实验六：波形生成与 gtkwave 的使用

1. 实验目的

1.1. 学习掌握基于波形的 RTL 代码调试手段，能够看懂读懂波形的实际含义；

1.2. 理解 VCD 波形的基本语法，能够仿真 Chisel 代码并生成 VCD 波形；

1.3. 掌握 gtkwave 的基本使用方法。

2. 实验内容

2.1. 学习有关波形调试的相关基础概念；

2.2. 安装并掌握 gtkwave 的基础使用方式；

2.3. 实现一个简单的三角波发生器，并能够使用 gtkwave 查看对应的三角波。

3. 实验相关 Chisel 语法介绍

本实验不涉及新介绍的 Chisel 语法。

4. 实验步骤

4.1. 波形调试基础

波形调试简介

在硬件设计和验证中，波形文件是重要的调试工具。它们记录了仿真过程中信号的变化，帮助工程师定位和解决问题。根据之前的几个实验我们可以知道在数字电路设计中，由于时序逻辑的存在，各种信号都是随时间随时钟信号在不断发生变化的，而这些变化往往很难通过简单的代码检查或者静态分析来捕捉。因此，波形文件的使用显得尤为重要。波形文件通过图形化的方式展示了数字电路中各个信号随时间的变化情况。工程师可以通过这些波形图准确地观察到信号在特定时刻的状态，从而找到设计中的潜在问题。例如，在同步电路设计中，某一信号在一个时钟周期内的变化可能会导致后续信号的错误，这种问题很难单纯通过代码来发现，但通过波形文件，可以直观地看到信号的传递过程和变化情况，从而能够很直观地反映电路的行为。

图 1 是一个波形图，可以看到一个波形图往往包括了横向的时间轴和纵向的信号轴。时间轴表示了信号随时间的变化，而信号轴则列出了所有被监控的信号。通过图中波形的高低变化，我们可以清晰地看到每个信号在不同时间点的状态。

在图 1 中，时间轴通常从左到右延展，标记了具体的时间刻度。这些时间刻度可以显示为绝对时间（如纳秒、微秒）或相对时间（如时钟周期）。信号轴则垂直排列，展示了每个信号在不同时间点的电平状态。数字信号通常用高低电平表示，高电平（通常为 1）和低电平（通常为 0）也可以分别用不同的颜色或线条样式来区分。

通过这种波形图，我们可以注意下面这两点：

（1）信号的上升沿和下降沿：这些是信号从低电平转变为高电平或从高电平转变为低电平的时刻。观察这些边沿可以帮助确认信号的变化是否符合预期。

（2）时钟信号：时钟信号通常以固定的周期变化，是驱动时序逻辑电路的核心。在波形图中，时钟信号的周期性变化可以用来对比其他信号的同步性。

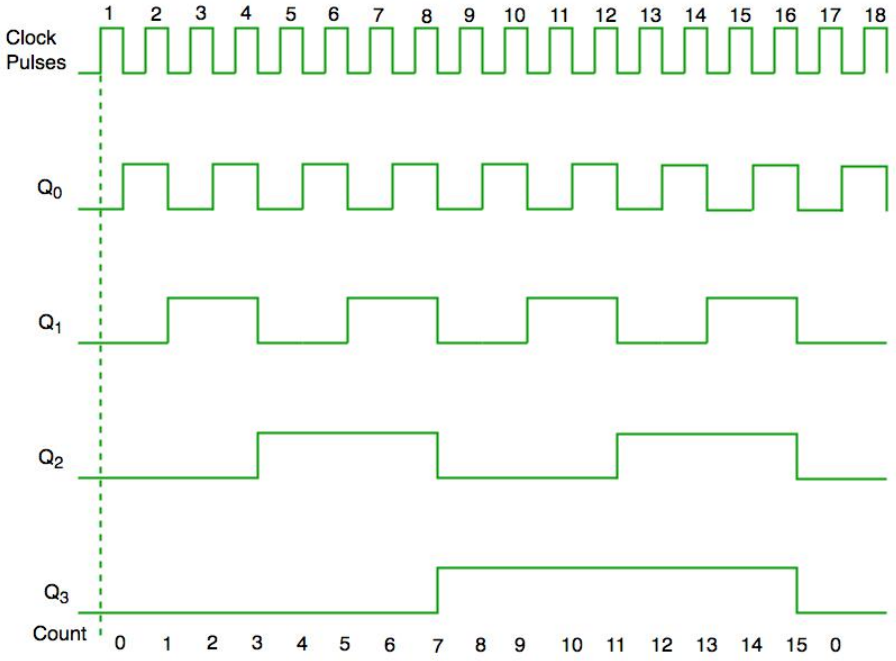


图 1 示例波形图

VCD 波形介绍

上一小节我们了解到波形图能够反映数字电路的行为，我们在仿真我们的设计的时候，就可以将仿真过程中生成的信号变化保存为波形文件，以方便后续的分析和调试，例如在编写 Chisel 的测试（chiseltest 测试）的时候就能够生成仿真过程中的波形文件。在众多波形文件格式中，VCD（Value Change Dump）文件是一种常见且广泛使用的格式。VCD 文件是一种文本文件格式，用于记录数字信号的变化情况。它由 IEEE 1364 标准定义，能够高效地存储仿真过程中信号的状态变化。VCD 文件的结构主要包括头部信息、信号声明和信号变化三部分。

1. 头部信息

头部信息包含了仿真开始的时间、时间单位、日期等基本信息。例如：

```
$date
    Date text. For example: November 11, 2009.
$end
$version
    VCD generator tool version info text.
$end
$comment
    Any comment text.
$end
$timescale 1ps $end
```

2. 信号声明

信号声明部分定义了仿真中涉及的信号和它们的标识符。每个信号都有一个唯一的标识符，通常是一个短字符串。这些标识符用于在后续的信号变化部分引用。例如：

```
$scope module top $end
$var wire 1 ! clock $end
$var wire 1 " reset $end
$var wire 8 # data $end
$upscope $end
$enddefinitions $end
```

在上述示例中，!、"和#是信号的唯一标识符，它们分别代表 clock、reset 和 data 信号。

3. 信号变化

信号变化部分记录了每个信号在仿真过程中不同时间点的状态变化。时间点用#符号表示，后续的行则描述了在该时间点发生变化的信号及其新值。例如：

```
#0
b00000000 #
1!
0"
#10
b00000001 #
0!
```

```
1"
```

在上面的例子中，#0 表示仿真开始时刻，b00000000 #表示 data 信号的初始值为 00000000，1!表示 clock 信号为高电平，0"表示 reset 信号为低电平。#10 表示仿真到 10 纳秒时刻，b00000001 #表示 data 信号变为 00000001，0!表示 clock 信号变为低电平，1"表示 reset 信号变为高电平。

由于 VCD 是在 Verilog 发布的时候就一同发布了，因此 VCD 一直以来都是作为一个通用且广泛的波形文件格式，VCD 文件是纯文本格式，易于阅读和解析，能够跨平台使用，大多数 EDA（电子设计自动化）工具都支持生成和解析 VCD 文件。

接下来我们可以试着生成一个波形文件，我们以实验 4 中的 CounterWithFlag 这个设计作为例子来生成 VCD 波形，首先切换到工程根目录，执行下面命令即可生成 CounterWithFlag 的 VCD 波形：

```
mill MyChiselProject.test.testOnly exp6.TestGenerateVCD
```

```
lc3@lc3-virtual-machine:~/chisel_exp$ mill MyChiselProject.test.testOnly exp6.TestGenerateVCD
[76/83] MyChiselProject.test.compile
[info] compiling 1 Scala source to /home/lc3/chisel_exp/out/MyChiselProject/test/compile.dest/classes ...
[info] done compiling
[83/83] MyChiselProject.test.testOnly
TestGenerateVCD:
maxCount is 30
Initial flag: 0
Counter at step 0: flagOut = 0
Counter at step 1: flagOut = 0
Counter at step 2: flagOut = 0
Counter at step 3: flagOut = 0
Counter at step 4: flagOut = 0
Counter at step 5: flagOut = 0
Counter at step 6: flagOut = 0
Counter at step 7: flagOut = 0
Counter at step 8: flagOut = 0
Counter at step 9: flagOut = 0
Counter at step 10: flagOut = 1
Counter at step 11: flagOut = 1
Counter at step 12: flagOut = 1
Counter at step 13: flagOut = 1
Counter at step 14: flagOut = 1
Counter at step 15: flagOut = 1
Counter at step 16: flagOut = 1
Counter at step 17: flagOut = 1
Counter at step 18: flagOut = 1
Counter at step 19: flagOut = 1
Counter at step 20: flagOut = 0
Counter at step 21: flagOut = 0
Counter at step 22: flagOut = 0
Counter at step 23: flagOut = 0
Counter at step 24: flagOut = 0
Counter at step 25: flagOut = 0
Counter at step 26: flagOut = 0
Counter at step 27: flagOut = 0
Counter at step 28: flagOut = 0
Counter at step 29: flagOut = 0
Counter after maxCount steps: 0
- TestGenerateVCD should generate VCD file
```

生成波形文件存放在 test_run_dir/TestGenerateVCD_should_generate_VCD_file 目录下，波形文件名为 CounterWithFlag.vcd，部分内容如下图所示，其格式和我们上面介绍的基本一致。接下来我们需要使用特定的波形软件来打开这个 VCD 波形文件就能打开对应的图形化波形，用于可视化的调试与查看，我们将在下一小节介绍这一点。

test_run_dir > TestGenerateVCD_should_generate_VCD_file > CounterWithFlag.vcd

```
$date
2024-08-02T14:59+0000
$end
$version
0.2
$end
$comment

$end
$timescale 1ns  $end
$scope module CounterWithFlag $end
 $var wire 1 ! io_flagOut $end
 $var wire 1 " flag $end
 $var wire 1 # clock $end
 $var wire 5 $ counter $end
 $var wire 1 % reset $end
$upscope $end
$enddefinitions $end
$dumpvars
0!
0"
0#
b00000 $
0%
$end
#0
1%
#1
1#
#6
0#
0%
#11
1#
b00001 $
#16
0#
#21
1#
b00010 $
#26
0#
#31
1#
b00011 $
#36
0#
#41
1#
```

4.2. gtkwave 介绍

在 3.1 中我们在最后生成了一个 VCD 波形，但是还没有将其可视化地打开并查看，在这一小

节中，我们将介绍如何使用 gtkwave 工具来查看和分析生成的 VCD 波形文件。

gtkwave 是一款免费的开源波形查看工具，广泛用于查看和分析由仿真工具生成的 VCD 文件。它支持多种波形文件格式，并提供了强大的信号浏览和分析功能，使得工程师能够直观地查看数字电路的行为。GTKWave 可以在多种操作系统上运行，包括 Windows、Linux 和 macOS。如果你使用的是 Linux（Debian/Ubuntu），那么安装 GTKWave 的只需要执行下面的命令即可进行安装，其他类型的 Linux 也是使用相对应的包管理器即可安装：

```
sudo apt install gtkwave
```

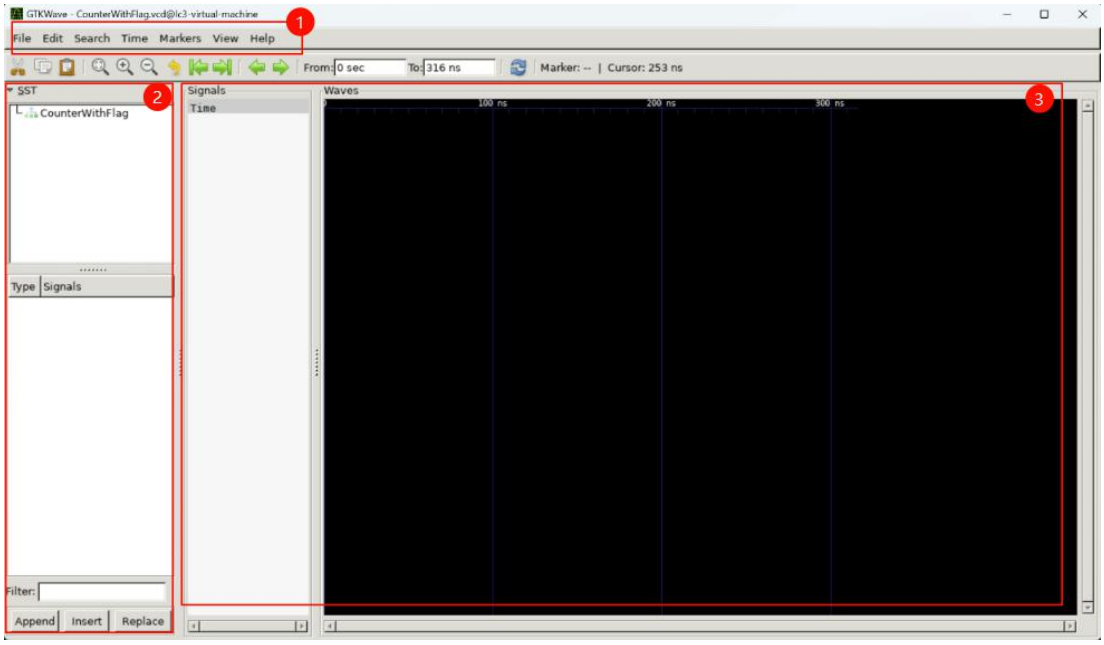
注意：如果你使用的是实验提供的虚拟机镜像，那么不需要另外安装，虚拟机镜像已经安装。

在安装完 gtkwave 后，在命令行中输入 `gtkwave --help` ，如果输出了下面的信息，则说明安装成功，后续可以正常使用 gtkwave。

```
lc3@lc3-virtual-machine:~$ gtkwave --help
Usage: gtkwave [OPTION]... [DUMPFILE] [SAVEFILE] [RCFILE]

  -n, --nocli=DIRPATH         use file requester for dumpfile name
  -f, --dump=FILE             specify dumpfile name
  -F, --fastload              generate/use VCD recoder fastload files
  -o, --optimize              optimize VCD to FST
  -a, --save=FILE             specify savefile name
  -A, --autosavename          assume savefile is suffix modified dumpfile name
  -r, --rcfile=FILE           specify override .rcfile name
  -d, --defaultskip           if missing .rcfile, do not use useful defaults
  -D, --dualid=WHICH          specify multisession identifier
  -l, --logfile=FILE          specify simulation logfile name for time values
  -s, --start=TIME            specify start time for LXT2/VZT block skip
  -e, --end=TIME              specify end time for LXT2/VZT block skip
  -t, --stems=FILE            specify stems file for source code annotation
  -c, --cpu=NUMCPUS           specify number of CPUs for parallelizable ops
  -N, --nowm                  disable window manager for most windows
  -M, --nomenus               do not render menubar (for making applets)
  -S, --script=FILE           specify Tcl command script file for execution
  -T, --tcl_init=FILE         specify Tcl command script file to be loaded on startup
  -W, --wish                  enable Tcl command line on stdio
  -R, --repscript=FILE        specify timer-driven Tcl command script file
  -P, --repperiod=VALUE       specify repscript period in msec (default: 500)
  -X, --xid=XID               specify XID of window for GtkPlug to connect to
  -1, --rpcid=RPCID           specify RPCID of GConf session
  -2, --chdir=DIR             specify new current working directory
  -3, --restore               restore previous session
  -4, --rcvar                 specify single rc variable values individually
  -5, --sstexclude            specify sst exclusion filter filename
  -I, --interactive           interactive VCD mode (filename is shared mem ID)
  -C, --comphier              use compressed hierarchy names (slower)
  -g, --giga                  use gigabyte mempacking when recoding (slower)
  -L, --legacy                use legacy VCD mode rather than the VCD recoder
  -v, --vcd                   use stdin as a VCD dumpfile
  -O, --output=FILE           specify filename for stdout/stderr redirect
  -z, --slider-zoom           enable horizontal slider stretch zoom
  -V, --version               display version banner then exit
  -h, --help                  display this help then exit
  -x, --exit                  exit after loading trace (for loader benchmarks)

VCD files and save files may be compressed with zip or gzip.
GHW files may be compressed with gzip or bzip2.
Other formats must remain uncompressed due to their non-linear access.
Note that DUMPFILE is optional if the --dump or --nocli options are specified.
SAVEFILE and RCFILE are always optional.

Report bugs to <bybell@rocketmail.com>.
```

我们切换到实验工程根目录的 test_run_dir/TestGenerateVCD_should_generate_VCD_file 文件夹，输入下面的命令即可使用 gtkwave 打开波形这个 VCD 波形文件：

```
gtkwave CounterWithFlag.vcd
```



打开 gtkwave 后，可以得到上面的这个图，gtkwave 的界面可以大致分为三个部：（1）菜单栏

菜单栏位于 GTKWave 窗口的顶部，包含了各种操作和设置选项。主要菜单包括：

File：用于打开、保存和关闭波形文件。

Edit：提供剪切、复制、粘贴等编辑功能，以及添加和删除标记。

Search：提供信号搜索和过滤功能。

Time：用于时间相关的操作，如设置时间单位、跳转到指定时间点等。

Markers：管理波形中的标记和光标。

View：用于调整界面视图，如放大、缩小、自动缩放等。

Help：提供帮助文档和关于 GTKWave 的信息。

（2）SST 信号（Signal Selector Tree）

SST 信号窗口位于 GTKWave 界面的左侧，是用于浏览和选择信号的区域。它按照模块层次结构展示了波形文件中的所有信号。

（3）波形展示界面

波形展示界面位于 GTKWave 的中间和右侧区域，是用于显示和分析信号波形的主要部分。主要功能包括：

Waveform Display：显示选中信号的波形图，用户可以通过缩放和拖动来调整视图范围。

Time Scale：显示当前波形图的时间刻度，用户可以通过点击和拖动时间刻度来移动视图。

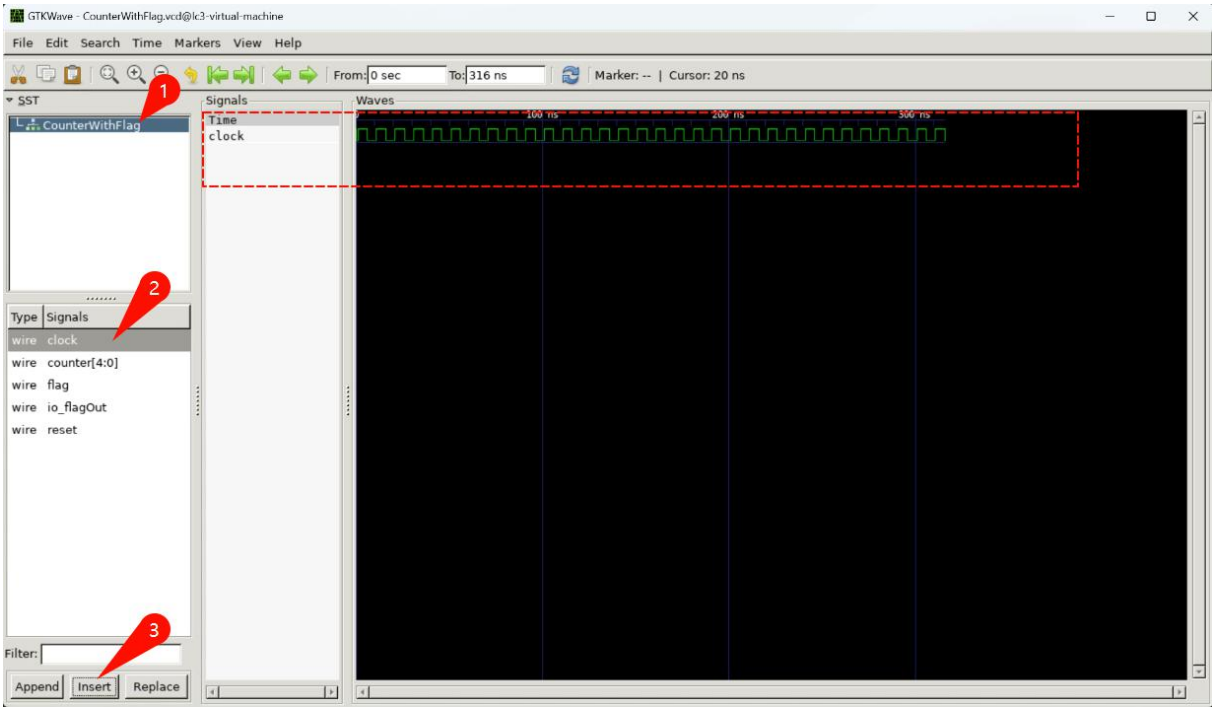
Cursors and Markers：允许用户添加光标和标记，以精确测量时间和信号值。可以通过

Edit 菜单添加光标。

gtkwave 基础教程

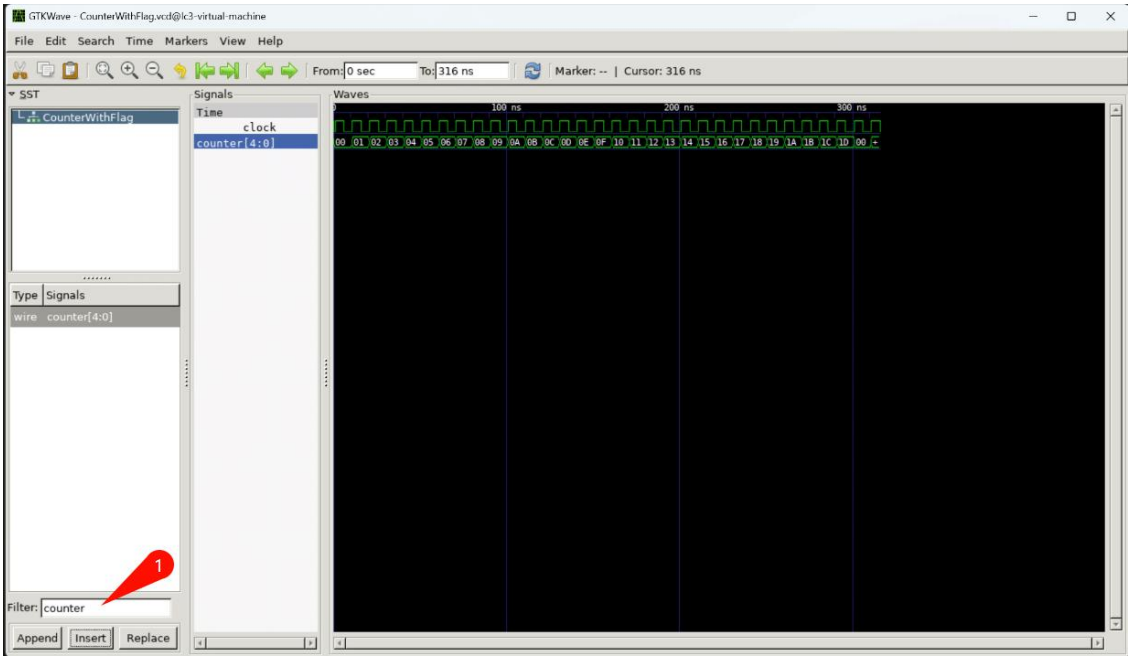
1）添加显示信号

首先需要在 SST 中选中模块，然后在 2 中选择要 insetr 的信号，点击 3 处的 insert 即可插入信号，并在右侧看到已经插入的波形。



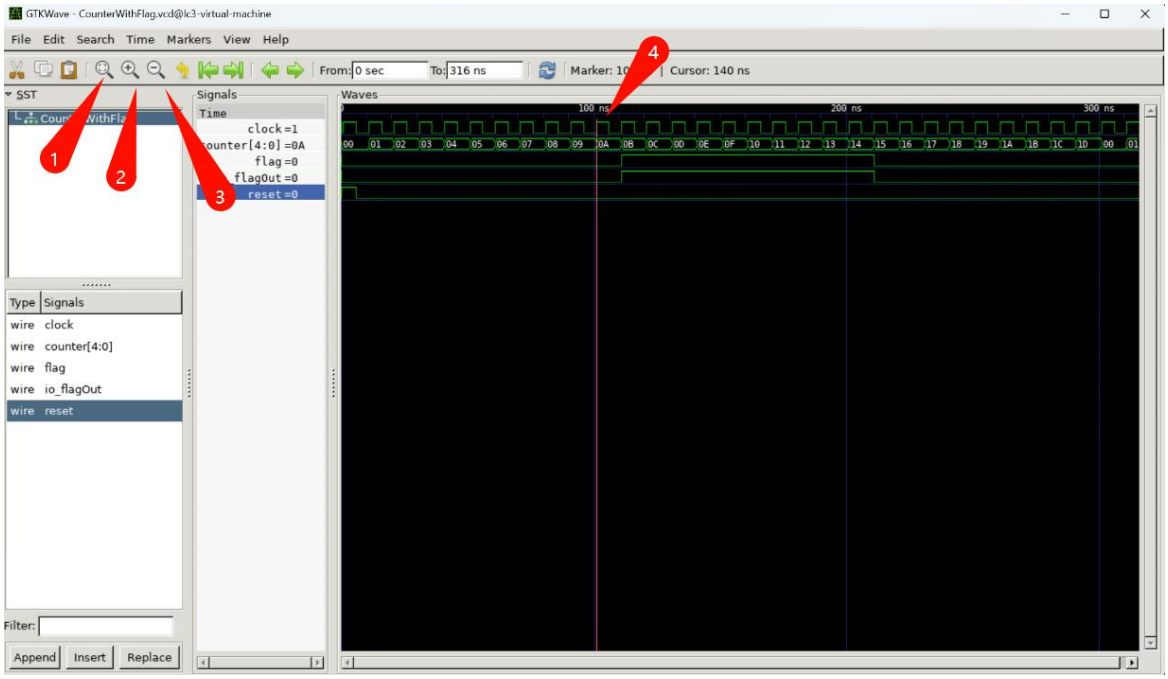
2）信号过滤

有时候我们的设计里面信号很多，这时候就需要用到信号过滤功能来搜索我们感兴趣的信号进行查看，例如下面我们在 1 处搜索了 counter 信号。



3）信号缩放

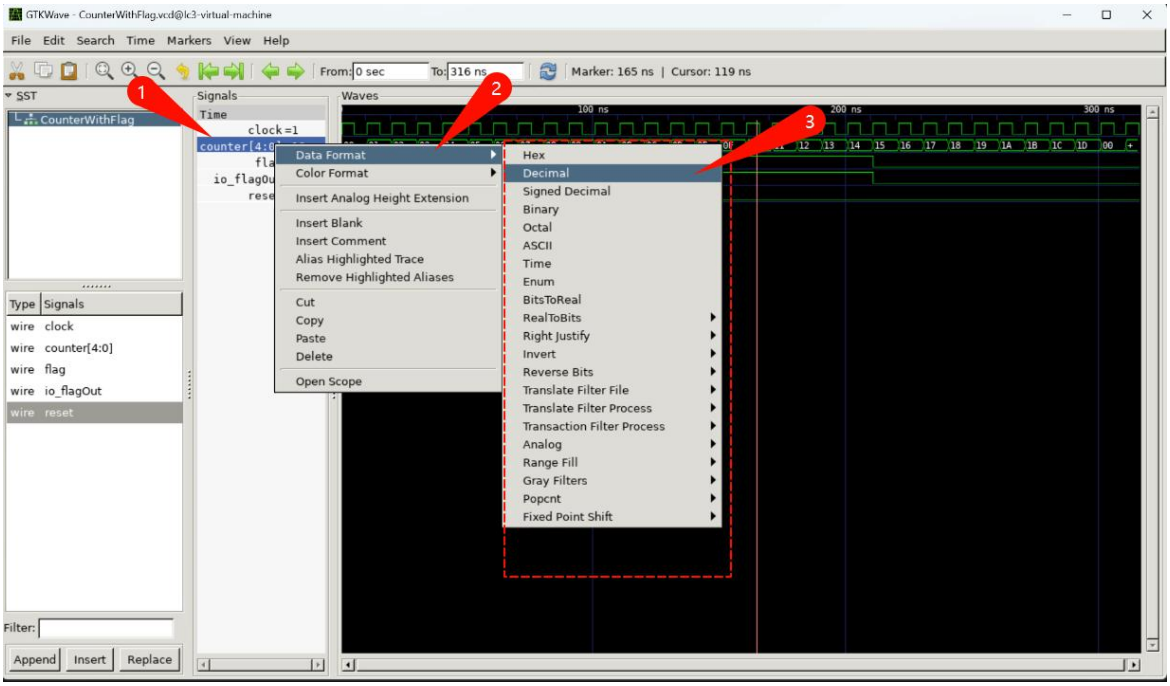
信号在波形图中的显示有时候比较拥挤或者比例不太适合查看，这时候需要用到信号缩放的功能，1 表示的是 Zoom Fit 自动调整缩放比例以适应全部波形。2 表示的是 Zoom In 放大波形视图，使得信号显示得更详细，或者使用快捷键（通常是 Ctrl + + ）。3 表示的是 Zoom Out 缩小波形视图，以查看更长时间范围内的信号变化，或者使用快捷键（通常是 Ctrl + - ）。我们可以在 4 中将光标移动到感兴趣的信号位置，然后使用放大或缩小功能调整波形视图，以精确查看该位置的信号变化。



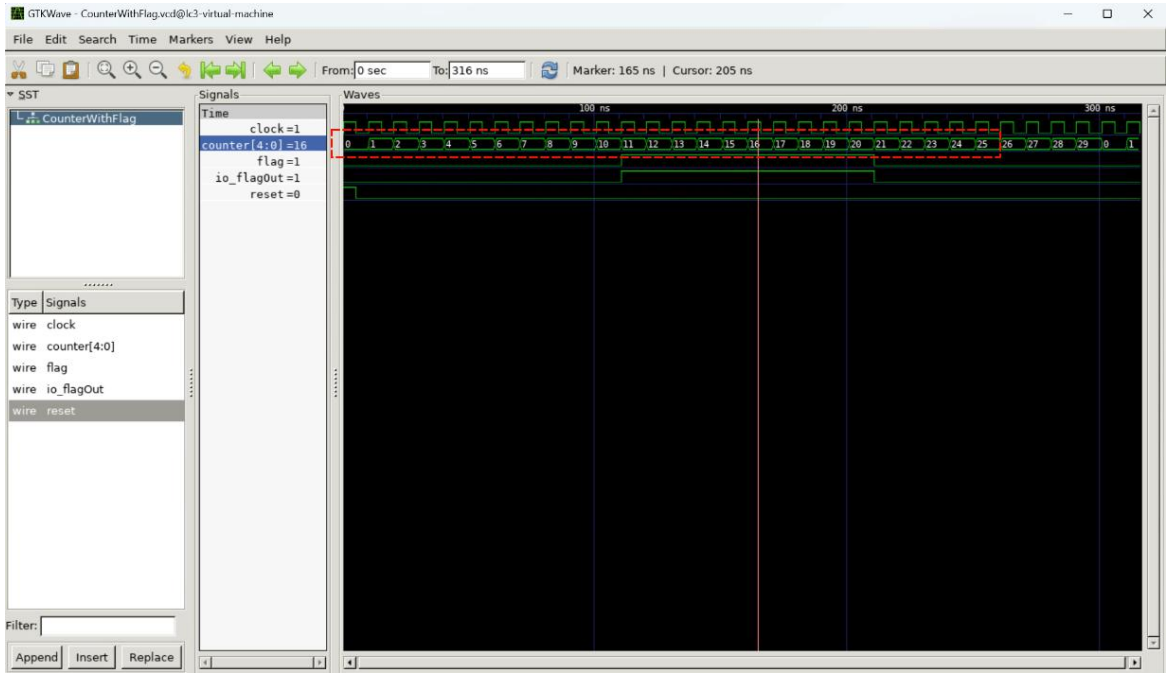
4）进制转化

在 gtkwave 的所显示的信号中，我们可以看到数值都是默认以 16 进制显示的，但是我们有时

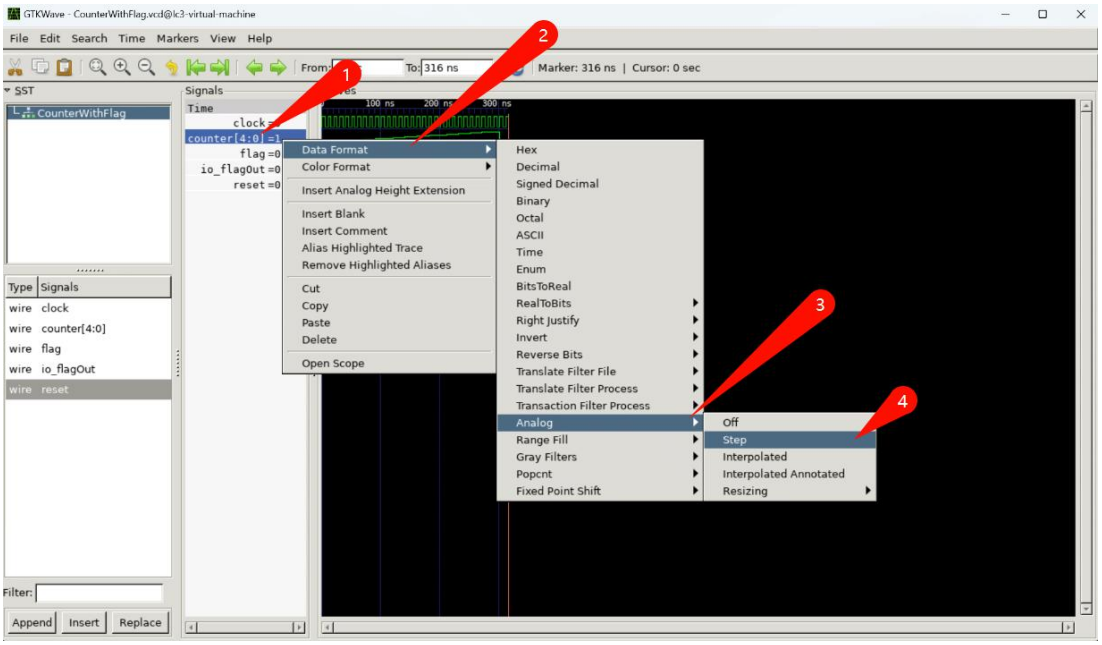
候希望以其他的各种进制来显示信号的数值，例如二进制、十进制等。我们可以在 1 中选中要操作的信号，右键后可以打开一个菜单，选择 2 中的 Data format，然后得到二级子菜单，其中列举了许多支持的格式，我们选择 3 中的 Decimal 也就是十进制来显示 counter 的值。

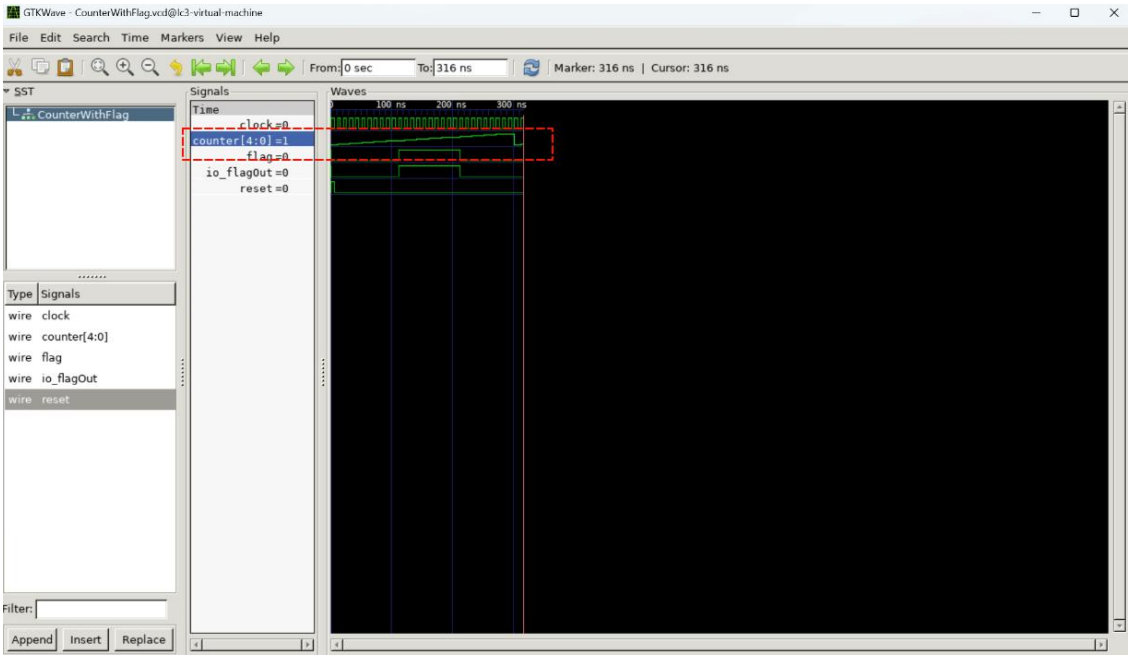


设置完 Decimal 格式后，可以看到数值都发生了变化。

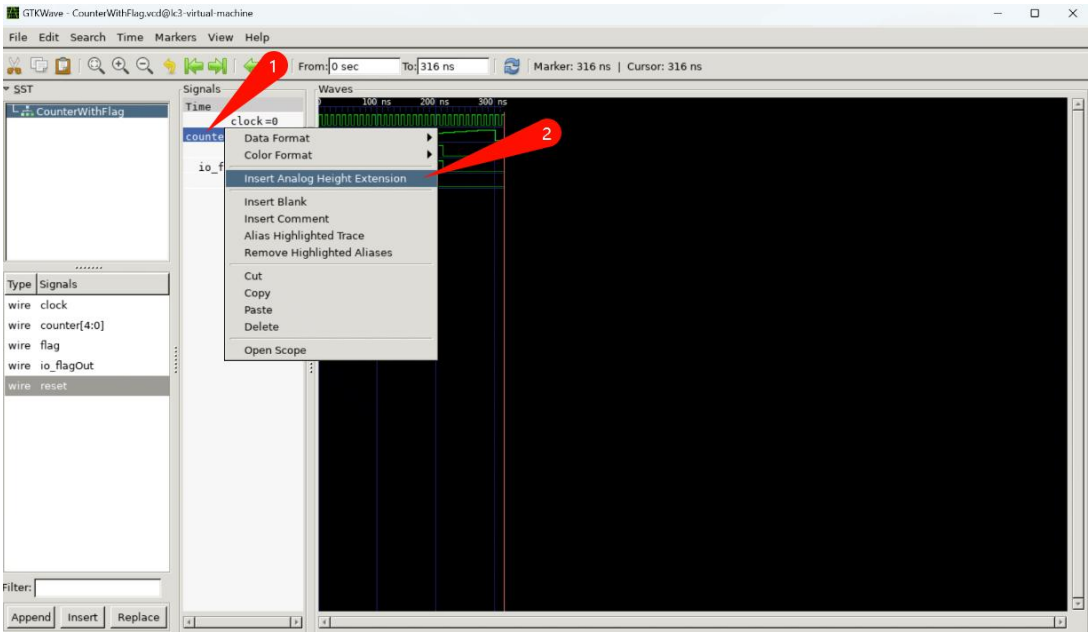


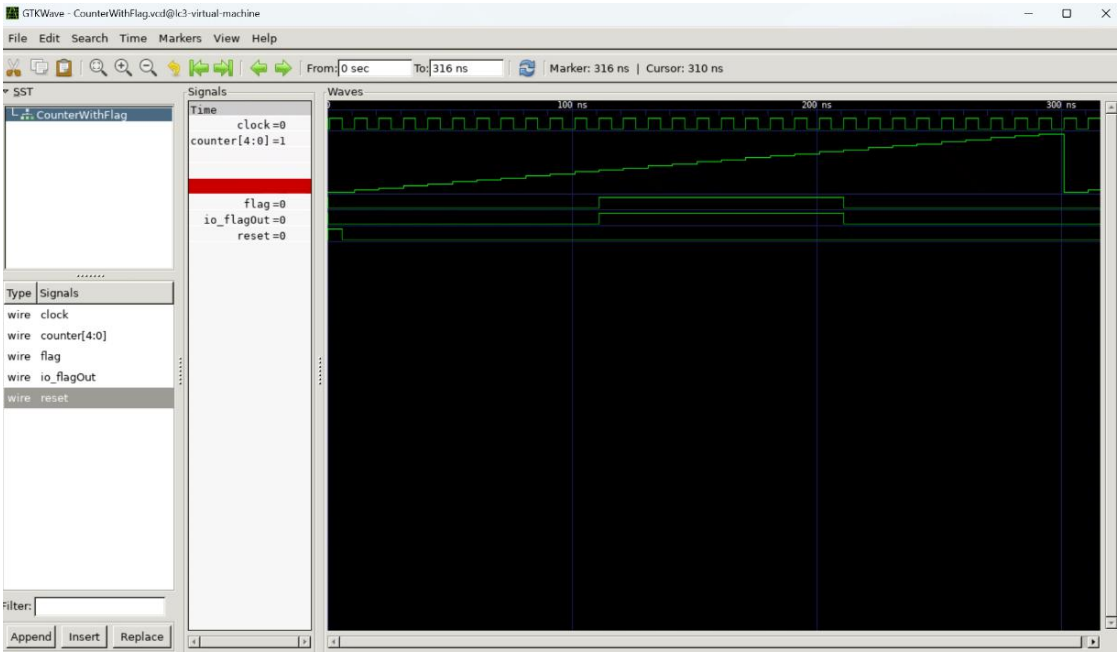
除了更改进制外，我们还可以更改显示的形式，例如以 Analog 的方式来显示，Analog 显示将信号以模拟波形的形式展示，Analog 显示将信号以连续的曲线形式展示，避免了数字波形中常见的阶段数值变化，使得波形看起来更平滑和自然。按照下面的 1~4 步骤来操作，即可将 counter 信号的值变为 Analog 显示。





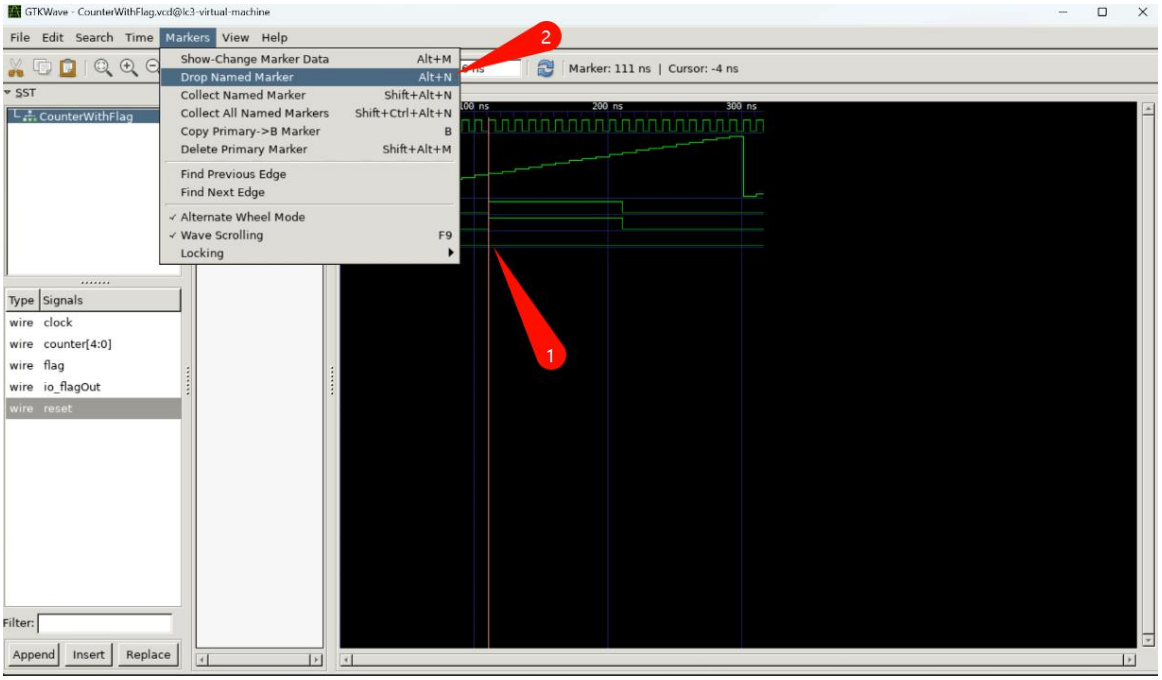
可以看到 counter 的值在 Analog 模式下类似一个锯齿的样式，也就是一个锯齿波的一部分，同时我们也发现这个锯齿的纵向宽度比较窄，我们可以右键信号选择 Insert Analog Height Extension 来将这个宽度扩展提供更好的可读性，这个选项也可以右键执行多次。

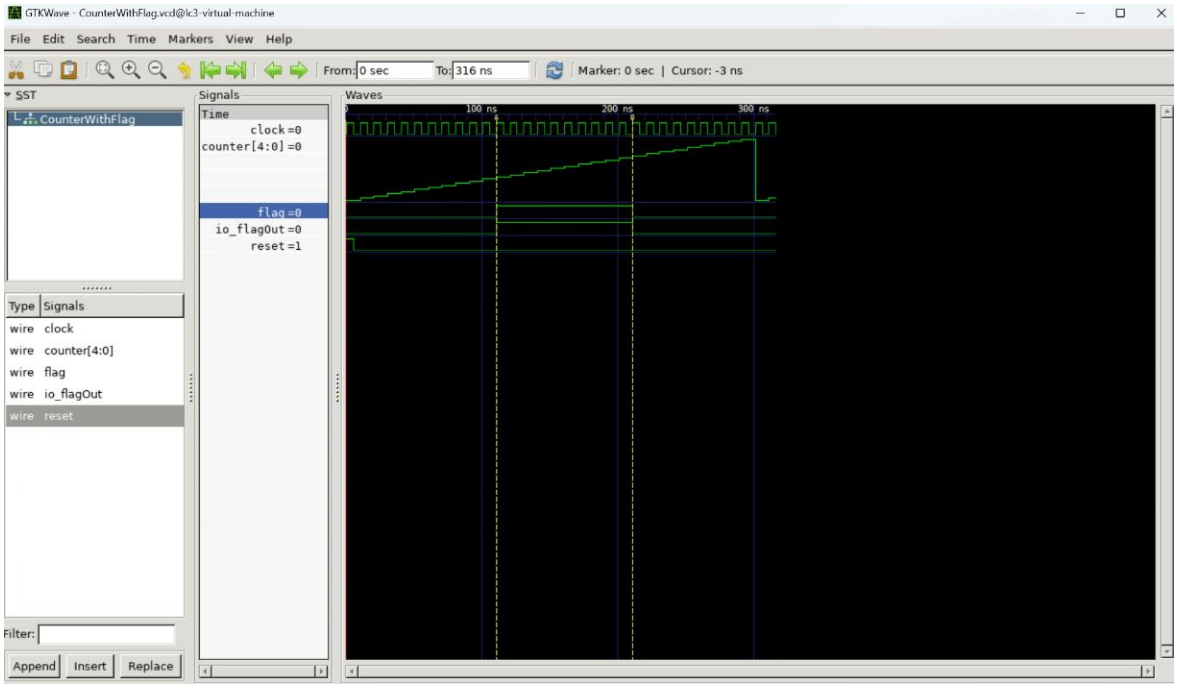




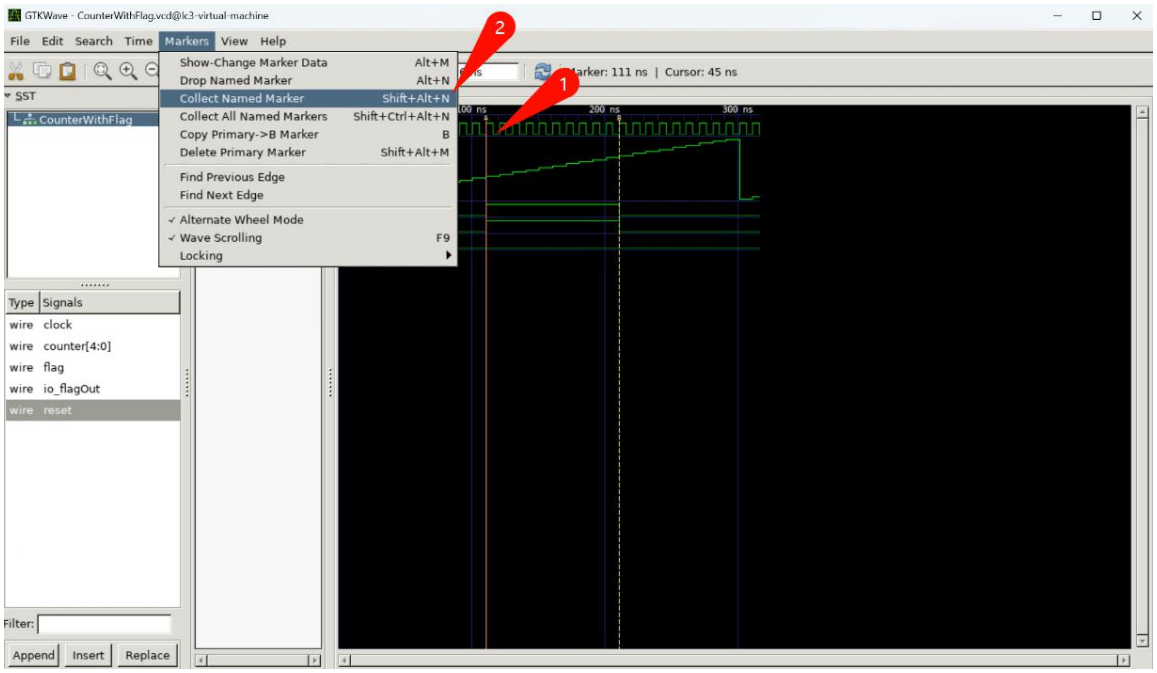
5) Mark 点添加

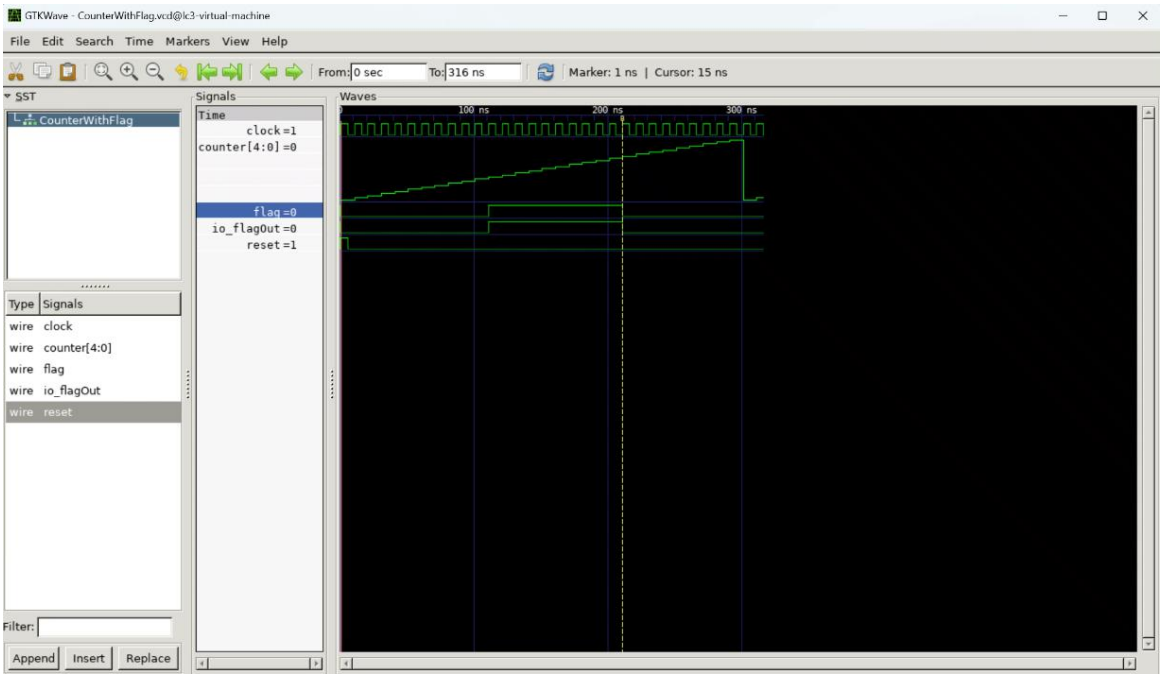
在查看波形的时候我们可以添加 Mark 点来标记特定的时间点或事件，Mark 点可以帮助用户快速定位到感兴趣的波形区域。在下图中，我们首先先在波形视图选中感兴趣的位置，然后到菜单栏的 Markers 选项下选择 Drop Named Marker 即可添加一个 Mark（或者使用快捷键 Alt+N），Mark 点也可以添加多个。





如果想要删除掉对应的 Mark 点，那么可以在波形视图选中 Mark 点然后到菜单栏中的 Marker 栏下选择 Collect Named Marker（或者用快捷建 Shift+Alt+N）。

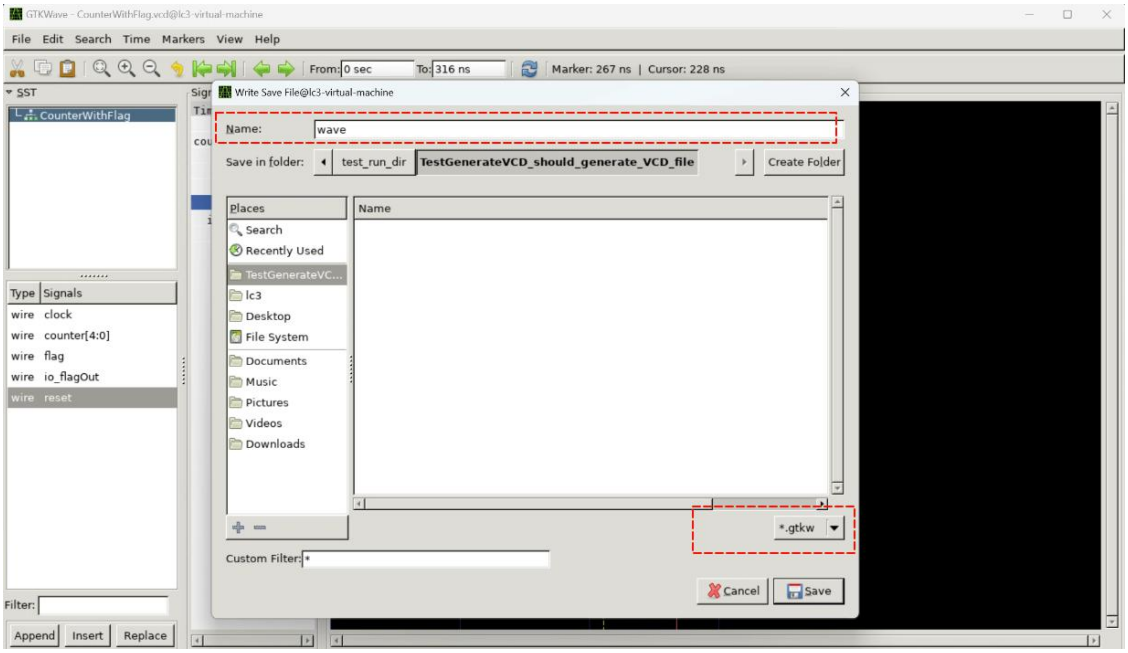




6）波形保存

在打开波形文件后，我们添加了许多我们想要查看的信号，但是在下次用同样的命令打开这个波形的时候我们还需要手动再次添加这些信号，这样很不方便，因此 gtkwave 允许我们将当前的信号选择和视图设置保存为一个文件，以便下次快速恢复。

在菜单栏中选择 File -> Write Save File，然后选择一个位置和文件名来保存当前的波形设置，同时需要输入保存的文件名，文件的后缀是.gtkw，下图中我们将这个波形视图保存在了和 VCD 波形相同目录下的 wave.gtkw 文件中。



当我们下次打开这个 VCD 波形的时候如果想要恢复这个视图，就可以通过 File-> Read Save File 来选择上次保存的.gtkw 文件，来快速恢复视图。当然你也可以在命令行打开波形的时候使用-a 选项来指定要打开的.gtkw 文件，例如：

```
gtkwave CounterWithFlag.vcd -a wave.gtkw
```

总结：

到目前为止我们已经能够使用 gtkwave 打开并查看 VCD 波形，同时也学习了一些 gtkwave 的相关使用技巧，例如进制转化、Mark 点添加、波形保存等，并且在进制转化这里我们还介绍了如何使用 Analog 模式来打开波形，Analog 模式允许我们用更为连续的曲线来查看波形的变化，我们在后续的实验中会经常使用这个模式用于展示实验的效果。

4.3.三角波生成器

在经过前面几个实验的学习后，我们已经掌握了时序逻辑电路中的 Counter 的设计，Counter 是一个寄存器，满足特地条件的时候计数值会加一，Counter 可以用于设计各种复杂的时序电路，例如我们在实验四中通过观察 Counter 计数值画出来后可以看到 Counter 的值类似一个锯齿波，同样地，如果我们改变 Counter 的计数规则，还能实现各种其他的波形，例如我们可以实现一个三角波发生器，首先我们考虑下面的代码：

src/main/scala/exp6/TriangleWave.scala

```
1.  package exp6
2.
3.  import chisel3._
4.  import chisel3.util._
5.
```

```
6.  class TriangleWave extends Module {
7.    val io = IO(new Bundle {
8.      val waveOut = UInt(16.W)
9.    })
10.
11.   val counter   = RegInit(0.U(16.W))
12.   val direction = RegInit(true.B) // true for up, false for down
13.   val waveReg   = RegInit(0.U(16.W))
14.
15.   when(counter(7) ^ ~direction) {
16.     direction := ~direction
17.   }
18.   counter := counter + 1.U
19.
20.   when(direction) {
21.     waveReg := waveReg + 1.U
22.   }.otherwise {
23.     waveReg := waveReg - 1.U
24.   }
25.
26.   io.waveOut := waveReg
27. }
```

这个 Chisel 代码定义了一个简单的三角波发生器模块。让我们逐行分析这个代码，解释每个部分的功能。

其中有三个重要的寄存器：

1）counter：一个 16 位的无符号整数寄存器，用于计数。

2）direction：一个布尔寄存器，用于控制波形上升或下降。true 表示上升，false 表示下降。

3）waveReg：一个 16 位的无符号整数寄存器，用于存储当前的波形值。

而在下面这块代码中，实现了波形方向的切换逻辑。当 counter 的第 8 位（counter(7)）和 direction 的取反值（~direction）异或（^）结果为 1 时，切换 direction 的值（从上升变为下降，或从下降变为上升）。这意味着每 128 个计数周期，方向会改变一次，从而形成三角波。

```
1.    when(counter(7) ^ ~direction) {
2.      direction := ~direction
3.    }
```

下面这里则是将 waveReg 根据 direction 进行不同方向的计数，最后将信号通过 io.waveOut := waveReg 输出到端口。

```
1.    when(direction) {
2.      waveReg := waveReg + 1.U
3.    }.otherwise {
```

```
4.         waveReg := waveReg - 1.U
5.     }
```

接下来我们可以切换到实验工程根目录输入下面的命令用于生成 VCD 波形：

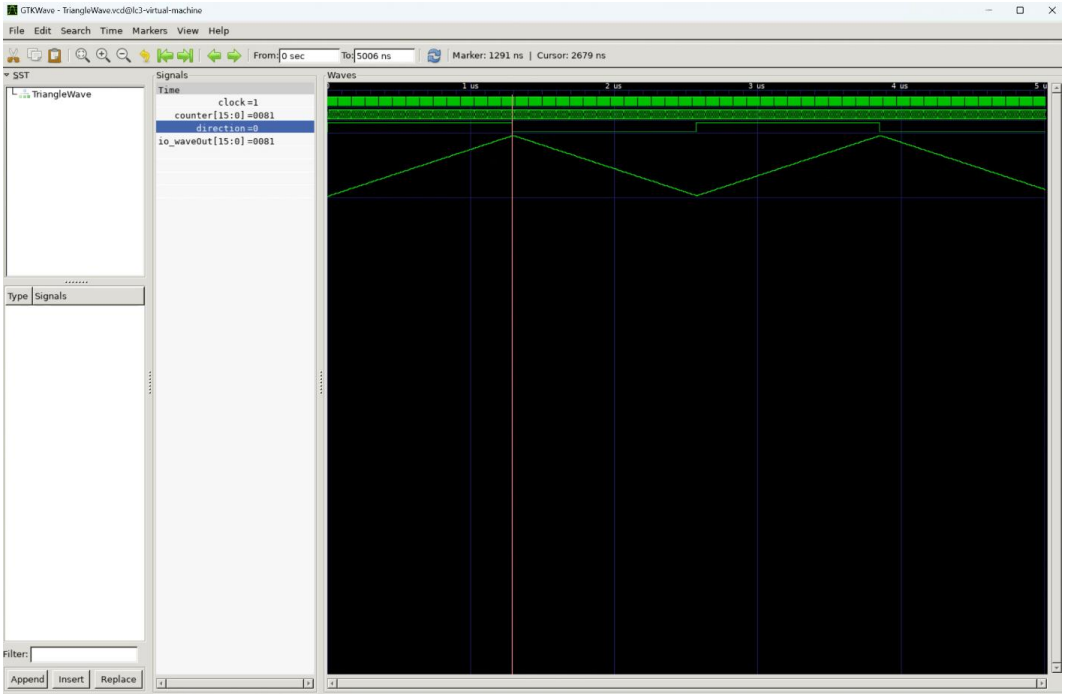
```
mill MyChiselProject.test.testOnly exp6.TestTriangleWave
```

```
lc3@lc3-virtual-machine:~/chisel_exp$ mill MyChiselProject.test.testOnly exp6.TestTriangleWave
[50/83] MyChiselProject.compile
[info] compiling 1 Scala source to /home/lc3/chisel_exp/out/MyChiselProject/compile.dest/classes ...
[info] done compiling
[76/83] MyChiselProject.test.compile
[info] compiling 1 Scala source to /home/lc3/chisel_exp/out/MyChiselProject/test/compile.dest/classes ...
[info] done compiling
[83/83] MyChiselProject.test.testOnly
TestTriangleWave:
- TestTriangleWave should generate VCD file
```

然后切换到 test_run_dir/TestTriangleWave_should_generate_VCD_file 目录下，波形文件名为 TriangleWave.vcd，使用 gtkwave 打开波形：

```
gtkwave TriangleWave.vcd
```

添加信号，并且对 waveOut 进行 Analog 模式显示后后我们可以看到下面的波形图：



此时 waveOut 显示的就是一个三角波的波形，并且我们可以看到在每次 direction 的值发生变化的时候 waveOut 的计数值就会发生发生变化，从最开始的 count up 到 count down，如此循环。

这里的计数峰值也可以观察到是 0x81，这个值主要受到 counter(7）这里的 7 的影响，因为 counter 是一个 16 位的计数器，每个时钟周期递增 1。counter(7)表示 counter 的第 8 位（从 0 开始计数），即 counter 的值在二进制表示下的第 8 位。当 counter(7)为 0 时，counter 的值范围是 0 到 127（0x00 到 0x7F）。当 counter(7)为 1 时，counter 的值范围是 128 到 255（0x80 到 0xFF）。由于 direction 在 counter(7)异或取反后的值为 1 时才会切换，这意味着每当 counter 的值从 127（0x7F）变为 128（0x80），或从 255（0xFF）变为 0 时，direction 都会切换一次。

具体来说：

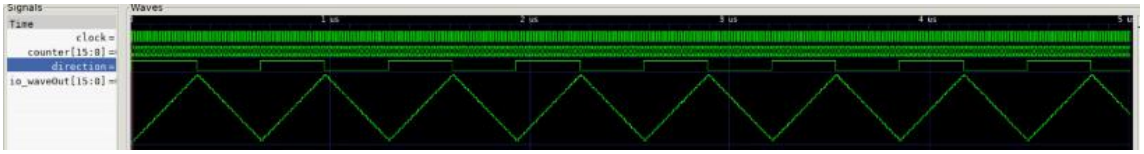
当 counter 从 0 增加到 127（0x7F）时，counter(7)为 0，~direction 为 1（假设最初 direction 为 true）。

当 counter 从 128 增加到 255（0xFF）时，counter(7)为 1，~direction 为 0（假设最初 direction 为 true）。

因此，direction 在 counter 达到 128 时切换，从而导致波形从上升变为下降，或从下降变为上升。这种切换导致 waveReg 的值在 0x00 到 0xFF 之间循环变化，而峰值为 128（0x80）。

通过这种设计，三角波的频率和幅度由计数器的位宽和方向控制寄存器决定。counter(7)的选择使得波形在 128 个计数周期后改变方向，从而形成一个对称的三角波波形。调整 counter 的位宽（如使用 counter(6)或 counter(8)）可以改变波形的频率和幅度（同时会改变）（这里的频率如果不理解可以简单看作是横轴的拥挤程度，越拥挤表示频率越高，反之则越小）。

如果我们想单独调整幅度可以调整 waveReg 的累加值，而如果我们想调整频率则可以调整 counter 的累加值，例如下面这个图中我们的频率比上面的波形图的频率更高，是其四倍。



具体的做法我们将作为实验任务，提供给同学进行实现。

任务一：

用 Counter 实现三角波发生器，满足以下要求：

1. 频率是 src/main/scala/exp6/TriangleWave.scala 的八倍；

   根据 4.3 中的 TriangleWave 模块，如果我们想将频率变为两倍，可以将 18 行的 counter := counter + 1.U 改为 counter := counter + 2.U，而如果想变为四倍，则可以变为 counter := counter + 4.U。

2. 幅度是 src/main/scala/exp6/TriangleWave.scala 的两倍；

   根据 4.3 中的 TriangleWave 模块，如果我们想将幅度变为四倍，可以将 21 行和 23 行的 1.U 变为 4.U，如果想变为八倍，则可以变为 8.U。

3. 使用 gtkwave 查看模拟波形，并将展示你实现的效果，还需要展示波形的幅度值。

代码框架如下：

实验框架代码位于 src/main/scala/exp6/todo/ModifedTriangleWave.scala 中（注意端口代码不可修改！）：

```
1.  package exp6.todo
2.
```

```
3.  import chisel3._
4.  import chisel3.util._
5.
6.  class ModifiedTriangleWave extends Module {
7.      val io = IO(new Bundle {
8.          val waveOut = UInt(16.W)
9.      })
10.
11.     // TODO: fill your code...
12. }
```

修改完代码后，你可以使用下面的命令执行测试代码，并在 test_run_dir/TestModifiedTriangleWave_should_generate_VCD_file/ModifiedTriangleWave.vcd 找到生成的 VCD 波形。

```
mill MyChiselProject.test.testOnly exp6.todo.ModifedTriangleWave
```

测试代码内容如下：src/test/scala/exp6/todo/TestModifiedTriangleWave.scala

```
1.  package exp6.todo
2.
3.  import chisel3._
4.  import chiseltest._
5.  import org.scalatest.freespec.AnyFreeSpec
6.
7.  class TestModifiedTriangleWave extends AnyFreeSpec with ChiselScalatestTester {
8.      "TestModifiedTriangleWave should generate VCD file" in {
9.          val maxCycles = 2000
10.
11.         test(new ModifiedTriangleWave).withAnnotations(Seq(WriteVcdAnnotation)) { dut
    =>
12.             dut.clock.setTimeout(0) // setting it to 0 means 'no timeout'
13.
14.             for (i <- 0 until maxCycles) {
15.                 dut.clock.step()
16.             }
17.         }
18.     }
19. }
```

如果你觉得生成的波形太短，可以修改第 9 行的 maxCycles 为更大的值，这里采用的是 ChiselTest 进行测试，具体的语法我们将在后面的实验进行介绍。