处芯积律自研项目相关资料

前言：

一句话定位：ISP DMA 双通道 + 覆盖率驱动 + QSPI UVM RAL——从「跑得通」走向「验得全」的进阶 SoC。
本项目在 SoC V1.1 硬件基座上，强化**覆盖率驱动验证**与 **UVM RAL** 方法学。工程提供块级 UVM 基线（ISP、QSPI）、PINMUX FPV 及 SoC 集成仿真入口；完整 testplan 与覆盖率 closure 需学员自行扩展。

用途	路径
VNC 工程根（学员环境）	/project/SOC2.0/
SoC 集成仿真入口	/project/SOC2.0/soc/
建议个人副本	cp -a /project/SOC2.0 ~/work/SOC2.0

V2.0 相对 V1.1 的升级点

维度	SoC V1.1（基础版）	SoC V2.0（进阶版）
定位	SoC 入门、简历从 0 到 1	验证方法学进阶、覆盖率 + RAL
ISP DMA	RDMA / WDMA 已有	文档与实验强调**双 AHB 主通道**验证与吞吐分析
QSPI 验证	UVM 基线 + RAL 雏形	**完整 UVM RAL**（ral_qspi.sv + reg_adapter）
覆盖率	Makefile 可选	**默认开启** -cm line+cond+tgl（soc / QSPI sim）
工程规范	—	`define I2C 等集成宏更规范
推荐前置	Verilog 入门即可	建议先完成 V1.1 或具备 UVM 基础

注意 EDA 工具的版本信息：
VCS 至少要 2016 版本以上。 VCS 2014 的不行。
UVM 至少是 UVM-1.2 。UVM-1.1 该项目会有很多问题。

一 IP 介绍

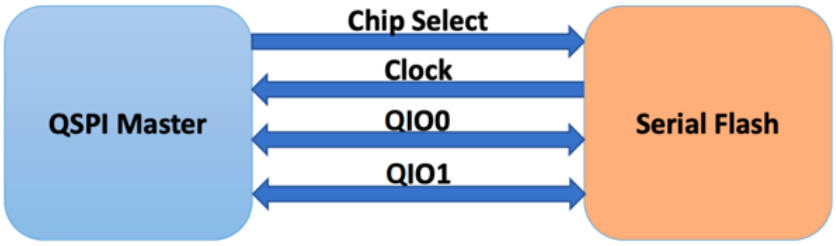
1. QSPI

QSPI 是 Quad SPI 的简写，表示 6 线 spi，是 Motorola 公司推出的 SPI 接口的扩展，比 SPI 应用更加广泛。
Queued SPI 相对于 SPI 来说增加了两根 I/O 线（SIO2、SIO3），提供了每次传输的 bit 数。QSPI 可以的工作模式 Single-Bit SPI、Dual SPI、Quad SPI。

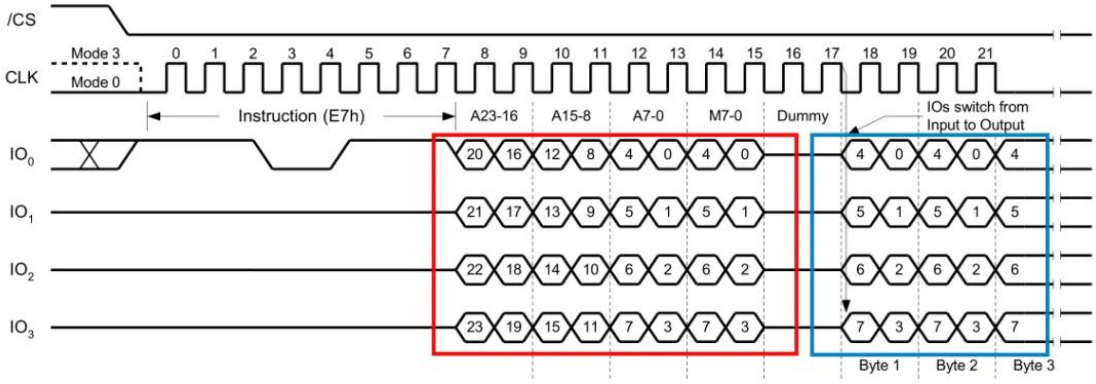
QSPI 的传输模式

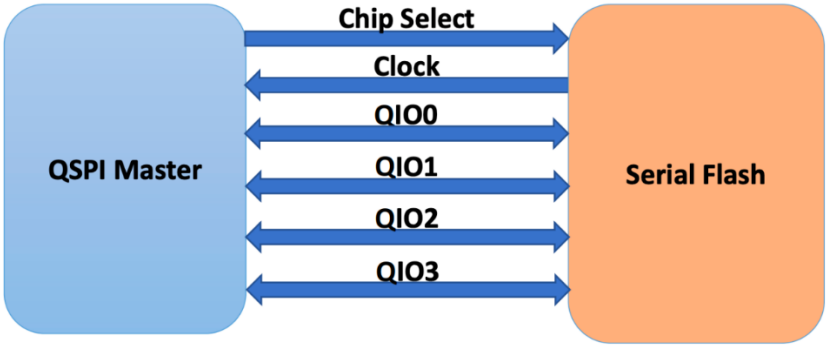
QSPI 单线传输模式

QSPI 双线传输模式



QSPI 四线传输模式





QSPI 时序图

QSPI 通过命令与 Flash 通信，每条命令包括指令、地址、交替(复用)字节、空指令和数据共五个阶段，而这五个阶段任一阶段均可跳过，但至少要包含指令、地址、交替字节或数据阶段之一。nCS 在每条指令开始前下降，在每条指令完成后再次上升。

具体的 QSPI 协议可以参考
Quad-SPI (QSPI) interface on STM32 microcontrollers

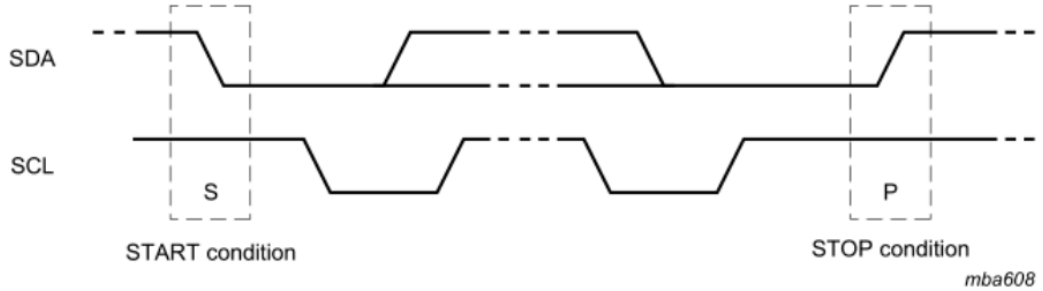
2. I2C

I2C 协议比较简单，本项目 I2C 的配置参见寄存器列表。
注意
PRE = Freq_I2C / ((5*Freq_SCL) - 1)
Freq_I2C is the clock frequency of I2C module.

START and STOP

START and STOP conditions由主机产生，具体参加下图:

START：SCL为高期间，SDA由高->低;
STOP ：SCL为高期间，SDA由低->高;



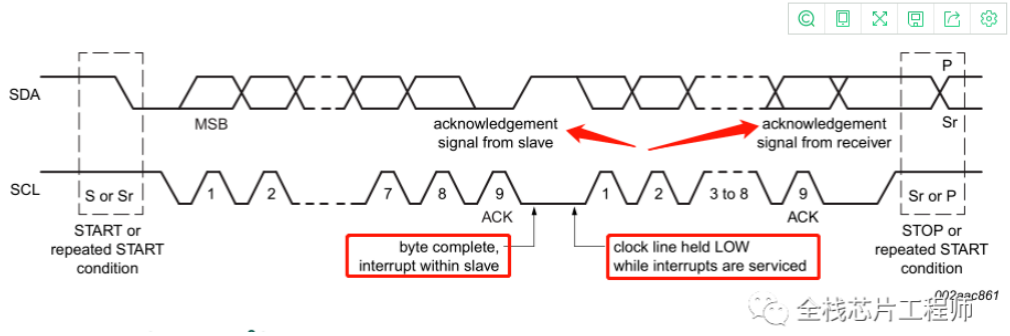
START and STOP conditions

Byte传输、ACK应答

信号传输以Byte为单位，每个字节后都由1bit的应答信号。

MSB先发，LSB最后（I2C/SPI都是MSB先发，而UART是LSB先发）。

当Slave正处理其内部中断而无法接收I2C Master数据时，Slave可拉低SCL以使得Master进入等待状态。



I2C写时序

向指定寄存器地址写入指定数据操作时序：

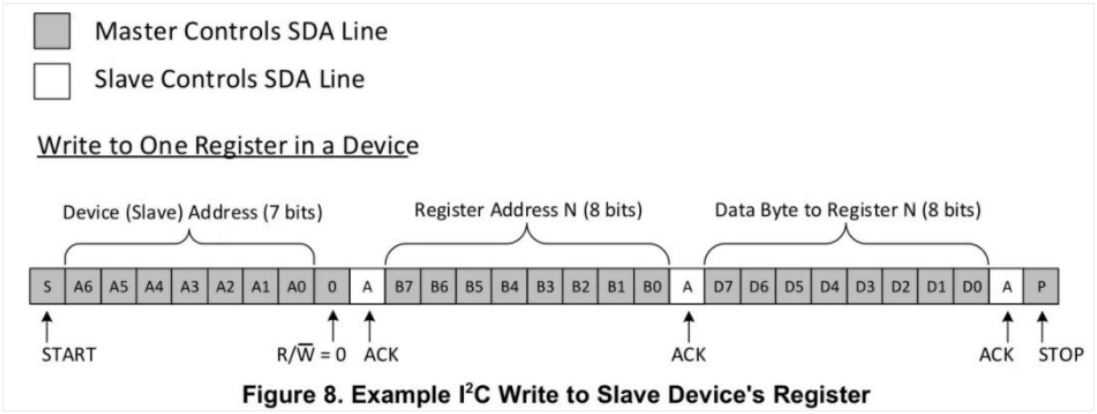


Figure 8. Example I²C Write to Slave Device's Register

3. UART

通用异步收发器（Universal Asynchronous Receiver/Transmitter），通常称作 UART，是一种串行、异步、全双工的通信协议，在嵌入式领域应用广泛。UART 时序图如下

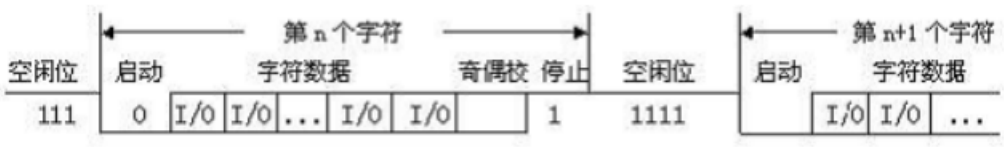


图 1 UART 工作原理

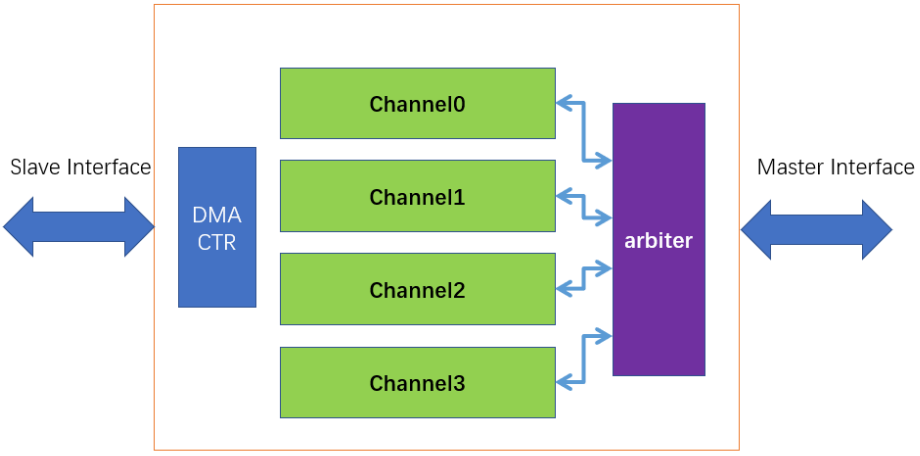
起始位：先发出一个逻辑“0”的信号，表示传输字符的开始。
数据位：紧跟起始位之后。数据位的个数可以是 4、5、6、7、8 等，构成一个字符。通常采用 ASCII 码。从最低位开始传送，靠时钟 来定位。
奇偶校验位：数据位加上这一位后（跟在数据位尾部），使得“1” 的位数应为偶数(偶校验)或奇数(奇校验)，以此来校验数据传送的正 确性。
停止位：它是一个字符数据的结束标志。可以是 1 位、1.5 位、 2 位的高电平（逻辑“1”）。
空闲位：处于逻辑“1”状态，表示当前线路上没有数据的传送。

波特率：
数据传输速率使用波特率来表示。单位 bps（bits per second），常见的波特率 9600bps、19200bps、115200bps 等等，如果串口波特率设置为 9600bps，那么传输一个比特需要的时间是 1/9600≈104.2us，即 1bit 传输时间大约为 104us，传送一个数据实际是 10 个比特（开始位-8 个数据位-停止位），传输速率实际为 9600*8/10=7680bps。

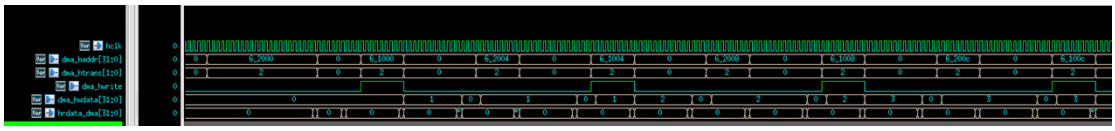
串口的具体配置参见寄存器列表，注意配置波特率时，需要 UART_LCR[7]先配置为 1 才能生效。

4. DMA

SOC V2 的项目中 DMA 接 AHB master 接口及 AHB slave 接口。该 DMA 有 4 通道(RR 仲裁)。支持地址非对齐读写、支持中断使能配置。使用非常简单，只需配置写地址、读地址、传输数据量大小(字节)即可。

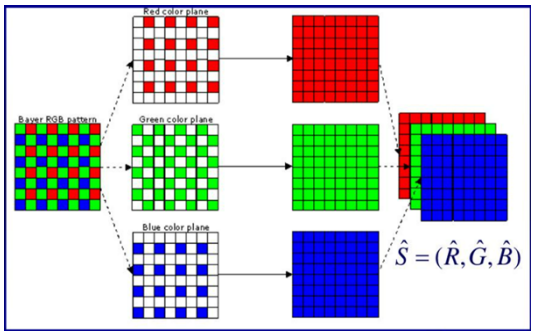


DMA 启动时，先从源地址指定的 memory 读数，然后将这些数写入目的地址的 memory 中。

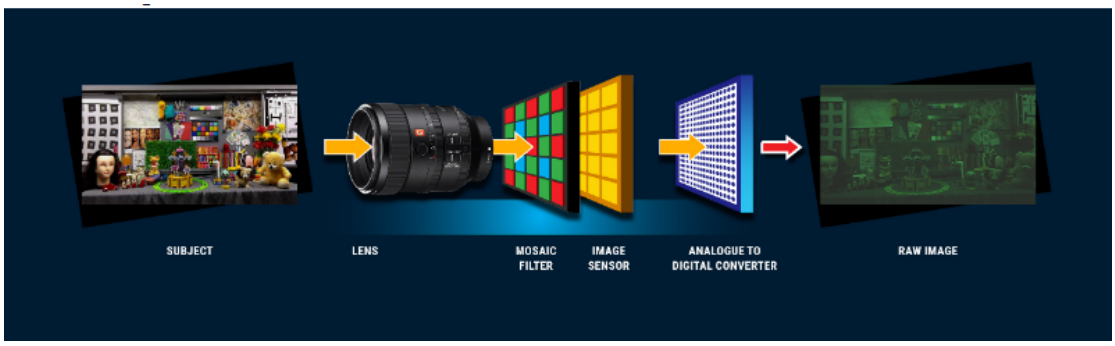


5. ISP

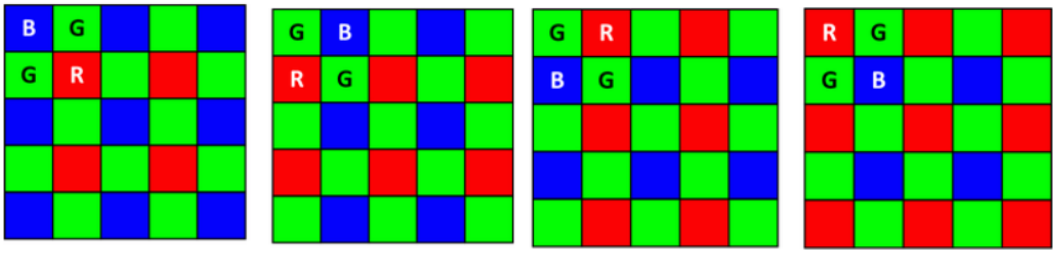
在本项目中，ISP 包括 Demosaicing 和 dgain 两种算法。
Demosaicing 是一种数位影像处理算法，目的是从覆有滤色阵列（Color filter array，简称 CFA）的感光元件所输出的不完全色彩取样中，重建出全彩影像。其过程如下。



摄像头感光后，从外界获取的信息是一个数据阵列，由于摄像头传感器的特性，每个像素只有唯一的色彩。



因此我们看到摄像头采集到的原始图像中每个像素点或红，或绿色，或蓝色。为了恢复每个像素点的 RGB 分量，将图像恢复成全彩图。我们需要 demosaicing 处理。
原始图像的格式根据像素点不同颜色排列可以分成四类，分别是 BGGR，GBRG，GRBG 和 RGGB，以下是四种格式的图像。



下面介绍 demosaicing 算法。ISP demosaicing 算法的具体内容
前提：图片的小平滑区域内，色差恒定。以下面的数据为例子。
在这个数据里面

B11, G12, B13, G14, B15, G16
G21, R22, G23, R24, G25, R26
B31, G32, B33, G34, B35, G36
G41, R42, G43, R44, G45, R46
B51, G52, B53, G54, B55, G56
G61, R62, G63, R64, G65, R66

$$R_{ij} - G_{ij} = R_{mn} - G_{mn}$$
$$B_{ij} - G_{ij} = B_{mn} - G_{mn}$$

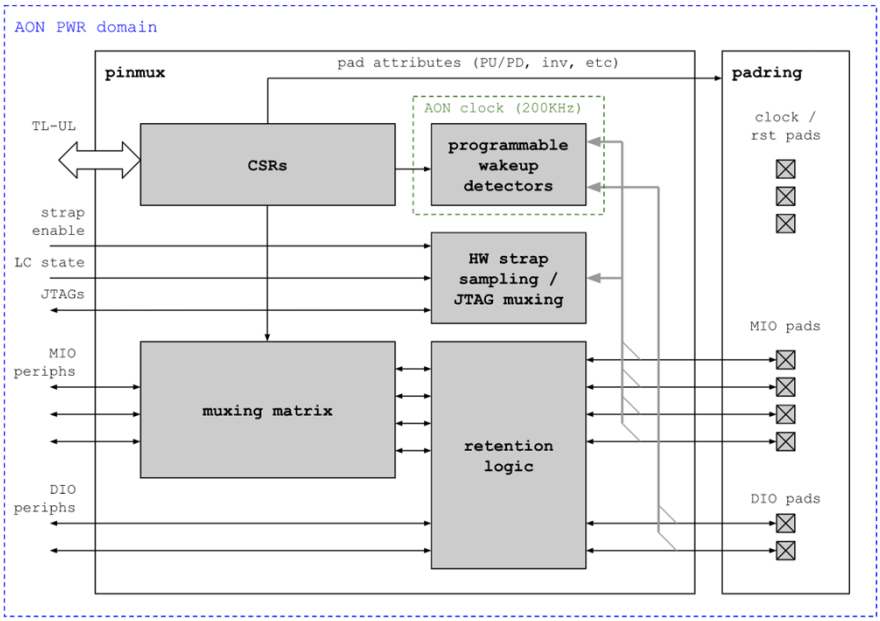
我们将根据上述前提，恢复各个像素点的 RGB 分量。

1. 基于 B33 计算出 G33,R33
   a. 计算 G33 边缘检测 在边缘方向上插值
      diffA = abs(G32 - G34) 水平方向差值
      diffB = abs(G23 - G43) 垂直方向差值
      G33 = (G32 + G34) / 2, 当(diffA < diffB) 垂直方向差值较大，取水平方向均值
      G33 = (G23 + G43) / 2, 当(diffA > diffB) 水平方向差值较大，取垂直方向均值
      G33 = (G32 + G34 + G23 + G43) / 4, 当(diffA == diffB) 差值一样大，取均值
   b. 计算 R33 小平滑区域内 色差恒定理论
      1.a 的方法得到 G33，同理可计算出 G22,G24,G42,G44
      R33 = G33 + (R22 + R24 + R42 + R44) / 4 - (G22 + G24 + G42 + G44) / 4
2. 基于 G34 计算出 R34,B34
   a. 计算 R34
      用 1.a 的方法可计算出 G24,G44
      R34 = G34 + (R24 + R44) / 2 - (G24 + G44) / 2
   b. 计算 B34
      用 1.a 的方法可计算出 G33,G35
      B34 = G34 + (B33 + B35) / 2 - (G33 + G35) / 2
3. 基于 G43 计算出 R43,B43 (同方法 2)
4. 基于 R44 计算出 G44,B44 (同方法 1)

Dgain 算法比较简单，就是对各个像素点乘以增加的系数即可。

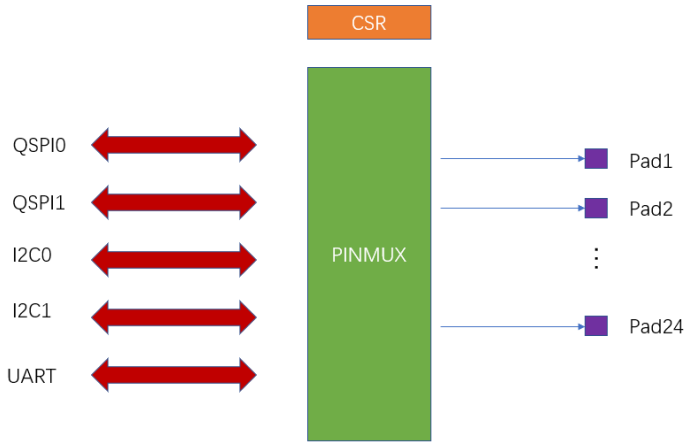
6. PINMUX

SOC 项目中也增加了 PINMUX。下面是一个 PINMUX 的框图（注意非本项目框图）。



上面是一个典型的 PINMUX 框图，可以看见不同外设的信号从不同的 pad 出去。

SOC V2 项目中的 PINMUX 框图如下



该项目中 QSPI0，QSPI1，I2C0 ，I2C1，UART 共五个外设复用 24 个 pad。Pad 和不同外设的关系图如下。

	AF0	AF1	AF2	AF3	AF4
PIN	QSPI0	QSPI1	I2C0	I2C1	UART
1	SPI0_CLK				
2	SPI0_CSN0	SPI1_CLK			
3	SPI0_SDO0	SPI1_CSN0	I2C0_SDA		UART_TXD
4	SPI0_SDI0	SPI1_SDO0	I2C0_SCL		UART_RXD
5	SPI0_CSN1	SPI1_SDI0		I2C1_SDA	
6	SPI0_SDO1	SPI1_CSN1		I2C1_SCL	
7	SPI0_SDI1	SPI1_SDO1			
8	SPI0_CSN2	SPI1_SDI1			
9	SPI0_SDO2	SPI1_CSN2			
10	SPI0_SDI2	SPI1_SDO2			
11	SPI0_CSN3	SPI1_SDI2			
12	SPI0_SDO3	SPI1_CSN3			
13	SPI0_SDI3	SPI1_SDO3			
14		SPI1_SDI3			
15	SPI0_CSN0	SPI1_CSN0			
16	SPI0_SDO0	SPI1_SDO0		I2C1_SDA	
17	SPI0_SDI0	SPI1_SDI0		I2C1_SCL	
18	SPI0_CSN1	SPI1_CSN1			
19	SPI0_SDO1	SPI1_SDO1	I2C0_SDA		UART_TXD
20	SPI0_SDI1	SPI1_SDI1	I2C0_SCL		UART_RXD
21	CPU_CLK				
22	CPU_RST_N				
23	PCLK				
24	PRST_N				

不同 pad 有对应的 register 去配置选择不同的信号。上图 AF0，AF1，AF2，AF4，AF4 是 pad 不同的选择。相关信息可以查看 PINMUX 的寄存器列表。

二 SOC 介绍

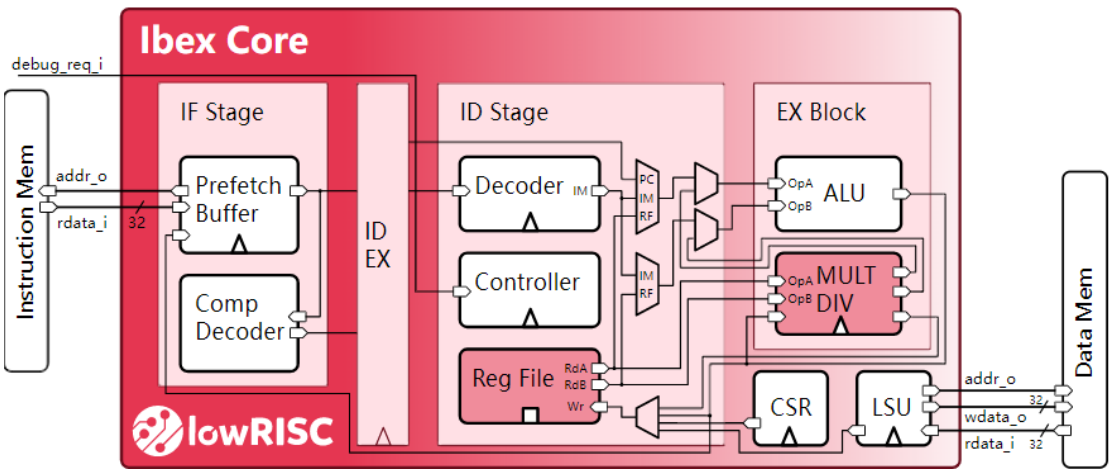
1、功能列表

本 SoC 设计是最精简的 RISC-V 处理器 SoC，包含最基本的 AMBA 总线、外设，满足 RISC-V 处理器爱好者的高效学习。

- CPU：RISC-V
- 总线：AMBA2.0,AHB/APB
- 外设：UART/I2C/QSPI/DMA/ISP/PINMUX
- 系统时钟：50MHz
- 主存：128KB

2. CPU 核介绍：

ibex 是一款 32 位开源 RISC-V 处理器，2 级流水，支持 RV32I、RV32C、RV32M、RV32B 等指令，支持 M-Mode 和 U-Mode。



Ibex 相关的资料介绍：
Ibex Reference Guide — Ibex Documentation 0.1.dev76+g056cb44.d20220830 documentation (ibex-core.readthedocs.io)

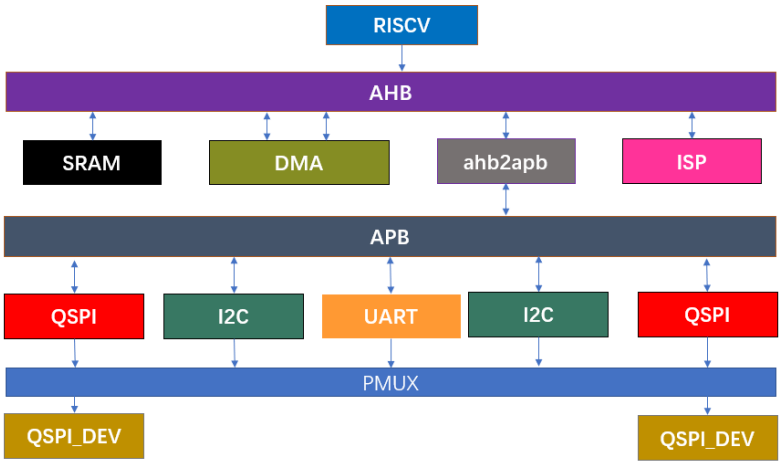
Ibex github 库：
ibex/vendor/lowrisc_ip/dv/sv/mem_model at master • lowRISC/ibex • GitHub

工具链信息：

https://github.com/lowRISC/lowrisc-toolchains/releases/download/20210412-1/lowrisc-toolchain-gcc-rv32imc-20210412-1.tar.xz

3、SOC 架构框图

本 SoC 设计包含 RISC-V 处理器、AHB/APB 总线、SRAM，外设包括 QSPI、UART、I2C，DMA，ISP，PINMUX，整体系统框架如下图。



4、顶层接口信号说明

详细描述模块的顶层信号，包含信号名、位宽、输入输出属性、详细说明等。

Signal	Width	I/O	Description
SoC Interface			
Pad_1	1	IO	具体功能请查看 PINMUX 的 table。根据不同
Pad_2	1	IO	
Pad_3	1	IO	

Pad_4	1	IO	的设定可以进行修改。
Pad_5	1	IO	
Pad_6	1	IO	
Pad_7	1	IO	
Pad_8	1	IO	
Pad_9	1	IO	
Pad_10	1	IO	
Pad_11	1	IO	
Pad_12	1	IO	
Pad_13	1	IO	
Pad_14	1	IO	
Pad_15	1	IO	
Pad_16	1	IO	
Pad_17	1	IO	
Pad_18	1	IO	
Pad_19	1	IO	
Pad_20	1	IO	
Pad_21	1	I	CPU_CLK
Pad_22	1	I	CPU_RST_N
Pad_23	1	I	PCLK
Pad_24	1	I	PRST_N

5、各模块信号说明及设计细节

各个模块详细介绍见上述 IP 部分。

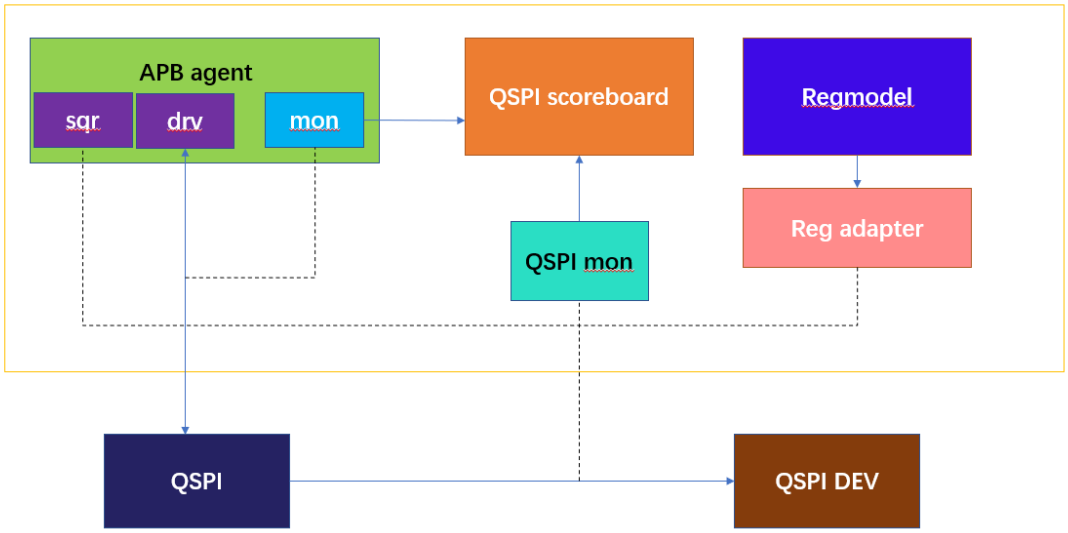
6、寄存器描述

参见 SoC 寄存器列表。

三. 验证环境

1. IP 验证环境

QSPI 的验证环境

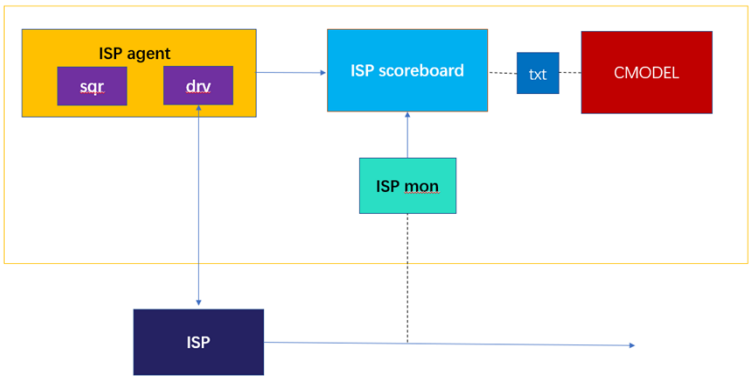


➢ 文件夹介绍：
QSPI/tb/env 各种 QSPI testbench 的 组件
QSPI/tb/tests 测试用例
QSPI/sim 跑 simulation 的目录

➢ 命令：
编译命令 make comp
运行指令 make run_{test_name}
Example : make run_qspi_read_test

大家根据自己需求更改 makefile 文件

ISP 的验证环境

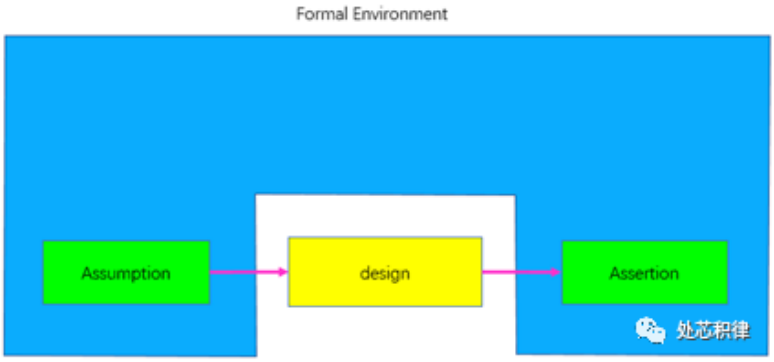


➢ 文件夹介绍：
ISP/tb/env 各种 QSPI testbench 的 组件
ISP/tb/tests 测试用例
ISP/sim 跑 simulation 的目录
ISP/cmodel 放 ISP cmodel 的源代码

0;
➢ 命令：
编译命令 make comp
运行指令 make run_{test_name}
Example : make run_isp_test

大家根据自己需求更改 makefile 文件

Pinmux Formal 验证环境



➢ 文件夹介绍：

PINMUX/rtl 放 pinmux 设计代码的地方

ISP/FPV/assertion 放 assertion 和 assumption 的地方

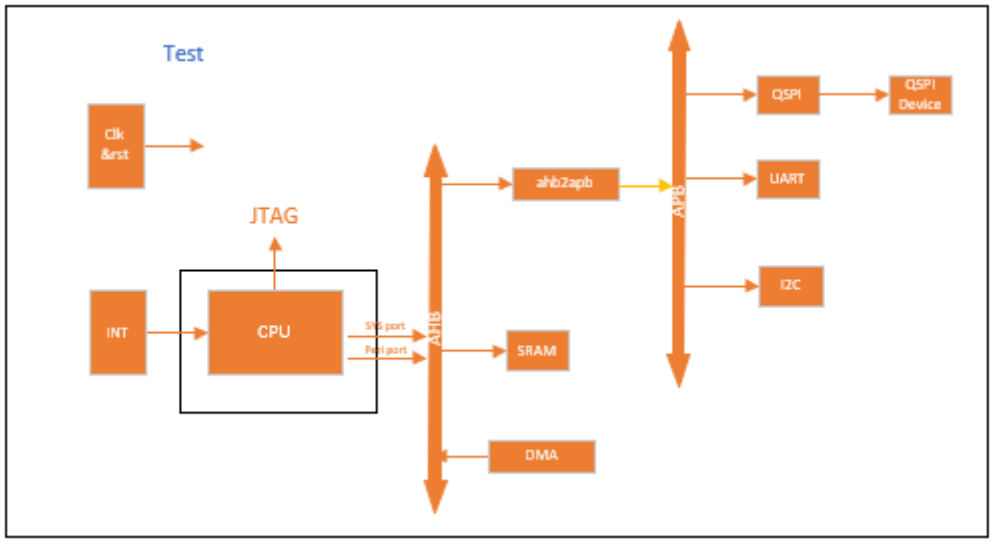
ISP/solution 放 formal TCL 脚本和跑 formal verification 的地方。

➢ 命令：

```
vcf -verdi -f batch.tcl
```

2. SOC 验证环境

仿真环境框图



➢ 文件夹介绍：

soc/filelist ： 环境的 filelist

soc/ibex ： ibex cpu 核

soc/ip ： 外设及总线信息，其中外设全部代码可见，总线目前是加密的。

soc/model: sram model

soc/sw ： 放置各个模块测试用的 C pattern

soc/test: 挂载 qspi device 等 vip

soc/top： soc 顶层文件，例化各个模块。

➢ 命令：

(1) 编译 test 环境 ： make comp

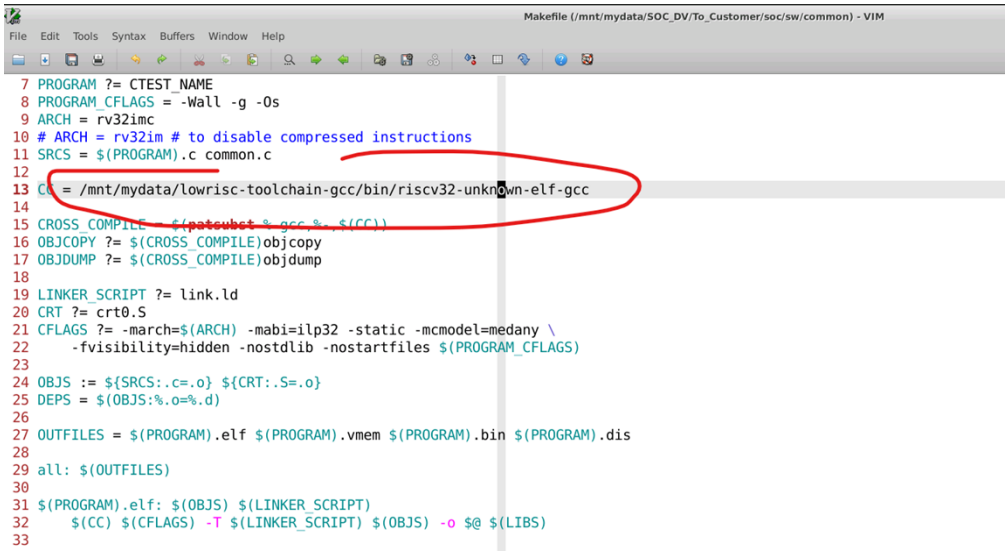
(2) 单独 C compile + run RTL simulation : make run_{test_name}

注意：

1. Sw 下面的 test_name 要和 c 的 name 一致，比如 spi_test 那么 spi_test 目录下的 c 的名字叫 spi_test.c
2. 运行 soc 的 toolchain 需要自己下载。地址在

https://github.com/lowRISC/lowrisc-toolchains/releases/download/20210412-1/lowrisc-toolchain-gcc-rv32imc-20210412-1.tar.xz

安装完修改 C test 路径下的 sw/common/makefile ，如



修改图中红色圈起来的地方即可。