天机x芯片控制器设计-微码版本与非微码版本与微码列表

日期：2022-10-25

编写人：类脑工程团队

第0期开发说明：

该版本控制器主要是基于天机2芯片控制器逻辑，继承了原有代码的主体结构，对代码风格进行调整，增加适配天机X芯片的数据传输和时序控制逻辑，可在多种形态的硬件平台上迁移，支撑芯片测试和验证。

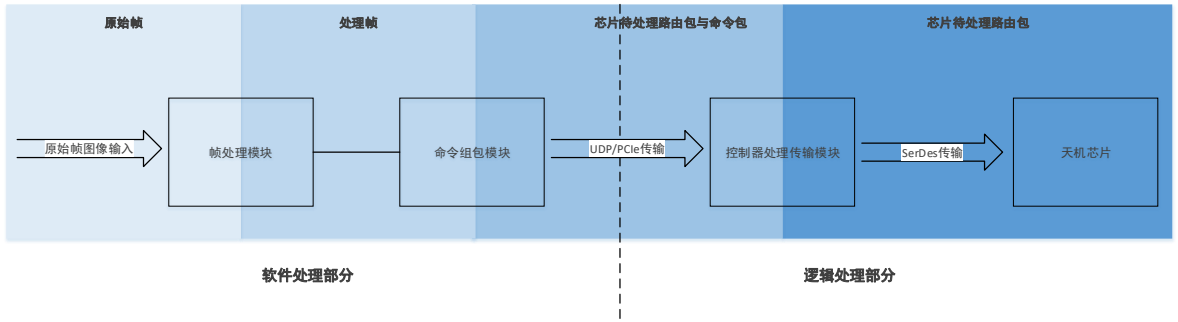
后续版本章节均在该基本功能版本上做补充说明，本章留存作为基本模块简述使用。

第1期开发说明：

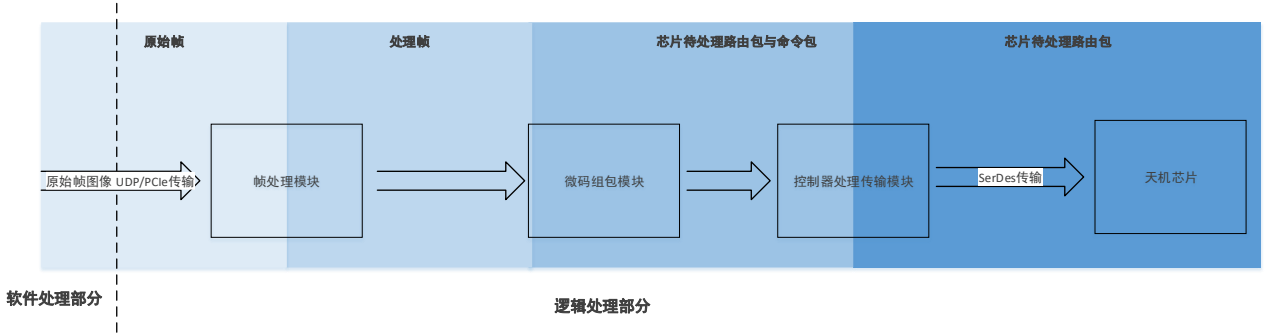
该版本控制器用于适配具体的无人车的算法应用，完成多种数据源的分类，优先级下发和单芯片的时序交互控制。主要在原路由数据包的保留位上定义了控制信息，用于区分不同的网络，标记Step起始结束等时序信息，执行不同的控制交互操作等。

第2期开发说明：

在第1期控制器的基础上，增加了FPGA端路由数据组包的功能，引用了微码机制实现该功能。前端DVPP模块将处理好的像素数据包缓存下来，组包模块依次读取微码并执行相应操作，组包时即从DVPP读取像素数据包，控制交互指令即等待交互完成后再执行下一条。



第1期开发说明



第2期控制器

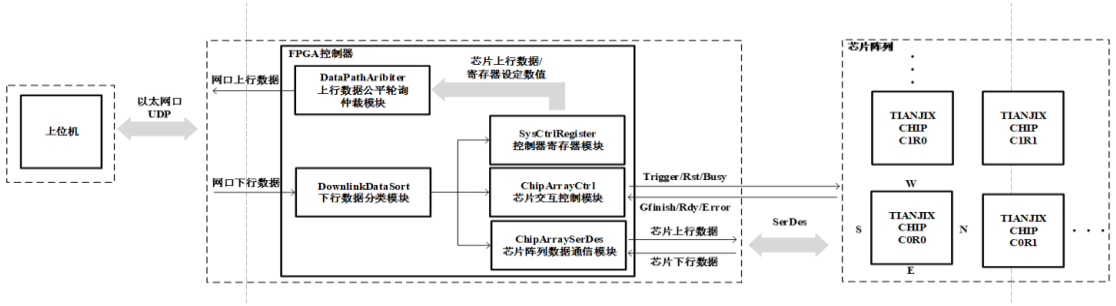


Figure 1.第1期逻辑实现框图

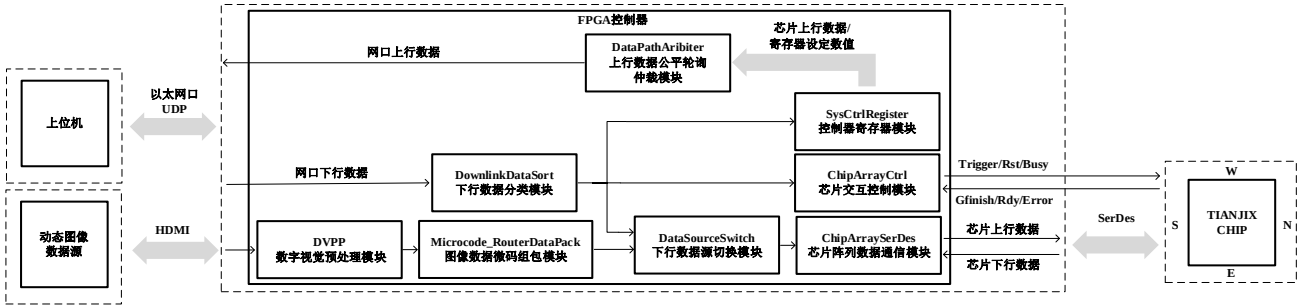


Figure 2. 第2期VA、VC版本逻辑实现框图

	tj-controllor ta	mc-tj-controllor mc
中文名称	无微码天机控制器	微码天机控制器
不同点关系	无微码天机控制器+微码控制器=微码天机控制器	
涉及新的组包概念	无微码天机控制器时序控制包（时序控制与天机路由混合包）	数据源时序微码控制指令包 微码数据包

概念：

（1）无微码天机控制器时序控制包：无微码的时序控制器包，用于无微码状态，可以称为**时序控制与天机路由混合包**：该协议包是多个协议包组成，目的端是时钟精确级别控制器，包括微码生成的控制时钟精确级别的时序控制和天机包

（2）**数据源时序微码控制指令包**：请按照微码列表和微码控制包以及算法时序组<数据源时序微码控制指令包>，用于微码数据包传输；

（3）**微码数据包**：请按照微码文档中的数据包格式组<微码数据包>，用于微码数据包传输；

（4）天机路由数据包：请参考天机文档实现。

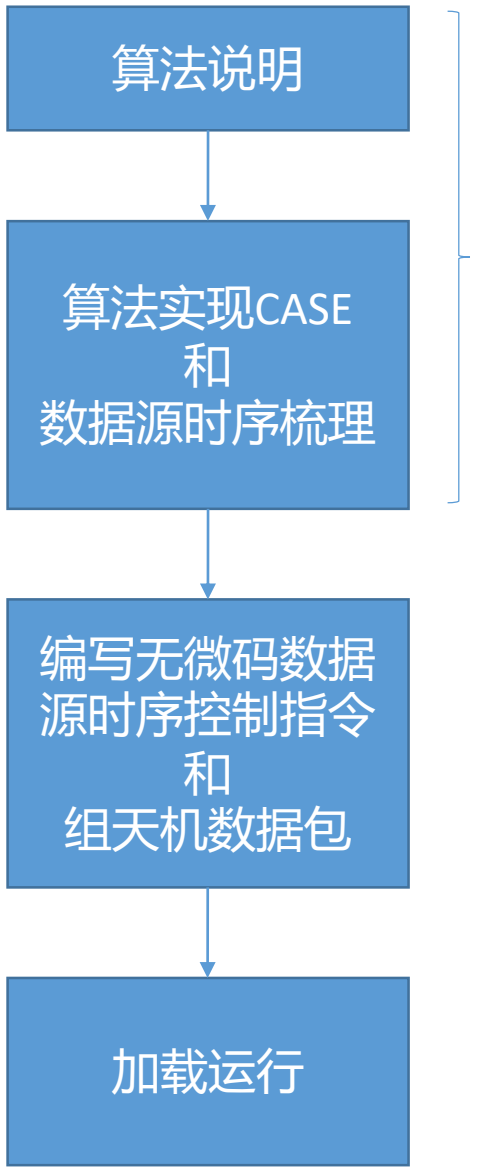
（5）原始数据源+处理后数据源+整形后数据源：对数据源不同性质的理解

（6）天机数据源处理时序流程特征：按照算法的时序图

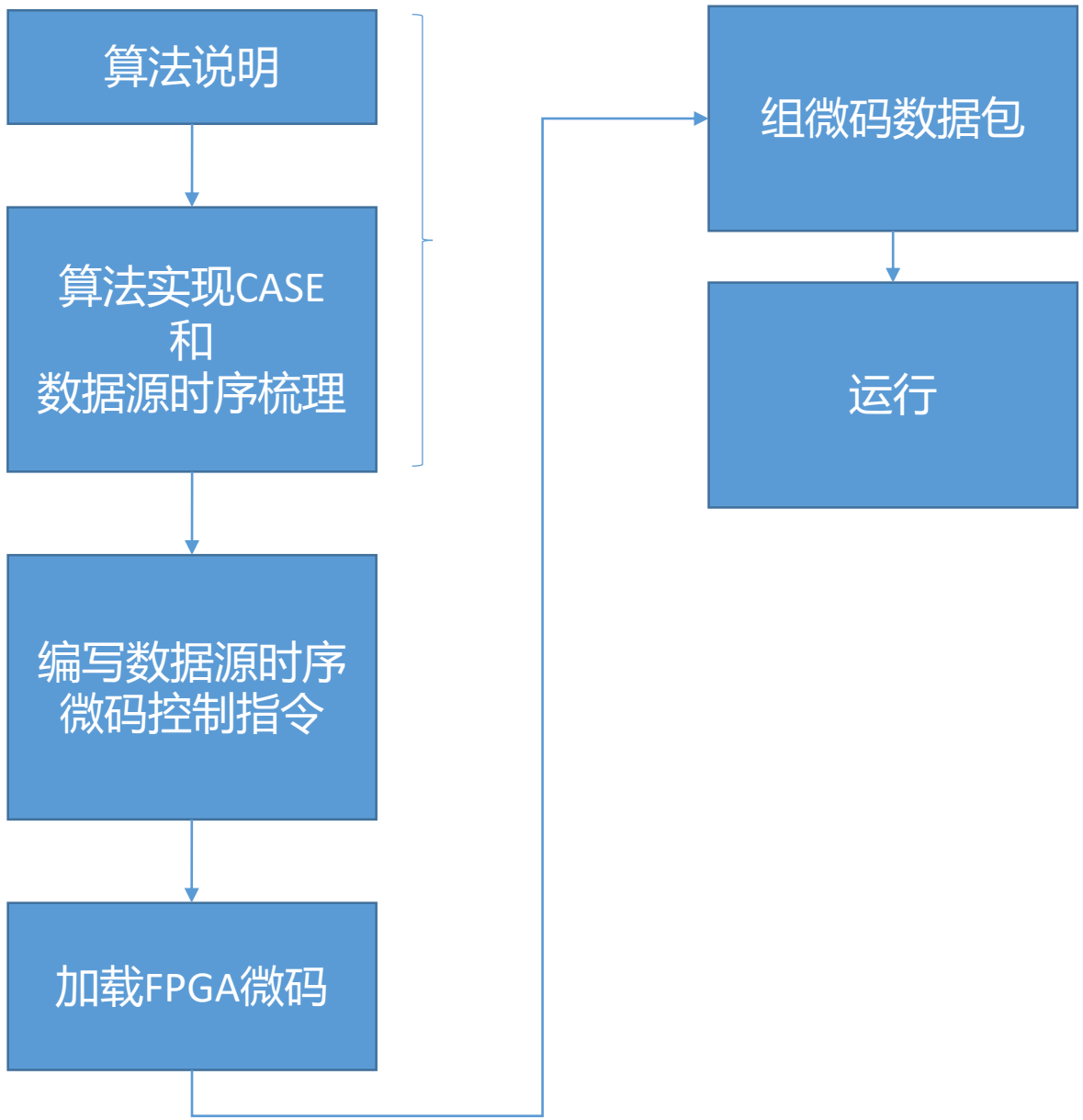
（7）绘制<Step-Phase-core-part图-SPCP图>：按照天机数据源处理时序流绘制的图形

配置流程

tj-controllor



mc-tj-controllor



仿真器和天机芯片的实际测试

问题描述：一般来讲，目前的时钟精确级仿真器不是严格精确地，导致仿真器跑出的case可能是实际硬件运行的60%-80%

手段：需要将仿真器生成的case放置于实际的天机芯片上，并提取硬件上实际运行的trigger和phase等信号和运行时间，将该运行时间手动回补到case的生成阶段

目标：通过调整使得最终生成的case_cfg_file能够在硬件上正常运行

备注： trigger和phase等信号和运行时间可以采用示波器，目前也可以采用软件的方式（实际上逻辑已支持该特征）

请参考《天机时序控制器操作步骤20220420----zlb.docx 》

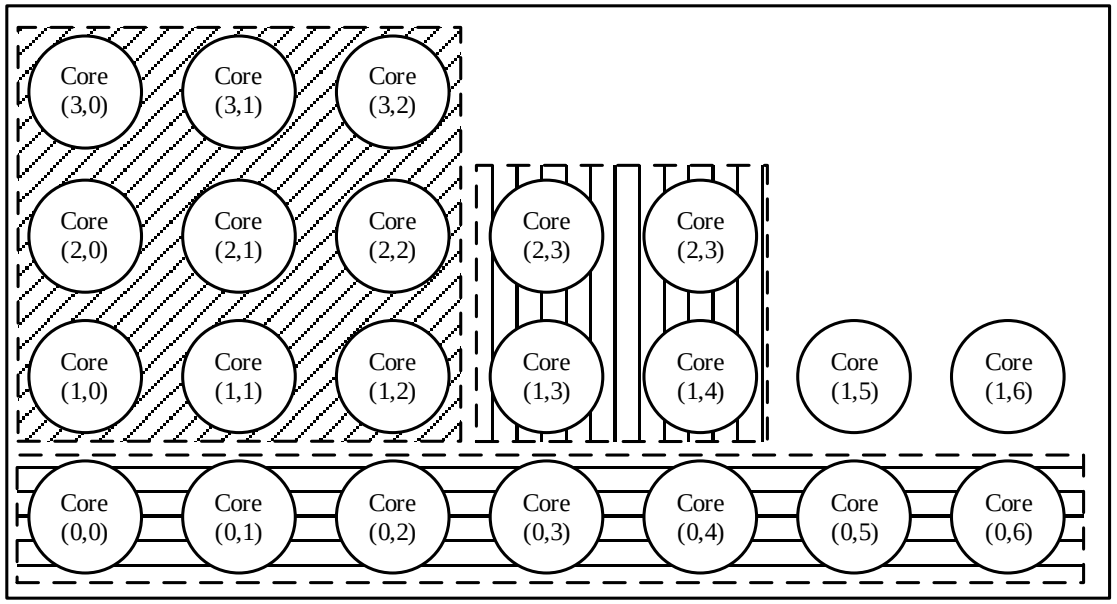
天机X芯片拥有10行16列共160个Core，每个Core计算的时间单位称之为Step，每个Step可进一步划分为多个Phase，可以说芯片运行的最小时间单位为Phase。

每个Core可划分到不同的Step Group来进行多核协同工作，一个芯片最多划分为4个Step Group，对应芯片的4个Trigger引脚来触发其运行。每个Step Group运行一次称之为一个Step。每个Step Group又可划分为多个Phase Group。一个芯片最多可划分为32个Phase Group。每个Phase Group运行一次称之为一个Phase。每个Phase运行结束的标志可通过配置芯片寄存器5来设置输出至芯片的4个Gfinish引脚，但芯片仅有Gfinish 0 – 3这四个引脚，故在同一时刻最多仅能监控4个Phase Group的运行情况。

【Step Group：core工作组最多4个，空间分组，】

【Phase Group：时段工作组，每个core组最多可设置32段连续工作时序】

下图所示的Core分别划分到3个Step Group中，分别用不同的底纹标出。



天机运行时长寄存器

Name	Address
ADDR_C0R0_GFINISH_0_PHASE_00_TIME	25'h000_1100
ADDR_C0R0_GFINISH_0_PHASE_01_TIME	25'h000_1101
ADDR_C0R0_GFINISH_0_PHASE_02_TIME	25'h000_1102
ADDR_C0R0_GFINISH_0_PHASE_03_TIME	25'h000_1103
ADDR_C0R0_GFINISH_0_PHASE_04_TIME	25'h000_1104
ADDR_C0R0_GFINISH_0_PHASE_05_TIME	25'h000_1105
ADDR_C0R0_GFINISH_0_PHASE_06_TIME	25'h000_1106
ADDR_C0R0_GFINISH_0_PHASE_07_TIME	25'h000_1107
ADDR_C0R0_GFINISH_0_PHASE_08_TIME	25'h000_1108
ADDR_C0R0_GFINISH_0_PHASE_09_TIME	25'h000_1109
ADDR_C0R0_GFINISH_0_PHASE_10_TIME	25'h000_110A
ADDR_C0R0_GFINISH_0_PHASE_11_TIME	25'h000_110B
ADDR_C0R0_GFINISH_0_PHASE_12_TIME	25'h000_110C
ADDR_C0R0_GFINISH_0_PHASE_13_TIME	25'h000_110D
ADDR_C0R0_GFINISH_0_PHASE_14_TIME	25'h000_110E
ADDR_C0R0_GFINISH_0_PHASE_15_TIME	25'h000_110F
ADDR_C0R0_GFINISH_0_PHASE_16_TIME	25'h000_1110
ADDR_C0R0_GFINISH_0_PHASE_17_TIME	25'h000_1111
ADDR_C0R0_GFINISH_0_PHASE_18_TIME	25'h000_1112
ADDR_C0R0_GFINISH_0_PHASE_19_TIME	25'h000_1113
ADDR_C0R0_GFINISH_0_PHASE_20_TIME	25'h000_1114
ADDR_C0R0_GFINISH_0_PHASE_21_TIME	25'h000_1115
ADDR_C0R0_GFINISH_0_PHASE_22_TIME	25'h000_1116
ADDR_C0R0_GFINISH_0_PHASE_23_TIME	25'h000_1117
ADDR_C0R0_GFINISH_0_PHASE_24_TIME	25'h000_1118
ADDR_C0R0_GFINISH_0_PHASE_25_TIME	25'h000_1119
ADDR_C0R0_GFINISH_0_PHASE_26_TIME	25'h000_111A
ADDR_C0R0_GFINISH_0_PHASE_27_TIME	25'h000_111B
ADDR_C0R0_GFINISH_0_PHASE_28_TIME	25'h000_111C
ADDR_C0R0_GFINISH_0_PHASE_29_TIME	25'h000_111D
ADDR_C0R0_GFINISH_0_PHASE_30_TIME	25'h000_111E
ADDR_C0R0_GFINISH_0_PHASE_31_TIME	25'h000_111F

芯片C0R0的Gfinish 0 的Phase 0 – 31的寄存器名称与地址的对应关系如下表所示，Gfinish 1的Phase 0 – 31的寄存器地址从地址25'h000_1200开始累加，Gfinish 2的Phase 0 – 31的寄存器地址从地址25'h000_1300开始累加，Gfinish 3的Phase 0 – 31的寄存器地址从地址25'h000_1400开始累加，此处不再赘述。

无微码天机控制器
(时序控制与天机路由混合包)

无微码天机控制器时序控制包

在天机芯片的路由包基础上，FPGA检查其10bit保留字段，并进行相应的芯片控制操作，如启动计算，监控运行状态等。在原有天机芯片的路由包字段定义的基础上，新增控制字段在下表中标红。

Symbol	M	CoreID	Data Type	Data Ctrl	Rsv	Step Group ID	Payload	CRC
Width	2b	4b	2b	4b	2b	2b	96b	16b
Value	11	/	00	/	/	/	/	/

其中Data Ctrl字段的定义如下所示：
Step开始标志: 4’b1000 = 4’h8
Step结束标志: 4’b1001 = 4’h9
每个Step的用时反馈标志：4’b1010 = 4’hA

触发Trigger标志：4’b0100 = 4’h4
等到Gfinish标志：4’b0101 = 4’h5

Phase开始标志：4’b0001 = 4’h1
Phase数据标志：4’b0011 = 4’h3
Phase结束标志：4’b0010 = 4’h2

读完Step RAM反馈标志：4’b1011 = 4’hB
写完Step RAM反馈标志：4’b1100 = 4’hC
Step数据异常错误反馈标志 堵了：4’b1101 = 4’hD
Step数据异常错误反馈标志 丢了：4’b1110 = 4’hE

Data Type设置为0
Step Group ID适用于算法为多Step Group的情况，若为单Step Group 则置为0即可

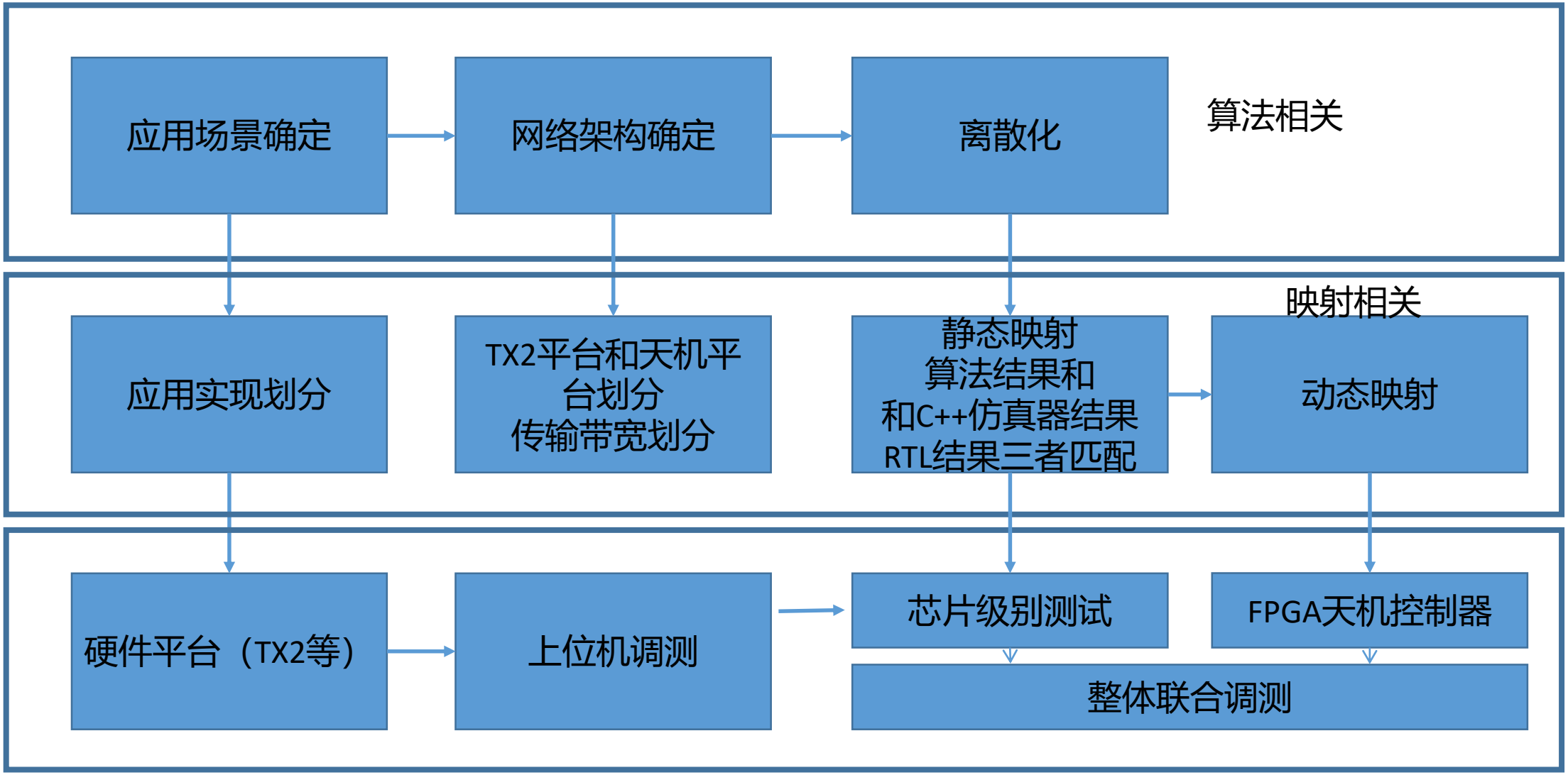
举例说明

Resnet50的运行说明：对于9分类case, 共需运行9个step,每个step运行6个phase，则每个step待处理数据按如下流程组包：

组包类别	备注	
STEP_S_CMD	step起始指令	无微码天机控制器时序控制包
TRIGGER_CMD	trigger拉起指令	无微码天机控制器时序控制包
GFINISH_CMD	空phase的Gfinish等待指令	无微码天机控制器时序控制包
PHASE_0_S_CMD	送数phase0起始指令	无微码天机控制器时序控制包
PHASE_0_DATA	送数phase0所需数据	天机包
PHASE_0_E_CMD	送数phase0结束指令	无微码天机控制器时序控制包
GFINISH_CMD	phase0的Gfinish等待指令	无微码天机控制器时序控制包
PHASE_1_S_CMD	送数phase1起始指令	无微码天机控制器时序控制包
PHASE_1_DATA	送数phase1所需数据	天机包
PHASE_1_E_CMD	送数phase1结束指令	无微码天机控制器时序控制包
GFINISH_CMD	phase1的Gfinish等待指令	无微码天机控制器时序控制包
PHASE_2_S_CMD	送数phase2起始指令	无微码天机控制器时序控制包
PHASE_2_DATA	送数phase2所需数据	天机包
PHASE_2_E_CMD	送数phase2结束指令	无微码天机控制器时序控制包
GFINISH_CMD	phase2的Gfinish等待指令	无微码天机控制器时序控制包
GFINISH_CMD	运算phase的Gfinish等待指令	无微码天机控制器时序控制包
GFINISH_CMD	运算phase的Gfinish等待指令	无微码天机控制器时序控制包
STEP_E_CMD	step终止指令	无微码天机控制器时序控制包

微码天机控制器

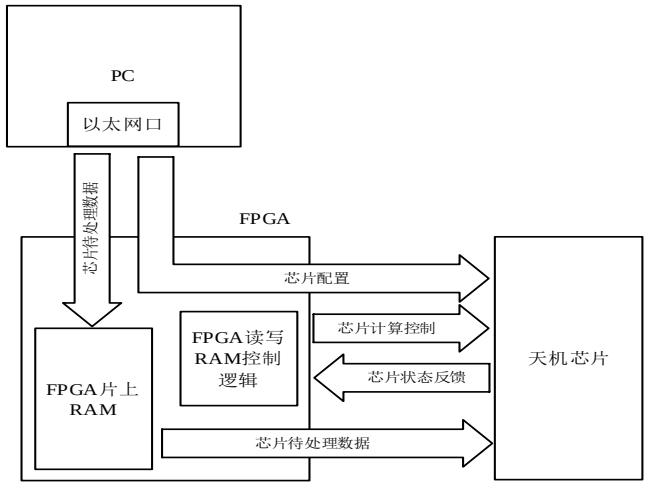
开发流程


应用场景确定
网络架构确定
离散化
算法相关
应用实现划分
TX2平台和天机平台划分
传输带宽划分
静态映射
算法结果和
和C++仿真器结果
RTL结果三者匹配
映射相关
动态映射
硬件平台（TX2等）
上位机调测
芯片级别测试
FPGA天机控制器
整体联合调测

概念说明

1. 部署用户需要提供天机配置文件：<tianji-cfg-file.txt>
2. 部署用户需要提供数据源-分为原始数据源+处理后数据源+整形后数据源
3. 部署用户需要提供：天机数据元处理时序流程特征并绘制Step-Phase-core-part图，和整形后数据源以及微码列表和微码相关包共同组成：微码指令生成文件

微码列表



Symbol	Location	Width	Value	Function
MC	[47:46]	2b	2'b10	微码起始标志
			2'b00	微码
			2'b01	微码结束标志
Pack	[39:36]	4b	4'b0110	生成Step起始标志
			4'b0101	生成Step结束标志
			4'b0010	生成Phase起始标志
			4'b0001	生成Phase结束标志
			4'b1000	生成Trigger触发标志
			4'b1001	生成等待Gfinish标志
			4'b0011	Phase数据组包

按照列表说明，其中微码共计7条微码，包括

1. 生成step起始标志
2. 生成step结束标志
3. 生成phase结束标志
4. 生成phase结束标志
5. 生成trigger触发标志
6. 生成等待gfinish标志
7. phase数据组包

微码天机控制器时序控制包

MC	Rsv	Pack	CoreID	S	T	P	Q	X	Y	A
2b	6b	4b	4b	1b	1b	1b	1b	8b	8b	12b
/	/	/	/	0	1	0/1	0	/	/	/

其中CoreID、S、T、P、Q、X、Y、A均为可参考《TianjicX1概要设计_20201128》中路由数据格式说明章节相关描述。

Symbol	Location	Width	Value	Function
MC	[47:46]	2b	2’b10	微码起始标志
			2’b00	微码
			2’b01	微码结束标志
Pack	[39:36]	4b	4’b0110	生成Step起始标志
			4’b0101	生成Step结束标志
			4’b0010	生成Phase起始标志
			4’b0001	生成Phase结束标志
			4’b1000	生成Trigger触发标志
			4’b1001	生成等待Gfinish标志
			4’b0011	Phase数据组包

MicroCode_Start :128'h1200000000000000_8_0_0_0_0_00_00_000_F0F0
MicroCode_End :128'h1200000000000000_4_0_0_0_0_00_00_000_F0F0
Step_Start :128'h1200000000000000_0_0_6_0_0_00_00_000_F0F0
Step_End :128'h1200000000000000_0_0_5_0_0_00_00_000_F0F0
Trigger :128'h1200000000000000_0_0_8_0_0_00_00_000_F0F0
Phase_Start :128'h1200000000000000_0_0_2_0_0_00_00_000_F0F0
Phase_End :128'h1200000000000000_0_0_1_0_0_00_00_000_F0F0
Gfinish :128'h1200000000000000_0_0_9_0_0_00_00_000_F0F0
Phase_Data :128'h1200000000000000_0_0_3_(CoreID)_(STPQ)_(X)_(Y)_(Addr)_F0F0

微码数据包

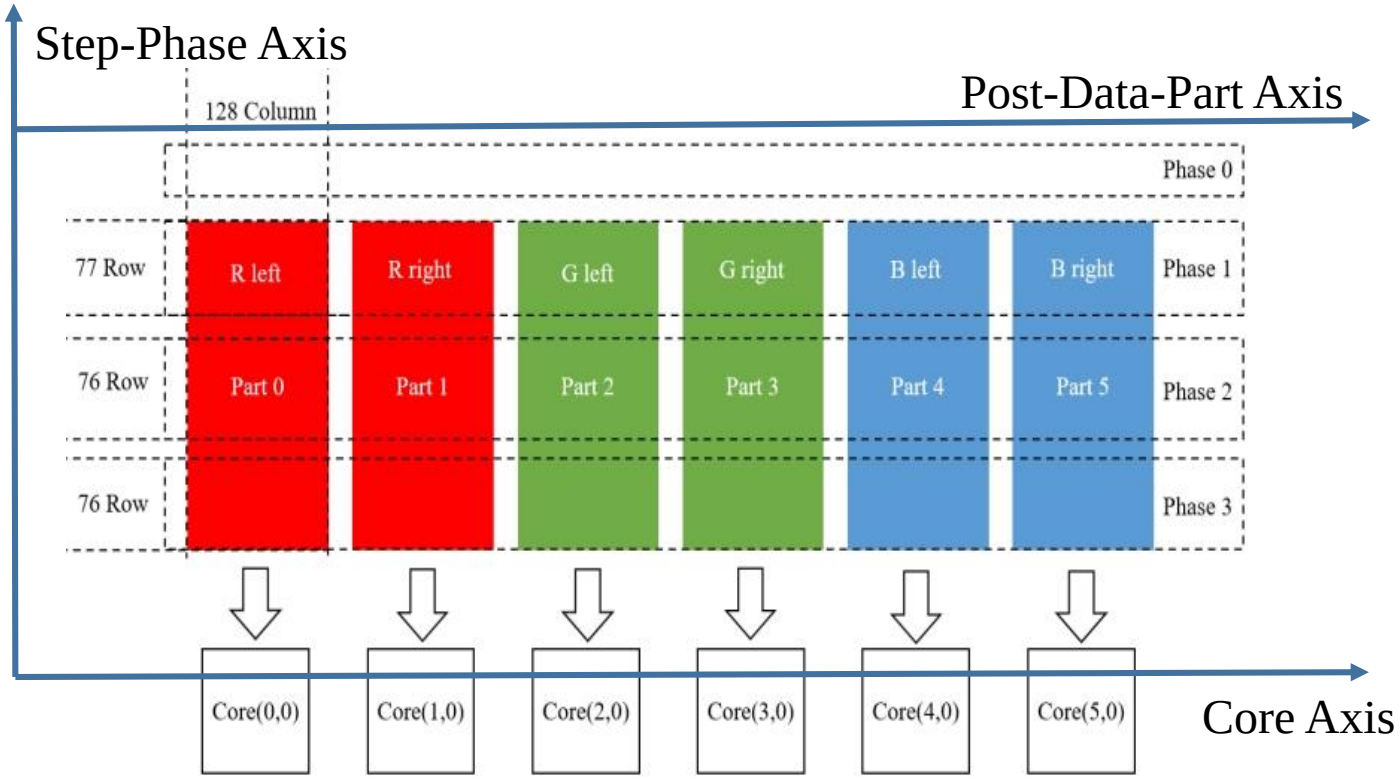
tianji_2g_VA版本只有一种包格式，如下表所示：

M[127:126]	F[125:124]	Fl[123:120]	Rsv[119:116]	ST[115:114]	CSE[113:112]	Data[111:48]	Rsv[47:0]
2’b01	2’b10	4’b0011	4’b0	2’b11/00	2’b/	64’b/	48’b/

M = 2'b01：DVPP包标志1。
F = 2'b10：DVPP包标志2。
Fl = 4'b0011：DVPP包标志3。
Rsv：保留(Reserve)。即暂时未用到，全填0即可。
ST：下发Step路由数据的起始标志。当ST = 2'b11时，标志开始下发Step路由数据，同时要求该包格式中的CSE为2'b10，Data无效，Data可全为0。
CSE：一个Phase里的一个Core的路由数据的start/end标志。
当CSE = 2'b10时，Data无效，Data可全为0，CSE标志开始下发一个Phase里一个Core的路由数据；
当CSE = 2'b00时，Data为传输的dvpp处理后的数据，也是TJ路由包所需路由数据；
当CSE = 2'b01时，Data有效，Data为传输的dvpp处理后的数据，CSE标志完成下发一个Phase里一个Core的路由数据。即携带每个Core最后一组路由数据的128bit包要求CSE = 2'b01。
Data：
传输的数据64bit(8 Byte)：第8 Byte，第7Byte … 第2 Byte，第1 Byte
(以处理后的一行数据为例，从左数的第1 Byte数据放最低位，也就是第1 Byte位置；第1 Byte数据放第2 Byte位置；…)

举例说明-Resnet50的绘制Step-Phase-Core-Part图

定义分辨率为1920 × 1080 × 3 (H,W,C) 的整幅图片为Frame，1个Frame包含分辨率为224 × 224 × 3 (H,W,C) 的小尺寸图片定义为Picture，1个Picture拆分和预处理后的6个分辨率为128 × 229 × 1 (R/G/B) 的图像单元定义为Part。 Phase传输数据与Part0~5定义如下图所示：



ResNet50所需的路由数据下发顺序为下传Phase1的Part0~5，再下传Phase2的Part0~5，再下传Phase3的Part0~5。共18块路由数据。每一块的路由数据传送到微码模块的接口，由于这些路由数据要送到不同Phase的不同Core，故以块为单位，定义了块的起始与结束标志如下表所示：

部署用户需要按照**天机数据元处理时序流程特征**绘制SPCP图

Resnet50微码- 天机数据元处理时序流程特征

除此之外，还需要按Phase下发组好包的数据，需要有控制芯片的交互指令，Trigger和Gfinish；

Step 0 开始
Trigger 0 触发
等待Phase 0 Gfinish
Phase 1 开始
Phase 1 数据下发
.....
Step 0 结束
Step 1 开始
.....

举例说明Resnet50微码控制器时序控制包

Resnet50的运行说明：对于9分类case, 共需运行9个step,每个step运行6个phase，则每个step待处理数据按如下流程组包：

1. 微码天机控制器时序控制包

```
32'd1:  data_nxt    = 128'h1200000000000000_8_0_0_0_0_00_00_000_F0F0;//MC_S
32'd2:  data_nxt    = 128'h1200000000000000_0_0_6_0_0_00_00_000_F0F0;//STEP_S                      //MC_D // Step
32'd3:  data_nxt    = 128'h1200000000000000_0_0_8_0_0_00_00_000_F0F0;//Trigger
32'd4:  data_nxt    = 128'h1200000000000000_0_0_2_0_0_00_00_000_F0F0;//PHASE_S 0 Empty             //Phase 0
32'd5:  data_nxt    = 128'h1200000000000000_0_0_1_0_0_00_00_000_F0F0;//PHASE_E 0
32'd6:  data_nxt    = 128'h1200000000000000_0_0_9_0_0_00_00_000_F0F0;//Gfinish                     //Phase 1
32'd7:  data_nxt    = 128'h1200000000000000_0_0_2_0_0_00_00_000_F0F0;//PHASE_S 1
32'd8:  data_nxt    = 128'h1200000000000000_0_0_3_0_4_00_00_000_F0F0;//PHASE_D 1 Data to Core(0,0)
32'd9:  data_nxt    = 128'h1200000000000000_0_0_3_1_4_00_00_000_F0F0;//PHASE_D 1 Data to Core(1,0)
32'd10: data_nxt    = 128'h1200000000000000_0_0_3_2_4_00_00_000_F0F0;//PHASE_D 1 Data to Core(2,0)
32'd11: data_nxt    = 128'h1200000000000000_0_0_3_3_4_00_00_000_F0F0;//PHASE_D 1 Data to Core(3,0)
32'd12: data_nxt    = 128'h1200000000000000_0_0_3_4_4_00_00_000_F0F0;//PHASE_D 1 Data to Core(4,0)
32'd13: data_nxt    = 128'h1200000000000000_0_0_3_5_4_00_00_000_F0F0;//PHASE_D 1 Data to Core(5,0)
32'd14: data_nxt    = 128'h1200000000000000_0_0_1_0_0_00_00_000_F0F0;//PHASE_E 1
32'd15: data_nxt    = 128'h1200000000000000_0_0_9_0_0_00_00_000_F0F0;//Gfinish
32'd16: data_nxt    = 128'h1200000000000000_0_0_2_0_0_00_00_000_F0F0;//PHASE_S 2                   //Phase 2
32'd17: data_nxt    = 128'h1200000000000000_0_0_3_0_4_00_00_000_F0F0;//PHASE_D 2 Data to Core(0,0)
32'd18: data_nxt    = 128'h1200000000000000_0_0_3_1_4_00_00_000_F0F0;//PHASE_D 2 Data to Core(1,0)
32'd19: data_nxt    = 128'h1200000000000000_0_0_3_2_4_00_00_000_F0F0;//PHASE_D 2 Data to Core(2,0)
32'd20: data_nxt    = 128'h1200000000000000_0_0_3_3_4_00_00_000_F0F0;//PHASE_D 2 Data to Core(3,0)
32'd21: data_nxt    = 128'h1200000000000000_0_0_3_4_4_00_00_000_F0F0;//PHASE_D 2 Data to Core(4,0)
32'd22: data_nxt    = 128'h1200000000000000_0_0_3_5_4_00_00_000_F0F0;//PHASE_D 2 Data to Core(5,0)
32'd23: data_nxt    = 128'h1200000000000000_0_0_1_0_0_00_00_000_F0F0;//PHASE_E 2
32'd24: data_nxt    = 128'h1200000000000000_0_0_9_0_0_00_00_000_F0F0;//Gfinish
32'd25: data_nxt    = 128'h1200000000000000_0_0_2_0_0_00_00_000_F0F0;//PHASE_S 3                   //Phase 3
32'd26: data_nxt    = 128'h1200000000000000_0_0_3_0_4_00_00_000_F0F0;//PHASE_D 3 Data to Core(0,0)
32'd27: data_nxt    = 128'h1200000000000000_0_0_3_1_4_00_00_000_F0F0;//PHASE_D 3 Data to Core(1,0)
32'd28: data_nxt    = 128'h1200000000000000_0_0_3_2_4_00_00_000_F0F0;//PHASE_D 3 Data to Core(2,0)
32'd29: data_nxt    = 128'h1200000000000000_0_0_3_3_4_00_00_000_F0F0;//PHASE_D 3 Data to Core(3,0)
32'd30: data_nxt    = 128'h1200000000000000_0_0_3_4_4_00_00_000_F0F0;//PHASE_D 3 Data to Core(4,0)
32'd31: data_nxt    = 128'h1200000000000000_0_0_3_5_4_00_00_000_F0F0;//PHASE_D 3 Data to Core(5,0)
32'd32: data_nxt    = 128'h1200000000000000_0_0_1_0_0_00_00_000_F0F0;//PHASE_E 3
32'd33: data_nxt    = 128'h1200000000000000_0_0_9_0_0_00_00_000_F0F0;//Gfinish
32'd34: data_nxt    = 128'h1200000000000000_0_0_9_0_0_00_00_000_F0F0;//Gfinish
32'd35: data_nxt    = 128'h1200000000000000_0_0_9_0_0_00_00_000_F0F0;//Gfinish
32'd36: data_nxt    = 128'h1200000000000000_0_0_5_0_0_00_00_000_F0F0;//STEP_E
32'd37: data_nxt    = 128'h1200000000000000_4_0_0_0_0_00_00_000_F0F0;//MC_E
```

Resnet50微码- 微码数据包

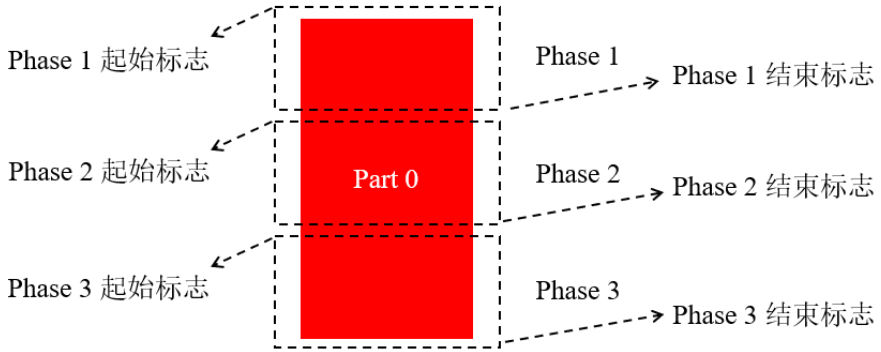
(以九宫格原始图像的一行像素为例，从左数的第1个RGB放最低位，也就是RGB_0位置；第2个RGB放RGB_1位置；…)

由于前端DVPP是顺序地从Part 0-5下发34bit数据包，但是路由数据包需要按Phase发送，每次均只发送某个Part的1/3部分。

为保留需要把收到的34bit像素数据包再次缓存下来，再按Phase发送。并且为了方便后续路由包多包组包，将34bit重组为66bit，包含8个8bit的像素数据和2bit的Phase标志位，格式与原34bit像素数据包格式类似。

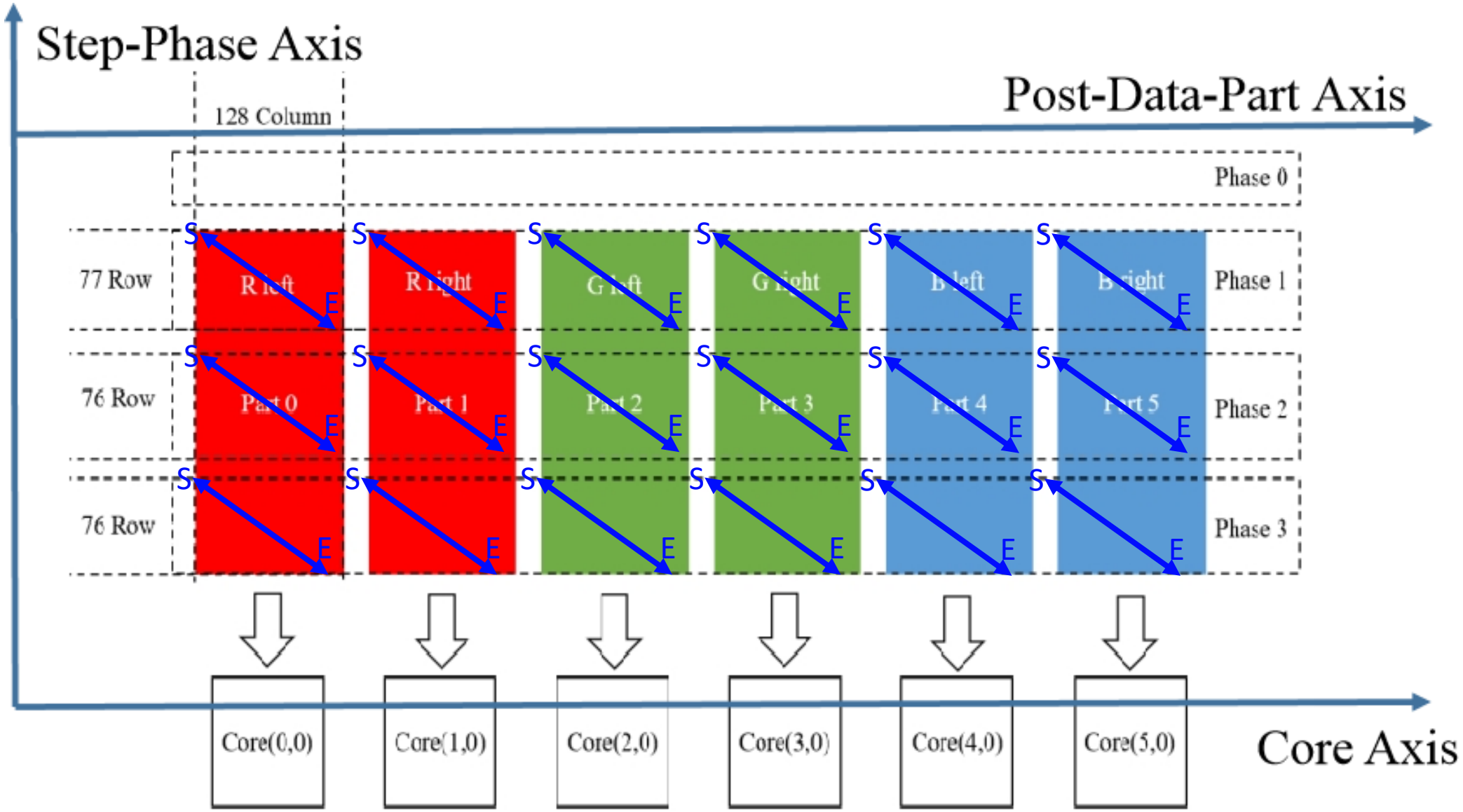
Phase起始标志位	Rsv
2b	64b
2’b10	/

Phase数据/结束标志位	Data7	Data6	Data5	Data4	Data3	Data2	Data1	Data0
2b	8b	8b	8b	8b	8b	8b	8b	8b
2’b00/2’b01	/	/	/	/	/	/	/	/



这里需要注意在读取到前端每个Phase的结束标志时，在路由数据组包的P标志位需要置为结束（每个Core的每个Phase的结束），而不是在发送完Part 0-5的1/3后才置为结束（所有Core的数据）。

Resnet50微码- 微码数据包-补充



微码天机控制器：配套工具

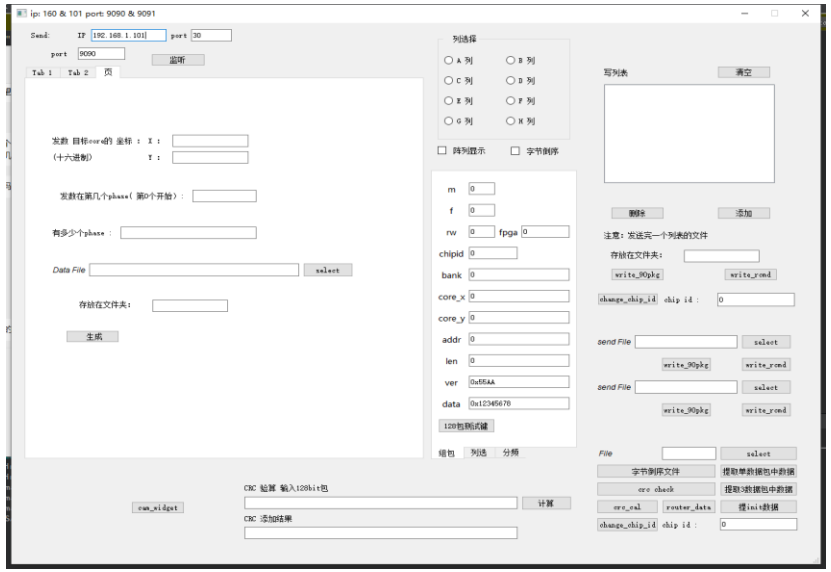


图-微码指令集生成器工具说明

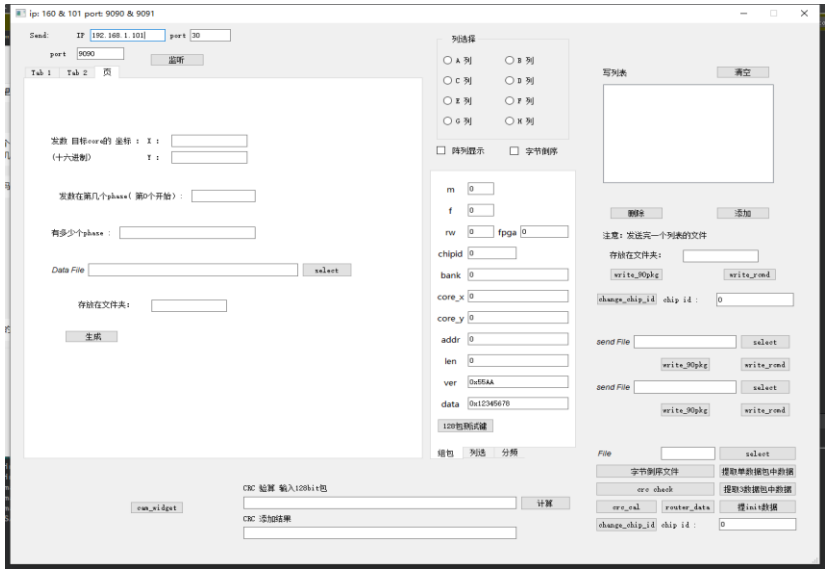


图-微码数据源整形工具

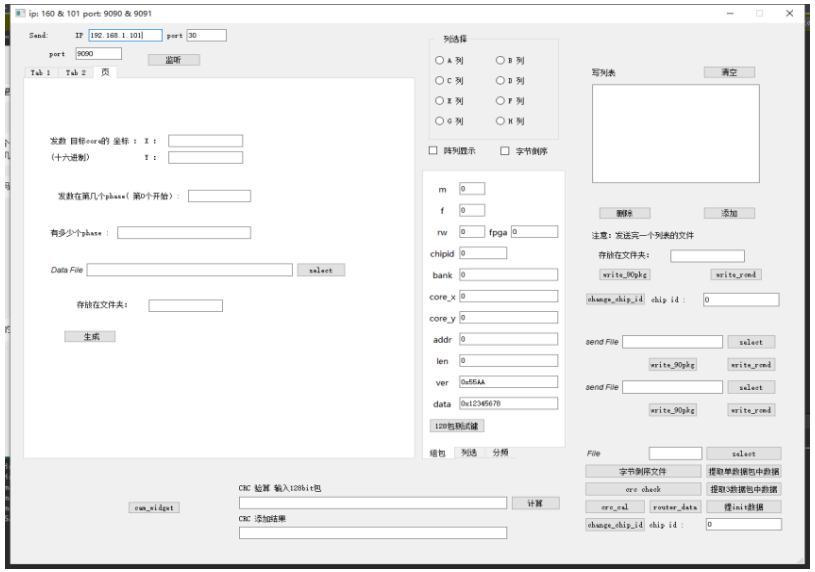
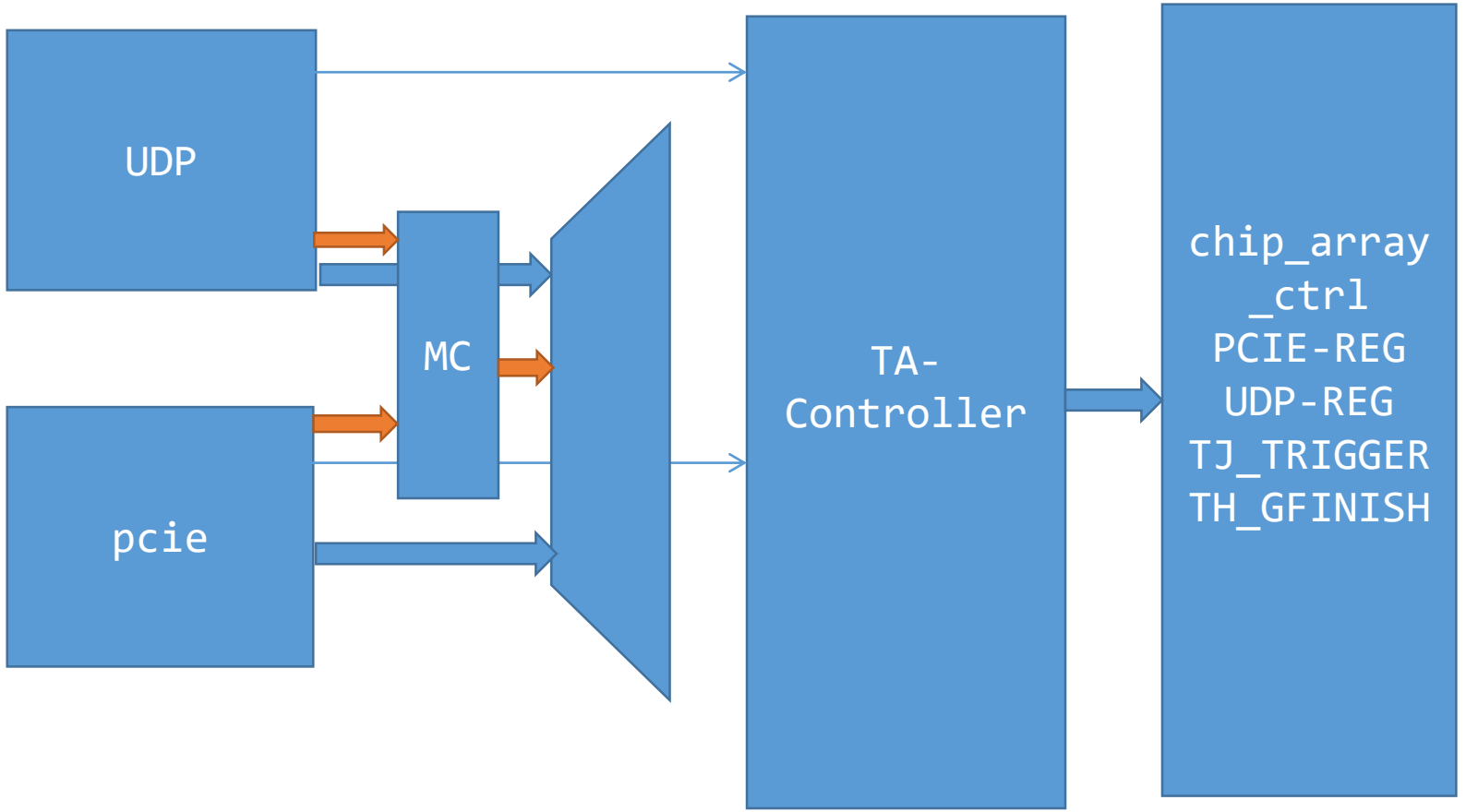


图-微码数据源生成工具和传输工具

序号	工具名称	功能描述
1	微码指令集生成器工具说明	将需要写入芯片的纯数据组成控制器使用的微码数据
2	微码数据源整形工具	将处理后的数据源结合时序流以及core配比进行划分
3	微码数据源生成工具和传输工具	微码数据源生成工具和传输工具，将微码数据源整形后的数据按照微码数据包的格式打包发送


UDP
pcie
MC
TA-
Controller
chip_array
_ctrl
PCIE-REG
UDP-REG
TJ_TRIGGER
TH_GFINISH

创新型集群

组包类别	备注		Host	FPGA0	FPGA1
STEP_S_CMD	step起始指令	无微码天机控制器时序控制包		step起始指令数据包	step起始指令数据包
TRIGGER_CMD	trigger拉起指令	无微码天机控制器时序控制包		trigger拉起指令数据包	trigger拉起指令数据包
GFINISH_CMD	空phase的Gfinish等待指令	无微码天机控制器时序控制包		空phase的Gfinish等待指令数据包	空phase的Gfinish等待指令数据包
PHASE_0_S_CMD	送数phase0起始指令	无微码天机控制器时序控制包		送数phase0起始指令数据包	送数phase0起始指令数据包
PHASE_0_DATA	送数phase0所需数据	天机包		Host-FPGA0-TJ0 TJ0-FPGA0-FPGA1-TJ1	TJ0-FPGA0-FPGA1-TJ1
PHASE_0_E_CMD	送数phase0结束指令	无微码天机控制器时序控制包		送数phase0结束指令数据包	送数phase0结束指令数据包
GFINISH_CMD	phase0的Gfinish等待指令	无微码天机控制器时序控制包		phase0的Gfinish等待指令数据包	phase0的Gfinish等待指令数据包
PHASE_1_S_CMD	送数phase1起始指令	无微码天机控制器时序控制包		送数phase1起始指令数据包	送数phase1起始指令数据包
PHASE_1_DATA	送数phase1所需数据	天机包		Host-FPGA0-TJ0 TJ0-FPGA0-FPGA1-TJ1	TJ0-FPGA0-FPGA1-TJ1
PHASE_1_E_CMD	送数phase1结束指令	无微码天机控制器时序控制包		送数phase1结束指令数据包	送数phase1结束指令数据包
GFINISH_CMD	phase1的Gfinish等待指令	无微码天机控制器时序控制包		phase1的Gfinish等待指令数据包	phase1的Gfinish等待指令数据包
PHASE_2_S_CMD	送数phase2起始指令	无微码天机控制器时序控制包		送数phase2起始指令数据包	送数phase2起始指令数据包
PHASE_2_DATA	送数phase2所需数据	天机包		Host-FPGA0-TJ0 TJ0-FPGA0-FPGA1-TJ1	TJ0-FPGA0-FPGA1-TJ1
PHASE_2_E_CMD	送数phase2结束指令	无微码天机控制器时序控制包		送数phase2结束指令数据包	送数phase2结束指令数据包
GFINISH_CMD	phase2的Gfinish等待指令	无微码天机控制器时序控制包		phase2的Gfinish等待指令数据包	phase2的Gfinish等待指令数据包
GFINISH_CMD	运算phase的Gfinish等待指令	无微码天机控制器时序控制包		运算phase的Gfinish等待指令数据包	运算phase的Gfinish等待指令数据包
GFINISH_CMD	运算phase的Gfinish等待指令	无微码天机控制器时序控制包		运算phase的Gfinish等待指令数据包	运算phase的Gfinish等待指令数据包
STEP_E_CMD	step终止指令	无微码天机控制器时序控制包		step终止指令数据包	step终止指令数据包
				TJ1-FPGA1-FPGA0-TJ0-Host	

QA?