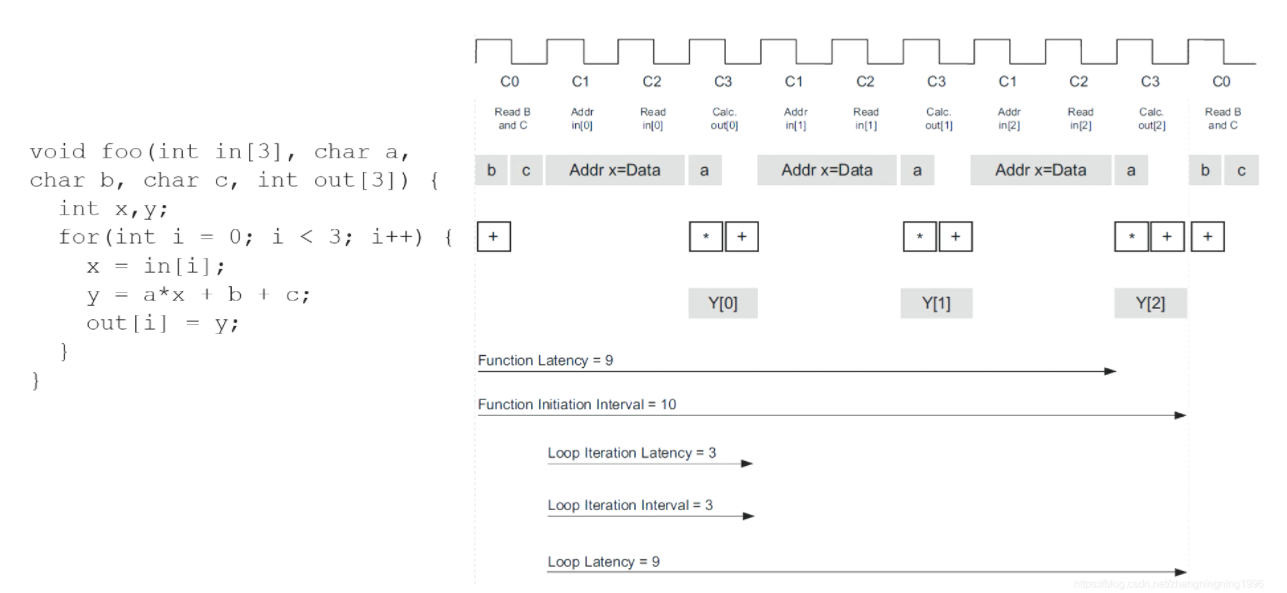
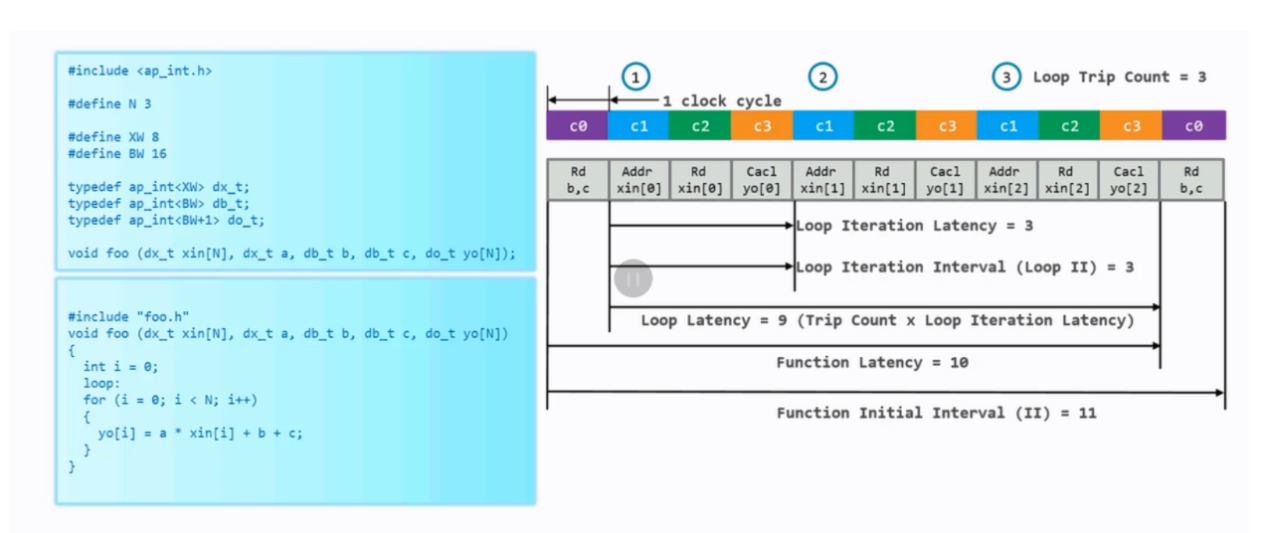


Area：实现该C代码多用的资源量，该资源包括LUT、registers、Block RAM、DSP48等等。

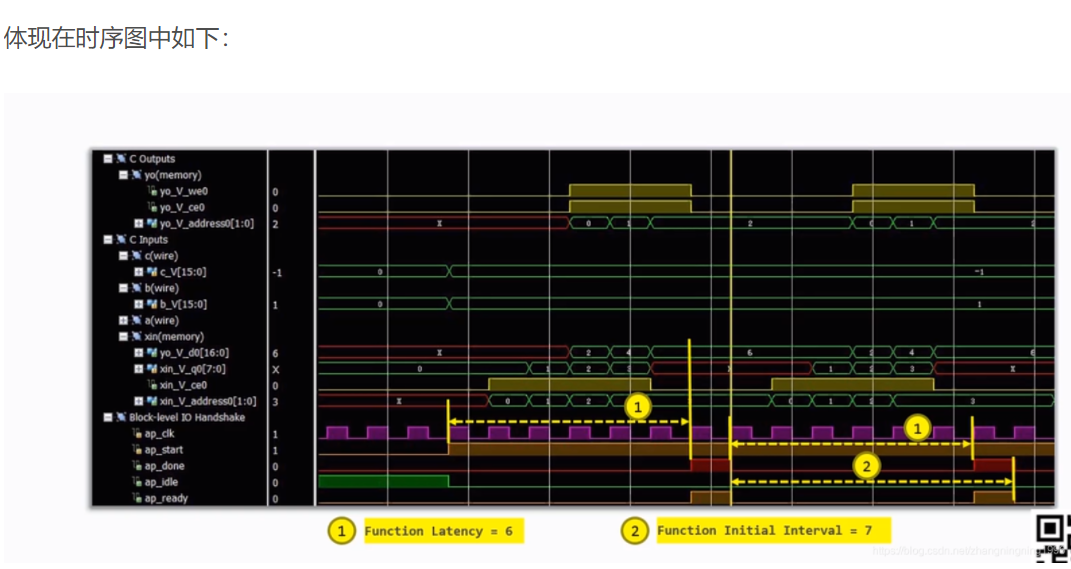
Latency：C函数完成所有的一次输出所需要的周期数。  
Initiation interval(II)：C函数需要多少时间才可以重新接受新的数据，也就是C函数本次开始到下一次开始所需要的周期数。





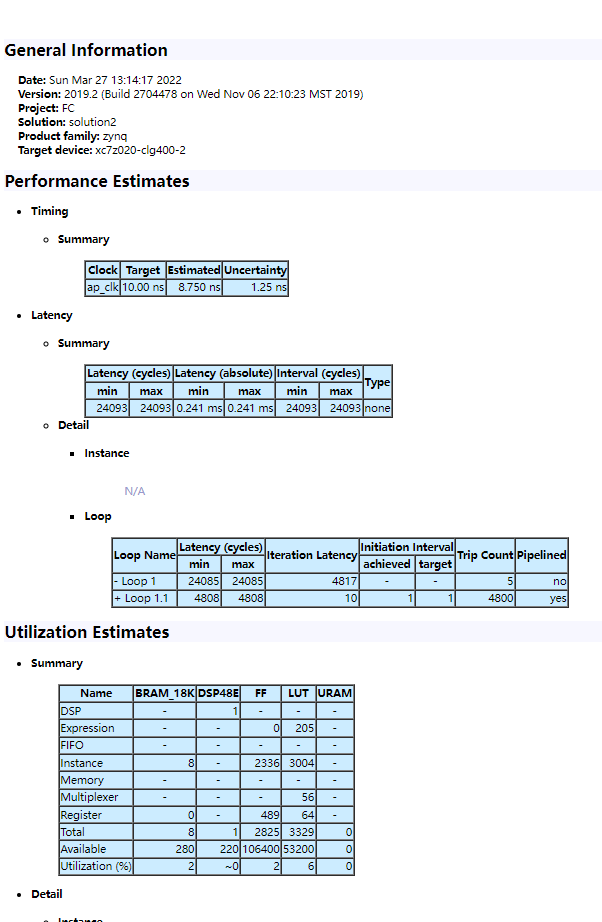






## 对于综合报告中各项的解释：

eg：



Performance Estimates >Timing：目标时钟频率、时钟的不确定性和对最快可实现的时钟频率的估计。

Performance Estimates > Latency > Summary： 延迟是产生输出所需的周期数。启动间隔是可以应用新输入之前的时钟周期数，在没有任何PIPELINE指令的情况下，延迟比启动间隔小一个周期（在写入最终输出时读取下一个输入）。

Performance Estimates > Latency > Detail：  
1）此块中的实例（子功能）和循环的延迟和启动间隔。如果任何循环包含子循环，则会显示循环层次结构

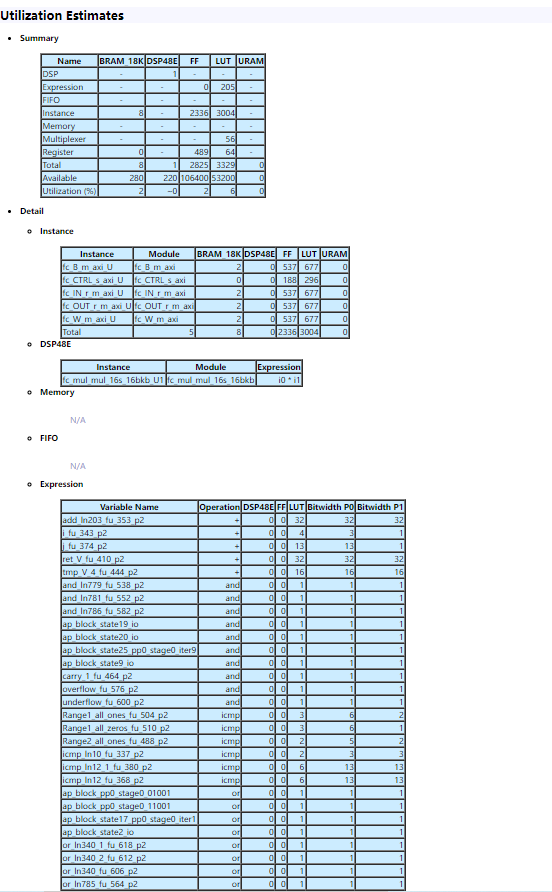
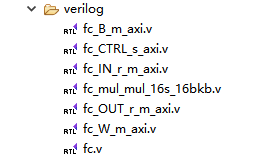
2）最小和最大延迟值表示执行循环的所有迭代的延迟。代码中条件分支的存在可能会使最小值和最大值不同。

3）迭代延迟是循环的单次迭代的延迟

4）如果循环具有可变延迟，则无法确定延迟值并显示为问号(?)。

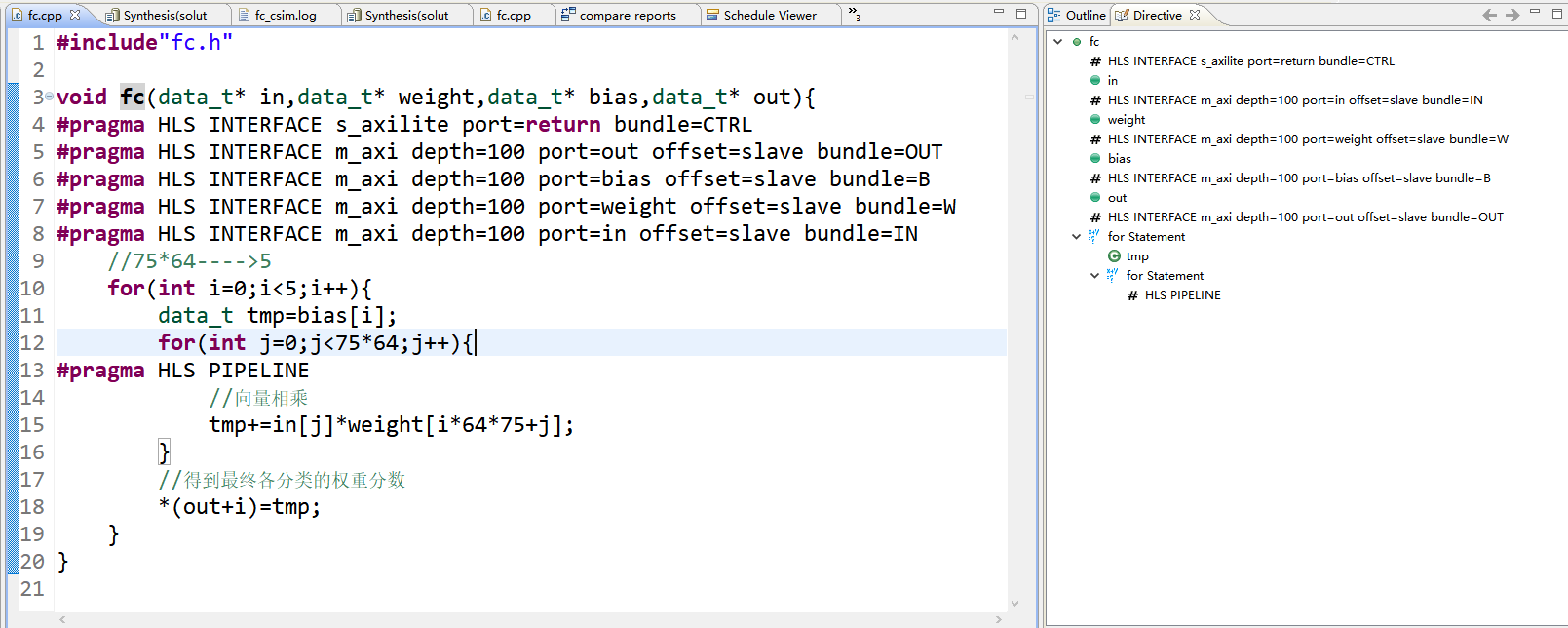
5）tripcount显示循环迭代的总数。

6）任何指定的目标启动间隔都显示在实际达到的启动间隔旁边



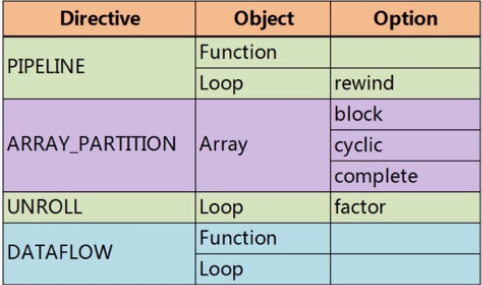
Utilization Estimates > Details > Instance：此处指定的资源由在此层次结构级别实例化的子块使用。如果设计仅没有RTL层次结构，则不会报告任何实例。如果存在任何实例，单击实例名称会打开该实例的综合报告

## Code：



## 优化指令：

1. 基于过程的优化指令

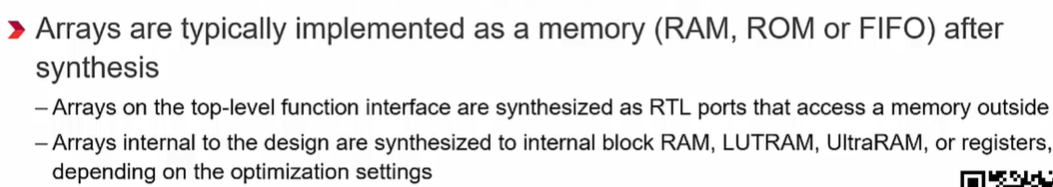


PIPELINE：流水线

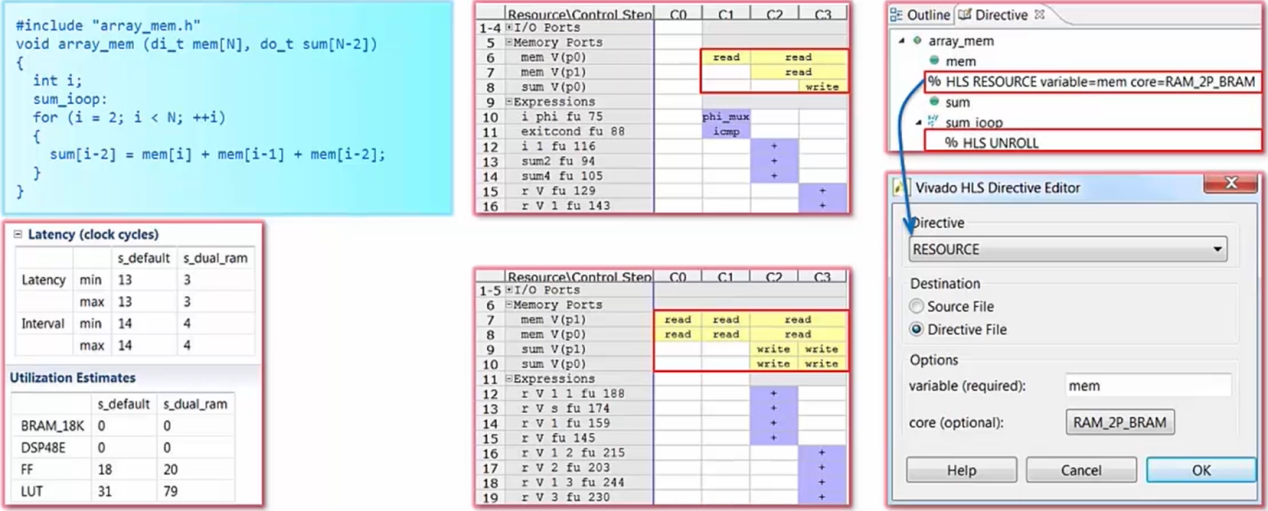
当对循环或函数作流水时，循环或函数下面的层次结构中的任何循环都会被展开。

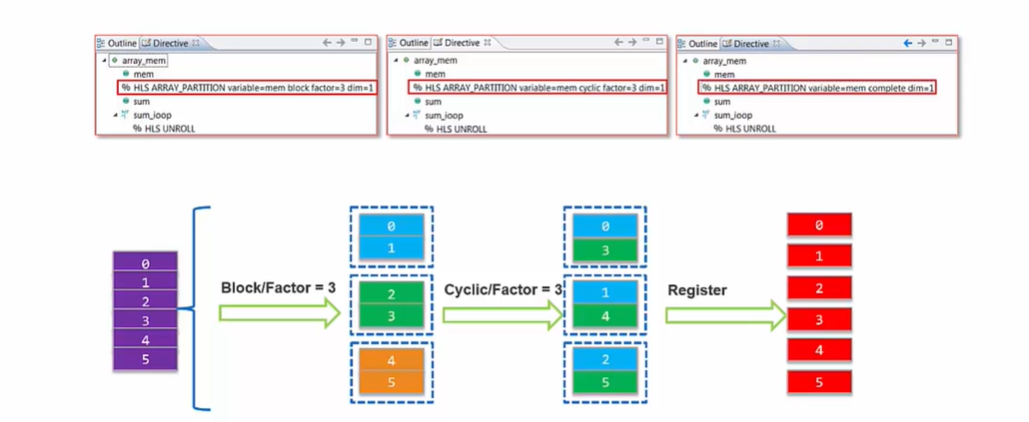
最内部for循作流水为大多数应用程序提供了一般可接受的延迟和的硬件资源。

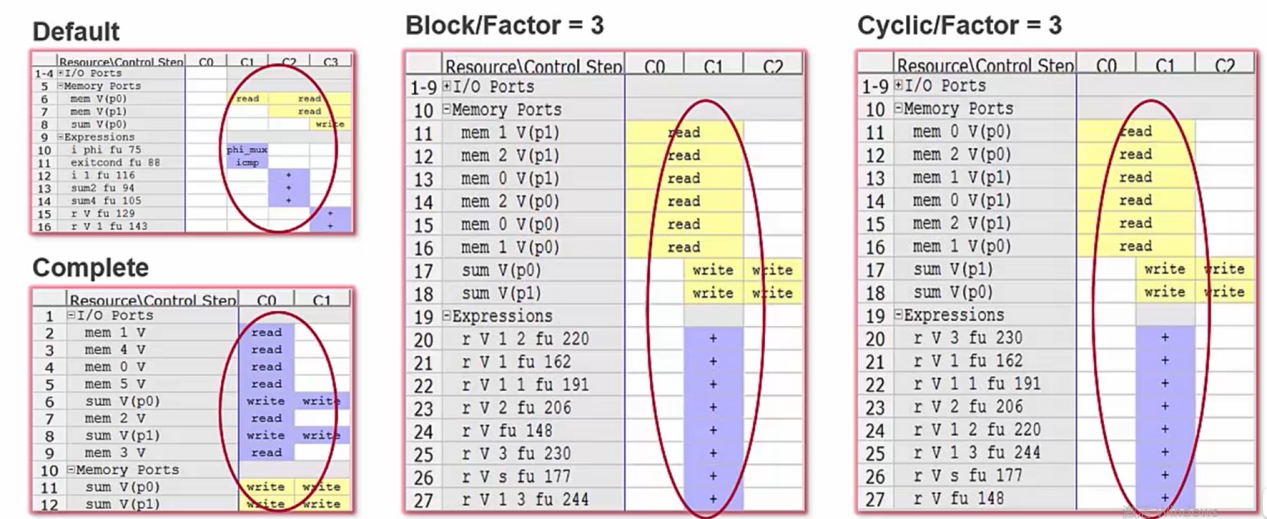
ARRAY\_PARTITION:数组分割，资源指令可以明确指定使用哪种类型的RAM，并因此创建哪些RAM端口（单端口或双端口）。如果未指定资源，Vivado HLS 将根据延迟和资源利用情况自行决定。



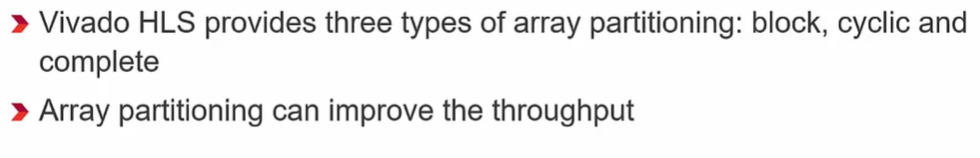
Eg：



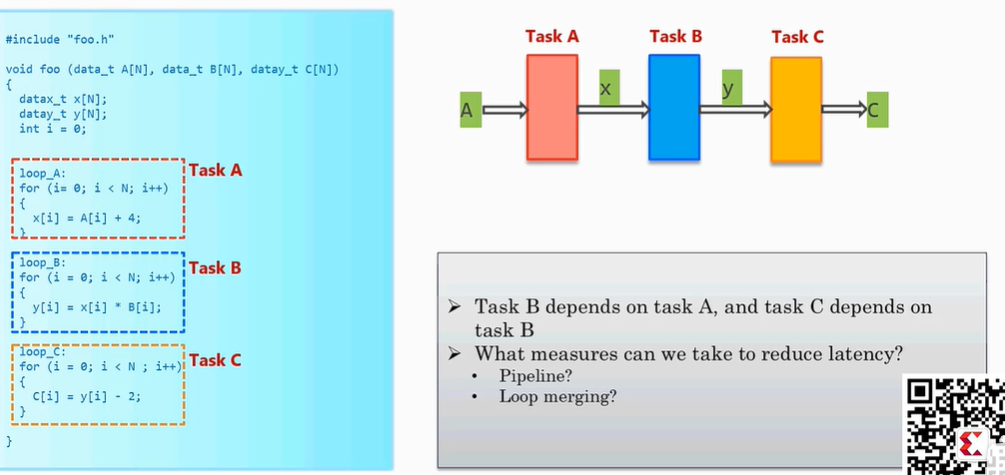




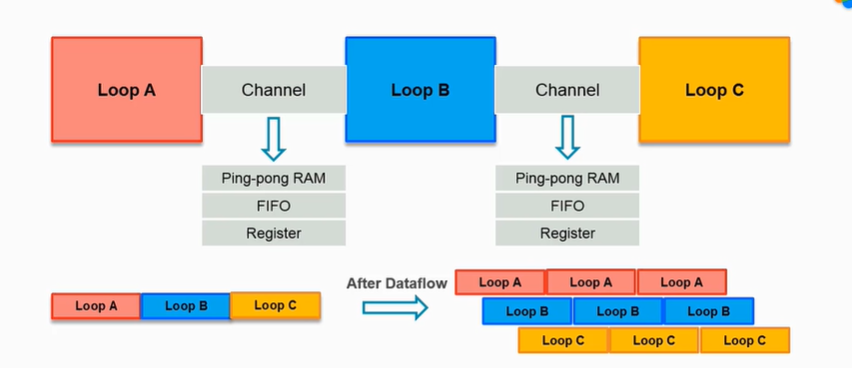
但是越多的block会产生更多的端口



UNROLL（for 循环展开指令）：折叠指的是所有for循环共用一套电路，电路被分时复用，相应的展开就是复制多个电路。

DATAFLOW：

如图，当A，B，C三个task具有依赖关系时，可以采取dataflow的优化方式！



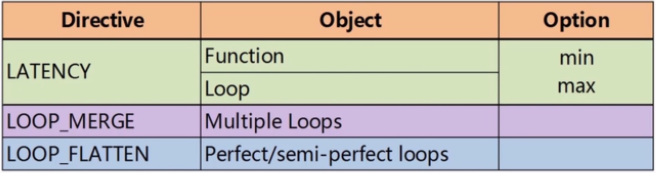
通过dataflow我们可以看到在执行用户B的时候，并不需要等到用户A执行完之后再去执行，各个任务之间有一些交叠，可以降低latency。

注：dataflow使用时有一些限制，并不是所有for循环可以使用dataflow（single—producer 或者是 bypassing）

Rewind：在没有rewind之前，for循环执行一次之间会有空挡，有了rewind就会连续执行

注：带rewind的pipeline可以帮助减少延迟，但它有一些限制

1. 基于延迟的优化



LOOP\_MERGE：循环合并指令，将两个并行的for循环合并，默认情况下，for循环间顺序执行，会使系统创建额外的状态机，这就会增加额外的资源和延迟。

Eg：loop region：

{

Loop1：

Loop2：

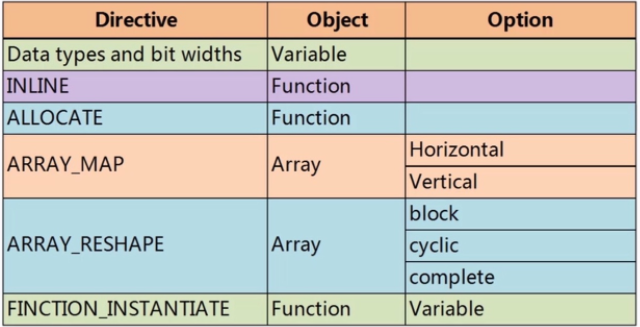
}

注：1）如果循环边界是两个不同的常数，在合并时以最大的作为循环边界

2）当for循环的边界一个是常数，一个是变量的情况下，没有办法合并

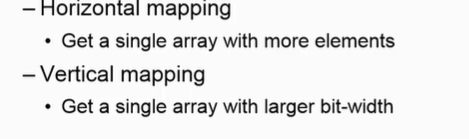
总结：当for循环都是变量的情况下，我们可以通过修改代码来实现for循环的合并（即当两个for循环的边界都是变量的时候，他们必须是同一个变量才能进行合并）

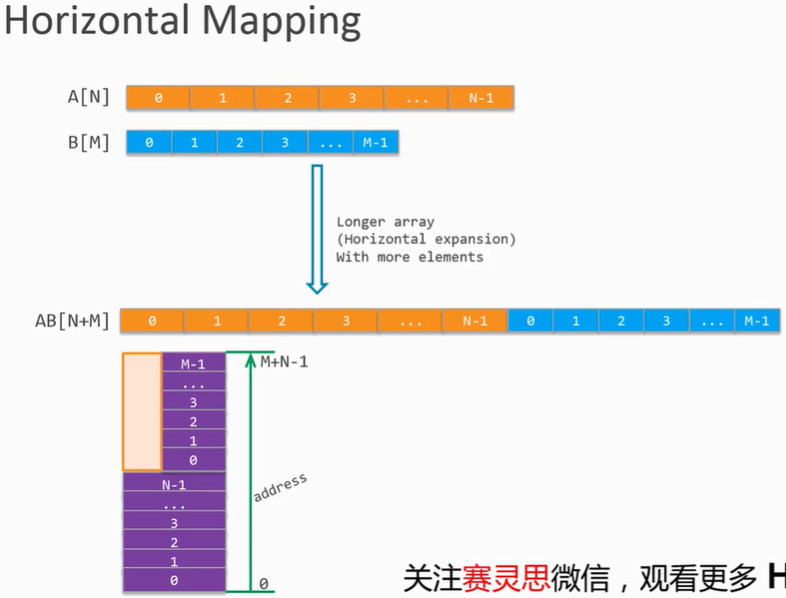
1. 基于资源的优化

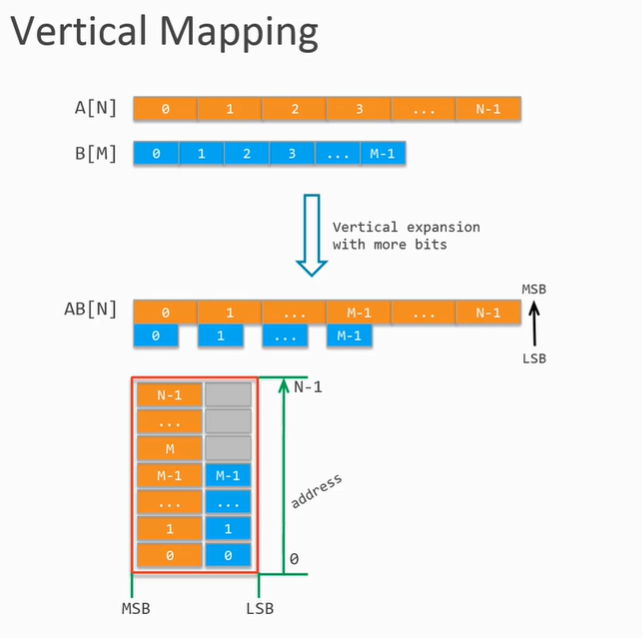


ALLOCATION：若两个功能相当的for循环时，我们可以将其封装成一个函数（节省硬件资源），并且通过顶层函数去实例化这两个函数，并且可以对顶层函数用到ALLOCATION使这两个实例并行执行

ARRAY\_MAP:ARRAY\_MAP指令支持将小数组映射到大数组的两种方式:







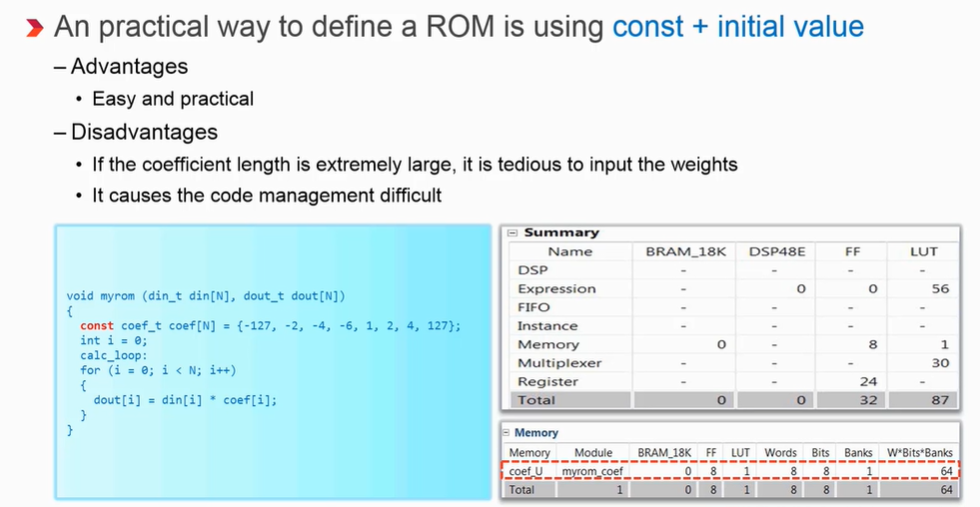
ARRAY\_RESHAPE:就是将ARRAY—PARTITION和纵向的ARRAY\_MAP结合在一起使用，这样做的好处是1）减少RAM块的数量 2）同时获取多个数据，提高数据的并行性，相应的也有三种分割方式（block cyclic register/complete）

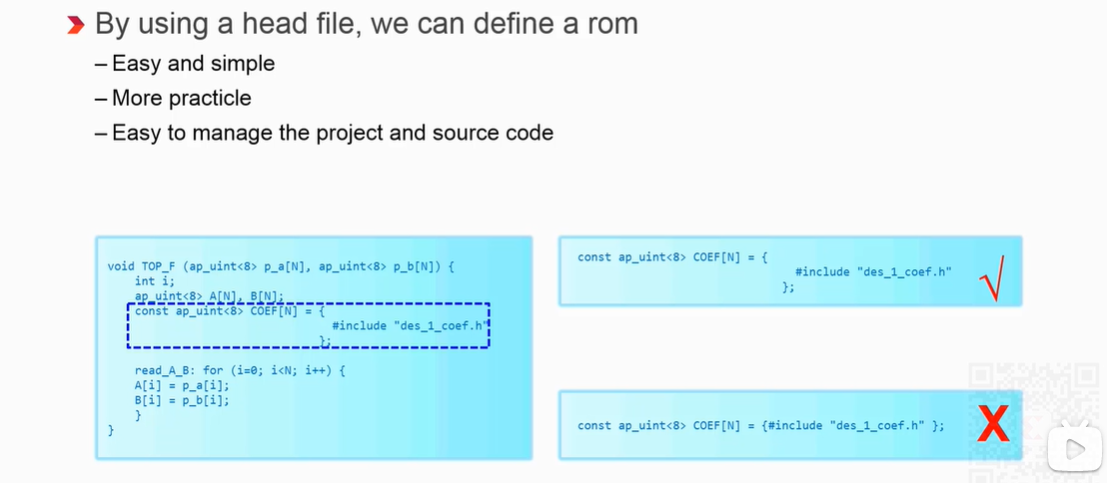
总结：arraymap可以减少RAM资源用量，但是不能减少吞吐率。而ArrayReshape可以通过本身包含的Array-partition来改善吞吐率，同时在一定程度上获得了资源和数据吞吐率的折中。

INLINE：对函数的INLINE实际上是去除了函数的层次化，其好处是可以改善相关资源（因为实例的函数模块就会出现在report中）， 对于一些小的函数，vivado HLS是会自动的去进行INLINE处理，如果我们不希望此操作发生，可以通过directive中的-off选项实现。

关掉inline的话资源会变多，latency也会变大

**如何去定义一个ROM：**



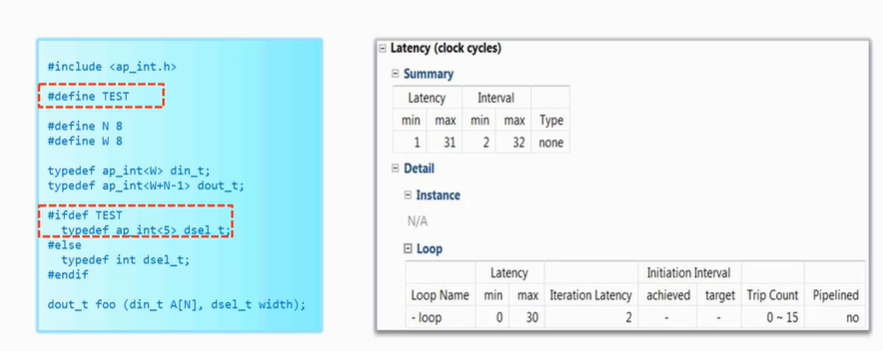


Xilinx强烈建议对打算用作memory的数组使用static限定符。如数组初始化中所述，static类型的行为方式与RTL中的memory几乎相同

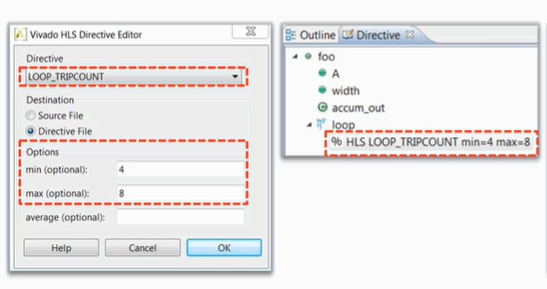
当仅读取数组时，也建议使用const限定符，因为Vivado HLS不能总是通过对设计的分析来推断应该使用ROM

**循环的边界为变量时，综合报告会出现？，我们有三种处理方式：**

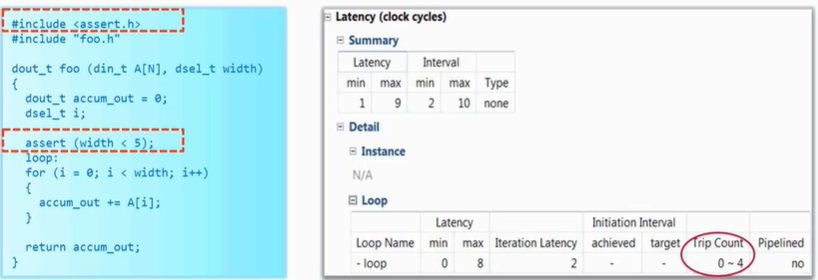
**1）：**



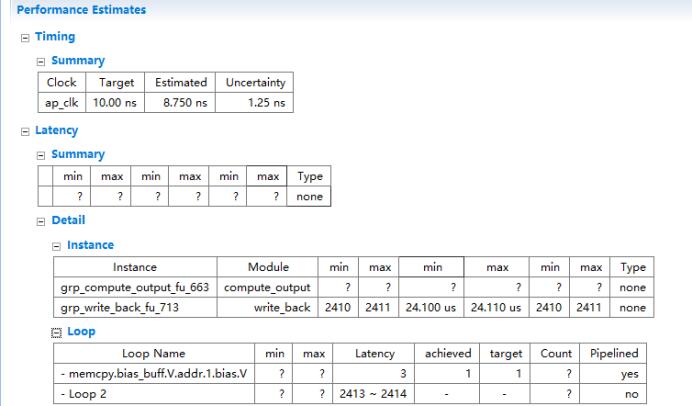
**2）：**



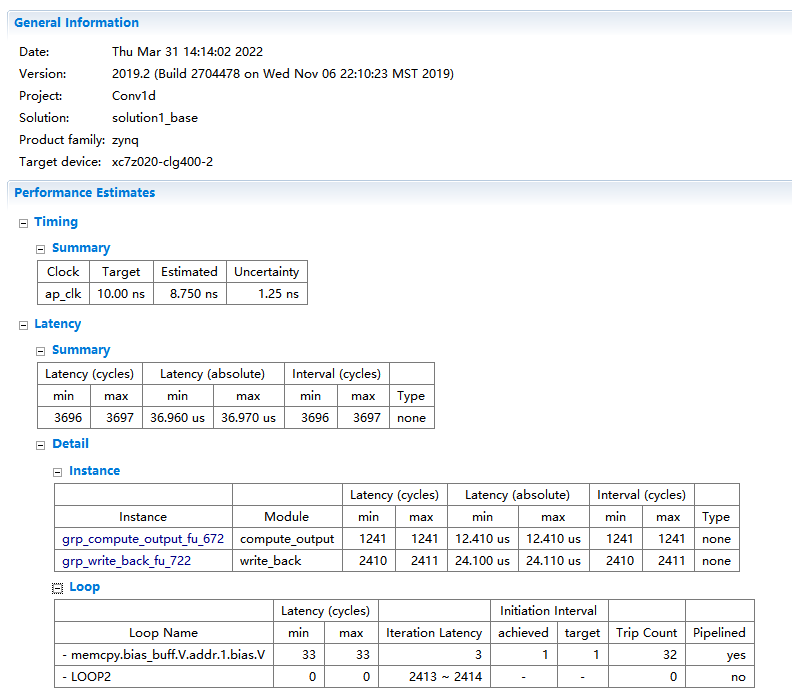
**3）：**



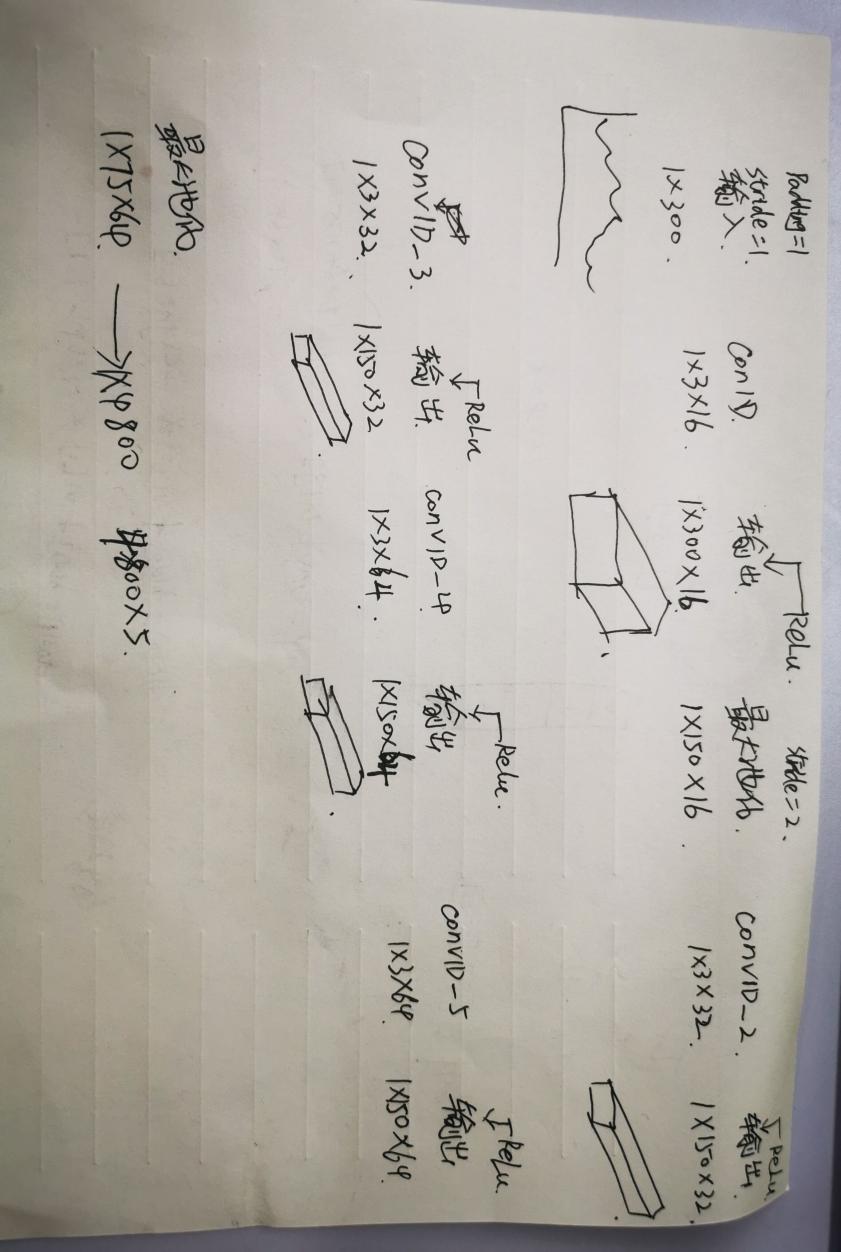
在Conv1d中的实例：

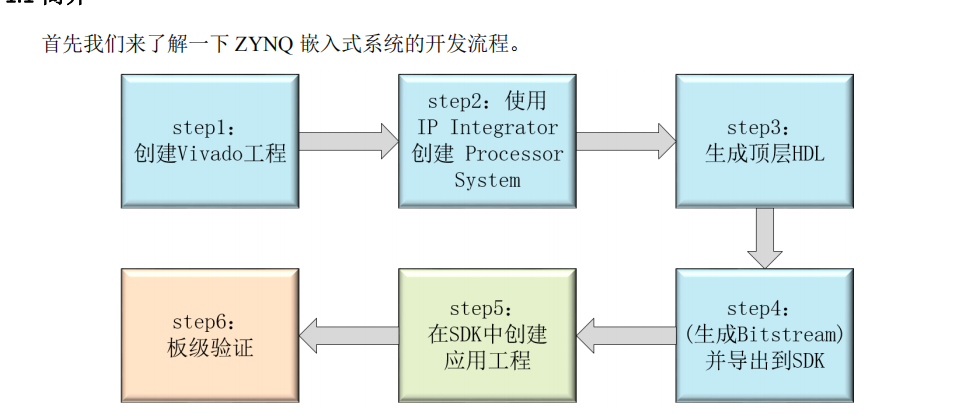


按照方法2修改过后：



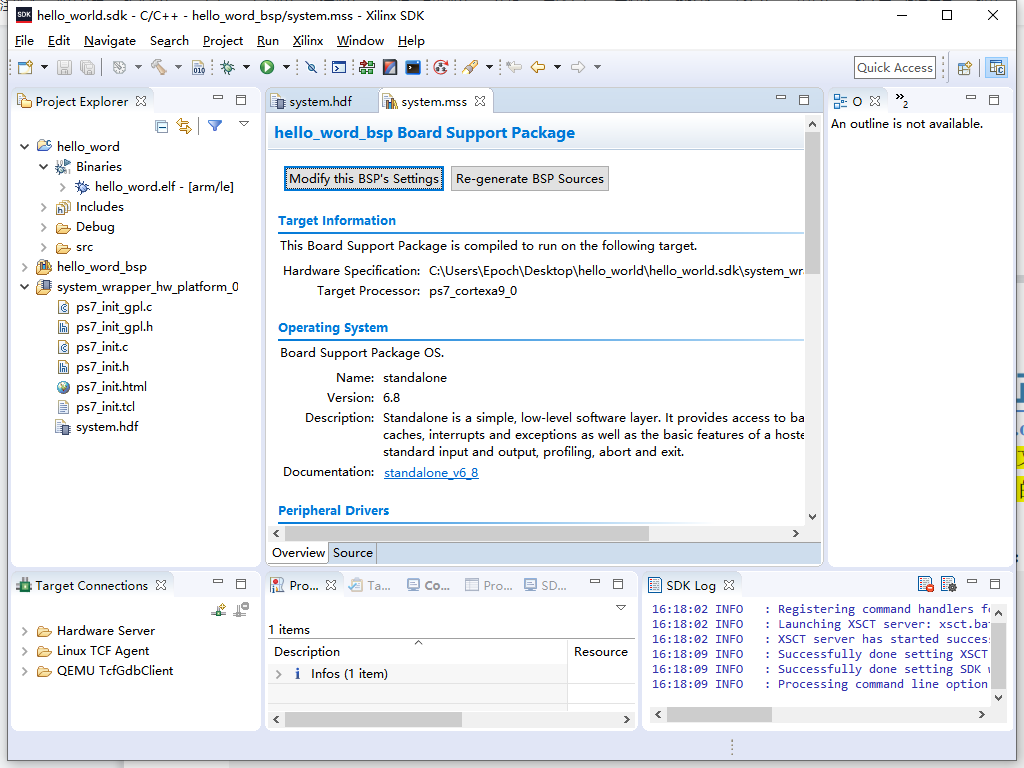
**网络结构：**





Vivado嵌入式系统开发流程大体可以分为6步。其中step1至step4为硬件设计部分，在Vivado软件中实现；step5为软件设计部分，在SDK软件中实现；step6为功能的验证。复杂的程序还涉及Debug，这个也是在SDK软件中实施。

Vivado 开发套件中提供了一个图形化的设计开发工具——IP 集成器（IP Integrator），在IP集成器中可以非常方便的插入各种功能模块（IP）。它支持关键 IP 接口的智能自动连接、一键式IP子系统生成、实时DRC等功能，能够帮助我们快速组装复杂系统，加速设计流程。



在菜单栏选择File>New>Application Project,新建一个SDK应用工程。

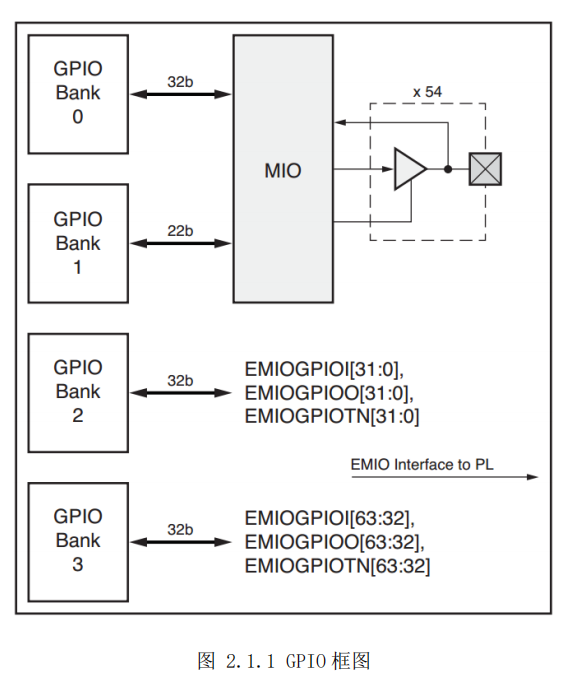
SDK创建了一个hello\_world应用工程和hello\_world\_bsp板级支持包（BSP）工程。同时工具会自动对工程进行编译，并生成ELF文件“hello\_world.elf”，如上图所示：另外工程创建完成后，SDK主界面会打开BSP工程目录下的system.mss文件。MSS是英文Microprocessor Software Specification的缩写，即微处理器软件说明。该文件包含BSP的操作系统信息、硬件设计中各个外设的软件驱动等信息。

关于PS，PL端各个引脚资源的信息：

ZYNQ PS中包含一组丰富的外设，如USB控制器、UART控制器、I2C控制器以及GPIO等等。他们提供了各种工业标准的接口，用于和外部设备进行通信。其中GPIO外设一般用于控制一些简单的外设，如LED和蜂鸣器，此时GPIO用作输出；也可以用于观测一些简单外设的状态，如按键，此时GPIO用作输入。GPIO可以通过MIO连接到PS端的引脚，也可以通过EMIO连接到PL。

GPIO是英文“general purposeI/O”的缩写，即通用的输入/输出。它是ZYNQPS中的一个外设，用于观测和控制器件引脚的状态。图2.1.1是GPIO的框图，从中我们可以看到GPIO分为4个Bank，其中Bank0和Bank1连接到MIO；而Bank2和Bank3连接到EMIO。

除Bank1之外的Bank都具有32bit，Bank1只具有22bit是因为总共只有54个MIO，其中32bit的Bank0控制了MIO[0~31]，剩下的MIO[31~53]就由22bit的Bank1控制。Bank2和Bank3用于控制扩展的MIO即EMIO，也就是说总共可以有32+32=64个EMIO。



从图2.1.2MIO一览表中我们可以看到MIO一但选定，引脚位置就已经确定下来了，不需要添加引脚约束。

