# Tutorial VCS

23 de agosto de 2013

## 1 Introdução

Neste tutorial será abordado como utilizar o software VCS para simular sistemas digitais em verilog e comparar os resultados com o de um modelo do sistema escrito em C. Para isso será utilizado Verilog PLI (Programming Language Interface) como mecanismo para invocar funções em C e C++ a partir de um código verilog. Como exemplo será usado um sistema digital que implementa uma SAD (soma das diferenças absolutas).

## 2 Verilog PLI

Através do PLI temos acesso a um conjunto de rotinas em linguagem C para acessar objetos declarados em Verilog. Estas rotinas podem ser divididas em três grandes grupos:

- Access Routines - Utilizadas para poder ler/escrever em objetos declarados em Verilog, como registers e variáveis(integer, real, time).
- Utility Routines - Rotinas diversas. Podem ser usadas para obter informações de tempo, exibir mensagens, terminar a simulação...
- VPI Routines - Com o PLI 2.0, access routines e utility routines foram combinadas em vpi routines.

Neste tutorial a maioria das rotinas utilizadas serão de acesso.

Abaixo será explicado alguns conceitos importantes para utilização do PLI, além da descrição de algumas das rotinas que serão usadas.

## 2.1 Chamar funções em C dentro do código Verilog

A chamada das funções em C podem ser feitas com a linha abaixo, na qual os argumentos são os objetos do código verilog(registers, wires...) que desajamos poder acessar dento do código em C.

$nome_da_função(Arg1,Arg2...)

## 2.2 Handles

Para cada objeto do código Verilog que precise ser acessado dentro do código em C deve ser criado um handle. Cada handle carrega informações sobre onde encontrar dados relativos ao objeto que ele se refere. Para criar um handle basta a seguinte linha:

handle nome_do_handle;

Para atribuir um handle a um objeto em específico deve se utilizar a access routine abaixo, na qual o número serve para identificar se é o Arg1, Arg2 da seção 2.1.

nome_do_handle = acc_handle_tfarg(1);

## 2.3 Outras Rotinas

### 2.3.1 acc_initialize

acc_initialize();

Inicializa o ambiente para as rotinas "acc". Deve ser chamada antes de utilizar qualquer outra rotina "acc".

### 2.3.2 acc_vlc_add

acc_vlc_add(object_handle, consumer_routine, user_data, vcl_flag);

Chama a consumer_routine sempre que valor do objeto dado pelo object_handle tiver seu valor lógico alterado.

### 2.3.3 acc_close

acc_close();

Efeito contrário à rotina acc_initialize. Libera a memória interna utilizada pelas rotinas "acc".

### 2.3.4 acc_set_value

acc_set_value(object_handle, &value_s, &delay_s);

Rotina utilizada para escrever um valor em um objeto verilog. O valor lógico a ser escrito e o delay para propagação são colocados em 2 estruturas referenciadas pelos seus ponteiros. Por isso a rotina acima só pode ser usada após o código abaixo.

```
//value_s é um ponteiro para a structure type s_setval_value
s_setval_value value_s;
//delay_s é um ponteiro para a structure type s_setval_delay
s_setval_delay delay_s;
//Define-se qual o formato em que os valores serão fornecidos.
value_s.format = formato;
//Define-se o tipo de delay
delay_s.model = delay;
//Define qual valor será colocado no objeto verilog.
value_s.value.str = valor;
```

O formato pode ser: accBinStrVal, accOctStrVal, accDecStrVal, accHexStrVal, accScalarVal, accIntVal, accRealVal, accStringVal, accVectorVal.

O delay pode ser: accNoDelay, accInertialDelay, accTransportDelay, accPureTransportDelay, accForceFlag, accReleaseFlag.

O tipo de variável usada para o valor deve estar de acordo com o formato escolhido.

### 2.3.5 acc_fetch_value

char * nome_string = acc_fetch_value(object_handle, format_string, value);

Usada para ler o valor de algum objeto verilog(registers, integers...).

O argumento format_string escolhe a forma como o valor deve ser lido, para binário usar %b, para decimal %d, para hexadecimal %h.

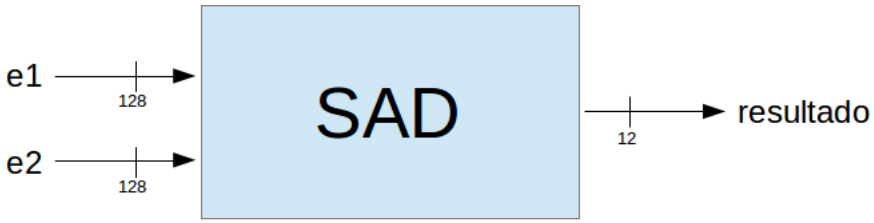


Figura 1: SAD

O argumento value é um ponteiro para uma structure, e deve ser fornecido antes através da seguinte linha:

p_acc_value value;

#### 2.3.6 io_printf

io_printf();

Funciona da mesma forma que a rotina em C printf.

# 3 Revisão SAD

A SAD é um algorítimo utilizado para medir o nível de similaridade entre dois blocos de imagem. O funcionamento consiste em somar as diferenças absolutas entre os pixels correspondentes entre estes blocos.

Neste exemplo utilizaremos uma SAD que compara dois blocos com 16 pixels, com cada pixel representado em 8 bits. A figura 1 mostra um esquema da SAD, a qual recebe duas entradas de 128 bits (cada entrada recebe todos os 16 pixels de um bloco de imagem concatenados), e uma saída resultado com 12 bits.

# 4 Detalhes sobre o Exemplo

Neste exemplo faremos com que tanto a função que gera vetores de entrada rândomicos para SAD, quanto o cálculo do valor esperado no resultado (Modelo) sejam descritos em linguagem C. Em Verilog será descrito o sistema digital e o testbench.

# 5 Arquivos Necessários

Para utilizar o VCS a pasta do projeto deve conter os seguintes arquivos:

- Modelo do Sistema digital em Verilog (ex.: SAD.v, difAbs.v)
- Testbench em Verilog (ex.: testbenchSAD.v)
- Arquivo com rotinas em C (ex.: SAD.c)
- Arquivo pli.tab
- Arquivo "filelist"

Neste tutorial será abordado o conteúdo dos quatro últimos

## 5.1 Testbench em Verilog

O arquivo de testbench em verilog funciona como um ambiente de simulação para o sistema digital que está sendo testado. Através dele é possível aplicar vetores de teste nas entradas, monitorar as saídas e comparar os resultados com valores esperados.

A primeira linha do arquivo deve conter imformações sobre a referência de tempo para a simulação. Isso é feito através da seguinte declaração:

```
`timescale 1ns/10ps
```

Na linha acima a primeira escala é utilizada como tempo de referência, de forma que cada unidade de tempo representará 1ns, fazendo com que uma declaração #20 seja entendida como 20ns. Já a segunda escala é utilizada como sendo a precisão máxima, escolhendo 10ps fazemos com que a menor variação de tempo que pode ser especificada no código verilog seja #0.01.

O próximo passo é declarar registradores para cada uma das entradas do DUT(Design Under Test), neste caso a SAD, e wires para cada uma das saídas, e instanciar o DUT.

```
module testbenchSAD;

        reg [127:0] e1, e2;
        wire [11:0] result;
```

```
        SAD SAD1 (
             .pA (e1),
             .pB (e2),
             .SAD (result));

endmodule
```

Agora vem a parte principal do arquivo, onde é criado um loop com iterações espaçadas em 1ns, e tambem onde é feita a chamada da função em C.

```
   initial begin
      //Chamada da função em C
      $sad_monitor(testbenchSAD.e1, testbenchSAD.e2,
                   testbenchSAD.result, testbenchSAD.i);
      //$monitor é utilizado  para imprimir no terminal
      //qualquer alteração nos registradores e1 e e2.
      //%h é usado para que os valores sejam impressos em
      //hexadecimal
      $monitor("e1 = %h,e2 = %h", e1, e2);

      //O número de iterações do loop abaixo define
      //quantos vetores de entrada serão testados.
      for (i=0; i < 50; i=i+1)
        begin
           #1;
        end
      //$finish termina a simulação
      $finish;
   end
```

## 5.2 Arquivo com rotinas em C

No inicio do arquivo são adicionados os includes, no qual acc_user.h é necessário para poder usar as access routines.

```
#include "acc_user.h"
#include <stdio.h>
```

Logo após são declarados os handles e as funções.

```
handle e1;
handle e2;
handle result;
handle i;

void sad();
void sad_entradas();
int converter(string in_string, int no_bits);
```

Neste arquivo serão criadas 4 rotinas:

### 5.2.1 sad_monitor

Esta é a rotina chamada pelo código Verilog. A sua função consiste em monitorar alguns dos objetos do código verilog e chamar outras rotinas caso estes objetos monitorados tenham seu valor alterado.

A rotina sad_entradas é chamada sempre que o inteiro i altera, garantindo que a cada nova iteração do loop no arquivo Verilog, novos vetores de entrada são criados no arquivo C.

A rotina sad é chamada sempre que result altera, garantindo que seja feito o cálculo pelo modelo e comparado o resultado cada vez que o sistema digital gera um resultado.

Como é a rotina sad_monitor que é chamada pelo código Verilog, é nessa rotina que devem ser atribuido os handles, e que deve ser chamado as funções acc_initalize e acc_close.

### 5.2.2 sad_entradas

Esta rotina é chamada para gerar os vetores de entrada randômicos para a SAD. Para fazer isso são criadas duas strings, uma para cada entrada da SAD.

```
char valor1[128];
char valor2[128];
```

Em cada uma das strings é colocado um valor randômico usando a função em C sprintf. São usados quatro rand(), pois cada um deles só consegue gerar até 32 bits, e cada entrada é formada por 128 bits.

```
sprintf (valor1, "%d%d%d%d",rand(),rand(),rand(),rand());
sprintf (valor2, "%d%d%d%d",rand(),rand(),rand(),rand());
```

Depois utiliza-se a rotina acc_set_value para colocar estes valores randômicos nas entradas da SAD. Importante observar para a escolha de "acc-DecStrVal"como o formato, já que valor1 e valor2 são strings com valores decimais.

### 5.2.3 sad

Esta rotina é chamada sempre que a SAD em verilog termina de fazer o cálculo. A função dela é obter os dois vetores que estão sendo fornecidos como entrada para a SAD em verilog usando a rotina acc_fetch_value, calcular o resultado da SAD para estes dois vetores usando o modelo, e comparar o resultado do modelo com o da SAD em verilog.

Como o modelo usado para este exemplo faz o cálculo usando "inteiros"e a rotina acc_fetch_value gera strings, é necessária fazer a conversão. Para fazer a conversão a string que carrega um número binário com 128 bits é quebrada em 16 partes(representando cada pixel) e convertida para um array de "inteiros"usando a função "converter".

### 5.2.4 converter

Esta função é utilizada para fazer a conversão de strings com números binários em inteiros. Um detalhe importante nesta conversão é a necessidade de substituir os valores da tabela ASCII(48, 49, 120, 122) pelos seus equivalentes 0, 1, x, z respectivamente.

## 5.3 Arquivo PLI.tab

O arquivo PLI.tab é utilizado pelo VCS para associar as system tasks/functions criadas pelo usuário dentro do código Verilog(neste exemplo $sad_monitor) com uma aplicação PLI.

Abaixo segue a sintaxe que deve ser usada neste arquivo:

$name PLI_specifications [access_capabilities]

Onde:

**$name** - Se refere ao nome da system task/function criada pelo usuário. Neste exemplo é $sad_monitor

**PLI_specifications** - Usado para passar uma ou mais especificações, como o nome da função em C a ser associada, tamanho do valor a ser retornado entre outros. Para passar o nome da função em C basta usar "call=nome_da_função"(sem aspas).

**access_capabilities** - Se refere a quais tipos de acesso a função em C pode ter em relação aos registers, nets... no código Verilog. Pode ser r, rw, wn. Neste exemplo pode ser omitido.

Para mais informações em relação ao arquivo PLI.tab basta ir na seção (C Language Inteface -> Using PLI -> PLI Table File) do manual do VCS.

## 5.4 Arquivo filelist

Neste arquivo vão listados todos os arquivos verilog e C que estão na pasta do projeto. O objetivo deste arquivo é só tirar a necessidade de fazer essa listagem na chamada do VCS.

Segue abaixo o conteúdo deste arquivo para este exemplo:

```
testbenchSAD.v
SAD.v
difAbs.v
SAD.c
```

# 6 VCS

## 6.1 Compilar

Tendo todos os arquivos na pasta do projeto, para fazer a compilação com o VCS basta rodar o seguinte comando:

vcs -debug_all -P pli.tab -f filelist

Onde:

**-debug_all** - Deve ser informado para o VCS usar full debug mode. Debug mode é necessário sempre que for usar PLI.

**-P** - Deve ser usado para informar o arquivo pli.tab.

**-f** - Deve ser usado para informar o arquivo filelist.

## 6.2 Simular

Para simular diretamente no terminal basta rodar a seguinte linha:

./simv

O resultado deve ser algo semelhante as linhas abaixo:

```
Chronologic VCS simulator copyright 1991-2012
Contains Synopsys proprietary information.
Compiler version G-2012.09; Runtime version G-2012.09;  Aug 23 14:49 2013
e1 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx,e2 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
************************************************
sad_verilog = 1353
sad_c = 1353
ok
e1 = 80feb4ae709c005252ff95adc9119d67,e2 = 0c6952b8e8da320ae8b5a1733388f851
************************************************
sad_verilog = 1754
sad_c = 1754
ok
e1 = 65997de86f6dc377147e1976bae9e129,e2 = 9404f09afa1c3bf5dc081b0f59d94ff2
```

Para simular usando a inteface gráfica basta adicionar -gui ao comando.

./simv -gui

Como um exemplo simples, as figuras 2,3,4,5,6 mostram os passos para visualizar o sinal "result"após a simulação.

# 7 Referências


- Manual VCS
- Tutorial de Verilog PLI - http://www.asic-world.com/verilog/pli.html

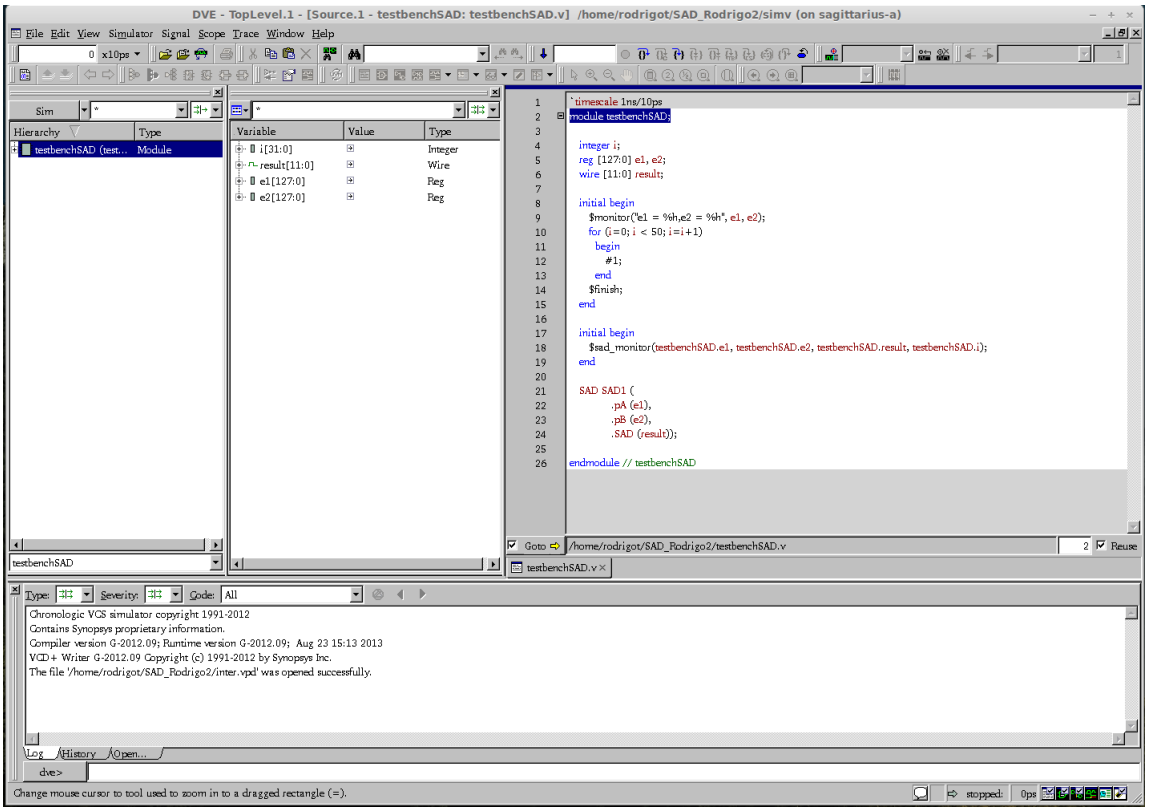


Figura 2: Primeiro passo

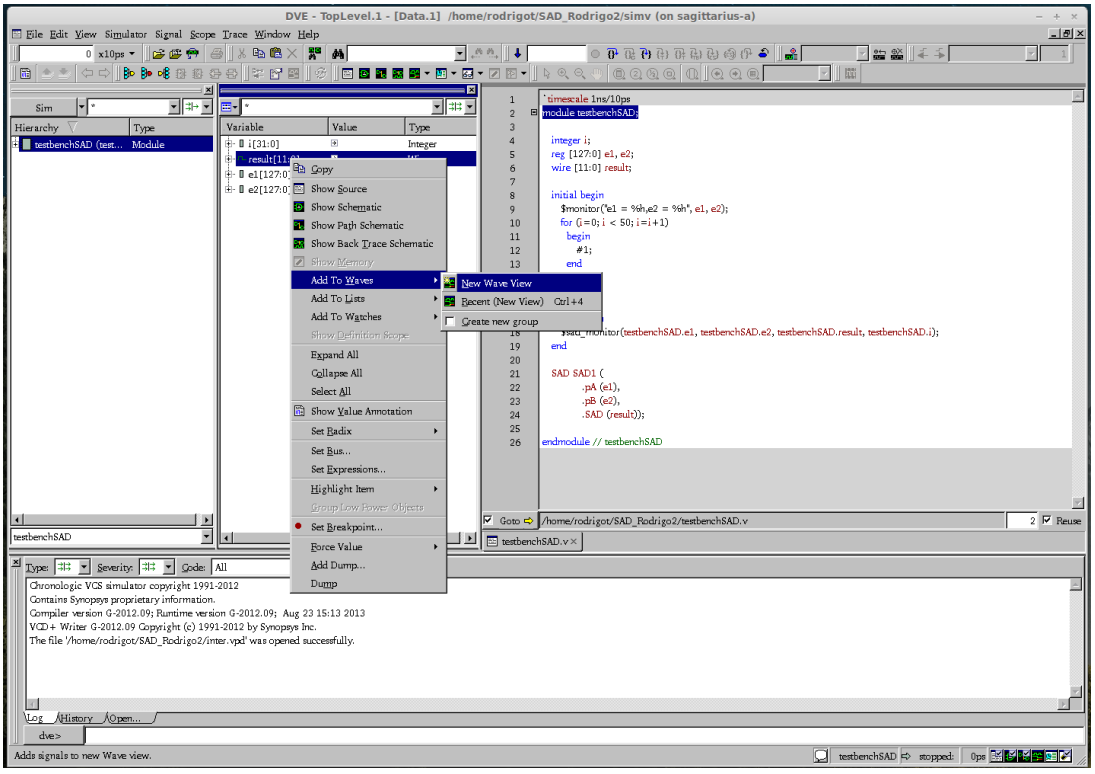


Figura 3: Segundo passo

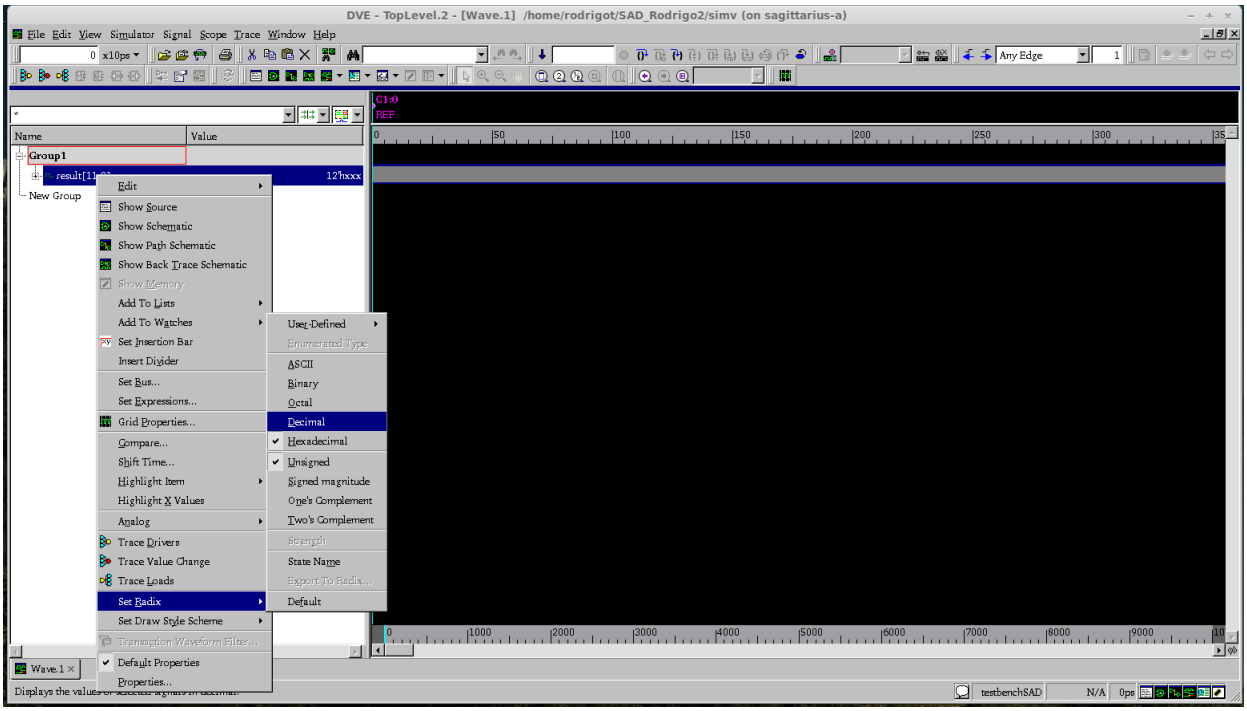


Figura 4: Terceiro passo

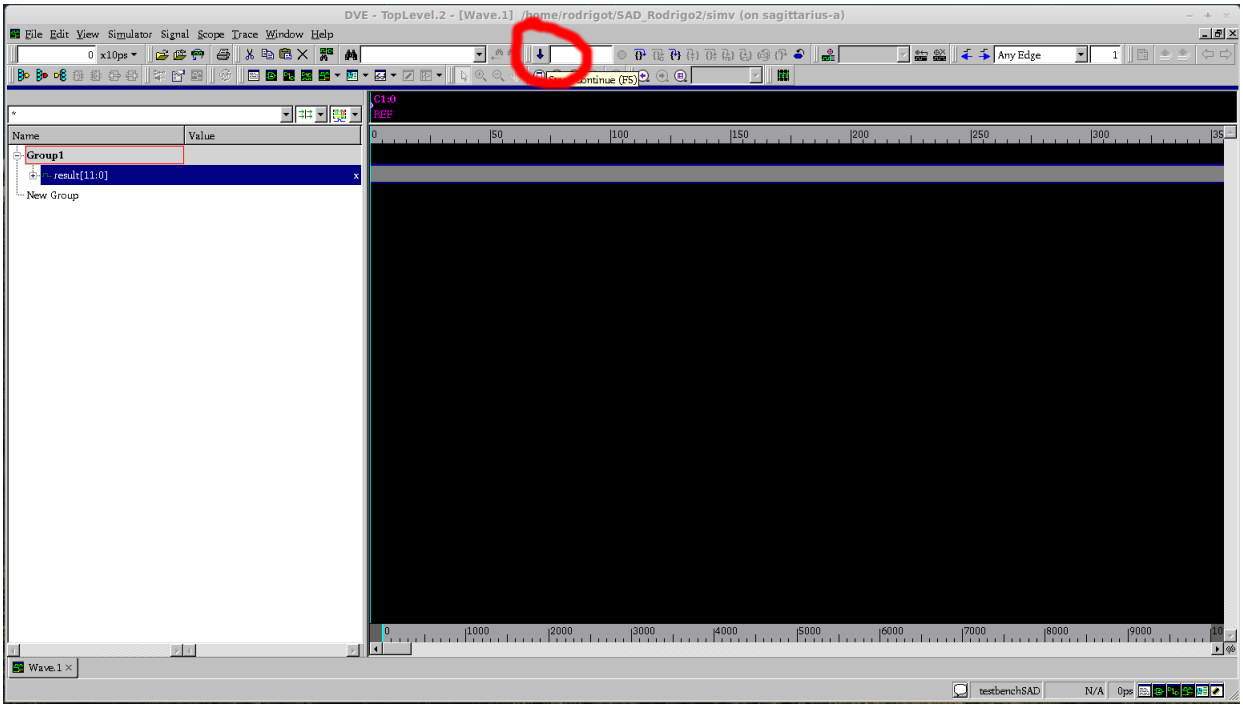


Figura 5: Quarto passo

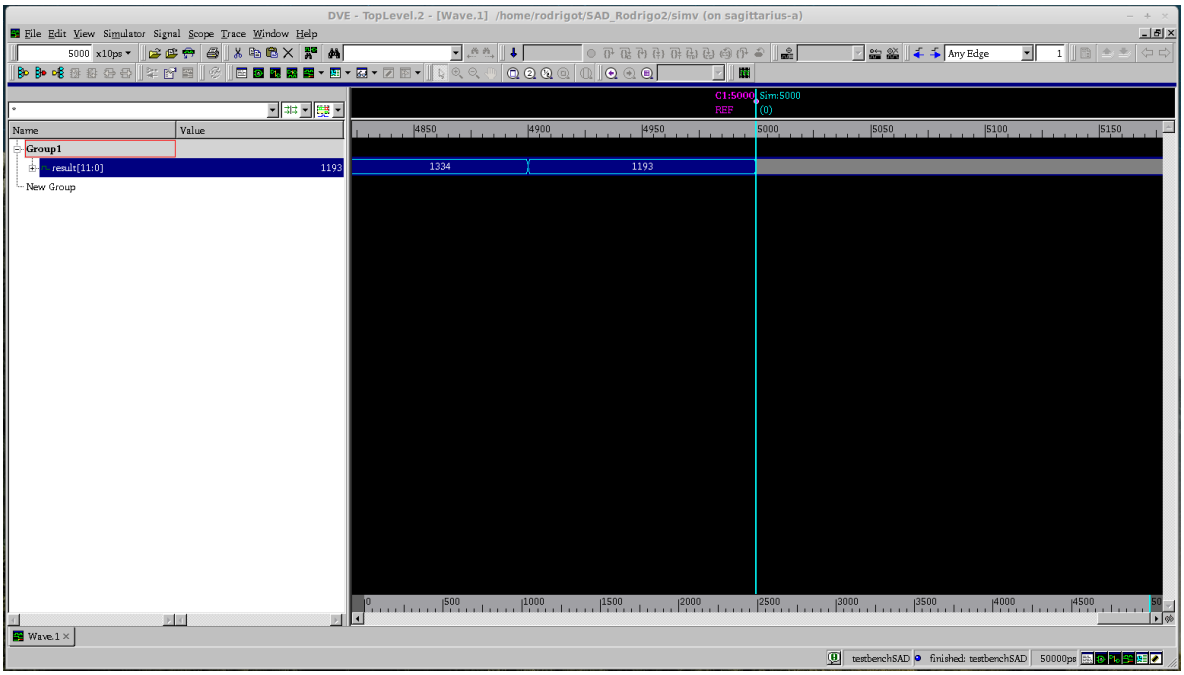


Figura 6: Quinto passo