# Wonton开发手册

2021年9月

目录

[Wonton开发手册 1](#_Toc83814652)

[1 项目结构 2](#_Toc83814653)

[1.1 ShapVLFD 3](#_Toc83814654)

[1.2 Wonton.Common 4](#_Toc83814655)

[1.3 Wonton.CrossUI.Web 5](#_Toc83814656)

[1.3.1 Controllers 5](#_Toc83814657)

[1.3.2 Extensions 7](#_Toc83814658)

[1.3.3 Models 8](#_Toc83814659)

[1.3.4 Services 8](#_Toc83814660)

[1.3.5 主程序 8](#_Toc83814661)

[1.3.6 ClientApp 8](#_Toc83814662)

[1.3.7 Resource 9](#_Toc83814663)

[1.3.8 Service 9](#_Toc83814664)

[1.3.9 Components 12](#_Toc83814665)

[1.4 Wonton.CrossUI.Web.HostApp 18](#_Toc83814666)

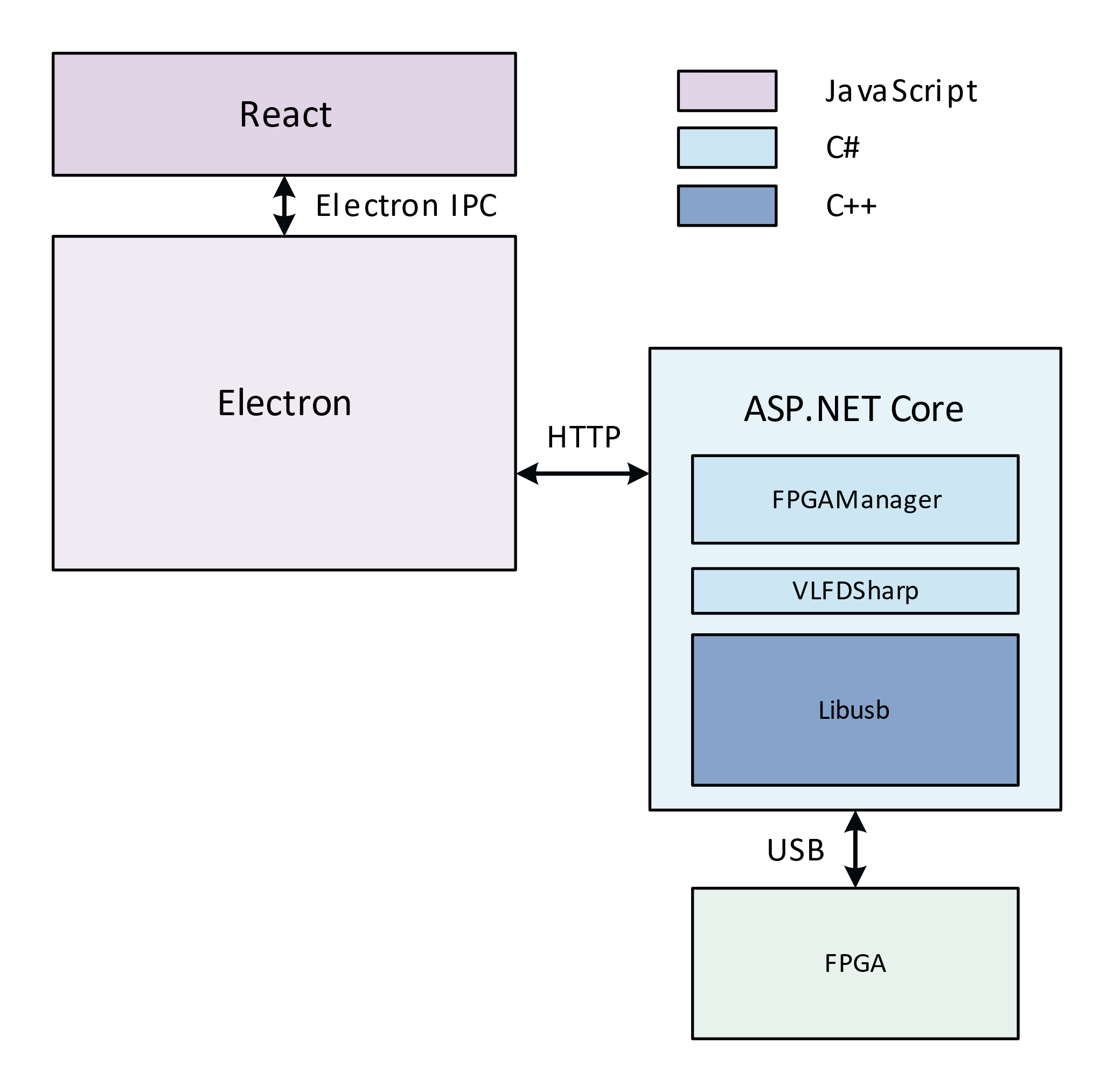
[1.5 Wonton.Test 18](#_Toc83814667)

[2 杂项 19](#_Toc83814668)

[2.1 安装脚本 19](#_Toc83814669)

[2.2 调试运行 20](#_Toc83814670)

# 1 项目结构



**图1 Wonton的项目结构**

这是Wonton软件的项目结构。简单来说，一共分为“四个框”、“三座桥”。“四个框”，分别是FPGA芯片、.NET程序、Electron界面、React组件。“三座桥”，分别是连接FPGA与.NET的USB接口、连接.NET与Electron的HTTP协议、连接Electron和React的IPC。

首先，FPGA芯片是整个系统的大脑。它负责数据的处理，其余的组件只是负责数据的传输和分类。但是本实验搭载FDP3P7芯片的VeriLite Fudan实验板只支持Micro USB接口传输数据。具体传输逻辑之后介绍。

其次，.NET程序是Wonton软件的后端，由C#语言编写。它负责前端界面和FPGA芯片之间的数据传输。其中又分为三个部分。

第一部分Libusb是通用的USB协议。

第二部分VLFDSharp封装的FPGA驱动程序。

第三部分FPGAManager是通过HTTP协议与Electron界面做交互的HTTP请求函数。Electron只需要调用FPGAManager的函数就可以调用由VLFDSharp生成的接口函数，并完成一些更具体的复杂操作。

接着，Electron界面是一个支持多平台的软件界面，基于JavaScript语言。其内核是浏览器内核，因此只要该系统装有通用浏览器，就可以安装并使用Electron，不需要对不同操作系统进行适配。

最后，React组件是和Electron搭配使用的器件布局语言，衍生自JavaScript的JSX语言。React可以生成标题、器件库、按钮等不同元件，并通过JavaScript和CSS语言进行排版布局，做出图形化的用户界面。

## 1.1 ShapVLFD

SharpVLFD是基于C#语言的封装后的驱动程序。

其中Native文件夹存有基于不同操作系统和架构编译而成的动态链接库。linux有arm64、armhf、x64、x86架构，默认为debian系的ubuntu系统；macos默认为11.2版本，10.14和10.15也有适配。windows也有x64和x86之分。

在该项目中有几个文件，分别为DependencyGraph、NativeExtension、NativeMethods、NativeMethods.Generate、PlatformApi、UnmanagedLibrary和VLFDInterop。

NativeMethods调用链接库，与FPGA进行数据交互。

PlatformApi负责探测所在系统或架构。其中.Net、Mono、Unity和Xamarin是C#的专属平台，对C#的语法都有特殊方法和解释。Linux、Mac OS X、Windows是操作系统。根据探测到的不同平台和系统对调用方法做出调整。一般我们会用到的都是Windows系统和.Net平台。

UnmanagedLibrary是SharpVLFD的核心。它负责根据不同操作系统和C#平台对链接库的调用方法做出调整。对于Windows系统，使用GetProcAddress；对于Linux和Mac OS X使用dlsym。不同的C#平台则调用对应平台的调用函数。

VLFDInterop负责各个接口函数的调用。Get方法来自NativeMethods里的定义。Span和Marshal则是C#中对指针的安全封装。因此，当需要对指针做精细操作的时候，需要加上unsafe的标识符。

DependencyGraph是针对Mac OS X系统的不同版本的动态链接库的管理。因为Mac OS X本身存在特殊性，它本身是一个闭环系统，对使用该系统的版本统一升级为更高的版本，而且不同版本之间存在兼容问题。所以需要对不同版本的Mac OS X系统进行编译的对应更新，开发软件时需要根据最新的系统版本进行适配。现在最新的Mac OS X系统是11.2版本。

NativeExtension则与PlatformApi、DependencyGraph协同，针对不同平台使用不同的NativeMethods。首先判断是什么平台的，Unity、Xamarin都有专属的调用链接库函数。如果不是官方平台的，则将其归类为UnmanagedLibrary，交给UnmanagedLibrary处理。再判断是什么系统的，Windows的链接库后缀为dll，Linux的链接库后缀为so，Mac OS X的链接库则交给DependencyGraph去索引。

## 1.2 Wonton.Common

Wonton.Common是Wonton软件中的一个命名空间。其中包含三个文件，分别是FPGABoard、FPGAException和FPGAManager。

FPGABoard是对于SharpVLFD给出的C#函数的又一封装。其中的函数简化了函数的使用逻辑，并且一旦函数运行出错后，不再简单地打印错误信息，而是将错误信息输出为Exception。这种Exception的包装规范了错误信息的输出，也可以重定向到Log日志中，并定义在FPGAException中。

FPGAManager主要包含了对RecentProjects的操作。在软件刚打开的时候，会检测曾经打开过的项目。历史记录存放在了软件专属文件夹中。由于该文件中的函数对于软件运行较重要，所以着重说明。

**表格 8 FPGAManager类的属性说明**

|  |  |  |
| --- | --- | --- |
| 属性名 | 属性类型 | 说明 |
| Board | FPGABoard | 自创类，见FPGABoard |
| CurrentProjectFile | string | 目前项目路径 |
| RecentProjects | List<string, string> | 项目历史记录 |
| RecentProjectsRaw | string | 历史项目的json集合 |

**表格 9 FPGAManager类的函数功能**

|  |  |  |  |
| --- | --- | --- | --- |
| 函数名 | 返回类型 | 参数类型 | 说明 |
| FPGAManager | 初始化 | void | 初始化Board和RecentProjects |
| AddRecentProject | void | string, string | 向RecentProjects属性中添加项目 |
| SaveRecentProjectAsync | Task | void | 将RecentProjects写入json文件 |
| ReadRecentProjectAsync | Task | void | 将RecentProjects.json读入RecentProjectsRaw |
| GetConfigDir | string | void | 获取保存上述文件的路径（Windows: User/AppData/Local/Wonton, Linux\Mac: home/.wonton） |

## 1.3 Wonton.CrossUI.Web

该命名空间存放着Wonton最重要的一些功能代码。现将其按所属功能分开说明。

### 1.3.1 Controllers

该文件夹下有两个文件，分别是FPGAControllers和WindowControllers。WindowControllers原本功能是界面的放大缩小，但该功能已在ClientApp中替代，择时将其删除。

FPGAControllers负责接收Electron前端发出的请求，使用的是HTTP协议，因此接收函数都以HttpGet开头。该文件下使用FPGAController类，现对其详细说明。

首先定义了ApiController的路径是api/[controller]，也就是前端的请求路径都是以api作为根目录的。

**表格 10 FPGAController类的属性说明**

|  |  |  |
| --- | --- | --- |
| 属性名 | 属性类型 | 说明 |
| \_logger | ILogger | 输出日志信息 |
| \_manager | FPGAManager | 调用FPGAManager |
| inputDict | Dictionary<string, int> | 输入引脚映射 |
| outputDict | Dictionary<string, int> | 输出引脚映射 |

**表格 11 FPGAController类的函数功能**

|  |  |  |  |
| --- | --- | --- | --- |
| 函数名 | 返回类型 | 参数类型 | 说明 |
| FPGAController | 初始化 | ILogger, FPGAManager | inputDict和outputDict都是固定映射，该映射规则查自SMIMS说明文档 |
| InitIO | FPGAResponse | int, int | 对存放输入输出数据的数组初始化 |
| Program | FPGAResponse | string | 烧录位流文件 |
| IoOpen | FPGAResponse | void | 打开实验板 |
| IoClose | FPGAResponse | void | 关闭实验板 |
| WriteReadFPGA | Task<FPGAResponse> | ushort\* | 读写数据，并将数据实时存入日志文件中，放入专属文件夹 |
| SetProjectFile | FPGAResponse | string | 设置当前项目 |
| Init | Task<FPGAResponse> | void | 打开软件时若无当前项目则读取历史项目 |
| WriteJson | Task<Response> | string, ProjectInfo | 保存项目信息为json文件 |
| ReadXmlToJson | FPGAResponse | string | xml格式转换为json格式 |
| NewProject | Task<FPGAResponse> | string, string, string | 新建项目，需要引脚约束文件和项目文件 |
| Waveform | Task<FPGAResponse> | int, string | 根据引脚约束文件、输入输出日志文件和运行时钟频率生成对应的vcd格式文件，用于波形调试，并调用GTKWave外部程序显示波形 |
| SplitRawData | List<int> | List<ushort> | 将FPGA返回的元数据从4个一组拆分为单个 |
| FilterRepetition | Dictionary<int, int> | List<int>, List<int> | 过滤两个时钟周期之间未更改的数据记录 |
| PrintVcd | string | Dictionary<int, string>, Dictionary<int, int>, int | 返回vcd所需格式数据记录 |

注：FPGAResponse是自定义类，定义在Wonton.CrossUI.Web/Models/FPGAResponse中，ushort是16位整数，ProjectInfo是自定义类，和FPGAResponse一起。

注意：这里WriteJson、ReadXmlToJson、NewProject将做出修改，可能现在用不上或不是最终版本。

RunExeByProcess是比较通用的调用外部程序的代码。

**图 27 FPGAController设计流程**

### 1.3.2 Extensions

该文件夹中只有一个文件名为WebHostBuilderExtensions。该文件主要用于HTTP协议的IP地址。

### 1.3.3 Models

该文件夹中有两个文件，FPGAResponse和ProjectInfo。它们都是前述FPGAController所用的自定义类的定义。实际上这两个自建类存储了一些必要的数据。

**表格 12 FPGAResponse类的属性说明**

|  |  |  |
| --- | --- | --- |
| 属性名 | 属性类型 | 说明 |
| Message | string | 成功或失败（也可以调试用） |
| ProjectPath | string | 项目路径（没用上） |
| Status | bool | 成功或失败 |
| Data | ushort\* | FPGA返回的元数据 |
| IsDarkMode | bool | 是否处于黑暗模式（为了该功能新加入的属性） |

ProjectInfo则只有一个string data，但为了可扩展性，姑且将其保留。

### 1.3.4 Services

该文件夹中有三个文件，分别是BridgeSttings、DarkMode和ElectronIPC。其中ELectronIPC已经因为性能原因而弃用。DarkMode则是因为Windows有不同于Linux和Mac OS X的自有的一套黑暗模式设置。因此仅对Windows系统的黑暗模式做出调整。Linux和Mac OS X的黑暗模式均由CSS文件设置，存于ClientApp中。BridgeSettings则设定了SocketPort和WebPort两个端口属性，交由WebHostBuilderExtensions使用。

### 1.3.5 主程序

Program和Startup是用于启动.Net、HTTP和软件界面的主程序，并且开启了一个logger日志进程，将调试信息输出到logs文件夹中。

### 1.3.6 ClientApp

该文件夹下所有文件都是由JavaScript和CSS语言写成，需对React、JavaScript、CSS、HTML语言都有一些了解。

### 1.3.7 Resource

该文件夹下是一些图片文件，负责显示软件缩略图等。

### 1.3.8 Service

该文件夹下一共四个文件，分别是Darkmode、FPGAManager、FPGAPortsMap、ProjectManager，功能对应着C#语言的Controllers所属文件。

Darkmode文件下两个函数，开启和关闭黑暗模式。操作是为该元素添加darkmode Class。而添加了darkmode Class的元件的颜色由CSS文件指定。

FPGAPortsMap存储了来自SMIMS说明文档提供的引脚映射，在C#的Controller里以Dictionary的形式写在程序里，而这里使用Map存放该映射，需要时导入。

FPGAManager里有许多非常重要的函数，在这里着重说明。

需要注意的是：原逻辑是需要什么就把什么当作属性存在FPGAManager里，因此属性之间的功能实际上是有重合的，也会导致意义不明和混乱，所以在下一版本中会将其底层逻辑进行更改。现存版本的函数功能依旧是需要的，只是会把实现的方式简洁化。

**表格 13 FPGAManager类的属性说明**

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 属性类型 | 说明 | 举例 |
| bitFile | string | 位流文件路径 |  |
| writeCount | int | 输入数据的计数器 |  |
| readCount | int | 输出数据的计数器 |  |
| subscribedInstances | Map | 响应输入的器件 | i1=>{data:[0], refresh:回调函数} |
| hardwarePortsMap | Map | 输出端口映射 | 32=>[{instance:”i2”, index:0} ……] |
| inputPortsMap | Map | 输入端口映射 | i1=>[31] |
| projectInstancePortsMap | Map | 器件与端口的映射 | i1=>[“sec\_out[0]”] |
| hardwareValues | Array | 输出数据 | [0, 0, ……, 0](64位) |
| inputValues | Array | 输入数据 | [0, 0, ……, 0](64位) |
| projectPortsMap | Map | 引脚约束映射 |  |
| projectInputPorts | Array | 引脚约束的输入 |  |
| projectOutputPorts | Array | 引脚约束的输出 |  |

注：这里的输入输出方向可能会造成困扰。这里定义，FPGA向外输出数据、软件读入数据的方向为输出，FPGA向里输入数据、软件写入数据的方向为输入。



**图 28 FPGAManager设计流程**

**表格 14 FPGAManager类的函数功能**

|  |  |  |
| --- | --- | --- |
| 函数名 | 参数类型 | 说明 |
| Update | void | 根据端口映射将输出数据分发到器件上 |
| MapOutputPorts | string, string, int | 更新输出端口映射 |
| GetProjectInstanceMapping | string, int | 返回projectPortName |
| GetProjectInstanceMappingAll | string | 返回projectPortName的集合 |
| MapInputPorts | string, int, string, int | 更新输入端口映射 |
| Register | string, int | 加载输入组件到输入端口映射上 |
| UnRegister | string | 写在输入组件 |
| RegisterProjectPorts | string, int | 加载任意组件 |
| UnRegisterProjectPorts | string | 卸载任意组件 |
| GenInputOutputGroup | void | 产生输入输出端口映射 |
| GetProjectOutputPorts | void | 查询所有输出端口 |
| GetProjectInputPorts | void | 查询所有输入端口 |
| Subscribe | string, Array, callback function | 加载输出组件 |
| UnSubscribe | string | 卸载输出组件 |
| UpdateInput | string, Array | 更新输入组件数据 |
| Cycle | void | 更新每个组件的界面值 |
| Program | string | 烧录位流文件 |
| IoOpen | void | 打开实验板 |
| IoClose | void | 关闭实验板 |
| InitIO | int, int | 初始化输入输出数组 |
| Split | Array | 将FPGA返回的数据从4个1组拆分为单个 |
| GenWriteData | void | 从单个数据组合为4个1组的数据用于输入FPGA |
| WriteReadData2 | void | 读写数据 |
| WriteReadData | Array | 读写数据 |

注：其中大量用到了数据收发的函数，需要和C#写成的FPGAController联合使用。

其中用到了许多async标识的异步函数，如果用作同步函数会阻塞其他功能，而且收发数据有设置响应时间，如果超出时间会报错。

因为即将对底层逻辑做出修改，这些属性和函数都有数据重叠，只是为了不同功能才加了这么多属性，所以不在这里详述了。

ProjectManager是和项目有关的管理代码，也是非常重要的一部分，但是由于项目文件存储器件布局的方法也会有所改变，因此其中也会有不少即将改变的函数和属性，在下一版本更新后会将这段一并修改。

**表格 15 ProjectManager类的属性说明**

|  |  |  |
| --- | --- | --- |
| 属性名 | 属性类型 | 说明 |
| projectPath | string | 项目路径 |
| bitFile | string | 位流文件路径 |
| constraint | string | 引脚约束文件路径 |
| projectName | string | 项目名称 |
| projectInitialize | bool | 项目是否初始化 |
| projectInfo | Object | 项目信息 |
| layout | Array | 器件布局 |
| layoutNoDevice | Array | 没有器件的xy布局（改） |
| recentProjects | Array | 历史项目 |
| isDarkMode | bool | 是否黑暗模式 |

**表格 16 ProjectManager类的函数功能**

|  |  |  |
| --- | --- | --- |
| 函数名 | 参数类型 | 说明 |
| GetInitData | void | 获取项目信息 |
| Initialize | void | 初始化 |
| ReadProjectIO | string | 读取项目数据（改） |
| MergeLayoutDevice | Array, Array | 获取正确长宽信息的布局 |
| WriteProjectFile | void | 保存项目 |
| ShowWaveform | int | 显示波形 |
| RegisterRefreshTitle | callback function | 更新更改标题的回调函数 |
| RegisterRefreshMainPanel | callback function | 更新更改主界面的回调函数 |
| mapToObj | Map | 将Map改为Object |
| objToMap | Object | 将Object改为Map |

注：在该文件中所有与布局有关的函数都会修改。

### 1.3.9 Components

以下是主要由React语言[5]写成的多层级组件嵌套，通过组件的嵌套，构成一个图形化用户界面。

**1.3.9.1 Devices**

该文件夹下存储了所有会用到的器件。通常每一个器件文件夹下，都有三个文件，分别是[Device].css、[Device].js、[Device]Core.js。[Device].css负责css语言的属性，[Device].js负责动态组件逻辑和FPGA的动态通信，[Device]Core.js负责静态组件逻辑。

**1.3.9.2 MainPanel**

该文件夹下是针对软件界面的排版布局而写的三个层次文件，分别是DeviceItem、Gallery、MainPanel。

DeviceItem定义了器件在Gallery中的样式。

Gallery即左侧可弹出的器件管理栏。

MainPanel包括了上述所有组件，并且也是整个软件的界面实现。首先它使用了WidthProvider去实现器件的缩放和移动。然后引入了Devices中的器件。Modal组件时为了器件设置界面的实现。如下图。



**图 35 器件设置菜单**

lodash则是比较方便的JavaScript工具库，方便生成一些典型的数据结构[[1]](#footnote-1)。FPGAManager和ProjectManager的引入代表它与FPGA的交互非常频繁。Gallery器件库也被包括在内。

componentWillReceiveProps是组件的内置函数，与其生命周期有关，简而言之就是组件接收改变过的数据，并将其更新为新数据的样式。

OnSettingClick则是按下设置按钮后，跳出上图设置界面，并将信息更改为按下设置按钮的器件信息。OnSettingModalToggle是按下确定后离开设置界面。OnRemoveClick是按下取消键后将该组件删除的功能。注意到使用了\_.reject是lodash的函数。

OnPortChange的功能是将输入输出端口的改变更新到FPGAManager中。GenPortsSelects是更新输入输出端口的函数。GenerateSettingTable则生成了上图设置界面的实际样式。需要注意的是，以上三个函数均已弃用，实际使用的是同名并加上2的函数。

OnAdd是添加函数后对layout做出改变的函数。这里的逻辑有一些问题，在下一版本也会重写代码。

GenLayoutDevices则是通过\_.map和React.createElement动态生成新添加组件样式的函数。

OnLayoutChange是将layout更新到ProjectManager中的函数。但不包含器件信息，所以实际上是一种比较笨重的实现方式。在下一版本会尝试重写该方式。

在render()函数中，各个回调函数都需要找寻其对应的实现。需要注意的是，在ReactGridLayout中规定了网格化显示的行列是24×24的。

**表格 17 MainPanel的状态说明**

|  |  |  |
| --- | --- | --- |
| 状态名 | 状态类型 | 说明 |
| isSettingModalOpen | bool | 设置界面是否打开 |
| instanceCounter | int | 器件名称的计数器（改） |
| projectConnections | Map | 器件端口映射（弃用） |
| layout | string | 布局信息 |
| selectedDevice | string | 选中器件的名称 |
| selectedDevicePorts | Array | 选中器件的端口 |
| selectedDevicePortsDirection | Array | 选中器件的端口方向 |
| selectedDevicePortsConnection | Array | 选中器件的变量 |

**表格 18 MainPanel的函数功能**

|  |  |  |
| --- | --- | --- |
| 函数名 | 参数类型 | 说明 |
| componentWillReceiveProps | Object | 内置函数，更新属性 |
| OnSettingClick | event | 打开设置界面 |
| OnSettingModalToggle | event | 关闭设置界面 |
| OnRemoveClick | event, string | 删除器件 |
| OnPortChange2 | event, int, string, string, Array | 更新输入输出端口 |
| GenPortsSelects2 | string, int, string, Array | 生成端口选项 |
| GenerateSettingTable2 | string, Array, Array, Array | 生成设置界面 |
| OnAdd | event, string | 添加器件，修改layout |
| OnLayoutChange | string | 将layout同步到ProjectManager |
| render | void | 渲染MainPanel样式 |

**1.3.9.3 title**

该文件是又一重要文件。MainPanel负责软件主界面的样式，则Title负责软件标题的样式。它主要使用了ipcRenderer去将Electron和软件界面联系，实际请求的响应端在Wonton.CrossUI.Web.HostApp的main.ts中使用ipcMain接收。



**图 36 Title设计流程**

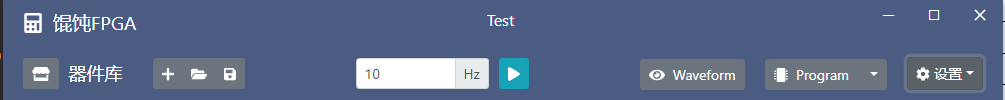
**表格 19 Title的状态说明**

|  |  |  |
| --- | --- | --- |
| 状态名 | 状态类型 | 说明 |
| focus | bool | 焦点 |
| isRunning | bool | 运行中 |
| isProgramToggle | bool | Program菜单栏是否下拉 |
| runHz | int | 运行频率 |
| bitfile | string | 位流文件路径 |
| isFileModalOpen | bool | 项目菜单是否打开 |
| isNewModalOpen | bool | 新建项目菜单是否打开 |
| isOpenModalOpen | bool | 打开项目菜单是否打开 |
| isMaximized | bool | 是否最大化 |
| isSettingDropdownOpen | bool | 设置菜单栏是否下拉 |
| titlePjName | string | 项目名称 |
| newPjName | string | 新建项目名称 |
| openPjName | string | 打开项目名称 |
| isStartModalOpen | bool | 开始菜单是否打开 |
| recentProjects | Array | 历史项目 |
| modified | bool | 是否更改（未保存） |
| iofile | string | 引脚约束文件路径 |
| pjdir | string | 项目文件夹 |
| isDevSwitchChecked | bool | 控制台按钮是否打开 |
| isDarkSwitchChecked | int | 黑暗模式是否打开 |

**表格 20 Title的函数功能**

|  |  |  |
| --- | --- | --- |
| 函数名 | 参数类型 | 说明 |
| constructor | 初始化 | ipcRenderer将界面和软件联系 |
| onWindowStateDarkmode | event, bool | 开关黑暗模式 |
| onExitCallback | void | 关闭窗口（回调） |
| onWindowStateMaximize | void | 最大化窗口 |
| onWindowStateUnMaximize | void | 撤销最大化窗口 |
| onWindowStateBlur | void | 窗口失去焦点 |
| onOpenProjectCallback | event, string | 打开项目（回调） |
| onOpenBitfileCallback | event, string | 打开位流文件（回调） |
| componentWillReceiveProps | Object | 内置函数，更新数据 |
| OpenFileModal | void | 打开位流文件 |
| OnBitfileChange | event | 改变位流文件（弃用） |
| FreqChange | event | 改变工作频率 |
| ClickRun | void | 运行项目（或停止运行） |
| RunFPGA | void | 运行FPGA |
| sleep | time | 休眠 |
| ClickProgram | void | 烧录位流文件 |
| ClickProgrammToggle | void | 打开Program下拉菜单 |
| ClickWaveform | void | 显示波形 |
| ClickClose | void | 关闭软件 |
| ClickMaxRestore | void | 最大化窗口 |
| ClickMin | void | 最小化窗口 |
| NewPjToggle | void | 打开新建项目菜单 |
| NewPj | void | 新建项目 |
| OnNewPjDirChange | event | 改变新建项目文件夹 |
| onOpenNewPjDirCallback | event, string | 改变新建项目文件夹（回调） |
| OnOpenNewPjDir | void | 改变打开项目文件夹 |
| OnNewPjNameChange | event | 改变新建项目名称 |
| OnNewPjIOfileChange | event | 新建项目引脚约束文件 |
| onOpenNewPjIOCallback | event, string | 新建项目引脚约束（回调） |
| OnOpenNewPjIO | void | 打开项目引脚约束文件 |
| OpenPjToggle | void | 打开项目 |
| OpenPj | void | 打开项目（弃用） |
| OnOpenfileChange | event | 打开项目（弃用） |
| Save | void | 保存项目 |
| SettingToggle2 | void | 打开设置下拉菜单 |
| DevToolsToggle | void | 打开控制台 |
| ToggleDevSwitchChecked | bool | 点击打开控制台的按钮 |
| ToggleDarkSwitchChecked | bool | 打开黑暗模式 |
| ClickAbout | void | 点击关于 |
| CloseApp | void | 关闭软件 |
| render | void | 渲染软件标题样式 |

在render()函数中，根据Mac和非Mac操作系统“最小化”、“最大化”、“关闭”三个按钮的位置也会不同。其他部分基本如图所示。



**图 37 导航栏组件**

**1.3.9.4 Start**

该文件主要写了刚进入软件时选择项目的弹窗。其中的回调函数都在Title中实现了。基本样式如图所示。



**图 38 开始菜单**

**1.3.9.5 Layout**

该文件比Title和MainPanel高一层级，负责一些高级属性的存储和更改。可见到的是标题、主界面、是否未保存、是否打开器件库、黑暗模式和所有ProjectManager交互的属性

App和index都是在初创软件时就存在的增加层级的页面，不具备实际功能。可以认为index是该软件的根组件，所以使用的是ReactDOM.render。

需要注意的是，在ClientApp里的README表明，该软件的一些层次结构是由Create React App[[2]](#footnote-2)自动生成的。

## 1.4 Wonton.CrossUI.Web.HostApp

该命名空间适用于启动Electron。最主要的文件是main.ts，使用的是TypeScript语言[[3]](#footnote-3)，但是可以转译成JavaScript语言。

首先导入了Electron的窗口。这里需要注意的是ipcMain是与ipcRenderer对应的响应端，会针对ipcRenderer的各类请求进行不同的响应，可以与Title.js对照参考。后来导入了端口和子进程库，并使用了ASP .NET Core[[4]](#footnote-4)。大部分逻辑都不需要修改，需要注意的是Debug模式下使用了55405端口，而Release情况下则是用portscanner寻找一个空闲端口使用。

## 1.5 Wonton.Test

该命名空间下使用了一个示例去检测Wonton是否可以正常使用。使用Visual Studio去调试代码时，可以点击测试菜单栏中的“运行所有测试”去使用该测试用例[[5]](#footnote-5)。

第一个测试是使用FPGABoard与FPGA做数据交互并输出。

第二个测试是读取引脚约束文件并转换为json格式。

第三个测试是寻找指定node\_modules中文件是否支持Electron。

第四个测试是更改已生成的安装包的名字。

第五个测试是输出所在Mac OS X系统的版本号。

# 2 杂项

## 2.1 安装脚本

Windows系统使用build.cmd或者build.ps1，Linux\Mac OS X使用build.bash。其中build.cmd其实也是调用了build.ps1，所以不予说明。

在build.ps1中，先设定了一些了变量，然后规定了运行程序和运行路径，接着安装.NET Core、NodeJs和Cake[4]，最后调用Cake运行build.cake。

build.bash也是类似的功能，但是使用了bash的语法。

在build.cake中，使用了Cake的专用语言[[6]](#footnote-6)，首先定义了一些全局变量。Setup函数定义了操作系统和架构，并输出必要的系统信息。BuildNative在必要时编译SharpVLFD驱动。InstallClientApp对Wonton.CrossUI.Web的npm进行安装。CopyPackage则是将已经编译好的安装包拷贝至发布路径。Clean任务在编译过程中没有用到。InstallHostApp则对Wonton.CrossUI.Web.HostApp的npm进行安装。BuildHostApp自动运行了在该文件夹内的build脚本。PublishBackendApp是用.NET对该软件进行发布。CopyHostApp是将所有HostApp文件夹内的有用文件拷贝至发布路径，InstallPrepackageHostApp则是将发布路径的npm进行安装。PackageDependency将GTKWave的可执行文件打包压缩在Dependency文件夹中，Windows和Mac系统都直接将可执行文件解压后使用。PackageApp则是将上述函数整合在一起，将参数也整合在一起，使用electron builder打包发布为可用的安装包。

electron builder的自定义设置都在Wonton.CrossUI.Web.HostApp文件夹中的package.json文件中[[7]](#footnote-7)。其中比较重要的是scripts、build和各种类型安装包的说明。scripts规定了三种启动方式，分别是”prestart”、”start”和”build”，前两者用于调试，最后一个用于编译。build则规定了编译安装包的基本信息、输出目录和编译依赖，也包括了在不同操作系统下的具体配置。

## 2.2 调试运行

当在Windows系统下调试运行Wonton软件时，Visual Studio选择运行Wonton.CrossUI.Web，使用cmd在Wonton.CrossUI.Web.HostApp文件夹下运行npm run start即可调试运行Wonton。前端JavaScript代码在更改后会实时更新，后端C#代码在更改后需要重新运行才会更新功能。

#### NuGet

Wonton所用的SharpVLFD在不特殊指定的情况下默认使用NuGet安装已发布的版本[[8]](#footnote-8)。

1. https://www.lodashjs.com/ [↑](#footnote-ref-1)
2. https://github.com/facebookincubator/create-react-app [↑](#footnote-ref-2)
3. https://www.tslang.cn/ [↑](#footnote-ref-3)
4. https://docs.microsoft.com/zh-CN/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-3.1 [↑](#footnote-ref-4)
5. https://docs.microsoft.com/zh-cn/visualstudio/tesesting?view=vs-2019&tabs=mstest [↑](#footnote-ref-5)
6. https://www.cakebuild.net/dsl/ [↑](#footnote-ref-6)
7. https://www.electron.build/ [↑](#footnote-ref-7)
8. https://www.nuget.org/ [↑](#footnote-ref-8)