处芯积律自研项目相关资料

前言:

一句话定位：完整 IP 图谱 + 大规模 SoC 集成 + CNN 加速器 + JTAG 在线调试 + 系统级 UVM 回归——把 SoC「拼装—验证—综合」讲清楚。
SoC V3.0 在 V2.0 基座上新增 **PWM、DM/JTAG 调试、CNN 加速器**，并引入 yrun/ycheck 回归框架与 Spyglass/DC/FM 设计脚本。工程提供块级 UVM（QSPI、ISP）、PINMUX FPV、SoC C/UVM 集成测试及覆盖率基线；完整 testplan 与 closure 需学员自行扩展（参见 soc/doc/testplan.md）。

用途	路径
VNC 工程根(学员环境)	/project/SOC3.0/
SoC 集成仿真入口	/project/SOC3.0/soc/sim/ （source env.csh 后使用 yrun）
SoC RTL 顶层	soc/top/top.sv（仿真/集成以 **top.sv** 为准）
建议个人副本	cp -a /project/SOC3.0 ~/work/SOC3.0

V3.0 相对 V2.0 的升级点

维度	SoC V2.0（进阶版）	SoC V3.0（精进版）
定位	覆盖率 + QSPI RAL + ISP DMA	大规模集成 + AI 加速器 + 在线调试 + 设计 Flow
新增 IP	—	**4 路 PWM、DM/JTAG、CNN**（conv / pool / line_mul）
总线矩阵	5 主 / 4 从	**7 路 AHB 主**（含 CNN、DM）/ **5 路从**
顶层文件	top.v	top.sv（含 CNN APB、中断、DM 系统总线）
回归框架	make run_*	**yrun / ycheck / ydebug**（soc/sim/script/flow_script/）
系统 UVM	基础集成	soc/env/agt_mem_bus + 多外设 UVM case（CNN/JTAG/PWM 等）
设计 Flow	—	flow/spyglass、flow/dc、flow/fm

推荐前置	完成 V1.1 或 UVM 基础	建议先完成 V1.1 / V2.0 任一

注意 EDA 工具的版本信息：

1. VCS 至少要 2016 版本以上。 VCS 2014 的不行。
2. UVM 至少是 UVM-1.2 ，UVM-1.1 该项目会有很多问题。
3. 使用 YGPT 需要海外的服务器

一 IP 介绍

1. QSPI

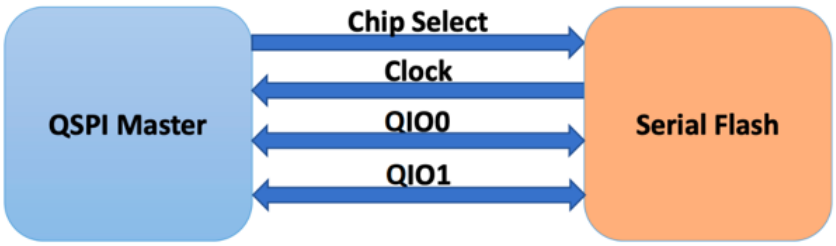
QSPI 是 Quad SPI 的简写，表示 6 线 spi，是 Motorola 公司推出的 SPI 接口的扩展，比 SPI 应用更加广泛。

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。

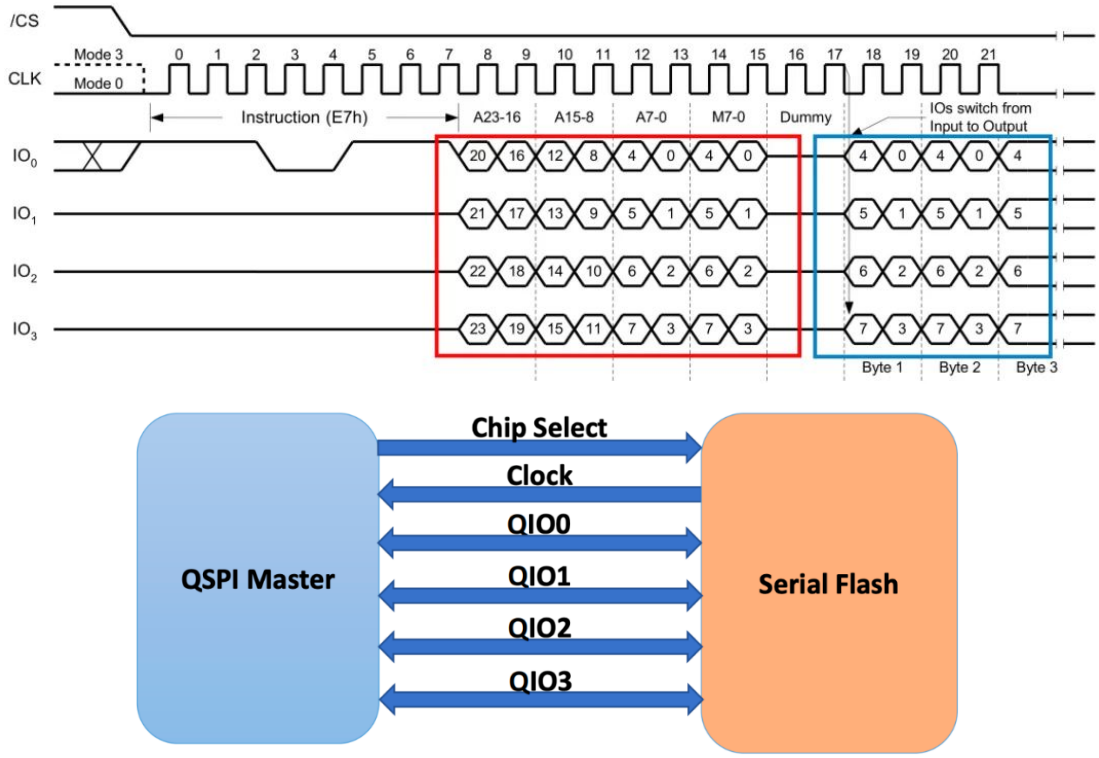
QSPI 的传输模式

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

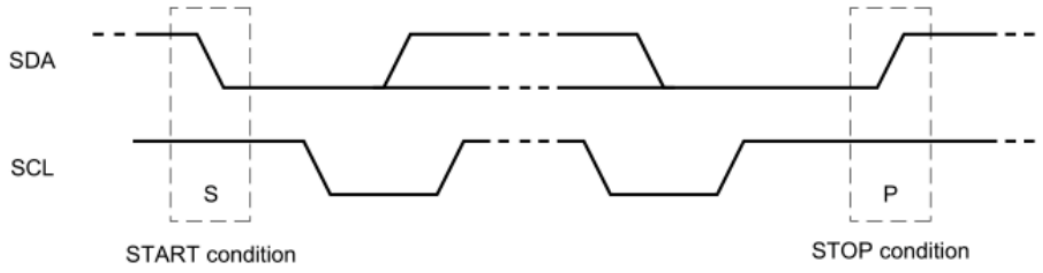
2. I2C

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；



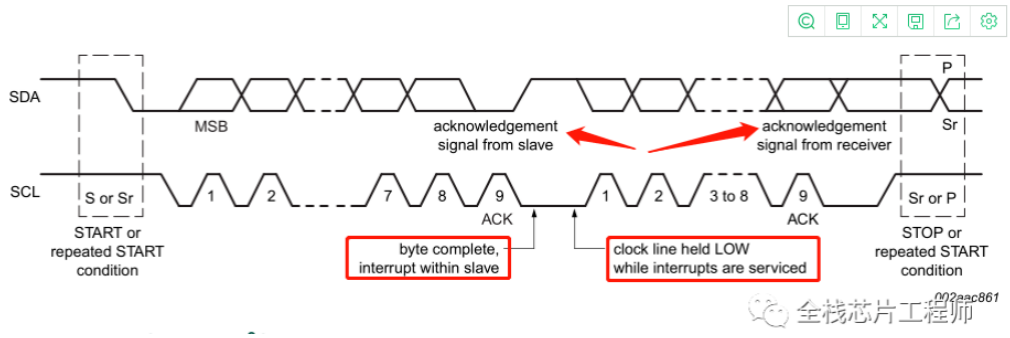
START and STOP conditions

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：

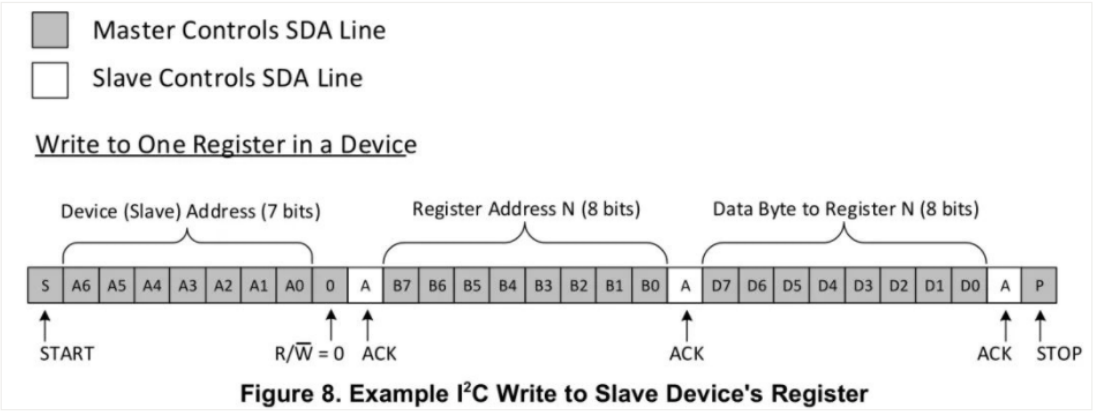


Figure 8. Example I²C Write to Slave Device's Register

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

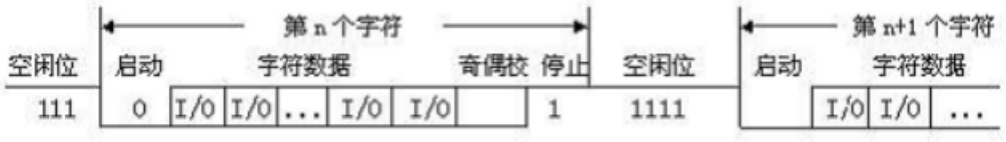


图 1 UART 工作原理

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。

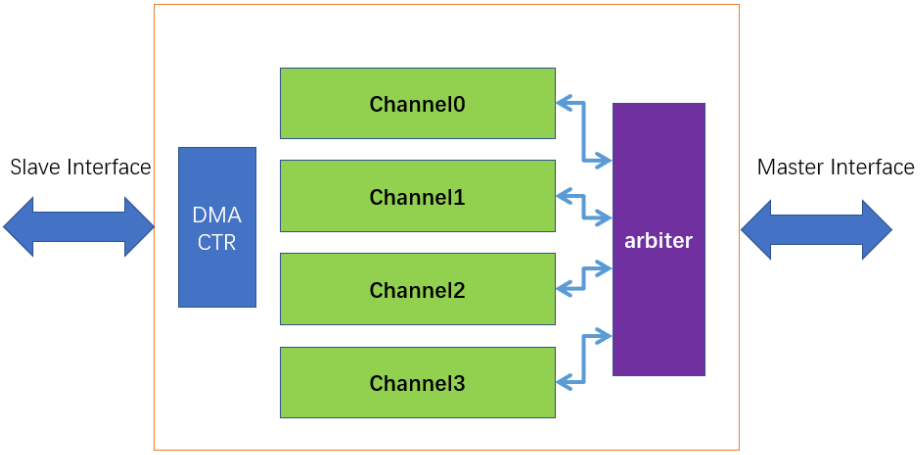
波特率：

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。

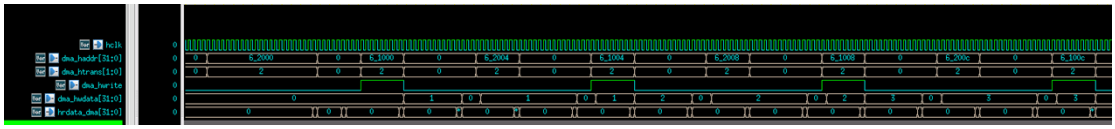
串口的具体配置参见寄存器列表，注意配置波特率时，需要 UART_LCR[7]先配置为 1 才能生效。

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。



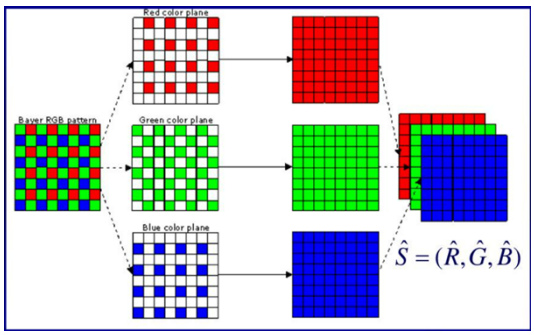
DMA 启动时，先从源地址指定的 memory 读数，然后将这些数写入目的地址的 memory 中。



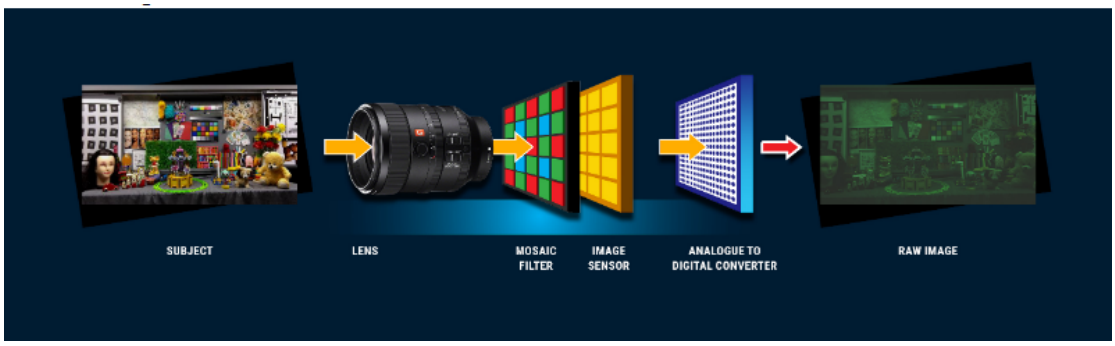
5. ISP

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。

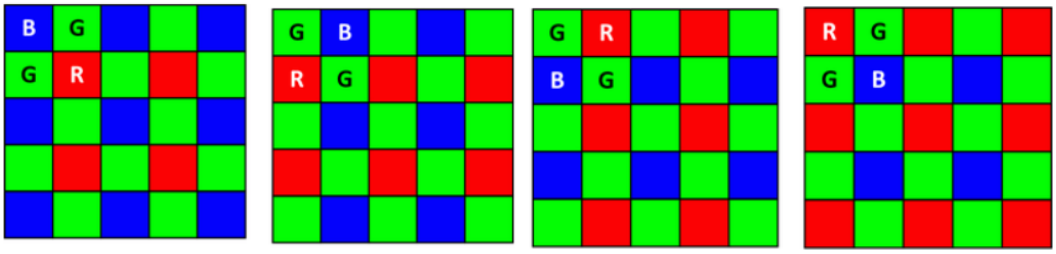


摄像头感光后，从外界获取的信息是一个数据阵列，由于摄像头传感器的特性，每个像素只有唯一的色彩。



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

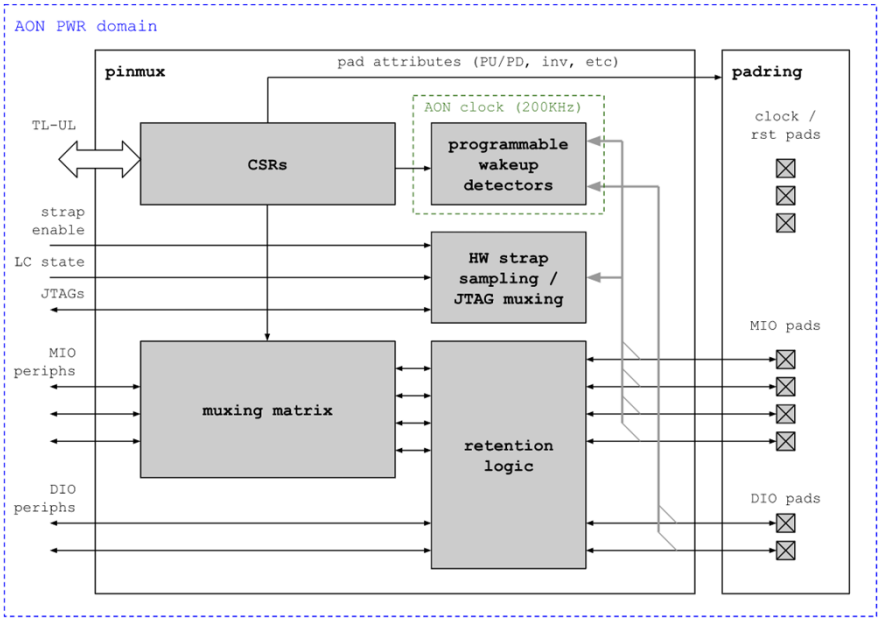
3. 基于 G43 计算出 R43,B43（同方法 2）

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。

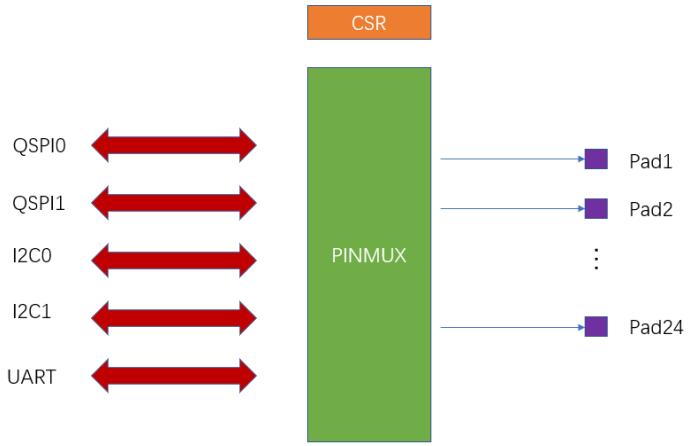
6. PINMUX

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。

7. PWM

Pulse Width Modulation

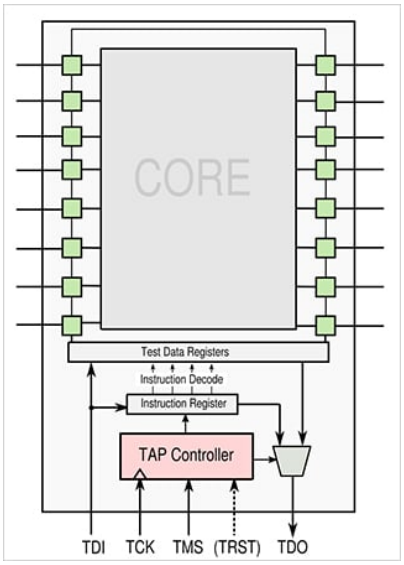
脉冲宽度调制是利用微处理器的数字输出来对模拟电路进行控制的一种非常有效的技术，广泛应用在从测量、通信到功率控制与变换的许多领域中。

通过脉冲调制，可以产生不同占空比的波形。

SOC3.0 项目中有 4 路 PWM， 下面是 SOC3.0 PWM 的寄存器。pulse_width 是 pulse 的宽度，counter_max 是 PWM 周期。

AddrH 基地址	AddrL	Register	Field	Bit	Attribute	Reset Value
0x2007_0000	0x00	PWD0	pulse_width	[31:0]	RW	0x0
	0x04	PWD0	counter_max	[31:0]	RW	0x0
	0x08	PWD1	pulse_width	[31:0]	RW	0x0
	0x0C	PWD1	counter_max	[31:0]	RW	0x0
	0x10	PWD2	pulse_width	[31:0]	RW	0x0
	0x14	PWD2	counter_max	[31:0]	RW	0x0
	0x18	PWD3	pulse_width	[31:0]	RW	0x0
	0x1C	PWD3	counter_max	[31:0]	RW	0x0

8. JTAG

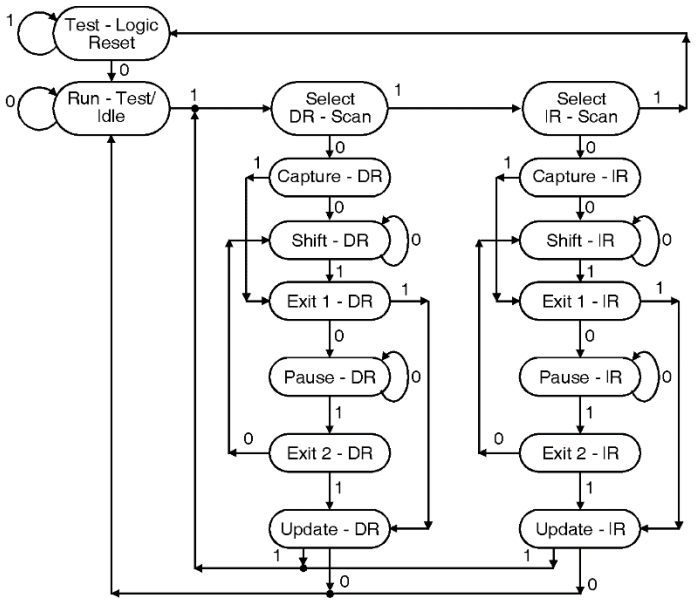


JTAG 架构

JTAG 是一种国际标准测试协议（IEEE 1149.1 兼容），主要用于芯片内部测试。现今多数的高级器件都支持 JTAG 协议，如 DSP、FPGA、ARM、部分单片机器

件等。标准的 JTAG 接口是 4 线：TMS、TCK、TDI、TDO,分别为模式选择、时钟、数据输入和数据输出线。 相关 JTAG 引脚的定义为：TCK 为测试时钟输入；TDI 为测试数据输入，数据通过 TDI 引脚输入 JTAG 接口；TDO 为测试数据输出，数据通过 TDO 引脚从 JTAG 接口输出；TMS 为测试模式选择，TMS 用来设置 JTAG 接口处于某种特定的测试模式；TRST 为测试复位，输入引脚，低电平有效。

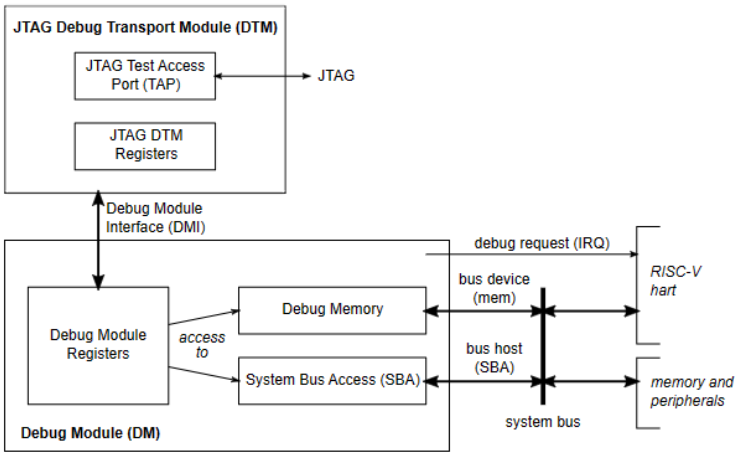
下图是 JTAG 的状态机，通常使用两个最重要的状态是 Shift-DR 和 Shift-IR。
IR：命令寄存器，你可以写值到这个寄存器中通知 JTAG 干某件事。每个 TAP 只有一个 IR 寄存器而且长度是一定的。
DR：TAP 可以有多个 DR 寄存器，与 IR 寄存器相似，每个 IR 值会选择不同的 DR 寄存器。



JTAG 状态机

DM(debug module)

DM 称为调试模块，该电路集成在芯片内部，方便开发者编程，调试，异常处理等功能。已经是芯片不可或缺的电路。下图是 RISCV 的 debug module



关于该 IP 的详细内容，可以参考下面

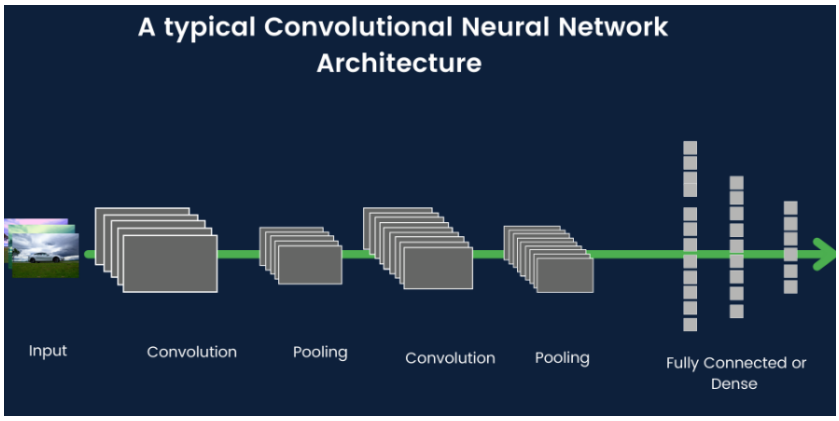
IP 地址 https://github.com/pulp-platform/riscv-dbg/

IP 文档：https://github.com/pulp-platform/riscv-dbg/blob/master/doc/debug-system.md

9. CNN

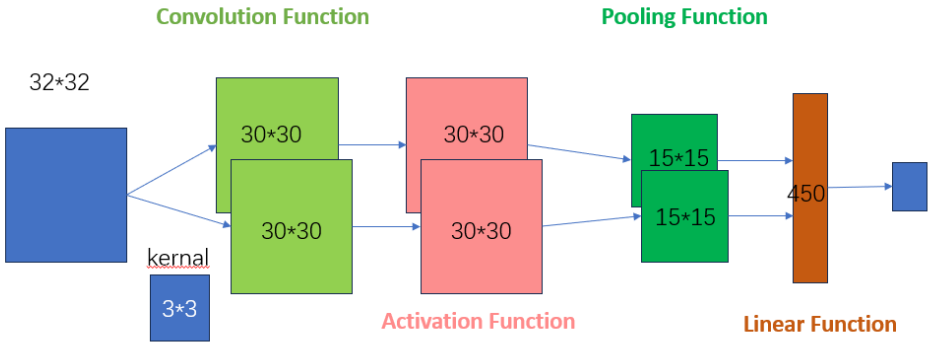
CNN 是神经网络的简称。

一个典型的神经网络如下图所示

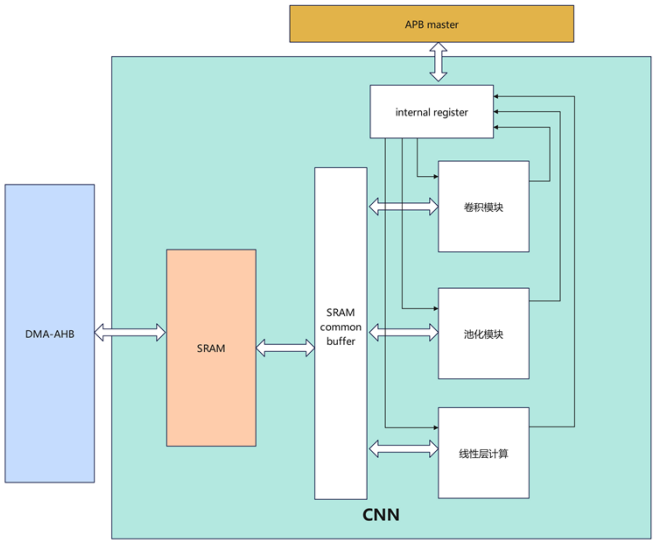


分为卷积层，池化层，连接层。

在我们 SOC3.0 的项目里面，卷积网络如下图所示。



SOC3.0 CNN 框架



SOC3.0 CNN 将卷积参数放在 SRAM 中，通过 DMA 获取 CNN 所需参数，CNN 在 SRAM 中数据结构如下，

SRAM 粗略分段		分段	备注
[1023:0]	data	数据段	1024
[2823:1024]	data after convolution		1800
[3273:2824]	data after max pooling		450
[4187:4186]	data after line		2
[3281:3274]	卷积层参数	参数段	18
[3283:3282]	卷积层偏置		2
[4183:3284]	线性层权重		900
[4185:4184]	线性层偏置		2

CNN 寄存器信息，参考 SOC 寄存器列表。

二 SOC 介绍

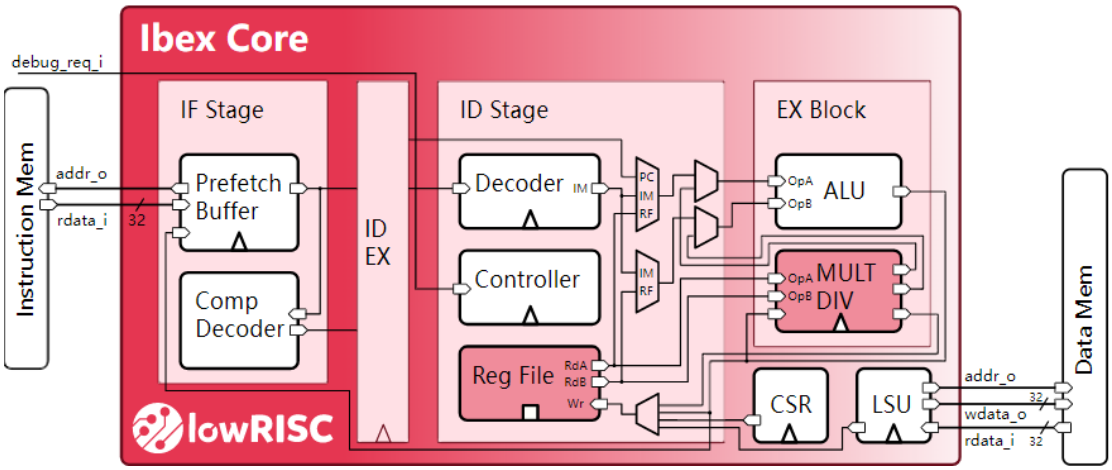
1、功能列表

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/PWM/JTAG/CNN
- 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：

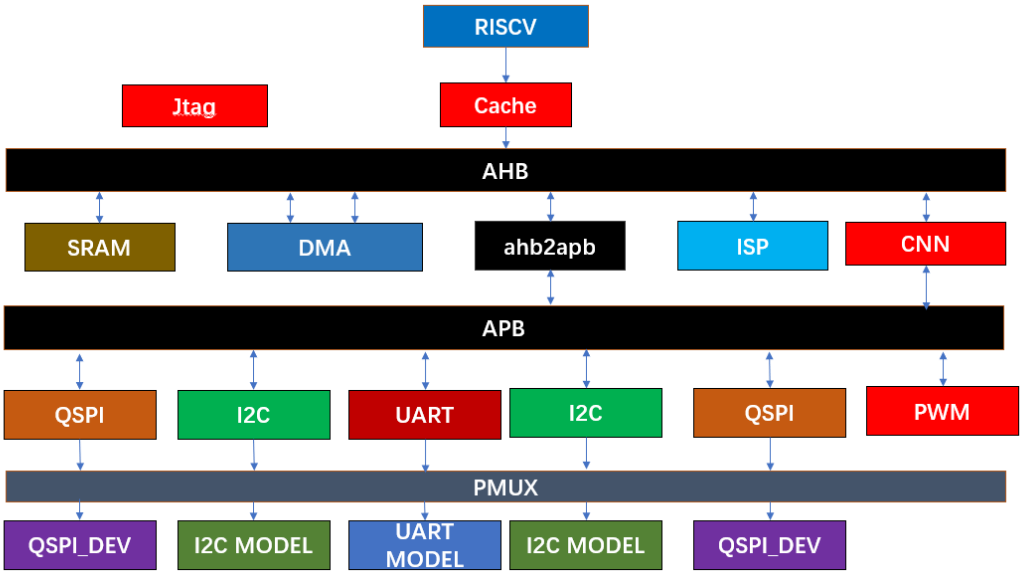
[ibex/vendor/lowrisc_ip/dv/sv/mem_model at master · lowRISC/ibex · GitHub]

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，PWM,JTAG,CNN 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。根据不同的设定可以进行修改。
Pad_2	1	IO	
Pad_3	1	IO	
Pad_4	1	IO	
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。

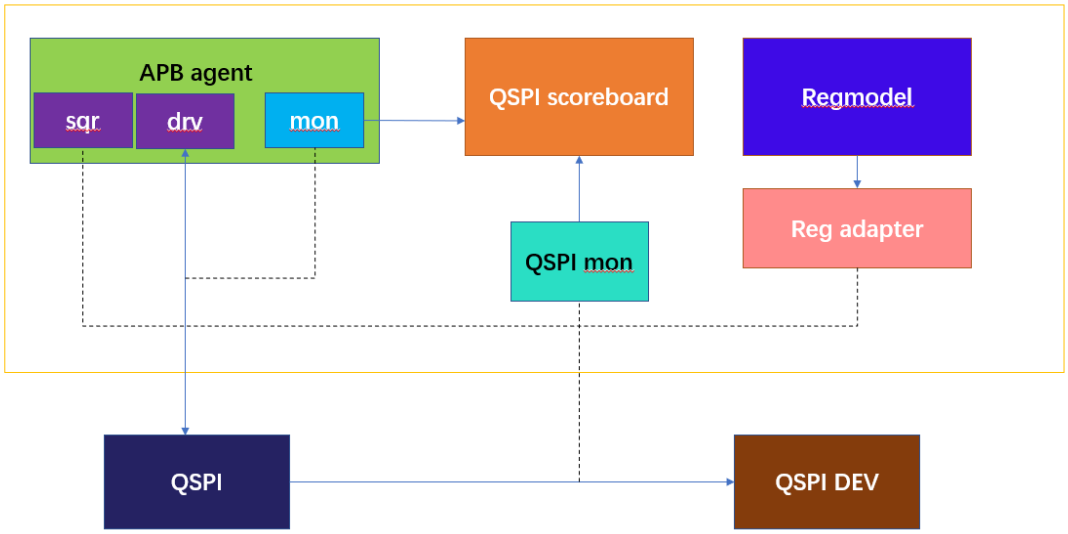
7. memory 空间分配

AHB	SRAM	32'h0000_0000~32'h007f_ffff
	APB	32'h2000_0000~32'h2010_ffff
	DMA	32'h3000_0000~ 32'h3010_ffff
	ISP	32'h40000000~32'h4010ffff
	DM	32'h5000_2000 ~32'h5010ffff
APB	SPI0	32'h20000000~ 32'h2000FFFF
	UART0	32'h20010000~ 32'h2001FFFF
	GPIO	32'h20020000~ 32'h2002FFFF
	I2C0	32'h20030000 ~ 32'h2003FFFF
	SPI1	32'h20040000~ 32'h2004FFFF
	I2C1	32'h20050000~ 32'h2005FFFF
	PINMUX	32'h20060000~ 32'h2006FFFF
	PWM	32'h20070000~ 32'h2007FFFF
	CNN	32'h20080000 ~ 32'h2008FFFF

三. 验证环境

1. IP 验证环境

QSPI 的验证环境



- 文件夹介绍：

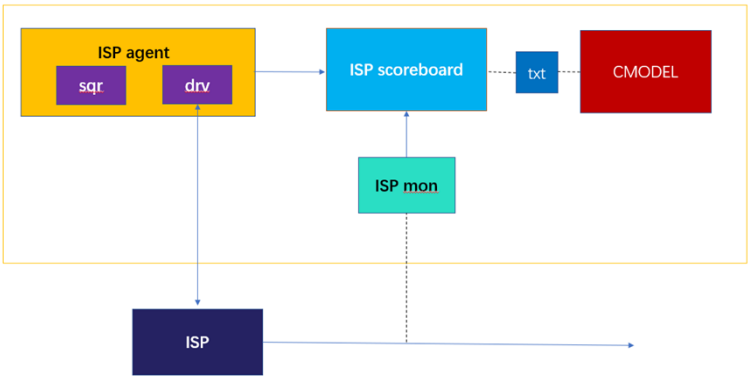
QSPI/tb/env 各种 QSPI testbench 的 组件
QSPI/tb/tests 测试用例
QSPI/sim 跑 simulation 的目录

- 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:

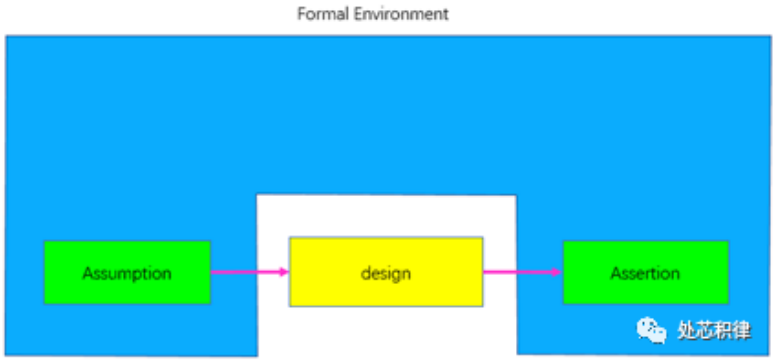
ISP/tb/env 各种 QSPI testbench 的 组件
ISP/tb/tests 测试用例
ISP/sim 跑 simulation 的目录
ISP/cmodel 放 ISP cmodel 的源代码

- 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

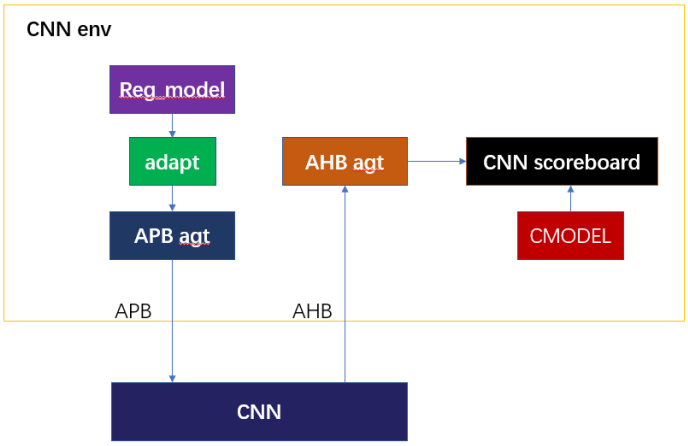
ISP/FPV/assertion 放 assertion 和 assumption 的地方

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

➢ 命令：

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

CNN 的验证环境



➢ 文件夹介绍：

CNN/tb/env 各种 CNN testbench 的 组件

CNN/tb/tests 测试用例

CNN/sim 跑 simulation 的目录

CNN/cmodel 放 CNN cmodel 的源代码

➢ 命令：

编译命令 make comp

运行指令 make run_{test_name}

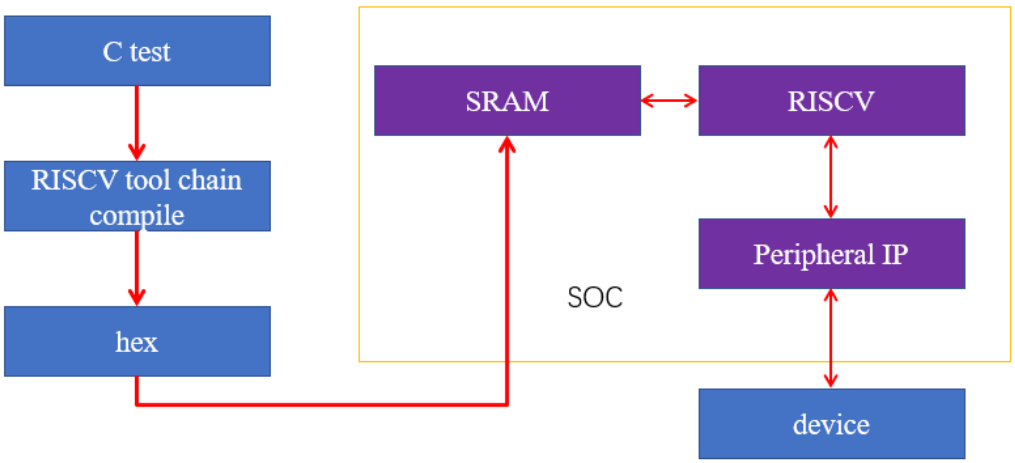
Example : make run_cnn_test

大家根据自己需求更改 makefile 文件

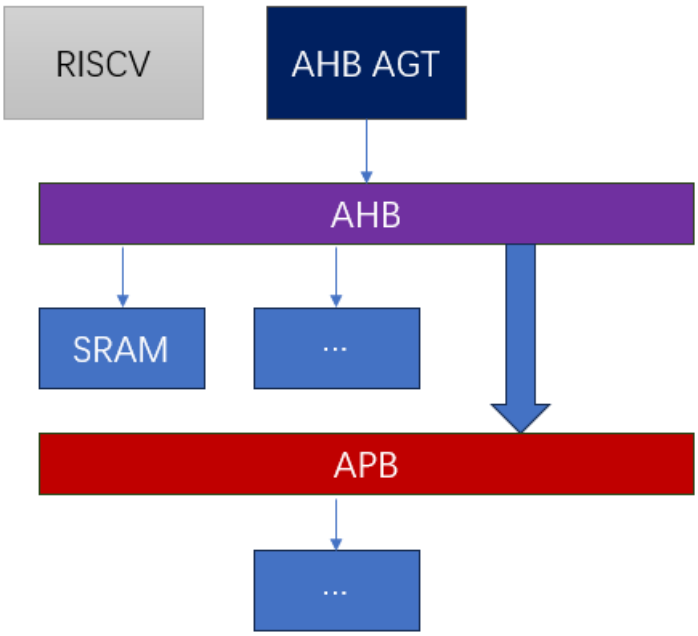
2. SOC 验证环境

仿真环境框图

SOC C 环境



SOC UVM 环境 用 AHB AGT 代替 RISCV



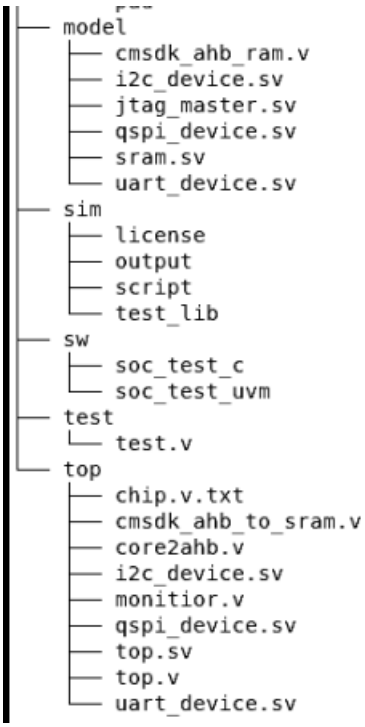
SOC 环境文件夹介绍：

```
.
├── env
│   ├── agt_mem_bus
│   ├── soc_cfg.sv
│   ├── soc_env.sv
│   ├── soc_pkg.sv
│   ├── soc_sqr.sv
│   ├── soc_tb_mod.sv
│   └── soc_test.sv
├── filelist
│   ├── apb_filelist.f
│   ├── busmatrix_filelist.f
│   ├── cnn_sram.f
│   ├── design_top.f
│   ├── dma.f
│   ├── dm.f
│   ├── ibex_core.f
│   ├── ibex_core_other.f
│   ├── isp.f
│   ├── model.f
│   ├── top.f
│   └── uvm_tb.f
```

env: SOC UVM 的环境
filelist : 环境的 filelist

```
├── ibex
│   ├── azure-pipelines.yml
│   ├── build
│   ├── check_tool_requirements.core
│   ├── ci
│   ├── CONTRIBUTING.md
│   ├── CREDITS.md
│   ├── doc
│   ├── dv
│   ├── examples
│   ├── formal
│   ├── ibex_configs.yaml
│   ├── ibex_core.core
│   ├── ibex_icache.core
│   ├── ibex_multdiv.core
│   ├── ibex_pkg.core
│   ├── ibex_top.core
│   ├── ibex_top_tracing.core
│   ├── ibex_tracer.core
│   ├── LICENSE
│   ├── lint
│   ├── Makefile
│   ├── python-requirements.txt
│   ├── README.md
│   ├── rtl
│   ├── rtl_test
│   ├── shared
│   ├── src_files.yml
│   ├── syn
│   ├── tool_requirements.py
│   ├── util
│   └── vendor
├── ip
│   ├── apb_subsystem
│   ├── busmatrix
│   ├── cnn_v1
│   ├── dm
│   ├── dma
│   ├── include
│   ├── isp
│   └── pad
```

ibex ： ibex cpu 核
ip ： 外设及总线信息，全部代码可见。



model: 各种外设 和 SRAM 的 model
SW ： 放置各个模块测试用的 C pattern 和 UVM tests
test: 挂载 qspi/uart/i2c 等 model
top: soc 顶层文件，例化各个设计模块。

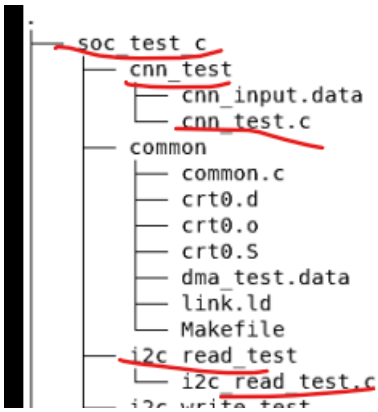
SOC 验证环境命令:
在 SOC 3.0 里面我们提供了 yrun，ycheck，ypgt 工具。

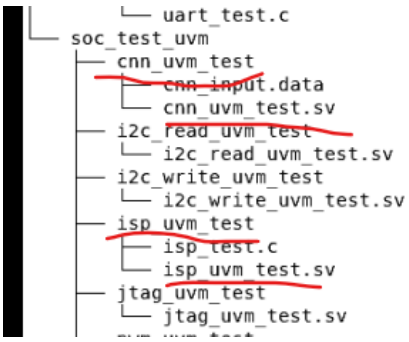
yrun: 提供编译，仿真，跑 regression 等功能。
使用案例: yrun -testfile test_file111
yrun 的 option 有下面这些
-sim_opt help="tmp add sim option from cmd" exmaple : yrun -testfile test_file111 -testname test1 -sim_opt " +dump " to add the +dump plusagrs into the simultion option
-comp_opt help="tmp add sim option from cmd" exmaple : yrun -testfile test_file111 -testname test1 -comp_opt " +define+SOC_SIM " to add the SOC_SIM into the compile option
-repeat help="repeat times" exmaple : yrun -testfile test_file111 -testname test1 -repeat 10 run 10 different seed of this case
-seed help="random seed select " exmaple : yrun -testfile test_file111 -testname test1 -seed 1234 run select 1234 random seed of this case
-tag help="select case with special tag" exmaple : yrun -testfile test_file111 -testname test1 -seed 1234 run select 1234 random seed of this case
-rerun_fail help="rerun the fail case in the result file" exmaple : yrun -rerun_fail <test_report_file>(will print out when the last yrun end)

-coverage help="coverage option" exmaple : yrun -testfile test_file111 -coverage <hier file>(collect what level's coverage)
-postsim help="post gate sim" exmaple : yrun -testfile test_file111 -postsim will auto change the <compile_name> to <compile_name>+"_postsim"
-presim help="pre gate sim" exmaple : yrun -testfile test_file111 -postsim will auto change the <compile_name> to <compile_name>+"_presim"
-fpga help="fpga sim" exmaple : yrun -testfile test_file111 -presim will auto change the <compile_name> to <compile_name>+"_fpga_"
-upf help="upf sim" exmaple : yrun -testfile test_file111 -upf will auto change the <compile_name> to <compile_name>+"_upf"
-sim help="only simulation" exmaple : yrun -testfile test_file111 -sim (skip the compile only run simulation)
-co help="only compile" exmaple : yrun -testfile test_file111 -co (only run compile)

注意:

1. Sw/soc_test_c 或者 sw/soc_test_uvm 下面的文件夹的名字 要和 c 或者 uvm test 的 name 一致,比如 spi_test 那么 spi_test 文件夹下的 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

ycheck：提供查询 regression 状态

用法案例：

ycheck -list

```
[root@myhost V3_PRJ_EXAMPLE]# ycheck -list
latest_date:20231007
file_name:test_file111
dir:/root/pwd/work/github/SOC_V3.0_final6/V3_PRJ_EXAMPLE/work/regression
date:20231007-093622
day:20231007
0
Regression Status
DATE                        TEST_FILE                       DIR                         STATUS                      DETAIL
20231007-093622             _  test_file111                 regression                  done                        ycheck -status test_file111_20231007-093622
```

ycheck -status

```
[root@myhost V3_PRJ_EXAMPLE]#  ycheck -status test_file111_20231007-093622
test_file:test_file111
date_time:20231007-093622
test_name                                                   status
test1                                                       SIM FASS
test2                                                       COMPILE FAIL,fail message: Error-[SE] Syntax error
```

ygpt ：调用 GPT 理解 design ，根据自己需求生成 testplan, testcase 等

用法案例：

```
[root@myhost sim]# ygpt -fname spi_master_rx.v
Please enter your question (or type 'exit' to end): please provide testplan from this code
Answer:
Testplan:

1) Verify that the module correctly handles the reception of data.

-Send data to the module and observe the outputs.
-Compare the expected output with the actual output.

2) Verify that the module correctly handles the reception of data with the quad input enabled.

-Send data to the module with the quad input enabled and observe the outputs.
-Compare the expected output with the actual output.

3) Verify that the module correctly handles the reception of data with the counter input update signal enabled.

-Send data to the module with the counter input update signal enabled and observe the outputs.
-Compare the expected output with the actual output.
Please enter your question (or type 'exit' to end):
```

四. 设计 flow

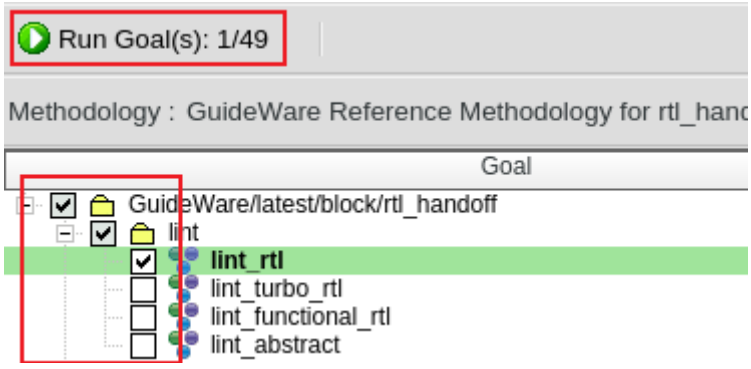
```
    └── tb
├── dc
│   ├── lib
│   ├── Makefile
│   ├── outputs
│   ├── reports
│   ├── scr
│   └── work
├── env_centos.csh
├── env_ubuntu.csh
├── flow
│   ├── dc
│   ├── filelist
│   ├── fm
│   └── spyglass
├── fm
│   ├── fm.log
│   ├── fm.tcl
│   └── Makefile
├── fm_read_sverilog.man
```

在设计 flow 中，我们提供了 spyglass，dc，formal 的脚本。

Flow- Spyglass(lint)

make clean ： 清除 log 和工程文件

make lint ： 启动 spyglass gui，可自行选择目标并运行。



Flow- DC

make clean ： 清楚工作工程文件、输出文件、报告文件。

make syn ： 启动 dc 完成综合

综合所用脚本:

综合步骤 dc/scr/syn.tcl

约束文件 dc/scr/soc_sdc.tcl

综合输出结果:

dc/outputs 包括: 综合后的 netlist/ddc 文件/sdc 文件

综合输出报告:

dc/reports/ 包括: area/power/timing/check_design.

Flow- FM

make clean ： 清楚工作工程文件、输出文件、报告文件。

make rtlvssyn: #rtl vs syn_netlist

make rtlvspnr: # rtl vs pnr_netlist

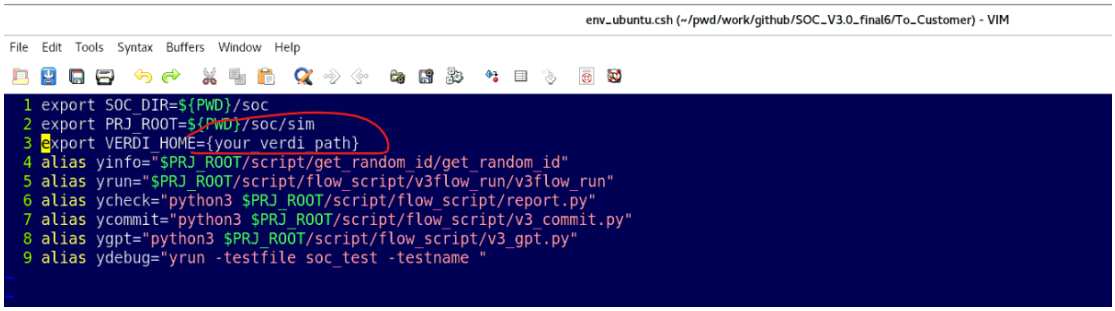
```
make synvspnr: # syn_netlist vs pnr_netlist
```

五.拿到环境后需要做哪些事情?

1. 修改变量

在交付的项目中有 env_ubuntu.csh 和 env_centos.csh 两个文件，根据自己的操作系统设置环境变量。

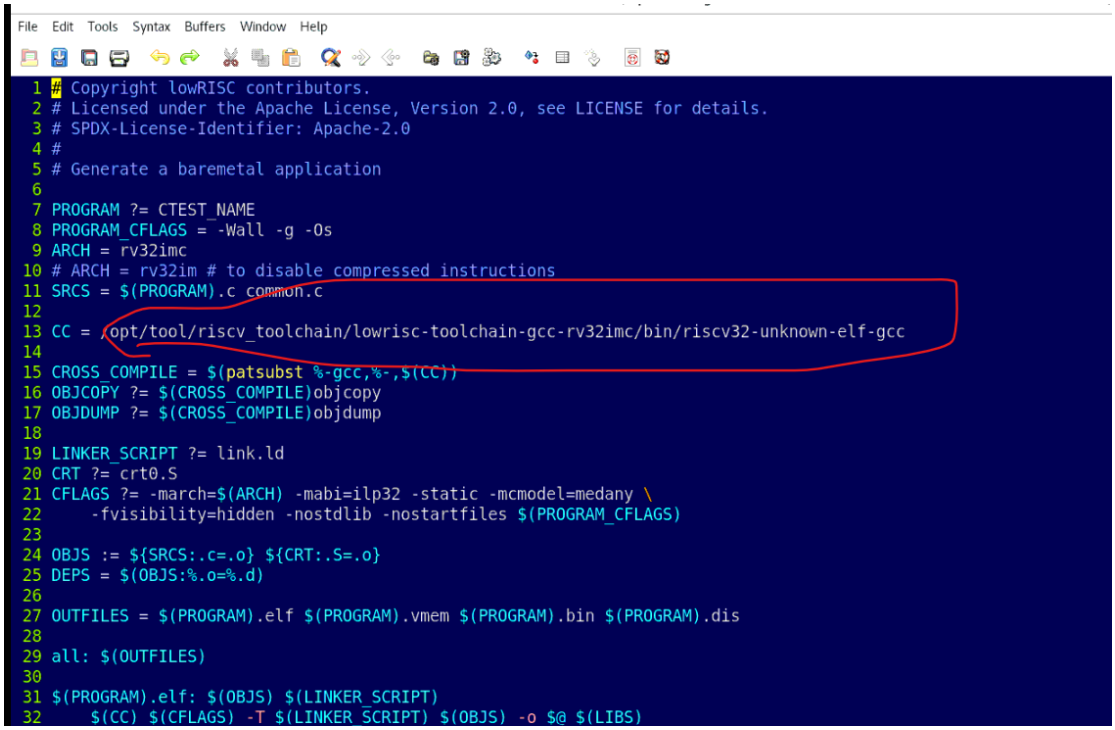
如果设置过 VERDI_HOME，把下面红圈的地方删掉就行，如果没有就需要将下面红圈的地方改成自己的 verdi 路径。



设置 Makefile 里面工具链的路径

在 soc/sw/soc_test_c/common 中找到 Makefile

修改工具链路径



修改图中红色圈起来的地方即可。

2. source 对应自己操作系统的 env_ubuntu.csh/env_centos.csh

```
)st To_Customer]# source env_centos.csh
)st To_Customer]#
```

3. 运行 yinfo ，将打印出来的数字发送给我们。我们会生成一个 signature.bin 的文件给你

```
[root@myhost To_Customer]# yinfo
random_id:
6e4cb90c6874351619a6e062a41547c5af2f7ff319d09639585b4a2e6e7db47e
```

将 signature.bin 的文件放到 soc/sim/license 下面。

4. 运行 ydebug spi1_test 看看有没有正常跑起来。

```
[root@myhost sim]# ydebug spi1_test
License is valid.
In pre_process now.
mkdir: cannot create directory ‘/root/pwd/work/github/SOC_V3.0_final6/To_Customer_ok/soc/sim/output’: File exists
In pre_process step2.
work_location: /root/pwd/work/github/SOC_V3.0_final6/To_Customer_ok/soc/sim/output
test_file_name: soc_test
test_name: spi1_test
/opt/tool/riscv_toolchain/lowrisc-toolchain-gcc-rv32imc/bin/riscv32-unknown-elf-gcc -march=rv32imc -mabi=ilp32 -static -mcmodel=medany -fvisib
st.o spi1_test.c
spi1_test.c: In function 'main':
spi1_test.c:40:30: warning: initialization of 'volatile int *' from incompatible pointer type 'volatile uint8_t *' {aka 'volatile unsigned cha
   40 |   volatile int  *uart_base = (volatile uint8_t *)  0x20010000;
      |                              ^
spi1_test.c:41:30: warning: initialization of 'volatile int *' from incompatible pointer type 'volatile uint8_t *' {aka 'volatile unsigned cha
   41 |   volatile int  *qspi_base = (volatile uint8_t *)  0x20000000;
      |                              ^
spi1_test.c:60:3: warning: implicit declaration of function 'write' [-Wimplicit-function-declaration]
   60 |   write(0x90008,2); // jtag start
      |   ^~~~~
spi1_test.c:62:3: warning: implicit declaration of function 'read' [-Wimplicit-function-declaration]
   62 |   read(0x20060004,&rdata);
      |   ^~~~
spi1_test.c:64:7: warning: implicit declaration of function 'puts' [-Wimplicit-function-declaration]
   64 |       puts("Get correct rdata \n");
      |       ^~~~
At top level:
spi1_test.c:33:12: warning: 'usleep' defined but not used [-Wunused-function]
   33 | static int usleep(unsigned long usec) {
      |            ^~~~~~
/opt/tool/riscv_toolchain/lowrisc-toolchain-gcc-rv32imc/bin/riscv32-unknown-elf-gcc -march=rv32imc -mabi=ilp32 -static -mcmodel=medany -fvisib
o common.c
common.c:4:28: warning: initialization of 'volatile int *' from incompatible pointer type 'volatile uint8_t *' {aka 'volatile unsigned char *'
    4 | volatile int  *uart_base = (volatile uint8_t *)  0x20010000;
      |                            ^
/opt/tool/riscv_toolchain/lowrisc-toolchain-gcc-rv32imc/bin/riscv32-unknown-elf-gcc -march=rv32imc -mabi=ilp32 -static -mcmodel=medany -fvisib
t.o common.o crt0.o -o spi1_test.elf
```