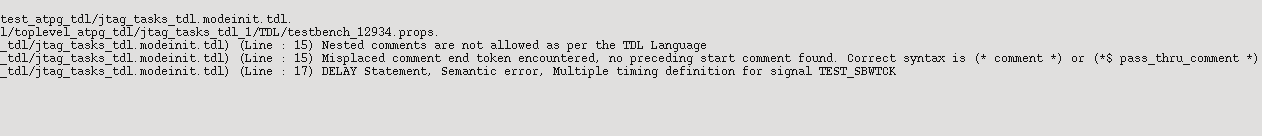
1. ASIC\_TITLE: no comments per line
2. CONNECT: DEFPIN
3. TIMING…END\_TIMING: CLOCK/DELAY/STROBE pin name must use the name defined in CONNECT
4. SETR P := T’XXXX’;
5. “Invalid vector stimulus H, for Inout 010 Clock signal P14\_UCA0TXD\_UCA0SIMO\_TA0P1\_TCK\_CAP00. Valid stimuli are LYF1XZ- ACS0MNTP”

error: L cannot be used in 101 timing; H cannot used in 010 timing.

solution: C/H/L不能同时出现，C/H,C/L或H/L

only H&L, define timing without CLOCK VAR in another file( .modeinit.tdl )

.modeinit.tdl文件一定要有，针对现在的脚本

1. 

env/bin/vtsim comment comment\_lines()

1. force\_signals not defined in the library

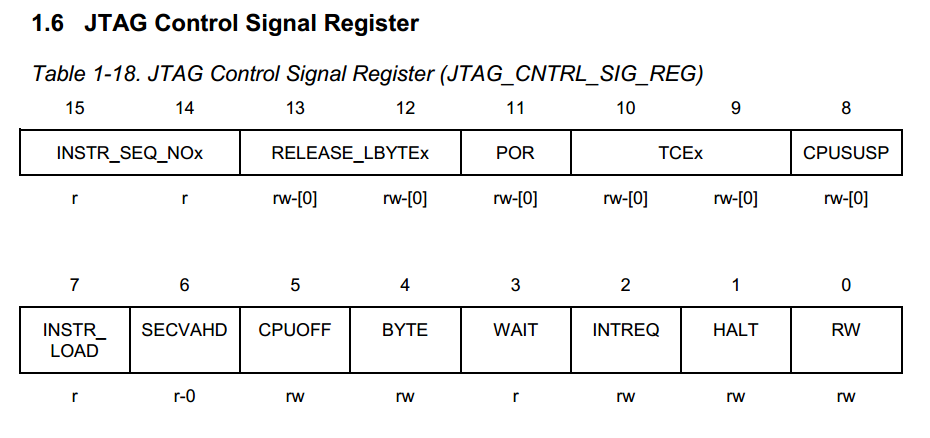
solution: toplevel/setting/define\_file.set add ‘source/force\_signal\_tdl\_rtl.v’to hdl-vlog

1. 

P14\_UCA0TXD\_UCA0SIMO\_TA0P1\_TCK\_CAP00\_DLY ??

Solution: toplevel/setting/clk.config 🡪specify the clock

1. i\_SyncJtag setreg\_16bits 0x1501 write it wrong value, 0x1401
2. 对cntrl\_sig\_register，在WriteMemWord和ReadMemWord过程中，需要关注的就是以下几个比特，默认值0501（读），在写WriteMemWord操作时，置为0500



1. 期望值：根据edt case中，如果setreg\_16bit等中没有M值（Mask），或者写的值为0x0000时（在检查某些比特值），不用在意输出值，TDL中输出值也Mask掉。setreg\_16bit中M值为0的比特mask掉。
2. 正常写FRAM之前，需要先WriteMemWord（0x00160,0xa500），打开看门狗，才能进行写操作。
3. 怎么判断是否真正写入：debug方法

vtsimgui ->interactive模式

prepare->debug 在waveform window中添加需要观察的信号，再run

添加u\_dig\_top/u\_fram\_iso/library cells/u\_feram中FUMEM[0:1023]信号

观察logical addr换算后对应的十进制的物理地址，是否在写的时候写入

1. case出现错误时，对比与其对应的正确case，将cursor放在值变化的位置，在schematic中追踪其driver，可以在schematic中看到driver值的不同，关键就是要找到driver值为什么不同！！！直到找到最原始的driver，才去看code！！！

不要轻易看code！不要轻易看code！不要轻易看code！！！

1. 写之后，确定写进了就可以读，读是有延迟的（时钟下降沿开始读，下下个时钟上升沿数据读出），在read\_addr和read\_data总线上并不会对齐，不用care这个，只需关注expected value是否等于real value。
2. 在读写FRAM操作中，需结合testbench中u\_dig\_top/u\_fram\_iso/library Cells/u\_feram模块

看下面这些信号，判断是否写入/读出

EZ：enable信号，低有效

WZ：读写信号，1read，0write

CLK：在EZ=0时，CLK上升沿进行数据操作

PORZ：复位信号

A[9:0]：读写地址 <-物理地址：取逻辑地址[12:3]

BYTESEL[7:0]：BYTE选择信号，

D[63:0]：data in

Q[63:0]: data out

1. 正常WriteMemWord/ReadMemWord跟data\_quick的区别：

正常读写：先写要读写的地址，后写数据/读数据

Data\_quick：写：先preparedataquick 将PC设为指定值，并执行data quick指令

然后WriteDataQuick（setreg\_16Bits w:$1 r:0000 m:0000 \n TCLK）

将值写入PC当前值指向的存储位置

（PC必须先被初始化，然后每个data\_quick操作自动增加2）

读：

DATA\_QUICK指令允许在IFCLK时钟下降沿使DATA\_16BIT指令自动递增PC。 因此，可以通过使用DATA\_16BIT指令以更快的方式读取或写入地址。

二者的区别：正常读写需要每次先打读写地址

data\_quick只需在读写前初始化PC，读写过程中只需打数据，PC自动+2

相较而言，data\_quick速度会快很多

1. EDT/TDL对比：
2. EDT case 有asm文件，编译成bin文件，测试机台会把bin文件load进memory，执行相应的bootcode操作。

而TDL，可直接将asm指令写在TDL文件中，不需要asm->bin->load into memory的操作。

1. for design & verification：EDT转成verilog语言；TDL同样转成verilog语言

for test：EDT转成test语言，TDL也转成test语言。

但是，test会在edt转换后的test文件中添加自己的语法，出现问题时，D&V会因为语法不一致出现理解困难。沟通起来也不方便。

而TDL比较通用一致，test和design/verification沟通起来比较方便。出现问题也比较容易理解和解决。

SUMMARY：

1. 继续尝试data\_quick的读写操作，可以与正常读写进行对比，作为final review的一个点
2. 对TDL case 的一些思考：
3. 之前的TDL用于scan for DFT，目前的正常读写case作为function的case，尝试发现这条路是行得通的，为后面JTAG使用TDL写case打下基础。
4. 使用脚本来将EDT转成TDL
5. 方案一：将EDT的Macro用TDL实现，也作为macro，用于TDL case
6. 方案二：将instruction/data作为参数，送给实现状态机的脚本，自动转成TDL语言。