看门狗简介：

在系统运行时，可能会在外界的干扰下，出现程序跑飞的情况，看门狗的出现就是为了解决这种故障。看门狗是一个独立于系统的计数器即**它的时钟和系统时钟不是同一个时钟**，在看门狗时钟的驱动下，计数器会不断地计数，并且**每隔一段时间会重置计数器（喂狗），然后重新开始计数，若没有接收到喂狗信号，说明系统出现故障**，就要发送给系统发送**复位信号，使系统重启。**

在由单片机构成的微型计算机系统中，由于单片机的工作常常会受到来自外界电磁场的干扰，造成各种寄存器和内存的数据混乱，会导致程序指针错误，不在程序区，取出错误的程序指令等，都有可能会陷入死循环，程序的正常运行被打断，由单片机控制的系统无法继续正常工作，导致整个系统的陷入停滞状态，发生不可预料的后果。

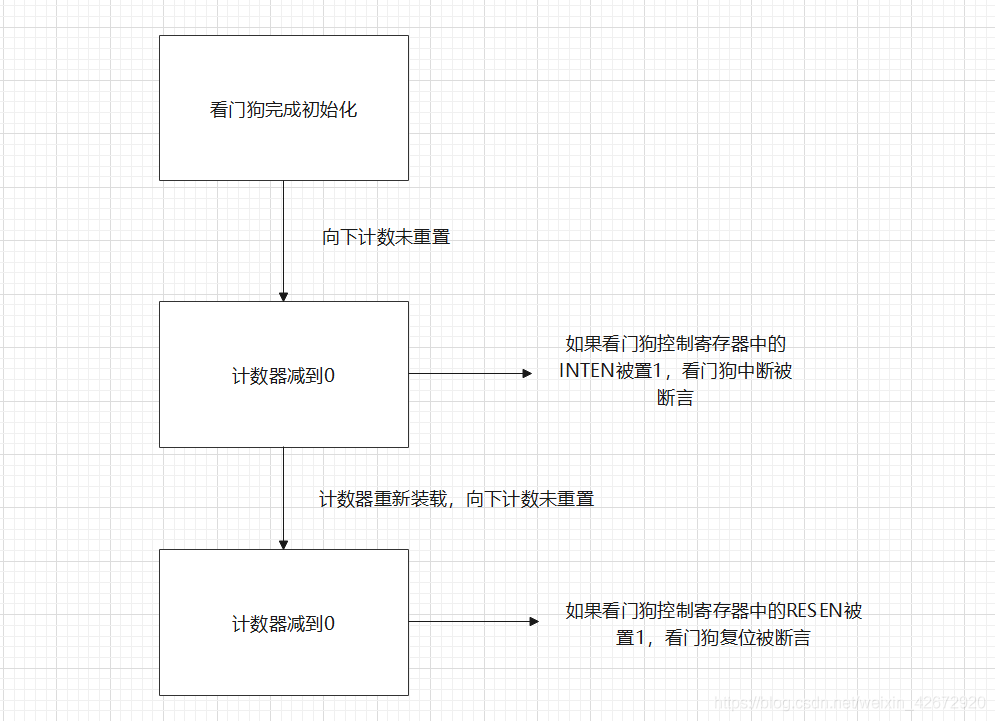
看门狗，又叫watchdog timer，是一个定时器电路，一般有一个**输入，叫喂狗**(kicking the dog/service the dog），一个**输出到MCU的RST端**，MCU正常工作的时候，每隔一段时间输出一个信号到喂狗端，给 WDT清零，如果超过**规定的时间不喂狗（一般在程序跑飞时），WDT定时超过，就会给出一个复位信号到MCU，使MCU复位。防止MCU死机. 看门狗的作用就是防止程序发生死循环，或者说程序跑飞。**

看门狗是一种[监控系统](https://baike.baidu.com/item/%E7%9B%91%E6%8E%A7%E7%B3%BB%E7%BB%9F/2354833?fromModule=lemma_inlink)的运行状况的手段，通过软硬件结合的方式实现对系统运行状况的监控。稳定运行的软件会在执行完特定指令后进行喂狗，若在一定周期内看门狗没有收到来自软件的喂狗信号，则认为系统故障，会进入中断处理程序或强制系统复位。系统上电后根据不同的工作模式可以选择使能看门狗的时机，若看门狗被使能则计数器开始计数，如果在设定的时间内没有及时喂狗则会发生看门狗超时。看门狗主要由寄存器、计数器和狗叫模块构成:通过寄存器对看门狗进行基本设置，计数器计算狗叫时间，狗叫模块决定看门狗超时后发出的中断或复位方式。[2]

一、watchdog简单功能介绍

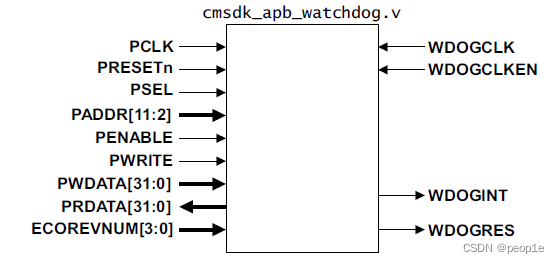
watchdog为一个32bit逐减计数器，由重载寄存器初始化，WDOGLOAD。当watchdog计数器减少到预计的数值时，输出端**WDPGINT将拉高，输出中断信号**。

在WDOGCLKEN信号为高时，**计数器会在每个WDOGCLK的上升沿减一**。计数到0时，若此时WDOGCONCROL寄存器部分的INT域与RES域为高则使能reset与interrupt功能（计数到0 ，计数停止，并且产生复位请求）。即输出中断信号并且watchdog会安装WDOGLAOD中预定值进行重置（在下一个使能的看门狗时钟沿，重装载[寄存器](https://so.csdn.net/so/search?q=%E5%AF%84%E5%AD%98%E5%99%A8&spm=1001.2101.3001.7020)使计数器重新装载数值，然后重新开始向下计数）。若WDOGCONTROL域中有未使能的情况则回等待重置或无法输出中断。



二、APB\_watchdog简单功能介绍

下图为apb\_watchdog接口，左侧接口为apb2总线接口（从信号方向可知其为从端），通过总线端的数据传输完成对watchdog的寄存器进行功能配置，右侧为watchdog本身接口。



信号描述：

WDOGCLK（input）

**看门狗模块的输入时钟，这个时钟必须和PCLK同步。**

WDOGCLKEN（input）

看门狗时钟的使能信号，其作用是使能看门狗输入时钟，只有在时钟使能信号WDOGCLKEN为1，并且在WDOGCLK的上升沿，计数器才会向下计数。

WDOGRESn（input）

看门狗时钟域的复位输入。

WDOGINT（output）

看门狗中断。

WDOGRES（output）

看门狗超时复位。

ECOREVNUM（input）

该输入连接到外设ID寄存器3中的ECO版本号（ ECO revision number ），以显示芯片设计过程ECO进程的版本更改。可以将此信号连接到低电平，或将其连接到特殊的连接单元，以便更改硅网表或更低级别（如硅掩模）的ECO修订号。

**和看门狗有关寄存器**

1、加载寄存器： WDOGLOAD **（读写寄存器）**

其作用是设置计数器初始值，只要写入新的初始值，计数会立刻重新开始计数，计数的最小值为1。

2、数值寄存器：WDOGVALUE**（只读寄存器）**

查看计数器当前值。

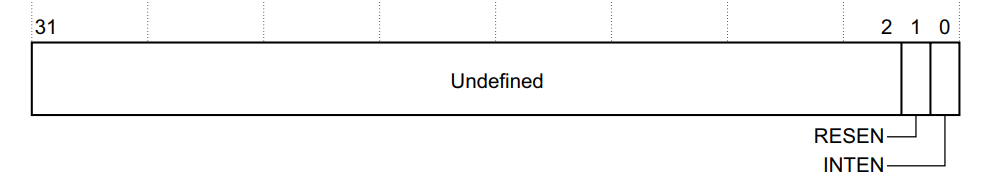
3、控制寄存器：WDOGCONTROL

其作用是能够让软件控制看门狗模块。

高30未保留，当作0来读取。

第1位：复位启用位，**置1时使能复位(RESEN)**，看门狗中断和看门狗复位都被使能。当选择这种模式时，看门狗计数器到达指定的值时，将置位其中断标志位 ，并产生看门狗中断请求，看门狗计数器到达零时复位微控制器。看门狗在到达设置值之前喂狗也会导致看门狗复位。

第0位：**中断启用位（INTEN）**，将会产生看门狗报警中断，但不会产生看门狗复位。当选择这种模式时，看门狗计数器到达指定的值时，将置位看门狗中断标志位 ，并产生看门狗中断请求。



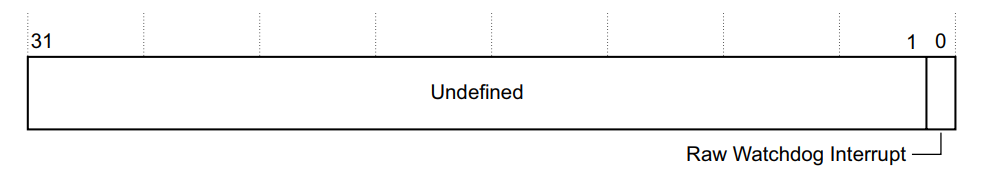
4、清除中断计数器：WDOGINTCLR**（只写寄存器）**

**写任何值到此寄存器，都会清除看门狗中断，然后从WDOGLOAD 那里重新加载数值并开始重新计数**。

5、初始中断状态寄存器：WDOGRIS（**只读寄存器）**

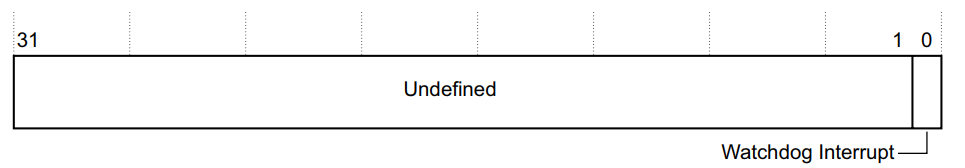
WDOGRIS寄存器，反映watchdog是否出现中断状态。即使INTEN为低，只要在之前的计数中完成计数即会拉高信号，该值与控制寄存器的**中断启用位INTEN**进行AND运算，以创建屏蔽中断，该屏蔽中断被传递到中断输出引脚。

第0位：计数器的原始中断状态



**6、中断状态寄存器**WDOGMIS**： (只读寄存器，反映中断情况)**

WDOGMIS寄存器用来查看计数器的屏蔽中断状态。该值是初始中断状态与控制寄存器的INTEN位的逻辑与，该值与传递到中断输出引脚的值相同。  
第0位，用来启用计数器中断

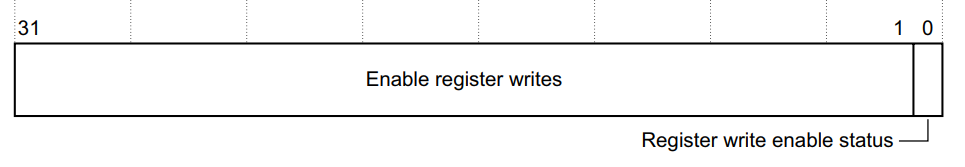


7、锁定寄存器：WDOGLOCK

用来禁用对所有其他寄存器的写入访问，目的是为了防止恶意软件禁用看门狗功能。**写入值0x1ACCE551可启用对所有其他寄存器的写入访问（解锁）**。写入任何其他值都会禁用写入访问。从该寄存器读取仅返回到最低位：

最低位为0：访问已启用，未锁定。（默认）

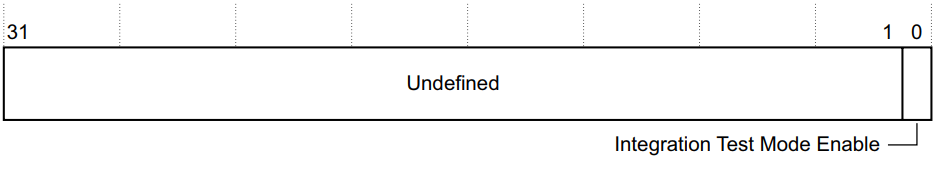
最低位为1：访问禁止，已锁。



8、**集成测试控制寄存器：WDOGITCR（地址0000\_0f00）**

用来启用集成测试模块，在此模式下，测试输出寄存器直接控制屏蔽中断输出**WDOGINT和复位输出**，

第0位：**置1时，处于集成测试模式**

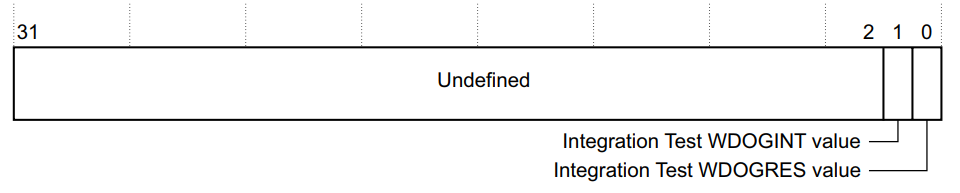


9、**集成测试输出置位寄存器：WDOGITOP(地址 0000\_0f04)**

当WDOGITOP寄存器处于集成测试模式时，该寄存器中的值直接驱动启用的中断输出和复位输出。

**第1位：处于集成测试模式时，WDOGINT上的值输出。**

**第0位：处于集成测试模式时WDOGRES上的值输出**。



**集成测试控制寄存器：WDOGITCR 置1后，处于集成测试模式，WDOGINT、WDOGRES信号直接由WDOGITOP寄存器低两位值控制。**

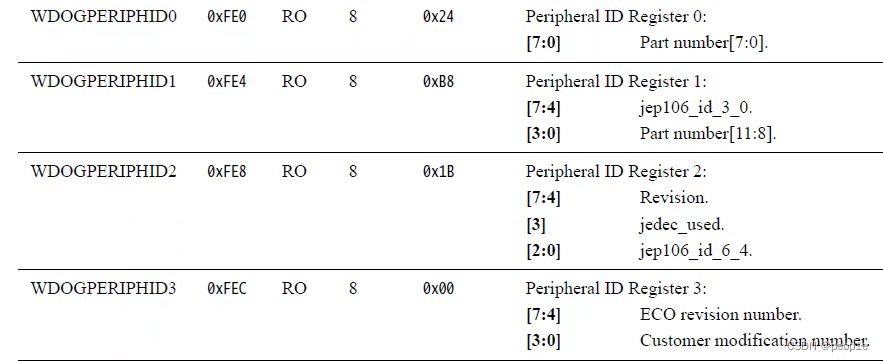
三、寄存器测试：

直接访问的方式就是调用vip中的apb2测试序列对总线进行访问以完成寄存器的配置，而register model访问的方式则是在验证环境中创建一个寄存器模型，通过adapter将其与总线连接起来，用户可以通过寄存器模型来配置硬件寄存器的值。下面将介绍二者在watchdog验证中的具体实现方法。

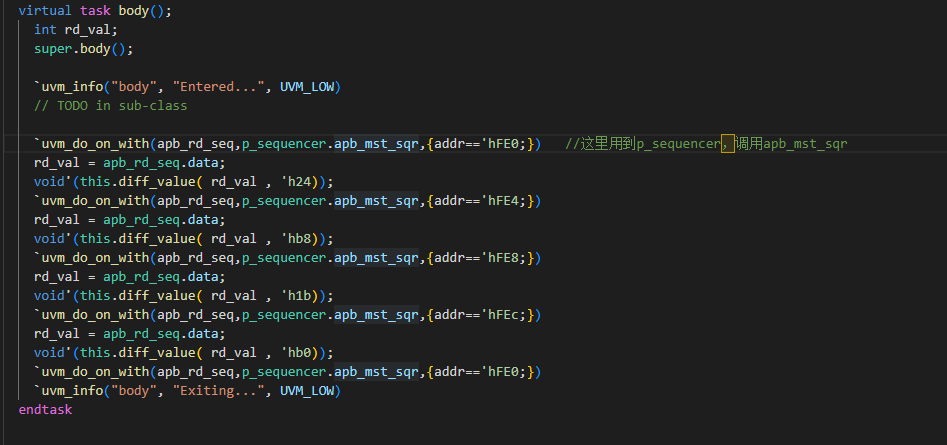
**一、apb直接访问（rkv\_watchdog\_apbacc\_test）**

apb直接访问就是直接在对应的seq中调用apb\_seq，挂载到env中与apb\_mst\_sequencer连接好的sequencer上，进行激励的发送。需要创建对应的seq与test文件，并将它们加入seq\_lib与test\_lib的编译文件中。

对于apb直接访问，我们可以**直接读取部分寄存器reset之后的数据**，如下代码读取的是**WDOGPERIPHID0到WDOGPERIPHID4的数据，其地址与reset之后的信号值如下：reset信号应分别为24，b8，1b与b0**。

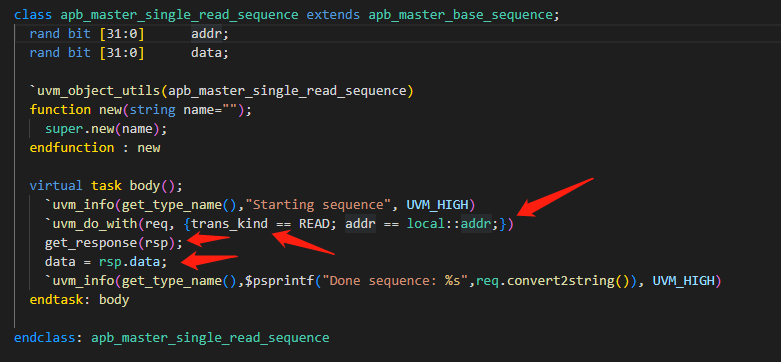


1、apbacc\_virtual\_sequence侧代码：

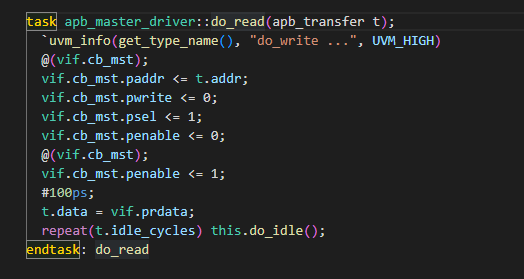


apb\_rd\_seq是访问之后返回的读取数据，其具体代码在vip中：

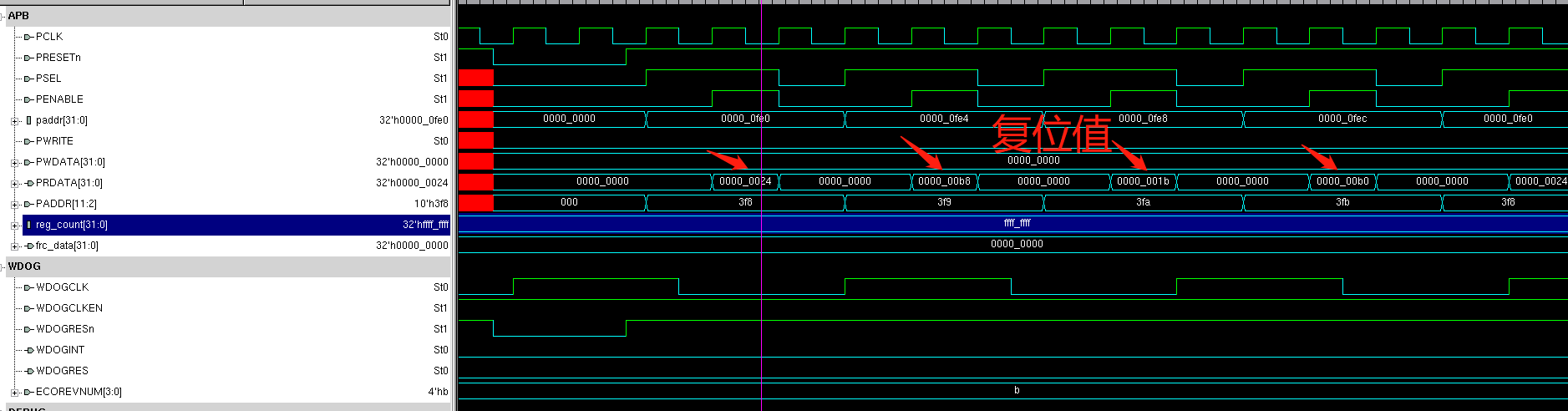
vip中的apb\_master\_seq\_lib（**为什么apbacc\_virtual\_sequence能直接调用apb\_read\_sequence，因为apbacc\_virtual\_sequence继承于rkv\_watchdog\_base\_virtual\_sequence，而rkv\_watchdog\_base\_virtual\_sequence里面集成了所有的element\_sequence和apb\_vip的所有sequence。**



Driver中：



波形图：



**二、寄存器模型访问**

（rkv\_watchdog\_regacc\_test）

寄存器模型的生成来源于python脚本与csv文件。本文中不介绍其生成具体过程，直接介绍模型的引入和在验证环境中对寄存器模型进行调用。自动生成的寄存器模型只提供了前门访问方案，本文的访问也采用前门访问。

1.引入registermodel

register model引入环境与cfg类似，也是**由test传递至env层次**中。需要考虑的是如何在对应的seq中调用register model（rgm），这里采用的方法是**将rgm传递至对应的sequencer中，在sequence挂载到sequencer之后，seq可以在body中拿到对应sequencer下的rgm，从而在sequence中实现寄存器模型的访问。**

因此第一步是将test层面的rgm传递至env中

function new (string name = "**rkv\_watchdog\_base\_test**", uvm\_component parent);

super.new(name, parent);

**rgm = rkv\_watchdog\_rgm::type\_id::create("rgm", this);**

rgm.build();

endfunction

function void build\_phase(uvm\_phase phase);

super.build\_phase(phase);

cfg = rkv\_watchdog\_config::type\_id::create("cfg");

uvm\_config\_db#(rkv\_watchdog\_config)::set(this,"env","cfg",cfg);

**uvm\_config\_db#(rkv\_watchdog\_rgm)::set(this,"env","rgm",rgm);**

env = rkv\_watchdog\_env::type\_id::create("env", this);

endfunction

2、env层次代码：

function void build\_phase(uvm\_phase phase);

super.build\_phase(phase);

apb\_mst = apb\_master\_agent::type\_id::create("apb\_mst", this);

if(!uvm\_config\_db#(rkv\_watchdog\_config)::get(this, "", "cfg",cfg)) begin

`uvm\_fatal("GTCFG","cannot get cfg from configdb")

end

if(!uvm\_config\_db#(virtual rkv\_watchdog\_if)::get(this, "", "vif",cfg.vif)) begin

`uvm\_fatal("GTCFG","cannot get vif from confgdb")

end

virt\_sqr = rkv\_watchdog\_virtual\_sequencer::type\_id::create("virt\_sqr", this);

uvm\_config\_db#(rkv\_watchdog\_config)::set(this,"virt\_sqr","cfg",cfg);

uvm\_config\_db#(apb\_config)::set(this,"apb\_mst","cfg",cfg.apb\_cfg);

if(!uvm\_config\_db#(rkv\_watchdog\_rgm)::get(this, "", "rgm",rgm)) begin

**rgm = rkv\_watchdog\_rgm::type\_id::create("rgm", this);**

**rgm.build();**

end

**uvm\_config\_db#(rkv\_watchdog\_rgm)::set(this,"\*","rgm", rgm);**

adp = rkv\_watchdog\_adapter::type\_id::create("adp", this);

predictor = uvm\_reg\_predictor#(apb\_transfer)::type\_id::create("predictor", this);

endfunction

function void connect\_phase(uvm\_phase phase);

super.connect\_phase(phase);

virt\_sqr.apb\_mst\_sqr = apb\_mst.sequencer;

**rgm.map.set\_sequencer(apb\_mst.sequencer, adp);**

apb\_mst.monitor.item\_collected\_port.connect(predictor.bus\_in);

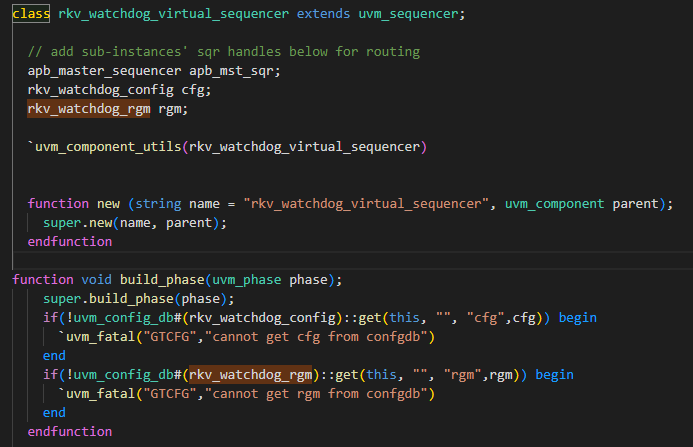
**predictor.map = rgm.map;**

predictor.adapter = adp;

endfunction

此中不但实现了rgm的配置并传递于下级，也实现了predictor与adapter的声明，**predictor为uvm提供的explicit predictor，只需声明例化即可使用**。adapter的具体代码会在后文提供。同时对predictor与adapter进行了连接，使其能够在rgm中发挥作用。

3、virtual\_sequencer



这样，rgm就从test一路传递至sequencer中，在base sequence中也能够拿的对应的rgm.

**4、regacc sequence**

最后，完成通过寄存器模型访问的sequence就可以通过验证环境中的寄存器模型完成watchdog中硬件寄存器的配置了。

`ifndef RKV\_WATCHDOG\_REGACC\_VIRT\_SEQ\_SV

`define RKV\_WATCHDOG\_REGACC\_VIRT\_SEQ\_SV

class **rkv\_watchdog\_regacc\_virt\_seq** extends rkv\_watchdog\_base\_virtual\_sequence;

`uvm\_object\_utils(rkv\_watchdog\_regacc\_virt\_seq)

function new (string name = "rkv\_watchdog\_regacc\_virt\_seq");

super.new(name);

endfunction

virtual task body();

int rd\_val;

uvm\_status\_e status = UVM\_IS\_OK;

super.body();

`uvm\_info("body", "Entered...", UVM\_LOW)

// TODO in sub-class

**rgm.WDOGPERIPHID0.read(status, rd\_val);**

**void'(this.diff\_value( rd\_val , 'h24));**

**rgm.WDOGPERIPHID1.read(status, rd\_val);**

**void'(this.diff\_value( rd\_val , 'hb8));**

**rgm.WDOGPERIPHID2.read(status, rd\_val);**

**void'(this.diff\_value( rd\_val , 'h1b));**

**rgm.WDOGPERIPHID3.read(status, rd\_val);**

**void'(this.diff\_value( rd\_val , 'hb0));**

`uvm\_info("body", "Exiting...", UVM\_LOW)

endtask

endclass

`endif

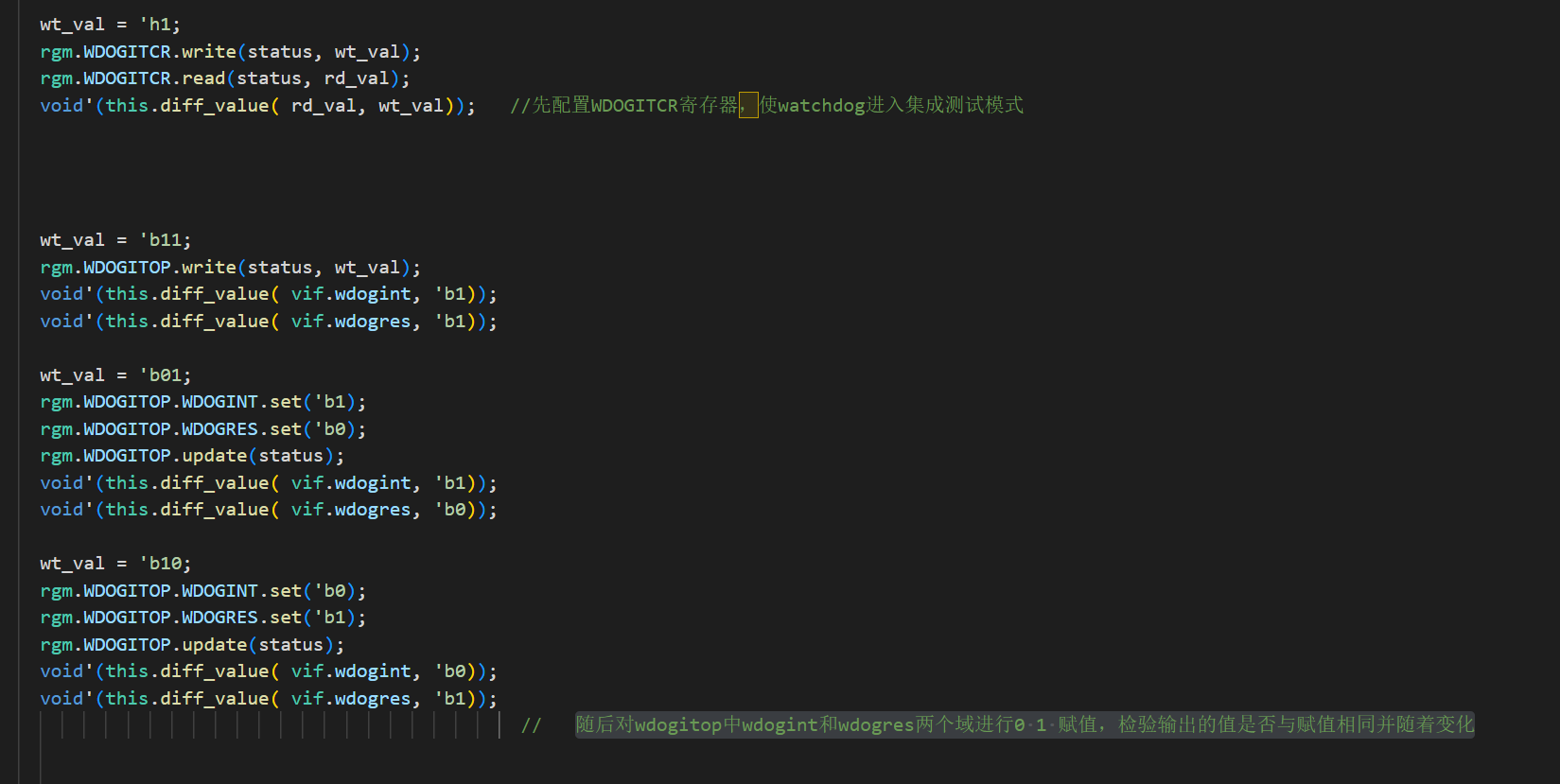
**与直接访问相比，寄存器模型访问的方式将硬件寄存器通过软件的方式进行了封装，不再需要查询每个寄存器的地址来进行配置。同时，也可以针对某个寄存器的域来进行配置，增加了配置的灵活程度。**

**三、集成模式测式**

**（rkv\_watchdog\_integration\_test）**

**集成测试模式在打开（/先配置WDOGITCR寄存器为1），使watchdog进入集成测试模式，之后WDOGINT与WDOGRES信号直接由WDOGITOP寄存器的低两位值来控制，集成测试的开关在寄存器WDOGITCR中，其最低位为高则集成测试模式打开。**

**由于之前已经实现apb寄存器访问，可通过此来直接实现集成测试。集成测试的sequence代码如下。**

****

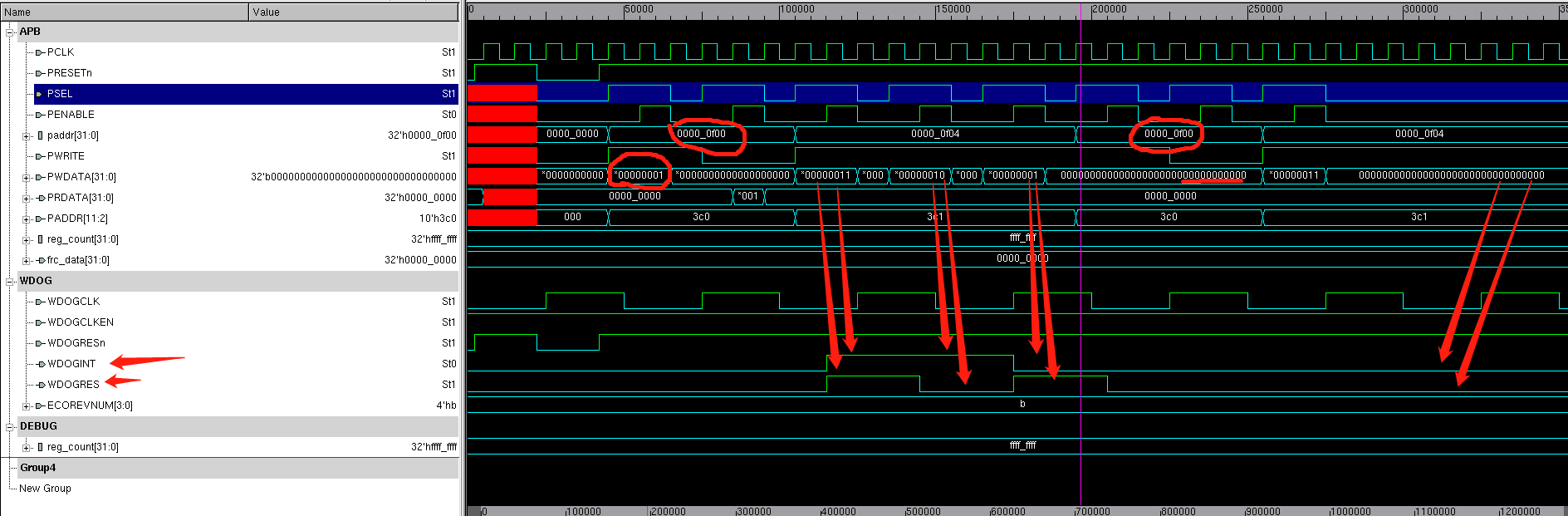
**// 这里WDOGITCR是读写寄存器，故可以对其进行读写操作，而WDOGITOP为只读寄存器，只能对其进行写操作。**

**//而功能验证则通过检查返回的信号是否与赋值相同实现即可。**

**//只读”的意思是在其工作时只能读出，不能写入。然而其中存储的原始效据，必须在它工作以前与入。**

**//只读存储器由寸工作可靠，保密性强，在计算机系统中得到广泛的应用**

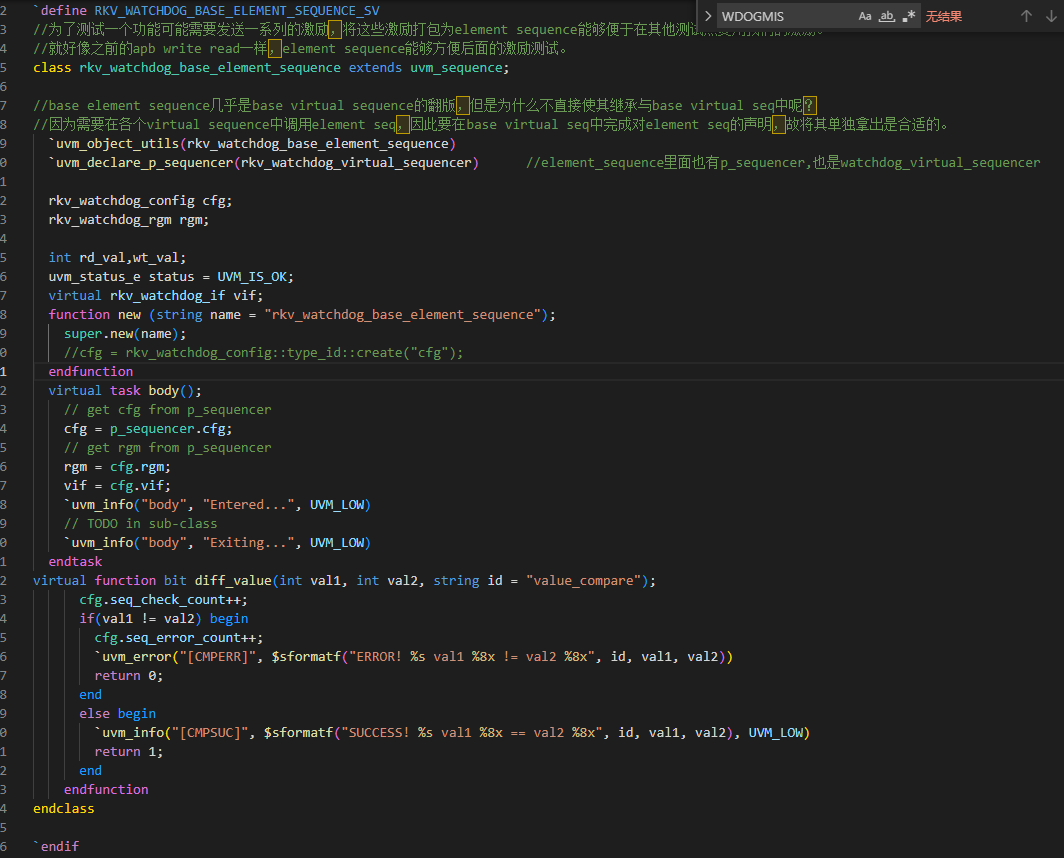
**波形图：从波形上看集成测试没有问题，输出的int和rst值与配置给寄存器的是对应变化的，**

****

四、element sequence 的创建

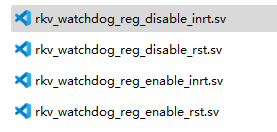
为了测试一个功能可能需要发送一系列的激励，将这些激励打包为element sequence能够便于在其他测试点复用我们的激励。就好像之前的apb write 、read一样，element sequence能够方便后面的激励测试。

1、base element sequence几乎是base virtual sequence的翻版，但是为什么不直接使其继承与base virtual seq中呢？因为**需要在各个virtual sequence中调用element seq，因此要在base virtual seq中完成对element seq的声明，故将其单独拿出是合适的**。base element seq代码如下：

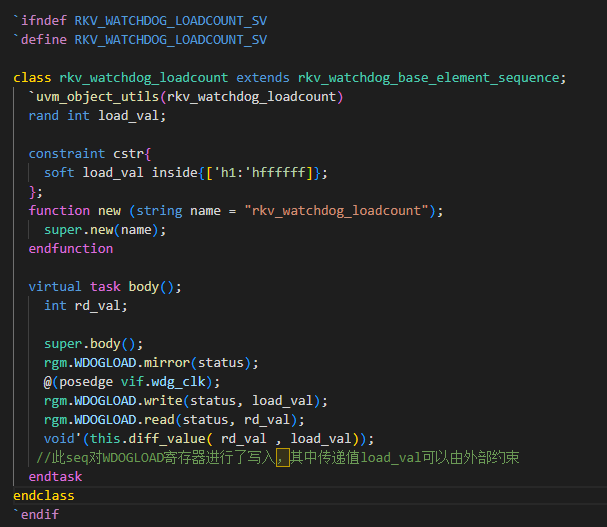


以下这些reg\_sequence都继承于**rkv\_watchdog\_base\_element\_sequence**

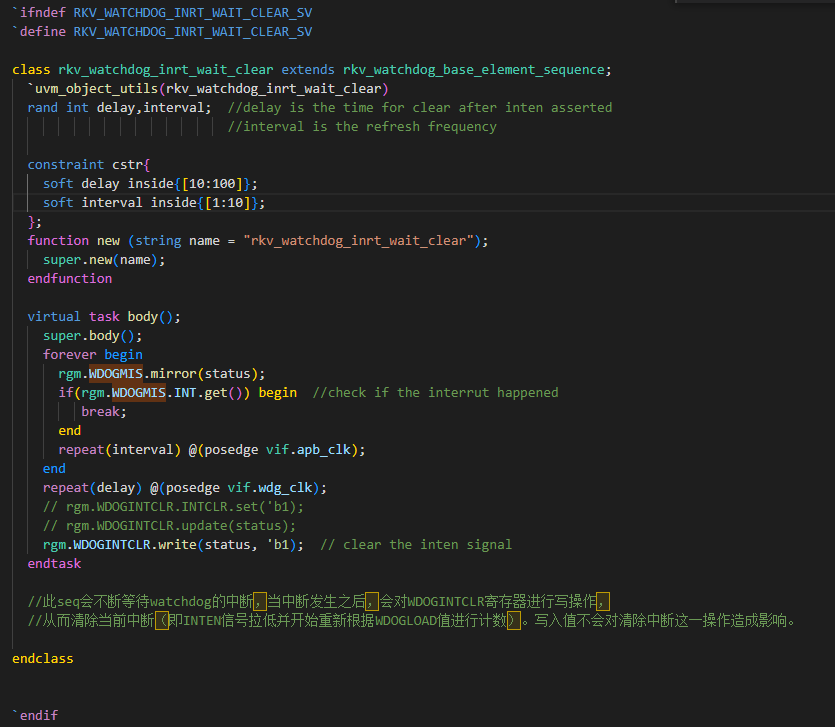
主要是控制寄存器：WDOGCONTROL的**中断“INTEN” 和复位”RESEN”的使能与不使能（关闭）sequence。**



2、loadcount sequence加载寄存器序列 （传递值load\_val可以由外部约束控制）



3、wait interrupt clear sequence 清除中断

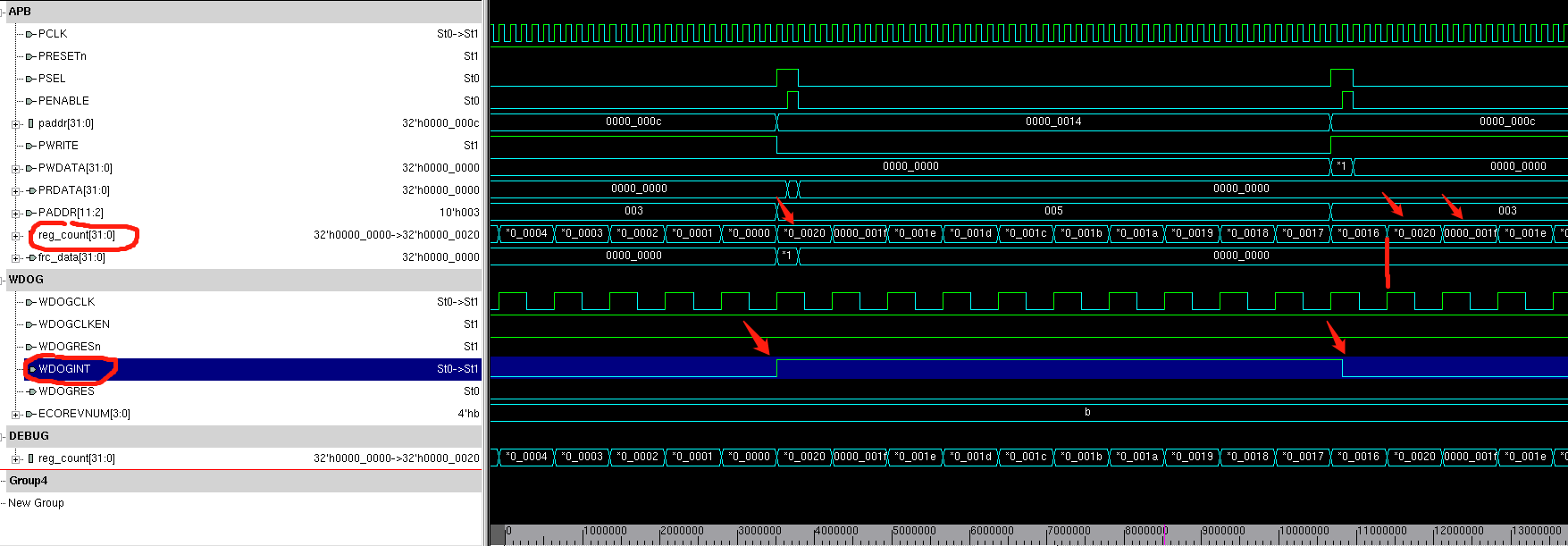


/此seq会不断等待watchdog的中断，当中断发生之后，会对WDOGINTCLR寄存器进行写操作，从而清除当前中断（即INTEN信号拉低并开始重新根据WDOGLOAD值进行计数）。写入值不会对清除中断这一操作造成影响。

五、计数、中断、上锁测试：

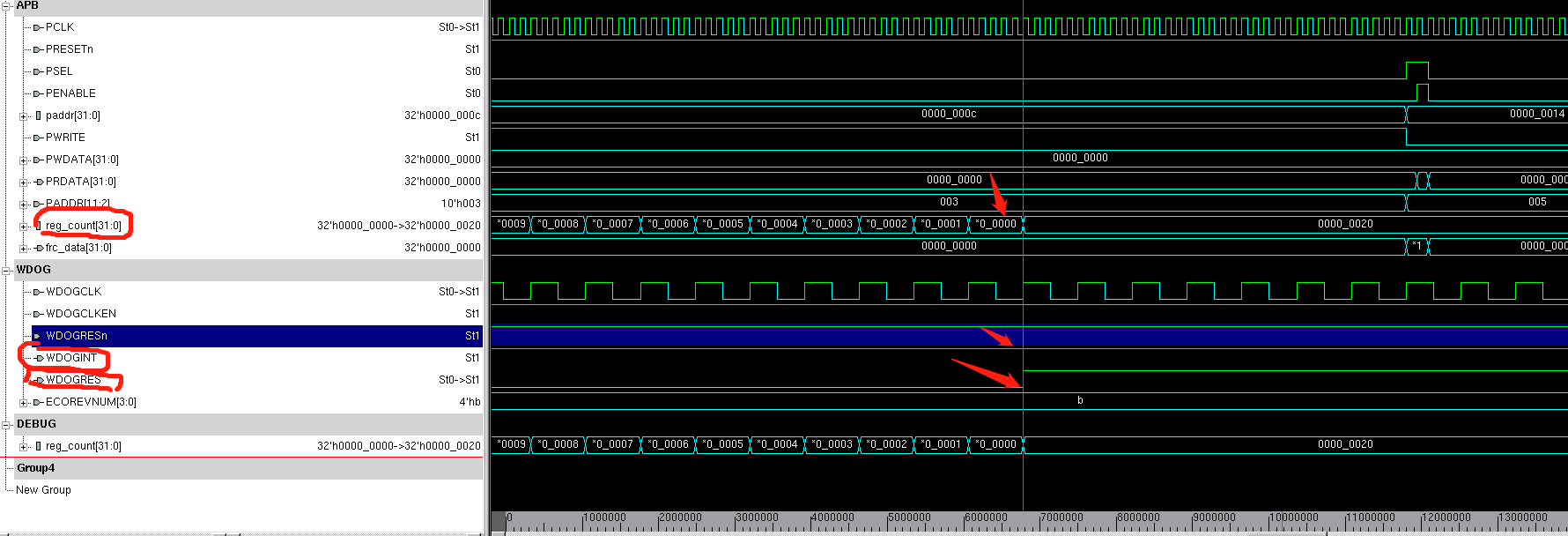
**1、计数与复位使能测试（rkv\_watchdog\_countdown\_test）**

**对watchdog配置’h20 开始计数**



当计数到0000\_0000时，WDOGINT中断信号会拉起。

在看门狗时钟的驱动下，计数器会不断地计数，并且**每隔一段时间会重置计数器（喂狗），就是中断信号拉高，中断信号拉低以后，在WDOGCLK的下一时钟周期重新开始计数。**

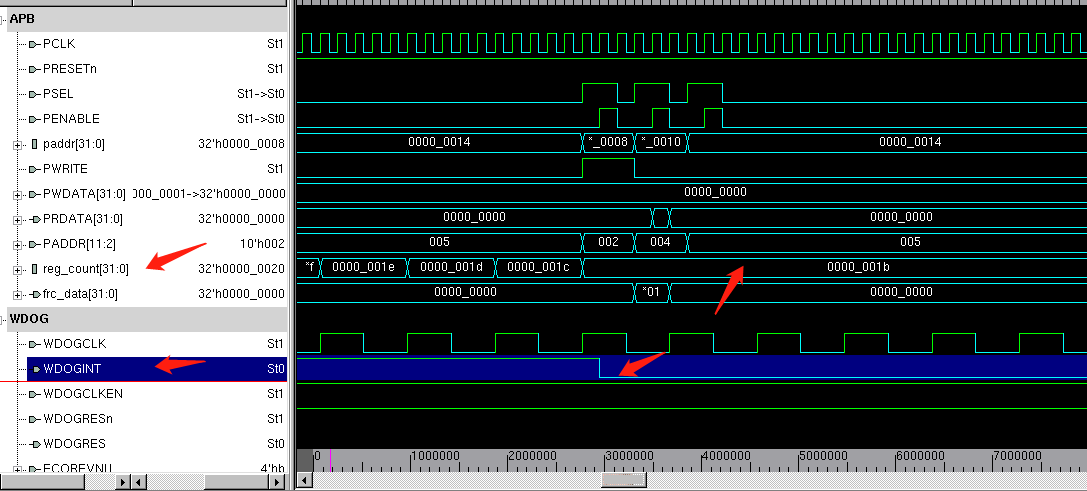


**若没有接收到喂狗信号，说明系统出现故障**，就要发送给系统发送**复位信号，使系统重启。**

**（这里就是计数到0000\_0000，本来中断信号应该拉起，但是没有，说明系统故障，所以复位信号拉起，所以计数器一直停在了‘h20 ）**

**2、中断功能关闭测试：（rkv\_watchdog\_disable\_inrt\_test）**

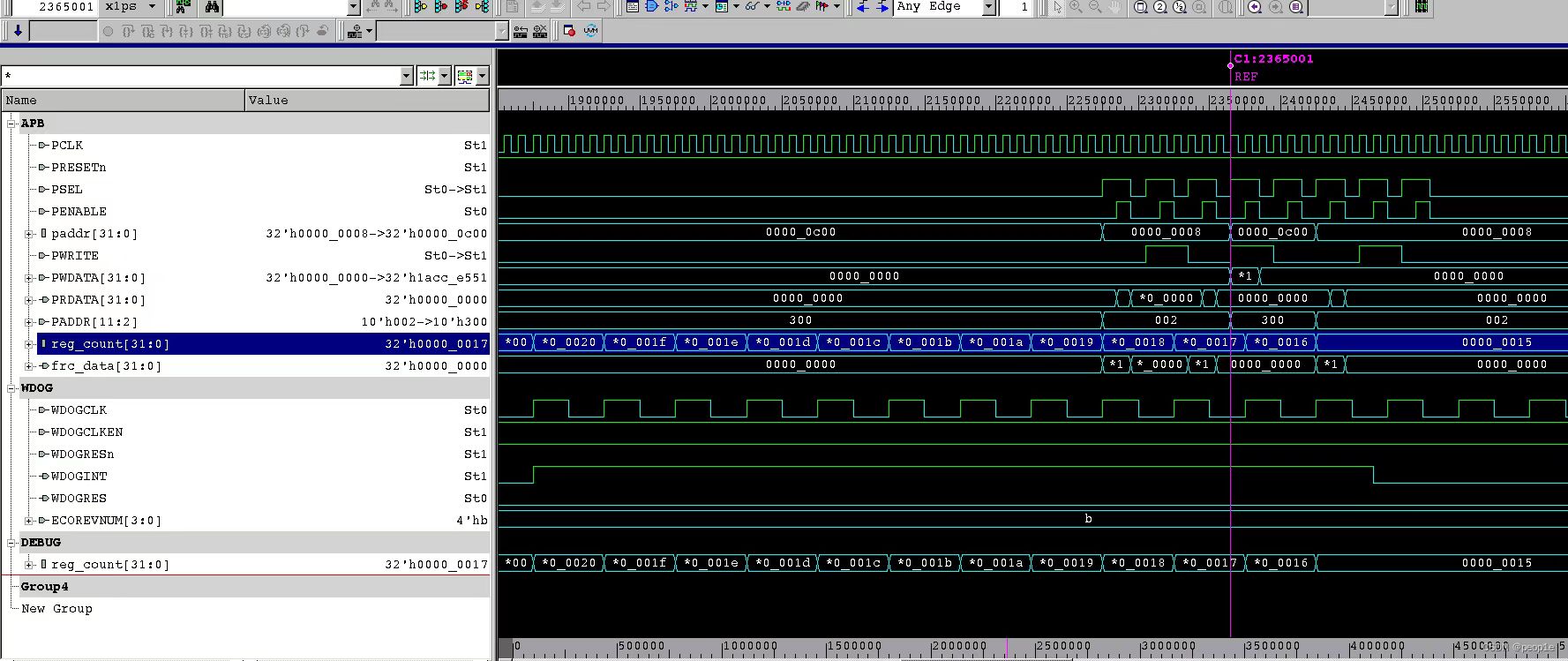
中断功能关闭，wdogint拉低之后未进行计数。



**2、上锁测试（rkv\_watchdog\_lock\_test）**

为了测试watchdog的上锁功能：即在写入除’h1acce551的其他值都将锁定watchdog，使其寄存器无法被访问。

测试的方法是先默认情况下配置watchdog寄存器使其开始计数并检查其上锁情况，随后对其进行锁定与解锁的检查，check\_unlock\_control\_status这一任务的逻辑是，通过读出控制寄存器的值，对控制寄存器进行与读出值相反的写入，最后读出，检查写入是否成功。成功则为非锁定状态，失败则说明处于锁定状态。

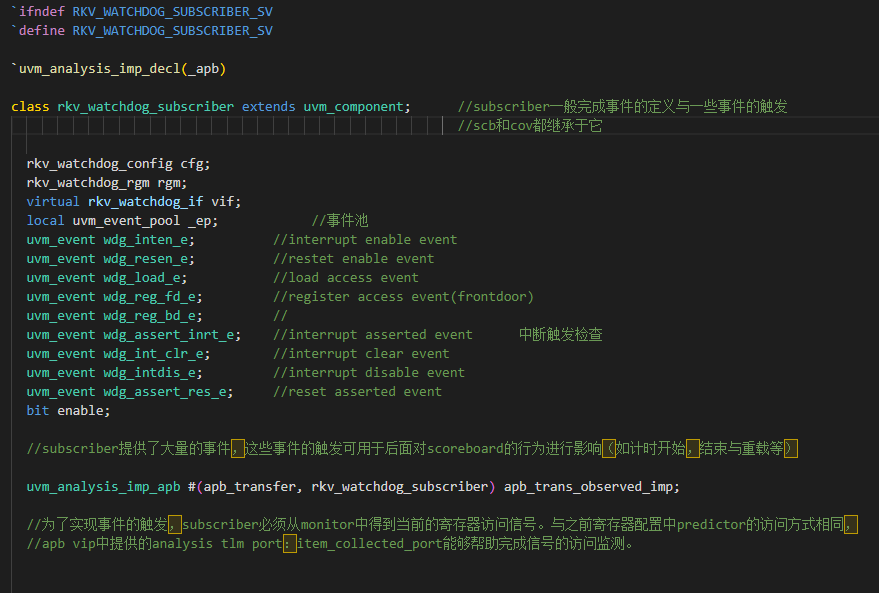


**计数完成后，只有在unlock的情况下才能将INTEN信号拉低。同时仿真无报错，每次lock check都成功**

六、事件触发器与计数检查器

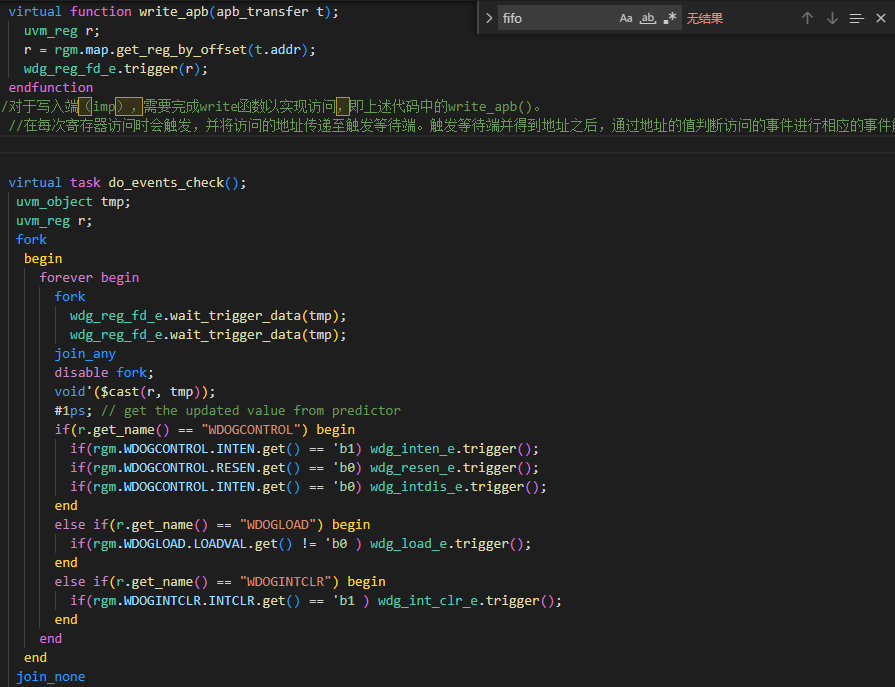
**1、事件触发器**

**事件触发器subscriber是指在环境中定义的事件触发模块，无论是scoreboard检查器还是覆盖率收集都需要基于一些事件的触发来进行功能比较与覆盖率收集。因此比较方便的做法是定义一个subscriber来完成事件的定义与一些事件的触发，而scoreboard与coverage模块继承于此父类，从而可以使用其中的事件与方法。**

****

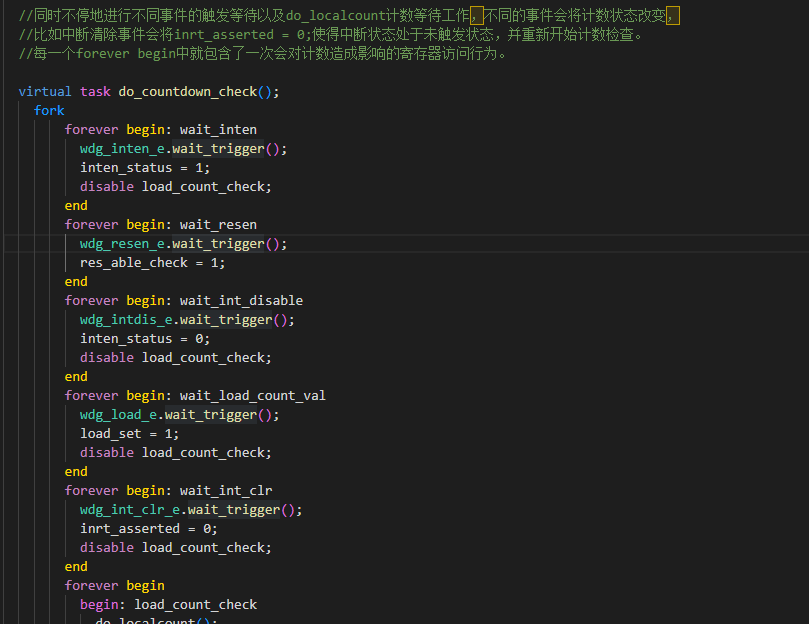
subscriber中除了定义所需的interface，register model与config之外。提供了大量的事件，这些事件的触发可用于后面对scoreboard的行为进行影响（如计时开始，结束与重载等）。

同样的，为了实现事件的触发，subscriber必须从monitor中得到当前的寄存器访问信号。与之前寄存器配置中predictor的访问方式相同，apb vip中提供的analysis tlm port：item\_collected\_port能够帮助完成信号的访问监测**。对于写入端（imp），需要完成write函数以实现访问，即上述代码中的write\_apb()。在每次寄存器访问时会触发，并将访问的地址传递至触发等待端。触发等待端并得到地址之后，通过地址的值判断访问的事件进行相应的事件触发。**

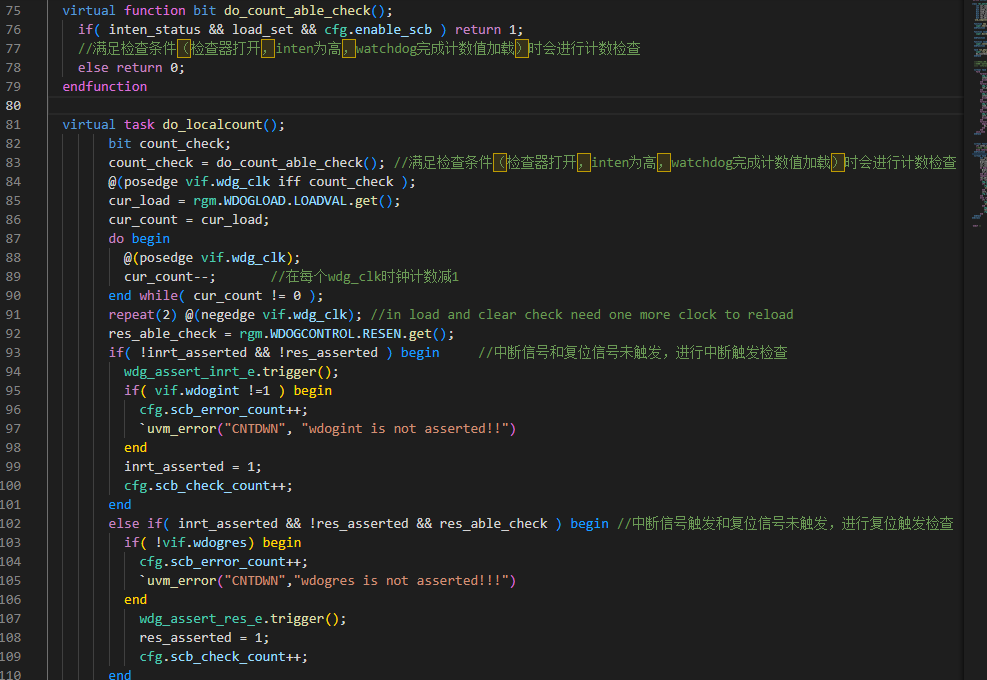
****

**2、scoreboard**

**scoreboard继承于subscriber，能够完成计数检查，并在不同事件触发时开始，结束或者中断计数的触发。**

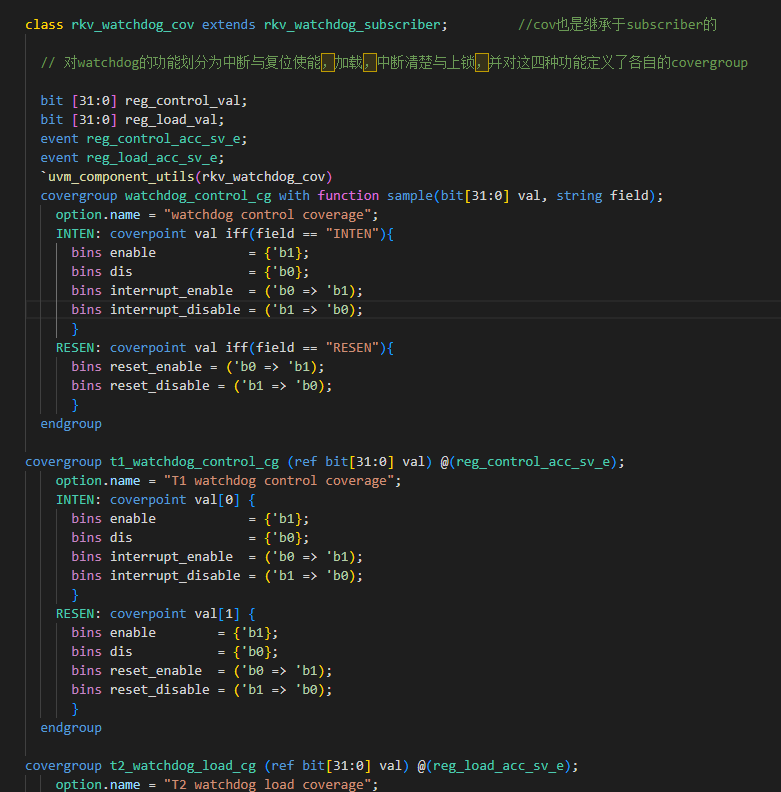
****

**do\_countdown\_check的逻辑是同时不停地进行不同事件的触发等待以及do\_localcount计数等待工作，不同的事件会将计数状态改变，比如中断清除事件会将inrt\_asserted = 0;使得中断状态处于未触发状态，并重新开始计数检查。每一个forever begin中就包含了一次会对计数造成影响的寄存器访问行为。**

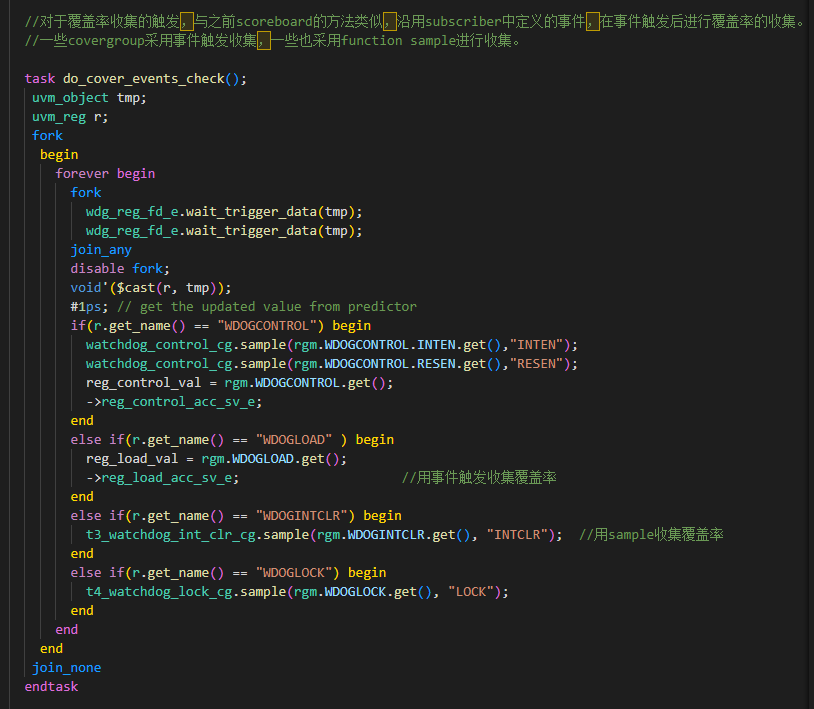
****

**在scoreboard中对计数进行检查与事件触发监测。这里对do\_localcount的逻辑功能进行简单介绍，在满足检查条件（检查器打开，inten为高，watchdog完成计数值加载）时会进行计数检查，计数检查完成之后，若中断信号未触发且watchdog复位信号未触发时，会进行中断触发检查。若中断已经触发且watchdog复位信号也未触发且watchdog复位使能为高时，会进行复位触发检查。**

七：覆盖率模块



对watchdog的功能划分为中断与复位使能，加载，中断清楚与上锁，并对这四种功能定义了各自的covergroup。

****

**对于覆盖率收集的触发，与之前scoreboard的方法类似，沿用subscriber中定义的事件，在事件触发后进行覆盖率的收集。一些covergroup采用事件触发收集，一些也采用function sample进行收集。**