

20203.29

EVE IDE 手册

V0.0.1(预览版)

By Adancurusul

chen.yuheng@nexuslink.cn

目录

一、 前言：	3
二、 背景：	3
三、 主界面图形设计介绍：	4
四、 主界面实现：	7
五、 周边软件组件：	10
1. 自动 makefile 生成工具：	10
2. Adan_rtos 嵌入式实时操作系统	12
3. 基于 basic 语法的高移植性科学计算语言	14
4. 串口监视器	15
Prv332_ide	16
附录 1： PRV332_IDE 操作手册	16
附录二： PRV464 处理器编程手册（部分）	25

一、前言：

本 IDE 的图形界面总体由 pyqt5 设计完成，有很高的移植性，同时为了使运行不卡顿，对部分逻辑用 lua 进行加速，利用的 python 的 lupa 库直接调用 lua 源码解释执行，而且直接执行 lua 字节码产生的多线程打破了 python 的 GIL 锁（仅对直接运行 python 字节码有效），对 cpu 利用增大，周边软件生态组件大量使用 c 和汇编，更加接近底层。

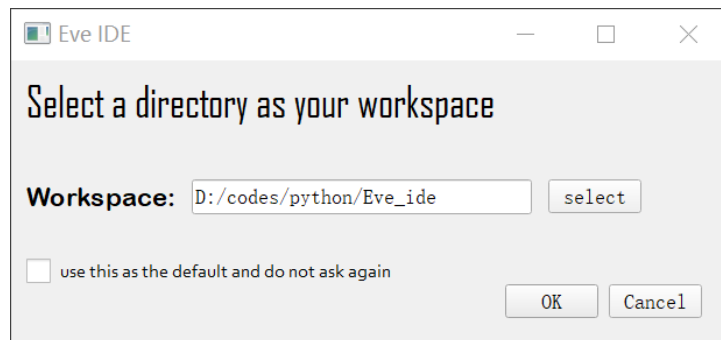
这是个非常轻量的 ide 整个 ide 仅由 4000 行代码十来个文件组成，最后 pyinstaller 打包环境直接变成可以直接使用的文件。周边软件的实现也是以精简和实用性为目标，高移植性为原则设计。尽量做到让硬件开发者解包即用，不再纠结于工具链之类的配置。同时让软件开发者能以最简单的方式移植到自己需要的地方。

二、背景：

目前利用开源架构 riscv 开发的 cpu 越来越多，riscv 作为一个优秀的架构在 aiot 领域也越来越火，但是国内目前很少有为 riscv 定制的开发工具。同时队友设计的 riscv 处理器已经迭代几次，为 riscv 生态做出很多贡献，且有很大的前景。但是软件方面支持较少，故决定着手开始为 riscv 软件方面生态增加一点自己的小玩具，eve_ide 及其周边生态组件的想法就这样诞生了。

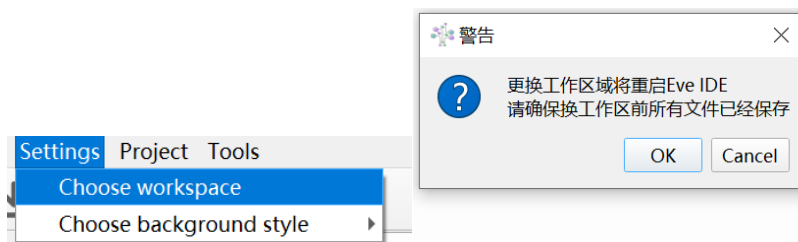
Eve_ide 的前身是 prv332ide，这是一个为 riscv 处理器开发者设计的软件，主要是重构了 riscv32 的汇编器，具体详见后文周边组件 prv332ide 以及附录 1：prv332ide 使用手册。

三、主界面图形设计介绍:

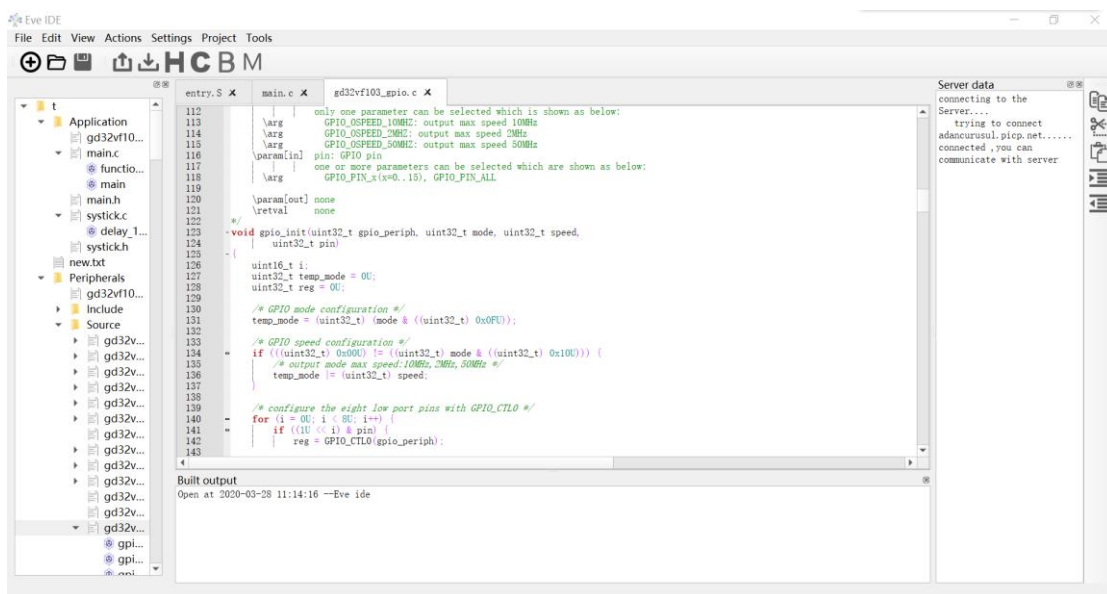


打开会出现个让你选择工作区的界面，点击选择然后 ok 即可，如果不想每次打开都弹出来，点击下面那个框就可以了

当使用过程想换工程地址。最上面有个 settings，点击更换即可。



重新选择工程地址



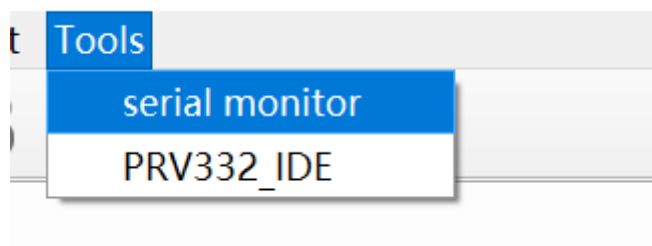
主图形界面

主界面布局参考大量 ide 设计而来，且不同使用习惯可以自行定义控件位置

最上面一排作为提供给用户的所有操作的菜单，后面将一个一个介绍，下面一排是常用操作



这些功能详细应用后文以及附录 1 中也会介绍到。

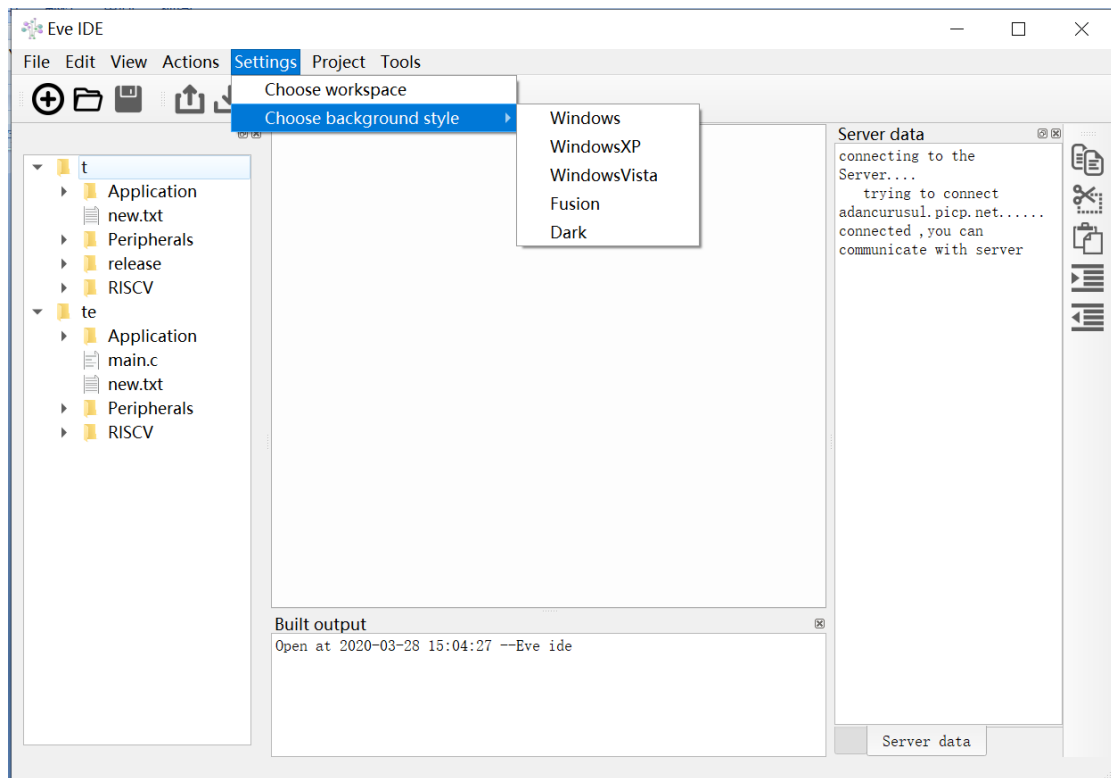


Tools 下面提供了两个组件，一个串口监视器和 prv332_ide 这些在后面组件一章会有讲到

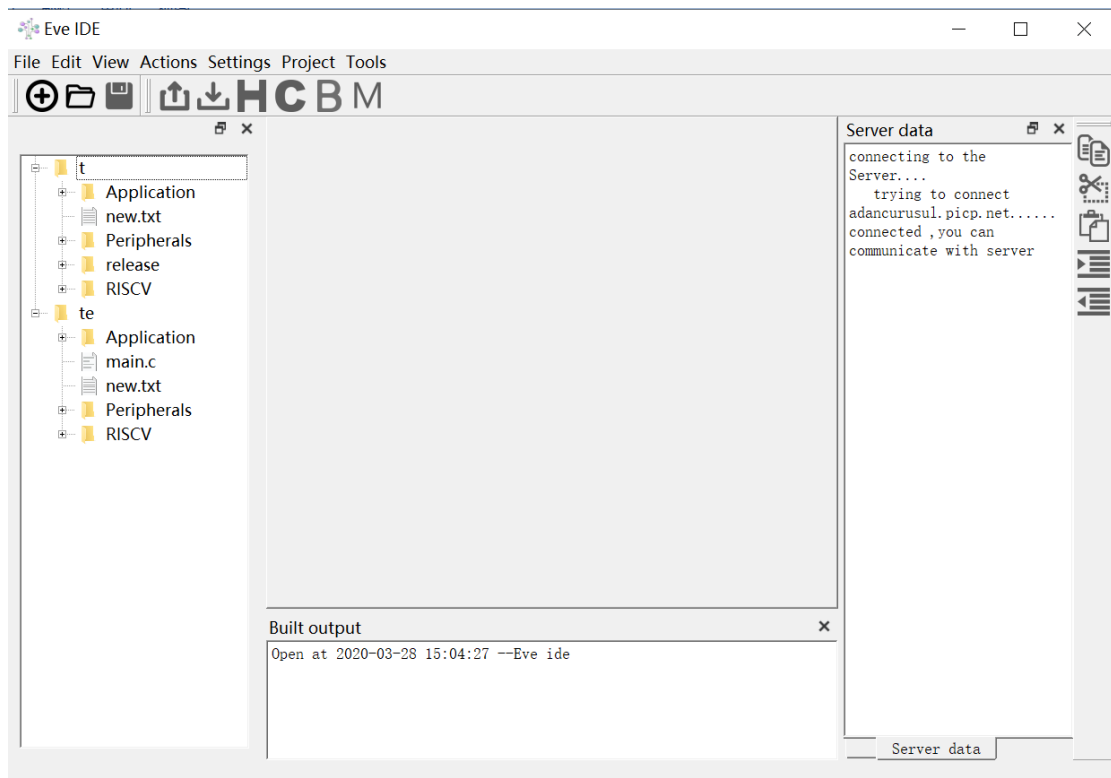
左侧是工程文件的树状视图，方便开发者寻找需要的文件，文件夹以及函数，中间下方是编译输出，所有编译都将定向到这里输出。右侧是连接服务器以及后续仿真操作等。正中间是文本编辑部分，具有代码高亮、自动缩进等功能，还能实现改动标记，远程跳转等一系列操作，这些马上在主界面实现中将详细介绍。

IDE 为用户提供了几种主题风格可以选择，**darkstyle** 适合较黑的环境使用。

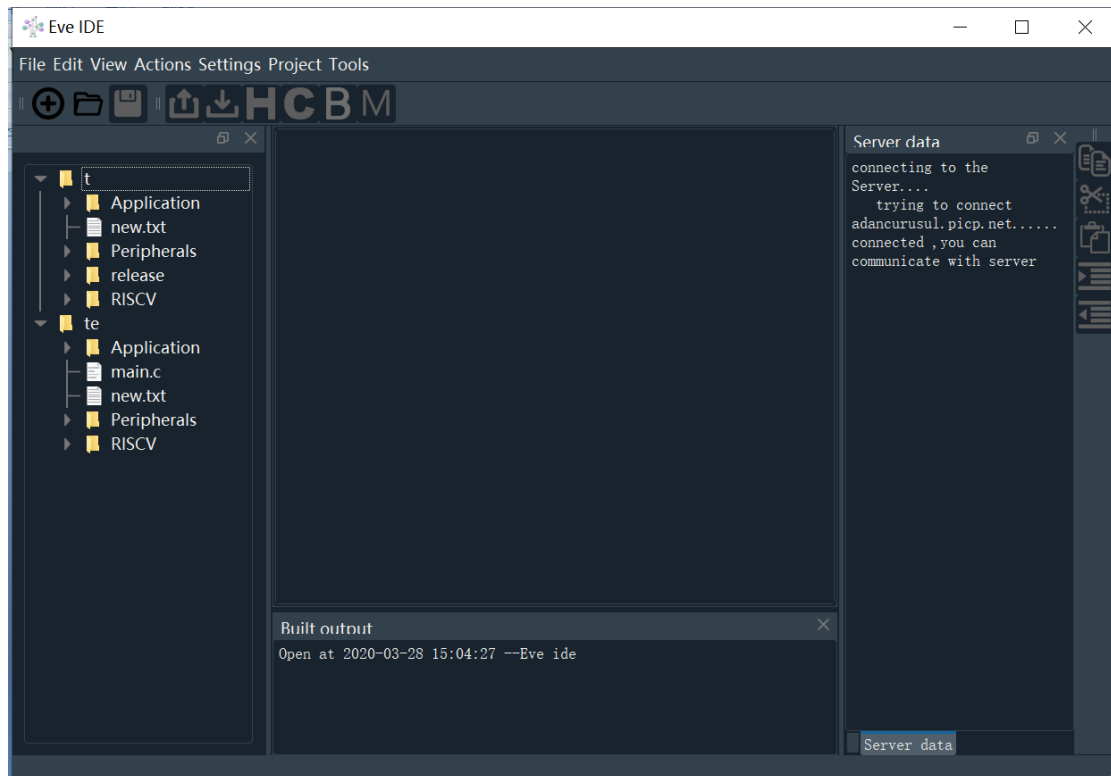
Eve ide



Fusion



Windows



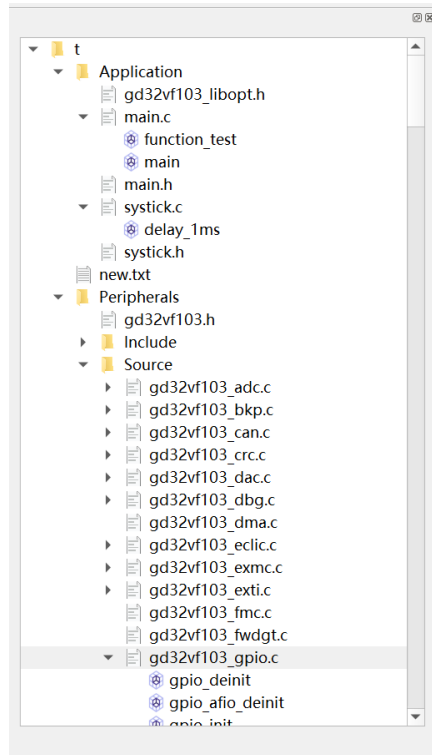
dark

四、主界面实现：

如果要找一个工程最重要的部分，查找每个板块的使用次数是个好方法。

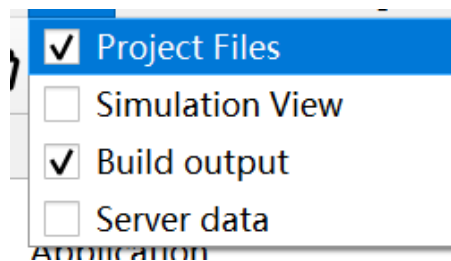
对于 EVE_IDE 展示出来最重要的就是中间的文本编辑区，而内核将这一切捆绑到一起的呢，是其中的文件-函数管理机制。

文件-函数管理系统是由哈希表形式实现的，依次是 工程文件夹->文件->函数名->函数名所在行数，每一次文件读写操作都会重置一次管理系统以实时展示目前工程情况，用户可以通过左侧树状结构直观的看到现在的树状结构



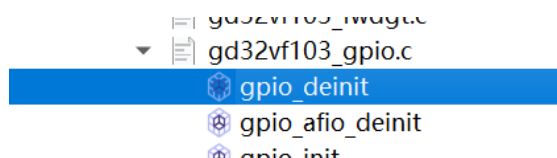
工程的树状结构

当然，你看他不爽也可以直接点叉叉关掉，如果又想念它了就去上面 view 找到他，当然这个 view 还管理了其他几个组件的打开与关闭



打开的位置

这个树状图实现前后牵扯了十多个函数，横跨三个自定义类，撑起了整个 ide 的文件管理。右键树结构可以手动刷新树结构以及后台的文件管理。双击树中的目录可以折叠或者打开。如果双击的是文件名那么就会在主编辑区打开该文件，无法打开则报错。如果双击的是函数名称则会打开函数所在文件并将光标指向该行




```
*/
void gpio_deinit(uint32_t gpio_periph)
{
```

直接打开对应函数

```
142 lineOfLeft = self.current_line
143 while (True):
144     # 查找'{' --> {
145     current_char = self.getchar()
146     if current_char == '{': # 后面可能有注释 /**/ 或者 // 跳过 ;还有=跳过
147         while (True):
148             current_char = self.getchar()
149             if current_char == '{': # 当前行中包含函数名, 记录行号和获取函数名
150                 str1 = self.getFunctionNameInLine(self.input_str[lineOfLeft])
151                 if str1:
152                     g_allFuncList.append(str1)
153                     return (str1, lineOfLeft)
154                 return None
155             elif current_char == '{':
156                 lineOfLeft = self.current_line
157                 continue
158             elif current_char == ';' or current_char == '=': # 分号表示此处为函数调用, 并非函数体跳过 --可能是函数指针数组
159                 self.current_line += 1
160                 self.current_row += 1
161                 return None
162             elif current_char == '/':
163                 next_char = self.getchar()
164                 if next_char == '/': # 单行注释跳过当前行, 下面已经是下一行
165                     self.current_line += 1
166                     self.current_row += 1
167                 next_char = self.getchar() # 换行的第一个是 { 认为是函数所在行
```

寻找函数是写 ide 遇到的一个小的难点, 我这里直接实现一个类似词法分析器的函数找到每个对应的函数以及行号, 并存入哈希表中, 索引为 function->line 然后整个函数返回当前的文件名以及函数哈希表。

然后来说说主要的中心编辑区, 这个由于人力能力限制做的很简陋, 但是也基本是个编辑器了吧。

首先是支持 c 语言的语法高亮以及自动缩进。

```
main.c x main.c x
1 //Created at 2020-02-12 16:16:57
2 //Eve ide for gd32vf103
3
4 #include<stdio.h>
5 #include "gd32vf103_libopt.h"
6
7 void function_test(){
8     printf("hello");
9 }
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
int main()
{
    int a = 0;
    while (a:10)
        gpio_init(GPIOA, GPIO_MODE_OUT_PP, GPIO_SPEED_50MHZ, GPIO_PIN_7);
        a++;
    function_test;
```

c 配色

然后会在修改的行号旁边给个小标记如下图, 这个标记会在保存后自动消失。



改动标记

当然，有的文件较大，EVE_IDE 支持折叠函数和循环。同时还有跳转功能当鼠标对应的在#include xx 或者是有个函数时，就类似 ide 的键跳转功能。

Eveide 提供了一个服务器端连接功能，最初设想是可以做到云端编译功能，但是目前很多情况需要脱离互联网使用，这样很鸡肋，故目前的作用就是有关 Adan_rtos 编译（见下文）

五、周边软件组件：

1. 自动 makefile 生成工具：

Makefile 自动生成工具目的是将 c 文件输出为 hex 文件（改变为 coe, mif 以及二进制不在本部分考虑之中）

因为 eveide 对 c 语言处理是利用自己编译的开源 gcc 完成的，为方便开发，开发工具有一套自己的 makefile 生成逻辑供开发者使用。eve_ide 针对 riscv32 架构单片机的 c 文件和头文件处理思路是：

首先拿到工程目录下文件的哈希表然后再遍历该目录并与原哈希表对比（这一步防止其它对文件的改动）。然后根据文件类型在非全部是头文件的目录下生成一个 sub.mk 作为该目录的 makefile 文件并写入相应编译规则，并找到依赖的头文件输入到.d 文件作为依赖项，这里放出一个 sub.mk 的截图。

```
C:\EVE> cat
./Peripherals/Source/gd32vf103_adc.d \
./Peripherals/Source/gd32vf103_bkp.d \
./Peripherals/Source/gd32vf103_can.d \
./Peripherals/Source/gd32vf103_crc.d \
./Peripherals/Source/gd32vf103_dac.d \
./Peripherals/Source/gd32vf103_dbg.d \
./Peripherals/Source/gd32vf103_dma.d \
./Peripherals/Source/gd32vf103_ecllc.d \
./Peripherals/Source/gd32vf103_exmc.d \
./Peripherals/Source/gd32vf103_exti.d \
./Peripherals/Source/gd32vf103_fsmc.d \
./Peripherals/Source/gd32vf103_fwdgt.d \
./Peripherals/Source/gd32vf103_gpio.d \
./Peripherals/Source/gd32vf103_i2c.d \
./Peripherals/Source/gd32vf103_pmu.d \
./Peripherals/Source/gd32vf103_rcu.d \
./Peripherals/Source/gd32vf103_rtc.d \
./Peripherals/Source/gd32vf103_spi.d \
./Peripherals/Source/gd32vf103_timer.d \
./Peripherals/Source/gd32vf103_usart.d \
./Peripherals/Source/gd32vf103_wdgt.d

Peripherals/Source/%.c: ../Peripherals/Source/%.c
riscv-nuclei-elf-gcc -march=rv32i -mabi=ilp32 -mcmodel=medlow -msmall-data-limit=8 -O2 -fmessage-length=0 -fsigned-char -ffunction-sections -fcommon -c $< -o $@
```

sub.mk

最后在 release 目录下生成总的 makefile 文件，当然 include 进所有 sub.mk。

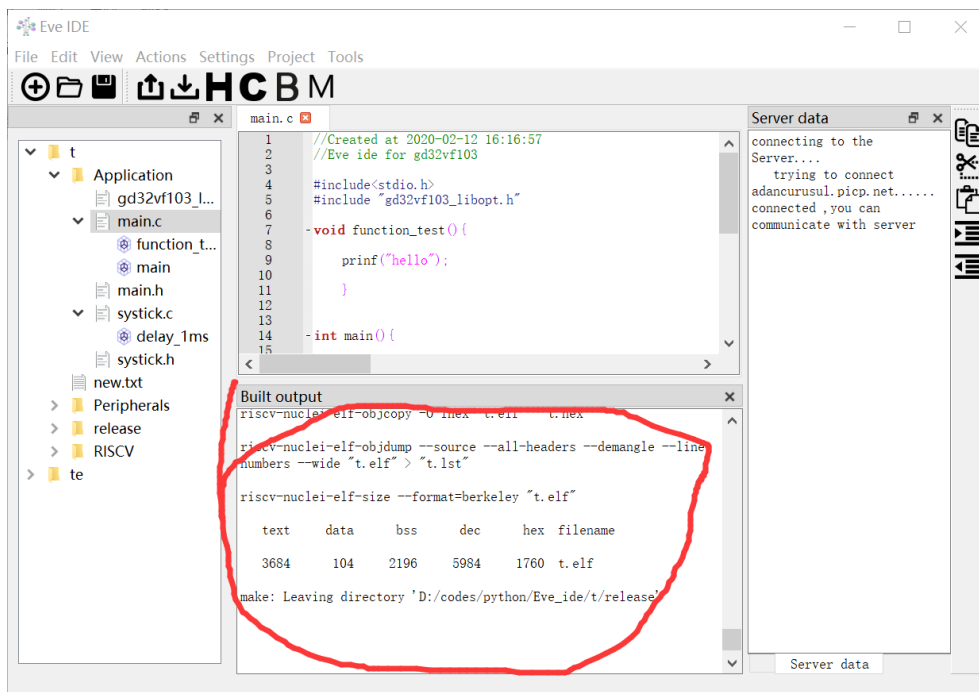
```

50 SECONDARY_SIZE += \
51 t.size \
52
53 all: t.elf secondary-outputs
54 t.elf: $(OBJS) $(USER_OBJS)
55     @echo 'Building target: $@'
56     @echo 'Invoking: GNU RISC-V Cross C++ Linker'
57     riscv-nuclei-elf-g++ -march=rv32i -mabi=ilp32 -mmodel=medlow -msmall-data-limit=8 -Os -fmessage-length=0 -fsigned-char -ffunction-sections -fcommon -o t.elf *.o
58     @echo 'Finished building target: $@'
59     @echo ' '
60 t.bin: t.elf
61     riscv-nuclei-elf-objcopy -O binary "t.elf" "t.bin"
62
63 t.hex: t.elf
64     riscv-nuclei-elf-objcopy -O ihex "t.elf" "t.hex"
65
66 t.lst: t.elf
67     riscv-nuclei-elf-objdump --source --all-headers --demangle --line-numbers --wide "t.elf" > "t.lst"
68
69 t.size: t.elf
70     riscv-nuclei-elf-size --format=berkeley "t.elf"
71
72 clean:
73     -$(RM) $(CC_DEPS) $(CXX_DEPS) $(OBJS) $(C_UPPER_DEPS) $(CXX_DEPS) $(SECONDARY_FLASH) $(SECONDARY_LIST) $(SECONDARY_SIZE) $(ASM_DEPS) $(S_UPPER_DEPS)
74
75 secondary-outputs: $(SECONDARY_FLASH) $(SECONDARY_LIST) $(SECONDARY_SIZE)
76
77 .PHONY: all clean dependencies
78
79 -include ../makefile.targets

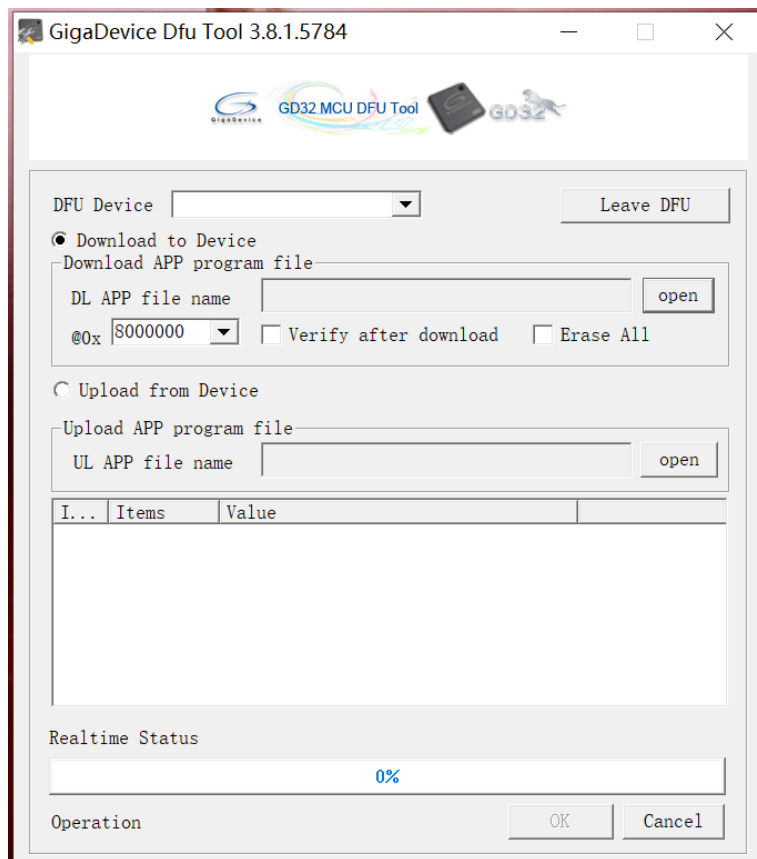
```

主 makefile 文件

为了方便查看结果 makefile 中还将生成的 hex 文件变成 list 和 size 让用户看到编译后结果（如下图红圈圈出部分），将终端输出重定向到 ide 的 build output 组件就可以直观看到结果。之后利用 gd32vf103 dfutool 就可以烧录。



生成文件详情



Dfu tool

生成之后刷新工作区旁边文件树，makefile 逻辑就走完了，如果用户选择的生成 coe 或者 mif，普通二进制，下载到板子里又会进入主界面逻辑中对应板块（上文有详细讲）。

自动 makefile 生成能让用户从繁琐的 makefile 依赖关系中脱离出来专心于驱动编写以及处理器验证

2. Adan_rtos 嵌入式实时操作系统

Adanrtos 是本团队独立开发的用于 prv464 处理器核心的一款嵌入式实时操作系统，但是由于处理器核心仍在验证阶段，所有全是基于仿真，这里就简略介绍一下。

设计目标是和 eveide 的云端服务器交互，用户可以写好自己需要的处理函数组件然后利用 eveide 云端功能将其嵌入系统中回传，这里主要讲一下自主研发实时性较高的调度算法，基于多级反馈算法的实时调度算法 – 双抢占式多级反馈算法。

本调度综合 FCFS（先到先服务）、RR（轮转法）、HPF（高权高优先）以及实时调度的抢占式算法实现。

首先按照优先级设置 n 个就绪队列，并赋予不同优先级，同时指定第一个队列为最高抢占式优先级，这个优先级中只要存在就绪且发出实时信号（自定义用来判断进程类型的信号）便待下当前时间片转完直接跳入执行，同时次任务不受时间片轮转影响，即运行结束为止。但是希望用户慎重使用，因为可能导致其他进程无法完整执行，推荐在高实时性环境使用。

对于其他就绪队列采用时间片递增的方式：例如第三个队列时间片比第二个队列时间片多一倍， $i+1$ 队列时间片比 i 队列时间片长一倍。

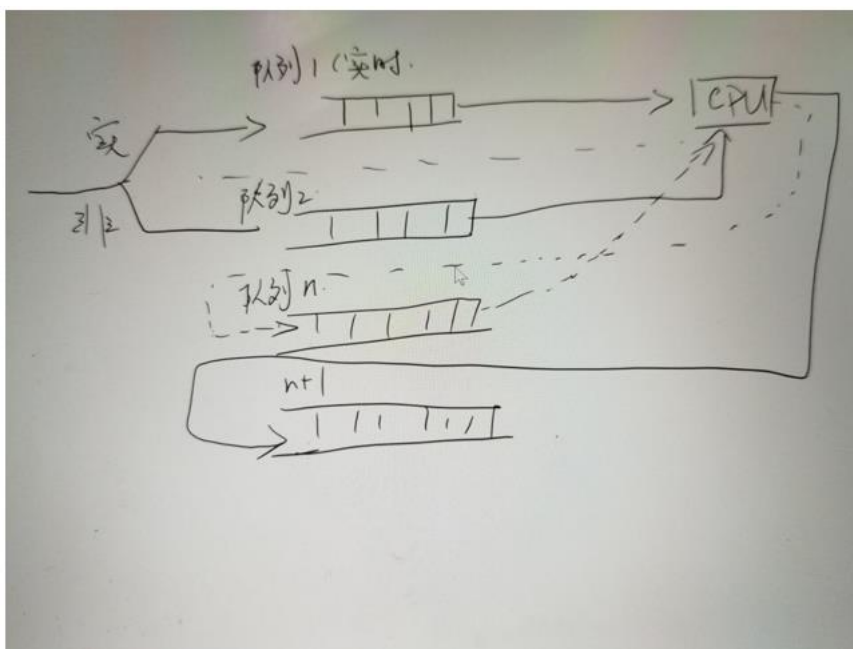
当一个新进程进入内存后，判断是否为抢占类型进程，是则丢入抢占队列，否则放入第二队列的末尾，按照 FCFS 原则派对等待调度。当轮到该进程时，如果能在该时间片完成则撤离系统，在时间片结束尚未完成或者被阻塞，调度程序则将该进程转入第三队列末尾，如果第二队列中运行仍然未完成，则继续丢到第四队列。

同时调度初期有个定义：仅当长进程从第二个队列降到更下面的队列时候，该队列使用 RR 调度来运转，其他时候队列内部仍然使用 FCFS 调度。这样尽量保证了进程的高完成度防止时钟周期过多浪费在调度中。

如果处理机正在第 n 个队列中，而此时又有新进程进入优先权更高的队列（如 $n-2$ ），那么新进程将会抢占现在进程的处理机，将现有程序放到第 n 队列末尾。

这个算法对长进程和高实时性进程有好处：例如一个进程需要 100 个时间片，如果采用 RR 算法，则需要被切换 100 次，大量始终周期浪费在切换中，而如果用我们的算法 第一次一个，第二次 2，第三次 4，第四次 8，。。。。这样切换 7 次就可以完成，大大提高 cpu 使用频率，同时随着运行优先级不断降低，运行频度放慢，给其他进程让出 cpu。而如果是需要实时性的进程，为保证实时性他的优先级式用户优先级中最高，这样就有充足时间留给该进程，这在嵌入式领域将会有很大的空间。

由于没有硬件来展示我们的调度算法实现，就给出设计时的实现图。



3. 基于 basic 语法的高移植性科学计算语言

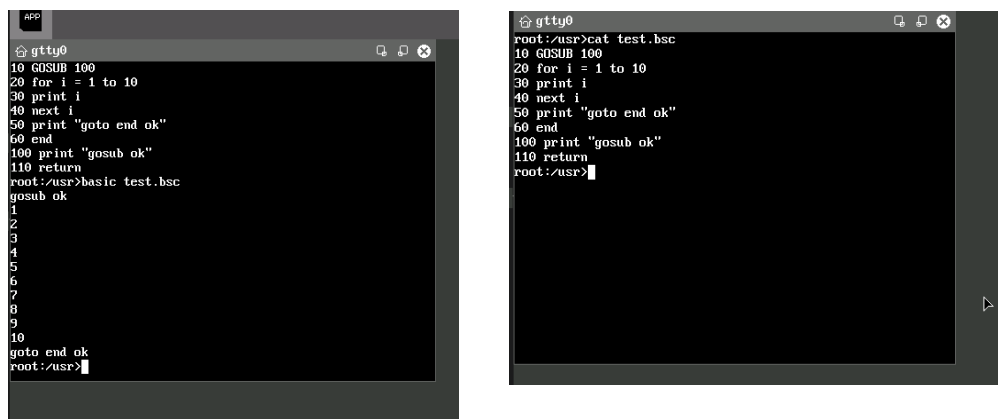
这是一个极高移植性的语言解释器，理论上任何支持 c 编译器且具有 1k 字节内存的均可快速移植，非常方便调用以及开发。代码量仅千行，但已经支持 basic 中 let, print, if, then, else, for, to, next, goto, gosub, return, call, end，peek, poke 等语句。可以直接接触底层硬件接口，加之语法简便，可以大大方便开发人员使用，并已支持四则整型运算。

使用时只用传入程序的字符串数组到 interpreter_init，然后 do_interpretation 即可，interpreter_finished 作为解释结束标志。

现已开源到 github：https://github.com/Adancurusul/some_tiny_interpreters

原先使用 rust+c 语言开发，但后考虑到移植的复杂度决定放弃 rust 改用纯 c 语言开发，在其中实现了 basic 的词法分析器以及基于前缀树的变量管理系统，在 x86 平台（windows, linux, mac; intel quark d2000）；arm 平台（armlinux 如树莓派, beagleboneblack 等, arm 裸机如 stm32 等）；riscv 平台（k210, gd32vf103 等）；Mips 平台（onion omega 等）；Xtensa 平台（esp8266, esp32）；已经实现移植，移植性超高。

值得一提，该解释器已被 bookos 项目组收录作为 bookos 操作系统支持语言之一。Bookos 上的运行截图如下：



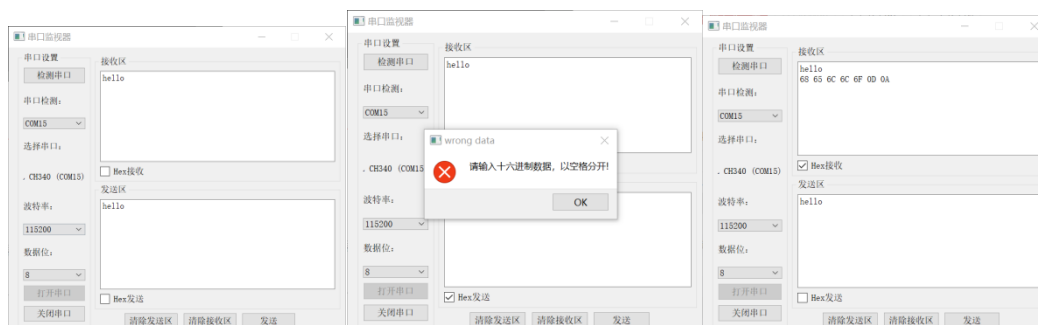
```
gtty0
10 GOSUB 100
20 for i = 1 to 10
30 print i
40 next i
50 print "goto end ok"
60 end
100 print "gosub ok"
110 return
root:/usr>basic test.bsc
gosub ok
1
2
3
4
5
6
7
8
9
10
goto end ok
root:/usr>
```

```
gtty0
root:/usr>cat test.bsc
10 GOSUB 100
20 for i = 1 to 10
30 print i
40 next i
50 print "goto end ok"
60 end
100 print "gosub ok"
110 return
root:/usr>
```

bookos 上运行截图

4. 串口监视器

一个面向嵌入式开发的 ide 怎么能少了串口监视器，相对于 putty 那种串口助手，eve_ide 的串口助手支持自动寻找可用串口；波特率选择；以及发送、接收格式选择。同时发送 16 进制时能自动检测十分格式正确以避免出错。界面利用 qt 制作并嵌入主程序串口，可以随时快速加载打开，且和主程序分处独立线程，互不干扰。实现非常简单，直接调用操作系统给出的接口即可。使用也非常简单，检测串口选择波特率和数据位数就可以直接发送和接收。下图中可以看到利用 ch340 做的收发测试。



串口监视器截图

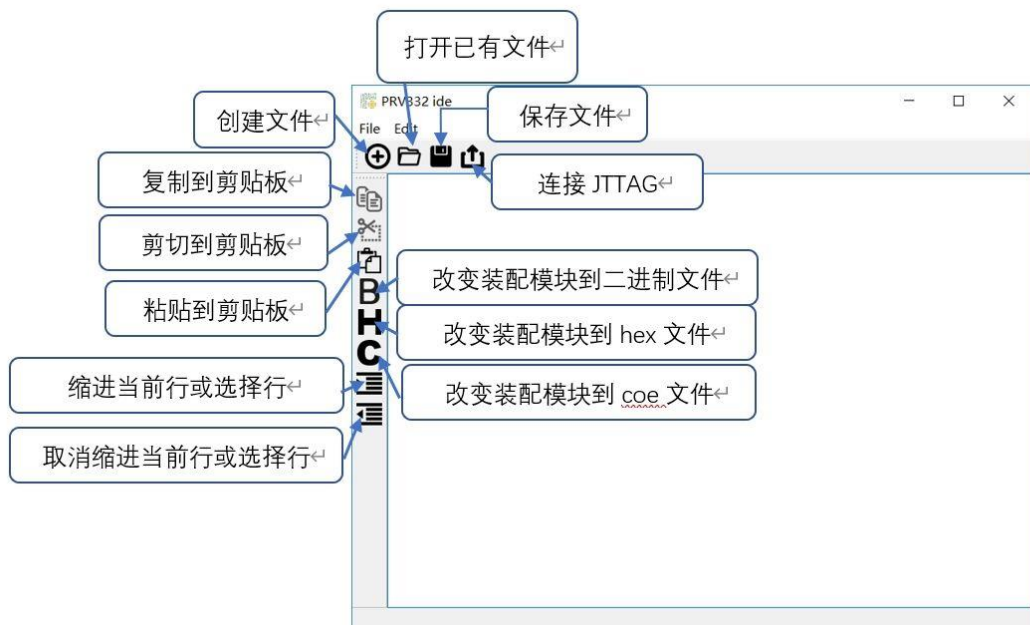
Prv332_ide

前文说了，eve_ide 的前身是 prv332_ide，很多功能也是延续了 332ide 的。Prv332_ide 核心是重构的 riscv32 汇编器且已经在社区内部开源，作为很多自制 rv 处理器验证的软件。具体操作详见附录 1（prv332_ide 操作手册）。

附录 1：PRV332_IDE 操作手册

利用 websocket 与板载烧录器无线连接，可以方便进行远程烧录和调试。

最新版 ide 还移植 gcc 工具链，并做一定修改：由 gcc 生成 s 文件（汇编文件）然后经过自主研发的汇编器生成相应代码。此 ide 能检测到所用的语言并自动高亮、翻译，同时有一定的报错机制。



主要功能如上图下面将详细说明

1) 远程烧录器的连接



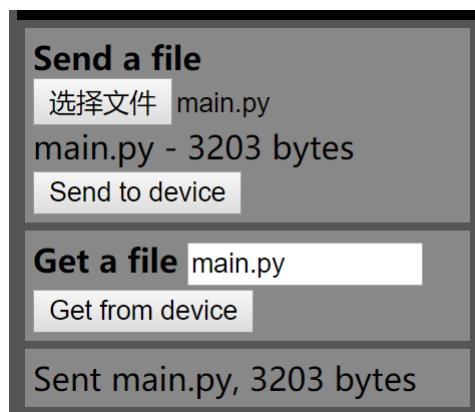
普通的文件操作这里不多赘述，连接 jtag 有两种方式：

①利用 jtag 中的 wifi 模块作为服务器，电脑端连接服务器，这种方式比较推荐使用同时是一个网页，需要访问 <http://micropython.org/webrepl/>



配好左上角的 ip 地址点击 connect，jtag 会提示输入密码，默认密码为 3（输入密码时不会有字符显示），然后就可以访问到 jtag。

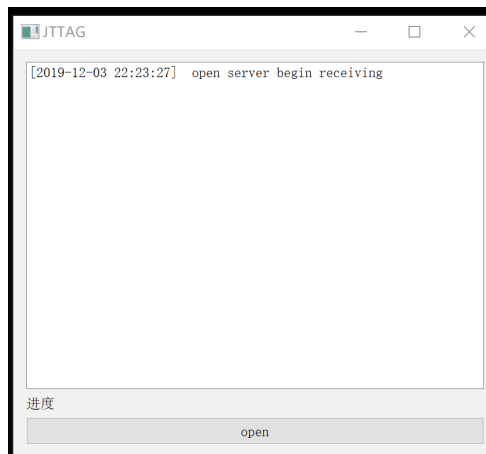
同时左下角可以向 jtag 发送，接收文件用于烧录并在线更新固件。



中间就是 jttag 发送来的数据, 可以看到输入字符后可以远程重启系统方便调试。

```
>>> PVS332
HELLO WORLD!
CORE PRV332
OCRAM TEST...
OCRAM OK -4KB
press any key to reset: 
```

②直接点击 connect to jttag 开启电脑端的网络服务器

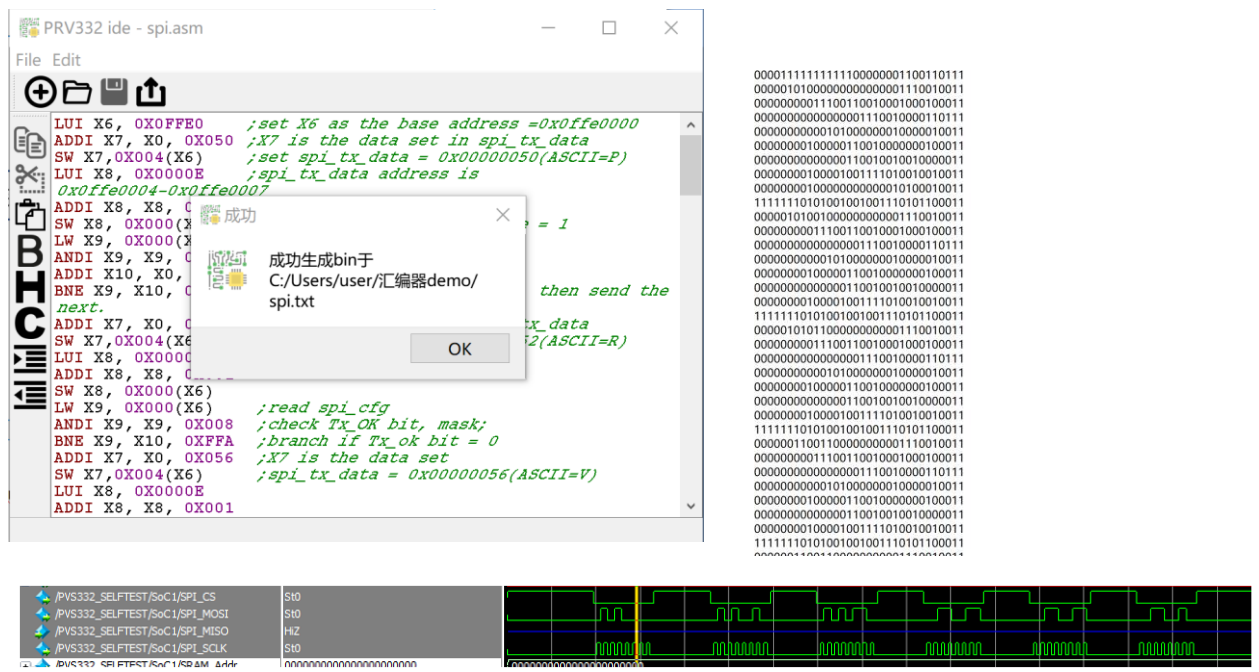


Jttag 会自动连接网络服务器并进行数据传输。此功能方便了开发者和学习者对 soc 的远程控制和监视, 且操作简单, 在最后实现中是以和 ide 不同的线程执行, 可以在 ide 崩溃时仍然能对 soc 进行监视和对一定行为的控制。

2) 普通二进制文件的生成

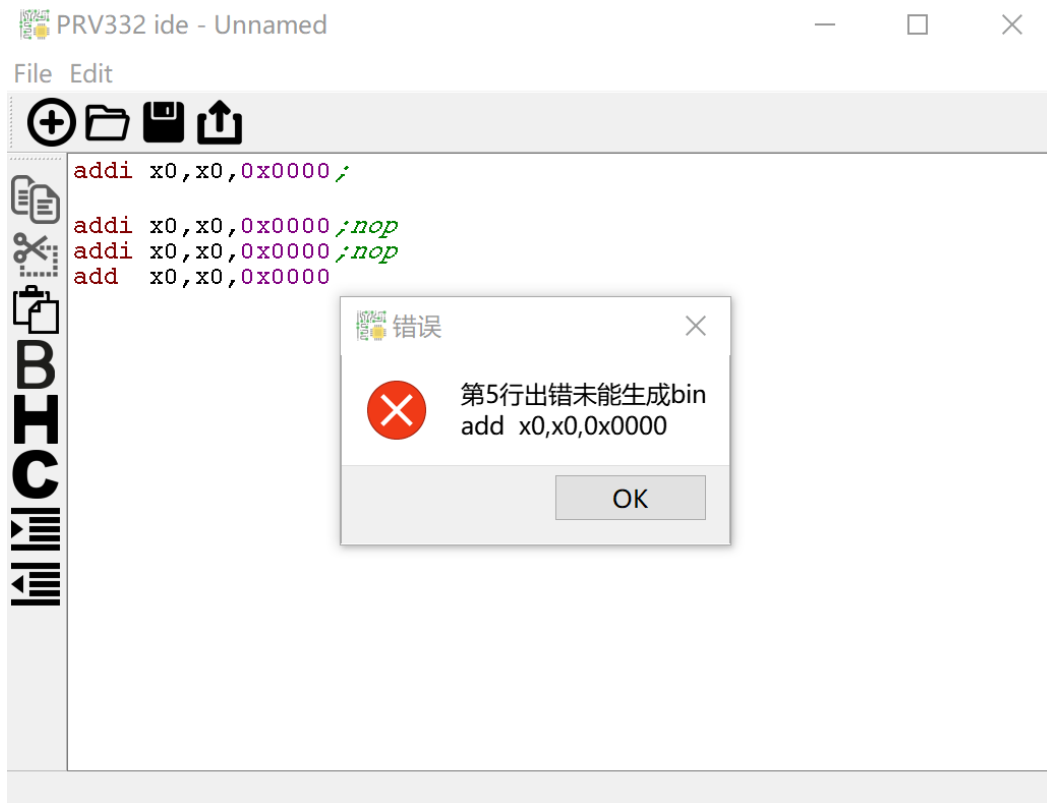


点击 B 按钮或者 edit 中的 change into binary 或者使用 ctrl+] 可以将汇编代码变为普通 2 进制文件可以直接用于 modelsim 仿真，生成成功后会显示生成地址方便后面寻找



spi 驱动代码生成的二进制文件 modelsim 仿真波形

同时 ide 有着较为完善的报错机制，可以显示错误以及行数，可以看到错误的汇编代码会直接提示到错误的行数以及无法翻译的语句，这大大方便了调试和修改，普通二进制文件在 soc 研发和开发过程中非常重要，且市面上部分开发工具不支持直接生成 2 进制文件，prv332ide 对此进行学习和教育方面优化。



3) COE 文件的生成



Coe 文件可以用于 xilinx 的 brom 的烧写,ide 中可以通过点击 c 按钮或者 edit 中的 change into coe 从而生成 coe 文件,并且可以自由选择保存位置和格式。

以下: 一个完整的 COE 文件——

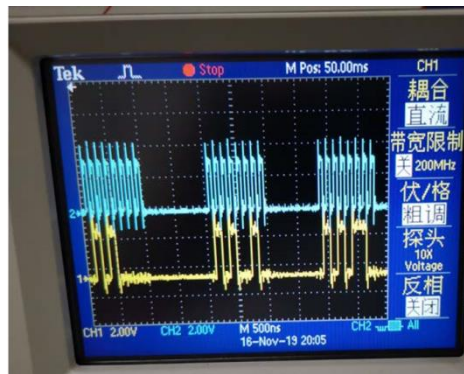
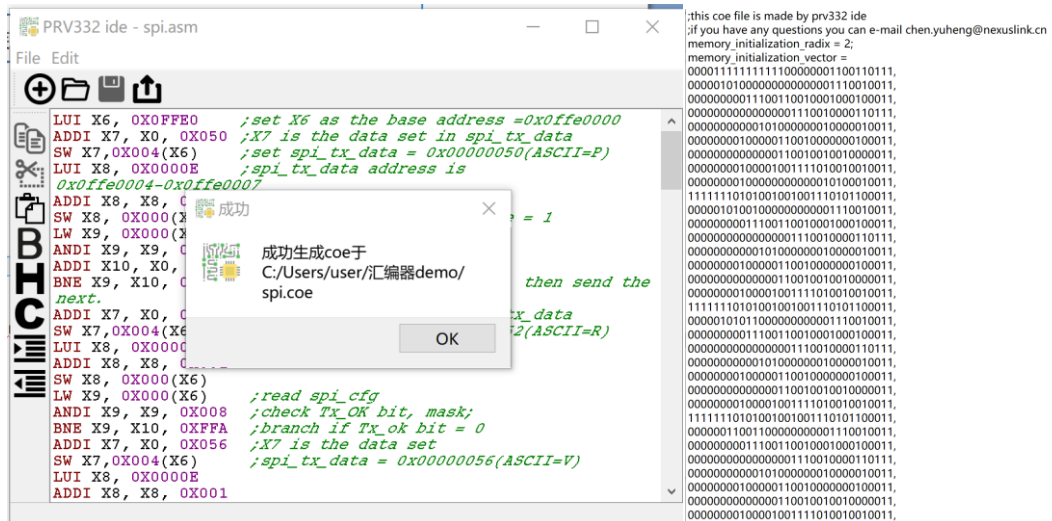
第一行: `MEMORY_INITIALIZATION_RADIX=2;`

表示的是几进制数据

第二行: `MEMORY_INITIALIZATION_VECTOR=`

表示你要初始化的内容,然后把初始化的数据从第三行开始写,注意每一个数据后面都要有一个逗号。最后一个数据是分号。然后保存重新命名为 xxx.coe 即可。

利用 python 的便捷性调用自研汇编器接口可以直接快速生成 spi 驱动对应的 coe 代码,可以直接保存到目录下,同时可以在示波器看到相应波形。



Coe 文件可以直接用于 xilinx 的 brom 的烧写，市面上支持该功能的 ide 几乎没有，这里我们为了方便教学与开发工作特地加入此功能。

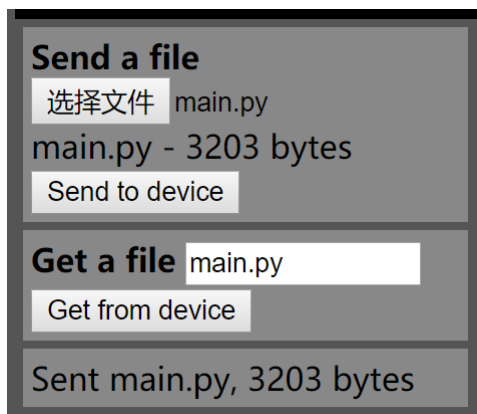
5)、HEX 文件的生成



要生成 hex 文件我们可以直接点击 H 符号，或者打开 edit 选择 change into hex 该操作快捷键为 CTRL+[。

最近我们移植了 riscv 官方的 gcc 但是做了一点改动：将生成的 s（汇编）文件直接配合我们开发的汇编器可以直接生成 prv332 可以使用的 hex 文件储存，同时我们也支持汇编语言的转化，ide 内部会自动识别是汇编还是 c 语言从而判断是否调用 gcc 工具链。

生成的 Hex 文件配合 prv332 板上的 bootloader 可以直接运行，所以在设计 ide 时考虑到这个可以让网络调试端直接发送 hex 文件到烧录器，烧录器开机发现 hex 文件后便会利用 spi 将 hex 放入 flash 中，等待 prv332 端的 bootloader 来取用到 ram 并执行。



由于 hex 生成会伴随其他文件生成，可能对开发和验证有用故在此我们直接给出生成的地址。

PRV332 ide - test.c

File Edit

#include<stdio.h>
void main()
{
int a ;
int b = 0 ;
a = b ;
while(a<10)
a++ ;
}

成功
成功生成hex于
C:/Users/user/Desktop/c/
OK

:020000040800F2
:04000000FE010113E9
:0400040000812E2326
:0400080002010413DA
:04000C00FEC4278384
:040010000178793BB
:04001400FEF42623AD
:0400180000000013D1
:04001C0001C12403F7
:0400200002010113C5
:0400240000008067F1
:0400000508000000ef
:00000001FF

t.txt	2019/12/3 17:26	文本文档	1 KB
test.a	2019/12/3 17:08	A 文件	1 KB
test.c	2019/12/3 16:54	C 文件	1 KB
test.hex	2019/12/3 17:26	HEX 文件	1 KB
test.o	2019/12/3 17:26	O 文件	1 KB
test.s	2019/12/3 17:26	S 文件	1 KB

下面是部分生成的代码：

test.hex - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

:020000040800F2
:04000000FE010113E9
:0400040000812E2326
:0400080002010413DA
:04000C00FEC42623A5
:04001000FE4278380
:04001400FEF42423AF
:04001800FE842783B8
:04001C000178793AF
:04002000FEF42423A3
:0400240000000013C5
:0400280001C12403EB
:04002C0002010113B9
:0400300000008067E5
:0400000508000000ef
:00000001FF

PRV332 ide - test.c

File Edit

#include<stdio.h>
void main()
{
int a ;
int b = 0 ;
a = b ;
while(a<10)
a++ ;
}

成功
成功生成hex于
C:/Users/user/Desktop/c/
OK

qt_test

"test.c"
nopic
ute arch, "rv32i2p0"
ute unaligned_access, 0
ute stack_align, 16

2
main
main, @function

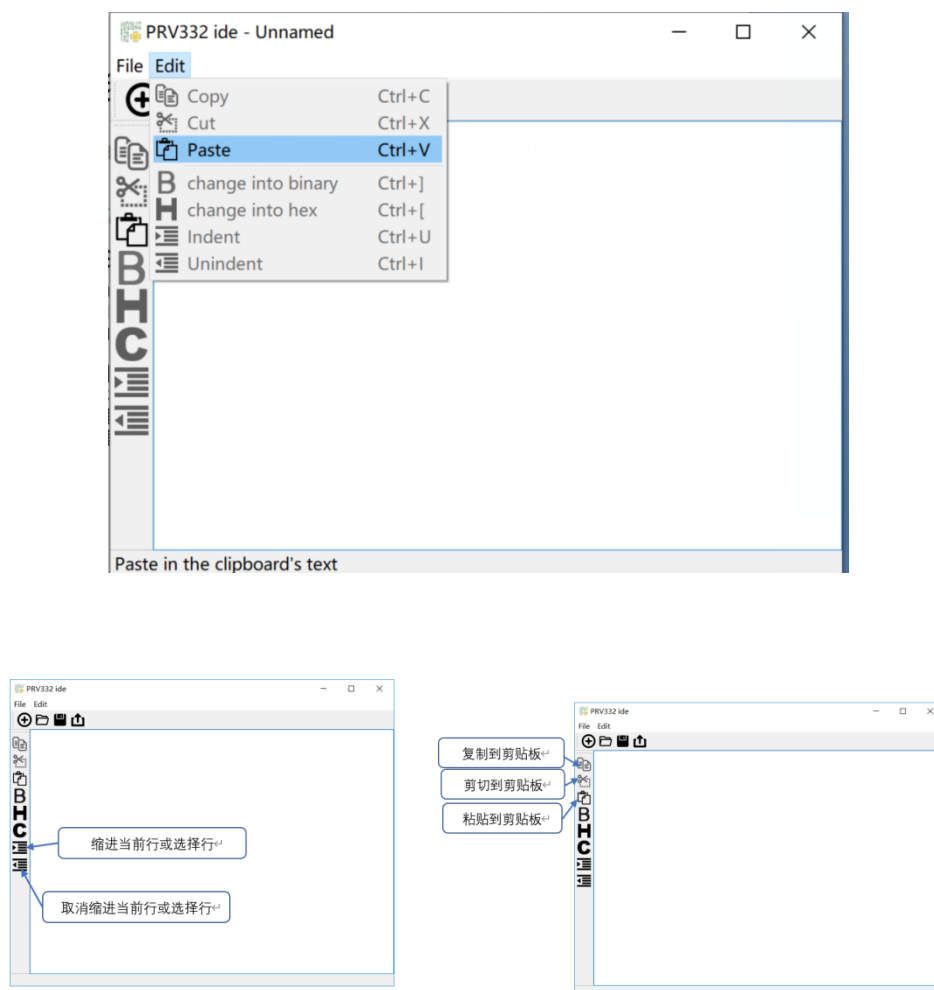
sp,sp,-32
s0,28(sp)
s0,sp,32
zero,-20(s0)
a5,-20(s0)
a5,-24(s0)
a5,-24(s0)
a5,a5,1
a5,-24(s0)
addi
sw
nop
lw
addi
jr
.size
.ident
main, -.main
"GCC: (GNU) 9.2.0"

名称 修改日期
rvc 2019/12/3 13:49
t.txt 2019/12/3 17:37
test.a 2019/12/3 17:08
test.c 2019/12/3 16:54
test.hex 2019/12/3 17:37
test.o 2019/12/3 17:37
test.s 2019/12/3 17:37
HEX 文件 1 KB
O 文件 1 KB
S 文件 1 KB

6) 部分其他功能和快捷键

prv332ide 同时支持很多其他功能，并会在无法使用该功能时改变透明度变为灰色，开发者可以方便的知道目前可以进行的操作

快捷键如下图所示：



附录二：PRV464 处理器编程手册（部分）

在目前计算机教学中，还有很多学校使用的是固定逻辑器件如 74xx 系列逻辑电路培训学生数字逻辑能力，这种方式下学生所学到的本领早已不适合当前高速发展的数字逻辑行业对人才的需求。随着近几年可编程逻辑器件，尤其是 FPGA 的高速发展：10K-100K 门级的 FPGA 价格进入到大部分人可以接受的价格，同时 FPGA 性能得到了长足的发展，使得在单片 FPGA 中实现一个功能完

备，性能较好的系统成为可能，部分学校开始考虑使用 FPGA 平台搭载计算机原理/数字逻辑教学平台来适应新时代对学生能力的需求。

RISC-V，或者简称 RV，是美国加州伯克利大学研发的第五代精简指令系统（RISC：精简指令系统计算机、V：第五代），它同时具备开源、简洁、可定制化的特点，目前采用 RISC-V 指令集处理器的系统已经在嵌入式市场占据了一部分市场份额，其市场占有率自 2015 年起逐步提高。

近数十年来，我国计算机专业学生学习的处理器指令体系主要是 X86-16（诞生于 1978），8051（诞生于 1975），或者一小部分学习 MIPS-32 和 arm32 指令集（诞生于 80 年代），这些指令集要么过于老旧（如 X86-16），要么已经在市场竞争中逐渐走向衰落（如 MIPS，MIPS 指令集因为 MIPS 公司的破产而变得群魔乱舞，各家指令系统之间都有区别），或者是因为在长期的发展中变得过于复杂而难以学习，如 ARM 公司的 ARM 系列指令集，其指令手册达到了 2000 页，X86-32/64 含拓展指令的指令文档更是达到了 6000 页的恐怖程度，这对于教学来说是非常困难的。RISC-V 指令集充分的吸取了上述指令集的特点，根据数十年来计算机工业发展的实践教训，大幅缩减了指令体系，采用模块化指令拓展，其指令文档仅有 219 页，特权架构文档仅有 79 页（2019608 版）。精简并不代表功能精简，相反，得益于设计年份很晚，RISC-V 在设计之初就考虑了 32 位，64 位乃至 128 位的指令集，而不像之前的指令集设计的时候压根就没考虑更高的位数。目前，RISC-V 处理器已经获得了 Openn-SBI，并建立了 UEFI 引导完整 linux 操作系统的能力。

本处理器旨在设计一种 64 位采用 I，A 指令拓展的 RISC-V 处理器软核（简称 RV64IA），这个处理器是之前设计的采用状态机设计的 RV32IA 处理器的一个进阶版本，为了能与之前设计的 RV32 处理器形成高低搭配。在设计上，本处理器将会采用 4 级流水线结构，采用 4KbyteL1 指令缓存+4KbyteL1 数据缓存，并采用 MMU，将会完全的具备运行 linux 操作系统的能力。

1) PRV464 流水线结构

PRV464 使用四级流水线，分别是：IF（取指令），ID（指令解码），EX（执行），WB（写回）

2) PRV464 支持的通用寄存器、CSR 列表

PRV464 支持 RV64IA 指令集，一共支持 32 个通用寄存器：

寄存器名称	ABI 名称	解释	
X0	Zero	常值 0	
X1	RA	返回地址	
X2	SP	堆栈指针	
X3	GP	全局指针	
X4	TP	线指针	
X5	T0	临时值	
X6-T7	T1-T2	临时值	
X8	S0/FP	保存寄存器/帧指针	
X9	S1	保存寄存器	
X10-X11	A0-A1		
X12-X17	A2-A7		
X18-X27	S2-S11		
X28-X31	T3-T6		

CSR 寄存器列表：

地址	权限	名称	
0xC00	URO	CYCLE	
0xC01	URO	TIME	
0xC02	URO	INSTRET	
0xC03	URO	HPMCOUNTER 3	
0xC04	URO	HPMCOUNTER 4	
0x100	SRW	SSTATUS	
0x102	SRW	SEDELEG	
0x103	SRW	SIDELEG	
0x104	SRW	SIE	
0x105	SRW	STVEC	

0x106	SRW	SCOUNTEREN	
0x140	SRW	SSCRATCH	
0x141	SRW	SEPC	
0x142	SRW	SCAUSE	
0x143	SRW	STVAL	
0x144	SRW	SIP	
0x180	SRW	SATP	
0xF11	MRO	MVENDORID	
0xF12	MRO	MARCHID	
0xF13	MRO	MIMPID	
0xF14	MRO	MHARTID	
0x300	MRW	MSTATUS	
0x301	MRW	MISA	
0x302	MRW	MEDELEG	
0x303	MRW	MIDELEG	
0x304	MRW	MIE	
0x305	MRW	MTVEC	
0x306	MRW	SCOUNTEREN	
0x3A0	MRW	PMPCFG0	没有功能
0x3B0	MRW	PMPADDR1	没有功能
0x3B1	MRW	PMPADDR2	没有功能
0xB00	MRW	MCYCLE	
0xB02	MRW	MINSTRET	
0xB03	MRW	MPHCOUNTER 3	
0xB04	MRW	MPHCOUNTER 4	

0x320	MRW	MCOUNTINHIB IT	
0x323	MRW	MHPMEVENT3	

3) PRV464 支持的指令集列表

