AG32 使用入门

目录:

- 一、 了解工程目录结构
- 二、 初识 IDE 中的项目
- 三、 认识 platformIO
- 四、工程配置

Platform.ini 配置、ve 配置;

- 五、 编译代码
- 六、 烧录程序 (和 ve 配置)

Jlink 烧录、串口烧录;

- 七、 Jlink 仿真
- 八、 进入开发
- 九、 增加编译目录
- 十、 开发中的注意项

阅读本文前,请参照《AG32开发环境搭建.pdf》,先搭建好 AG32 运行环境。

一、了解工程目录结构:

SDK 工作目录结构如下:



以上红色标出的几个目录,是在后续编写代码时常用的几个目录。

其中:

framework-agrv_sdk 是整个 SDK 的基础支撑部分。包含全部的驱动代码、board 代码、输出重定向代码。该 framwork 是用户在创建自己工程时必须依赖的 framwork。

除了 agrv_sdk,还有几个已经集成进 SDK 的 framwork,包括:ips/lwip/usb。用 户可根据自己需求决定是否使用它们。

展开 framework-agrv_sdk 的目录,可以看到:

✓ framework-agrv_sdk	
🗸 📙 etc	
> 📙 boards 👉 board代码对接	
> 📙 jlink	
> 📙 misc	
resources	
🔜 src 🛹 全部的驱动代码	

再看 examples 下,是全部的调用驱动的样例代码:



除了以上代码部分,还有两处重要的配置文件:

1. example_board.ve 芯片配置:

位于路径: AgRV_pio\platforms\AgRV\examples\example

该配置文件中,配置芯片主频及 pin 脚映射。

2. platformio.ini 工程配置:

位于路径: AgRV_pio\platforms\AgRV\examples\example

该文件是工程配置文件,里边定义 IDE 的编译/烧录/仿真及工程宏的选项。

以上, 先有个整体印象, 后续会深入介绍。

二、初识 IDE 中的项目:

看完目录结构,再来看 Demo 工程。

1	文件(E) 编辑(E) 选择(S) 查看(V)	转到(G) 运行(R)	终端(I) 帮助(H)	example.c - example - Visual
ſſ	资源管理器	🤯 PIO Home	C example.c >	<
G	v example	src > C exa	mple c >	
0			(-/ »	
γ	> .pio	25		
	2 .vscode	20	main(unid)	
্বুহ	✓ sic	28	marn(vord)	
	c example_analog.c	20 /	/ This will init (lock and wart on the board
1	C example_can.c	30 b	oard init():	
æ	C example_crc.c	31		
~0	C example_fcb.c	32 /	/ The default isr	table is plic_isr. The defaul
Ш	C example_flash.c	33 /	/ GPIO0_isr(), and	d can be re-assigned.
10000	C example_gpio.c	34 p	lic_isr[BUT_GPI0_]	[RQ] = Button_isr;
ð	C example_gptimer.c	35 /	/ Any interrupt p	riority needs to be greater th
	C example i2c.c	36 I	NT_SetIRQThreshold	(MIN_IRQ_PRIORITY);
(2)	C example rtc.c	37 /	/ Enable interrup	t from BUT_GPIO
	C example snic	38 I	NT_EnableIRQ(BUT_C	<pre>SPIO_IRQ, PLIC_MAX_PRIORITY);</pre>
	C example system c	39		
	C example_system.c	40 /	/ TestMtimer(500)	
	c example_timer.c	41 /	/ TestAnalog();	
	example_uart.c	42 /	/ TestCan();	
	C example_watchdog.c	43 /	/ TestEch():	
	C example.c	45 /	/ TestGnTimer():	
	C example.h	46 /	/ TestGpTimerPwm()):
	 .gitignore 	47 /	/ TestI2c();	
	example_board.asf	48 /	/ TestRTC();	
	example_board.post.asf		<pre>/ TestSpi();</pre>	
		50 /	<pre>/ TestSystem();</pre>	
_	E example board ve	51 /	<pre>/ TestTimer();</pre>	
	di platformio ini	52 /	<pre>/ TestWdog();</pre>	
	Plationinoani	53 /	<pre>/ TestUart();</pre>	

AGM 是基于 VSCode 来搭建的环境,编辑、调试、烧录也都基于此。

在 Demo 里, example.c 中的 main 总领了全部的驱动测试入口;

board_init 函数中, 会进行时钟初始化、GPIO 初始化、log 串口初始化;

INT_EnableIRQ 是使能某种类型的中断;

在 board_init 中初始化完串口后,代码中的 printf,将重定向到该串口输出。具体使用哪个串口输出,是通过 ve 配置中的 logger_if 字段配置的。

三、认识 platformIO:

在开始正式的编码和配置前,先了解下 platformIO 是什么。

在前边安装完 VSCode 后,首先下载的插件就是 platformIO IDE。即,它是 VScode 的一款第三方插件,这款插件是一套 IDE 开发环境。

事实上, platformIO 是一套开放的 Iot 集成环境平台, 允许芯片厂商通过配置和对接, 方便的实现芯片开发环境。

也就是说, VSCode + platformIO + 芯片厂商对接 = 该芯片的 IDE 编译环境。

在嵌入式开发中,大家比较熟悉的 IDE 会有 Keil、IAR。而上边构建出来的 IDE 环境就是类似 Keil、IAR 的一整套可开发仿真的环境。只是这套环境比 Keil 和 IAR 更强大。

既然是基于 platformIO 平台,那么项目中大多数的配置就是围绕 platformIO 来展开的。后边用到时会逐步展开。

扩展信息:

查看 platformIO 的官网: platformIO 的定位是新一代的 IoT 集成开发环境。它是基于 VSCode 的一款插件。VSCode 这款强大的文本编辑器辅以 PlatformIO 插件就可以化身为 一款强大的 MCU 开发环境,支持绝大多数流行的单片机平台。

我们知道, 嵌入式 Iot 开发中, 最让人不舒服的就是不同厂家的芯片要使用不同的集成开 发环境。例如: STM32 要使用 Keil 或 IAR, Arduino 默认使用自家 Arduino 开发环境, ESP32 要使用 linux 环境或者在 windows 下部署 eclipse 再用交叉编译。那么,有没有一个 IDE 可以大一统起来,集成大多数常用的芯片和模块的开发任务,只要配置完成之后就一 劳永逸的呢? 没错,那就是 PlatformIO。PlatformIO 试图整合起目前所有主流的硬件平 台: TI/ST/EspressIf/Intel/Silicon/...等,并且提供更便利的接口和更友好的交互,以提高 开发效率。

关于 platformIO, 有兴趣可以去官网获取更多的信息: https://platformio.org/

前边说到,每一款芯片在 PlatformIO 中需要配置,按照 PlatformIO 的格式配置后,才能 被正常使用(platformio.ini 有大量的标准的控制选项,可进入官网查看)。

官方配置: <u>https://docs.platformio.org/en/latest/projectconf/index.html#projectconf</u>

在实际使用中,除了官方标准配置(编译、烧录)外,芯片方也会在这个开放平台上自定 义一些自己特有的配置项。

四、工程配置:

大致了解完项目结构,接下来了解工程配置。(fpga 不在本文介绍) 两部分工程配置:**platformio.ini 和 agrv2k_103.ve** 其中:

platformio.ini 配置 工程的编译、烧录、仿真的选项;

与 IDE 相关。

agrv2k_103.ve 配置 芯片系统时钟、芯片 IO 引脚映射 等;

该文件中的配置是要被写入到 flash 中的,做为该芯片独特的配置。

platformio.ini 配置项:

从名字就可以看出, platformio.ini 是应用到 platformIO 的配置。

基于 platformIO 的项目,都必须配置一个 platformio.ini 文件,文件中对必要的配置字段进行赋值,从而告诉 platformIO 如何对该项目进行编译、烧录和仿真。

(这些除了标准字段,还有些是自定义字段,用于编译中)

先看 Demo 中的配置项:

boards_dir = boards //指定 board 对应的工作路径(用于代码编译的 path)

board = agrv2k_103 //使用 boards_dir 路径下的哪个硬件版本

//注: boards_dir 和 board 共同组成了完整 path。

//如:以上对应路径\packages\framework-agrv_sdk\etc\boards\agrv2k_103。

//这个路径下的.c和.h 会被编译到工程,这个路径下的资源被默认查找;

board_logic.ve = example_board.ve

//工程使用到的 ve 配置文件 (要烧写到 flash 中的)

//如果用相对路径,是相对于 platform.ini 的文件路径(即:工程路径)

framework = agrv_sdk, agrv_lwip //使用工程中的哪些库。SDK 是必带的。

//工程中默认带的库有: sdk、lwip、tinyUSB、Ips。

//使用多个库时名字中间加逗号隔开。

program = agm_example //目标工程名

src_dir = user //参与编译的 c 文件基目录 (路径相对于工程路径)

include_dir = user //参与链接的h文件基目录(路径相对于工程路径)

src_filter = "-<*> +<*.c> +<print/*.c> " //参与编译的 c 文件路径列表

//*用于通配,+增加,-去除。路径基于上边 src_dir 的基路径

src_build_flags = -Iuser -Iuser/print //头文件路径列表

//-I 后边是一个个文件夹, 各项之间用空格来分开

logger_if = UART0 //芯片串口输出 log 用的串口号(对应代码中 printf 函数)

//串口的波特率在用户程序中初始化串口时设置。

//注意: UARTO 同时是代码烧录的指定串口

monitor_port = COM3 //platform monitor 功能对接的端口, usb 的要填 usb 口 monitor_speed = 57600 //monitor 的速度(如: 串口的 115200/57600/...)

//monitor 功能是把配置端口收到的信息重映射到 VSCode 终端窗口输出

//更多用法参考: pio device monitor — PlatformIO latest documentation

upload_port = COM3 //串口烧录时 PC 端接的串口号 (usb 烧录的要填 usb 口)

upload_protocol = jlink-openocd //烧录固件的工具, 串口要填: serial

//upload 烧录程序时使用的方式和端口号

debug_tool = jlink-openocd //jtag 的工具,目前只能选 jlink-openocd

debug_speed = 10000 //jlink 的数据速度

//jlink 在线跟踪时的设置

更多的配置项,可自行参考 platformio 官方文档: <u>https://docs.platformio.org/en/latest/projectconf/index.html#projectconf</u>

example_board.ve 配置项:

AGM 芯片支持 fpga,这个.ve 文件就是映射芯片管脚定义的。

如下图,该 ve 是通过 platform.ini 进行关联配置的。配置后,通过命令再把 ve 文 件中的配置内容烧录到 flash 中去。



ve 文件内容:

SYSCLK 100 #系统时钟频率, M 为单位 HSECLK 8 #外部晶振频率, M 为单位, 取值范围 4~16

UART0_UARTRXD PIN_69 #串口 0 的收引脚 ----目前被用于 log 输出。 UART0_UARTTXD PIN_68 #串口 0 的发引脚

GPIO6_2 PIN_23 #IO_Button1 GPIO6_4 PIN_24 #IO_xxxx

•••••

•••••

关于 GPIOx_y 的说明:

AGM 芯片内共有 10 组 GPIO (GPIO0 – GPIO9), 每组 8 个 IO (0~7)。所以,可用的对外映射到 PIN 的 GPIO 共有 80 个。 (100pin/64pin/48pin 只是外部封装的区别,不影响内部 GPIO 的定义和数量)

这里的 GPIOx_y 表示的是第 x 组第 y 个 IO。

除了 GPIO,更多 IO 宏的定义(如:UART,SPI,CAN 等)参考文档《AGRV2K_逻辑设置.pdf》

PIN_XX 对应芯片的外引脚。每个 GPIOx_y 可以配置映射到任意一个引脚。

注意:有些模拟输出的引脚,在 64pin 和 100pin 上的对应是不同的。因而,使 用时最好在各自的 ve 模板上修改,不要混用。

五、编译程序:

程序编译可以通过 3 种方式: 命令行、pio 下栏按钮、pio 左栏按钮;

1. 命令行方式:

在"终端"通过命令行进行的工程编译。命令: pio run -e dev -v

庙	Ħ	⊅ ⊓	夂	•
灭	НЈ	хн	-	٠



2. Pio 下栏按钮:



这里的编译和1中的命令一样。编译后, 会自动弹出终端窗口。

3. Pio 左栏按钮:



不管使用上边的哪种方式,编译和烧录成功时,会有 success 的提示如下:

问题 2	出调试控制	台 <u>终端</u>	
** Programmi ** Verify St	ng Finished	**	
** Verified ** Resetting shutdown com	OK ** g Target ** mand invoke	· _	====== [SUCCESS] Took 5
Environment	Status	Duration	
release	SUCCESS	00:00:05.706	
PS D:\AGM MC	U doc\AgRV (pio\platforms\AgRV\pr	======= 1 succeeded in 0 oject\loraGtw> []

六、烧录程序(和 ve 配置):

注意: AGM 烧录程序和烧录 VE 配置是分开的。

程序有改动,就编译后烧录程序; VE 有改动,就烧录 VE。两者相互独立,各烧 各的。烧录没有先后顺序之分。两者的烧录接线都相同,只有命令不同。

新开发板的第一次烧录,两者都需要烧录。

烧录支持采用两种方式:jlink 和串口。

Jlink 烧录:

1. 在首次烧录前,需要先安装插件 zadig-2.7.exe。

注: 该插件是 jlink 正常驱动的插件。在安装该插件前,确保电脑上已经安装过 jlink 的驱动,并能正常使用 jlink。

安装插件时,需要将 jlink 连接到电脑,然后再安装。

该插件位于 SDK 解压后的根目录下。

安装方式参下图:

🗾 Zadig	- 0	×
<u>D</u> evice <u>Options</u> <u>H</u> elp		
J-Link 1	~ □	Edit
Driver (jlink (v2.70.8.0)	7600. 16385) More Information WinUSB (libusb)	on
USB ID 1366 0101 Replace	Driver	<u>t)</u>
5 devices found.	4 Zadig 2.7	.765

2. 配置烧录方式:

如果使用 jlink 来烧录,需要在 platform.ini 中进行对应配置。

配置方法:

修改 upload_protocol 项, 使: upload_protocol = jlink-openocd

(注,使用 jlink 烧录时,无需配置 upload_port 项)

3. 烧录程序:

烧录方法和上边的编译相似,也是三种方式:命令方式、pio 左边和下边按钮。

烧录命令: pio run -e release -t upload

```
(Pio下边和左边按钮,紧邻`编译"按钮,不再赘述)
```

4. 烧录 ve 文件:

在烧录 ve 配置时,只支持一种方式:命令方式。

烧录命令: pio run -e release -t logic

注: 烧录到 flash 的 ve 文件,就是在 platform.ini 中配置的 board_logic_ve 项。如 样例中的: board_logic.ve = **example_board.ve**

5. 烧录结果提示:

在烧录固件或者 ve 配置完成时,都会有 SUCCESS 提示。

如果烧录失败, 会有红色 Error 信息给出对应的失败原因。

www.hizyuan.com

烧录中, 最常见的报错是 "Error connecting DP: cannot read IDR"。

如果是新焊接的板子,需要检查:芯片是否有虚焊、芯片供电、jlink 接线管脚是 否正常,是否对应;

如果是使用中出现,尝试重新上电芯片,重新插拔 jlink。

如果中途有重装过电脑,或者重新安装过 jlink 驱动,需要再次安装驱动插件。

串口烧录:

1. 串口烧录前,要先使芯片进入烧录模式;

进入烧录模式的方法: boot1 接地, boot0 接高。

2. 在 platform.ini 的配置里, 配置成串口烧录并指定 PC 使用的串口号;

配置方法:

修改 upload_protocol 项, 使: upload_protocol = Serial

修改 upload_port 项, 使: upload port = COMx (x 是编号)

烧录时的波特率可以在[env:serial] 中修改, 如: upload speed = 115200

(注: 烧录时芯片端必须使用 UARTO)

3. 烧录程序和烧录 ve;

串口烧录和 jlink 烧录时相似 (可参照上边 jlink 烧录)

命令也相似(把 release 换成 serial)。

烧录程序的命令: pio run -e serial -t upload

烧录 ve 的命令: pio run -e serial -t logic

4. 烧录成功后的反馈;

同 jlink 烧录相似,成功也会有 SUCCESS 提示;

烧录失败会有红色 FAIL 提示错误原因。

总结下几个常用命令:

编译 (debug) : pio run -e dev -v 串口烧录 ve 配置: pio run -e serial -t logic 串口烧录 code: pio run -e serial -t upload jlink 烧录 ve 配置: pio run -e release -t logic jlink 烧录 code: pio run -e release -t upload

七、jlink 仿真:

烧录完成后,如果要仿真跟踪代码时,可在 jlink 下按以下步骤启动:

Ch	资源管理器・・		🍯 plat	formio.ini ×	C etherne	
С	∼ loragtw		🍯 platformio.ini			
ر م	 () c_cpp_properties.json () extensions.json () launch.json () settings.json 		18 19 20 21	lwip_imp_ tinyusb_i logger_if	dir = mp_dir = = UART0	
₽ D	「行和调武 (Ctrl+Shift+D) > ITELE US > src 1		22 23 24 25	<pre>#monitor_ #upload_p monitor_p upload_po</pre>	port = /de ort = /dev ort = COM3 rt = COM3	
щ.	> temp-bak-nouse ~ user ~ arch		26 27 28	upload_pr #upload_p debug_too	otocol = j rotocol = l = jlink-	
ð	C cc.h C ethernetif.h C sys_arch.h ∽ print		29 30 31 32 33	<pre>monitor_s #monitor_ debug_spe build_fla</pre>	peed = 576 speed = 11 ed = 10000 gs = -DBAL	



正常运行起来后,这个样子:



接下来就可以单步程序了。debug 调试中的快捷键和 VS 一致。

八、进入开发:

前边已经了解 SDK 工程的编译烧录,这里简单介绍几个常用驱动的使用。

举例 gpio 的使用:

举例:用 pin3 引脚接 led 灯,并控制亮灯(高为亮),则需要做:

- 1. ve 文件中设置: GPIO4_5 PIN_3
- 2. 代码中定义三个宏: #define LED_G_GPIO GPIO4 #define LED_G_GPIO_MASK APB_MASK_GPIO4 #define LED_G_GPIO_BITS (1 << 5)
- 代码中调用: SYS_EnableAPBClock(LED_G_GPIO_MASK); GPIO_SetOutput(LED_G_GPIO, LED_G_GPIO_BITS); GPIO_SetHigh(LED_G_GPIO, LED_G_GPIO_BITS);
- 4. 先烧录 ve 部分;
- 5. 再编译烧录代码部分;

以上代码的解释:

1. 不管外部封装是多少 pin(48pin/64pin/100pin),芯片内部可用 gpio 都有 80 个。

在使用 gpio 时,需要把使用到的 gpio 关联到外部 pin 脚。

(没被关联的 gpio,即使在代码中操作它,也不会传导到外部 pin 脚)

这步关联动作是在 ve 文件中进行的。

ve 文件里的配置, GPIOx_y PIN_z, 就是关联的语句。

PIN_z,对应外部哪个引脚。z 取值范围(48 脚封装:0~47;64 脚封装:0~63)

GPIOx_y,对应内部哪个 GPIO。x 取值范围(0~9), y 取值范围(0~7);

80个 gpio 分为 10 组,每组 8个。用 GPIOx_y 的格式来表达。如:

GPIO0_1 对应第0组的第1个IO;

GPIO4_5 对应第4组的第5个IO;

GPIO9_7 对应第9组的第7个 IO...

2. 在代码中如何操作 GPIO:

仍以 GPIO4_5 举例:

在代码里定义:

#define LED_G_GPIO GPIO4

#define LED_G_GPIO_BITS (1 << 5),</pre>

则意味着 LED 对应到了 GPIO4_5。

调用函数 GPIO_SetOutput 设置改 IO 为输出模式:

GPIO_SetOutput(LED_G_GPIO, LED_G_GPIO_BITS);

然后就可以通过 GPIO_SetHigh/GPIO_SetLow 来置高置低 IO 口。

如果要上拉 GPIO, 在\platforms\AgRV\boards\agrv2k_x0x\board.asf 增加:

set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to PIN_3

下拉用关键字: WEAK_PULL_DOWN_RESISTOR

以上是 GPIO 的引脚配置,其他配置 (如 mac,spi,iic 等引脚配置)需要使用的宏,参考文档《AGRV2K_逻辑设置.pdf》

举例 log 输出:

通过串口烧录 mcu 程序时,是固定使用 mcu 的 uart0。建议 mcu 在运行时也使用 uart0 来输出 log。工程代码中已经重新定向,在代码中 printf 函数将通过 uart0 来输出。

1. 在 ve 中配置输出 Log 的串口:

 $logger_if = UART0$

2. 在 ve 中设置串口 0 的映射引脚, 如:

UARTO_UARTRXD PIN_69 UARTO_UARTTXD PIN_68

3. 在程序的 board_init()函数中,确认有 UART0 的初始化:

在该初始化中,有设置串口的波特率、位宽、停止位、奇偶位。



如果不想使用 UARTO 输出 log,要改用 UART1。那么在 ve 配置中对应修改:

 $logger_if = UART1$

UART1_UARTRXD PIN_xx UART1_UARTTXD PIN_xx

即可。

关于系统宏的定义:

在代码里的宏,在 VSCode 中通过鼠标停顿在上边,一般可以看到其定义值。

这些宏,除了代码中定义的外,还有"系统预制宏"也可以被代码中使用(编译时使用)。

系统宏,有些是直接定义在 platform.ini 中的 build_flags 字段中;有些则是在 platform.ini 中定义后再由 main.py 经过转换拼接,才最终使用,比如上边截图中的 LOGGER_UART。

在 platform.ini 中的 build_flags 字段中的宏,这个容易理解,无需再描述。

另外的宏("系统预制宏"),比如上边截图中的 LOGGER_UART,即没有在代码中定义,也 没在 build_flags 中定义。它的使用方式:

- a. 在 platform.ini 定义 logger_if = UART0
- b. 在 agrv_sdk.py 中引用该字段 logger_if, 并判定是 RTT 还是 UART,

如果是 UARTx,则定义 LOGGER_UART 为 x。

c. 这样, logger_if = UART0 时, 在 CPP 中就可以认为存在宏: #define LOGGER_UART 0
 这类宏使用很少, 在应用开发中不建议使用。

九、增加编译目录:

在开发中,新增了c文件和h文件,怎么自动编译进来?

- 1. 如果新增文件在原有路径下,则会被自动关联编译进來;
- 2. 如果新增一个目录文件,则要把该目录加入到编译选项中;

如果该目录存放 C 文件:在 src_filter 中增加该目录

如果该目录存放 h 文件:在 src_build_flags 中增加该目录

举例:

在项目里新增一个文件夹 testFolder 到 user 目录下,里边有.c 和.h,要全部编译进去。

原先的 src_filter 和 src_build_flags 对应如下:

src_filter = "-<*> +<*.c> +<print/*.c> "

src_build_flags = -Iuser -Iuser/print

那么, 增加 testFolder 后要变为:

src_filter = "-<*> +<*.c> +<print/*.c> +<testFolder/*.c>"

src_build_flags = -Iuser -Iuser/print -Iuser/testFolder

注意:在 src_filter 和 src_build_flags 中,都可以使用相对路径。他们的相对路径 是相对于 src_dir/include_dir 定义的那个路径。

这里的*是通配符,不让某个.c进入编译,则用-号:-<testFolder/nowork.c>

十、开发中的注意项:

关于芯片 flash 大小:

不管所选型号的 flash 是多大,请注意最后 100K 是留给 fpga 使用的。

如果使用的芯片是 256K 的 flash,那么就是 156K 程序+100K fpga,用户程序不 能超过 156K。如果超过 156K 编译是可以通过的,但烧录后会冲掉 ve 配置部分。

ve 配置被冲掉后,程序运行会表现出各种异常(连系统时钟初始化都跑不过)。

如果程序使用的空间较大, fpga 又刚好比较小, 可以调整这个界限的值。调整方法如下:

board_logic.compress = true // (可选) 对 fpga 部分进行压缩,更省空间 board_upload.logic_address = 0x80034000 //根据实际情况调整该边界值

flash 的大小是在 agrv2k_103.json 中定义的。

flash 起始地址是 0x8000000, ram 是 0x20000000。