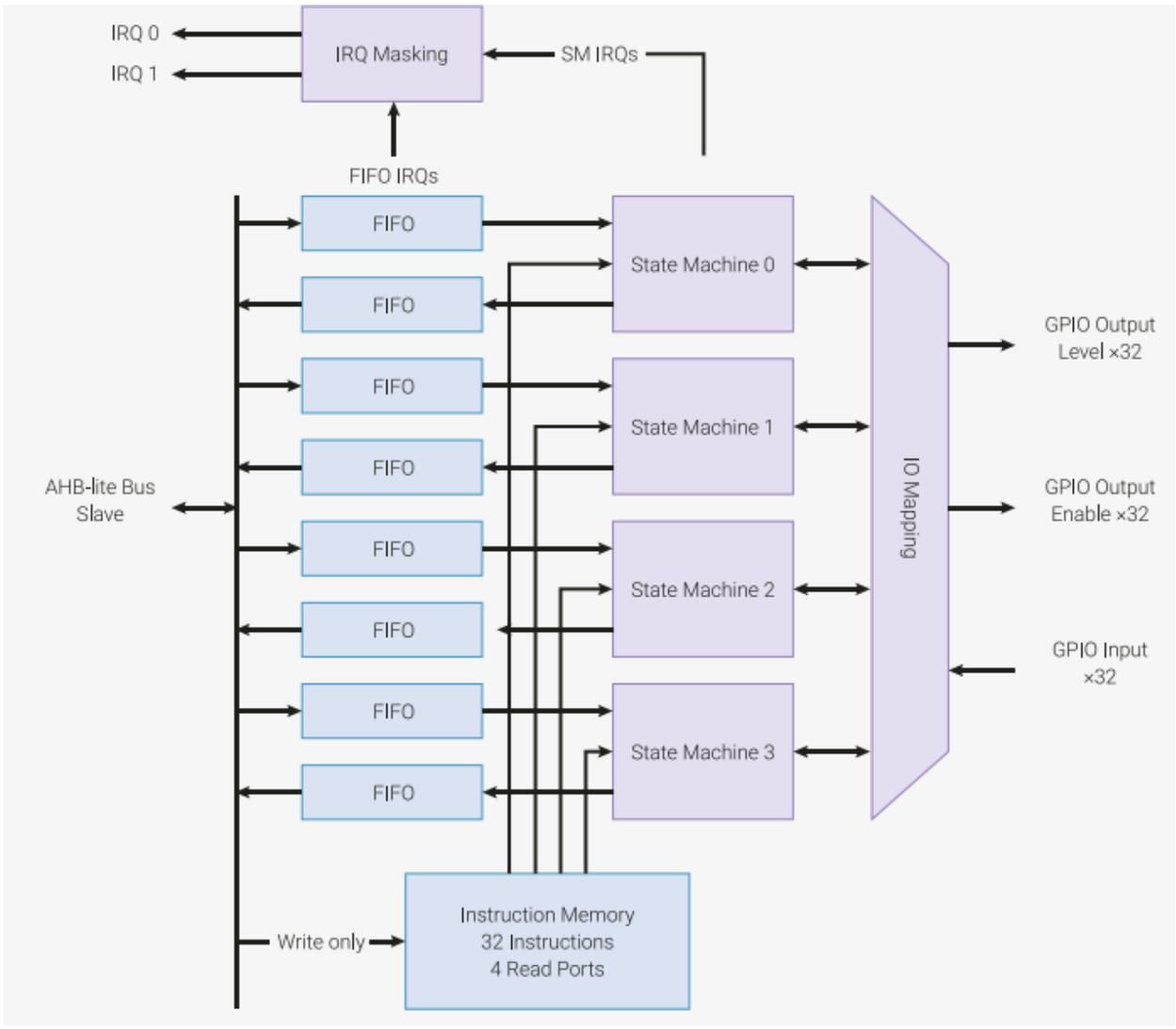
RP2040(树莓派Pico) PIO – 外设概述

RP2040中有2个相同的PIO块,每个PIO块都有专用的连接到总线结构,GPIO和中断控制器.单个PIO块的示意图如图所示.



PIO是一种通用的硬件接口,它可以支持多种IO标准.包括实现以下功能:

- 8080/6080 并行接口
- I2C
- I2S
- SDIO
- SPI/DSPI/QSPI
- UART
- DPI/VGA (利用电阻网络)

PIO和处理器一样,都是可以编程的.有两个PIO块,每个块有四个状态机,可以独立执行顺序程序来操作GPIO和传输数据.与通用处理器不同的是,PIO状态机对IO的专业化程度很高(highly specialised),它注重确定性,精确的时序,并与固定功能硬件紧密结合.每个状态机都配备有以下内容:

- 两个32位移位寄存器 (任意方向/任意移位数)
- 两个32位临时寄存器
- 4 * 32B FIFO (双向) 或 8 * 32 FIFO (单向)
- 小数分频器 (16整数 + 8小数)
- 可编程GPIO映射
- DMA/IRQ

每个状态机及其支持的硬件,占用的硅面积与SPI/I2C大致相同.然而,PIO状态机可以动态地配置和重新配置,以实现许多不同的接口,自由度很高.

以类似软件的方式使状态机可编程,而不是像CPLD那样的完全可配置的逻辑结构,可以在相同的成本和功率范围内提供更多的硬件接口,这也为那些希望通过直接编程而不是使用PIO库中的预制接口来利用PIO的全部灵活性的人提供了一个更熟悉的编程模型和更简单的工具流程.

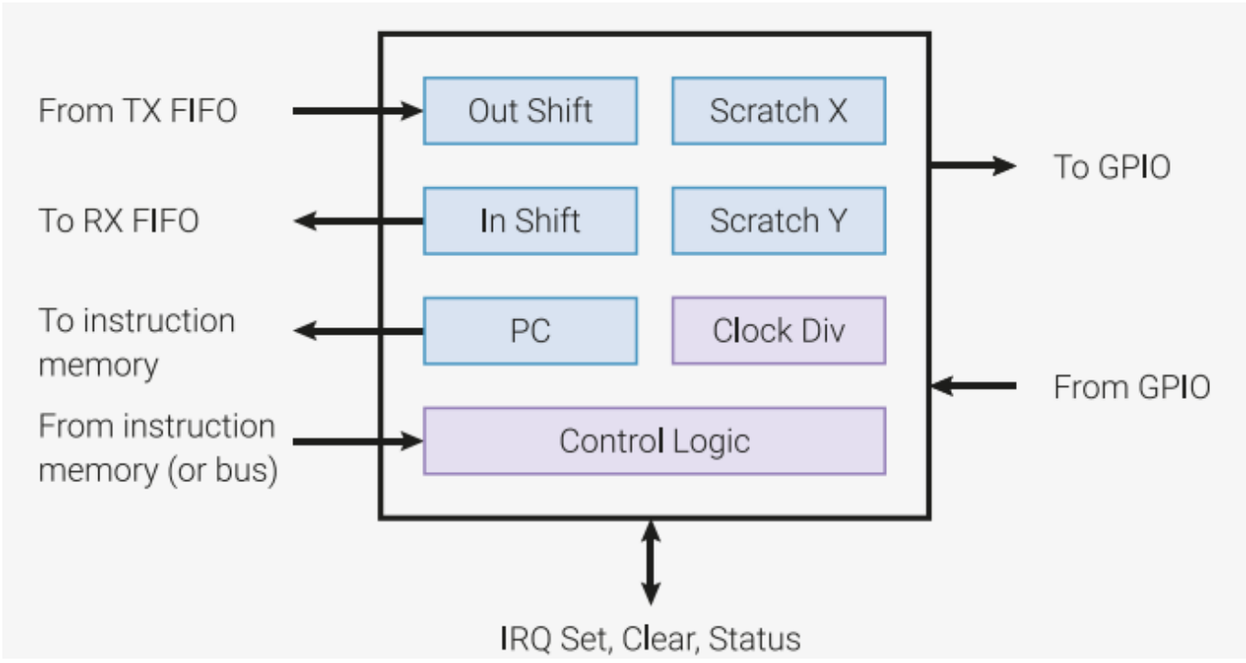
PIO具有很高的性能以及灵活性,这得益于每个状态机内部精心挑选的一组固定功能硬件.在输出DPI时,当使用48MHz系统时钟运行时,PIO可以在活动扫描线期间维持360Mb/s的速度.在这个例子中,一个状态机负责处理帧/扫描线时序和生成像素时钟,而另一个状态机负责处理像素数据,并解包运行长度编码的扫描线.

状态机的输入和输出最多可以映射到32个GPIO(RP2040限制为30个GPIO),所有状态机都可以独立地同时访问任何GPIO.例如,标准UART代码允许TX/RX/CTS/RTS成为任意四个GPIO.I2C允许SDA/SCL也是如此.可用的自由度取决于给定的PIO程序究竟如何选择使用PIO的引脚映射资源,但至少,一个接口可以自由地选择一些数量的GPIO.

四个状态机从一个共享指令存储器中执行,系统软件将程序加载到这个存储器中,配置状态机和IO映射,然后设置状态机运行.PIO程序的来源多种多样:

- 由用户直接组装
- 从PIO库中抽取
- 由用户软件编程生成

从这一点上看.状态机一般是自主的,系统软件通过DMA/IRQ和控制寄存器进行交互,与RP2040上的其他外设一样.对于比较复杂的接口,PIO提供了一套小而灵活的基元,使系统软件可以更多地亲自动手处理状态机控制流程.



PIO状态机执行短小的二进制程序.

在PIO库中可以找到UART/SPI/I2C等常用接口的程序,所以在很多情况下,不需要编写PIO程序.但是,PIO在直接编程时就灵活多了,它支持各种各样的接口,这些接口可能是设计者没有预料到的.

PIO共有9条指令,分别如下:

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JMP	0	0	0	Delay/side-set					Condition			Address				
WAIT	0	0	1	Delay/side-set					Pol	Source		Index				
IN	0	1	0	Delay/side-set					Source			Bit count				
OUT	0	1	1	Delay/side-set					Destination			Bit count				
PUSH	1	0	0	Delay/side-set					0	IfF	Blk	0	0	0	0	0
PULL	1	0	0	Delay/side-set					1	IfE	Blk	0	0	0	0	0
MOV	1	0	1	Delay/side-set					Destination			Op		Source		
IRQ	1	1	0	Delay/side-set					0	Clr	Wait	Index				
SET	1	1	1	Delay/side-set					Destination			Data				

下面是一个PIO程序集的例子:

```
.program squarewave
    set pindirs, 1   ; Set pin to output
again:
    set pins, 1 [1]  ; Drive pin high and then delay for one cycle
    set pins, 0      ; Drive pin low
    jmp again        ; Set PC to label `again`
```

PIO汇编器包含在SDK中,名为pioasm.该程序处理一个PIO汇编输入文本文件,其中可能包含多个程序,并写出组装好的程序供使用.对于SDK来说,这些组装好的程序是以C头的形式发出的,其中包含const数组.

在每一个系统时钟周期,每个状态机都会获取/解码并执行一条指令.每条指令恰好需要一个周期,除非它明确地停顿(如WAIT指令),指令还可以在下一条指令执行前插入最多31个周期的延迟,以帮助编写周期精确的程序.

比如pio例子中的squarewave,输出一个12.5MHz的方波(如果系统时钟频率为125 MHz).

主程序可以向指令缓存(32个插槽)写入数据,指令缓存是仅写入的,4个状态机都可以看到.

```
for (int i = 0; i < count_of(squarewave_program_instructions); ++i)
    pio->instr_mem[i] = squarewave_program_instructions[i];
```

状态机自身可以运行得和系统一样快,也可以进行分频,现在让状态机0进行2.5分频.

```
pio->sm[0].clkdiv = (uint32_t) (2.5f * (1 << 16));
```

上面的代码片段是一个完整的代码例子的一部分,它驱动一个12.5 MHz的方波从GPIO0输出.我们还可以使用WAIT PIN指令来延缓状态机的执行一段时间,或者使用JMP PIN指令来对引脚的状态进行分支,因此控制流可以根据引脚的状态而变化.

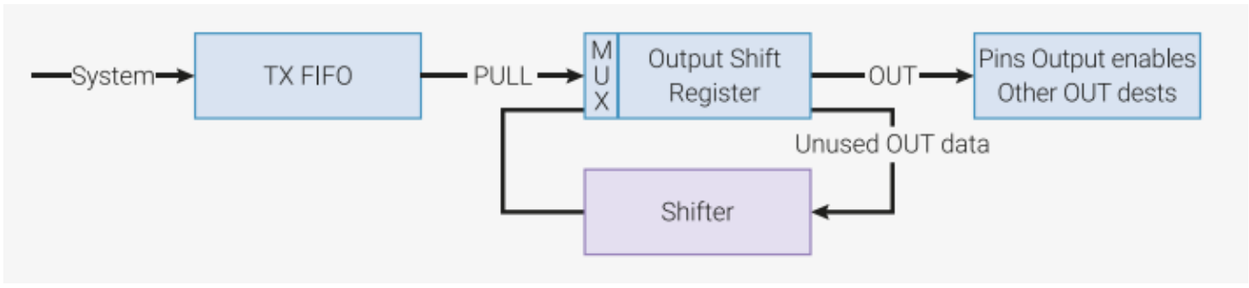
系统可以通过CTRL寄存器随时启动和停止每个状态机,多个状态机可以同时启动,PIO的确定性意味着它们可以保持完美的同步,下方代码就是配置IO方法和启动状态机,实际上我们无需了解这么深入,因为每个SDK给封装了很好的函数,这里目的是为了知晓基本的工作流程.

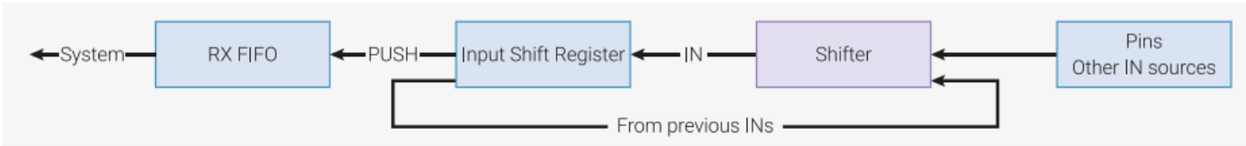
```
pio->sm[0].pinctrl =
        (1 << PIO_SM0_PINCTRL_SET_COUNT_LSB) |
        (0 << PIO_SM0_PINCTRL_SET_BASE_LSB);
gpio_set_function(0, GPIO_FUNC_PIO0);
hw_set_bits(&pio->ctrl, 1 << (PIO_CTRL_SM_ENABLE_LSB + 0));9
```

在PIO内部只有少数的几个寄存器,面向我们编程使用的只有OSR/ISR(输出/输入移位),Scratch(分为X寄存器和Y寄存器).

可以通过PULL显式地把数据从TX FIFO存入OSR,然后使用OUT命令取出数据,可以一次性取出32位,也可以取出1-32位.同理,ISR也是一样,只是PULL改成PUSH,OUT改成IN.

而Scratch是普通寄存器,除了IN/OUT之外,还可以SET和MOV.





接下来了解下PIO的核心功能.

Side-set 是一种允许状态机改变最多5个引脚的电平或方向的功能,与当前指令同时进行.

一个需要这样做的例子是快速SPI接口,在这里,时钟转换(切换1→0或0→1)必须与数据转换同时进行,即一个新的数据位从OSR转移到GPIO,在这种情况下,一个带Side-set的OUT就可以同时实现这两个功能.(注意是同时设置)

这使得接口的时序更加精确,减少了整个程序的大小(因为不需要单独的SET指令来翻转时钟引脚),也提高了SPI可以运行的最大频率.

Side-set也使得GPIO映射更加灵活,因为它的映射与SET无关.示例中I2C代码允许SDA和SCL被映射到任意两个任意引脚,.通常情况下,SCL切换来同步数据传输,SDA包含被移出的数据位.然而,一些特殊的I2C时序,如开始和停止线条件,需要一个固定的模式在SDA以及SCL上驱动.I2C用来实现这一目的的映射如下:

- Side-set -> SCL
- OUT -> SDA
- SET -> SDA

具体代码解释(需要在之前声明哪个引脚是Side-set引脚):

```
.side_set 1 ;声明了1个Bit的Side-set

out x, 1    side 0     ; 保持SCK无效状态(主命令:移位读X寄存器1个Bit)
mov pins, x side 1 [1] ; 设置SCK(主命令:把上一次[不是上一句]读到的Bit电平挪到引脚,并延迟一个周期.)
in pins, 1  side 0     ; 拉低SCK(主命令:把当前引脚输入移入X寄存器)
```