

TJ-FPGA 仿真平台介绍

2018 年 12 月 10 日星期一

目 录

1. 背景.....	3
2. 仿真平台特性.....	3
3. 仿真平台使用方法.....	3
3.1 平台目录.....	3
3.1.1 可综合代码目录 rtl.....	3
3.1.1.1 举例说明.....	4
3.1.2 基本验证文件目录 tb/btc.....	4
3.1.2.1 举例说明.....	5
3.1.3 仿真执行目录 runsimlin.....	5
3.1.3.1 举例说明.....	5
3.1.4 验证目录 vf.....	6
3.1.4.1 目录介绍.....	6
3.1.4.2 扩展性开发.....	7
3.2 使用方法.....	7
3.2.1 基于 vivadoSim 的仿真使用方法.....	7
3.2.1.1 编写 rtl 代码和验证文件.....	7
3.2.1.2 修改 tc.cfg 配置文件.....	9
3.2.1.3 执行 makefile 脚本.....	9

1. 背景

为方便开发 RTL-CNN 推理加速器，方便组内人员协同工作，共同完成 VLSI 规模的集成电路设计工作，保证设计的每个阶段都向着正确方向前进，特此开发了 TJ-FPGA 仿真平台。

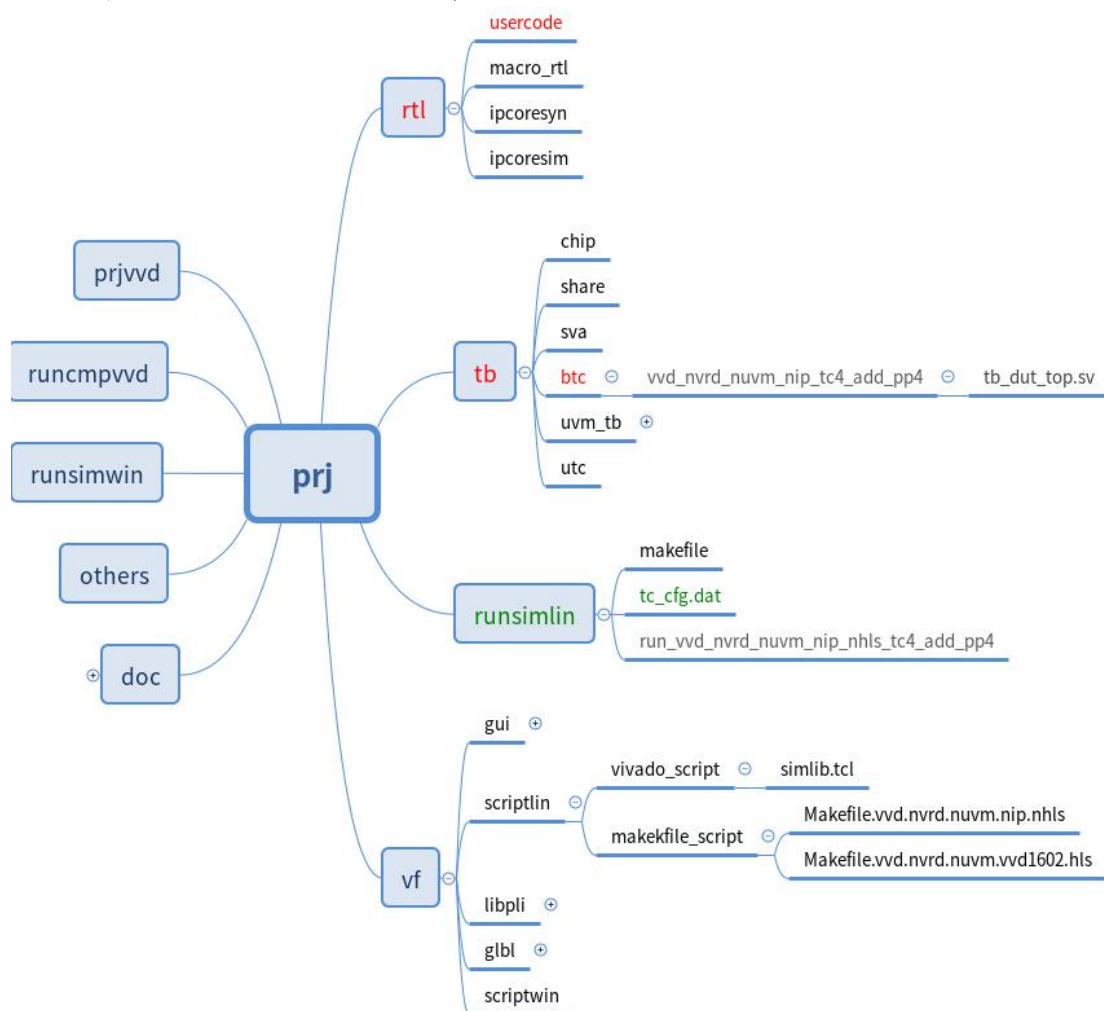
2. 仿真平台特性

- 支持模块单元测试
- 支持多用例测试
- 支持系统级别验证
- 支持工程碎文件自动清理
- 支持 git 版本管理
- 支持每日模拟冒烟，并通过 jenkins 统一管理

3. 仿真平台使用方法

3.1 平台目录

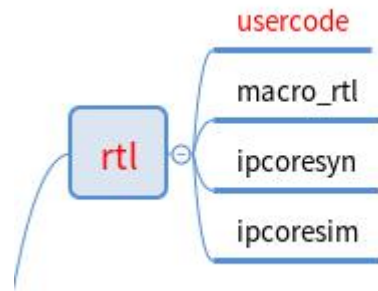
目录结构主要分为 4 个，如图所示。



3.1.1 可综合代码目录 rtl

该目录下放置可综合的代码，也就是实际要在板子上，经过综合工具能生成电路的代码，具体结构如图所示，usercode 表示要实际放的代码文件，比如要设计流水线加法器，那么

就可以把流水线加法器替换 usercode。

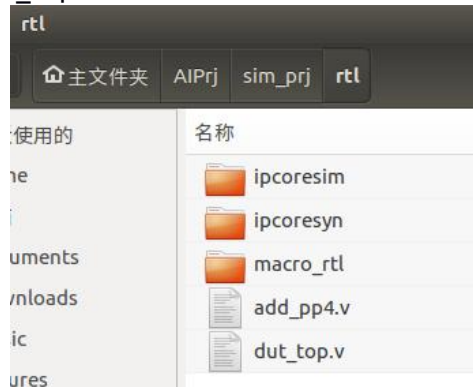


如果代码中有调用 xilinx 的 ip，比如调用 fifo，ram 之类的 ip，若该 ipcore 要支持仿真，就把它放到 ipcoresim 目录下，若要支持综合，就放到 ipcoresyn 目录下。

另外，还有些情况需要设置一些宏参数，我们可以把宏参数统一汇总成一个文件后，放到 macro_rtl 目录下。

3.1.1.1 举例说明

比如要设计流水线加法器，直接把写好的文件放到 rtl 目录下，这里流水线加法器用了 2 个文件 add_pp4.v 和 dut_top.v。



3.1.2 基本验证文件目录 tb/btc

目录结构如图所示。



一般的，写完 rtl 设计，需要再自己写 testbench 文件，用来测试设计代码是否满足既定的要求，这一点和 c/c++ 中的测试用例非常相似。

验证文件放到 tb/btc 下面，一般给验证文件起名为 tb_dut_top.sv 这种名字，当然起其他的名字也可以。

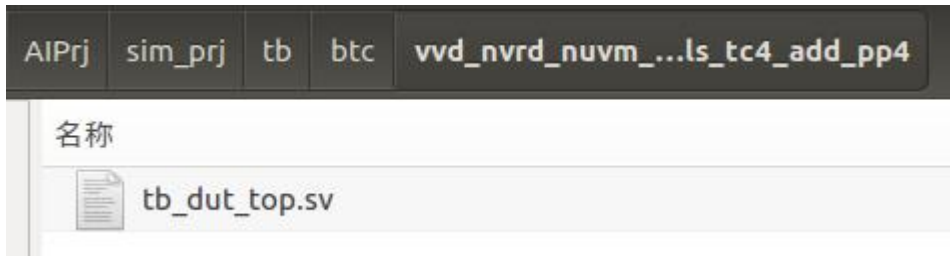
tb(testbench 的缩写)

btc(base testbench 的缩写，表示基本测试用例)

后续开发中，还会有 uvm testcase，这个版本暂时不提供，主要是因为 vivado 工具不支持 uvm。

3.1.2.1 举例说明

比如要设计流水线加法器的验证文件，该文件名字是 tb_dut_top.sv，我们把它放到 tb/btc 目录下。



特别注意：放该文件时，需要先创建一个文件夹，该文件夹的名字是 vvd_nvrd_nuvm_nip_nhls_tc4_add_pp4，这个文件夹名字很奇怪，这个文件夹的名字，就是我们跑仿真时候的用例名称。

解释：

我们仿真平台的用例支持 5 个维度的功能，每个维度之间用“_”来分割。

分析“vvd_nvrd_nuvm_nip_nhls_tc4_add_pp4”名称。

第一维度：仿真器的类型，取 vvd 字符串，表示 vivado sim，意思是我们选择该用例的仿真器是 vivado sim。

第二维度：表示是否用到 verdi 波形查看工具，取 nvrd 字符串，表示 no verdi，意思是我们选择不用 verdi 工具。

第三维度：表示是否用到 uvm 库，取 nuvm 字符串，表示 no uvm，意思是我们不用 uvm 库。

第四维度：表示用到了哪个版本的 ip 库，取 nip，表示 no ipcore，意思是我们不用任何 ipcore 库。

第五维度：表示是否用到 hls，取 nhls，表示 no hls，意思是这个是我们自己写的代码，不是 hls 工具生成的代码。

第六维度：用例真实的名字，取 tc4_add_pp4，这个名字我们可以随意起名，这里的意思是，表示它是第 4 个仿真流水线加法器的用例。

3.1.3 仿真执行目录 runsimlin

目录结构如图所示：



设计和验证文件都写完后，需要进行真实的仿真，进入 runsimlin 目录下，该目录下有 makefile 和 tc.cfg 配置文件。

makfile 文件负责执行 make 命令，进行仿真命令解析执行等功能。

tc.cfg 文件是仿真用例配置文件，用来告诉仿真器，需要仿真那些测试用例。

3.1.3.1 举例说明

比如要仿真流水线加法器，打开 tc.cfg 文件后，按照如下格式填写用例名称。我们知道，

要用的仿真器是 vivadosim。因此名字前缀 1 vvd

我们不用 verdi，因此前缀 2,nvrd

我们不用 uvm,因此前缀 3,nuvm

我们不用 ip，因此前缀 4,nip

我们不用 hls，因此前缀 5,nhls

我们给用例起名字时 tc4_add_pp4

每个项之间用_分割开，

根据这些需求，因此用例全称为：vvd_nvrd_nuvm_nip_nhls_tc4_add_pp4

```

29 #####Simulation TestCase Setting Begin #####
30 ##TC_NAME=qst_vrd_nuvm_nip_nhls_tc2
31 ##TC_NAME=qst_nvrd_nuvm_nip_nhls_tc1
32 ##TC_NAME=vvd_nvrd_nuvm_nip_nhls_tc3
33 ##TC_NAME=qst_vrd_nuvm_nip_nhls_tc4_add_pp4
34 TC_NAME=vvd_nvrd_nuvm_nip_nhls_tc4_add_pp4
35 ##TC_NAME=qst_vrd_nuvm_vvd1602_ddr3_tc5_ddr3_example
36 ##TC_NAME=qst_vrd_nuvm_vvd1602_pcie_tc6_pcie_example
37 ##TC_NAME=qst_vrd_nuvm_nip_nhls_tc7_j2k
38 ##TC_NAME=qst_vrd_nuvm_vvd1602_hls_tc8
39 ##TC_NAME=vvd_nvrd_nuvm_vvd1602_hls_tc9
40
41 #####Simulation TestCase Setting End #####

```

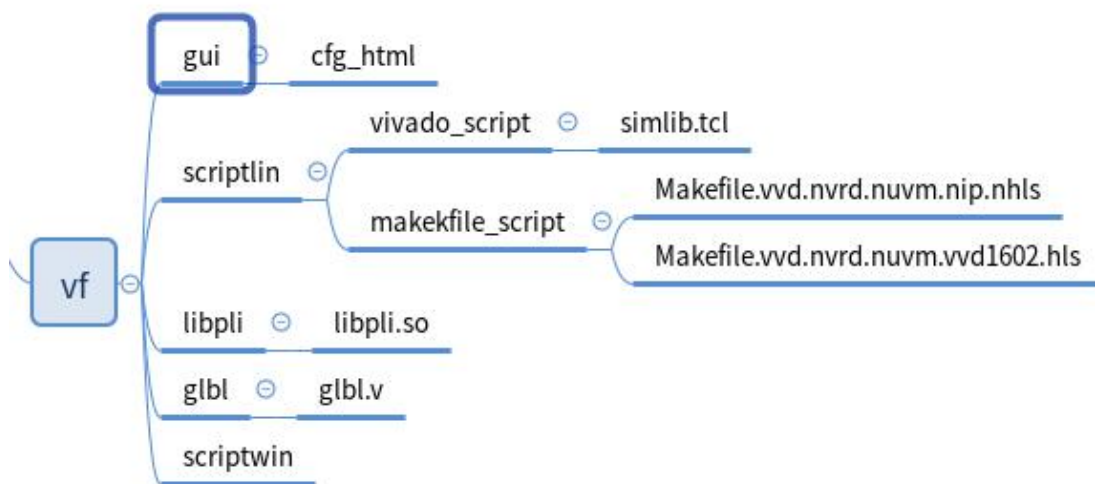
用例格式：

TC_NAME=xxx

这种方式，如果注销掉这个用例，只需要在行首添加#即可。

3.1.4 验证目录 vf

目录结构如图所示，验证目录 vf(verification 的缩写)，这个目录下存放了仿真执行的脚本信息，以及一些库信息，是仿真工具得以自动化执行的基础，仿真平台开发的新功能以及需求，都是在这个目录下。



说明：一般入门用户不需要关注和修改这个目录，如果在这个仿真平台上开发新需求，需要关注该目录。

3.1.4.1 目录介绍

gui 目录下，存放页面开发需求中相关的 html 文件，比如仿真环境中，testcase 用例的页面配置等等，该需求尚未开发；

scriptlin 目录，表示 linux 脚本目录，很重要，里面的 vivado_script 目录下的 simlib.tcl 文件是自动生成的，用户不用关心。里面的 makefile_script 目录很重要，实际建立仿真工程时，新建测试用例中的 5 个前缀信息，对应了 Makfile.xx1.xx2.xx3.xx4.xx5，这 5 个信息，主 makefile 会解析测试用例名称，根据名称的 5 个前缀，在 makefile_script 下找到对应的 Makefile 子文件，进行执行相应配置下的 makefile 命令，完成仿真功能。

libpli 目录存放库；

glbl 目录存放 xilinx 仿真需要的 glbl.v 文件

scriptwin 目录存放 windows 环境下仿真执行的脚本信息，一般都是 bat 脚本，预留文件夹，该需求暂未开发。

3.1.4.2 扩展性开发

比如实际使用过程中，发现目前的 5 个前缀信息中，都不能很好的支持新的仿真需求，那么可以自定义新的前缀信息，再开发新的 Makefile 子文件即可，该仿真平台可以满足大部分应用需求。

3.2 使用方法

以流水线加法器为例，介绍仿真器的具体使用方法，其他开发方式也类似。

3.2.1 基于 vivadoSim 的仿真使用方法

仿真平台本身不支持编辑器环境，也没有 gui 界面，用户可以根据自己的喜好使用编辑器，常见的比如 vscode，gvim 等等，都很好。

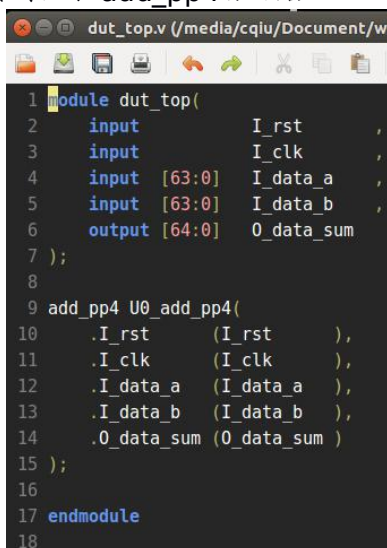
3.2.1.1 编写 rtl 代码和验证文件

仿真平台本身不支持编辑器环境，也没有 gui 界面，用户可以根据自己的喜好使用编辑器，常见的比如 vscode，gvim 等等，都很好。

3.2.1.1.1 编写 rtl 代码

3.2.1.1.1.1 dut_top.v 顶层

代码见下，这个文件就是实例化了 add_pp4 加法器。

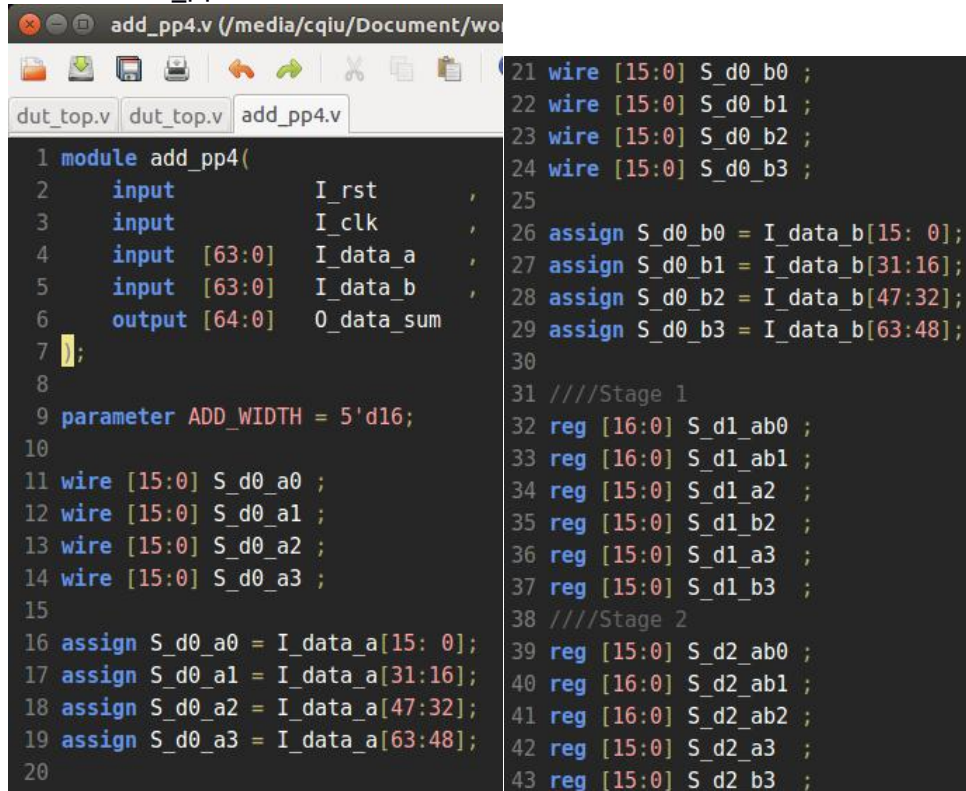


```

1 module dut_top(
2     input      I_rst
3     input      I_clk
4     input [63:0] I_data_a
5     input [63:0] I_data_b
6     output [64:0] O_data_sum
7 );
8
9 add_pp4 U0_add_pp4(
10     .I_rst      (I_rst)
11     .I_clk       (I_clk)
12     .I_data_a    (I_data_a)
13     .I_data_b    (I_data_b)
14     .O_data_sum  (O_data_sum)
15 );
16
17 endmodule
18

```


3.2.1.1.1.2 add_pp4.v 部分代码

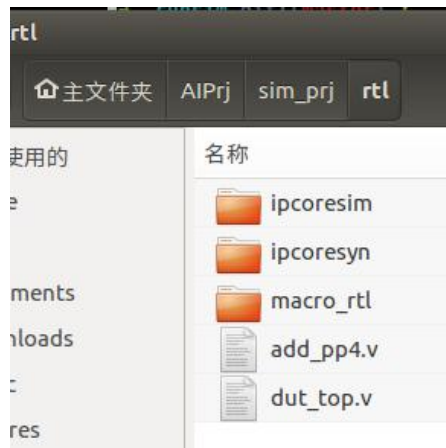


```

1 module add_pp4(
2     input        I_rst      ,
3     input        I_clk      ,
4     input [63:0]  I_data_a   ,
5     input [63:0]  I_data_b   ,
6     output [64:0]  O_data_sum
7 );
8
9 parameter ADD_WIDTH = 5'd16;
10
11 wire [15:0] S_d0_a0 ;
12 wire [15:0] S_d0_a1 ;
13 wire [15:0] S_d0_a2 ;
14 wire [15:0] S_d0_a3 ;
15
16 assign S_d0_a0 = I_data_a[15: 0];
17 assign S_d0_a1 = I_data_a[31:16];
18 assign S_d0_a2 = I_data_a[47:32];
19 assign S_d0_a3 = I_data_a[63:48];
20
21 wire [15:0] S_d0_b0 ;
22 wire [15:0] S_d0_b1 ;
23 wire [15:0] S_d0_b2 ;
24 wire [15:0] S_d0_b3 ;
25
26 assign S_d0_b0 = I_data_b[15: 0];
27 assign S_d0_b1 = I_data_b[31:16];
28 assign S_d0_b2 = I_data_b[47:32];
29 assign S_d0_b3 = I_data_b[63:48];
30
31 ///Stage 1
32 reg [16:0] S_d1_ab0 ;
33 reg [16:0] S_d1_ab1 ;
34 reg [15:0] S_d1_a2 ;
35 reg [15:0] S_d1_b2 ;
36 reg [15:0] S_d1_a3 ;
37 reg [15:0] S_d1_b3 ;
38 ///Stage 2
39 reg [15:0] S_d2_ab0 ;
40 reg [16:0] S_d2_ab1 ;
41 reg [16:0] S_d2_ab2 ;
42 reg [15:0] S_d2_a3 ;
43 reg [15:0] S_d2_b3 ;

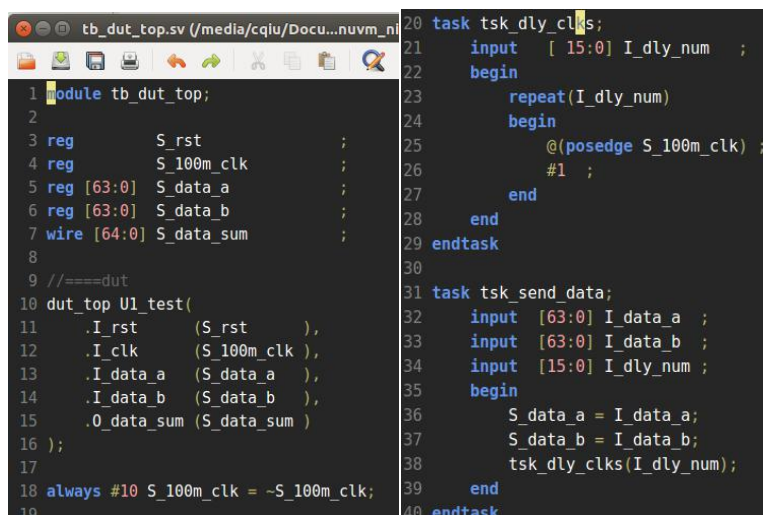
```

3.2.1.1.1.2 放到 rtl 目录下



3.2.1.1.1.3 编写 testbench 代码

代码文件是 tb_dut_top.sv，部分代码见下



```

1 module tb_dut_top;
2
3 reg S_rst ;
4 reg S_100m_clk ;
5 reg [63:0] S_data_a ;
6 reg [63:0] S_data_b ;
7 wire [64:0] S_data_sum ;
8
9 //====dut
10 dut_top U1_test(
11 .I_rst (S_rst ),
12 .I_clk (S_100m_clk ),
13 .I_data_a (S_data_a ),
14 .I_data_b (S_data_b ),
15 .O_data_sum (S_data_sum )
16 );
17
18 always #10 S_100m_clk = ~S_100m_clk;
19
20 task tsk_dly_clks;
21 input [ 15:0] I_dly_num ;
22 begin
23 repeat(I_dly_num)
24 begin
25 @(posedge S_100m_clk) ;
26 #1 ;
27 end
28 end
29 endtask
30
31 task tsk_send_data;
32 input [63:0] I_data_a ;
33 input [63:0] I_data_b ;
34 input [15:0] I_dly_num ;
35 begin
36 S_data_a = I_data_a;
37 S_data_b = I_data_b;
38 tsk_dly_clks(I_dly_num);
39 end
40 endtask

```

3.2.1.1.4 为测试用例起名字，在 tb/btc 下新建文件夹

进入 tb/btc 目录下，新建用例名称对应的文件夹，界面和终端都可以。



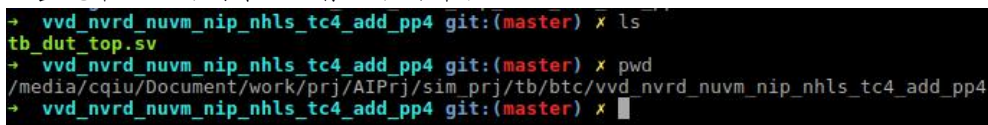
```

→ btc git:(master) x
→ btc git:(master) x
→ btc git:(master) x mkdir vvd_nvrd_nuvm_nip_nhls_tc4_add_pp4

```

3.2.1.1.5 Testbench 代码文件放到新建文件夹下

代码文件是 tb_dut_top.sv，放到 vvd_nvrd_nuvm_nip_nhls_tc4_add_pp4 文件夹下面，放置完毕后，终端界面上看如下图所示。



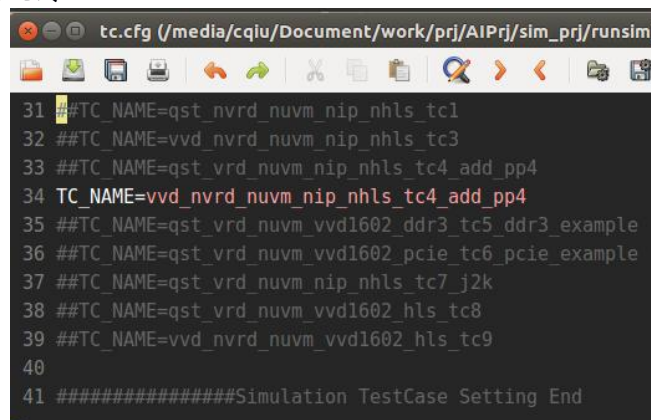
```

→ vvd_nvrd_nuvm_nip_nhls_tc4_add_pp4 git:(master) x ls
tb_dut_top.sv
→ vvd_nvrd_nuvm_nip_nhls_tc4_add_pp4 git:(master) x pwd
/media/cqiu/Document/work/prj/AIPrj/sim_prj/tb/btc/vvd_nvrd_nuvm_nip_nhls_tc4_add_pp4
→ vvd_nvrd_nuvm_nip_nhls_tc4_add_pp4 git:(master) x

```

3.2.1.2 修改 tc.cfg 配置文件

在工程目录下，进入到 runsim 目录下，打开 tc.cfg 配置文件，修改内容如下，可以只使能一个 testcase，这个 testcase 的名称就是 testbench 验证文件中，新起的文件夹名称，改完后退出关闭。



```

31 ##TC_NAME=qst_nvrd_nuvm_nip_nhls_tc1
32 ##TC_NAME=vvd_nvrd_nuvm_nip_nhls_tc3
33 ##TC_NAME=qst_vrd_nuvm_nip_nhls_tc4_add_pp4
34 TC_NAME=vvd_nvrd_nuvm_nip_nhls_tc4_add_pp4
35 ##TC_NAME=qst_vrd_nuvm_vvd1602_ddr3_tc5_ddr3_example
36 ##TC_NAME=qst_vrd_nuvm_vvd1602_pcie_tc6_pcie_example
37 ##TC_NAME=qst_vrd_nuvm_nip_nhls_tc7_j2k
38 ##TC_NAME=qst_vrd_nuvm_vvd1602_hls_tc8
39 ##TC_NAME=vvd_nvrd_nuvm_vvd1602_hls_tc9
40
41 #####Simulation TestCase Setting End

```

3.2.1.3 执行 makefile 脚本

打开终端，在 runsim 目录下，方可执行 makefile 脚本。

3.2.1.3.1 执行 make 查看帮助命令

```

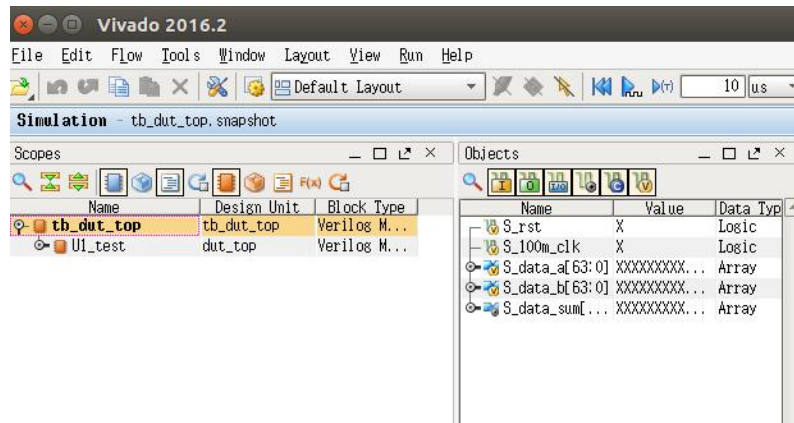
runsim git:(master) x make
*****
Build Library *****
make build_vvd1602*****
Most common commands*****
make run*****
make verdi*****
make clean*****
runsim git:(master) x

```

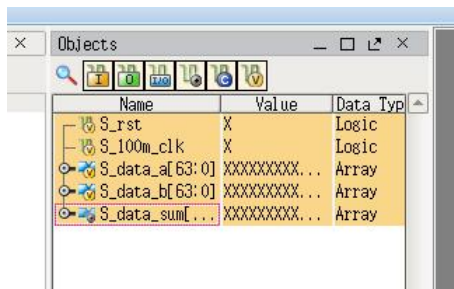
如图所示，执行 make 后，可以看出目前 makefile 创建命令并不多，最常用的就是 make run，执行仿真。

3.2.1.3.2 执行 make run 进行仿真

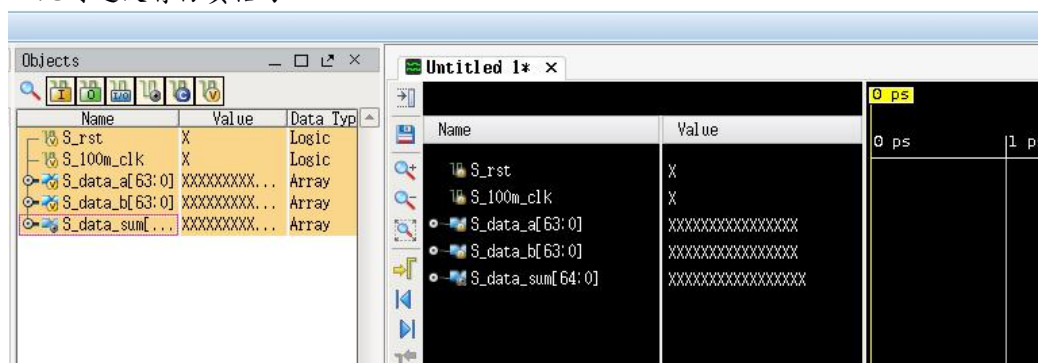
执行 make run 后，终端会自动打开 vivado 工具，如图所示，此时可以看到仿真顶层以及一些内部信号。



3.2.1.3.3 添加波形信号

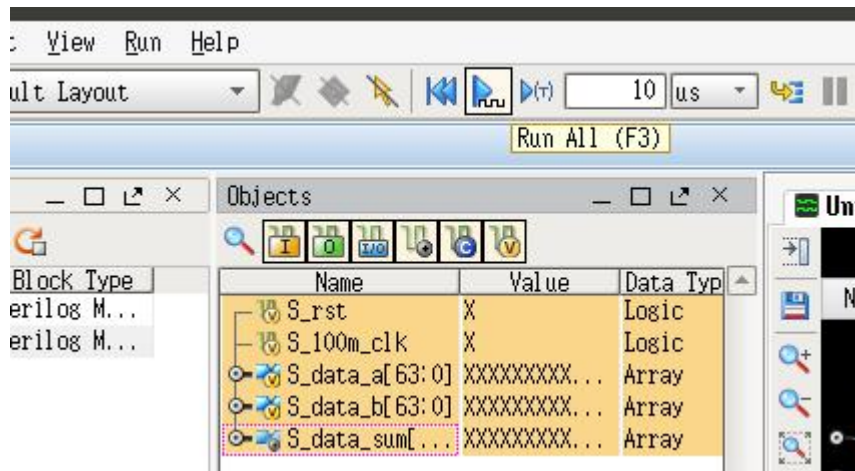


选中内部信号，鼠标右键，在下拉菜单中选择“Add wave to window”，界面如下图所示，此时还没有仿真信号。

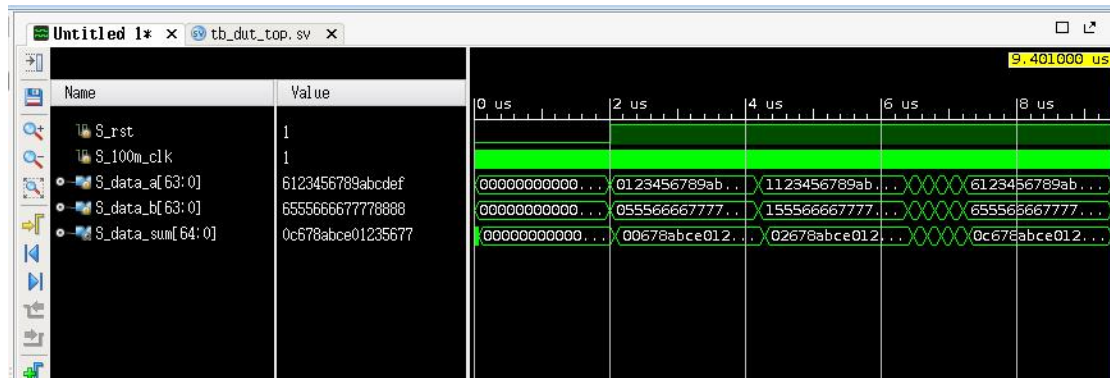


3.2.1.3.4 选择蓝色带时钟波形并 Run All

选择方法如下图所示



点击后，即可显示仿真后的波形。



至此，基于 vivado sim 的仿真介绍完毕。

3.2.1.3.5 删除仿真工程

执行 make clean 即可删除 vivado sim 建立的仿真工程，但是 rtl 设计，和仿真验证文件是保留的，这种方法可以将仿真平台进行瘦身操作，节约磁盘空间。

3.2.1.3.6 多用例仿真

如果要新建多个测试用例，则只需要新建多个 testbench 即可，并且再 tc.cfg 文件下，添加多个 TC_NAME 即可。