技术文档

一、 系统功能简介

1.1 系统功能

本系统利用 ARM Cortex-M3 处理器内核以及由我们团队在 FPGA 平台上独立设计的 SOC 架构，实现了对摄像头采集人脸数据的实时检测功能。

系统运行期间，摄像头将采集一帧图像数据并存放到 DDR，Cortex-M3 处理器在检测到图像数据成功写入后，将执行智能算法的处理流程，协同硬件加速器计算所采集图像中包含人脸的数量及所处位置等信息。

将得到的人脸检测结果通过 HDMI 输出到显示器，在显示器上框出被检测到的人脸，并通过 FPGA 板载 LED 灯将人脸的数目以二进制的格式显示出来，同时，所检测到的人脸在图像中的位置、大小、检测时间等信息可以通过 UART 连接到计算机串口助手中显示。为了满足在 Cortex-M3 处理器上对图像进行实时检测的要求，我们特地采用了经过我们团队根据 Cortex-M3 处理器的特点进行精心优化的决策树算法，测试表明该算法在我们的系统中实现了检测准确率及检测速度的优秀平衡。

1.2 应用场景分析

我们的实时人脸检测系统有着巨大的市场前景。在家居安防领域，可以通过我们的系统实现人类闯入报警装置，在摄像头捕捉到的区域检测到人脸后触发报警。在智慧城市建设中，可以在人行道的两端安装我们的系统，实现交通灯的实时智能联动，当检测到等待过马路的行人数目大于一定阈值或有行人正在过马路后，给机动车道亮警示信号灯，以保障行人安全。在新冠疫情期间，我们的系统可以安装在商圈、旅游景点，实时检测人流密度，为实时限流措施提供参考。

不难发现，我们的生产生活中有着大量的轻量级人脸检测需求，它们并不需要检测分析人脸的细节，因此不需要动辄耗费大量昂贵的硬件处理器和能耗的卷积神经网络算法去实现对这些轻量级应用场景的覆盖。我们开发的基于 ARM Cortex-M3 处理器的实时人脸检测系统为这些应用场景提供了低成本、高效率、低能耗的绝佳解决方案。

二、 系统框架及硬件介绍

2.1 系统总框图

本系统的主要功能部件包括 ARM Cortex-M3 核、决策树算法硬件加速器、AHB 总线矩阵、DDR 控制器、片上存储模块、摄像头模块、显示器模块、APB 外设等，系统框图见图 2.1。

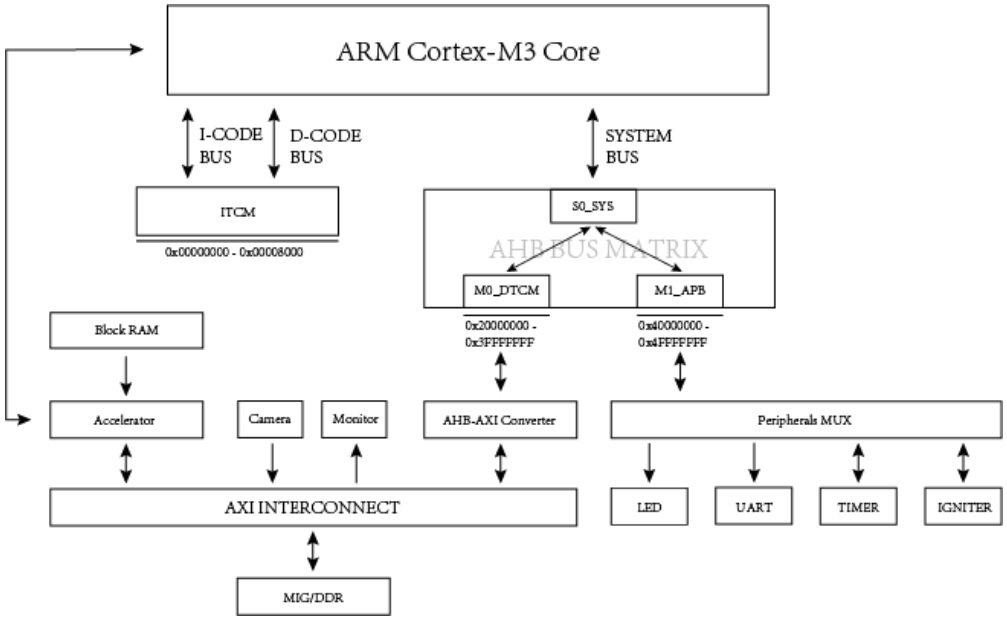

图 2.1 系统总框图

本系统采用 ARM 公司提供的 DesignStart Eval 版本 Cortex-M3 软核 IP 作为中央处理单元，SOC 整体部署在 FPGA 平台上。

如图 2.1 所示，Cortex-M3 处理器通过 3 根总线与其余部件通信。其中的 I-CODE 总线和 D-CODE 总线是基于 AHB‐Lite 总线协议的 32 位总线，负责在 0x00000000‐0x00008000（32KB） 之间的取指、数据访问操作。ITCM 相当于代码区（CODE 区，起始于 0x0 地址）的硬件实现，主要是采用 FPGA 的 BRAM 资源实现的存储区域，用来保存编译好的处理器操作指令，以供 I-CODE 和 D-CODE 总线访问。编译得到的可执行文件需要通过 fromelf 工具转换成面向 Verilog HDL 内存模型的 hex 格式文件，并用 readmemh 系统函数初始化到 ITCM 的硬件代码中。在使用 keil 进行调试时，更新后的软件代码可以通过 D-CODE 端写进 ITCM。

Cortex-M3 的系统总线控制了 M3 存储模型的大部分地址区域，包括本系统的 DTCM 及 APB 外设部分。其中，DTCM 相当于 SRAM 区（起始于 0x20000000 地址）的硬件实现，是采用 DDR 实现的内存区域。DTCM 的主要功能是为软件代码的执行提供内存空间，如为堆与栈开辟空间。

ARM 公司还提供了一个相当有用的工具——Cortex-M System Design Kit（CMSDK），为开发者开发 SOC 应用提供了便利。CMSDK 包含了许多关于 AHB 总线的转接模块，例如在本项目中用到的“AHB to APB”模块以及“APB slave example”参考模块（用于搭建 APB 桥以及连接低速外设）。

更为重要的是，CMSDK 提供了可配置的总线矩阵模块（AHB Bus Matrix）。我们可以利用总线矩阵模块作为系统总线与 DTCM 及低速外设的互联模块，同时配置好各个部件的地址映射。图 2.2 展示的是本项目利用 CMSDK 总线矩阵模块配置地址映射的脚本代码。其中，总线矩阵只有一个 slave 端（S0_SYS），连接到 Cortex-M3 的系统总线端口；以及两个 master 端（M0_DTCM 及 M1_APB_BRIDGE），分别连接 DTCM 及 APB 低速外设部分。slave 端与两个 master 端将在总线矩阵内部进行互联与仲裁，从而使 M3 能够通过预设的地址方便地访问、控制系统的各部件。

```
<!-- Slave interface definitions -->

<slave_interface name="S0_SYS">
  <sparse_connect interface="M0_DTCM"/>
  <sparse_connect interface="M1_APB_BRIDGE"/>
  <address_region interface="M0_DTCM" mem_lo="20000000" mem_hi='3fffffff' remapping='none'/>
  <address_region interface="M1_APB_BRIDGE" mem_lo="40000000" mem_hi='4fffffff' remapping='none'/>
</slave_interface>

<!-- Master interface definitions -->

<master_interface name="M0_DTCM"/>
<master_interface name="M1_APB_BRIDGE"/>
```

图 2.2 CMSDK 总线矩阵地址映射配置

为了达到更好的性能，我们并不直接将低速的 LED、UART 等外设连接到 M3 的 AHB 系统总线上。对于低速外设，我们采用的是 APB 协议，这将涉及到 AHB 协议与 APB 协议的相互转换。得益于 CMSDK 的“AHB to APB”转接模块，我们可以先将 LED、UART 等外设通过 APB 协议挂载到 APB Bridge 模块上，然后再把

APB Bridge 模块挂载到 AHB 总线上。

从下面的小节开始，我们将详细讨论一些关键模块的实施。

2.2 摄像头模块

我们使用 OmniVision 公司的 ov5640 型摄像头实时采集图像数据。经过测试，我们发现当采集图像大小设置为 640×480 时能够在检测速度和检测准确度上取得良好平衡。

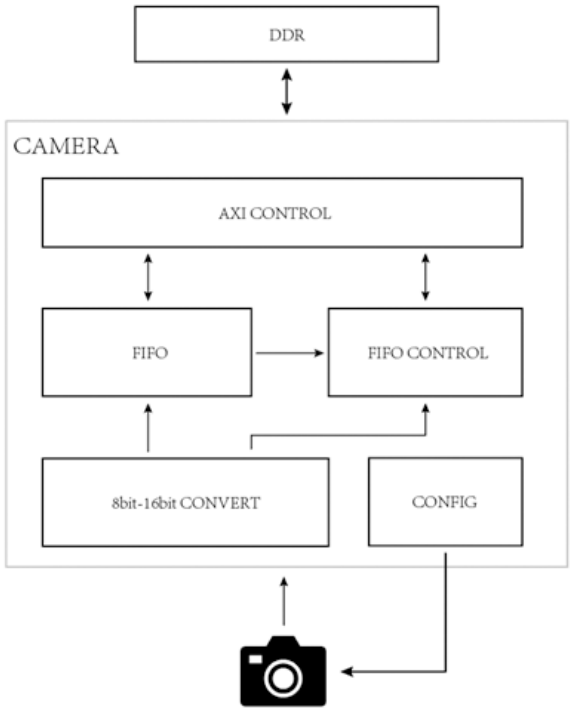

图 2.3 摄像头模块

图 2.3 展示了摄像头模块的详细框图。首先，我们的配置单元（CONFIG）将摄像头的配置信息通过 I2C 协议传输给摄像头外设。摄像头经过配置后将逐帧采集图像数据，摄像头采集到的图像为 RGB565 格式，即每个像素点占据 16 位数据空间（红色通道 5 位、绿色通道 6 位、蓝色通道 5 位）。由于每个时钟仅能传输 8 位数据，因此需要 2 个时钟周期才能完成一个像素数据的传输，为此我们加入了 8bit 转 16bit 的转换单元。

摄像头采集到的图像数据经过位宽转换后将被缓存到 FIFO 单元，FIFO 控制单元负责统计 FIFO 中缓存的数据量，在数据量大于一个 AXI burst 传输量后向

AXI 控制单元发送控制信号，使能 AXI 传输，直到缓存在 FIFO 中的图像数据全部存进 DDR 内。负责摄像头图像数据传输的 AXI 控制单元通过 AXI4 协议与 MIG 模块进行通信，AXI 控制单元作为 master 端向 DDR 写入数据，而 MIG 作为 slave 端接收写入的图像数据，并保存在 DDR 中供处理器使用。

摄像头模块在成功把一帧图像数据写入 DDR 后，会向 Cortex-M3 处理器发出中断请求，将 Cortex-M3 处理器从待机状态唤醒，软件代码将开始执行图像处理的控制流程，配合硬件加速器检测图像中的人脸信息。得到检测结果后，Cortex-M3 处理器再向摄像头模块发出采集新图像的请求，同时进入待机状态，等待下一帧图像的到来，如此循环。

2.3 显示器模块

显示器模块的任务是输出 DDR 内已经处理好并画好人脸检测框的结果图像。其结构如图 2.4 所示，与上述的摄像头模块基本一致，在此不再敷述。

显示器模块要解决的关键问题是如何做到图像读写不冲突？若写图像的地址与读图像的地址是相同的，显示器显示的图像数据有可能是正在从摄像头写入的数据，从而产生图像割裂。因此，我们设置了两个不同的地址，它们交替作为图像的写地址和读地址，实现读写分离，如图 2.5 所示。

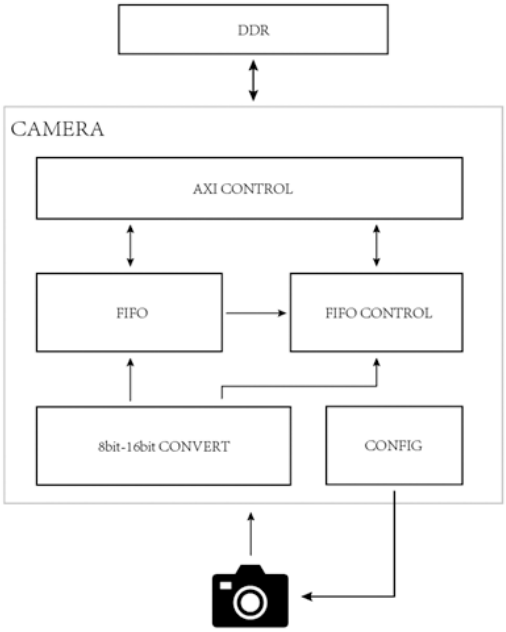

图 2.4 显示器模块

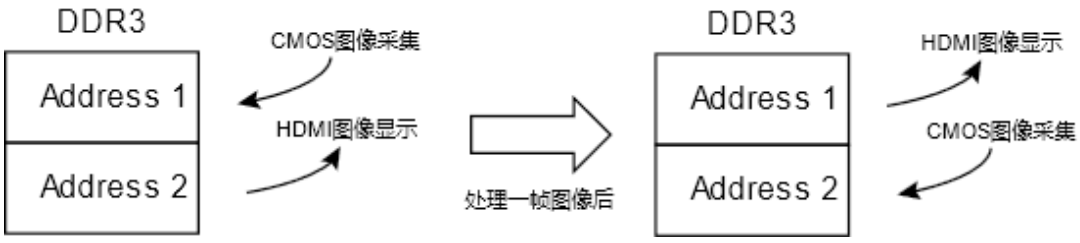

图 2.5 图像读写地址分离与交替变换过程

2.4 APB 外设

本系统的 APB 外设有 LED 灯、UART、动作发生器（Igiter）、计时器（Timer）这几个模块，它们由 APB 总线挂载到系统中，负责实现系统的辅助功能。

2.4.1 LED 灯

开发板上有 4 颗 LED 灯，低电平点亮。LED 的物理地址在 0x40000000，通过软件部分代码往该地址写入一个 32 位的无符号整型数据来控制 4 颗 LED 的点亮状态。

LED 状态显示的是当前检测到的人脸数量，用二进制表示，也就是说，4 颗 LED 最多能显示 15 张人脸。若检测到的人脸数量大于 15，则 4 颗 LED 灯同时点亮。

2.4.2 UART

若检测到的人脸数量超过 15，则可以通过 UART 连接到电脑的串口调试助手中显示具体的数量。除人脸数量外，软件代码还可以将具体的人脸位置坐标、人脸在图像中的尺寸大小等详细信息通过 UART 输出，从而可以查看更详细的检测结果或者监视系统的运行状况。

2.4.3 动作发生器（Igiter）

动作发生器（Igniter）负责接收来自 Cortex-M3 处理器的指令，根据具体的指令，唤醒智能算法硬件加速器或者摄像头模块进入工作状态。

动作发生器（Igniter）的物理地址为 0x40004000，且与 Cortex-M3 处理器、智能算法硬件加速器、摄像头模块连接。当 Cortex-M3 处理器往该地址写入 0x5EA 后，可以唤醒硬件加速器，同时 Cortex-M3 处理器进入待机状态，直至硬件加速器工作完毕。当 Cortex-M3 处理器往该地址写入 0xCA 后，可以唤醒摄像

头模块，同时 Cortex-M3 处理器进入待机状态，直至摄像头模块已经将新的一帧图像数据写入 DDR。

此外，动作发生器（Igniter）还负责显示器模块读取 DDR 地址以及摄像头模块写入 DDR 地址的相互切换，当 Cortex-M3 处理器往该地址写入 0xDA 后，进行地址切换，实现读写地址的分离。

2.4.4 计时器（Timer）

为了统计系统检测人脸所需要耗费的时间，我们还加入了独立于处理器外部的计时器（Timer）模块。计时器接入板载 50MHz 晶振作为时钟源，计时精度可以达到 1 毫秒。

在计时之前，可以通过软件代码将计时器归零，同时启动计时器，并在执行完相应的计算之后读出计时器的计时结果，从而实现对算法执行时间的记录功能。

三、 软件设计

3.1 算法模型简介

我们团队把 Nenad 等人提出来的基于像素强度对比的决策树算法适配到我们的 Cortex-M3 系统中，并取得了优异的识别性能。

该算法以图像特定位置的两个像素点的强度关系作为决策树内部节点的判断依据，每个内部节点都有且只有两个分支，也就是说，这是个二叉决策树模型。每棵决策树只有 6 层（深度为 6），最后一层的叶子节点代表经过这棵决策树之后的累加置信度。

该决策树算法由 468 棵决策树级联构成。在级联结构后端的决策树比前端的决策树的过滤要求更严格，从而只有真正包含人脸的图像区域能顺利通过完整的级联决策树，而负样品被早早地过滤掉，大大节省了计算时间。

级联决策树将在采集到的图像上进行移窗检测，在经过一轮移窗后，根据缩放因子改变检测区的大小，进行新的一轮移窗。每一步移窗都会通过级联决策树判断该位置是否为人脸区域，并进行记录与累加。在所有移窗结束后，记录到的人脸位置将被检查是否存在重叠、重复记录的情况。检查完毕后，将得到最终的人脸数目及位置信息。

该算法的特点是检测十分迅速，经过适配优化后，我们的系统在检测速度和检测质量上取得了十分优秀的效果。

3.2 软件设计

本系统软件部分的设计主要可以分为两方面：启动代码和驱动程序、用户程序。

Cortex-M3 的启动文件需要用汇编语言编写，主要可以分为三个部分：设置堆栈大小、中断向量表以及中断处理，如图 3.1 所示。

```
; <h> Stack Configuration
;   <o> Stack Size (in Bytes) <0x0-0xFFFFFFFF:8>
; </h>

Stack_Size      EQU     0x00100000

                AREA    STACK, NOINIT, READWRITE, ALIGN=3
Stack_Mem       SPACE   Stack_Size
__initial_sp


; <h> Heap Configuration
;   <o>  Heap Size (in Bytes) <0x0-0xFFFFFFFF:8>
; </h>

Heap_Size       EQU     0x00100000

                AREA    HEAP, NOINIT, READWRITE, ALIGN=3
__heap_base
Heap_Mem        SPACE   Heap_Size
__heap_limit
```

```
Reset_Handler   PROC
                EXPORT  Reset_Handler             [WEAK]
                IMPORT  SystemInit
                ;IMPORT  __main
                IMPORT  main
                LDR     R0, =SystemInit
                BLX     R0
                ;LDR     R0, =__main
                LDR     R0, =main
                BX      R0
                ENDP
```

图 3.1 启动文件部分代码

对于在 M3 上运行的软件而言，之前搭建的 SOC 中的 ITCM 就相当于 ROM 空间，软件的启动代码及操作指令都保存在这里。而 DTCM 相当于 RAM 空间，堆栈空间由 RAM 开辟，我们把堆和栈的大小分别设置为 1MB。

Cortex-M3 一共支持 256 个中断，其中包含 240 个外部中断。最高优先级的中断是复位中断，中断请求由 Reset_Handler 处理。进入复位中断后，会先后依次执行系统初始化函数及 main 函数，从而进入主程序。

对于低速外设的访问，我们还需要定义简单的驱动函数，通过对外设所在地址进行读写操作，从而控制外设的动作。以 LED 的控制为例，我们先定义一个结构体，方便访问 LED，如图 3.2。注意，需要在定义 LED 变量时加入 volatile 关键字，禁止编译器对变量访问优化。如果不加这个 volatile 关键字，程序可能

会利用 cache 当中的数据，可能读取的是该变量过时的值，或者不能把新的值写入物理地址中，加了 volatile，就在需要用的时候，程序重新去对应的地址去提取，保证是最新的。

```
typedef struct
{
  volatile      uint32_t  LEDS;
} APB_LED_TypeDef;
```

图 3.2 LED 访问结构体

而用户程序部分，我们采用 C 语言实现所选用的决策树智能算法。在利用 keil 编译软件代码时，需要用 fromelf 工具将 axf 可执行文件转换为面向 Verilog HDL 内存模型的 hex 文件，如图 3.3 所示。hex 文件是由 axf 文件的 32 位指令翻译成 16 进制表示后的可执行代码。得到特定格式的 hex 文件后，我们就可以在 ITCM 的 Verilog 代码中通过 readmemh 语句将编译得到的可执行代码初始化到 ROM 中。需要注意的是，需要在执行 fromelf 工具时添加--vhx 参数（转换为面向 Verilog HDL 内存模型的 hex 文件），否则系统软件将无法启动，这个问题曾阻碍了我们队伍较长时间。

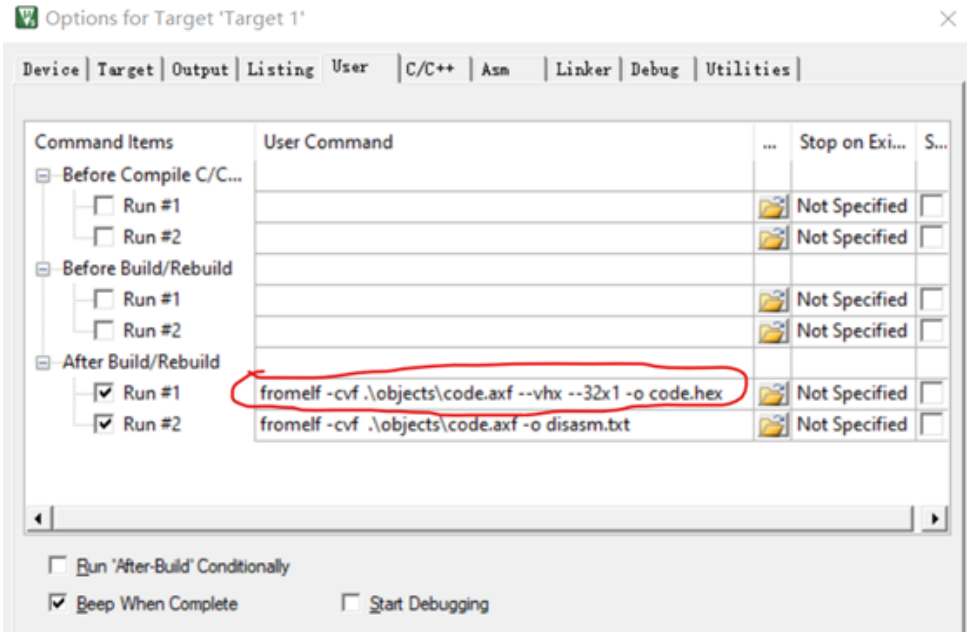

图 3.3 将 axf 文件转换为特定格式的 hex 文件

四、 智能算法硬件加速器设计

通过分析最初版本的人脸检测系统的性能瓶颈，我们团队为检测系统提出了软硬件高度配合与定制化的多项加速方案，包括图像灰度转换单元、片上存储优化、多线程移窗并行加速器等硬件加速模块。它们共同组成了一个高效的加速系统集群，极大提升了本系统检测人脸的速度。

4.1 灰度转换单元

由于我们采用的算法仅需要对比像素强度大小，需要先将摄像头采集到的RGB565 格式的彩色图像预处理为灰度图像。

经过我们的测试发现，原先部署在软件部分并由 Cortex-M3 处理器执行的灰度图像转换过程，每个像素需要进行三个按位与操作、三个整型乘法和两个加法，消耗了大量的时间。为此，我们设计了一个灰度转换硬件单元，将转换操作转移到硬件上实现。将原本在 Cortex-M3 处理器上串行执行的多个操作改为由在 FPGA 上并行执行的灰度转换电路完成。

4.2 片上存储优化

我们在检查决策树硬件加速器的运行时序时发现，受到 DDR 传输延迟的限制，将数据从 DDR 读取到加速器内需要消耗大量的时间。如图 4.1 所示，尽管在优化了 DDR 访问的配置后，加速器从 DDR 读取一个 32 位数据仍然需要耗费 40~50 个时钟，这足以完成近 20 个加速器的常规操作。

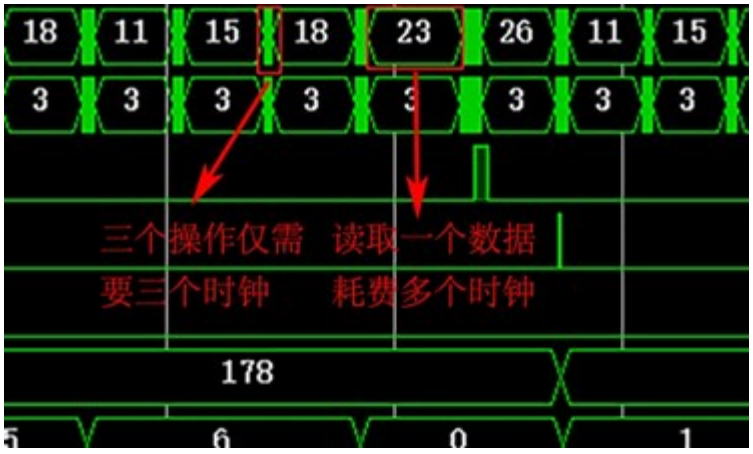

图 4.1 从 DDR 读取数据耗时与加速器常规操作耗时对比

我们通过计算分析发现，算法读取参数文件的数据传输量占据总数据传输量的 40%。为了尽可能减少 DDR 的访问频率，我们将预先训练好的决策树参数文件从 DDR 转移到 FPGA 片上块状存储（Block RAM）内。相比于延时较长的 DDR 访问，加速器对片上存储 Block RAM 的访问仅需要 2 个时钟延时就能完成。

4.3 多线程移窗并行加速器

多线程移窗并行加速器是硬件加速系统的核心成员。它由 AXI 控制单元、加速核、中断控制单元等部件所组成，如图 4.2 所示。

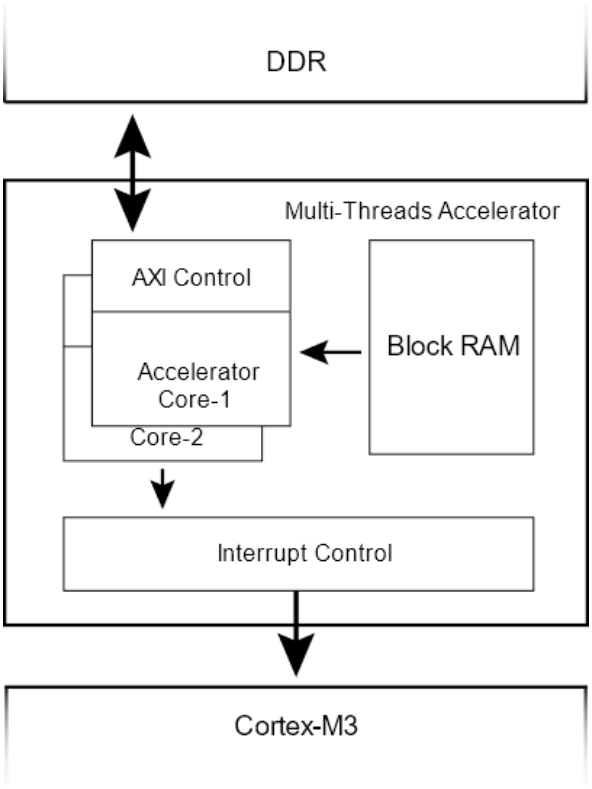

图 4.2 多线程移窗并行加速器框架

4.3.1 加速器的数据流动

为了尽可能减短数据的传输路径，我们把移窗并行加速器直接接在 DDR 上，所有待处理的输入数据都不会经过 Cortex-M3 处理器，加速器将从片上存储模块（Block RAM）以及 DDR 中直接读取所需要的数据，并将检测结果直接存回 DDR。因此，我们为每个加速器核都配备了一个独立的 AXI 控制单元来和 DDR 进行数

据传输，使加速器核能够独立存取 DDR 上的数据。同时，我们采用了双端口的片上存储 Block RAM 单元，为多线程加速器访问片上数据提供带宽支持。

当 Cortex-M3 处理器执行到决策树核心运算时，会通过控制动作发生器（Igniter）模块向多线程移窗并行硬件加速器发出启动请求，硬件加速器开始进行决策树运算，同时 Cortex-M3 处理器进入待机状态，等待加速器工作完成。硬件加速器将直接从 DDR 以及片上存储单元（Block RAM）读取所需要的数据进行决策树算法的加速运算。

当加速器做完决策树的运算并将结果保存到 DDR 后，中断控制单元将向 Cortex-M3 处理器发送中断请求，唤醒 Cortex-M3 处理器。Cortex-M3 处理器开始执行剩余的算法流控制任务。

4.3.2 多线程移窗并行

我们算法中的决策树级联检测器需要将检测区或检测窗口在摄像头采集到的图像上进行多轮移窗检测。常规的移窗操作是逐个移窗位置依次进行的，即只有在一步移窗运算结束后，才进行下一步的移窗，同时运行下一次的级联决策树运算。但由于总移窗检测的次数比较多，逐个串行操作需要耗费比较多的时间去完成所有移窗检测任务。

为此，我们的人脸检测系统引入了多线程移窗的概念，采用多个硬件加速核组成了多线程的并行硬件加速器，每个加速核都可以独立历遍级联决策树，处理一个移窗位置的运算。图 4.3 展示了单线程移窗和多线程移窗的区别，在多线程移窗模式下，加速器可以同时处理多个移窗位置的检测任务，从而极大提高了人脸检测的速度。

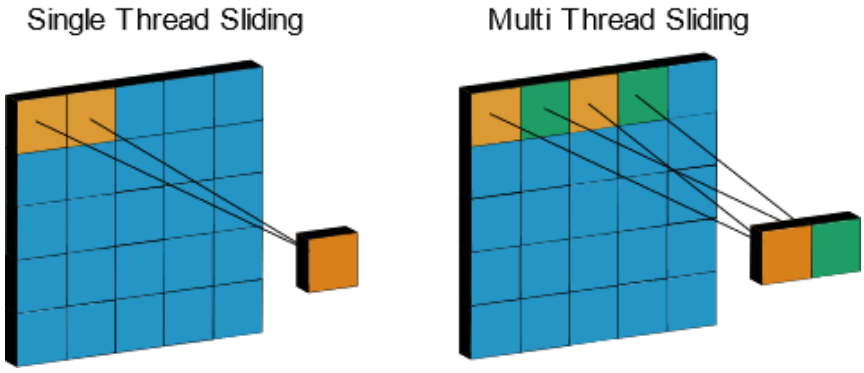

图 4.3 单线程串行移窗与多线程并行移窗对比

4.4 其他加速优化策略

我们团队还持续监测检测系统每一次迭代优化后的性能瓶颈，并根据瓶颈分析结果给下一步优化方向提供参考。因此，除了上述由我们团队独立设计的硬件加速模块外，我们还找到了许多细节性而十分关键的算法加速方法。

其中，我们关注的比较多的仍然是对 DDR 的访问优化。尽管我们为硬件加速器添加了片上存储（Block RAM），并有效减少了 40%的 DDR 访问量，而剩下 60%的 DDR 访问量仍然是消耗计算时间的大头。经过分析发现，绝大部分的 DDR 访问都是从 DDR 读数据，因此在内存界面控制器（MIG）中将读数据的优先级提到写数据之前，使得读数据的操作周期得以减少。

我们对 DDR 进行的数据访问都是随机地址的，为此我们将用户层地址到物理层地址的映射模式设置为更适合数据随机访问的 Bank-Row-Column 模式，进一步优化了 DDR 的访问速度。

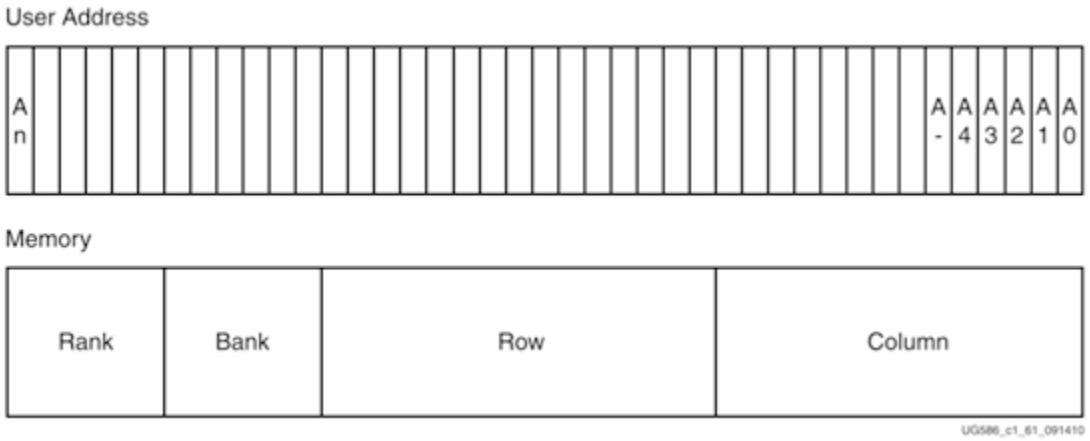

图 4.4 Bank-Row-Column 模式的内存地址映射

在 AXI 互联模块中，我们用同步整数比时钟转换（Synchronous integer-ratio conversion）代替了需要利用更多硬件资源且会引入更大延时的异步时钟转换（Asynchronous clock conversion）。

对于软件部分，我们在 keil 的编译设置中将代码优化等级从无优化提高到三级优化，提升了代码的执行效率。同时，我们将检测区每一次移窗与上一次移窗的重叠面积从 90%降低到 82%，这使得在人脸检测准确率没有明显下降的情况下，大大加快了检测速度。

4.5 加速性能总结

在为我们的人脸检测系统加入了硬件加速器模块以及进行了各方面的综合优化之后，我们为最终版本的人脸检测系统与基准版本的检测系统进行了性能比较。

我们将最初实现的、能首次成功进行人脸识别的检测系统作为性能基准。该基准版本没有采用硬件加速器，仅由 Cortex-M3 处理器执行算法的所有运算和控制流程，所有 IP 未经专门优化设置。在这种未经优化的情况下，检测到一帧人脸图像所需要的时间平均为 3000 ms。

在经过一系列的系统优化后，检测人脸的速度提升了约 54 倍，检测一帧人脸图像平均仅需 55 ms。

图 4.5 展示了采用硬件加速器前后的人脸检测时间对比情况。

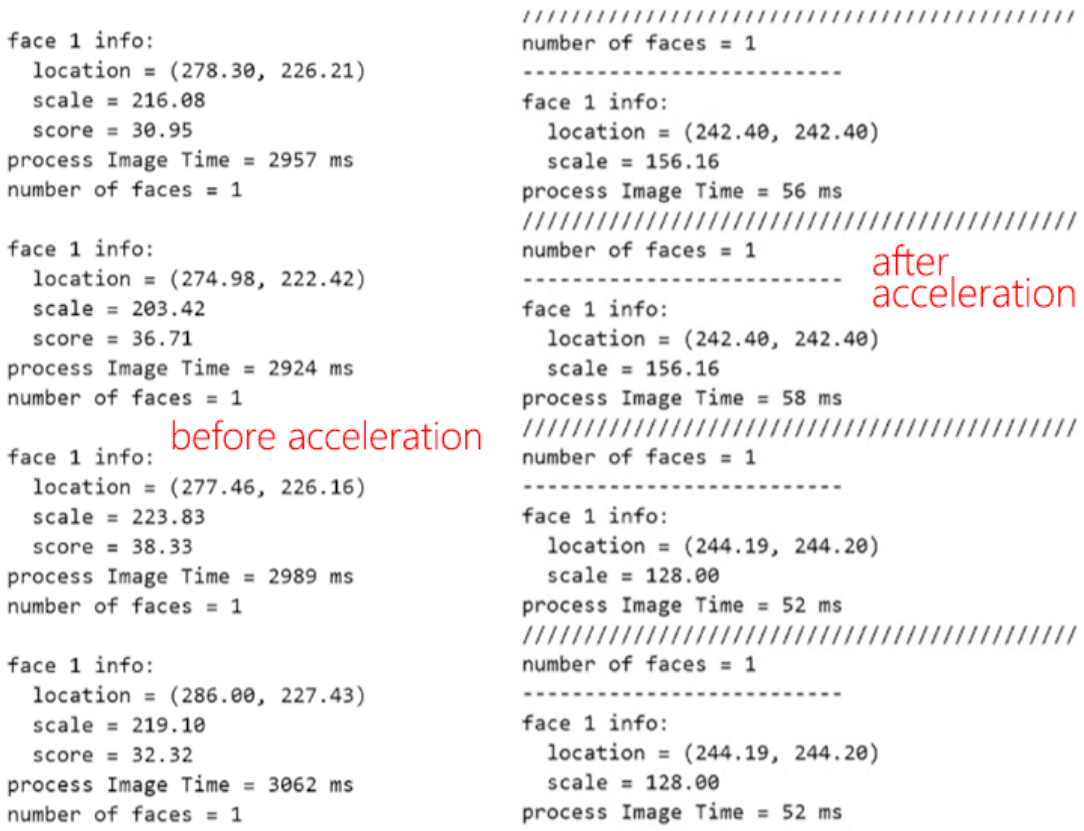

图 4.5 未采用硬件加速器（左）和采用硬件加速器（右）

五、 参考文献

[1] Nenad Marku V S, Frljak Miroslav, Pand V Z I C Igor-S, et al. Object detection with pixel intensity comparisons organized in decision trees[J]. arXiv preprint arXiv:1305.4537, 2013.

[2] ARM. Cortex-M System Design Kit Technical Reference Manual[M].

[3] AMBA 3 AHB-Lite Protocol Specification v1.0[S]. 2010.

[4] AMBA 3 APB Protocol Specification v1.0[S]. 2004.

[5] AMBA AXI and ACE Protocol Specification AXI3, AXI4, and AXI4-Lite ACE and ACE-Lite[S]. 2013.

[6] ARMv7-M Architecture Reference Manual[S]. 2018.

[7] Cortex-M3 Devices Generic User Guide[S]. 2010.

[8] Floating-Point Operator v7.1 LogiCORE IP Product Guide[S]. 2019.

[9] Block Memory Generator v8.4 LogiCORE IP Product Guide[S]. 2019.

[10]HDL Coding Practices to Accelerate Design Performance[M]. 2006.