一、引言

1 研究背景

1

2

2 国内外研究现状

3 本文组织架构

本文共有六个章节，各章节概述如下。

第一章引言即本章，主要描述相关的背景内容，包括FPGA测试的概念，国内外各类测试平台概述，本文所在的BR0101测试平台概述，以及本文所做的工作等。

第二章整体框架介绍，主要描述本文构建的测试波形可视化系统整体框图，基本工作流程采用的，包括个。

第三章后端设计实现，主要描述本文使用的服务器后端运行的相关软件系统组件的原理以及相关的工具。

第四章前端设计实现，主要描述本文使用的浏览器前端运行的相关软件系统组件的设计思路与运行流程。

第五章实验结果与分析，主要描述波形可视化系统与BR0101测试平台整体软件系统的整合，使用若干个测试用例进行的波形绘制、控制与测量过程，以及相应的性能指标的测定与提高。

第六章总结与展望，总结本文工作的贡献与不足，以及未来的改进方向。

二、整体框架介绍

本章是对本文设计的波形绘制、处理与分析系统的整体框架的介绍。作为BR0101测试平台的一个功能组件，本文设计的系统并不是独立存在的，必须与BR0101测试平台的其他组件达成良好的协作。本章将简要介绍BR0101测试平台的相关背景，本文设计的系统的工作流程，以及本文设计的系统与BR0101测试平台其他组件的交互和依赖关系，作为后续模块设计讨论部分的基础。

1 BR0101测试平台概述

BR0101测试平台是由复旦大学专用集成电路与系统国家重点实验室的课题组自主开发的一套基于FPGA的远程测试平台。其硬件部分主要由一颗作为主控IC的Xilinx Zynq-7000® Z-7100 FPGA与一颗作为测试向量发生器IC的Xilinx Virtex-4® XC4VSX55 FPGA及其它外设构成，待测IC需要通过一个装载板连接到测试平台。软件部分则是运行在主控IC中的ARM软核上，包括嵌入式Linux系统，基于Python Django框架的服务器软件，另有需要终端用户在浏览器中运行的前端软件。

使用BR0101测试平台进行测试时，由嵌入式Linux系统发起控制命令，测试向量发生器IC接到控制命令及测试图形后生成具体的测试向量并下载到待测IC。待测IC中的信号引出后被测试向量发生器IC收集生成测试结果，主控IC则将测试结果反馈给用户。对用户而言，他们只需要提供待测IC的位流以及发起测试任务，就能获得测试结果，测试平台本身是不会被感知到的。

BR0101测试平台的设计采用了模块化的思想。在系统层级，测试平台按照功能进行软硬件划分，软件模块与硬件模块只进行控制流与数据流交互而不涉及对方的实现细节；在软件层级，又可以分为嵌入式系统、服务器后端与浏览器前端，三个模块同样只进行控制流与数据流交互；在服务器后端模块内部，测试流程的模块、用户模型模块、波形绘制模块等组件也是如此。BR0101测试平台软硬件的模块化特性的优点是黑盒化，每个模块的功能可以单独开发，只需要约定模块间传递的信息的格式即可完成跨模块的协同，模块内部的实现细节只需要由该模块设计者掌握，其他设计者可以在不了解此模块的具体工作流程前提下使用模块。黑盒化的模块可以缩小系统bug的排查范围，简化多人协作开发的流程。

2 波形可视化系统的工作流程

波形可视化系统作为BR0101测试平台的一个子模块，其功能是明确而专一的：提供基于Web前端的波形绘制与分析功能。为了达成这一目标，既需要设计实现前端浏览器的波形绘制与分析程序，也要设计实现后端服务器的数据处理程序及其接口。整个波形可视化系统需要在前后端协作下完成其功能，具体的实现流程如图所示。

此流程中，以用户在浏览器中打开结果页面为起点，前端向后端传递一个目标文件名，在后端中进行BR0101的其他模块通过适当处理。后端处理完毕后，通过Django框架的模板渲染机制将后端数据发送回前端。返回的后端数据中除用于生成页面内容的变量外，还包括两个重要的变量：一个是环回的目标文件名，它与前端请求时发送的文件名完全一致；一个是布尔值，它是通过判断目标文件名是否对应一个有效的VCD文件产生。若布尔值为真，则目标文件名会再次传递到后端，并且调用VCD文件解析器得到一个JSON格式的波形数据包，并且前端中的波形绘制组件也将运行起来输出最终的波形结果；若布尔值为假，则不会传递进行目标文件名，且波形绘制组件将隐藏。

3 波形可视化系统与其他组件的交互和依赖关系

波形可视化系统的运行需要若干前端参数与后端参数，这些参数需要通过BR0101测试平台其他组件以适当方式传递给波形可视化系统。与波形可视化系统存在控制流或数据上的依赖的其他组件已在图中标记，其中一部分组件同时具有前端和后端部分。现具体解释其依赖项。

前端组件中，文件树模块提供文件的上传与打开功能，并且传递被打开的文件的文件名到后端，波形可视化系统利用了这一机制来获取并检验VCD文件；工程流程模块负责发送执行相关测试流程的指令，在代表测试波形的VCD文件损坏或丢失的情况下提供重新生成VCD文件用于波形绘制的途径；文件预览编辑模块将被打开的文件的内容以适当形式显示，熟悉工程所用的文件格式的用户可以直接阅读比对文件内容，判断波形绘制结果的正误，同时提供文件。

后端组件中，用户模型管理模块控制当前用户的测试工程信息与工程文件夹路径，在前端打开的所有文件的路径都需要经过该模块处理；文件树模块负责加载当前用户的工程文件夹内容，包括波形绘制所需的VCD文件；文件预览编辑模块读取被打开的文件并将二进制文件的数据转化为可阅读的ASCII文本，处理后结果可以在前端显示。

三、后端设计实现

本章是对本文设计的波形处理与分析系统的后端设计部分的描述。BR0101测试平台的服务器后端软件系统是一个基于Django后端框架构建的系统，包括用户模型管理、测试任务调度、硬件驱动、测试结果生成等。本文的工作是在已有的软件系统基础上，设计波形绘制与分析所必需的解析器代码模块，以及相关的服务器接口模块，并利用Python语言的模块化特性，整合进BR0101测试平台的服务器后端软件系统。

1波形数据格式概述

1 VCD格式波形数据

BR0101测试平台产生的波形数据以VCD（Value Change Dump，值变转储）格式保存在服务器中。VCD格式是根据1996年[IEEE](https://en.wikipedia.org/wiki/IEEE)制定的1364标准中规定的一种转储文本格式，它将波形的变化状况以ASCII文本形式记录下来。一般的，一个完整的VCD文件包含三个模块：头模块，变量定义模块和值变模块，各个模块之间的划分是根据特定的关键字实现的。头模块包含timescale等与测试相关的背景信息，变量定义模块为每个信号定义了一个代表符号，值变模块则记录了各个信号具体的变化历史，其中模块首部的$dumpvars关键字提供了各个信号的初始值，而总线信号的值含有前缀字符b。

一个完整的VCD文件示例如图所示。左中右三图分别对应头模块，变量定义模块和值变模块。根据实际波形绘制的需求，将该VCD文件所包含的波形信息整理如表。

表

|  |  |
| --- | --- |
| 模块名 | 信息 |
| 头模块 | timescale=1ns |
| 变量定义模块 | 定义有clk和en两个信号 |
| 值变模块 | clk信号：初值0，10ns为1，20ns为0，30ns为1，40ns为0 |
| en 信号：初值0， 30ns为1 |
| 测试向量总长度为50ns |

将该表格中的信息适当整理，即可形成适合于浏览器前端绘制流程的波形数据。解析VCD文件采用的算法流程将在节中叙述；算法的输出是一个JSON数据包，将对此格式做进一步的说明。

2JSON格式的波形数据

经过VCD文件解析算法处理的数据是以JSON数据包的形式发送回用户端浏览器。JSON格式是一种轻量级的存储和交换文本信息的格式。在语法上，JSON使用了与创建 JavaScript 对象相同的键值对语法结构，使得JSON具有良好的可读性。目前包括Python、JavaScript在内的多种语言均允许JSON作为一种输入输出的格式，使得JSON具有良好的跨语言移植性。

根据本文的整体框架，解析出的JSON数据包应当包含以下两方面的内容：一是VCD文件包含的信号，二是各个信号的变化历史，即对应于VCD文件中的变量定义模块和值变模块。VCD文件中其他的信息与波形绘制和分析的关系不密切，在解析时可以丢弃。使用本文提出的算法将节中的示例VCD文件进行解析，得到的目标JSON数据包示例如图。该图中的波形信息与表的波形信息等价。

2VCD文件解析算法概述

根据VCD文件的格式特点，可以编写一个解析器提取其中的波形变化数据。在的讨论中，已经知道VCD文件是由三个模块组成，而每个模块的格式略有差异，因此解析器可以分解成三个相对独立的子解析器，由子解析器分别完成对应模块的解析工作。本节即围绕三个子解析器的设计展开讨论。

1头模块子解析器设计

头模块无论是其规模还是信息量都相对较小，故对其进行解析的流程较为直接。由图可知，头模块包含若干个关键字，不同关键字对应了相应的信息，其中只有$timescale关键字下含有有效信息。故设计如图所示的解析的流程。

此流程中，解析器逐行读取VCD文件，直到遇见$timescale关键字；$timescale关键字的下一行保存有timescale数据，可以提取出timescale的时间值与单位；最后，解析器继续读取VCD文件，直到进入变量定义模块，头模块子解析器退出。

2变量定义模块子解析器设计

变量定义模块是介于$scope关键字和$enddefinitions关键字之间的内容，其中被定义的变量均通过$var关键字引导的变量定义行完成。因此该模块子解析器的要点是正确提取变量定义行包含的内容。通过观察，可以看出变量定义行具有固定的格式，出于简便起见，本节使用BNF范式描述变量定义行的语法如下。

<变量定义行> ::="$var" <信号类型> <位宽> <信号符号> <信号名> "$end"

<信号类型> ::= "reg"|"wire"|"parameter"

<位宽>::= 数字|[数字]

<信号符号> ::= ASCII字符|[ ASCII字符]

<信号名> ::= Verilog标识符| ["["总线宽度"]"]

<总线宽度> ::= (数字":"数字)

根据以上语法描述，变量定义行是规定了信号名与符号之间的一一对应。在Python语言中，一一对应关系可以通过字典这样数据结构表示，因此只需要建立一个字典，以信号名作为键，符号等其他信息作为值。解析的流程如图所示。

对每一个变量定义行的分析步骤都是统一的。由于变量定义行的各个组成部分是通过空白符分隔，使用Python字符串处理函数split()可以直接将变量定义行划分成相互独立的数据片段，再根据变量定义行的语法将数据片段对应到各个组成部分。最后，向字典中添加新的键值对，用于值变模块子解析器的数据查询。

3值变模块子解析器设计

值变模块是VCD文件的主体，其解析器的设计也较为复杂。同样使用BNF范式描述其语法如下。

<值变模块> ::= <值变单元> |[<值变单元>]

<值变单元> ::= <时间行> |[<值变行>]

<时间行>::= "#" 数字|[数字]

<值变行> ::= [<总线标记>] <信号值> [空白符] <信号符号>

<信号值> ::= ("0" | "1" | "x" | "z") | [<信号值>]

<总线标记> ::= "b"

<信号符号> ::= ASCII字符|[ ASCII字符]

根据以上语法描述，值变模块是若干个值变单元组成，每个值变单元包含一个时间行和若干个值变行，表示在时间行所示时刻，值变行所涉及到的信号已发生了跳变。值变行则保存了信号跳变后的新值，总线与普通信号的值变行的格式略有差异，体现在总线标记的有无，但不影响对值变行的解析。解析的流程如图所示。

解析值变模块的实质是解析每个值变单元。时间行的识别和解析都比较简单，若一行以字符#开头则判断为时间行，#字符后为时间值。两个时间行之间的各行均为值变行。值变行可以细分为两类：一是总线信号行，一是普通信号行。总线信号行以字符b开头，且信号符号与信号值之间有空白符，而普通信号行既没有总线标记也没有信号值后缀的空白符。因此解析器也要对两种值变行分别处理：对总线信号行提取介于字符b与空白符直接的数据作为波形值；对普通信号行提取该行的第一个字符作为波形值。波形值提取完毕后，还要遍历字典，找到与当前信号符号对应的信号名，在该信号的波形值列表中添加新提取的波形值。

在值变模块解析完成后，可以根据解析得到的字典对象，生成最终的JSON数据包。由于字典对象中含有冗余信息，在JSON生成步骤之前，尚需去除冗余，以提高传输效率。显然，信号名与信号符号在VCD文件中是等价的，但是波形绘制与分析等前端功能不需要信号符号；普通信号的状态分类值与具体信号值也是等价的，两者取其一即可，但为了减少前端设计的代码冗余度，保留了状态分类值。将上述冗余通过del语句从字典对象中去除，才能调用Python标准库中的相关函数输出JSON数据包。

接口实现

本章叙述的解析器代码模块并不是Django框架所必需的模块。换言之，对Django框架而言，解析器代码模块是一个独立的第三方模块，它要与Django框架产生联系就需要通过Django框架内的标准调用此模块完成。因此需要设计一个封装有解析器代码模块的接口模块，由此接口完成与Django框架的关联。

在Django框架规范中，views.py这一文件集中了各类后端业务逻辑，每个具体的处理函数应当定义在或导入到该文件中，且必须接受一个名为request的参数。与此同时，这些处理函数还需要在urls.py中绑定到一个url，当浏览器前端通过某个url向后端发送请求时，Django框架就可以调用对应的处理函数。请求的内容则通过request参数传递，该参数是一个Python字典，包含请求的类型和请求参数。

本节实现的接口模块的处理流程如图所示。该流程中从浏览器前端获取的数据是待解析的VCD文件的文件名，为了传递该文件名，采用环回手段，即先在页面加载时，通过Django框架中模板渲染机制将当前文件名渲染入前端页面，前端页面得到校验结果后通过AJAX技术发送新的请求，将当前文件名重新发回后端Django框架，其原因将在节中说明。后端在第二次响应请求时返回前端的则是解析VCD文件得到的JSON数据包。

四、前端设计实现

本章是对本文设计的波形处理与分析系统的前端设计部分的描述。实现波形可视化功能并不一定需要复杂的前端设计，但是为了使波形可视化系统具有一定的交互性，需要引入前端行为；波形可视化系统还需要具有较为整洁美观的外观，这可以通过适当的样式设计完成。本文的工作是设计波形绘制与分析所必需的前端行为与样式，以及相关的服务器数据传输模块，并利用Django框架的模板渲染机制，整合进BR0101测试平台的服务器前端软件系统。

1前端设计工具简介

前端设计工具主要包括HTML（HyperText Markup Language，超文本标记语言），JavaScript，CSS（Cascading Style Sheets，层叠样式表），它们分别控制前端页面的结构，行为与样式。本节对前端设计工具做一扼要说明，便于后续阐释具体的设计思路与内容。

1 HTML概述

HTML与传统意义上的编程语言不同，它是一种组织网页文档的标签的规范，通过适当使用标签以通知浏览器如何正确展示页面的内容。在HTML中，标签是基本的语义单元，一个标签必须被一对<>符号包围，标签内需要声明该标签对应的元素名称，并且可以以键值对的形式附加元素的属性。由于HTML中预定义了基本的元素的含义及其显示方法，只需要在HTML文件中以标签的形式安排这些元素，浏览器就能将它们显示在页面上，并且允许JavaScript和CSS修改其行为与样式。

本文采用的HTML版本是HTML 5，该版本是。相比于其他版本的HTML，HTML 5的重要变化是引入了若干支持多媒体的新标签类型，其中最为重要的是canvas标签，它是本文的波形可视化功能的载体。节是关于canvas标签和波形可视化功能的实现的相关叙述。

2 JavaScript

JavaScript是一种运行于浏览器中的脚本语言，它最初由网景公司设计，后经Ecma国际确立为ECMAScript标准。完整的JavaScript实现除ECMAScript标准外，还包含文档对象模型（DOM）与浏览器对象模型（BOM）。JavaScript采用了类似C语言的语法，具有面向对象的特性，且可以增强网页的交互能力与动态能力。浏览器能够接受用户触发的各种输入事件，这些输入事件能派发到适当的JavaScript函数中进行处理，使用户感受到交互性的存在；另一些事件由定时器触发，经过适当的JavaScript函数处理，用户则可以感受到动态性的存在。现代的网页一般都需要具备交互性和动态性，这给JavaScript的兴起宇广泛应用创造了条件。

本文所完成的前端设计内容主要是各种波形绘制与分析所必需的前端行为，因此必须采用JavaScript作为这些行为的实现语言。相关的介绍将在节中叙述。

3 CSS

CSS的作用是定义显示 HTML 元素的方式。浏览器中定义了一组默认的显示 HTML 元素方法，然而实际的网页显示一般需要自定义HTML 元素的样式，此时需要通过CSS进行配置，使页面元素具有较为美观的样式。

CSS具有高度规整的结构。一条CSS 规则由选择器和声明两个部分构成。选择器用于匹配HTML 元素，一个选择器可以匹配多个HTML 元素；声明则由大括号包围，每条声明规定了被匹配的HTML 元素所赋予的样式，包括颜色，大小，相对位置等。CSS 规则具有优先级区分，使得多个选择器匹配到同一个HTML 元素时，高优先级规则能覆盖同名的低优先级规则，以保证网页显示的统一性。

本文所完成的前端设计内容主要是各种波形绘制与分析所必需的前端行为，因此必须采用JavaScript作为这些行为的实现语言。相关行为的介绍将在节中叙述。

2设计实现

1波形绘制功能实现

波形绘制是波形可视化功能的核心。在HTML前端中，有多种实现绘图的方式，其中较为成熟的方法是利用canvas元素提供的绘图功能实现。为提高设计与运行效率，本文拟采用canvas元素作为波形可视化功能的载体，通过canvas元素提供的线条、文本等基本要素绘制功能，实现一个较为完备的前端波形可视化系统。

1 canvas元素简介

canvas元素的概念在发展早期就已经提出，最早是，后，直到最终并入HTML 5标准。canvas元素的实质是在HTML页面上插入的一个动态的绘图区域，本身并不具有任何绘图能力，其内容取决于用户所调用的JavaScript脚本。要修改canvas元素所呈现的内容，必须通过DOM获取canvas元素的一个context对象，利用context对象的成员函数间接控制canvas元素中的内容。本文主要是利用canvas元素绘制线条、文本等以构成波形图像，相应的，要使用到context对象的以下成员：

由于在ModelSIM等仿真软件中，普通信号与总线信号的波形有一定的差异，因此本系统对普通信号与总线信号分别采用了不同的绘制方法，目的是使绘制结果具有与传统的仿真软件接近的风格，降低用户使用本文设计系统的难度。

1普通信号绘制

普通信号的状态只有四种：0、1、x、z，因此波形的状态也是四种，只要确定某一时刻的信号状态，波形的状态也能唯一确定，相应的波形图像也就唯一确定了。

文献是几种模拟量的可视化，其共同点是使用JavaScript脚本计算区域内的待绘制点，通过描点连线生成图案。本文实现的测试波形虽然是数字量绘制，但就其本质而言与模拟量是一致的，并且由于数字量的取值离散，绘制时所取的待绘制点不需要过多的数学计算， 通过预定义的常量即可获得。相应的处理流程如图所示。

进行绘制前，需要获取欲绘制的时间范围以及此波形的状态变化列表，根据时间范围精简待绘制的波形状态数目。在描点阶段，按照各状态的起始时间计算起点的横坐标，纵坐标则由当前波形状态查表得出，波形图像的颜色则按照ModelSIM配色进行分配，0和1使用绿色，x使用红色，z使用蓝色。连线阶段的工作是根据坐标和分配的颜色绘制线条，通过context对象的两个成员函数可以由计算出的起点与终点坐标完成连线。

2总线信号绘制

总线信号的绘制与普通信号有所区别。前述普通信号的绘制中，无论波形的状态是四种中的任何一种，完成绘制的操作都仅包含一段横线与一段竖线，而总线信号的绘制则涉及到更多的要素。

在ModelSIM等仿真软件中，总线信号波形的组成包括以下几部分：总线体和信号值，其中总线体是由一对平行线和若干段斜线构成的一个封闭图案，信号值则由信号的位宽和信号的实际值共同组成。无论信号值怎样改变，总线的图案都是相似的，只有平行线部分的长度以及选用的颜色在变动，这个特点使总线信号绘制时只需要通过各段的起始和结束时间计算平行线起点与终点的横坐标，纵坐标总是取特定的值以构成平行线。相应的处理流程如图所示。

此流程与普通信号绘制有相似之处，都需要计算绘制的时间范围和的信号值的变化列表，标记出需要绘制的点的横坐标与线的颜色。不同之处在于，普通信号绘制时，笔画是连续的，只有在更换绘制的线的颜色时调用闭合路径函数以完成切换；总线信号绘制时，必须分段绘制，这是因为同一段内总线信号的取值不发生变化，一旦发生变化就说明要进行下一段绘制，而各段都是封闭图案，使用统一的绘制函数显然无法构成连续笔画。信号值位置与平行线起点一致，但是如果平行线长度较小，无法完整显示信号值文本，则需要裁剪信号值文本并使用省略号作为信号值文本的后缀，否则将导致不同段总线信号的信号值文本重叠。

1波形控制功能实现

实现了波形绘制功能，但是仅有绘制功能并不能体现出Web可视化的优点，需要引入具有交互的波形控制功能以充分利用前端的灵活性。本文设计的系统实现了基本的波形控制功能，包括波形平移、波形缩放、波形排列、波形跳转等。本节将对上述控制功能分别展开分析。

1波形平移

波形平移从结果上与数学上对函数图像的平移是相近的，但是波形平移只涉及绘制中各点的坐标变换，波形数据本身没有改动。触发波形平移的条件有两个：用户通过在canvas元素上拖拽触发和用户通过移动滑杆触发。相应的处理流程如图所示。

触发这两个条件之一时，需要更新绘制的时间范围。若为拖拽触发，则记录鼠标拖拽过的位移，求出此位移与波形图像宽度的比例，时间增量计算为当前绘制的时间范围的长度乘以所求比例；若为移动滑杆触发，时间增量则是通过滑杆位置的变化量确定，具体计算方法与拖拽触发一致，但以滑杆位置的变化量代替鼠标拖拽过的位移，滑杆长度代替波形图像宽度。确定新的时间范围后，按照正常的绘制流程重新绘制当前波形。

2波形缩放

波形缩放是ModelSIM等仿真软件中常用的操作，因此本文设计的系统同样要具有缩放功能。缩放操作包括放大和缩小两种操作，它们对绘制的时间范围的计算流程是有所区别的，但是放大与缩小的触发条件是相同的，都是用户通过在canvas元素上拖拽触发，因此可以与拖拽平移共用一个入口函数。相应的处理流程如图所示。

触发缩放时，会根据拖拽缩放的方向决定是放大还是缩小。若从左向右则判断为放大，将鼠标按下与松开的坐标按与波形图像宽度的比例转化为两个偏移量，偏移量先与原有的时间范围相乘得到时间增量，起始时间与时间增量相加则为新的时间范围；若从右向左则判断为缩小，计算新的时间范围的过程是放大的逆过程，同时要进行边界检查，若超过了从VCD文件中解析出的测试向量总长度必须修改回合理值，其中合理值的下限是0，上限是测试向量总长度，以免由于出现访问越界的错误，导致波形图像无法绘制。最后按照正常的绘制流程重新绘制当前波形。

3波形排列

波形排列不影响波形图像，是对整个波形显示区域的布局调整。在ModelSIM等仿真软件中，各个信号的名称一般是与其波形图像分离的，这样的优点是简化绘制时坐标的计算过程，并且提供了一定的样式自由度。因此波形排列操作的触发条件与波形平移缩放不同，不需要拖拽波形图像本身，而是通过拖拽信号名元素来进行触发，再根据拖拽的结果反馈到波形图像的布局中去。相应的处理流程如图所示。

在一次信号名拖拽动作中，可以分解成两个先后发生的事件：鼠标的按下与弹起。按下事件中，记录按下时鼠标所在的元素（记做甲元素）；弹起事件中，同样记录弹起时鼠标所在的元素（记做乙元素）。获得了两个元素后，判断能否拖拽。允许拖拽的条件较多，但禁止条件只有一种：拖拽双方中，一方是一条总线下的某个分支，另一方则不是该总线下的任何一个分支。若判断为允许拖拽，则将甲元素放置于乙元素之前，再根据拖拽的结果反馈到波形图像的布局中去。若拖拽双方其中一方是总线，其下属所有分支也需要参与到拖拽。

4波形跳转

波形跳转可以视作精确的波形平移或波形缩放。在页面中，提供了直接修改当前时间范围的入口。该入口提供了两个文本框，允许以文本形式输入一个时间范围，波形图像可以直接切换至所提供的时间范围。相应的处理流程如图所示。

默认情况下，波形跳转使用的HTML元素都是隐藏的，显示出来的是文本格式的当前时间范围。只有当鼠标点击了当前时间范围时，这些元素才会出现。文本框默认显示的同样是当前时间范围，可以手动输入一个目标时间值及其单位，在点击按钮提交后，相应元素再次隐藏。同时输入的时间值要通过单位转换，得到与VCD文件中提供的timescale的单位一致的时间值。最后利用此时间值，按照正常的绘制流程重新绘制当前波形。

相比于波形平移或波形缩放，波形跳转有更高的精确度，其控制精度可以与VCD文件中提供的timescale值一致，且要得到相同的时间范围只需进行一次操作，较为简便高效，代价则是必须明确待观察的时间范围，在时间范围不够明确时优势不明显。

5滚动条同步

严格意义上滚动条同步不属于波形控制，因为所有的波形图像都没有被改变，改变的是波形显示区域的显示属性。在HTML中，一个容器中可以添加任意多个HTML元素，他们在竖直方向上依次排布堆放，其结果是页面高度被拉长。当元素数量较多时，页面中的其他元素无法与该容器同时显示出来，用户交互的难度增加，布局的美观度受到影响。因此需要增加高度限制以抑制高度的过度增长。相应的处理流程如图所示。

在本文设计的页面中，待显示信号较少时，页面尚可保证所有的信号都能正常显示，这是通过浏览器自动控制容器高度增长实现的；当信号数较多，容器实际高度达到预设的高度限制值时，容器显示出的高度不再增长，并自动出现滚动条。

根据节的叙述，信号名与波形图像分别属于两个不同的容器，因此他们都需要进行高度限制，且显示的位置应当保持同步。在ModelSIM等仿真软件中，信号名与波形图像显示位置是通过一个全局滚动条同时控制的，而本文仅保留信号名容器的滚动条，拖动信号名容器的滚动条时，波形图像容器进行跟随，以代替全局滚动条的功能。相应的处理流程如图所示。

1波形分析功能实现

可视化系统也提供了一定的波形分析能力。实现了基本的波形分析功能，包括光标模式、波形值读取等。本节将对上述分析功能分别展开讨论。

1光标模式

在ModelSIM等仿真软件中，提供了光标模式。通过光标能够获知当前光标所在的时刻以及各个信号的波形值，这是由于光标元素的颜色与波形图像的背景色具有较高的对比度，便于人眼观察。相应的处理流程如图所示。

类似于波形缩放，光标模式需要通过鼠标在canvas元素上动作触发。获取当前时间的方法也与波形缩放接近，但是仅保留鼠标松开的坐标及其对应的时间值。获取坐标后，通过设置光标元素的CSS属性，将光标元素移动到相应的位置上，并且更新波形图像下方的文本框内容，显示当前光标所在的时间值。考虑到光标操作与波形控制动作实际上是共用一个入口，为了避免混淆，在页面中添加了一个控制菜单栏，必须先选择光标模式才能出现光标和执行光标操作。

2波形值读取

波形值读取功能依赖于光标模式的正确实现。在光标激活时，可以将光标所在的坐标上的各个信号值记录下来，以文本形式呈现，便于观测和记录测试结果。相应的处理流程如图所示。

根据节的流程解释，在执行光标移动操作时，是保留了光标所在的时间值的。以此时间值为基准，可以将各个信号的波形数据划分成两个区段：早于此时间值和晚于此时间值。待读取的波形值即为早于此时间值区段中最后一个波形值。页面中亦有保存波形值的容器，根据所属的信号名填充波形值容器，最终体现在页面的效果相当于当前波形值随光标位置实时改变。

接口

在已经简要介绍了后端接口的实现。本节介绍前端接口的工作流程。

根据图，前端组件中的都需要相应的后端组件协同。但是，本节定义的前端接口必须是能够主动同后端组件发生交互，而它们的全部协同动作是在浏览器加载页面时通过浏览器发送请求完成的，页面加载完毕后前后端组件不再发生交互，因此不作为前端接口看待。

波形可视化系统所需的前端接口的功能是配合节的后端接口，即发送被打开VCD文件的文件名，并在请求被后端响应后，通知波形绘制分析模块进行绘制操作。实际上，在节中介绍的文件预览编辑模块与波形绘制分析模块都依赖于同一个文件名，但是波形绘制分析模块完成其功能需要通过AJAX技术向后端进行第二次请求，这是因为文件预览编辑模块适用于所有的文件格式，而波形绘制分析模块不接收除VCD文件外的任何输入文件，且在打开非VCD文件时是隐藏的。这个行为需要在前端中实现，如果仅向后端发送一次请求则不能实现。采用两次请求的另一个理由是VCD文件是可以在测试流程中实时生成的，两次请求时用户可以不刷新页面就获取新生成的波形图像。

五、实验结果与分析

本章是对本文设计的波形处理与分析系统的实际测试。进行的测试包括两方面：一是通过与BR0101测试平台的测试流程整合联动，检验系统功能的正确性与完备性；二是通过处理若干个标准的测试样例，获取系统的性能指标，并进行一定的分析与优化。

1. 系统与BR0101测试平台的整合

本文设计的系统包含前端和后端两部分，需要分别整合进入BR0101测试平台。

进行后端整合时，由于Django后端框架是使用Python语言编写，而Python语言是模块化的，只需要将新的后端处理代码保存在相应文件夹下，使用import语句将其作为一个子模块导入即可完成整合，对既有代码的影响降到最低。后端部分的文件数目相对较少，且相互关联性强，可以统一放置在一个文件夹下，对后期维护的影响很小。

进行前端整合时，根据Django后端框架的规范，HTML文件应当保存在template文件夹中，而JavaScript脚本和CSS样式表应当保存在static文件夹中，按此规范提交相关前端文件后才能被Django框架识别。由于Django框架是利用模板渲染机制来获取保存在static文件夹中的文件，相应的需要在HTML文件添加模板渲染的占位符。与后端相反，前端部分的文件数目相对较多，且是几个较独立的功能块，除了都需要加载到页面外，基本没有共同点，但是为了后期维护管理方便，将它们按照文件类型分别保存。

整合结束后，浏览器打开测试页面时，页面内容将按照如图所示流程完成后端响应与前端渲染，用户所见的将是一个功能完整的页面。

图为整合后的测试APP的界面。在此页面中进行章介绍的各类操作，均可获得与介绍一致的结果，模块运行无故障。

BR0101测试平台内部的后端逻辑中，集成有一个波形处理绘制模块，其处理流程与本文的工作有相似之处，整理如图。该模块与本文提出的系统的比较如表。

图

|  |  |  |
| --- | --- | --- |
|  | BR0101原有模块 | 本文系统 |
| 波形结果生成场合 | 后端，进行测试时 | 前端，用户打开相应页面时 |
| 信号数据表示方式 | 每个时刻的信号值均保存 | 仅保存信号跳变时刻 |
| 后端向前端发送的数据 | 图片文件本身 | JSON数据包 |
| 波形结果的呈现形式 | JPG格式图片 | canvas元素 |
| 波形结果的交互性 | 无 | 有 |
| 波形结果能否测量 | 否 | 能 |

可见，原有的流程具有冗余数据多、数据传输量大、灵活性差的缺点，本文提出的流程则以延长一定的前端加载时间为代价，部分克服了以上的缺点。表是对个测试样例分别按照新旧两种流程运行时，波形结果显示完毕所需的时间与加载的数据量，可以验证以上的分析。

|  |  |  |
| --- | --- | --- |
| 测试样例 | BR0101原有模块耗时 | 本文系统耗时 |
| AccuShift |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| 测试样例 | BR0101原有模块数据传输量 | 本文系统数据传输量 |
| AccuShift |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

2后端性能分析与优化

本节对本文的后端设计部分中的VCD文件解析器进行性能分析与优化。

根据的讨论，接口模块是按照Django后端框架的规范编写的，为解析器提供与整体流程交互的接口，其性能不完全由自身代码决定，而是受到后端框架其它组件影响。而且接口的代码量非常小，对其进行性能分析的意义不显著，也无法对其性能做有效优化。

解析器部分则不同。它的代码量适中，执行过程较少和其他Python模块产生关联，主要使用了Python内建函数与数据类型，且是本文的后端设计的主要内容，对其进行性能分析与优化具有一定的价值。

Python标准库中，profile模块提供了基本的性能分析数据记录。该模块通过统计每个函数被调用的频次及其执行时间，生成一份文字报告。报告中执行时间较长的函数一般是这个程序的性能瓶颈。利用profile模块获得的性能数据也可以进一步使用PyFlame模块处理，该模块可以将profile模块的输出转化为焰状图，通过观察焰状图中存在的“平顶”函数，找到潜在的性能瓶颈。

图是未经优化的解析器性能焰状图。容易观察到解析器的几个性能瓶颈是。与优化前相比，“平顶”的长度有所缩短，即整体执行时间减少。

除针对特定输入样例的性能分析外，解析器的算法复杂度也是性能讨论的一个方面。根据节的分析，VCD文件的头模块长度相对固定，变量定义模块的长度只与定义的变量总数有关，值变模块的长度既与定义的变量总数有关又与测试的时长有关。表与图是不同输入VCD文件下，三个子解析器的执行时间统计。根据图，可以验证此判断的正确性。

为了比较准确地得到解析器的执行时间与VCD文件中的模块规模的关系，另取了一组VCD文件作为测试用例。与真正的通过运行BR0101测试流程获得的VCD文件不同，这批VCD文件是由ModelSIM仿真软件导出后添加到测试结果文件夹中，且只包含若干个完全同步的信号，力图减少由于值变模块中各个值变单元中跳变个数不一致的影响。表与图是不同输入VCD文件下，变量定义与值变解析器的执行时间统计。使用软件进行拟合，可以求出一个经验公式：

3前端性能分析与优化

本文的波形处理系统具备了基本的交互性，包括用户触发的鼠标事件、按钮事件、滑杆事件等。显然，用户希望响应时间尽可能短，页面显示内容的帧频尽可能高，因此需要对系统进行性能分析与优化。

目前主流浏览器中都集成有所谓“开发者工具”菜单，其中提供了基本的运行时间记录，调用栈分析等性能分析工具。本节的前端浏览器环境是Firefox浏览器，其开发者工具界面如图所示。

与在后端进行性能分析类似，前端的性能分析也必须基于特定的输入样例，不同之处在于前端中的性能分析对系统本身的依赖性更强，负载波动等因素对分析结果干扰更严重。

表是，考虑到

六、总结与展望

本文

参考文献

致谢