GPIO 扩展基础应用说明

1 概述

本文是 AW GPIO 扩展芯片技术、工具、软件和设计指南相关信息的集合，它提供了利用 AW GPIO 开发工作所需的一切。

本指南被分为以下部分。

GPIO 扩展芯片方案应用 - 本章概述了 GPIO 扩展芯片方案应用。

器件家族 – 介绍了 AW GPIO 扩展芯片规格。

设计指南 - 为 AW GPI 扩展芯片的典型硬件原理图设计和布局提供指导。

EVB 及 GUI 使用说明 - AW GPIO 扩展芯片 GUI 是一款快速调试工具，本章详细介绍开发人员如何通过 AW GPIO demo 板搭配 GUI 完成项目开发调试。

MCU 驱动设计说明 – 此章节讲解了 AW GPIO 扩展芯片初始化流程以及功能实现。

2 GPIO 扩展芯片方案应用

2.1 GPIO 扩展芯片做为外设输入输出方案应用

主流的 MCU 平台 IO 口支持输入、输出和中断，内部上下拉，推腕和开漏，中断触发方式选择等功能，艾为 IO 扩展芯片已集成所有上述功能，可以无缝适配。

- **内部上下拉功能**：扩展 IO 芯片每一路 io 口内部有上下拉电阻，可以用于设置外设使能脚和中断脚的默认状态。
- **中断使能功能**：扩展 IO 芯片每一路 IO 口可单独使能中断，IO 口的电平变化会拉低扩展 IO 芯片的中断脚，通知平台 IO 的实时状态。
- **中断触发方式可配置**：IO 口电平变化触发中断支持电平触发，上升沿触发，下降沿触发，任意边沿触发四种方式。
- **输出模式可配置**：扩展 IO 芯片每一路 IO 口支持推腕和开漏两种方式，开漏模式需要外部上拉才能输出高电平，一般用作中断、按键

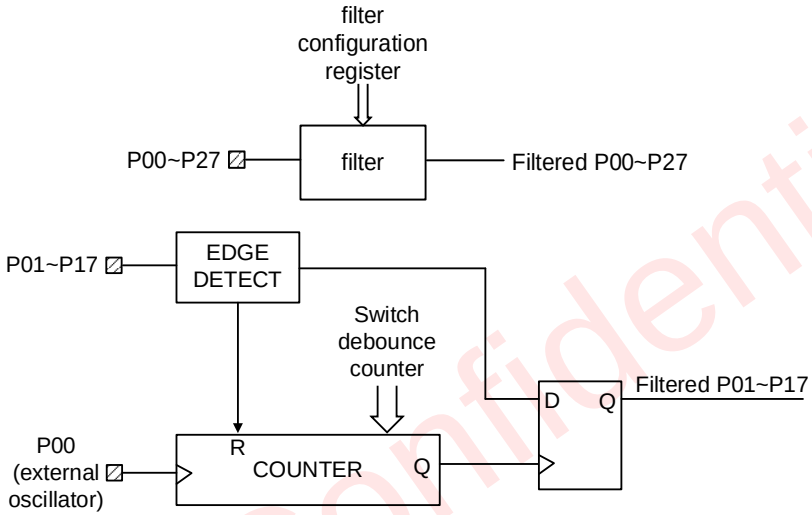
模式；推腕模式用作输出模式。

- **极性反转功能**：扩展 IO 芯片使能反转功能可使高电平反馈的信号是 0，低电平反馈的信号是 1。
- **IO 驱动能力可适配**

GPIO 输出切换（从高电平/低电平转变）时，存在一个与输出驱动选择相关的峰值电流。这个峰值电流会通过封装电感流动，并会产生噪声（部分是辐射噪声，但更重要的是开关噪声（SSN））。换句话说，同时对多个输出进行切换会创建接地和电源噪声。艾为 IO 扩展 IC 独立配置四个输出电流水平（0.25x，0.5x，0.75x，1x），通过改变晶体管通路对数，减少 io 的驱动电流，从而降低 io 电平切换产生的噪声。

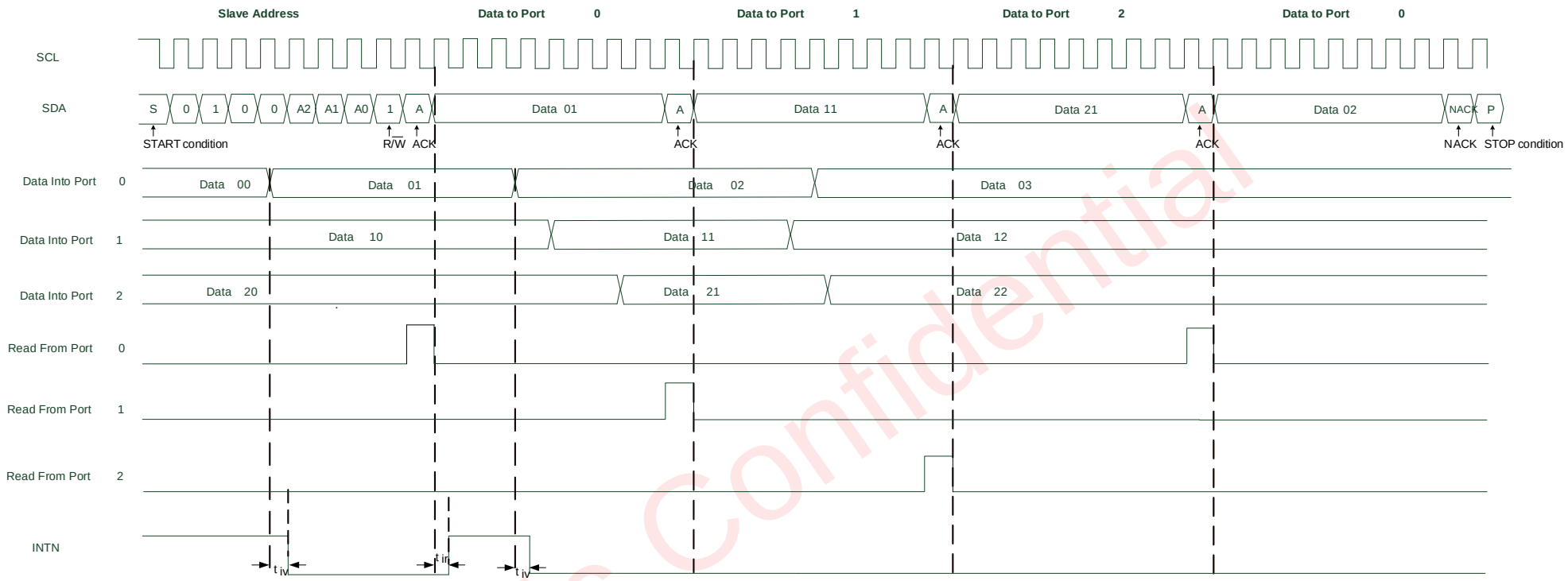
- **IO 硬件防抖功能**

当机械开关的触点闭合或断开时，理想情况下应该是瞬间完成，从“开”到“关”或从“关”到“开”的状态跳变。但现实中，由于触点的弹性、材质以及碰撞后的微小反弹，在毫秒级的时间内，触点会经历一个快速的、多次的物理通断过程，然后才达到稳定状态，这个过程就像是按一下按钮，电路实际上在极短时间内“快速开关”了很多次，为了解决这个问题，就必须进行“去抖”,其核心思想是在物理状态变化后，等待一段时间，待信号稳定后再进行采样。去抖可以分为两种方式：软件去抖和硬件去抖，软件去抖处理方式简单，就是连续读两次 io 的电平状态，以第二次状态为准；艾为扩展 IC 的硬件去抖方案——每个通道都有一个独立的模拟滤波器，能够滤除低于 20 纳秒的毛刺信号，除此之外，芯片内部有专门去抖电路处理电平切换的抖动，当输入状态发生变化时，边沿检测计算器会复位，当输入状态在完整的验证周期内保持稳定时，计数器会驱动触发器，更新状态。



- **IO 输入锁存功能**

GPIO 扩展 IO 作为中断使用时，如果设备的中断比较频繁并且中断处理逻辑比较复杂比如 NFC、矩阵按键，经常会出现漏中断的现象，艾为 IO 扩展芯片拥有“保留现场“的能力，主控没有处理对应 io 事件之前，该 io 的状态会被保存在寄存器，直到寄存器被读取之后，才更新成 io 的实时状态。



- Transfer of data can be stopped at any time by a Stop condition. When this occurs, data present at the latest acknowledge phase is valid (output mode). It is assumed that the command byte previously has been set to 00 (read Input Port register).
- This figure eliminates the command byte transfer from the P port, a restart, and slave address call between the initial slave address call and actual data transfer

2.2 GPIO 扩展芯片作为 LED 方案应用

工业、IOT 设备需要指示灯提示机器运行的状态，对于同时有 IO 扩展和 LED 指示灯需求的设备，扩展 IO 芯片是一个不错的选择。相比于直接用 GPIO 去驱动 LED 指示灯，扩展 IO 芯片每一路 IO 内置恒流源，最大电流可达 37ma，支持电流 255 级可调，也有型号内置呼吸引擎，可以实现自主呼吸功能，呼吸频率和颜色可动态调整。除此之外，还支持 8*8 的矩阵灯，可以应用在数码管等产品上。

2.3 GPIO 扩展芯片作为按键方案应用

无论工业上仪表操作器、机床控制面板还是消费电子的 pos 机，键盘等都需要按键功能的支撑，扩展 IO 芯片支持单独按键和矩阵按键两种方案。单独按键顾名思义跟 MCU 的 IO 按键方案一样，通过开漏模式让 IO 模式保持常高，当用户按下时会产生一个下降沿信号。矩阵按键则是利用行列扫描，将 M 个行引脚和 N 个列引脚组成一个 M×N 的网格，每个交叉点放置一个按键，这样每当一个按键按下时，就轮询切换信号产生列中每一路 IO 口的输出状态，直达找到对应的信号源。扩展 IO 芯片支持 12*12 = 144 键可以覆盖市面上大部分 104 键的键盘需求。

3 器件家族

芯片型号	GPIOs	LED Driver	Power Supply(V)	Level_shift	Reset	Internal Pull-up	Temperature	Package(mm)
AW9555TSR	16	-	1.65-5.5	-	-	√	~40℃~105℃	TSSOP-24L
AW9535TSR	16	-	1.65~5.5	-	-	-	~40℃~105℃	TSSOP-24L
AW9535QNR	16	-	1.65~5.5	-	-	-	~40℃~105℃	QFN 4×4-24L
AW95016ABGR	16	-	1.65~5.5	√	√	√	~40℃~85℃	BGA 3×3-24B
AW95016QNR	16	-	1.65~5.5	√	√	√	~40℃~85℃	QFN 4×4-24L
AW9523BTQR	16	√	2.5~5.5	-	√	-	~40℃~85℃	QFN 4×4-24L
AW9527QNR	16	√	2.5~5.5	-	√	-	~40℃~85℃	QFN 3×3-24L
AW9110CQNR	10	√	2.5~5.5	-	√	-	~40℃~85℃	QFN 3×3-20L
AW9106CQNR	6	√	2.5~5.5	-	√	-	~40℃~85℃	QFN 3×3-20L
AW95124QNR	24	-	1.08~3.6	√	√	-	~40℃~105℃	QFN 4×4-32L
AW95116SPR	16	-	1.08~3.6	-	√	-	~40℃~105℃	TSSOP-24L
AW9554SPR	8	-	1.08~3.6	-	-	√	~40℃~105℃	TSSOP-16L

4 设计指南

4.1 AW GPIO 扩展带 LED 模式典型原理图

4.1.1 AW9106CQNR/AW9110CQNR原理图介绍

以 AW9110C 为例，采用 1 颗芯片驱动 10 路 LED 或者 6 路背光的应用参考电路图如 4.1 所示：

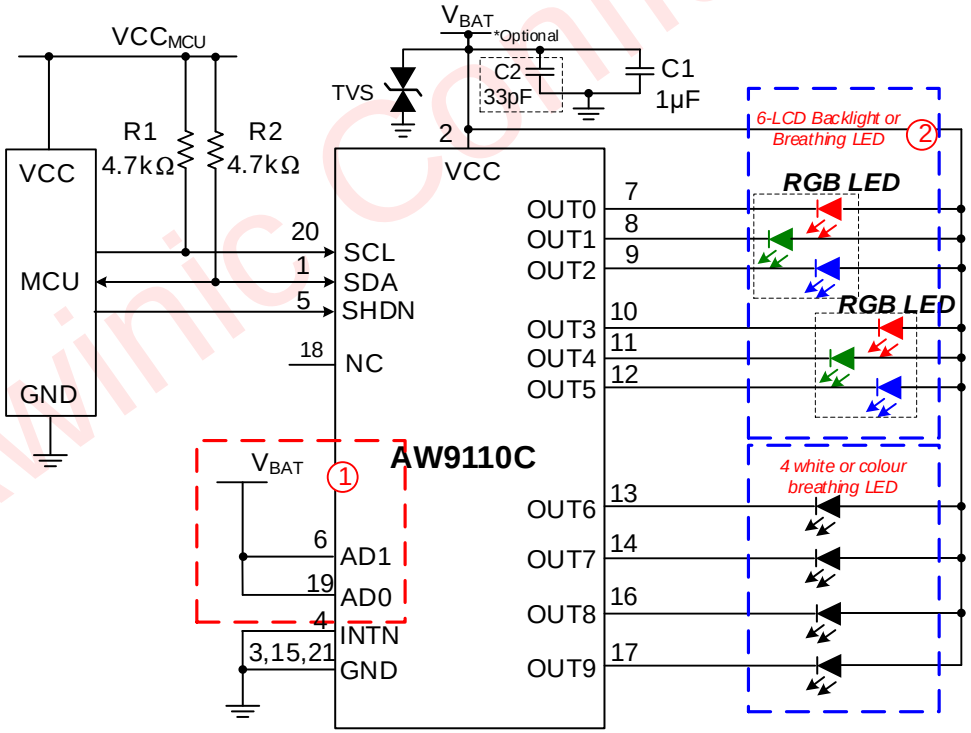


图 4.1 AW9110C 应用电路图

需要注意的是，当 LED 阳极供电为 VBAT 时， AD1/AD0 需要接至 VBAT，以保证芯片在上电后 OUTx 的初始态为高，而后再通过寄存器配置芯片进入 LED 模式。如未按照此硬件连接，芯片在上电后，灯会出现误点亮的现象。

采用 AW9110C 实现驱动 8 颗 RGB 的应用原理图如下，采用此方式需要重点注意对灯效的影响，如果是同时点亮同颗 RGB 不同颜色的灯，灯的阴极不能接同一个 OUTX 引脚，因为不同颜色灯的 VF 有差异，同时点亮 RGB1 和 RGB4 的时候，RGB2 和 RGB3 也会被点亮。在设计硬件的时候需要注意对灯效的影响。

建议在电源端，以及需要外接到 PCB 板外的接口处预留 TVS 位置用于防止可能会发生的 ESD 或浪涌电压。

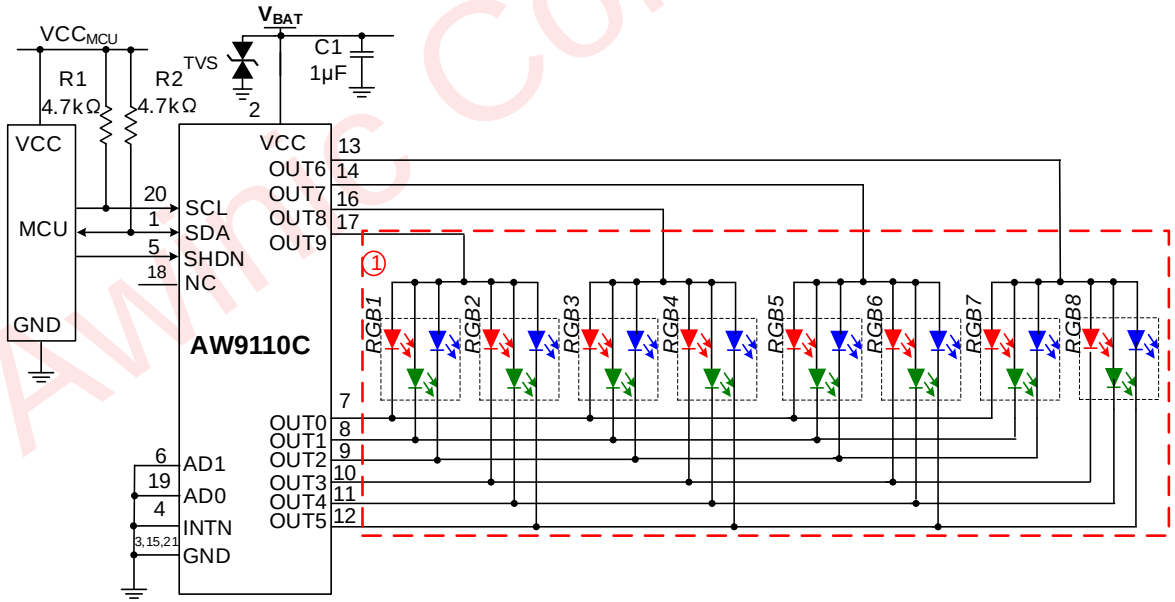


图 4.2 AW9110C 应用电路图

4.1.2 AW9527QNR/AW9523BTQR原理图介绍

AW952X 可以作为 GPIO 扩展或 LED 驱动使用，作 GPIO 扩展时的参考应用原理图如图 4.3 所示：

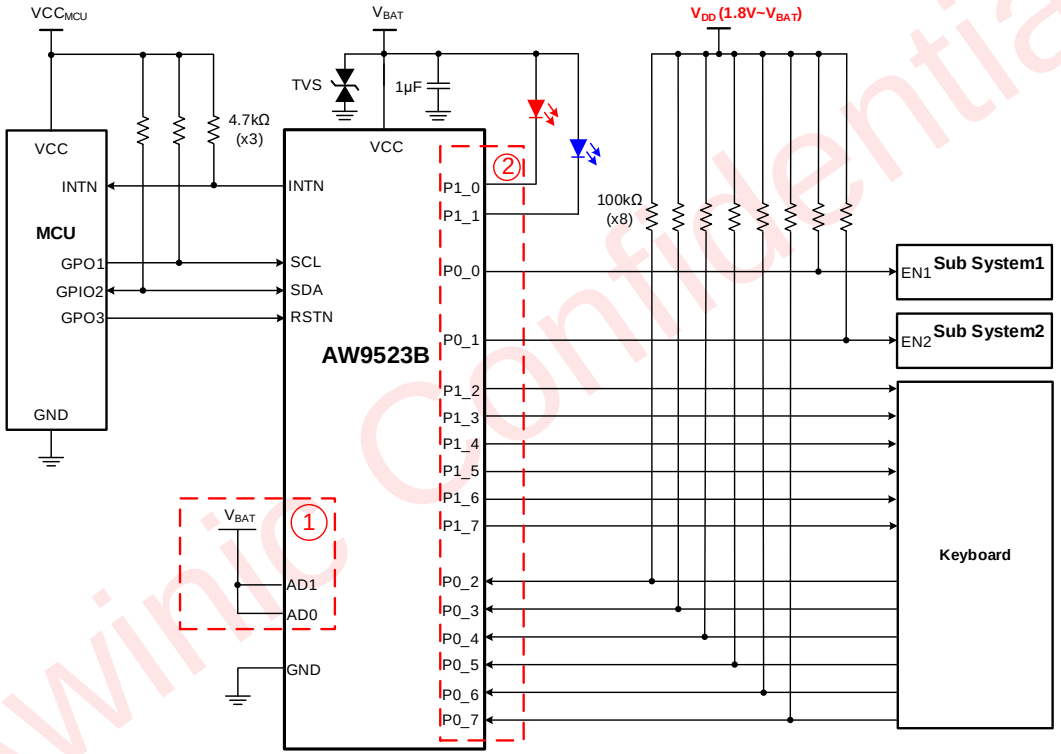


图 4.3 AW952X 作 GPIO 扩展应用电路图

- 输入电容推荐使用 1μF 的 X5R 或 X7R 电容，SCL、SDA、INTN 需要配置一个上拉电阻；
- AW952X I2C 地址可通过 AD1、AD0 选择，在输出存在 LED 时，需将 AD1、AD0 连接至 VBAT，保证 GPIO 的初始状态为高电平或高阻态，从而使得 LED 不会在系统上电时被错误地点亮，更多地址选择细节详见< AW952X I2C 地址选择 > ；

- AW952X 的每个输出都可以独立的被配置为 GPIO 或 LED 驱动。作 GPIO 输出时，P1 端口为 Push-Pull 输出；P0 端口默认为 Open-Drain 输出，也可以将其配置为 Push-Pull 输出，当作为 Open-Drain 输出时则需要为其准备上拉电阻；
- AW952X 在驱动开关器件如 MOSFET 或 BJT 时，建议在 GPIO 输出端预留一组 RC 延时电路；
- 建议在电源端，以及需要外接到 PCB 板外的接口处预留 TVS 位置用于防止可能会发生的 ESD 或浪涌电压。

AW952X 作为 LED 驱动时参考应用图如图 4.4 所示：

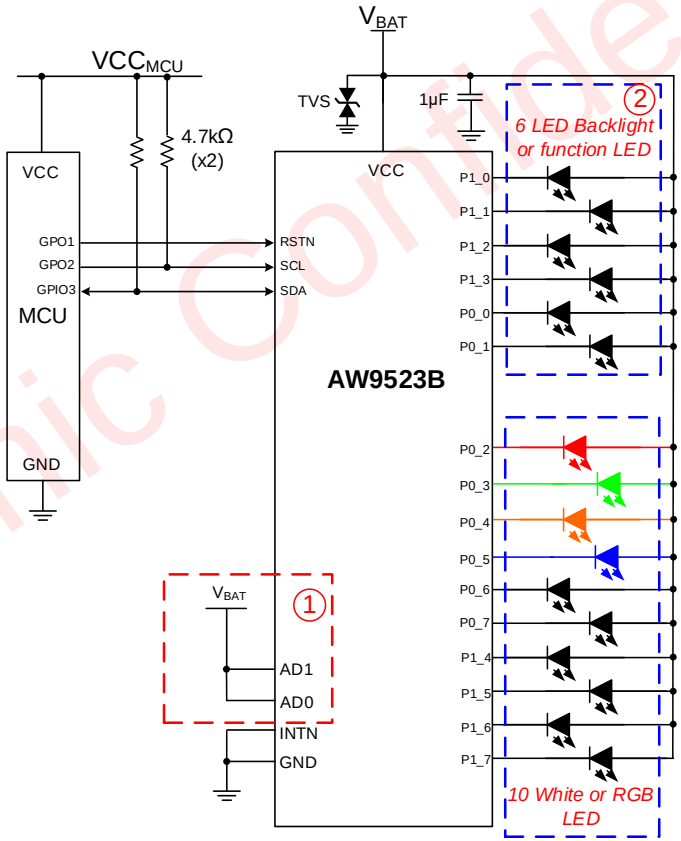


图 4.4　AW952X LED 应用电路图

- 输入电容推荐使用 1μF 的 X5R 或 X7R 电容，SCL、SDA 需要配置一个上拉电阻；
- AW952X 作 LED 驱动时，AD1 和 AD0 必须连接至 VBAT，以保证输出端口的初始状态为高电平或高阻态，避免 LED 被误点亮；
- AW952X 有六个通道的 Dropout 电压被专门优化过：P1_0~P1_3、P0_0~P0_1，这些通道在需要时可以作为 LED 背光使用；
- 建议在电源端，以及需要外接到 PCB 板外的接口处预留 TVS 位置用于防止可能会发生的 ESD 或浪涌电压；

AW9527 在小电流应用时，可以作为矩阵灯驱动，如图 4.5 所示：

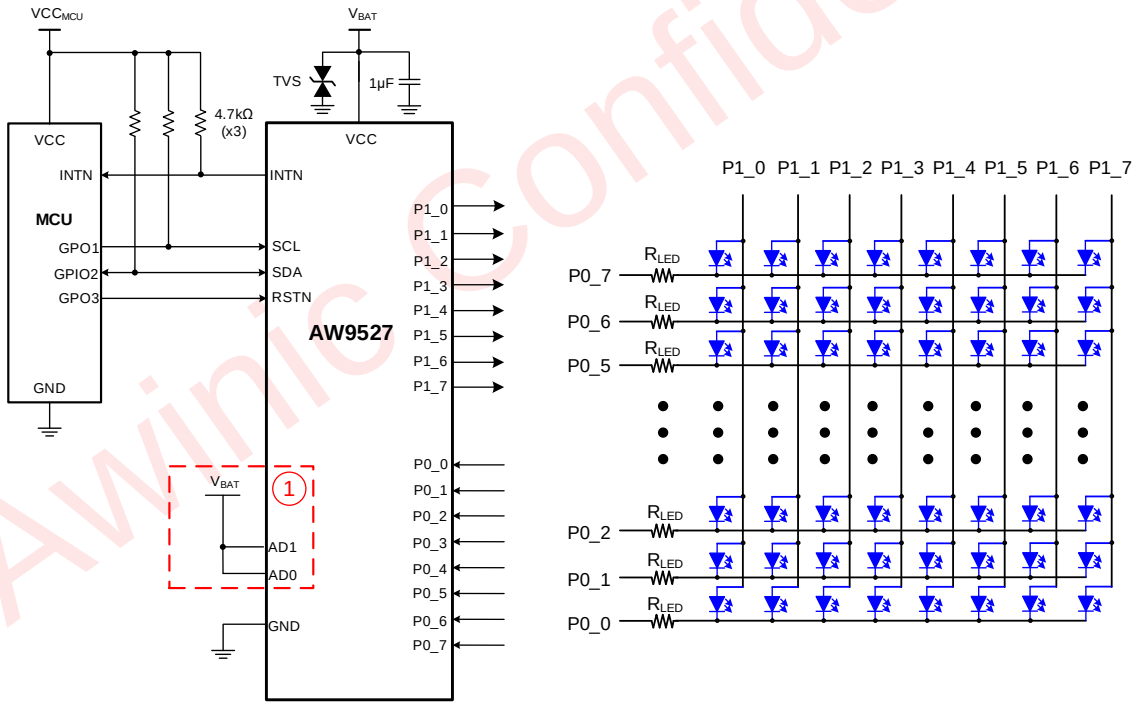


图 4.5　AW952X LED 应用电路图

- AW9106/AW9110/AW9523/AW9527 的 I2C 地址可通过 AD1 和 AD0 进行拓展；7 位 I2C 地址可设置为 0x58~0x5B，同一路 I2C 总线上

最多可支持 4 个器件，的 I2C 地址选择详见下表：

固定位	A1	A0	I2C 地址
10110	0	0	0x58
	0	1	0x59
	1	0	0x5A
	1	1	0x5B

4.2 AW GPIO 扩展不带 LED 模式典型原理图

4.2.1 AW95016ABGR/AW95016QNR原理图介绍

- AW95016(A)是一款带有 I2C 接口的 16 位 GPIO 扩展器，应用于外接键盘场景的典型应用图如图 4.6 所示；

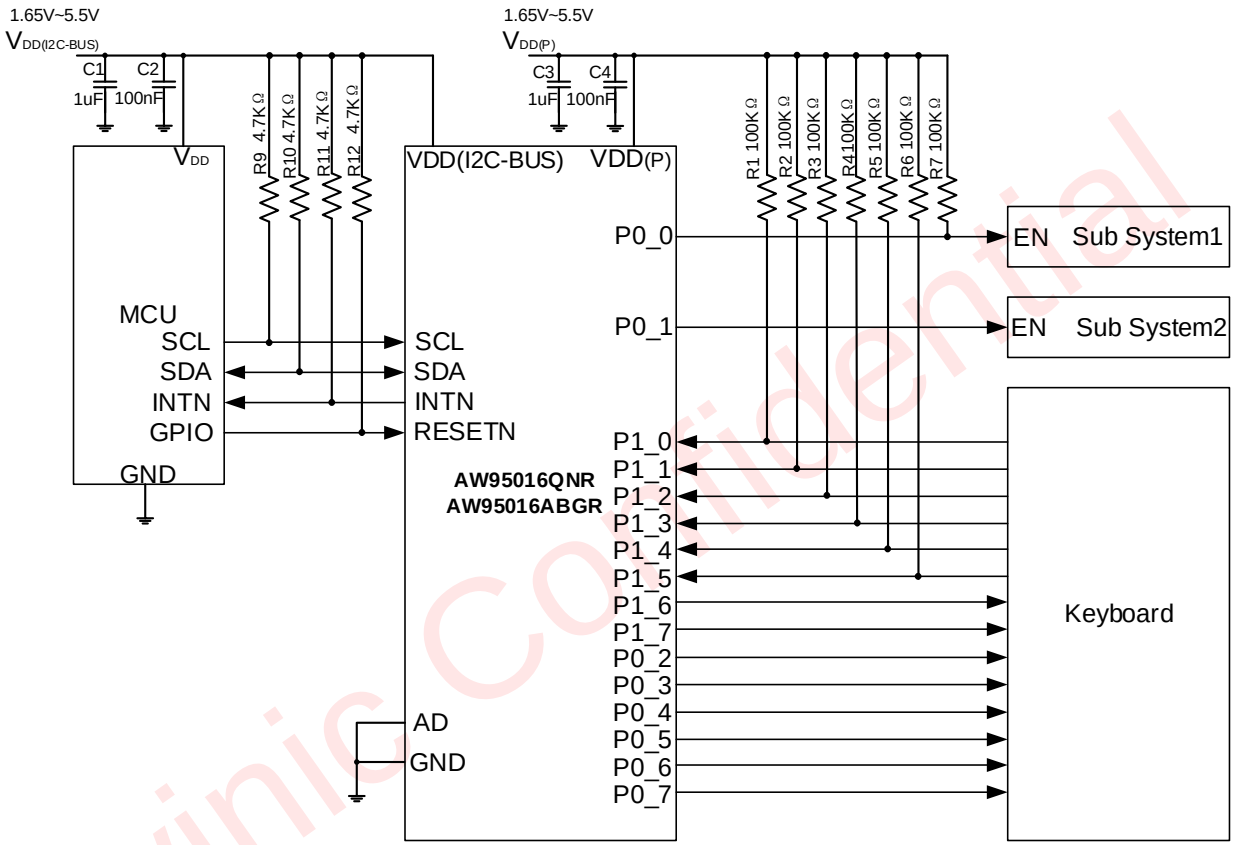


图 4.6 AW95016 键盘应用电路图

- AW95016(A)的电压范围为 1.65V～5.5V，电源输入端的并联电容推荐使用 1μF 和 100nF，当电源电压 VDD(I2C-BUS)或 VDD(P)不大于 3.3V 时，C1、C2、C3 和 C4 选用额定电压为不小于 6.3V 的电容；当电源大于 3.3V 时，C1、C2、C3 和 C4 选用额定电压为不小于 10V 的电容；
- 16 位 GPIO 口(P0_0~P0~7、P1_0~P1_7)，默认为输入接口，可以独立设置为输入接口或者输出接口；当作为输出端口时，可将端口设置为推挽模式或开漏模式；

- 未使用的 GPIO 口，建议将其配置成推挽输出；若为输入时，需连接固定电平，否则容易受外部干扰造成状态的变化；
- 当 GPIO 口作为推挽输出时，不需要额外配置上拉电阻；当 GPIO 口作为开漏输出时，需要配置上拉电阻；当 GPIO 口作为输入端口且浮空时，若内部没有配置成上拉或下拉模式，则需要外侧配置上拉电阻或者下拉电阻；
- 当 GPIO 口设置为开漏模式时，推荐上拉电阻的阻值为 100KΩ，具体阻值可根据调试进行选择，上拉的电平 VIO 默认不能大于 VDD(P)；当上拉的电平大于 VDD(P)时，会出现往芯片内部倒灌电流的现象；

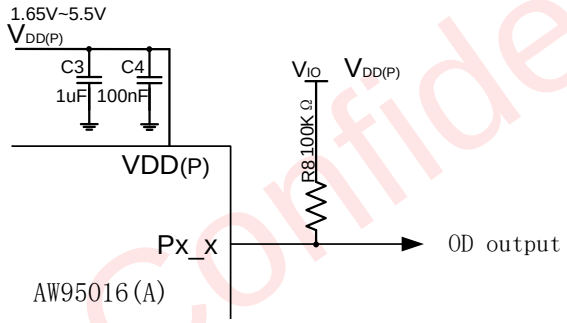


图 4.7 AW95016GPIO 口为开漏输出时的典型应用图

- GPIO 口为输入状态时，可以通过寄存器配置成内部上拉或者下拉（内部上拉电阻和下拉电阻为 100KΩ），但芯片复位重启后，会恢复成输入高阻态，需要配置寄存器才能恢复；应用为输入 IO 口时，推荐客户预留外部上拉电阻；
- 当 GPIO 口设置为输入模式时，输入的高电平 VIO 推荐选用 VDD(P)，当输入高电平高于 VDD(P)，会出现电流倒灌问题；当输入高电平低于 VDD(P)时，芯片功耗会增加(每路会增加 75μA 的功耗)；

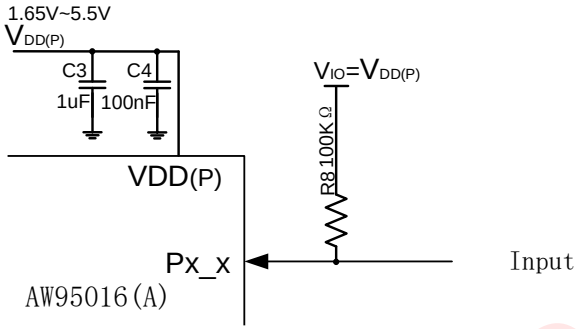


图 4.8 AW95016GPIO 口为输入时的典型应用图

- SCL、SDA、RESETN 信号的上拉电平 Vcc_host 不能低于 VDD(I2C_BUS)，建议接上拉电阻至 V DD(I2C_BUS)；当输入高电平低于 V DD(I2C_BUS)时，芯片的功耗会增加（每路会增加 20μA 的功耗）；

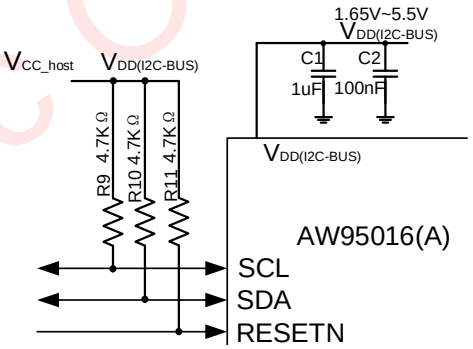


图 4.9 AW95016 控制信号的推荐上拉电平图

- 地址拓展位 AD 脚，需要接高电平时，建议接 VDD(P)，当输入高电平低于 VDD(P)时，芯片功耗会增加 75μA 的功耗；
- I2C 的上拉电阻 R9 和 R10，推荐上拉至 V DD(I2C_BUS)，当 I2C 是 400KHz ，建议上拉电阻取值为 4.7KΩ，当 I2C 是 1MHz 建议上拉

电阻取值为 1KΩ，具体阻值可根据调试进行选择；

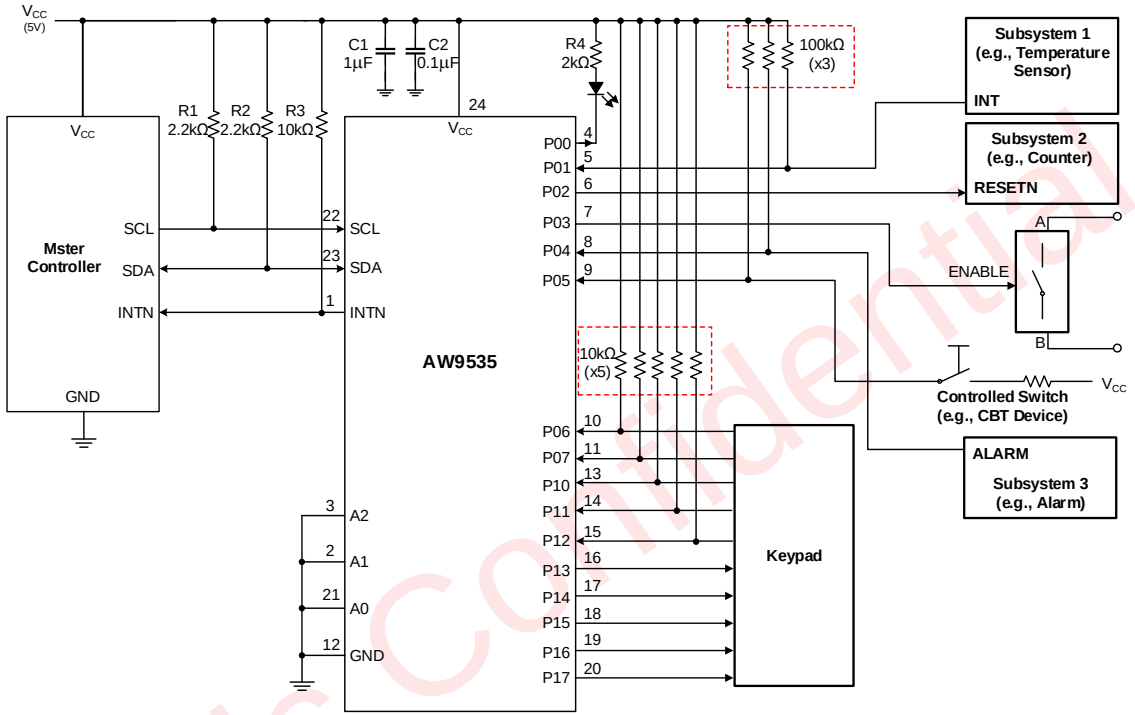
- RESETN 信号推荐上拉 R12 至 V DD(I2C_BUS)，建议接上拉电阻取值为 4.7KΩ，具体阻值可根据调试进行选择；
- INTN 端口是开漏输出，需要电阻 R11 上拉至 V DD(I2C_BUS)，推荐使用 4.7KΩ，具体阻值可根据调试进行选择；与 INTN 端口相连的 MCU 的 IO 口，需要支持唤醒功能；
- I2C 地址可通过 AD 选择，推荐 AD 引脚接 GND，7 位地址为 0x20，地址选择详见下表：

  芯片的 I2C 地址真值表：

AD PIN	A7:A3	A2:A1	A0	Chip Address	Broadcast Address
GND	01000	00	0/1	20h	1Ch
VDD		01		21h	

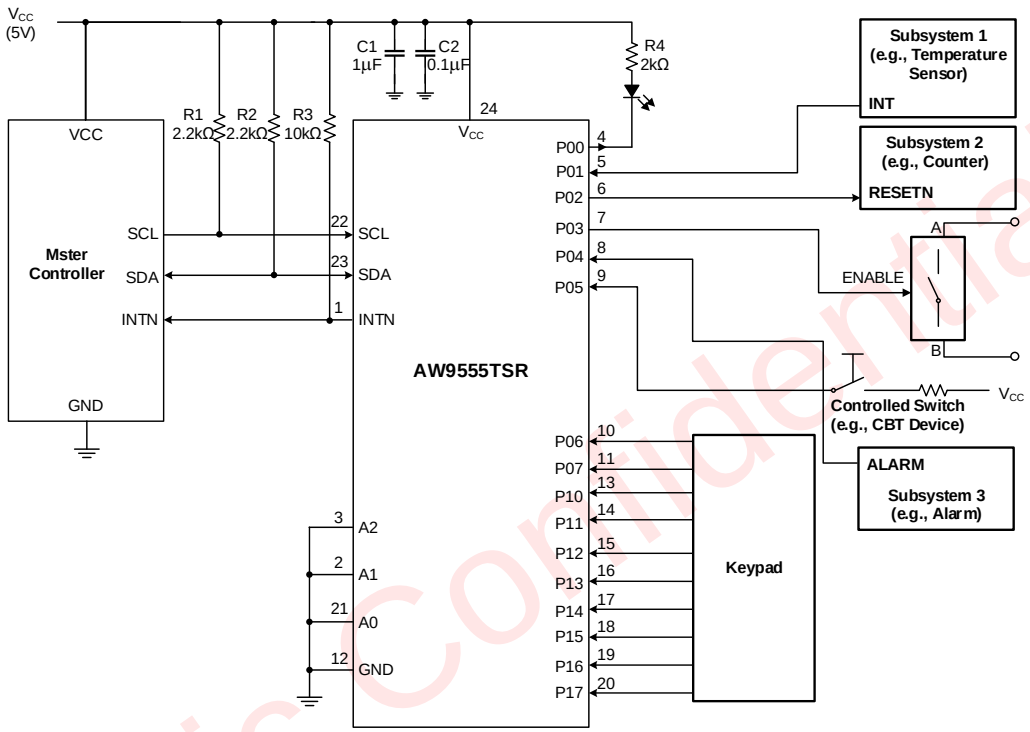
4.2.2 AW9535QNR/AW9555TSR原理图介绍

- AW9535/AW9555 作为 I2C 从设备通常位于主机的远程位置，靠近主机需要监控的 GPIO 端，图 2 为 AW9535 的典型应用电路，图 3 为 AW9555 的典型应用电路，AW9535/AW9555 作为 IO 扩展器通常用于控制 LED（用于反馈或状态灯）、控制其他设备的启用或重置信号，甚至读取其他设备或按键的输出状态等。



(1) In this application schematic, P00, P02, P03 and P13~P17 are configured as outputs.
(2) P01, P04~P07 and P10~P12 are configured as inputs.
(3) Device address is configured as 0100000 for this example.
(4) Pin numbers shown are for TSSOP packages.

图 4.10 AW9535 典型应用电路



(1) In this application schematic, P00, P02 and P03 are configured as outputs.
(2) P01, P04~P07 and P10~P17 are configured as inputs.
(3) Device address is configured as 0100000 for this example.

图 4.11 AW9555 典型应用电路

- AW9535/AW9555 的电压范围为 1.65V～5.5V，电源输入端的并联电容推荐使用 1μF 和 0.1μF，当电源电压 Vcc 不大于 3.3V 时，C1、C2 选用额定电压为不小于 6.3V 的电容；当电源大于 3.3V 时，C1、C2 选用额定电压为不小于 10V 的电容。
- AW9555 芯片内部自带弱上拉电阻 100kΩ，当 GPIO 口作为输入端口，默认输入为高电平。当 GPIO 口配置成输出时，为推挽输出，若外加的电压大于 Vcc，则会出现往芯片内部倒灌电流的现象，同时若在 IO 外部拉低时，则会进一步增加系统功耗。AW9535 芯片内部无上拉电阻，当 GPIO 口作为输入端口时，需连接固定电平，否则容易受外部干扰造成状态的变化，推荐预留外部上拉电阻；当 GPIO 口配置成输出时，为推挽输出。

- 当 GPIO 口设置为输入模式时，若输入的高电平大于 Vcc 时，则 AW9555 应用中会增加系统功耗，若输入高电平低于 Vcc，则 AW9535/AW9555 芯片功耗会增加(AW9555 每路会最大增加 15μA 的功耗，AW9535 每路会最大增加 5μA 的功耗)。
- 当使用 AW9555 的 IO 口用于驱动 LED 灯时，若 LED 的阳极电压与 Vcc 相差较大时，如 LED 阳极电压为 5V，AW9555 的 Vcc 电压为 1.65V，则上电后，LED 可能会出现微亮的现象（I/O 口端自带的弱上拉电阻导致），在应用时，应合理选择 AW9555 的 Vcc 电压与 LED 的阳极电压，推荐 LED 的阳极电压与芯片的 Vcc 电压一致。
- AW9535 未使用的 GPIO 口，必须通过适当阻值的电阻器（推荐 10 kΩ）将其连接到 Vcc 或 GND，避免 GPIO 口引脚悬空而受到干扰产生误中断，而且悬空的 I/O 口会造成芯片的功耗增加；AW9555 未使用的 GPIO 口，可直接悬空处理。
- 每个 I/O 灌电流必须外部限制为最大 25mA，每组 IO 口（P07~P00 和 P17~P10）必须限制为最大 100mA 电流，芯片总电流为 200mA。
- 所有 I/O 的总的输出电流能力限制在 160mA（P07~P00 为 80mA，P17~P10 为 80mA）。
- INTN 端口是开漏输出，需要电阻 R3 上拉至 Vcc，推荐使用 10kΩ，具体阻值可根据调试进行选择；与 INTN 端口相连的 MCU 的 IO 口，需要支持唤醒功能。
- I2C 端口是开漏类型，需要上拉电阻 R1、R2 至 Vcc 才能够正常通信，在应用时，SCL、SDA 信号的上拉电平不能低于 Vcc，建议接上拉电阻至 Vcc；当输入高电平低于 Vcc 时，芯片的功耗会增加（每路会最大增加 5μA 的功耗）；当 I2C 通信速率为是 400kHz 时，默认上拉电阻 R1 与 R2 取值为 2.2kΩ，若主机端已有上拉电阻，则 SCL/SDA 端的外部上拉电阻可省略，R1/R2 具体阻值应根据以下推荐的外部上拉电阻 RP 计算方式进行选择；

  外部上拉电阻 RP 的计算方式如下：

  RP(min)是 Vcc，VOL(max)与 IOL 的函数：$R_P(\min)=\frac{V_{CC}-V_{OL}(max)}{I_{OL}}$

  RP(max)是最大上升时间 tr（针对快速模式，tr=300ns，fSCL=400kHz）与总线电容 Cb 的函数：$R_P(\max)=\frac{t_r}{0.8473\times C_b}$

其中，参数 VOL，IOL，tr 与总线电容 Cb 可参考下表中的规范所示。总线电容 Cb 可以通过将 AW9535/AW9555 上增加的电容，SCL 引脚电容 Ci 或者 SDA 引脚电容 Cio，走线与连接处等的寄生电容以及同一总线上其他从设备的电容相加来近似评估，最终根据功耗与速度的要求选择出合适的上拉电阻。对于 I2C 总线的标准模式或快速模式，最大总线电容 Cb 均不得超过 400pF。

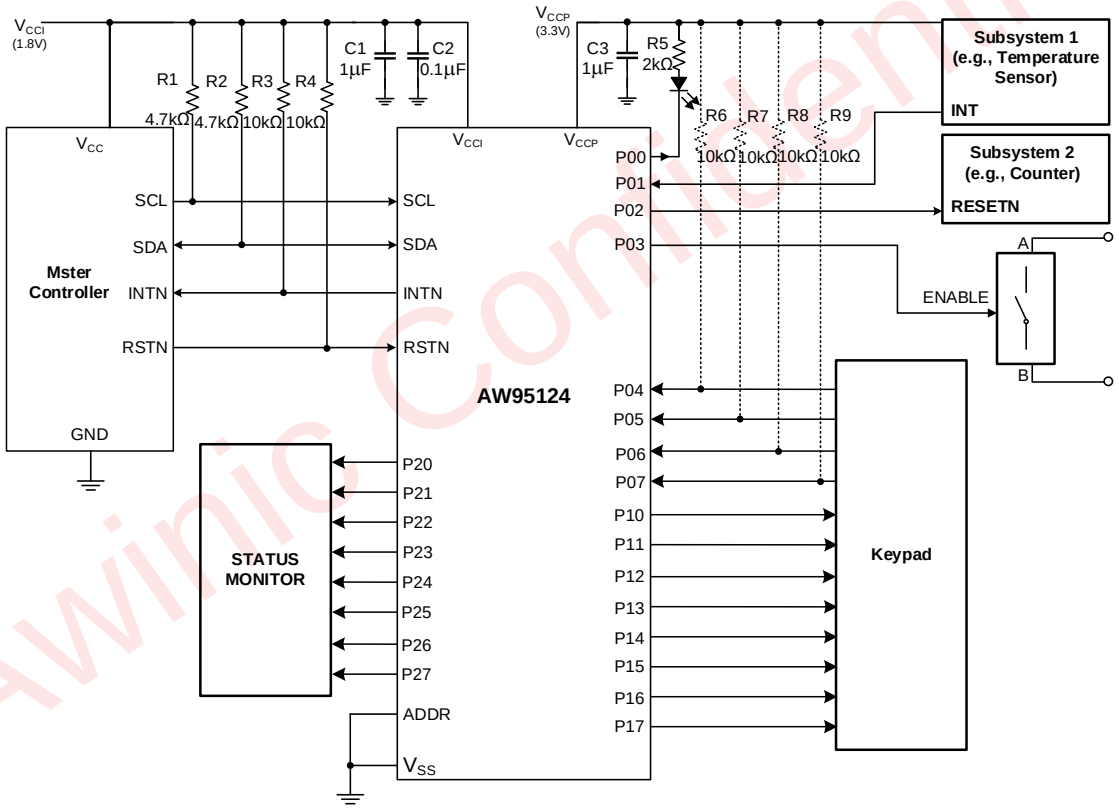
参数			标准模式（Max）	快速模式（Max）	单位
t_r	SDA 和 SCL 信号的上升时间		1000	300	ns
C_b	每条总线的容性负载		400	400	pF
V_{OL}	低电平输出电压(在3mA灌电流，Vcc >2V时)		0.4	0.4	V
	低电平输出电压(在2mA灌电流，Vcc≤2V时)		-	0.2*Vcc	V

- AW9535/AW9555 的 I2C 地址可通过 A0~A2 进行拓展，A0~A2 引脚可直接接 GND 或者高电平，需要接高电平时，建议直接接 Vcc，当输入高电平低于 Vcc 时，AW9535/AW9555 会最大增加约 5μA 的功耗；AW9535/AW9555 的 7 位 I2C 地址可设置为 0x20~0x27，同一路 I2C 总线上最多可支持 8 个 AW9535/AW9555 器件，AW9535/AW9555 的 I2C 地址选择详见下表：

固定位	A2	A1	A0	I2C 地址
0100	0	0	0	0x20
	0	0	1	0x21
	0	1	0	0x22
	0	1	1	0x23
	1	0	0	0x24
	1	0	1	0x25
	1	1	0	0x26
	1	1	1	0x27

4.2.3 AW95124FOR原理图介绍

- AW95124 作为 I2C 从设备通常位于主机的远程位置，靠近主机需要监控的 GPIO 端，图 1 为 AW95124 的典型应用电路，AW95124 作为 IO 扩展器通常用于控制 LED（用于反馈或状态灯）、控制其他设备的启用或重置信号，甚至读取其他设备或按键的输出状态等。



(1) In this application schematic, P00, P02, P03 andP10~P27 are configured as outputs.
(2) P01 and P04~P07 are configured as inputs.
(3) Device address is configured as 0100010 for this example.

图 4.12 AW95124 典型应用电路

- AW95124 的电压范围为 VCCI：1.08V～3.6V，VCCP：1.08V～3.6V，电源输入端的并联电容推荐使用 1μF 和 0.1μF，当电源电压 VCCI 不大于 3.3V 时，C1~C4 选用额定电压为不小于 6.3V 的电容；当电源大于 3.3V 时，C1~C4 选用额定电压为不小于 10V 的电容。
- AW95124 芯片的 GPIO 口配置为输入状态时，可通过寄存器配置内部的上下拉功能（芯片内部的上拉电阻为 100kΩ，内部的下拉电阻为 100kΩ）；输入的高电平需不小于 Vccp， AW95124 应用中不会增加系统功耗，若输入高电平低于 VCCP，则 AW95124 芯片功耗会增加(AW95124 每路会增加功耗)。
- AW95124 芯片的 IO 口配置为输出状态时，默认是推挽输出模式，当 GPIO 口配置成推挽输出模式时，对应的 GPIO 口不要与其他电压相连，若外加的电压大于 VCCP，则会出现往芯片内部倒灌电流的现象，若外加的电压低于 VCCP 或若在 IO 外部拉低时，则会进一步增加系统功耗。
- 当 GPIO 口配置为开漏输出时，外加的电压与 VCCP 电压大小无关，高于或者低于 VCCP 都不会出现倒灌电流和芯片的整体功耗增加问题(外加的电压不能高于 3.6V)，注意端口的上拉电压与相连芯片的 VIL/VIH 匹配；
- 当 AW95124 的 IO 口为输出状态时，内部的上下拉电阻配置无法使用，在应用时需注意；
- AW95124 芯片的 GPIO 口作为输入端口，默认输入为高阻态，不需要的管脚推荐接 VCCP 或 GND（需与固定电平相连，否则容易受外部干扰造成状态的变化，可以选择芯片内部上下拉电阻配置或预留外部电阻连接至 VCCP 或 GND），但选择内部上下拉电阻配置时，芯片上电默认状态为输入悬空态，应用时需注意；
- 每个 I/O 灌电流必须外部限制为最大 25mA，每组 IO 口（P07~P00、P17~P10 和 P27~P20）必须限制为最大 100mA 电流，芯片总电流为 200mA。
- 每个 I/O 口的输出电流限制为 10mA，每组 IO 口 (P07~P00、P17~P10 和 P27~P20) 必须限制为最大 80mA 电流，芯片总电流为 160mA。
- I2C 端口是开漏类型，需要上拉电阻 R1、R2 至 VCCI 才能够正常通信，在应用时，SCL、SDA 信号的上拉电平不能低于 VCCI，建议接上拉电阻至 VCCI；当输入高电平低于 VCCI 时，芯片的功耗会增加（每路会增加额外的功耗）且需注意通讯电平是否满足对应端口的 VIH/VIL 要求；

- 当 I2C 通信速率为是 400kHz 时，默认上拉电阻 R1 与 R2 取值为 2.2kΩ，若主机端已有上拉电阻，则 SCL/SDA 端的外部上拉电阻可省略，R1/R2 具体阻值应根据以下推荐的外部上拉电阻 RP 计算方式进行选择；当 I2C 通信速率为是 1000kHz 时，默认上拉电阻 R1 与 R2 取值为 1kΩ，若主机端已有上拉电阻，则 SCL/SDA 端的外部上拉电阻可省略，R1/R2 具体阻值应根据以下推荐的外部上拉电阻 RP 计算方式进行选择；

  外部上拉电阻 RP 的计算方式如下：

  RP(min)是 VCCI，VOL(max)与 IOL 的函数：$R_P(\min)=\frac{V_{CCI}-V_{OL}(max)}{I_{OL}}$；

  RP(max)是最大上升时间 tr（针对快速模式，tr=300ns，fSCL=400kHz；针对快速+模式，tr=120ns，fSCL=1000kHz）与总线电容 Cb 的函数：$R_P(\max)=\frac{t_r}{0.8473\times C_b}$；

  其中，参数 VOL，IOL，tr 与总线电容 Cb 可参考下表中的规范所示。总线电容 Cb 可以通过将 AW95124 上增加的电容，SCL 引脚电容 Ci 或者 SDA 引脚电容 Cio，走线与连接处等的寄生电容以及同一总线上其他从设备的电容相加来近似评估，最终根据功耗与速度的要求选择出合适的上拉电阻。对于 I2C 总线的标准模式或快速模式，最大总线电容 Cb 均不得超过 400pF。对于 I2C 总线的快速+模式，最大总线电容 Cb 均不得超过 550pF。

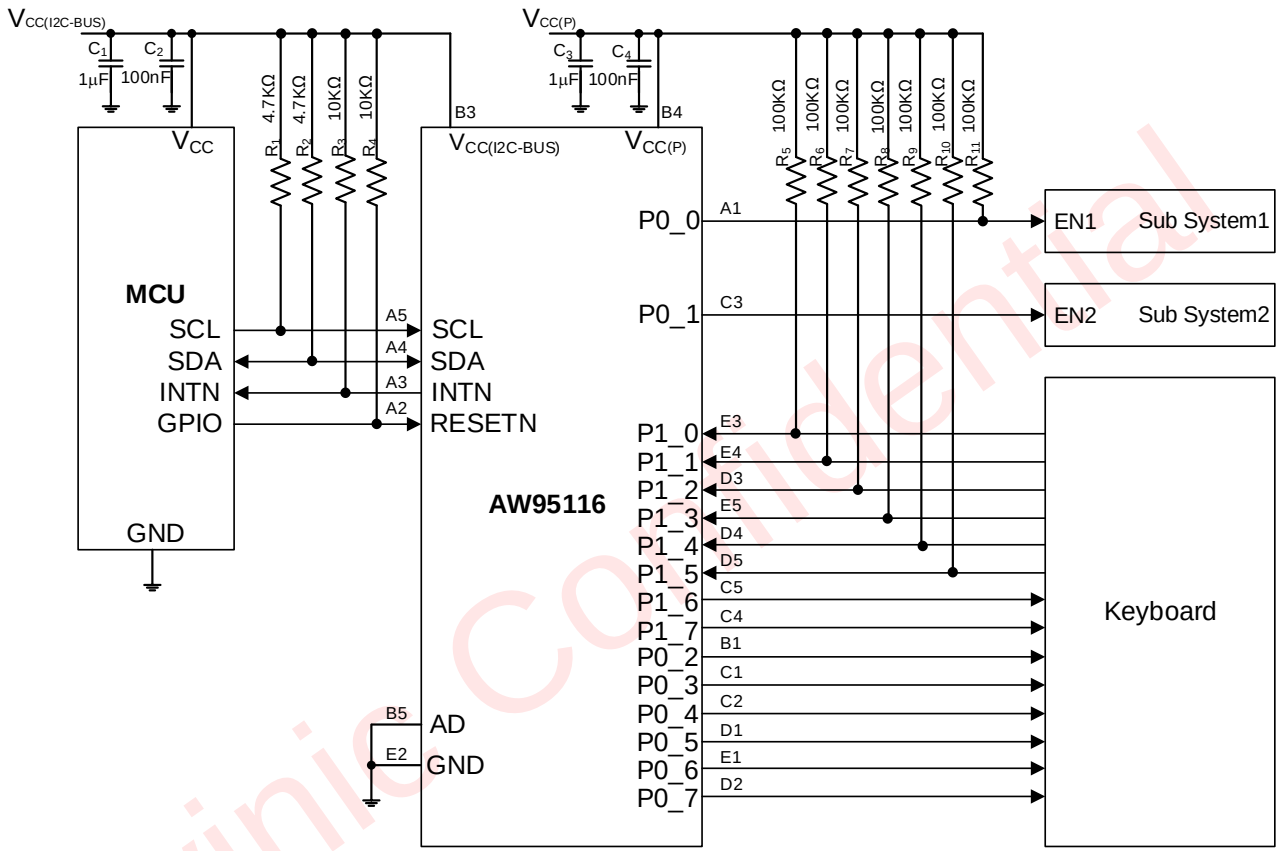
参数		标准模式		快速模式		快速模式+		单位
		Min	Max	Min	Max	Min	Max	
t_r	SDA 和 SCL 信号的上升时间	-	1000	20	300	-	120	ns
C_b	每条总线的容性负载	-	400	-	400	-	550	pF
V_{OL}	低电平输出电压(在3mA灌电流，V_{CCI} >2V时)	0	0.4	0	0.4	0	0.4	V
	低电平输出电压(在2mA灌电流，V_{CCI}≤2V时)	0	-	0	0.2*V_{CCI}	0	0.2*V_{CCI}	V
I_{OL}	低电平输出电流	3	-	3	-	20	-	mA

- INTN 端口是开漏输出，需要电阻 R3 上拉至 VCCI，推荐使用 10kΩ，具体阻值可根据调试进行选择；与 INTN 端口相连的 MCU 的 IO 口，需要支持唤醒功能。
- 当 IO 口只用做输出模式时，中断 INTN 可以 NC 处理，或者将 INTN 引脚固定上拉。
- RSTN 端口是输入引脚，需要电阻 R4 上拉至 VCCI，推荐使用 10kΩ，具体阻值可根据调试进行选择；与 RSTN 端口相连的 MCU 的 IO 口为 OD 输出引脚。上拉的电源建议直接接 VCCI，当输入高电平低于 VCCI 时，AW95124 芯片会增加功耗且且需注意上拉电平是否满足对应端口的 VIH/VIL 要求；
- AW95124 的 I2C 地址可通过 ADDR 进行拓展，ADDR 引脚可直接接 GND、SCL、SDA、高电平，需要接高电平时，建议直接接 VCCI，当输入高电平低于 VCCI 时，AW95124 会增加功耗且需注意上拉电平是否满足对应端口的 VIH/VIL 要求；AW95124 的 7 位 I2C 地址可设置为 0x20~0x23，同一路 I2C 总线上最多可支持 4 个 AW95124 器件，AW95124 的 I2C 地址选择详见下表：

固定位	ADDR	I2C 地址
0100	SCL	0x20
	SDA	0x21
	GND	0x22
	V_{CCI}	0x23

4.2.4 AW95116FOR原理图介绍

- AW95116 作为 I2C 从设备通常位于主机的远程位置，靠近主机需要监控的 GPIO 端，图 1 为 AW95116 的典型应用电路，AW95116 作为 IO 扩展器通常用于控制 LED（用于反馈或状态灯）、控制其他设备的启用或重置信号，甚至读取其他设备或按键的输出状态等。



(1) In this application schematic, P00, P01, P16, P17andP02~P07 are configured as outputs.
(2) P10~P15 are configured as inputs.
(3) Device address is configured as 0100000 for this example.

图 4.13 AW95116 典型应用电路

- AW95116 的电压范围为 VCCI：1.08V～3.6V，VCCP：1.08V～3.6V，电源输入端的并联电容推荐使用 1μF 和 0.1μF，当电源电压 VCCI 不大于 3.3V 时，C1~C4 选用额定电压为不小于 6.3V 的电容；当电源大于 3.3V 时，C1~C4 选用额定电压为不小于 10V 的电容。
- AW95116 芯片的 GPIO 口配置为输入状态时，可通过寄存器配置内部的上下拉功能（芯片内部的上拉电阻为 100kΩ，内部的下拉电阻为 100kΩ）；输入的高电平需不小于 Vccp， AW95116 应用中不会增加系统功耗，若输入高电平低于 VCCP，则 AW95116 芯片功耗会增

加(AW95116 每路会增加功耗)。

- AW95116 芯片的 IO 口配置为输出状态时，默认是推挽输出模式，当 GPIO 口配置成推挽输出模式时，对应的 GPIO 口不要与其他电压相连，若外加的电压大于 VCCP，则会出现往芯片内部倒灌电流的现象，若外加的电压低于 VCCP 或若在 IO 外部拉低时，则会进一步增加系统功耗。
- 当 GPIO 口配置为开漏输出时，外加的电压与 VCCP 电压大小无关，高于或者低于 VCCP 都不会出现倒灌电流和芯片的整体功耗增加问题(外加的电压不能高于 3.6V)，注意端口的上拉电压与相连芯片的 VIL/VIH 匹配；
- 当 AW95116 的 IO 口为输出状态时，内部的上下拉电阻配置无法使用，在应用时需注意；
- AW95116 芯片的 GPIO 口作为输入端口，默认输入为高阻态，不需要的管脚推荐接 VCCP 或 GND（需与固定电平相连，否则容易受外部干扰造成状态的变化，可以选择芯片内部上下拉电阻配置或预留外部电阻连接至 VCCP 或 GND），但选择内部上下拉电阻配置时，芯片上电默认状态为输入悬空态，应用时需注意；
- 每个 I/O 灌电流必须外部限制为最大 25mA，每组 IO 口（P07~P00、P17~P10）必须限制为最大 100mA 电流，芯片总电流为 200mA。
- 每个 I/O 口的输出电流限制为 10mA，每组 IO 口（P07~P00、P17~P10）必须限制为最大 80mA 电流，芯片总电流为 160mA。
- I2C 端口是开漏类型，需要上拉电阻 R1、R2 至 VCCI 才能够正常通信，在应用时，SCL、SDA 信号的上拉电平不能低于 VCCI，建议接上拉电阻至 VCCI；当输入高电平低于 VCCI 时，芯片的功耗会增加（每路会增加额外的功耗）且需注意通讯电平是否满足对应端口的 VIH/VIL 要求；
- 当 I2C 通信速率为是 400kHz 时，默认上拉电阻 R1 与 R2 取值为 2.2kΩ，若主机端已有上拉电阻，则 SCL/SDA 端的外部上拉电阻可省略，R1/R2 具体阻值应根据以下推荐的外部上拉电阻 RP 计算方式进行选择；当 I2C 通信速率为是 1000kHz 时，默认上拉电阻 R1 与 R2 取值为 1kΩ，若主机端已有上拉电阻，则 SCL/SDA 端的外部上拉电阻可省略，R1/R2 具体阻值应根据以下推荐的外部上拉电阻 RP 计算方式进行选择；

外部上拉电阻 RP 的计算方式如下：

RP(min)是 VCCI，VOL(max)与 IOL 的函数：$R_P(\min) = \frac{V_{CCI} - V_{OL}(max)}{I_{OL}}$；

RP(max)是最大上升时间 tr（针对快速模式，tr=300ns，fSCL=400kHz；针对快速+模式，tr=120ns，fSCL=1000kHz）与总线电容 Cb 的函数：$R_P(\max) = \frac{t_r}{0.8473 \times C_b}$；

其中，参数 VOL，IOL，tr 与总线电容 Cb 可参考下表中的规范所示。总线电容 Cb 可以通过将 AW95116 上增加的电容，SCL 引脚电容 Ci 或者 SDA 引脚电容 Cio，走线与连接处等的寄生电容以及同一总线上其他从设备的电容相加来近似评估，最终根据功耗与速度的要求选择出合适的上拉电阻。对于 I2C 总线的标准模式或快速模式，最大总线电容 Cb 均不得超过 400pF。对于 I2C 总线的快速+模式，最大总线电容 Cb 均不得超过 550pF。

参数		标准模式		快速模式		快速模式+		单位
		Min	Max	Min	Max	Min	Max	
t_r	SDA 和 SCL 信号的上升时间	-	1000	20	300	-	120	ns
C_b	每条总线的容性负载	-	400	-	400	-	550	pF
V_{OL}	低电平输出电压(在3mA灌电流，V_{CCI} >2V时)	0	0.4	0	0.4	0	0.4	V
	低电平输出电压(在2mA灌电流，V_{CCI}≤2V时)	0	-	0	0.2*V_{CCI}	0	0.2*V_{CCI}	V
I_{OL}	低电平输出电流	3	-	3	-	20	-	mA

- INTN 端口是开漏输出，需要电阻 R3 上拉至 VCCI，推荐使用 10kΩ，具体阻值可根据调试进行选择；与 INTN 端口相连的 MCU 的 IO 口，需要支持唤醒功能。
- 当 IO 口只用做输出模式时，中断 INTN 可以 NC 处理，或者将 INTN 引脚固定上拉。

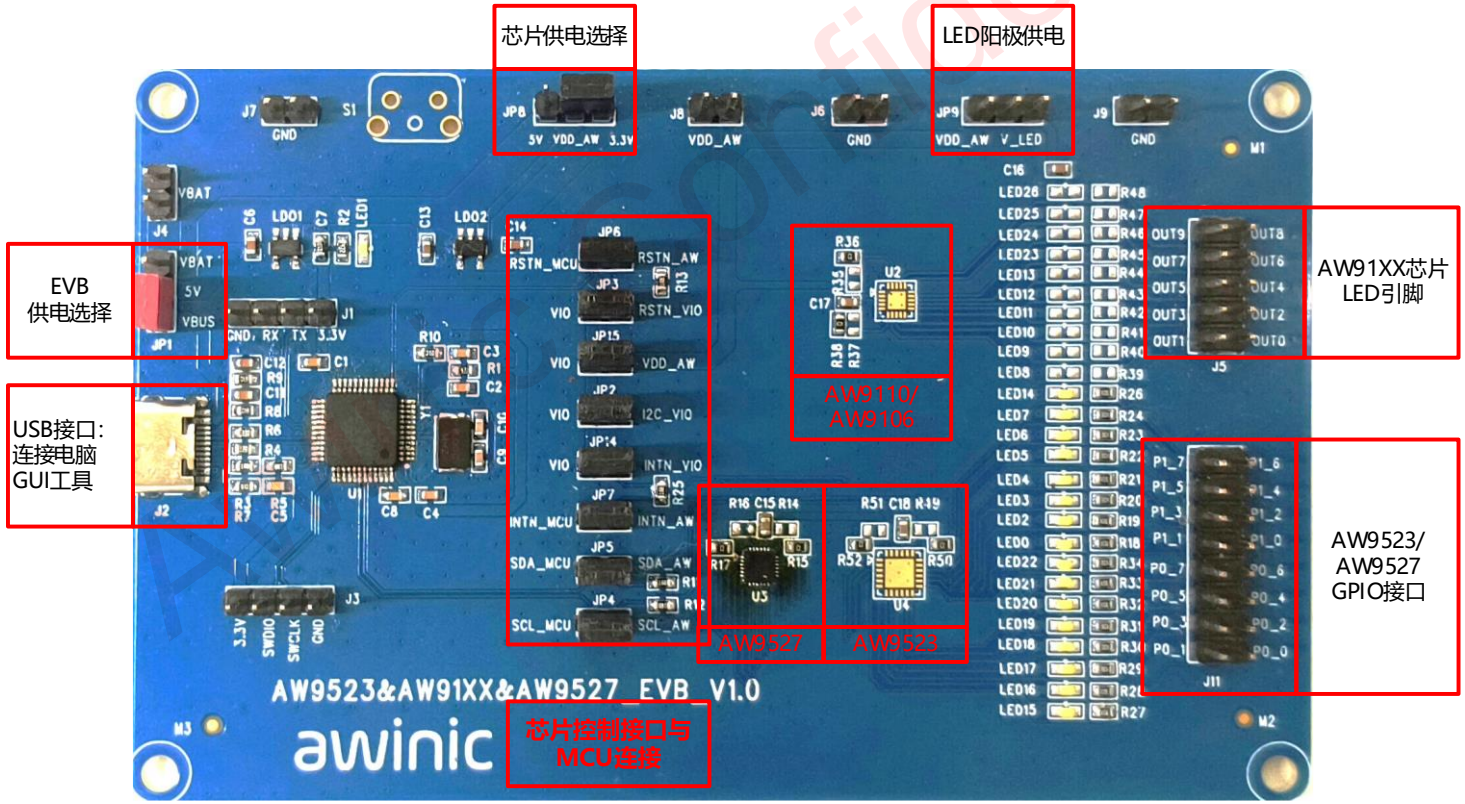
- RSTN 端口是输入引脚，需要电阻 R4 上拉至 VCCI，推荐使用 10kΩ，具体阻值可根据调试进行选择；与 RSTN 端口相连的 MCU 的 IO 口为 OD 输出引脚。上拉的电源建议直接接 VCCI，当输入高电平低于 VCCI 时，AW95116 芯片会增加功耗且且需注意上拉电平是否满足对应端口的 VIH/VIL 要求；
- AW95116 的 I2C 地址可通过 ADDR 进行拓展，ADDR 引脚可直接接 GND、高电平，需要接高电平时，建议直接接 VCCI，当输入高电平低于 VCCI 时，AW95116 会增加功耗且需注意上拉电平是否满足对应端口的 VIH/VIL 要求； AW95116 的 7 位 I2C 地址可设置为 0x20~0x21，同一路 I2C 总线上最多可支持 2 个 AW95116 器件，AW95116 的 I2C 地址选择详见下表：

固定位	ADDR	I2C 地址
0100	GND	0x20
	V_{CCI}	0x21

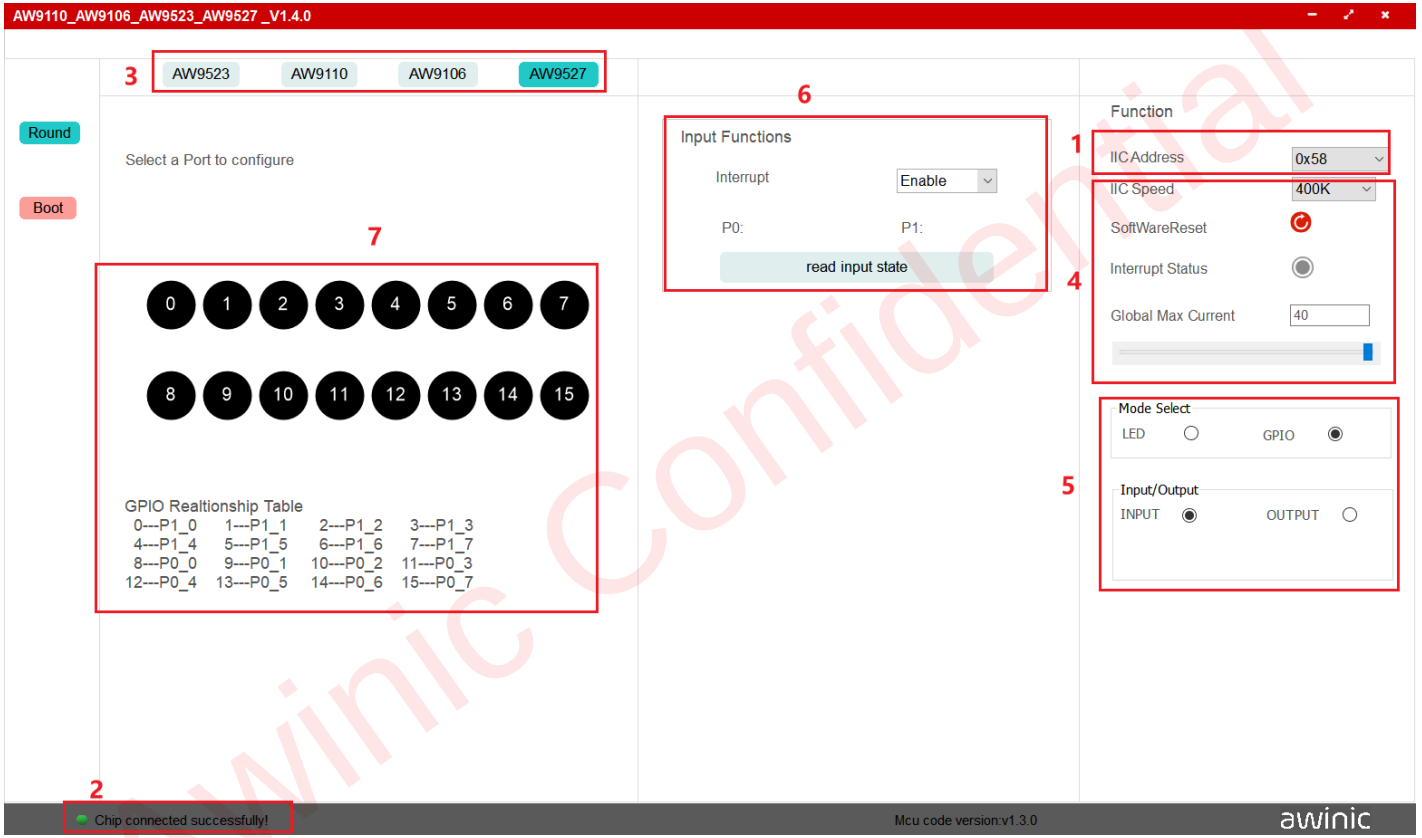
5 EVB 及 GUI 使用说明

5.1 AW9523/AW9110C/AW9106C/AW9527 EVB 硬件介绍

5.1.1 AW9523_AW91XX_AW9527_EVB_V1.0 EVB硬件介绍



5.1.2 AW9523_AW91XX_AW9527_EVB_V1.0 UI工具



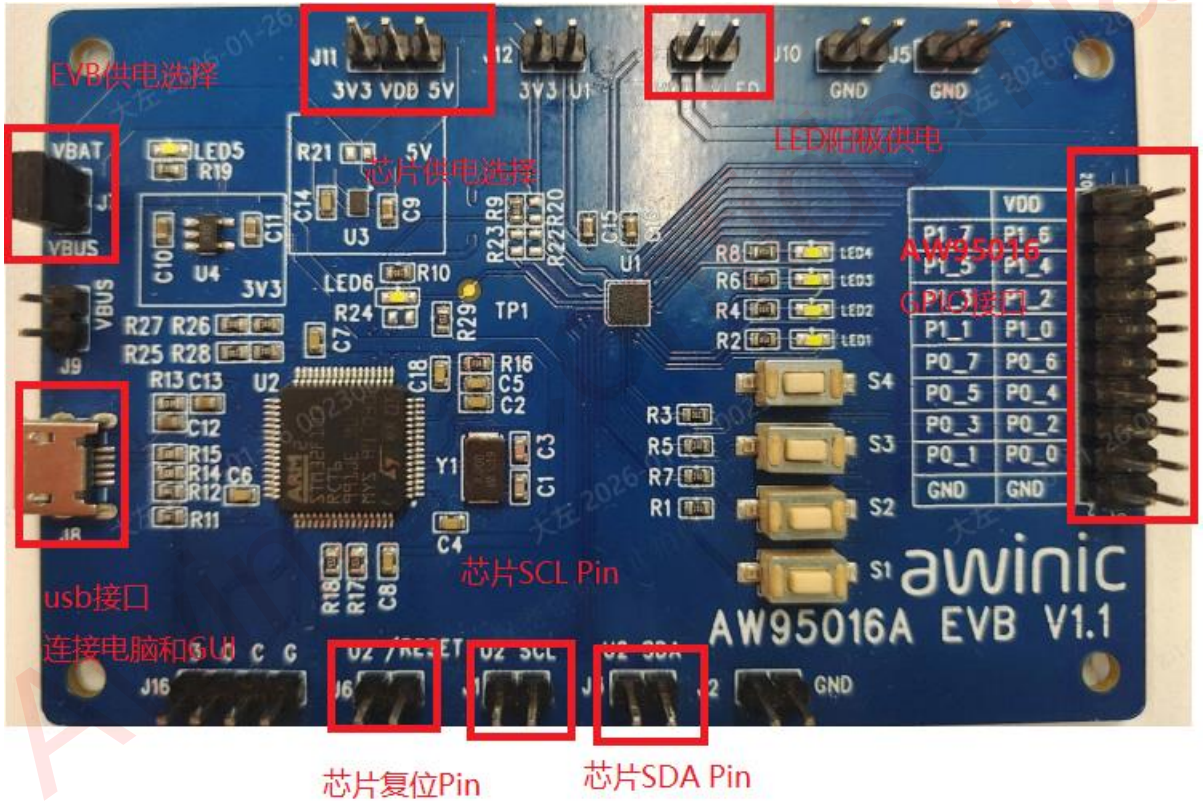
- 芯片地址选择，可根据外部电阻更改地址，按照板上实际电阻选择对应地址即可；
- 连接状态，插入 USB 线后，显示 Chip connected successfully!即为连接成功；
- demo 板上实际贴片的芯片，插入 USB 连接 UI 后自动选择；
- 芯片软复位以及 LED 模式下全局电流调节；

- 芯片模式选择，可选择 LED 或者 GPIO 模式，以及 GPIO 模式下 IO 口输入输出模式选择；
- 在 5 处选择不同模式后，显示对应的调试界面。上图为默认输出模式时界面，可选择 open drain 或者 push pull 输出以及输出电平；
- 对对应的 IO 口进行配置，数字对应芯片引脚如图所示。

更多细节请参考 [AW9110CQNR/AW9106CQNR/AW9523BTQR EVB 文档]

5.2 AW95016 EVB 硬件介绍

5.2.1 AW95016 EVB硬件介绍

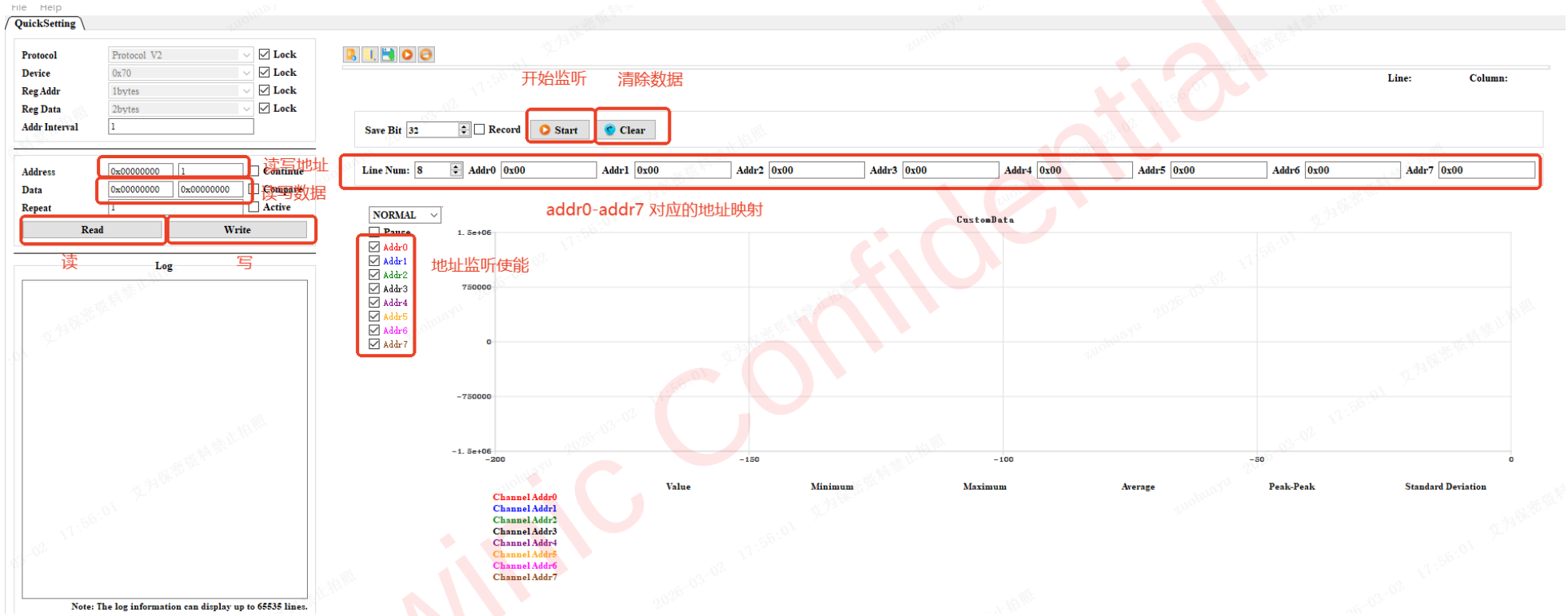


5.2.2 AW95016 EVB硬件 UI工具

5.3 AW9535/AW9555 EVB 硬件介绍

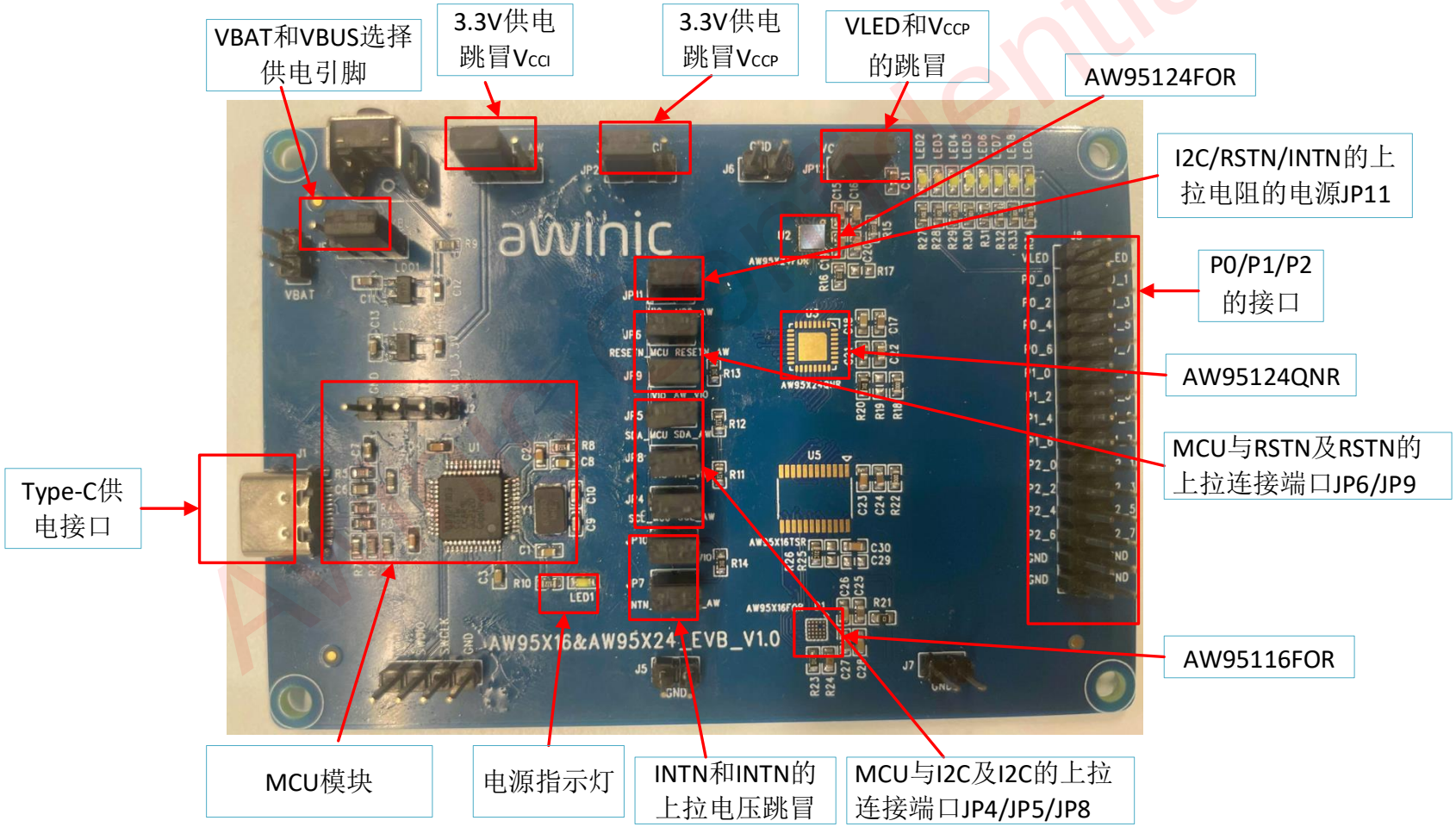
5.3.1 AW9535/AW9555 EVB硬件介绍

5.3.2 AW9535/AW9555 EVB UI工具

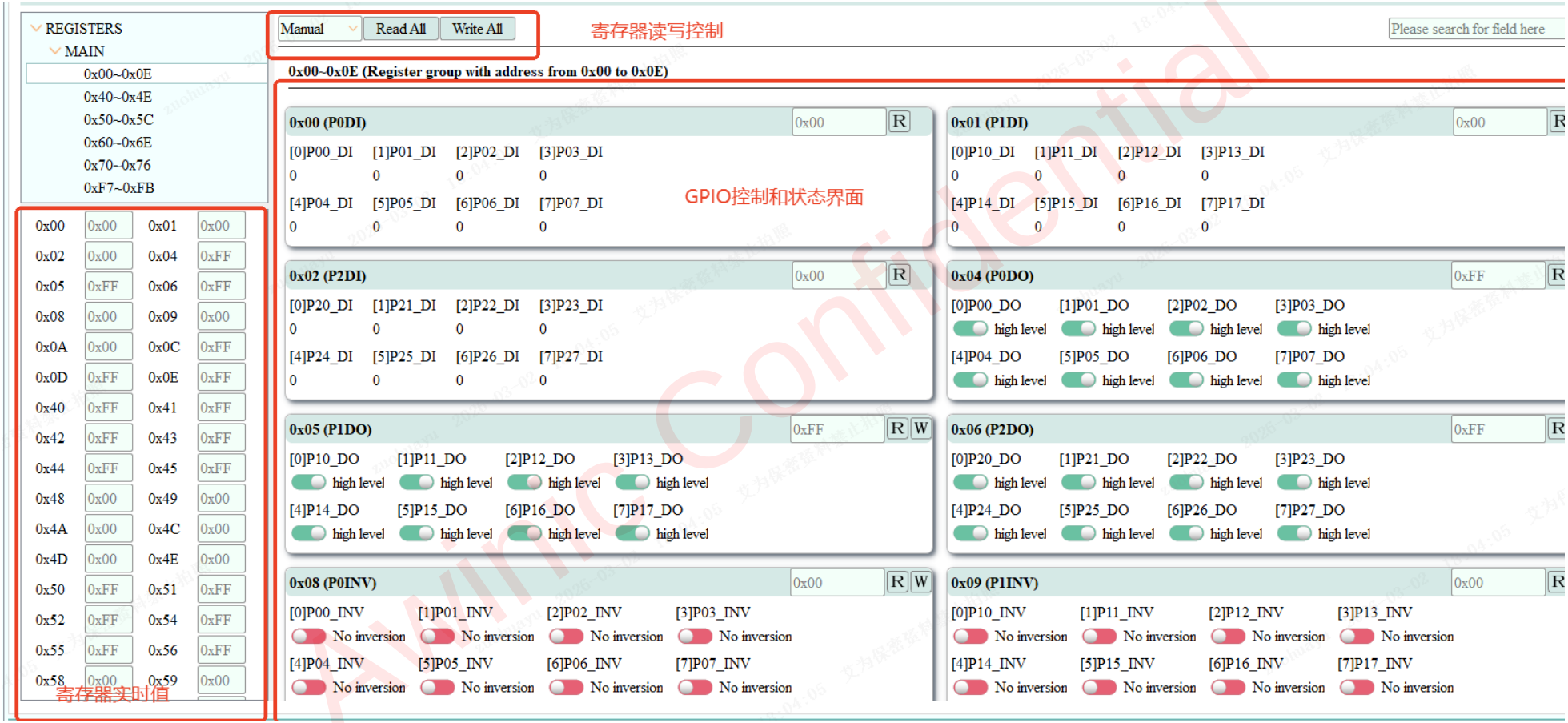


5.4 AW95116/AW95124 EVB 硬件介绍

5.4.1 AW95116/AW95124 EVB硬件介绍



5.4.2 AW95116/AW95124 EVB UI工具



6 驱动设计说明

6.1 驱动移植介绍

艾为 GPIO 扩展芯片目前支持 input 框架和 gpiochip 架构两种框架，input 架构是将 gpio 作为一个外设来使用，所有使用 io 设备都需要在 gpio 驱动这边统一管理和控制，gpiochip 架构则是将艾为 gpio 扩展芯片的所有 io 注册为平台的的 io，各个设备可以在自己的驱动里面申请和调用，方式跟调用平台 io 一样。以下是对两种驱动的移植方式和接口介绍。

6.1.1 input架构驱动移植指南

- DTSI 移植

```
&i2c_3 {
        status = "ok";
        aw951xx_ex@22 {
                compatible = "awinic,aw951xx";
                reg = <0x22>;
                reset-gpio = <&tlmm 20 0>; //rst gpio
                irq-gpio = <&tlmm 72 0>; //irq gpio
                status = "okay";
                aw951xx,single_key_enable = <0>; //enable single key function, 0 to disable
                aw951xx,matrix_key_enable = <1>; //enable matrix key function, 0 to disable
                aw951xx,gpio_enable = <1>; //enable gpio function, 0 to disable
                aw951xx,key{
```

```
        aw951xx,wake_up_enable = <1>; //enabel key function in suspend mode,0 to disable
        aw951xx,input_port_mask = <0x0000FF>; // 0000 0000 1111 1111 Identifies the pin port for the input. here: P0_0-P0_7
        aw951xx,output_port_mask = <0x00FF00>; //1111 1111 0000 0000 Identifies the pin port for the output here: P1_0-P1_7
    };
    aw951xx,gpio{
        aw951xx,gpio_mode = <0>;
        gpio16{
            aw951xx,gpio_idx = <16>; // The specific port identifier used, This is used here:P2_0
            aw951xx,gpio_dir = <0>; // The specific port work in output(0) or input(1)
            aw951xx,gpio_default_val = <0>;
        };
        gpio17{
            aw951xx,gpio_idx = <17>; // The specific port identifier used, This is used here:P2_1
            aw951xx,gpio_dir = <0>; //The specific port work in output(0) or input(1)
            aw951xx,gpio_default_val = <1>;
        };
        gpio18{
            aw951xx,gpio_idx = <18>; // The specific port identifier used, This is used here:P2_2
            aw951xx,gpio_dir = <0>; //The specific port work in output(0) or input(1)
            aw951xx,gpio_default_val = <1>;
        };
        gpio19{
            aw951xx,gpio_idx = <19>; // The specific port identifier used, This is used here:P2_3
            aw951xx,gpio_dir = <0>; //The specific port work in output(0) or input(1)
            aw951xx,gpio_default_val = <1>;
        };
        gpio20{
            aw951xx,gpio_idx = <20>; // The specific port identifier used, This is used here:P2_4
            aw951xx,gpio_dir = <1>; // The specific port work in output(0) or input(1)
            aw951xx,gpio_default_val = <0>;
```

```
                };
                gpio21{
                        aw951xx,gpio_idx = <21>; // The specific port identifier used, This is used here:P2_5
                        aw951xx,gpio_dir = <1>; //The specific port work in output(0) or input(1)
                        aw951xx,gpio_default_val = <0>;
                };
                gpio22{
                        aw951xx,gpio_idx = <22>; // The specific port identifier used, This is used here:P2_6
                        aw951xx,gpio_dir = <1>; //The specific port work in output(0) or input(1)
                        aw951xx,gpio_default_val = <0>;
                };
                gpio23{
                        aw951xx,gpio_idx = <23>; // The specific port identifier used, This is used here:P2_7
                        aw951xx,gpio_dir = <1>; //The specific port work in output(0) or input(1)
                        aw951xx,gpio_default_val = <0>;
                };
        };
    };
};
```

变量	功能介绍	备注
reset-gpio	芯片使能引脚	连接平台的 gpio
Irq-gpio	芯片中断引脚	连接平台的 irq
aw951xx,single_key_enable	单按键功能使能	用于 io 口作为独立按键，最多支持 24 键

aw951xx,matrix_key_enable	矩阵按键功能使能	用于行列扫描键盘，最多支持 12*12=144 键
aw951xx,gpio_enable	gpio 功能使能	用于给连接扩展 ic 的 io 设备
aw951xx,wake_up_enable	中断唤醒系统	
aw951xx,input_port_mask	单按键引脚索引/矩阵按键输入索引索引	24bi bit0-bit23 代表 P0_0_P0_7， P1_0-P1_7, P2_0-P2_7
aw951xx,output_port_mask	矩阵按键输出 io 索引	24bit bit0-bit23 代表 P0_0_P0_7， P1_0-P1_7, P2_0-P2_7
aw951xx,gpio_mode	输出模式 0：推腕 1：开漏	
aw951xx,gpio_idx	gpio 模式 io 序号索引 范围 0-23	
aw951xx,gpio_dir	gpio 模式 io 的方向 0：输出 1：输入	
aw951xx,gpio_default_val	Gpio 模式 输出的默认电平状态 0：low 1:high	

- 驱动移植

➢ 将驱动包里面的 input 架构代码添加到/drivers/input/misc/aw951xx 目录下

➢ 在/arch/arm64/configs/xxx_defconfig 添加 CONFIG_AW951XX=y 编译选项

➢ 在/drivers/input/misc/Kconfig 添加 source "drivers/input/misc/aw951xx/Kconfig" 关联驱动的 Kconfig

➢ 在/drivers/input/misc/Makefile 添加 obj-$(CONFIG_GPIO_AW951XX) += aw951xx/

6.1.2 gpiochip架构驱动移植指南

- DTSI 移植

```
awgpio: gpio@22 {
        compatible = "awinic,aw95124";
        reg = <0x22>;
        interrupt-parent = <&tlmm>;
        interrupts = <48 IRQ_TYPE_EDGE_FALLING>;
        interrupt-controller;
        #interrupt-cells = <2>;
        reset-gpio = < &tlmm 8 0 >;
        vcc-supply = <&pm8953_l5>;
        gpio-controller;
        #gpio-cells = <2>;
        gpio-line-names = "hdmi-ct-hpd", "hdmi.ls-oe", "p02", "p03",
                          "vibra", "fault2", "p06", "p07", "en-usb",
                          "en-host1", "en-host2", "chg-int", "p14", "p15",
                          "mic-int", "en-modem", "shdn-hs-amp",
                          "chg-status+red", "green", "blue", "en-esata",
                "fault1", "p26", "p27";
    gpio_ex_info = [
        //REG   DATA
        5C 00
    ];
    status = "ok";

    awgpio_3: awgpio-3 {
        pins = "gpio3";
```

```
            drive-strength = <2>;
            bias-disable;
            input-enable;
        };
        awgpio_10_20_5: awgpio-10-20-5 {
            pins = "gpio10", "gpio20", "gpio5";
            drive-strength = <1>;
            bias-pull-up;
            drive-push-pull;
            output-high;
        };
        awgpio_8_9: awgpio-8-9 {
            pins = "gpio8", "gpio9";
            drive-strength = <3>;
            input-enable;
        };
        awgpio_15_2: awgpio-15-2 {
            pins = "gpio15", "gpio2";
            drive-open-drain;
            output-low;
        };
        };
```

dtsi 变量	功能介绍	备注
interrupt-parent	平台中断源	
interrupts	平台中断脚	
reset-gpio	复位引脚	
vcc-supply	芯片供电引脚	
gpio-controller;	Gpio 控制器使能	
interrupt-controller;	中断控制器使能	
gpio-line-names	gpio 别名	
gpio_ex_info	gpio 初始化寄存器配置	
pins	Gpio 编号	
driver-strength	驱动能力可配置 0-3 依次变强	
bias-disable/bias-disable	关闭上下拉/打开上下拉	
Input-enable/input-disable	配置为输入/配置为输出	
Driver-push-pull/Driver-open-drain	推挽输出/开漏输出	

Ouput-high/output-low	输出高/输出低	

- 驱动移植

➢ 将驱动包里面的 gpio 架构代码添加到/drivers/gpio/aw951xx 目录下

➢ 在/arch/arm64/configs/xxx_defconfig 添加 CONFIG_GPIO_AW951XX=y 编译选项

➢ 在/drivers/gpio/Kconfig 添加 source "drivers/gpio/aw951xx/Kconfig" 关联驱动的 Kconfig

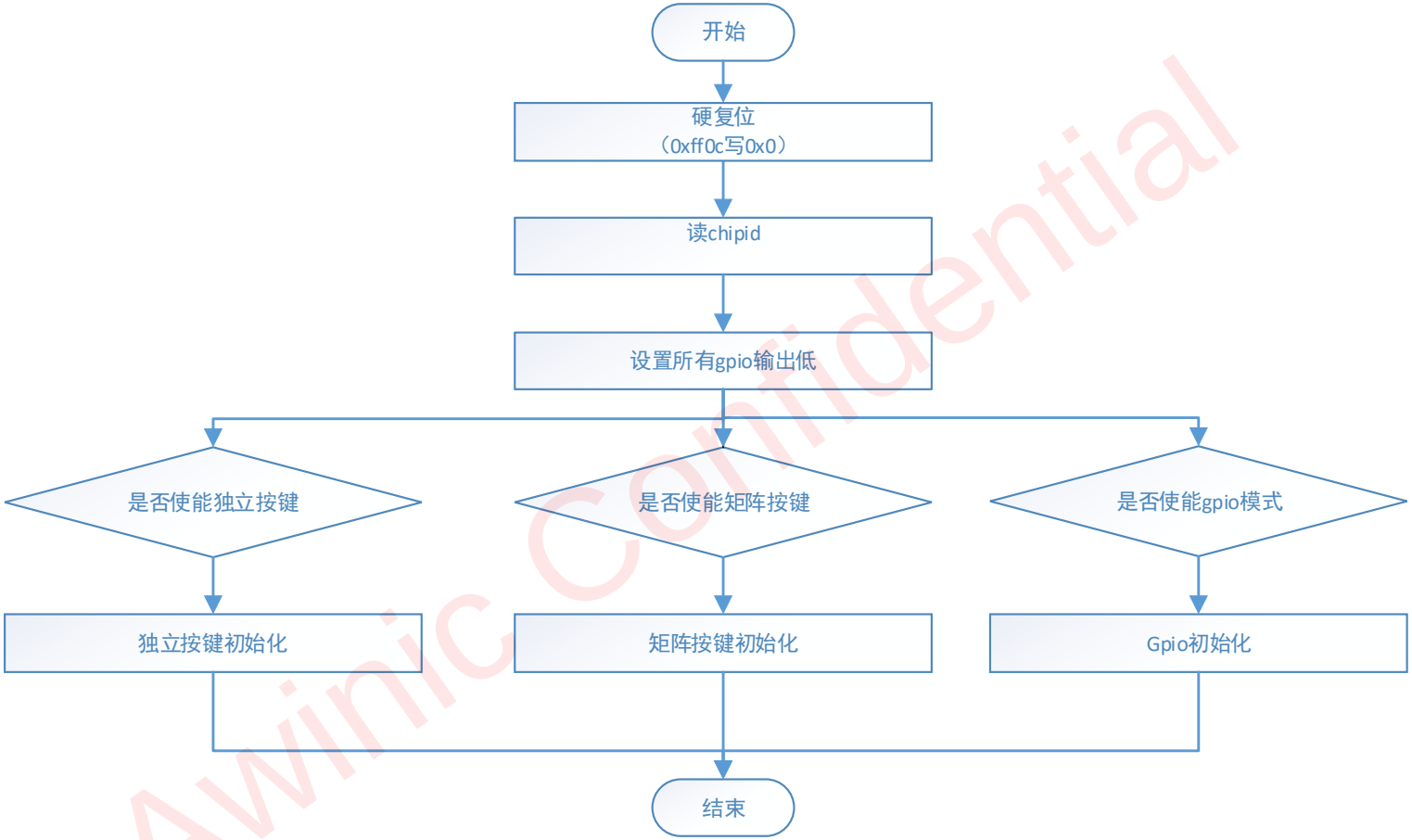
➢ 在/drivers/gpio/Makefile 添加 obj-$(CONFIG_GPIO_AW951XX) += aw951xx/

6.2 驱动功能介绍

6.2.1 input架构驱动功能介绍

- 驱动初始化流程

  驱动初始化流程如下图所示



- 独立按键功能

➢ 独立按键初始化

1） 将 dtsi 的 input_port_mask 对应的 io 设置为输入模式

2） 将 dtsi 的 input_port_mask 对应的中断使能打开

3） 使能平台中断

```
static void aw951xx_single_key_chip_init(struct aw951xx_key *p_key_data)
{
    unsigned int all_mask = 0;
    struct aw951xx *aw951xx = p_key_data->priv;
    disable_irq_nosync(p_key_data->priv->irq_num);
    aw951xx_enbale_interrupt_by_mask(aw951xx, AW951XX_INT_MASK, 0);
    all_mask = p_key_data->input_port_mask;
    p_key_data->old_input_state = p_key_data->input_port_mask;
    //aw951xx_set_port_mode_by_mask(aw951xx, all_mask, 1); /*AW951XX 默认就是 gpio 模式 */
    aw951xx_set_port_direction_by_mask(aw951xx, p_key_data->input_port_mask, 1); /* input mode */
    aw951xx_set_port_output_mode(aw951xx, 1); /* set output port pull push mode */

    aw951xx_enbale_interrupt_by_mask(aw951xx, p_key_data->input_port_mask, 1);/* enable input interrupt */
    /* clear inputerrupt */
    aw951xx_clear_interrupt(aw951xx);
    enable_irq(p_key_data->priv->irq_num);
}
```

➢ 独立按键键值处理

```
if (aw951xx->single_key_enable) {
        p_key_data->new_input_state = aw951xx_get_port_input_state(aw951xx);
        p_key_data->new_input_state &= p_key_data->input_port_mask;
        real_idx = 0;
        if (p_key_data->new_input_state != p_key_data->old_input_state) {
            for (i = 0; i < AW951XX_KEY_PORT_MAX; i++) {
                if (p_key_data->input_port_mask & (0x01 << i)) {
                    if ((p_key_data->new_input_state & (0x01 << i)) != (p_key_data->old_input_state & (0x01 << i))) {
                        key_code = p_key_data->keymap[real_idx].key_code;
                        key_val = (p_key_data->old_input_state & 0x01 << i) ? 1 : 0; /* press or release */

                        key_num = aw951xx_separate_key_data[real_idx];
                        /*if (key_val == 1) */
                        /*  AW_DEBUG("%s, key%d pressed\n", __func__, key_num); */
                        /*else */
                        /*  AW_DEBUG("%s, key%d release\n", __func__, key_num); */
                        /* AW_DEBUG("%s, key_code = 0x%x, key_val = 0x%x\r\n", __func__, key_code, key_val); */
                        input_report_key(p_key_data->input, key_code, key_val);
                        input_sync(p_key_data->input);
                    }
                    real_idx++;
                }
            }
        }
```

- 矩阵按键功能

➢ 矩阵按键初始化

1） 将 dtsi 的 input_port_mask 对应的 io 设置为输入模式

2）将 dtsi 的 input_port_mask 对应的 io 中断使能打开

3）将 dtsi 的 output_port_mask 对应的 io 设置为输出

4）将所有 io 设置为推挽模式

5）将 dtsi 的 output_port_mask 对应的 io 输出低电平

6）使能平台中断

```
static void aw951xx_key_chip_init(struct aw951xx_key *p_key_data)
{
    unsigned int all_mask = 0;
    struct aw951xx *aw951xx = p_key_data->priv;

    disable_irq_nosync(p_key_data->priv->irq_num);
    aw951xx_enbale_interrupt_by_mask(aw951xx, AW951XX_INT_MASK, 0); /* disale p0 p1 interrupt */
    all_mask = p_key_data->input_port_mask | p_key_data->output_port_mask;
    //aw951xx_set_port_mode_by_mask(aw951xx, all_mask, 1); /*AW951XX 默认就是 gpio 模式 */
    aw951xx_set_port_direction_by_mask(aw951xx, p_key_data->input_port_mask, 1); /* input mode */
    aw951xx_set_port_direction_by_mask(aw951xx, p_key_data->output_port_mask, 0); /* output mode */
    aw951xx_set_port_output_mode(aw951xx, 1); /* set output port pull push mode */

    aw951xx_set_port_output_state_by_mask(aw951xx, p_key_data->output_port_mask, 0);/* set output low */

    aw951xx_enbale_interrupt_by_mask(aw951xx, p_key_data->input_port_mask, 1); /* enable input interrupt */
    /* clear inputerrupt */
    aw951xx_clear_interrupt(aw951xx);
    enable_irq(p_key_data->priv->irq_num);
}
```

➢ 矩阵按键中断处理上报

1) 读 OEN 方向寄存器，将 dtis 里定义的 output_port_mask 每个 io 口依次设置为输出（其它 io 设置为输入），

2) 每次切换时读输入寄存器，保存每次切换时输入 io 口的电平状态值

3) 比较每次切换时的输入状态，如果前后状态不一致，记录对应行列（输出 io 索引和输入 io 索引），算出对应的键值才上报

4) 开机定时器轮询 1-3 步，如果所有 io 恢复成高电平，结束轮询。（用于检测多个按键不松手依次按下）

```
if (aw951xx->matrix_key_enable) {
        for (i = 0, real_idx = 0; i < AW951XX_KEY_PORT_MAX; i++) {
            if (p_key_data->output_port_mask & (0x01 << i)) {
                i2c_read_multi_byte(aw951xx->client, AW951XX_P0OEN_REG, reg_val, ARRAY_SIZE(reg_val));

                input_val = reg_val[0] | (reg_val[1] << 8) | (reg_val[2] << 16);
                input_val |= p_key_data->output_port_mask;
                input_val &= ~(0x01 << i);

                reg_val[0] = input_val & 0xff;
                reg_val[1] = (input_val >> 8) & 0xff;
                reg_val[2] = (input_val >> 16) & 0xff;

                i2c_write_multi_byte(aw951xx->client, AW951XX_P0OEN_REG, reg_val, ARRAY_SIZE(reg_val));
                i2c_read_multi_byte(aw951xx->client, AW951XX_P0DI_REG, reg_val, ARRAY_SIZE(reg_val));
                new_state[real_idx] = (reg_val[0] | (reg_val[1] << 8) | (reg_val[2] << 16)) & p_key_data->input_port_mask;
                real_idx++;
            }
        }
```

```
        /* key state change */
        if (memcmp(&new_state[0], &old_state[0], p_key_data->output_port_nums * sizeof(unsigned int))) {
            /* stage changed */
            for (i = 0, real_col = 0; i < AW951XX_KEY_PORT_MAX; i++) {
                if (p_key_data->output_port_mask & (0x01 << i)) {
                    if (new_state[real_col] != old_state[real_col]) {
                        for (j = 0, real_row = 0; j < AW951XX_KEY_PORT_MAX; j++) {
                            if (p_key_data->input_port_mask & (0x01 << j)) {
                                if ((new_state[real_col] & (0x01 << j)) != (old_state[real_col] & (0x01 << j))) {
                                    key_code = p_key_data->keymap[real_row * p_key_data->output_port_nums +
real_col].key_code;
                                    key_val = (old_state[real_col] & (0x01 << j)) ? 1 : 0;/* press or release */
                                    AW_DEBUG("aw951xx report: key_code = %d, key_val = %d\n", key_code, key_val);
                                    AW_DEBUG("aw951xx real_row = %d, real_col = %d\n", real_row, real_col);
                                    input_report_key(p_key_data->input, key_code, key_val);
                                    input_sync(p_key_data->input);
                                }
                                real_row++;
                            }
                        }
                    }
                    real_col++;
                }
            }
        }

        memcpy(&old_state[0], &new_state[0], p_key_data->output_port_nums * sizeof(unsigned int));

        /* all key release */
        if (!memcmp(&new_state[0], &p_key_data->def_output_state[0],
```

```
            p_key_data->output_port_nums * sizeof(unsigned int))) {
            aw951xx_set_port_direction_by_mask(aw951xx, p_key_data->output_port_mask, 0); /* set output mode */
            aw951xx_clear_interrupt(aw951xx); /* clear inputerrupt */
            enable_irq(aw951xx->irq_num);
            aw951xx_enbale_interrupt_by_mask(aw951xx, p_key_data->input_port_mask, 1);/* enable input interrupt */
            retry_state = aw951xx_get_port_input_state(aw951xx);
            if ((retry_state & p_key_data->input_port_mask) != p_key_data->input_port_mask) {
                disable_irq_nosync(aw951xx->irq_num);
                aw951xx_enbale_interrupt_by_mask(aw951xx, p_key_data->input_port_mask, 0);
            } else {
                AW_DEBUG("%s enter, all key release.\n", __func__);
                return;
            }
        }
        hrtimer_start(&p_key_data->key_timer, ktime_set(0, (1000 / HRTIMER_FRAME) * 1000), HRTIMER_MODE_REL);
    }
```

- GPIO 功能

  1）设置输出模式（推挽和输出，根据 dtsi 的 gpio-mode 决定）

  2）设置 io 的方向

  3）如果 io 配置为输出，设置输出状态

```
static void aw951xx_gpio_chip_init(struct aw951xx *aw951xx, struct aw951xx_gpio *p_gpio_data)
```

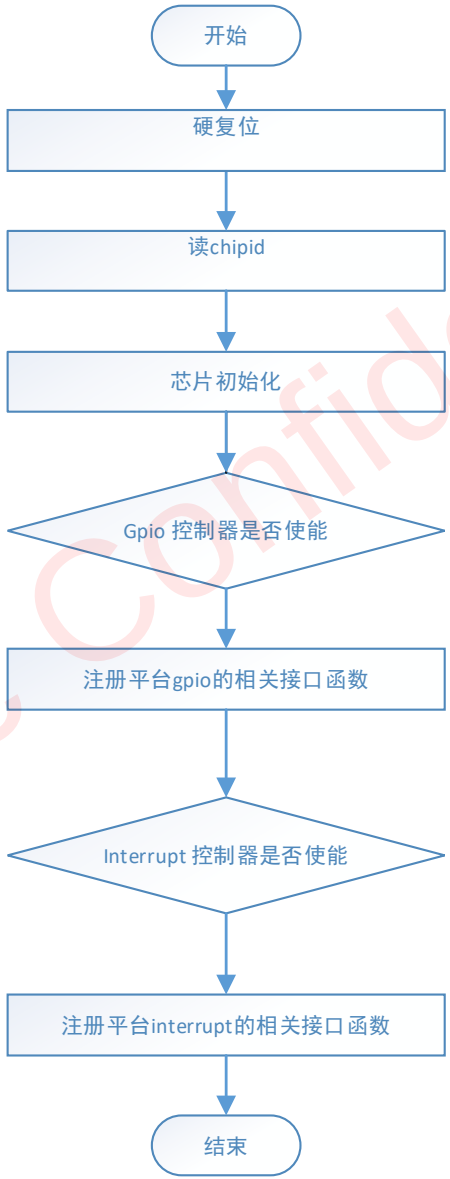
```
{
    int i = 0;
    struct aw951xx_singel_gpio *p_single_gpio_data = p_gpio_data->single_gpio_data;

    /*AW951XX default is gpio mode */
    //aw951xx_set_port_mode_by_mask(aw951xx, p_gpio_data->gpio_mask, 1);
    aw951xx_set_port_output_mode(aw951xx, p_gpio_data->output_mode); /* OD or pull push mode */
    for (i = 0; i < p_gpio_data->gpio_num; i++) {
        aw951xx_set_port_direction_by_mask(aw951xx, 0x1 << p_single_gpio_data[i].gpio_idx,
                            p_single_gpio_data[i].gpio_direction);
        if (p_single_gpio_data[i].gpio_direction == 0x00) { /* output */
            aw951xx_set_port_output_state_by_mask(aw951xx, 0x1<<p_single_gpio_data[i].gpio_idx,
                            p_single_gpio_data[i].state);
        }
    }
}
```

6.2.2 gpiochip架构驱动功能介绍

- 驱动初始化流程

  驱动初始化流程如下图所示



- gpio 接口函数定义

接口名字	功能介绍	参数	操作寄存器
aw951xx_gpio_direction_input	配置扩展 io 为输入	off：io 索引	P0OEN-P2OEN
aw951xx_gpio_direction_output	配置扩展 io 为输出	off：io 索引	P0OEN-P2OEN
aw951xx_gpio_get_value	获取扩展 io 的电平状态	off：io 索引，返回：io 的电平状态	P0DIST-P2DIST
aw951xx_gpio_set_value	设置扩展 io 的电平状态	off：io 索引　val：io 的电平状态	P0DO-P2DO
aw951xx_gpio_get_direction	获取扩展 io 的方向	off：io 索引，返回：io 的方向	P0OEN-P2OEN

- Interrupt 接口函数定义

接口名字	功能介绍	参数	操作寄存器
aw951xx_irq_mask	打开中断使能掩码	d->hwira：扩展 io 在平台的中断号	
aw951xx_irq_unmask	关闭中断使能掩码	d->hwira：扩展 io 在平台的中断号	
aw951xx_irq_bus_lock	中断锁存		
aw951xx_irq_bus_sync_unlock	中断解锁		
aw951xx_irq_set_type	设置中断触发状态	d->hwira：扩展 io 在平台的中断号；type：触发方式	
aw951xx_irq_set_wake	中断唤醒使能	On：1，打开中断唤醒功能，0，关闭中断唤醒功能	

6.3 驱动节点介绍

6.3.1 reg

节点名字	**reg**
功能描述	用于获取 gpio 所有寄存器值/往指定寄存器写入值
使用方法	获取寄存器值： cat reg 写寄存器值： echo 0x05 0xff > reg

6.3.2 chipid

节点名字	**chipid**
功能描述	用于获取芯片的 chipid
使用方法	cat chipid

7 常见问题

7.1 常见问题及排查方案

序号	Part	Item
1	输出电压与输出寄存器不一致	1、检查芯片是否 iic 正常
		2、检查 IO 口是否配置 GPIO 模式
		3、检查 IO 口是否配置输出模式
		4、如果配置高电平，实际输出低电平，检查输出模式是否被配置 push-pull 模式
		5、读取输入寄存器，是否与输出寄存器一致，如果不一致就是被外设影响
		6、检查外设是否有拉低 io 的能力
2	配置成 LED 模式时开机突然闪烁	1、检查 AD 脚的硬件连接，是否默认输出状态会导致灯亮
		2、检查驱动代码默认输出状态是否会导致灯亮
3	配置成按键模式按键无响应	1、检查输入寄存器是否在按下和松开也会动态改变
		2、检查对应 io 的中断使能寄存器是否被配置
		3、检查是否平台中断脚状态，是否在按下或者松开有低电平下降沿触发
		4、如果常拉低一般是漏中断，如果能正常拉低再拉高，检查平台中断使能是否被关闭
		5、如果常拉低配置平台中断低电平触发，如果还是不能恢复，一般是平台中断控制器异常

7.2 原理图 Layout Checklist

详细可参考 AW95124_Checklist

8 版本记录

版本	日期	说明
V1.0	2026-05-08	初版

免责声明

此文档中包含的信息被认为是准确、可靠的。但是，上海艾为电子技术股份有限公司（以下简称艾为）对这些信息的准确性或完整性均不作任何明示或暗示的陈述或保证，且对这些信息的使用后果不承担任何责任。

艾为保留在任何时间、没有任何通报的前提下修改本文档中发布的信息，包括但不限于产品资料和规格的权利。客户在下订单前应自行获取最新的相关信息，并验证这些信息是最新且完整的。本文档信息覆盖并取代所有先于此次公布的文档。

此文档为产品应用的参考文档，请根据实际项目应用进行修改，不保证没有瑕疵且不做任何担保。

对于艾为产品技术文档的信息，仅在没有对内容进行任何篡改且带有相关授权、条件、限制和声明的情况下才允许进行复制。艾为对篡改过的文件不承担任何责任或义务。复制第三方的信息可能需要遵守额外的限制条件。