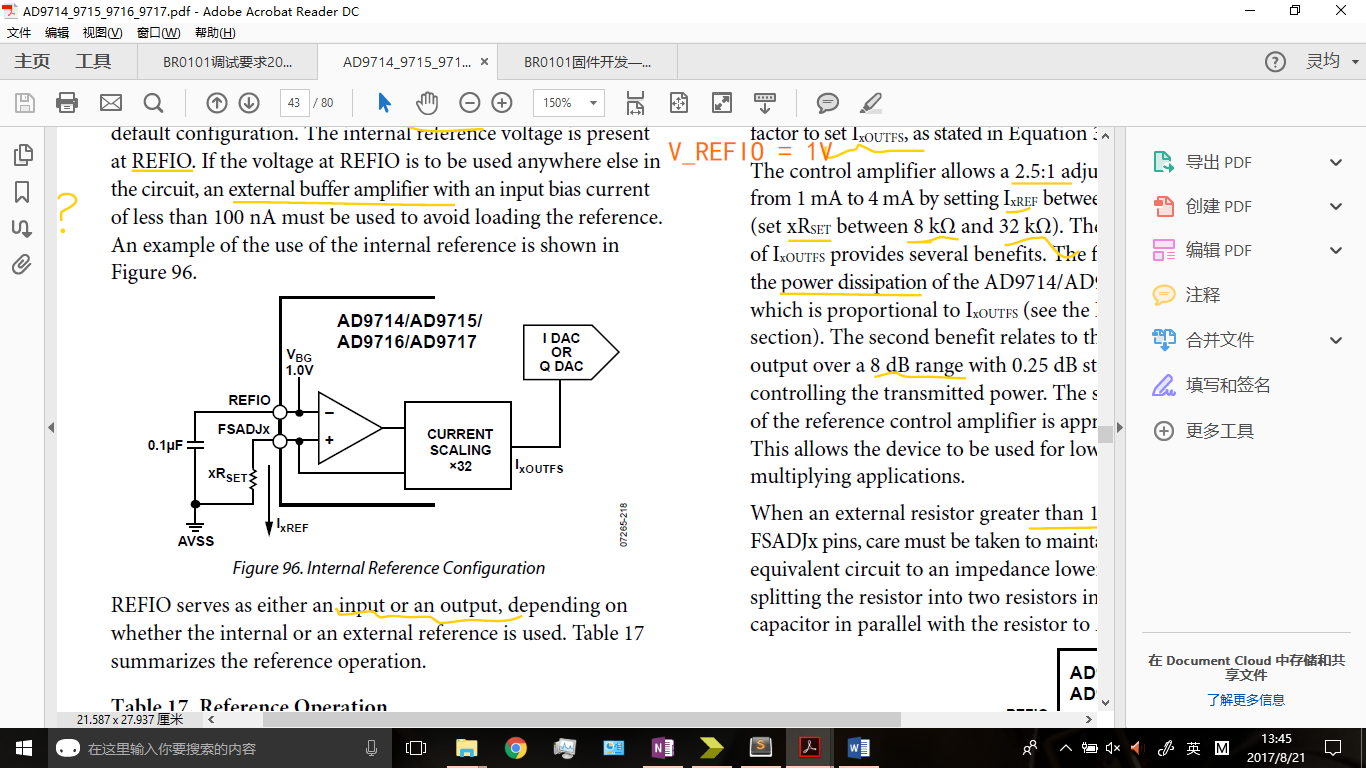


图 15 AD9715参考电流的产生

按照数据手册上的描述，AD9715的参考电流是由REFIO管脚上的电压和FSADJx管脚上的外接电阻共同决定的。根据电路图，REFIO和FSADJx连接在一个运放的反相端和正相端。根据运放的原理，这两个端口的电平应该十分接近。但是我们在实验中测量发现，在运放采样率较低（数字信号变化延时为10000次循环以上）时，REFIO和FSADJx的电平都为1V；而在运放采样率较高（数字信号变化延时为2500次循环以下）时，REFIO的电平仍为1V，FSADJx的电平却下降到了0V。这就导致随着采样率的升高，参考电流降低，DAC的输出满标电流减小，输出电压变化范围大大减小。暂时还没有找到解决方法。

# 参考资料

1. Xilinx,  [Embedded System Tools Reference Manual](https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/est_rm.pdf)
2. Xilinx,  [EDK Concepts, Tools, and Techniques](https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/edk_ctt.pdf)