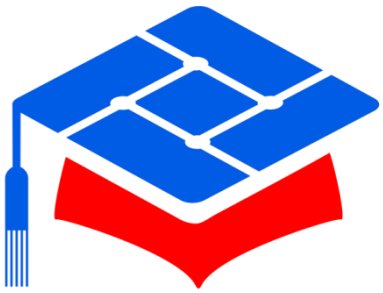


第四届

全国大学生集成电路创新创业大赛

CICIEC

项目设计

参赛题目：RISC-V 子赛题 2

队伍编号：BHL115476

团队名称：燕子

目录

1 概述

本设计完成了基于 ARTY A7 35T 开发平台和蜂鸟 E203 开源 CPU 的一个**低成本**的身份证号识别嵌入式解决方案。开发板通过以太网从上位机的 tftp 服务器中下载 jpg 文件。E203 CPU 运行 jpg 解码程序，完成待识别图片的解码，解码为 RGB888 格式，之后对图片进行字符定位和二值化。最后将提取出来的包含单个数字的标准图片传送到 KNN 加速电路进行加速计算，并得到最终的识别结果。

本设计展现了一个在成本和资源受限的情况下，如何完成较复杂的图像识别任务的思路与具体方案。**开源和低成本是本设计的最大优势，通过减少对片外内存的访问和对 KNN 算法进行硬件加速，我们也尽量提高了系统的整体运行效率。**

硬件设计中，我们在原有 E203 SoC 的基础上，增加了 DDR3 控制器，EMAC，VGA Controller 和 KNN Accelerator。用 ICB 总线将这些外设和 E203 内核连接起来。**我们建立了基于 modelsim 的全系统仿真平台**，包括处理器内核，SoC 外设，以及 DDR3 芯片，用于行为仿真。同时，我们对新添加的 SoC 外设都进行了 FPGA 测试。

软件设计中，我们在 EMAC 外设的基础上，建立了精简的 UDP/IP 协议栈，并以此建立了 TFTP 客户端。Jpg 解码采用了开源的 picojpeg 项目。并且针对打印字体大小形状规整的特点，采用 KNN 算法对字符进行了识别。

在最终的性能测试中，我们从上位机下载了 10 张图片，并对图片中的身份证号进行识别，总用时为 **34s，图像识别的结果全部正确**。

2 需求分析

2.1 项目要求

经过总结，题目的要求为以下五点：

1）获取开源 risc-v IP，正确使用内部总线连接存储器与外设；

2）在硬件平台上正确编译，下载软件，能够操作 GPIO 外设；

3）编写软件实现 ftp/tftp 通信，能够传输图像，并在显示屏上显示；

4）设计算法，对含有身份证号数字的图片，进行片上识别；

5）增加人脸识别和 OCR 功能。

通过对题目的总结，我们从硬件和软件两个方面进行需求分析：

2.2 硬件需求

1） 开源 CPU IP 核：确定一个 risc-v IP 核是整个方案设计的首要问题。根据题目要求，CPU 应具有如下特点：开源协议允许二次开发；有统一的片上总线；面积和资源占用较小；支持 GDB 交互调试；最好有集成的 SoC；最好是 verilog 语言实现，具有好的可读性。

2） 硬件平台：逻辑资源足够实现 CPU 和配套外设；Jtag 接口（比特流文件和软件下载）；包括至少 30 个外接的 IO 接口（外接显示器和逻辑分析仪）；有 phy 芯片和 RJ45 接口（实现以太网应用）；成本要足够的小。

3） EMAC 外设：要实现以太网应用，在开发平台提供 phy 芯片的基础上，我们需要利用 FPGA 的片上逻辑实现数据链路层的功能，并集成到 SoC 中。

4） 显示控制外设，连接 VGA 或 LCD，实现图形显示功能，并集成到 SoC 中。

5） DDR3 控制器：根据组织方提供的图片，图片的像素为 506*319，如果解码为 RGB888 格式的话，最少需要 473kbytes，加上其他软件的内存需求，即使直接解码为灰度图像也会占用大量的内存资源。XC7A100T 的片上内存在 600KBytes 左右，符合要求，但是会额外增加成本。对于 XC7A35T，片上内存是不够的。从成本考虑，我们选择 XC7A35T 的话，就需要利用片外的 DDR3 内存，并增加 DDR3 控制器。

6） 图像处理算法硬件加速电路：从性能角度考虑，我们希望能够通过增加外设或协处理器的方式来增加图像识别的速度。

2.3 软件需求

1） 以 SDK 为基础，建立基础工程。

2） TCP/IP 协议栈或 UDP/IP 协议栈。题目要求实现 FTP 或 TFTP 通信。FTP 是建立在 TCP 基础上的，TFTP 是建立在 UDP 基础上的。需要提供的接口函数包括 eth_ini() 初始化 EMAC 外设和以太网协议栈，eth_receive() 接收 UDP 或

TCP 报文，eth_send() 发送 UDP 或 TCP 报文。

3） TFTP 或 FTP 客户端。需要提供的接口函数包括：send_file_req() 发送文件传输请求，receive_data_frame() 接收数据报文，send_ack() 发送 ack 报文，receive_ack() 接收 ack 报文。

4） Jpg 解码程序，需要查找一些开源项目。需要提供的接口函数包括：decoder_jpg()读取 jpg 文件的二进制数据，并解码为 RGB 或 YUV 格式，并存入内存的指定位置。

5） 图像字符定位，可以通过检测最左端身份证号的开始位置实现。

6） 印刷体数字识别算法：因为是印刷体，字体大小一致，形状规整，因此考虑用 KNN 算法，或神经网络。

3 总体设计方案

针对上面的需求分析，我们最终选择蜂鸟 E203 作为我们 CPU 内核，一个以功耗低资源占用少为优势的两级流水处理器；Arty A7 35T 作为我们的硬件开发平台，该平台是 Digilent 公司设计的低成本 risc-v 开发平台。我们在 E203 SoC 平台集成了 GPIO，SPI，EMAC，VGA Controller 以及 KNN Accelerator 等外设。来满足图像识别任务的硬件需求。

软件方面，我们在 EMAC 的基础上建立了 UDP/IP 协议栈，并建立了 TFTP 客户端；我们选择 picojpeg 一个开源项目[1]，作为我们的 jpg 解码解决方案；在 KNN Accelerator 的帮助下,实现了完整的 KNN 算法,完成了整个图像识别任务。并且，为了减少对片外内存的访问，我们创新地采用位图来压缩存储二进制训练和测试图像（单比特表示单像素）。有限的片上内存就可以存储所有的训练集图像和单个压缩后的测试集图像。该存储方式配合 KNN Accelerator 可以更加高效地完成图像识别任务。

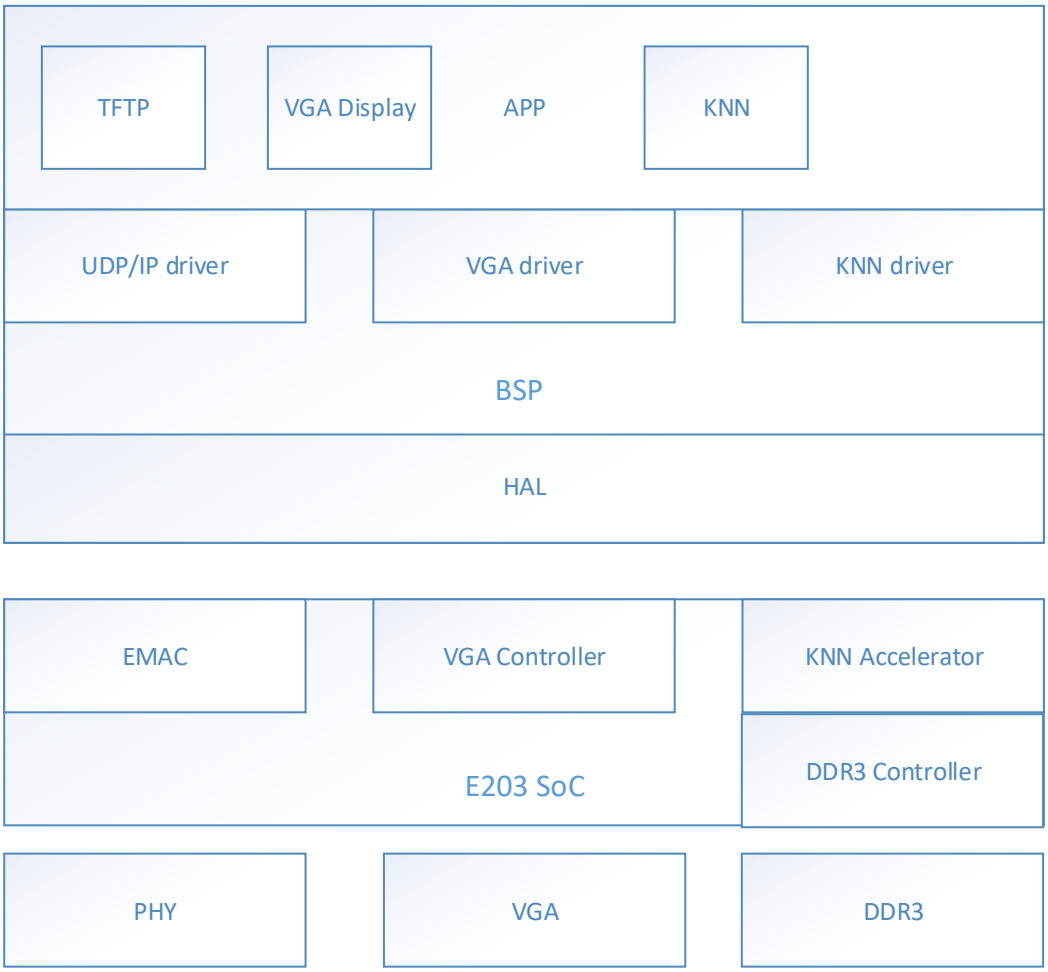


Figure 1：设计方案层次结构

3.1 硬件平台

Arty a7 35T 是 DIGILENT 公司的一款低成本 fpga 开发平台。平台使用 XC7A35T FPGA，含有 33,280 个逻辑单元；1800Kbits 的快速 block RAM；5 个时钟管理器，每一个带有锁相回路；90 个 DSP 逻辑片[2]。

板上提供 RJ45 接口和 10/100Mbps 以太网 phy 芯片，可以建立以太网应用；USB-JTAG 编程电路，可以下载比特流文件；100MHz 晶振；具有 16 位总线@667MHz 的 256MB DDR3L，可用于扩展内存；4 个开关，4 个按钮，4 个 LED，4 个 RGB LED 可用于 GPIO 验证实验；4 个 Pmod 连接器和 Arduino/chipKIT 扩展板连接器作为扩展接口，可以扩展作为 JTAG，VGA 以及摄像头接口[2]。

同时因为通常的软件调试采用的是 20 针或 10 帧的 JTAG 接口，我们自制了一个蜂鸟 JTAG 转 PMOD 的转接板。比起杜邦线连接更加可靠方便。

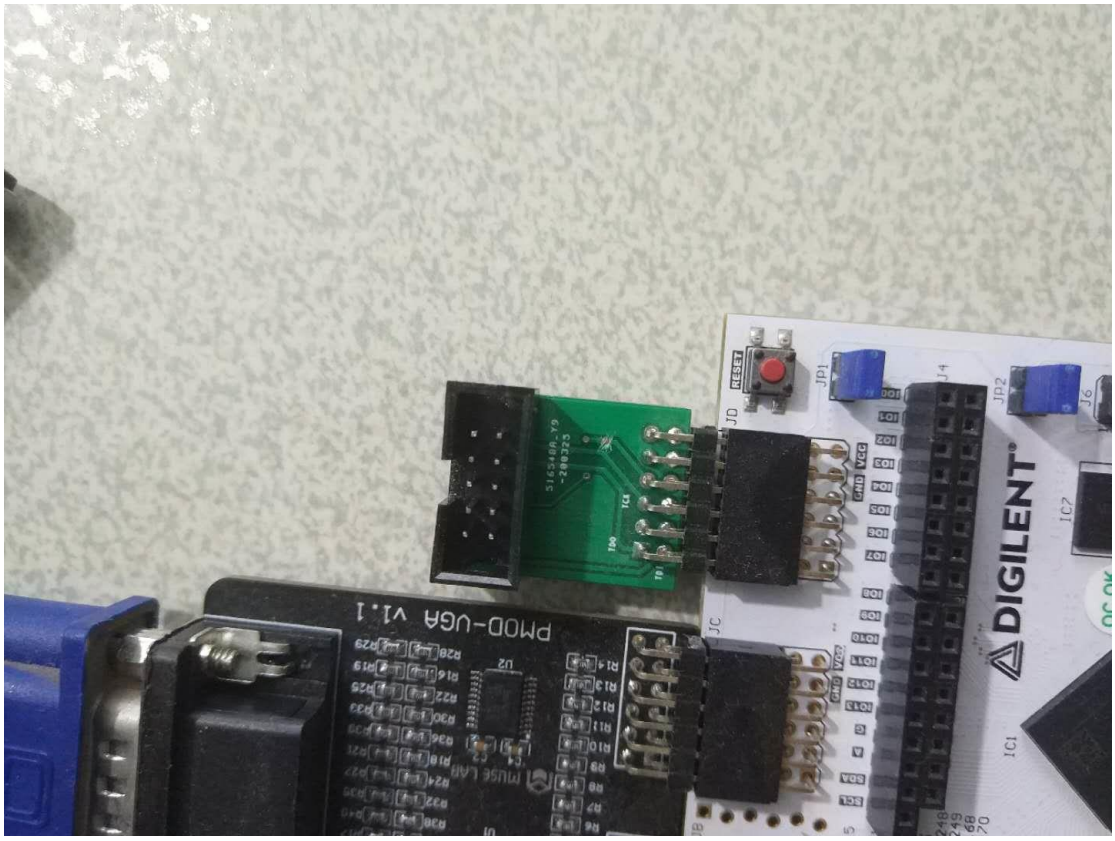


Figure 2：自制 jtag pmod 转接板

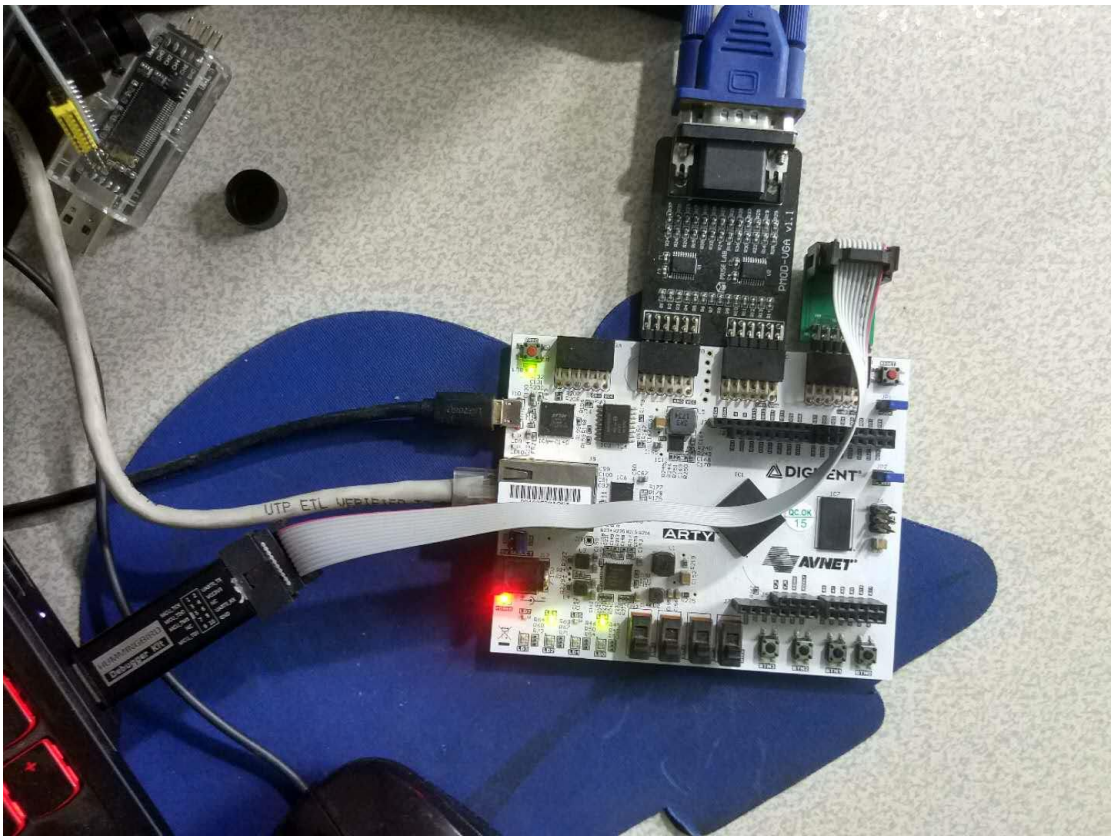


Figure 3：项目整体硬件连接

整体的硬件连接，jtag/uart 接口用于下载比特流文件和 uart 通信，JB 和 JC 两个 pmod 接口连接 VGA 显示器，JD pmod 接口连接 jtag 调试器下载软件程序，RJ45 接口连接网线。

3.2 蜂鸟 E203 SoC 概述

蜂鸟 E203 主要面向极低功耗与极小面积的场景，是一款对标 Cortex-M0 或 M0+的处理器。处理器核采用两级流水线结构；支持 RV32I/A/M/C 指令子集的配置组合，支持机器模式；提供标准的 JTAG 调试接口；私有的 ITCM 和 DTCM，实现指令与数据的分离存储[3]。有关 E203 流水线的更详细介绍请见 doc/学习文档/ddr3.md。

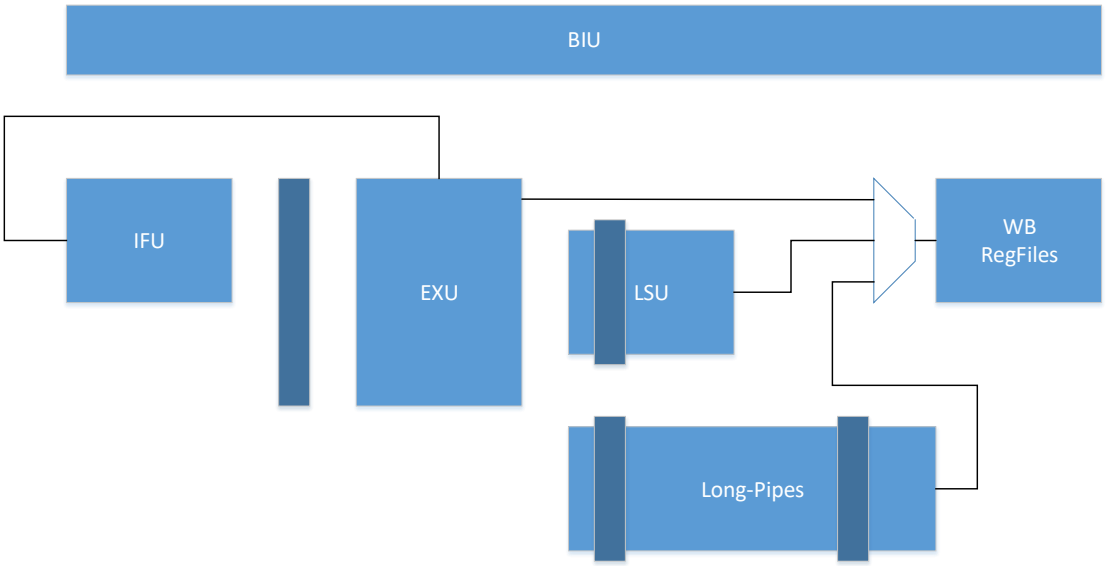


Figure 4：E203 流水线结构

E203 处理器配套 SoC 提供紧耦合系统 IP 模块，包括中断控制器，计时器，UART，QSPI 和 PWM 等；中断接口通过 PLIC 与处理器核相连；提供 ICB 系统总线用于访存内存数据和访问外设；提供调试接口用于 SoC 级别的 JTAG 调试。有关 ICB 总线的详细信息请见 doc/学习文档/ddr3.md。

3.3 硬件系统设计简介

根据需求分析，我们在 E203 SoC 的基础上增加了 DDR3 控制器，EMAC 外设，VGA Controller，以及 KNN Accelerator。我们将升级后的 SoC 命名为 E203Plus SoC。

DDR3 控制器使用 Xilinx 的 MIG IP 核，采用 native 接口。DDR3 控制器通过 LSU 模块预留的 dcache 接口连接到 LSU 模块。Dcache 接口采用 ICB 总线，我们设计了 ICB_Bridge 模块完成了 MIG IP 核的 native interface 到 ICB interface 的接口协议转换和跨时钟域功能。DDR3 扩展内存占用的物理内存空间为 0xA000_0000 ~ 0xA7FF_FFFF。可以直接按字，半字，字节访问。有关 DDR3 扩展内存的详细信息请见 4.2 DDR3 内存扩展部分。

EMAC 外设使用 Xilinx 的 EthernetLite IP 核，AXI 接口。通过 ICB2AXI 模块转换为 ICB 接口协议并与私有外设模块相连，最后通过 BIU 模块连接到 LSU。EMAC 模块占用的物理内存空间为 0x1005_0000 ~ 0x1005_1FFF。有关 EMAC 外设的详细设计信息请参考 4.3 EMAC 外设部分。

KNN Accelerator 模块接收标准大小的训练图片和测试图片各一张，并返回他们之间的距离，采用状态机实现。KNN Accelerator 采用 ICB 总线接口，连接到私有外设模块，再通过 BIU 连接到 LSU。KNN Accelerator 模块占用的物理内存空间为 0x1004_1000~0x1004_1FFF。有关 KNN Accelerator 的详细信息，请参见 4.4 KNN 硬件加速带电路。

VGA Controller 模块包含 VGA Display 和 VGA driver 两个子模块，VGA Display 模块接收来自 CPU 发送的图像或字符信息，并将对应像素点的颜色信息发送给 VGA driver 模块。VGA driver 模块控制 VGA 的通信时序。采用 640x480 分辨率，对外提供 ICB 总线接口。VGA Controller 模块对应的物理内存空间为 0x1001_0000 ~ 0x1001_0FFF。有关 VGA Controller 的详细信息，请参见 4.5 VGA 显示控制电路。

下图为 E203 SoC 示意图，其中红框部分为在原有 SoC 基础上增加和修改的部分。

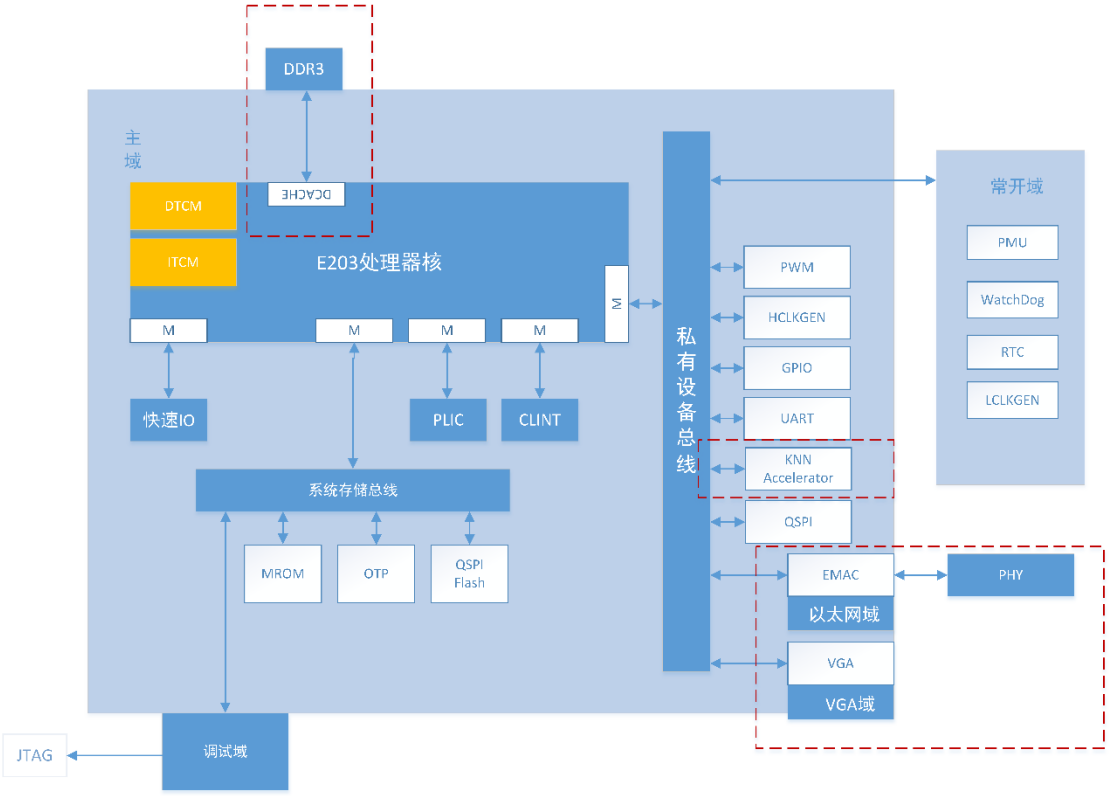


Figure 5：E203Plus SoC **组成**

下表为 E203Plus SoC 物理内存空间分配。

Table 1：E203Plus SoC **物理内存分配**

总线分组	组件	地址区间
处理器核直属	CLINT	0x0200_0000 ~ 0x0200_FFFF
	PLIC	0x0C00_0000 ~ 0x0CFF_FFFF
	ITCM	0x8000_0000 ~ 0x8000_FFFF
	DTCM	0x9000_0000 ~ 0x9000_FFFF
	Dcache	0xA000_0000 ~ 0xA7FF_FFFF
系统存储接口	调试模块	0x0000_0000 ~ 0x0000_0FFF
	Mask-ROM	0x0000_1000 ~ 0x0000_1FFF
	Off-Chip QSPI0	0x2000_0000 ~ 0x0003_FFFF

	On-Chip OTP Read	0x0002_0000 ~ 0x0003_FFFF
	常开域	0x1000_0000 ~ 0x1000_7FFF
私有外设接口	HCLKGEN	0x1000_8000 ~ 0x1000_8FFF
	VGA Controller	0x1001_0000 ~ 0x1001_0FFF
	GPIO	0x1001_2000 ~ 0x1001_2FFF
	UART0	0x1001_3000 ~ 0x1001_3FFF
	QSPI0	0x1001_4000 ~ 0x1001_4FFF
	PWM0	0x1001_5000 ~ 0x1001_5FFF
	UART1	0x1002_3000 ~ 0x1002_3FFF
	QSPI1	0x1002_4000 ~ 0x1002_4FFF
	PWM1	0x1002_5000 ~ 0x1002_5FFF
	QSPI2	0x1003_4000 ~ 0x1003_4FFF
	PWM2	0x1003_5000 ~ 0x1003_5FFF
	KNN Accelerator	0x1004_1000 ~ 0x1004_1FFF
	EMAC	0x1005_0000 ~ 0x1005_1FFF

下表为系统设计资源占用：

Table 2：E203Plus Soc **资源占用**

Resource	Utilization	Available	Utilization %
LUT	16783	20800	80.69
LUTRAM	628	9600	6.54
FF	15806	41600	38.00

BRAM	34	50	68.00
IO	138	210	65.71
BUFG	12	32	37.50
MMCM	3	5	60.00
PLL	1	5	20.00

3.4 软件系统设计简介

根据需求分析，我们在硬件平台和 HBird-E-SDK 的基础上，建立了 UDP/IP 协议栈和 TFTP 客户端，添加了 VGA dirver 功能模块和 KNN driver 功能模块，用于方便对外设进行操作。使用开源项目 picojpeg 进行 jpg 文件解码，使用 KNN 算法对图像中的数字进行识别。

整体的应用层软件工作流程为：

1）下载图片，存入内存中的指定位置

2）使用 picojpeg 中的函数对 jpg 文件进行解码

3）图像灰度化

4）字符定位

5）根据字符定位结果提取一个字符的标准图片

6）送入 KNN Accelerator 模块与训练集图片进行比较

7）选择距离最短的训练集字符作为识别结果

8）是否完成 18 个字符识别，是进行下一步，否则返回 5）

9）是否完成了 10 张图片的识别任务，是进行下一步，否则返回 1）

10）将识别结果的字符信息通过 TFTP 协议发送给上位机

11）将识别结果的字符信息发送给 VGA 模块进行显示

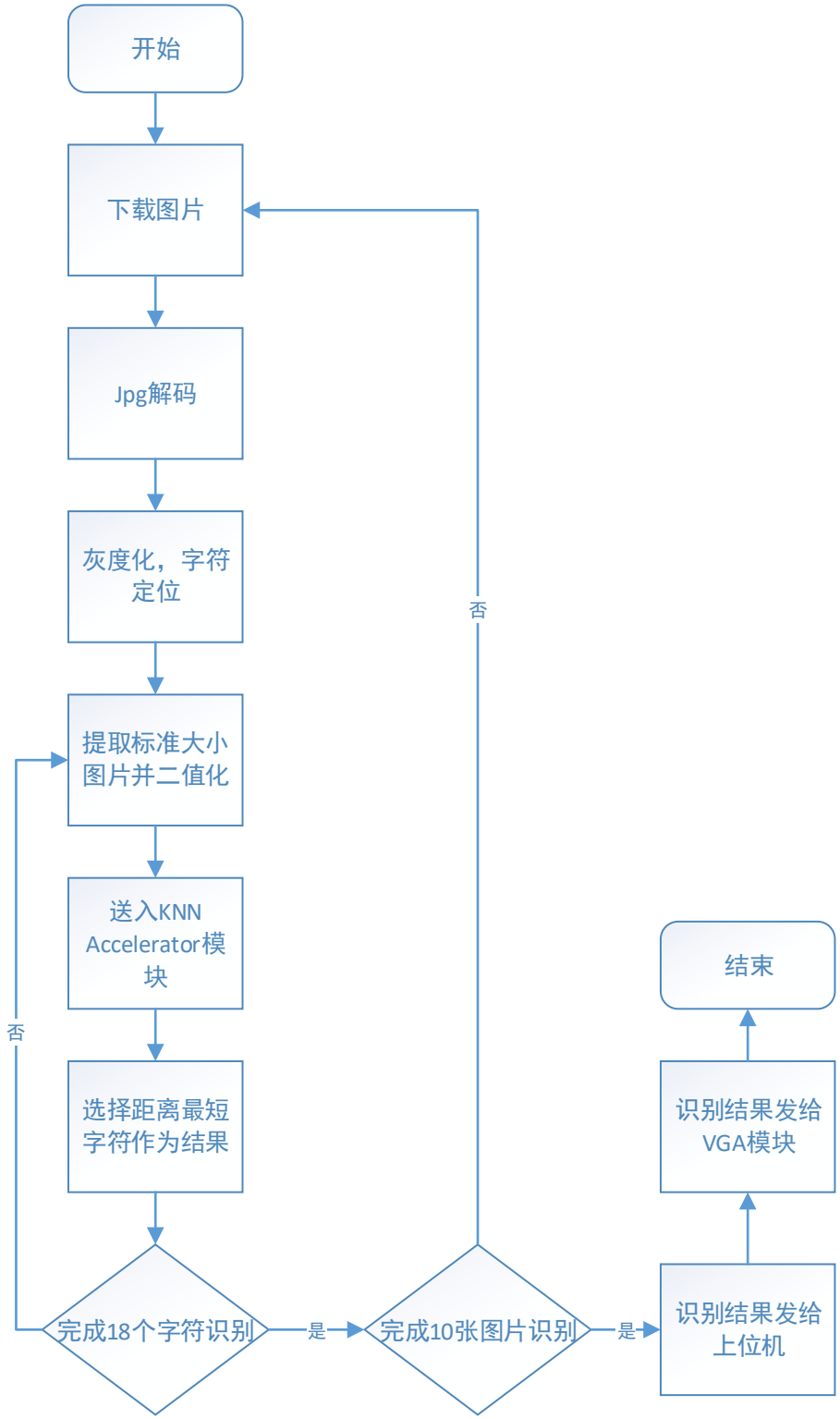


Figure 6：程序流程图

4 硬件分系统设计方案

根据需求分析，我们在 E203 SoC 的基础上增加了 DDR3 内存扩展，EMAC 外设，VGA Controller，以及 KNN Accelerator。

4.1 时钟

使用板载的 100MHz 晶振作为时钟源，通过 MMCM1 模块产生 eth_ref_clk (25M)，ref_clk_i (200M)和 sys_clk_i (166.667M)。其中 eth_ref_clk 为板载的 phy 芯片和 VGA Controller 模块提供时钟；ref_clk_i 和 sys_clk_i 为 MIG IP 核提供时钟。在 MIG IP 核内部通过 pll 产生了 ui_clk (83.333M)。我们将 ui_clk 作为时钟源送入 MMCM2 模块，产生 CLK16MHz (16.667M)，CLK8388 (8.388M)。其中 CLK16MHz 是 CPU 的运行时钟，CLK8388 用于产生 RTC 时钟。

4.2 DDR3 内存扩展

4.2.1 功能描述

将板载 DDR3 加入内存空间，实现内存扩容。内存扩展功能通过两个模块实现，分别是 DDR3 控制器模块和 ICB_Bridge 模块。DDR3 控制器模块使用 Xilinx MIG IP 核；ICB_Bridge 模块完成了 MIG IP 核的 native interface 到 ICB interface 的转换，同时也是一个 CDC 电路。

4.2.2 地址映射

增加的内存空间为 0xA000_0000 ~ 0xA7FF_FFFF，共 128M。接受按字节，半字和字进行访问。

4.2.3 接口定义

有关 MIG IP 核的接口定义请参见官方 product guide ug586。包括两个接口一个是面向应用的 user interface，另一个是对应 DDR3 的物理层 phy 接口[4]。ICB_Bridge 也有两个接口，分别是 user interface 和 ICB interface。详细接口信息如下：

Table 3：ICB Bridge 模块 IO 接口定义

Name	Direction	Bits	Description
Cmd_valid	I	1	主设备发送读写请求
Cmd_ready	O	1	从设备返回读写接收信号
Cmd_read	I	1	读或者写操作指示
Cmd_addr	I	32	读写地址
Cmd_wdata	I	32	写操作数据
Cmd_wmask	I	3	写操作字节掩码
Rsp_valid	O	1	从设备发送读写反馈请求信号

Rsp_ready	I	1	主设备返回读写反馈接收信号
Rsp_rdata_r	0	32	读反馈数据
App_addr	0	28	读写操作地址
App_cmd	0	3	读或者写操作指示
App_en	0	28	读或者写操作使能
App_wdf_data	0	128	写操作数据
App_wdf_end	0	1	写操作结束标志
App_wdf_wren	0	1	写操作使能信号
App_rd_data	I	128	读操作反馈数据
App_rd_data_end	I	1	读操作反馈结束标志
App_rd_data_valid	I	1	读操作反馈数据有效标志
App_rdy	I	1	控制器准备接收读写操作命令
App_wdf_rdy	I	1	控制器准备接收写操作数据
App_wdf_mask	0	1	写操作字节掩码
Ui_clk	I	1	DDR3 控制器时钟
Cmd_clk	I	1	ICB 侧时钟
Myrst	I	1	复位信号，低有效

有关接口时序请参考，doc/ICBBridge spec 文件。

4.2.4 本系统在全局的位置

在 E203 内核的 LSU 中，给 DCACHE 留了 ICB 接口，但外部并没有接 DCACHE 设备，我们就利用这个接口来做内存扩容。ICB 接口参见 e203_lsu.v。同时我们在 e203_lsu.v 实例化了我们的桥接协议 ICBBridge。有了桥接协议之后 MIG 的 83MHz 的 native interface 就变成了 16.6MHz 的 ICB interface。从而能够从 LSU 接收访存指令，并能够将读到的数据传递给写回模块。

之后我们需要把 native interface 的信号引出到顶层模块。并在顶层模块实例化 DDR3 控制器。

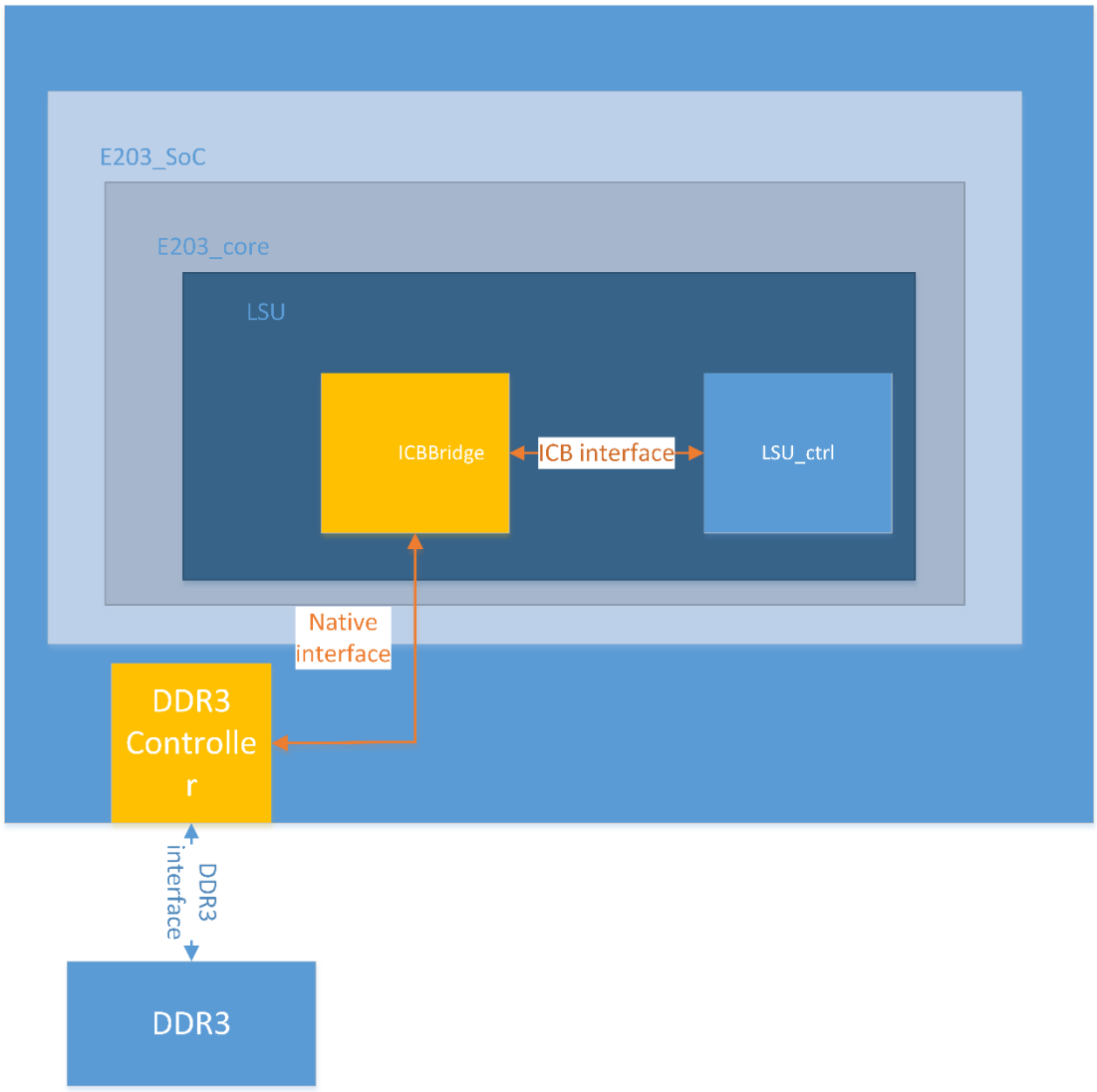


Figure 7：DDR3 内存扩容连接关系

4.2.5 算法设计与实现框图

对于原有 E203 工程增加了两个模块，一个是 DDR3 控制器 IP 核，另一个是 ICB 协议到 DDR3 控制器的桥接转换模块。DDR3 控制器采用了 MIG IP 核。对于 DDR3 控制器 IP 核的接口时钟却是 IP 核自己生成的时钟。这里面就可能存在跨时钟域的问题。同时接口的数据位宽是 128 位的，与 CPU 内核的 32 位位宽不匹配。为此我们设计了 ICB Bridge 模块完成跨时钟域和协议转换的功能。

关于 ICB Bridge 模块的设计这里简单做简单描述，详细信息请参考《ICBBridge 设计文档》和源代码 ICBBridge.v。设计的主要思想是利用了两个时钟的整数倍关系。输入用了一个深度为 1 的 FIFO 进行缓冲，FIFO 的空满作为 cmd_ready 信号，当 FIFO 为空且 cmd_valid 有效时，使能写 FIFO。写指针

的改变自动使 ready 信号拉低，这个时候输入 FIFO 进入了等待状态。直到内部模块和 DDR3 controller 完成通信，发出一个可读信号，在下一个内核时钟上升沿完成读 FIFO 指针改变，FIFO 再次为空可以接收新的 valid 指令。

输出 FIFO 同理，当内部模块完成与 DDR3 controller 的通信后，会发出一个可写信号，在内核时钟的上升沿，输出 FIFO 写指针加一，FIFO 进入满状态，同时 rsp_valid 表示这个满状态，如果下个时钟周期 rsp_ready 有效，则读指针加 1，再次将输出 FIFO 变为空状态，rsp_valid 也失效。

整个内部模块通过一个状态机实现，分为 IDLE，READ，WRITE，WAIT，WAITRSP，RSP 六个状态。

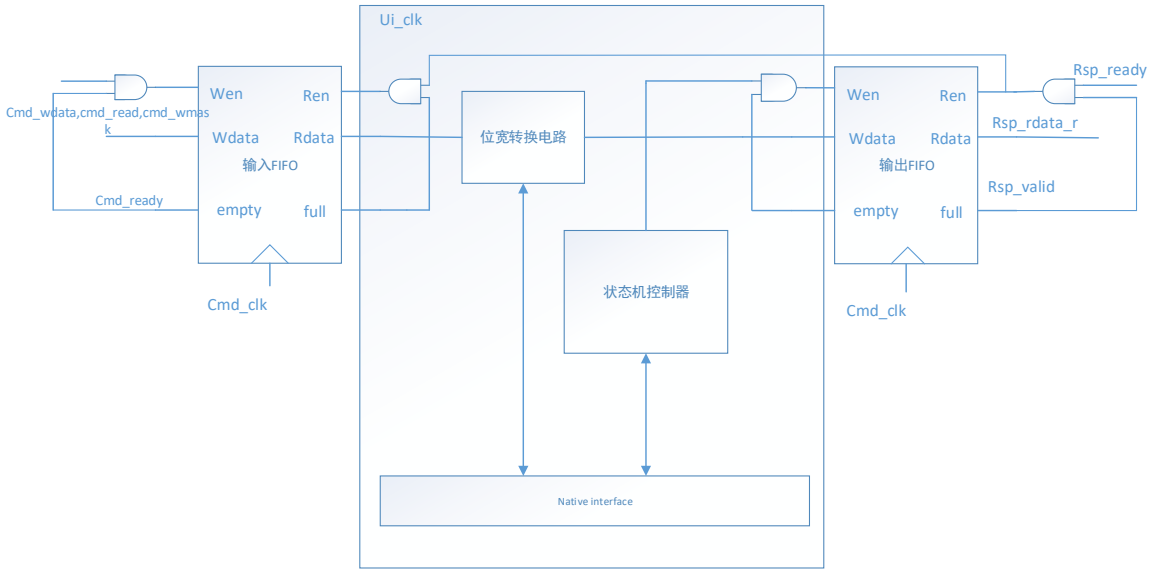


Figure 8：ICB Bridge **实现框图**

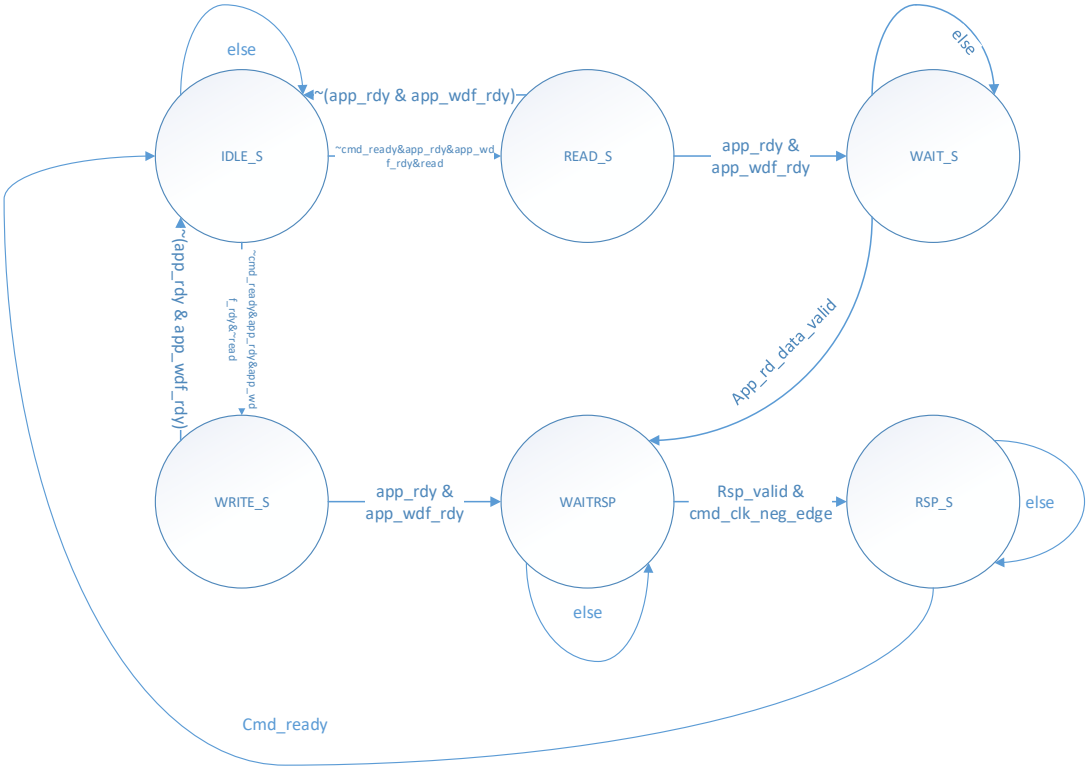


Figure 9：ICB Bridge **状态转移图**

4.2.6 行为仿真

我们建立了基于 modelsim 的全系统仿真平台，包括处理器内核，SoC 外设，以及 DDR3 芯片。

行为仿真使用汇编程序，从 0xA000_8000 到 0xA000_8064 分别用 sw，sh 和 sb 指令存入 0 到 100 这 101 个数再通过 lw，lh 以及 lb 读出来。汇编程序在工具链的作用下生成.verilog 文件。.verilog 文件被 tb 程序读入 itcm 中，然后取指模块逐条取指执行指令。

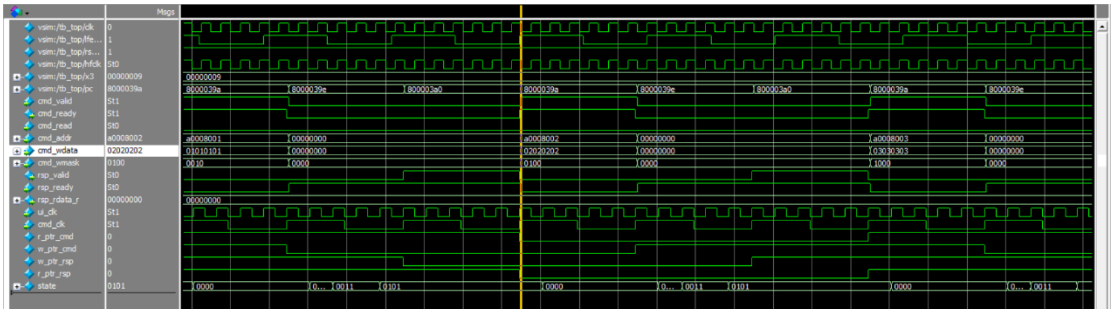


Figure 10：**按字节存入数据行为仿真**

图中可以看到 CPU 正在向地址为 0xA0008001 到 0xA0008003 的位置写入 1，2，3。

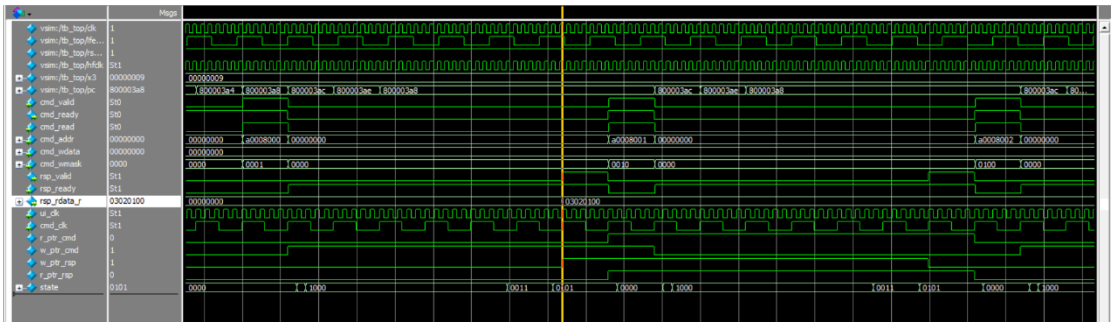


Figure 11：按字节读出数据行为仿真

从图中可以看出 CPU 从内存 0xA0008000 到 0xA0008003 的位置分别读出了 0，1，2，3。

通过检查整个读入读出仿真波形，可以发现 0 到 100 的数字都被正确读出了。

4.2.7 FPGA 测试

将生成好的比特流文件下载到 fpga 开发板后，通过 jtag 接口向 cpu 下载程序。程序执行的任务是从 0xa000_0000 地址开始往后依次存入 0 到 99（分别以字，半字，字节方式存入）再以字，半字，字节方式读出。观察数据是否被正确读出。

Figure 12：DDR 内存扩展 FPGA 原型测试

可以看到无论是按字，半字还是字节访问，都可以正常的存取数据。

4.3 EMAC 外设

4.3.1 功能描述

完成以太网的数据链路层功能。对外通过 MII 接口结合物理层 phy 芯片通

信，对内通过 ICB 总线和 cpu 的 IP 层软件协议进行通信。并能够产生中断信号被 PLIC 模块接收。

4.3.2 地址映射

发送缓冲区是从 0x1500_0000 到 0x1500_07FF 的位置

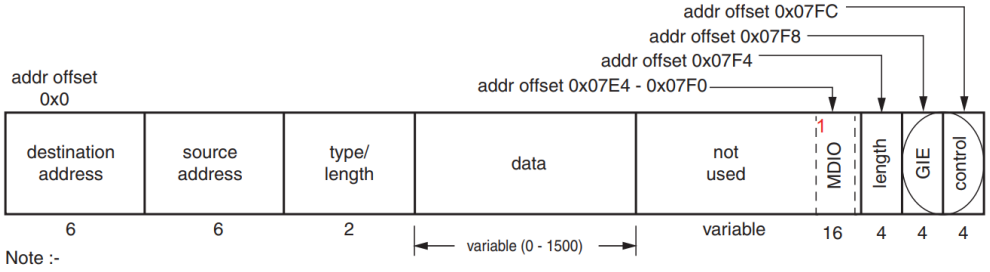


Figure 13：EMAC 外设发送缓冲区地址映射

接收缓冲区是从 0x1500_1000 到 0x1500_17FF 的位置

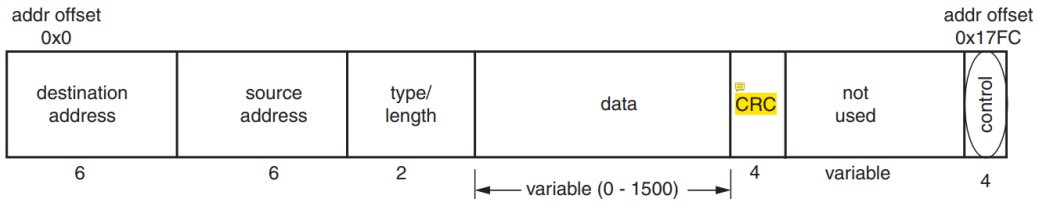


Figure 14：EMAC 外设接收缓冲区地址映射

寄存器具体含义见下表：

Table 4：EMAC 外设寄存器

Address Offset	Register Name	Description
1500_07E4h	MDIOADDR	MDIO address register
1500_07E8h	MDIOWR	MDIO write data register
1500_07ECh	MDIORD	MDIO read data register
1500_07F0h	MDIOCTRL	MDIO control register
1500_07F4h	TX Ping Length	Transmit length register for ping buffer
1500_07F8h	GIE	Global interrupt register
1500_07FCh	TX Ping Control	Transmit control register for ping buffer
1500_0FF4h	TX Pong Length	Transmit length register for pong buffer
1500_0FFCh	TX Pong Control	Transmit control register for pong buffer
1500_17FCh	RX Ping Control	Receive control register for ping buffer
1500_1FFCh	RX Pong Control	Receive control register for pong buffer

有关寄存器各比特位的含义请参考 Xilinx 官方 product guide pg135[5]。

4.3.3 接口定义

Table 5：EMAC 模块 IO 接口定义

Name	Direction	Bits	Description
Cmd_valid	I	1	主设备发送读写请求

Cmd_ready	O	1	从设备返回读写接收信号
Cmd_read	I	1	读或者写操作指示
Cmd_addr	I	32	读写地址
Cmd_wdata	I	32	写操作数据
Cmd_wmask	I	3	写操作字节掩码
Rsp_valid	O	1	从设备发送读写反馈请求信号
Rsp_ready	I	1	主设备返回读写反馈接收信号
Rsp_rdata	O	32	读反馈数据
ip2intc_irpt	O	1	上升沿触发的中断信号
Phy_tx_clk	I	1	发送时钟来自 phy 芯片
Phy_rx_clk	I	1	接收时钟来自 phy 芯片
Phy_rx_data	I	4	来 MII 接口的接收数据
Phy_tx_data	O	4	向 MII 接口发送的数据
Phy_dv	I	1	接收数据有效
Phy_rx_er	I	1	接收数据出错
Phy_tx_en	O	1	发送数据使能
Phy_mdc	O	1	MDIO 接口时钟
Phy_mdio_i	I	1	MDIO 接口输入数据
Phy_mdio_o	O	1	MDIO 接口发送数据
Phy_mdio_t	O	1	MDIO 接口使能输出

4.3.4 本系统在全局的位置

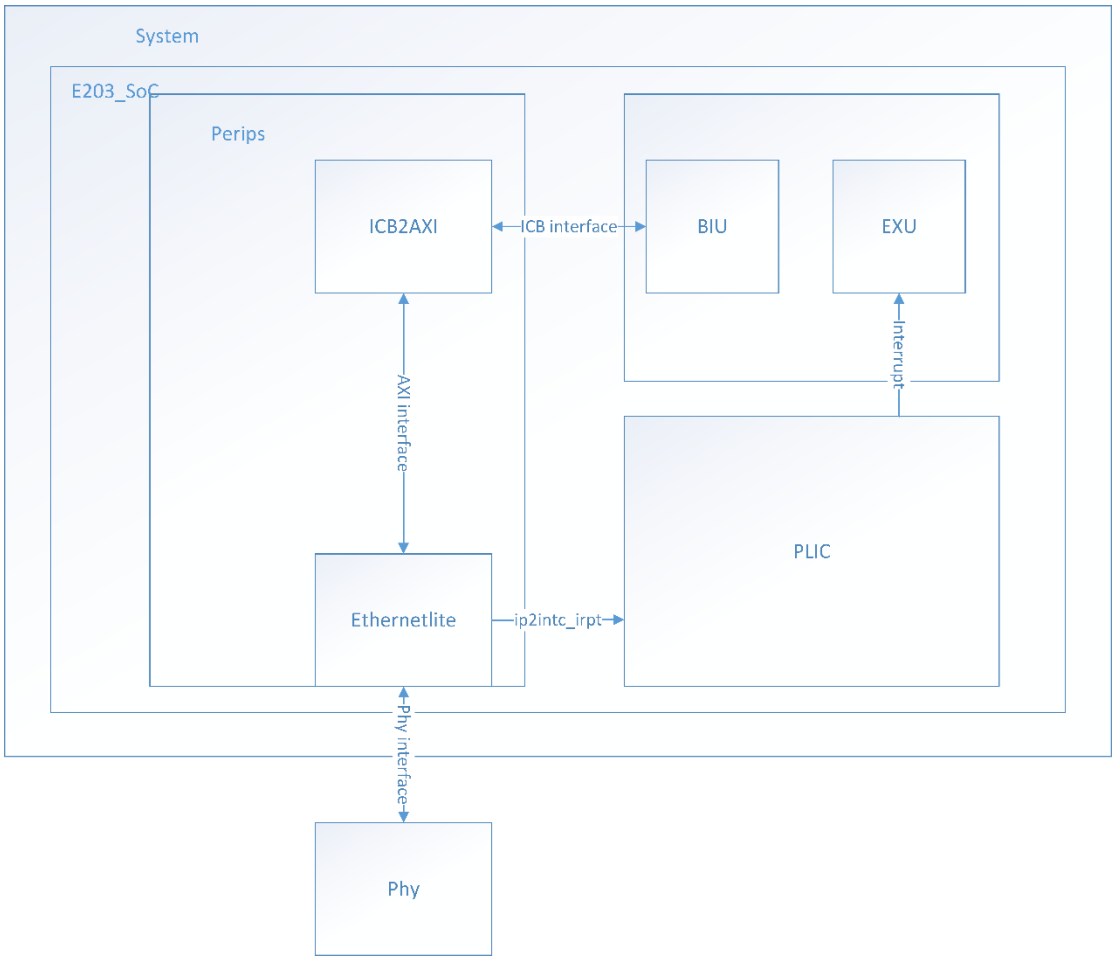


Figure 15：EMAC 模块连接关系

4.3.5 算法设计与实现框图

以太网传输层通过 Xilinx EthernetLite IP 核实现，该 IP 核用户层接口为 AXI 总线协议。我们通过 ICB2AXI 模块将，将接口协议转换为 ICB，然后连接到私有外设模块。私有外设模块通过 BIU 最终连接到 LSU 模块。

4.3.6 行为仿真

我们只对 EthernetLite IP 核的功能进行了行为仿真，仿真为环回测试，MAC 层发送的数据不从 MII 接口发出而是直接进入接收端口。

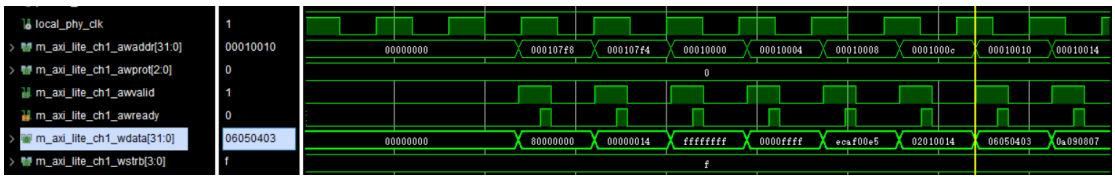


Figure 16：以太网发送行为仿真

通过 AXI 端口向发送缓冲区存入数据。

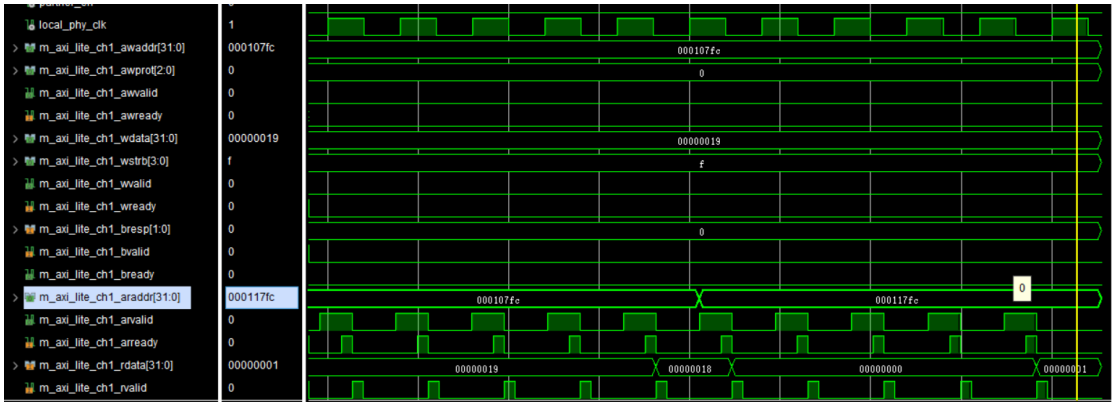


Figure 17：**以太网接收行为仿真**

向发送 status 寄存器写 0x19 表示使能环回，使能发送中断，开始发送，最终从接收端 status 寄存器检测到了 0 到 1 的跳变，说明接收成功。

4.3.7 FPGA 测试

FPGA 测试内容也为环回测试，首先再 c 程序中准备好报文，将报文从 EthernetLite 发送端口发出，然后直接从接收端口接收。并把发送的报文内容和接收的报文内容，都通过串口打印到上位机。详见 sim/EMAC/EMAC.c 和 sim/EMAC/EMAC_irq.c。

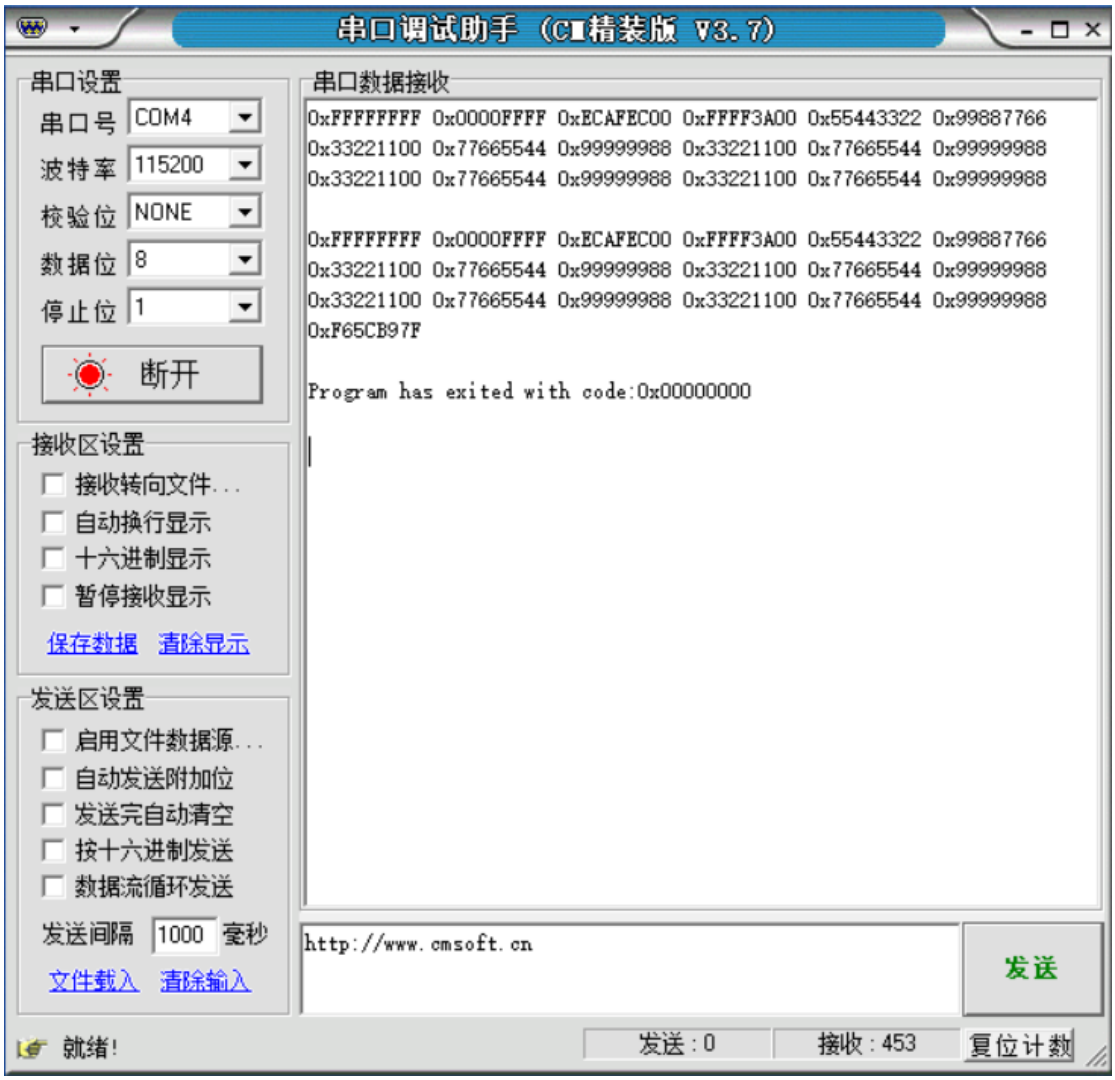


Figure 18：EMAC 外设 FPGA 原型测试

我们可以看到，接收的报文内容和发送的报文内容是一致的，说明 EthernetLite IP 核加入 SoC 是初步成功的。详见 ethernet 文件夹下 EMAC.c 和 EMAC_irq.c。

之后我们将接收方式从轮询改为中断，仍然得到了一致的结果。

4.4 KNN 硬件加速电路

4.4.1 功能描述

KNN Accelerator 是计算图片间距离的加速电路。图片大小为 37×22（行×列），二值图像，单个像素点用单个比特表示。接口采用 ICB 总线接口。时钟为 60ns。

4.4.2 地址映射

KNN 加速电路占据 0x1004_1000~0x1004_1FFF 的地址空间：其中 0x1004_1000 到 0x1004_1025 为测试图片的存储位置；0x1004_1100 到 0x1004_1125 为训练图片的存储位置；0x1004_1FF8 为结果寄存器；0x1004_1FFC 为 status 寄存器。

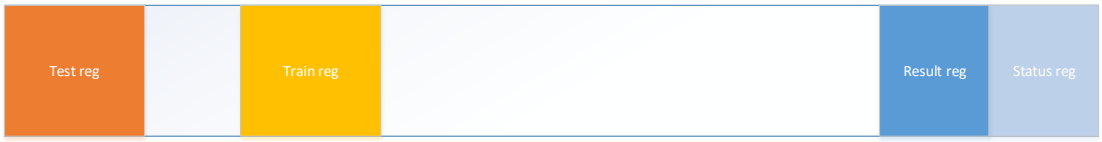


Figure 19：KNN Accelerator **地址映射**

Result reg 为 32 位只读寄存器，存储上一次运算的结果。

Status reg 为 32 位可读可写寄存器，只有最低比特位有效，表示系统的状态。为 1 表示系统忙碌，为 0 表示系统空闲。将 status 写 1 会开启一次运算。

Test reg 为 37 个 32 位只写寄存器，在 status 为 1 也就是系统忙碌时，写忽略。

Train reg 为 37 个 32 位只写寄存器，在 status 为 1 也就是系统忙碌时，写忽略。

4.4.3 接口定义

采用 ICB 总线接口协议，有关 ICB 总线接口信号，详见 4.3.3 EMAC 外设接口定义。

4.4.4 本系统在全局的位置

KNN Accelerator 模块通过 ICB 总线与私有外设模块连接，私有外设模块通过 BIU 连接到 LSU。

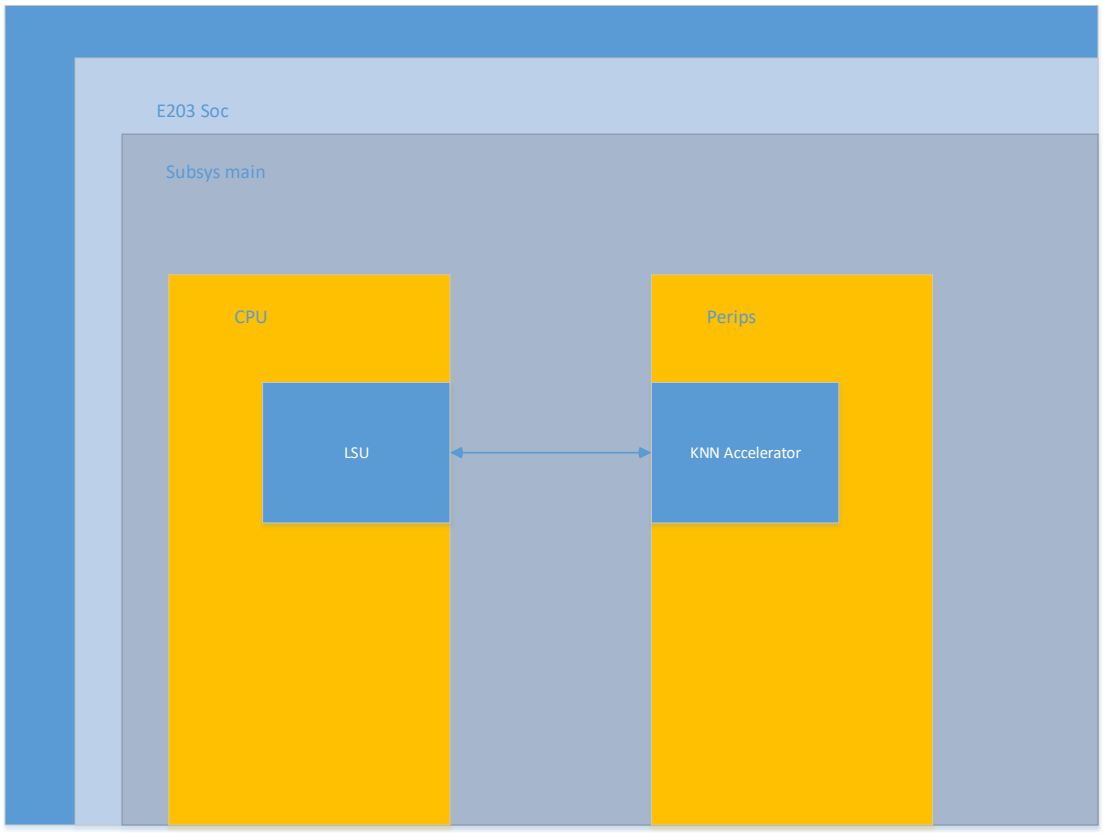


Figure 20：KNN Accelerator **连接关系**

4.4.5 算法设计与实现框图

因为图像为二值图像，所以可以通过异或计算，求取两个图片对应像素点的差值，然后通过计算异或结果中 1 的个数就可以得到两个图片间的距离了。

二值化图片的存储方式为用 37 个 32 位寄存器存储，这样，37 个寄存器的低 22 位就保留了图片的像素点信息。

整体的实现是通过状态机的方式：状态分为 IDLE，XOR，COUNT，SUM1，SUM2，RSP。IDLE 态等待 CPU 发出的命令；XOR 态计算两组寄存器的异或并存储到差值寄存器，COUNT 态计算每个差值寄存器中 1 的数目；SUM1 态以四个为一组对每个差值寄存器中 1 的数目进行累加，SUM2 在上一次累加结果的基础上再以四个一组进行累加。最终累加的结果会再 RSP 态被缓存到结果寄存器中。

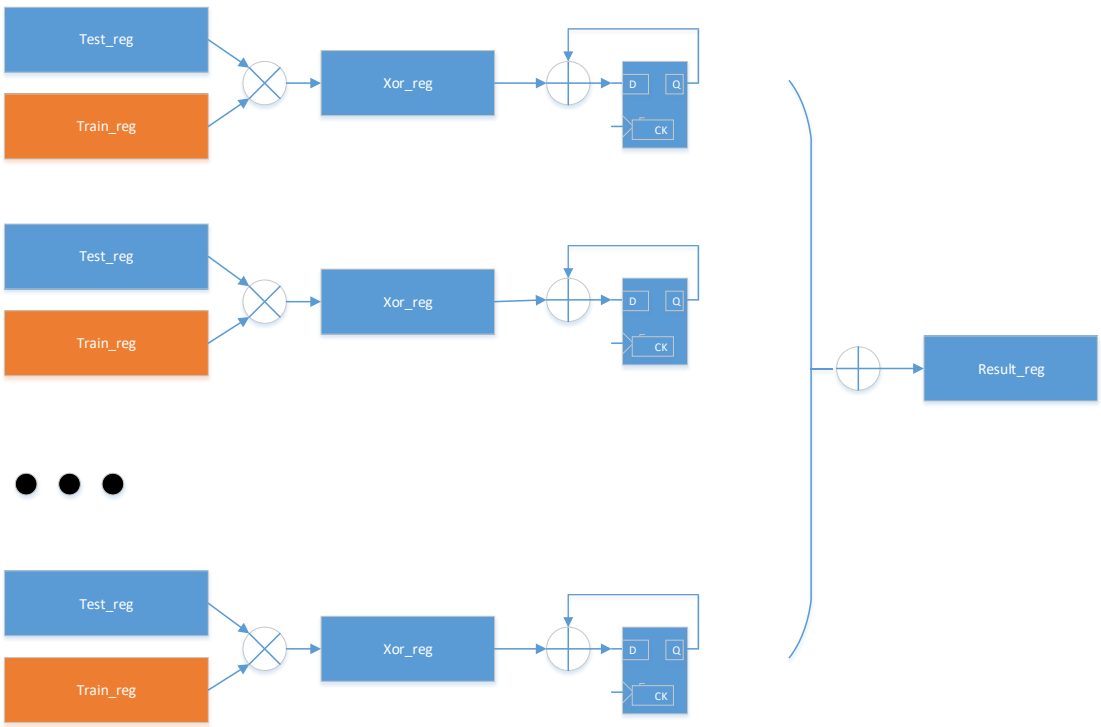


Figure 21：KNN Accelerator **实现框图**

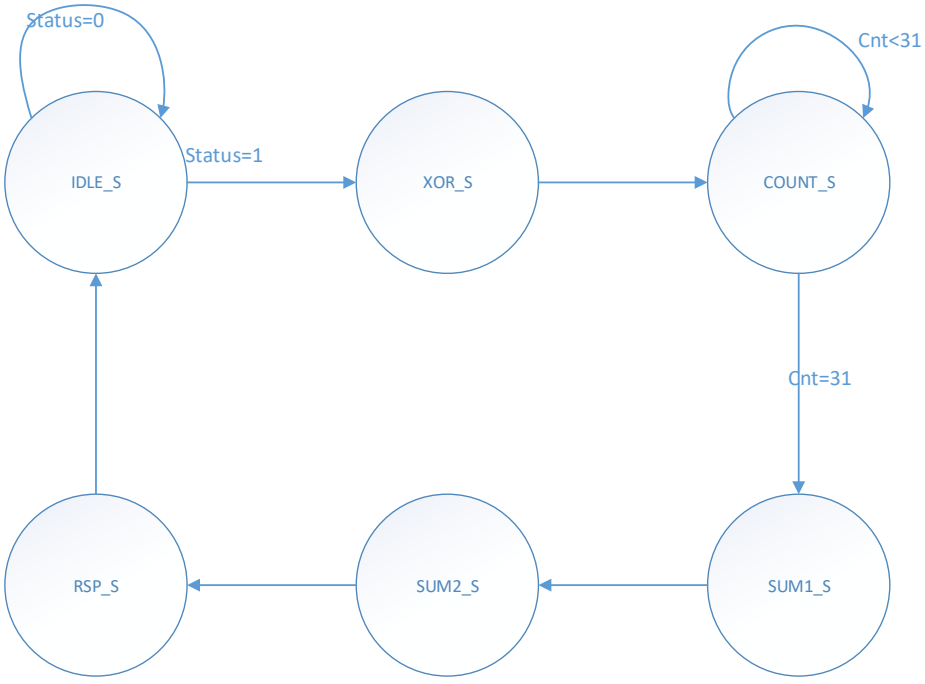


Figure 22：KNN Accelerator **状态转移图**

4.4.6 行为仿真

行为仿真的 tb 文件设计为，测试集赋值为 0 到 36 这 37 个数字。训练集赋值为对应的寄存器地址。这样测试集和训练集对应寄存器的异或值为 4，总的差值为 0x94。

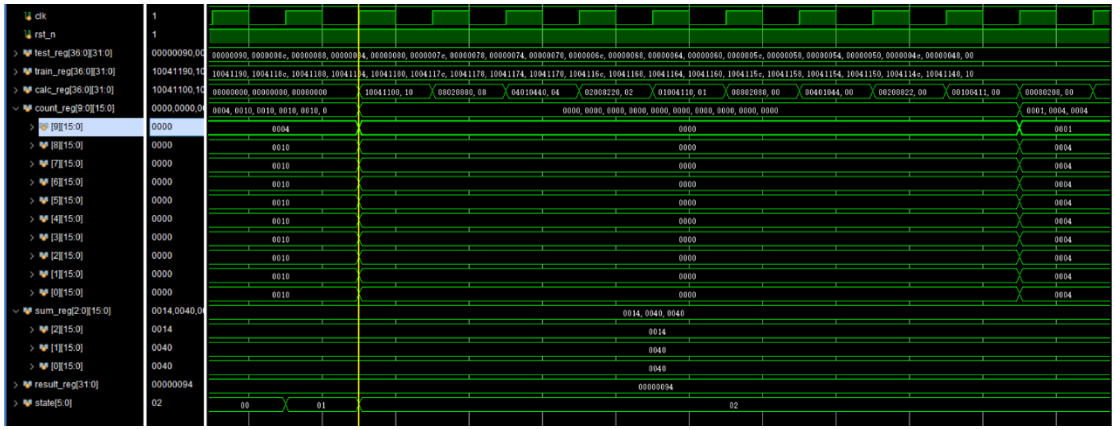


Figure 23： KNN XOR 与 COUNT 状态行为仿真

可以看出来系统在异或状态计算异或值。随后进入了数 1 状态。

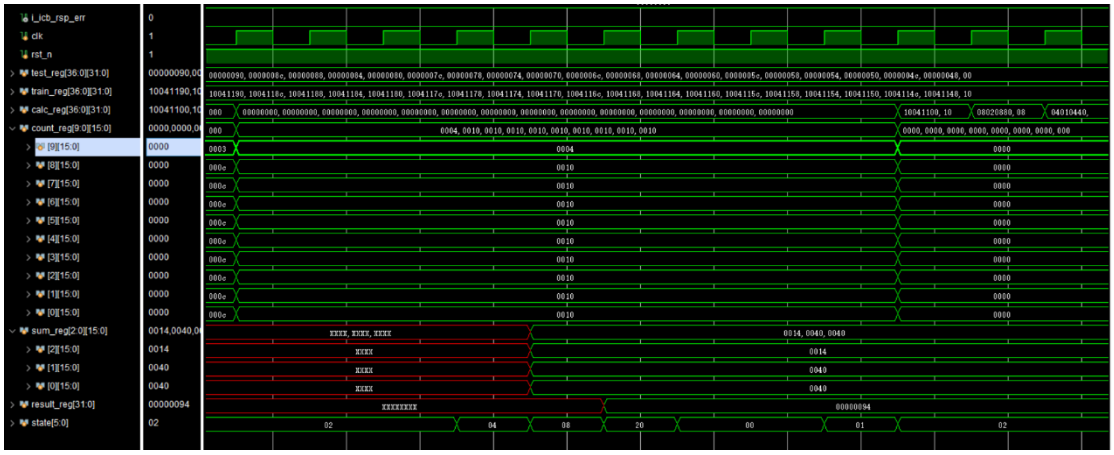


Figure 24：KNN SUM 与 RSP 状态行为仿真

随后进入加法求和阶段，最终得到的结果为 0x94 和预期结果一致。

4.4.7 FPGA 测试

我们用训练集中符号 0 的图片与训练集中的其他符号的图片进行对比，求取距离。得到的距离与在 matlab 中计算得到的距离是一致的。

4.5 VGA 显示控制电路

4.5.1 功能描述

VGA Controller 模块接收来自 CPU 的字符显示数据，并将字符像素通过 VGA 显示到屏幕上。显示分辨率为 640x480，色彩格式为 RGB444。总共接收 18x10 共计 180 个字符信息。显示字符包括 10 个阿拉伯数字和大写字母‘X’。

4.5.2 地址映射

VGA Controller 使用了 0x1001_0000 ~ 0x1001_0FFF 的物理内存空间。

其中 0x1001_0000 ~ 0x1000_1083 暂存 180 个字符信息。

0x1001_0FFC 为模块使能寄存器（只写）。当最低比特位置 1 使能 VGA 显示模块，最低比特位置 0 停止 VGA 显示模块。

4.5.3 接口定义

用户侧采用 ICB 接口，详见 4.3.3 接口定义。

VGA 侧采用 RGB444 输出格式。

Table 6：VGA Controller 模块 IO 接口定义

Name	Direction	Bits	Description
Vga_R	O	4	VGA 四位红色信息
Vga_G	O	4	VGA 四位绿色信息
Vga_B	O	4	VGA 四位蓝色信息

4.5.4 本系统在全局的位置

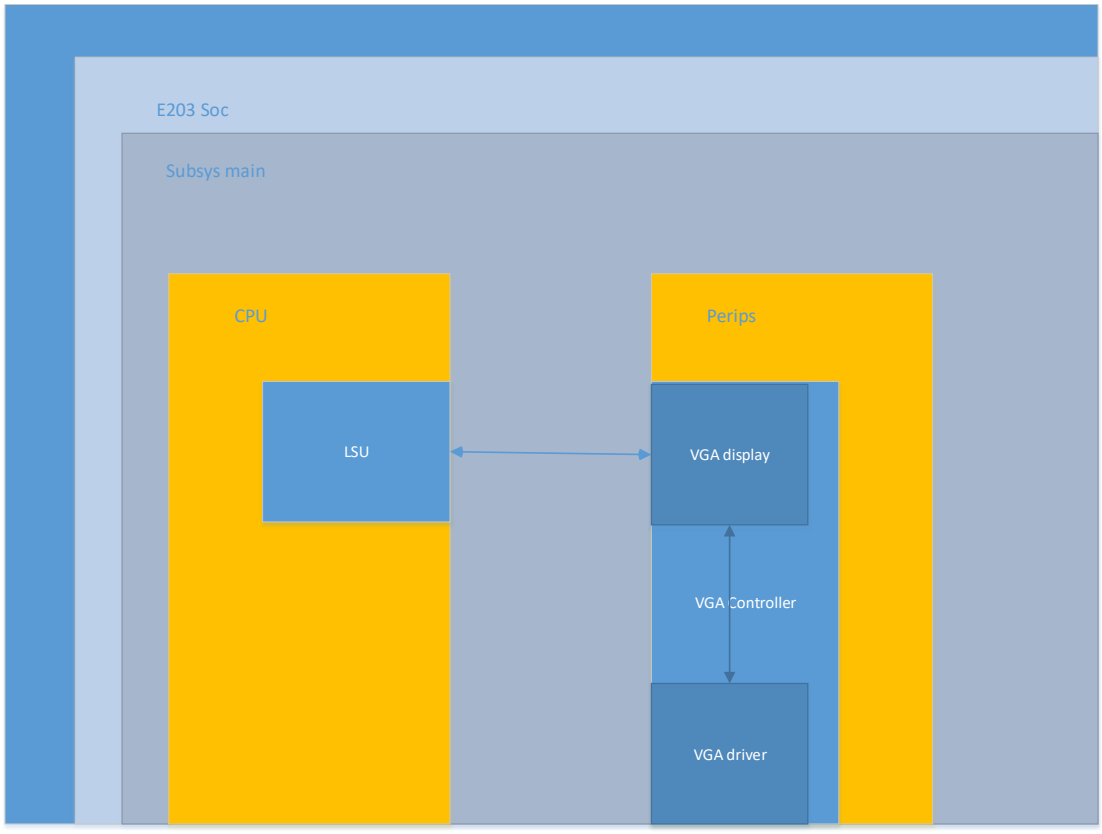


Figure 25：VGA Controller 连接关系

4.5.5 算法设计与实现框图

模块包括 VGA display 模块和 VGA driver 模块。其中 VGA display 模块接收从 CPU 发来的字符信息，并将每个像素点的颜色信息发送给 VGA driver 模块。

VGA driver 模块控制 VGA 信号的时序。

VGA driver 模块中含有两个计数器，cnt_h 和 cnt_v 分别为行同步计数器和场同步计数器。分别在行场计数器达到指定值的时候输出行场同步信号，同时行同步计数器也指示 y 轴坐标信息，场同步计数器指示 x 轴坐标。pixel_xpos 和 pixel_ypos 会回馈到 VGA display 模块。

VGA display 模块，中包含两个寄存器组，char 寄存器组存储了 11 种字符的字模信息。Data_file 寄存器组存储了 180 个字符对应的字符信息。VGA display 模块接收来自 VGA driver 模块提供的 pixel_xpos 和 pixel_ypos。我们会用 pixel_xpos 和 pixel_ypos 的高比特位来检索对应的字符信息，低比特位检索字符对应的每个像素信息，再通过 pixel_data 信号返回给 pixel_driver 模块。

4.5.6 FPGA 测试

FPGA 测试为显示模拟身份证号字符，在显示器左上角，依次显示 10 个身份证号数字。身份证号数字是从 0 到 X 的循环。下图可见 VGA 测试结果与预期一致。

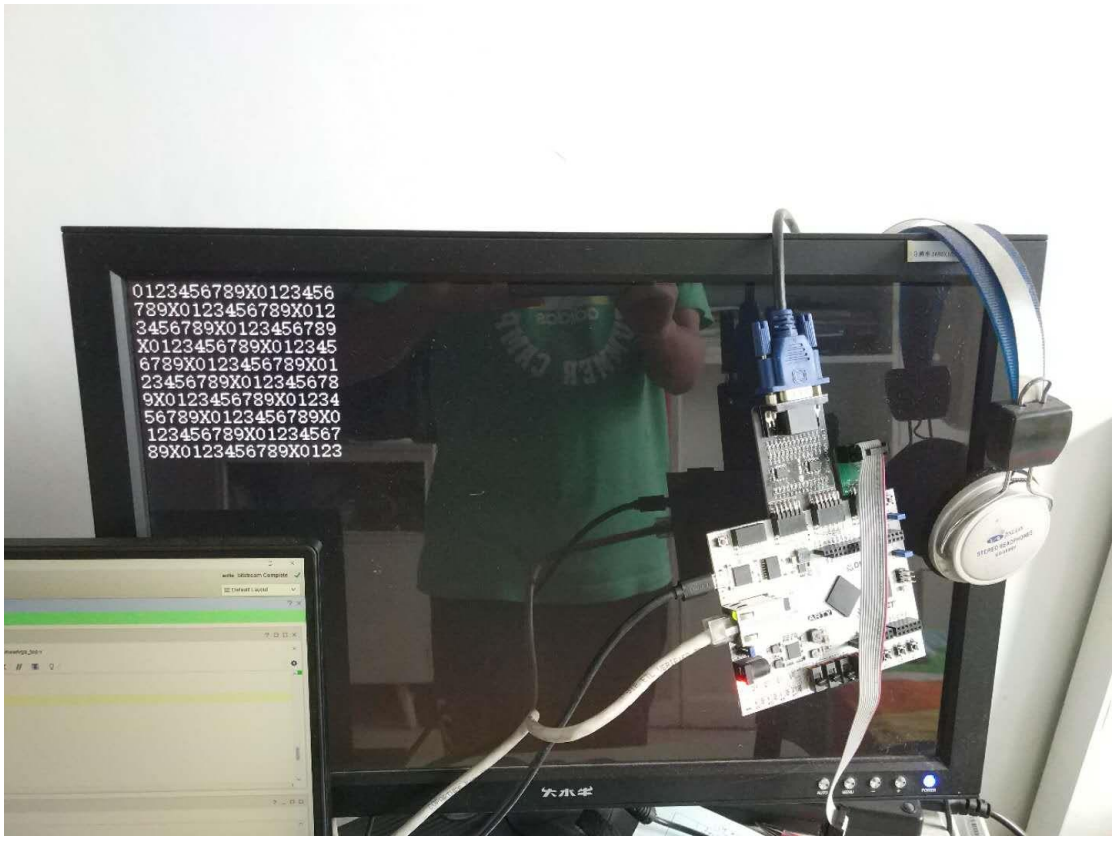


Figure 26：VGA 模块 FPGA 原型测试

5 软件分系统设计方案

5.1 UDP/IP 协议栈

传输层最重要的协议就是 TCP 和 UDP。TCP 协议复杂，是面向连接的传输协议且传输可靠；而 UDP 协议简单，是面向无连接的传输协议，传输速度快但传输不可靠。可以将 UDP 协议看作 IP 协议暴露在传输层的一个接口。

IP 协议是不可靠且无连接的网络层传输协议。所有的 TCP、UDP 等数据都是以 IP 数据报的形式进行传输的。IP 协议通过 IP 地址进行目的电脑的识别与连接。待传输的数据经过 UDP 头部信息的封装后，还需进一步经过 IP 头部信息的封装，进而通过 IP 协议传输到目的地址。

Udp/ip 协议栈的建立主要是数据帧的封装和解封装的过程。我们提供了三个 api 函数。

```
void eth_ini();
```

初始化以太网物理层芯片，设置以太网的 MAC 地址等。

int receive_udp(uint8_t * buf);

这是 UDP 的接收函数，参数是指向接收缓存区的指针。主要过程是将数据 load 以及 IP head 和 udp head 从以太网帧中解析出来，判断 IP 地址和协议类型是否满足要求。满足要求则将数据放入缓存区，否则丢弃。

void send_udp(uint16_t data_len,uint8_t * buf);

这是 UDP 的发送函数，第一个参数是要发送的报文的长度，第二个参数是指向发送缓存区的指针。整个过程就是为数据加 udp head，ip head 以及以太网帧头，并计算 check sum。

5.2 TFTP 客户端

TFTP 是一个传输文件的简单协议，它基于 UDP 协议而实现，此协议设计的时候是进行小文件传输的。它只能从文件服务器上获得或写入文件，不能列出目录，不进行认证，它传输 8 位数据。传输中有三种模式：netascii，这是 8 位的 ASCII 码形式，另一种是 octet，这是 8 位源数据类型；最后一种 mail 已经不再支持，它将返回的数据直接返回给用户而不是保存为文件。

任何传输起自一个读取或写入文件的请求，这个请求也是连接请求。如果服务器批准此请求，则服务器打开连接，数据以定长 512 字节传输。每个数据包包括一块数据，服务器发出下一个数据包以前必须得到客户对上一个数据包的确认。如果一个数据包的大小小于 512 字节，则表示传输结束。如果数据包在传输过程中丢失，发出方会在超时后重新传输最后一个未被确认的数据包。通信的双方都是数据的发出者与接收者，一方传输数据接收应答，另一方发出应答接收数据。

提供以下 api 函数：

Void send_tftp_req(char* filename, unsigned char is_download)

发送文件请求，第一个参数是要上传或者下载的文件名，第二个参数表示是上传（1）还是下载（2）。

void send_tftp_ack()

发送 ack 应答报文。

void send_tftp_data(unsigned char data_len, uint8_t* buf)

发送 tftp 数据报文，第一个参数是发送的数据的长度，第二个参数是文件缓冲区指针。

```
int receive_tftp_data(uint8_t* buf)
```

接收数据报文，返回值是接收的数据长度，参数是指向接收缓冲区的指针。

5.3 图像识别算法

5.3.1 jpg 解码

1）读取 Huffman 表

2）构建 Huffman 树

3）DC 系数的 Huffman 解码

4）AC 系数的 Huffman 解码

5）反量化

在译码得到了 8*8 的系数矩阵之后，我们需要进行反量化工作。该步骤，就是将前一个步骤得到的 8*8 系数矩阵分别乘以 8*8 的量化矩阵即可。

6）反 Zig-zag 扫描

JPEG 编码过程中，为了编码方便，采用了 Zig-zag 扫描，因此，这里需要进行反 Zig-zag 扫描，重新排列 8*8 的反量化系数矩阵。反 Zig-zag 扫描的输入时 8*8 矩阵，输出依然是 8*8 矩阵，只不过，数据的排列方式有所不同而已。

7）DCT 逆变换

DCT 逆变换，就是要将数据从频域变换回时域

8）颜色模式转换

BMP 图片是以 RGB 颜色空间进行保存的，因此，将 JPEG 解码为 BMP 必须进行颜色模式的转换。另外，由于 DCT 要求的定义域对称，所以，在编码的时候将 RGB 的数值范围从[0,255]统一减去 128，将数值范围转换到[-128,127]的范围内。因此，解码的时候，必须为每个颜色分量加上 128。另外需要注意的是，通过解码变换之后得到的 RGB 的值有可能超过 255 或者小于 0；如果小于 0，就截断为 0，如果大于 255，就截取为 255；

5.3.2 字符定位与分割

因为提供的图片中的字符均为打印体且大小一致，所以对于字符的定位与分割我们只需要首先找到字符区域的上边界和左边界之后按照固定大小依次向右分割图片就可以了。

5.3.3 KNN 算法

KNN 的全称是 K Nearest Neighbors，意思是 K 个最近的邻居，KNN 的原理就是当预测一个新的值 x 的时候，根据它距离最近的 K 个点是什么类别来判断 x 属于哪个类别。

我们将从测试集中提取出来的标准大小的字符图片与训练集的各个图片进行比较。比较过程在 KNN Accelerator 中进行，对各个字符比较的结果进行排序，选择距离最小的字符训练集图片作为识别结果。

6 系统性能与测试

6.1 GPIO 测试

GPIO 测试的任务为将 GPIO 的输入端口和输出端口连接在一起，输入跟随输出变化。我们将红色 led 的输出赋值为 button0 的输入，绿色 led 对应 button1，蓝色 led 对应 button2。测试程序见：测试/gpio/gpio_test.c。

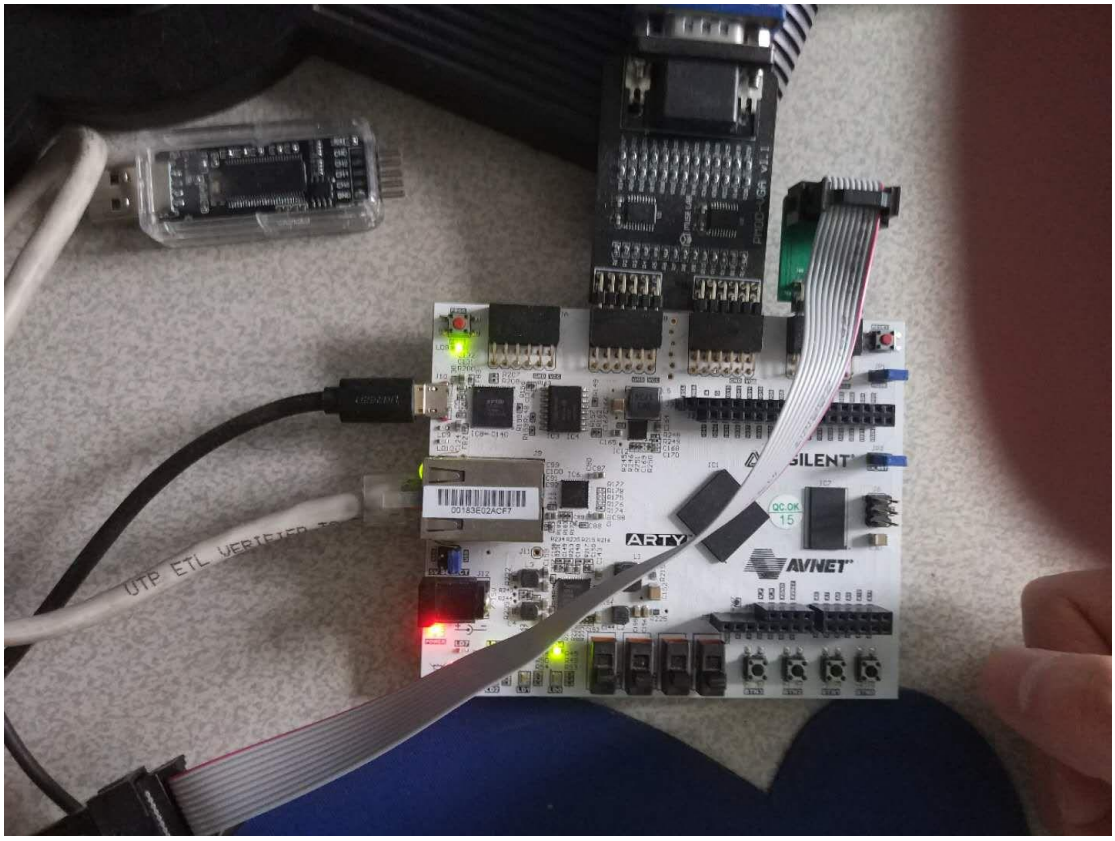


Figure 27：GPIO 不按按键

没有按下按键，三色 led 不亮

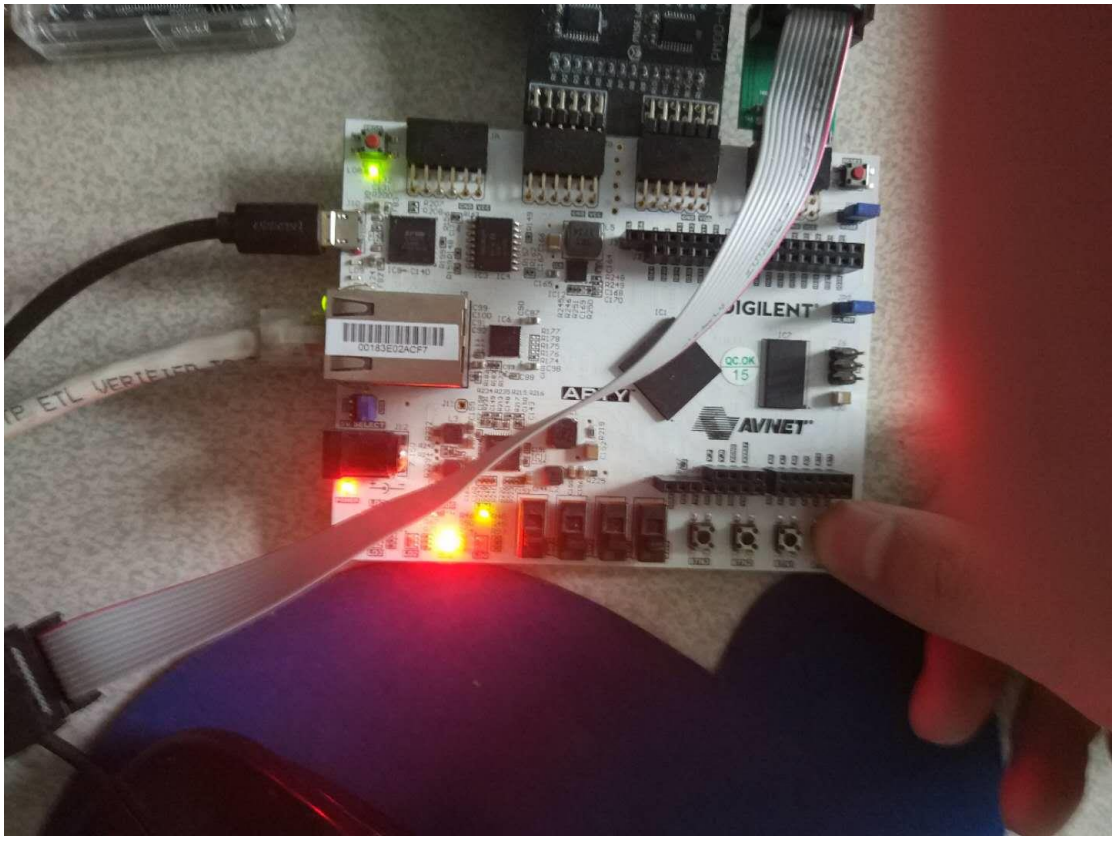


Figure 28：GPIO button0 对应红色 led

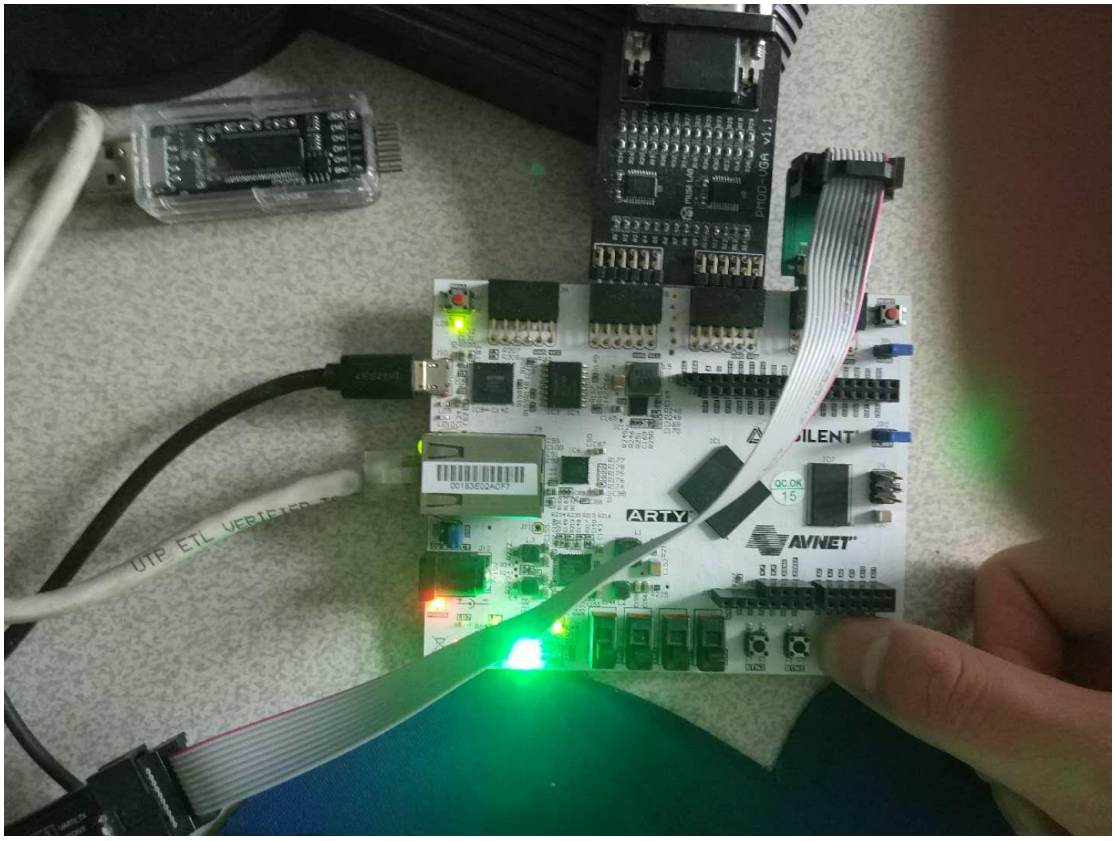


Figure 29：GPIO button1 对应绿色 led

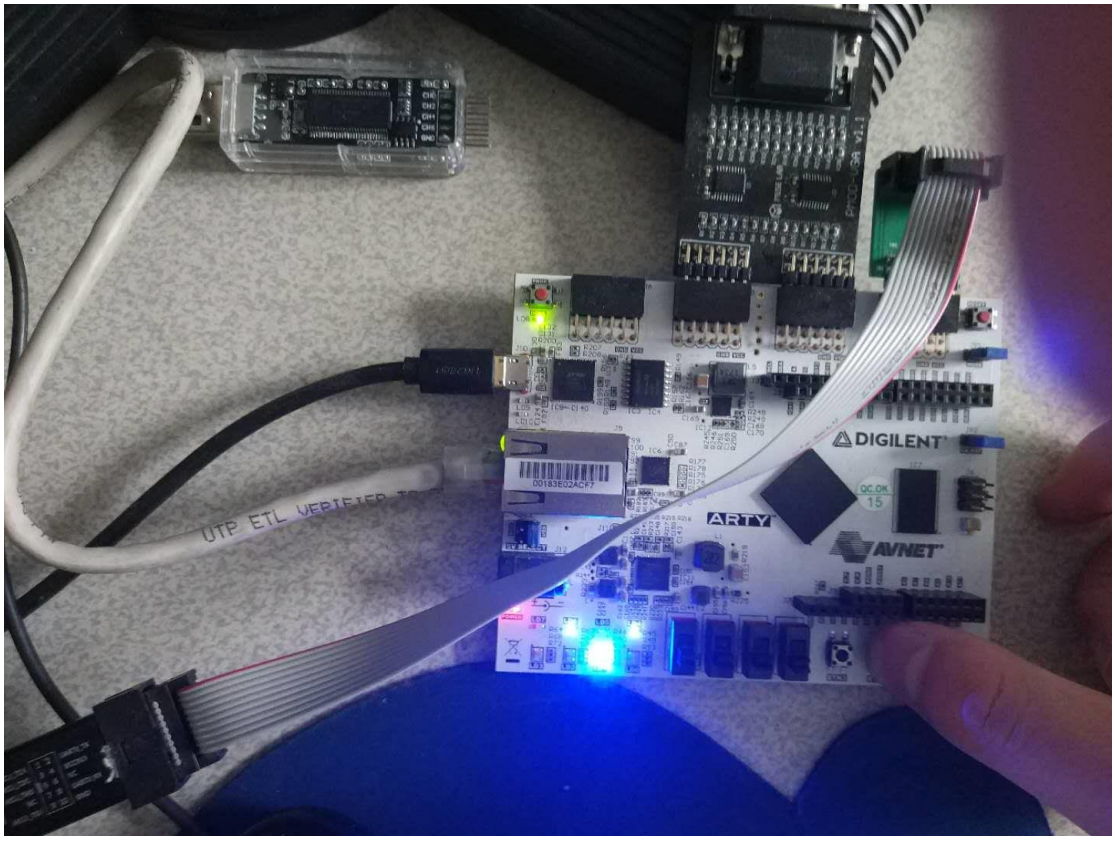


Figure 30：GPIO button2 **对应蓝色** led'

6.2 RISC-V 字符显示测试

测试任务将 RISC-V SoC 的 8 路数字 I/O 连接到硬件逻辑分析仪，在硬件逻辑分析仪上显示出“RISC-V”字样（任意字体）。我们将 SoC 的 GPIO 引脚连接到 JB PMOD 接口上，并在 PMOD 接口上连接逻辑分析仪，逻辑分析仪再通过 USB 连接到电脑上。程序通过控制 GPIO 的高低电平顺序，实现了”RISC-V”字样。测试程序详见：测试/risc-v/risc_v.c

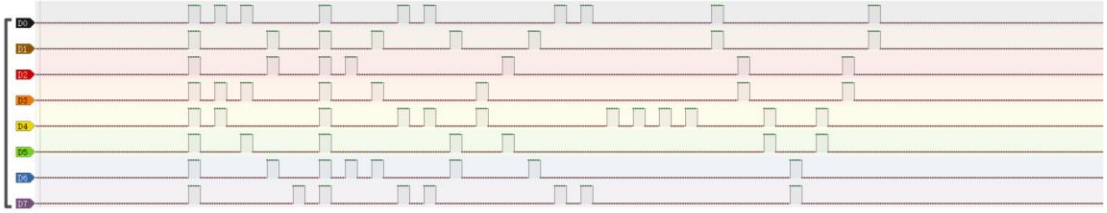


Figure 31：RISC-V **字样显示测试**

我们可以看到图片中显示出了“RISC-V”字样。

6.3 以太网通信测试

6.3.1 udp 报文环回实验

测试任务为网口调试助手每间隔 1s 向开发板发送一个 udp 报文，报文内容为“hello risc-v”，开发平台接收到 udp 报文后，会原封不动的把数据回传给网口调试助手。双方都是用 1234 作为源和目的端口号。

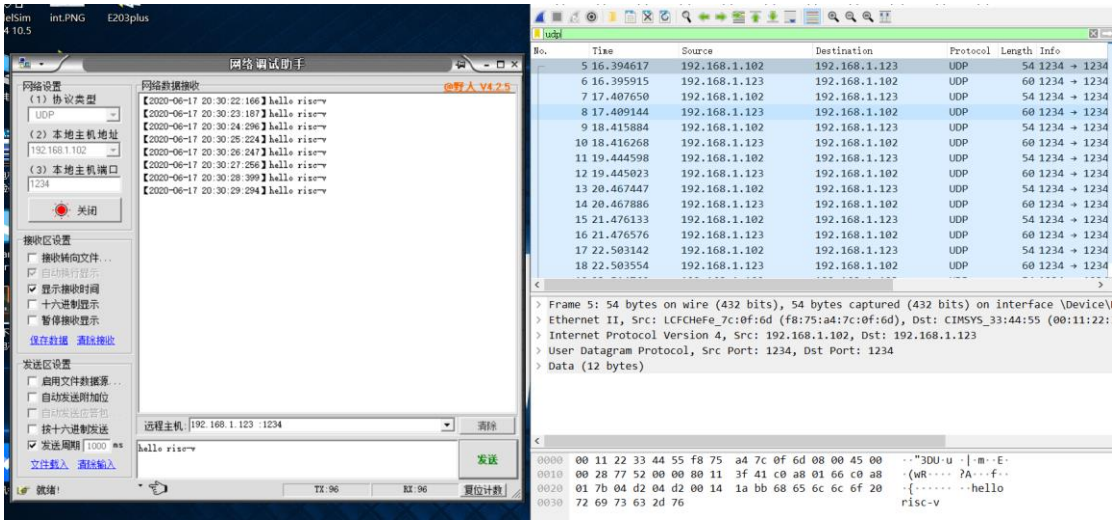


Figure 32：以太网 UDP 报文回传测试

我们可以看到，左端网口调试助手显示了回传的 hello risc-v 报文内容。右端 wireshark 中可以看到交替的发送和接收 udp 报文。

6.3.2 tftp 图片传输实验

测试任务为开发板从 tftp 服务器下载 10 张图片。测试程序请见：测试/tftp/main.c。

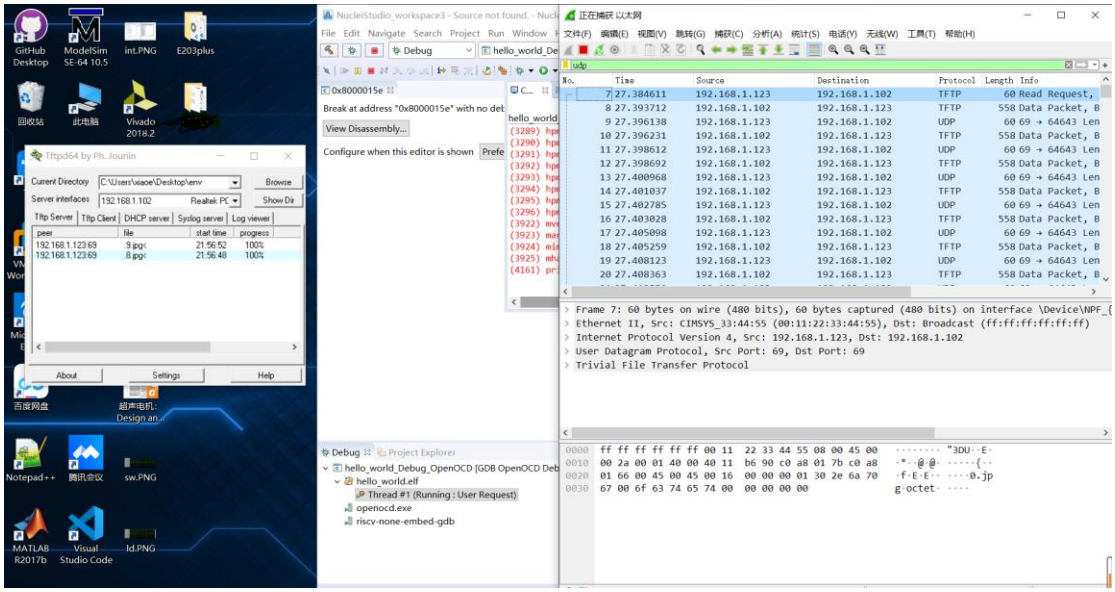
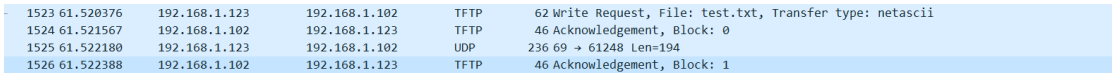


Figure 33：tftp **文件传输测试（下载）**

可以看到左边的 tftp 服务器正在完成图片传输任务，右边 wireshark 可以看到传送第一个文件请求报文后，服务器向客户端发送数据报文，客户端返回 ack 报文。



1523	61.520376	192.168.1.123	192.168.1.102	TFTP	62 Write Request, File: test.txt, Transfer type: netascii
1524	61.521567	192.168.1.102	192.168.1.123	TFTP	46 Acknowledgement, Block: 0
1525	61.522180	192.168.1.123	192.168.1.102	UDP	236 69 → 61248 Len=194
1526	61.522388	192.168.1.102	192.168.1.123	TFTP	46 Acknowledgement, Block: 1

Figure 34：tftp **文件传输测试（上传）**

可以看到客户端向服务器发送上传文件文件请求，服务器向客户端发送 ack 报文，客户端向服务器发送数据报文。

6.4 身份证号识别任务测试

图像识别任务测试主要内容为开发平台，从 tftp 服务器下载 10 张图片，对这 10 张图片的身份证号进行识别。并将识别结果写在一个 test.txt 文件中，并上传到服务器文件夹。测试程序请见：测试/final_test/main.c。最终我们的开发平台用时 34s，结果全部正确。

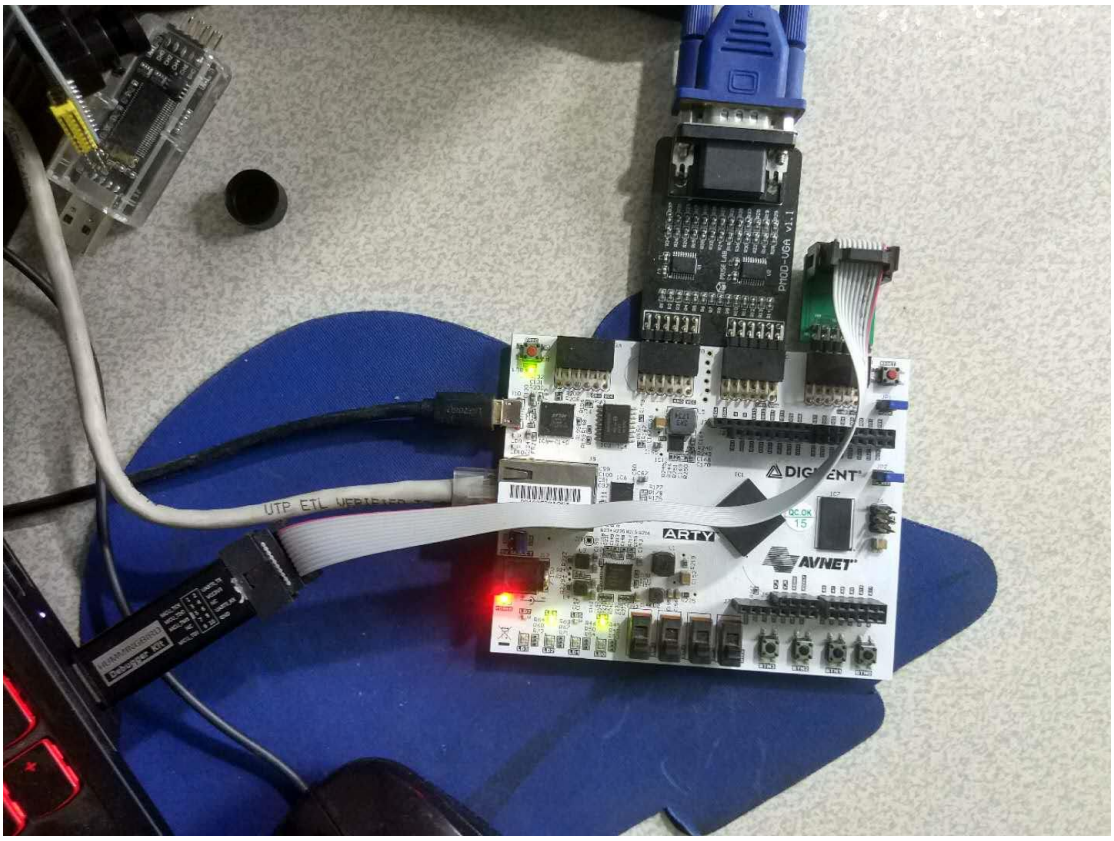


Figure 35：**身份证号识别任务硬件连接**

整体的硬件连接，分别来连接了 jtag/uart 接口下载比特流文件和进行 uart 通信，JB 和 JC 两个 pmod 接口连接 VGA 显示器，JD pmod 接口连接 jtag 调试器调试软件程序，RJ45 接口连接网线。

有关图像传输的结果请见 6.3.2 部分。

```
Connection received from 192.168.1.123 on port 69 [17/06 21:56:21.550]
Read request for file <0.jpg>. Mode octet [17/06 21:56:21.550]
Using local port 64643 [17/06 21:56:21.550]
<0.jpg>: sent 75 blks, 38159 bytes in 0 s. 0 blk resent [17/06 21:56:21.755]
Connection received from 192.168.1.123 on port 69 [17/06 21:56:25.043]
Read request for file <1.jpg>. Mode octet [17/06 21:56:25.043]
Using local port 64644 [17/06 21:56:25.043]
<1.jpg>: sent 75 blks, 38122 bytes in 0 s. 0 blk resent [17/06 21:56:25.168]
Connection received from 192.168.1.123 on port 69 [17/06 21:56:28.455]
Read request for file <2.jpg>. Mode octet [17/06 21:56:28.455]
Using local port 64645 [17/06 21:56:28.455]
<2.jpg>: sent 76 blks, 38438 bytes in 0 s. 0 blk resent [17/06 21:56:28.578]
Connection received from 192.168.1.123 on port 69 [17/06 21:56:31.862]
Read request for file <3.jpg>. Mode octet [17/06 21:56:31.862]
Using local port 64646 [17/06 21:56:31.878]
<3.jpg>: sent 76 blks, 38466 bytes in 0 s. 0 blk resent [17/06 21:56:31.988]
Connection received from 192.168.1.123 on port 69 [17/06 21:56:35.289]
Read request for file <4.jpg>. Mode octet [17/06 21:56:35.289]
Using local port 60862 [17/06 21:56:35.289]
<4.jpg>: sent 74 blks, 37604 bytes in 0 s. 0 blk resent [17/06 21:56:35.388]
Connection received from 192.168.1.123 on port 69 [17/06 21:56:38.664]
Read request for file <5.jpg>. Mode octet [17/06 21:56:38.664]
Using local port 60863 [17/06 21:56:38.664]
<5.jpg>: sent 76 blks, 38699 bytes in 0 s. 0 blk resent [17/06 21:56:38.801]
Connection received from 192.168.1.123 on port 69 [17/06 21:56:42.091]
Read request for file <6.jpg>. Mode octet [17/06 21:56:42.091]
Using local port 60864 [17/06 21:56:42.091]
<6.jpg>: sent 75 blks, 38147 bytes in 0 s. 0 blk resent [17/06 21:56:42.201]
Connection received from 192.168.1.123 on port 69 [17/06 21:56:45.481]
Read request for file <7.jpg>. Mode octet [17/06 21:56:45.481]
Using local port 60865 [17/06 21:56:45.481]
<7.jpg>: sent 74 blks, 37679 bytes in 0 s. 0 blk resent [17/06 21:56:45.591]
Connection received from 192.168.1.123 on port 69 [17/06 21:56:48.858]
Read request for file <8.jpg>. Mode octet [17/06 21:56:48.858]
Using local port 62707 [17/06 21:56:48.879]
<8.jpg>: sent 74 blks, 37702 bytes in 0 s. 0 blk resent [17/06 21:56:48.988]
Connection received from 192.168.1.123 on port 69 [17/06 21:56:52.265]
Read request for file <9.jpg>. Mode octet [17/06 21:56:52.265]
Using local port 58722 [17/06 21:56:52.265]
<9.jpg>: sent 75 blks, 38181 bytes in 0 s. 0 blk resent [17/06 21:56:52.389]
Connection received from 192.168.1.123 on port 69 [17/06 21:56:55.667]
Write request for file <test.txt>. Mode netascii [17/06 21:56:55.667]
Using local port 61248 [17/06 21:56:55.667]
<test.txt>: rcvd 1 blk, 190 bytes in 0 s. 0 blk resent [17/06 21:56:55.667]
```

Figure 36：**身份证号识别任务** tftp **服务器** log

我们可以看到第一帧 req 的时间为 21:56:21，最终发送 test.txt 完成的时间为 21:56:55，总用时为 34s。

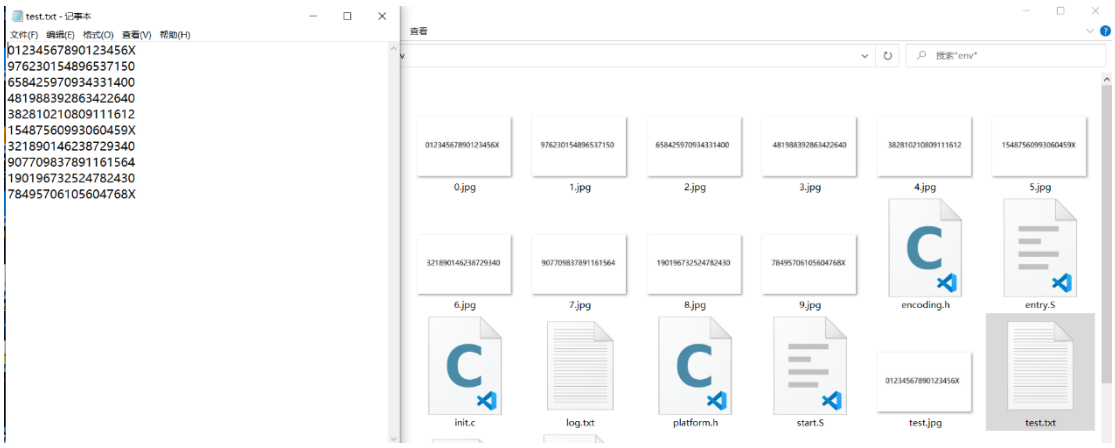


Figure 37：**身份证号识别任务** txt **结果**

我们可以看到识别的结果和图片中的内容是完全一致的。

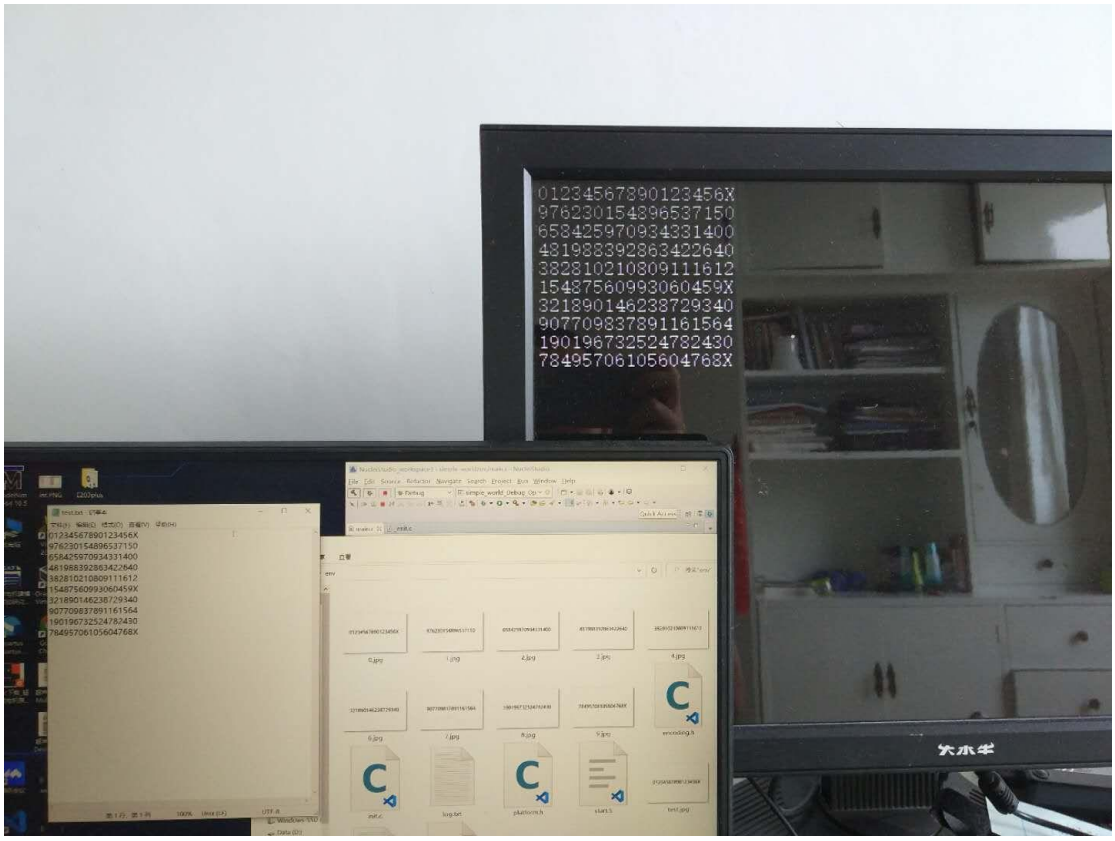


Figure 38：**身份证号识别任务显示器显示结果**

外接显示器中也显示了辨识结果。

7 进一步工作

进一步工作主要从三个方面开展：

1）将 tftp 改为 ftp，并建立 tcp/ip 协议栈，准备采用 uIP 开源协议。uIP TCP/IP 协议栈的目标是：即便是 8 位微控制器也可以使用 TCP/IP 协议栈进行网络通信。该协议栈代码量小占用资源少，适合嵌入式且 ram 资源紧张的情况下使用。

2）加速 jpg 解码，和图像识别任务。加速的关键是减少对外部 DDR 的访问。目前减少 DDR 访问主要突破点在 jpg 解码阶段。原始的解码程序将图片解码为 RGB888 格式，并全部存入内存。探索方法直接将图片解码为灰度图像。这样可以从存储角度减少 2/3 的访问次数。

3）探索人脸识别任务。

参考文献

[1] https://github.com/richgel999/picojpeg

[2] DIGILENT. Arty™ FPGA Board Reference Manual. 2017.

[3] 胡振波. 叫你设计 CPU——RISC-V 处理器. 人民邮电出版社. 2018.

[4] XILINX. Zynq-7000 AP SoC and 7 Series Devices Memory Interface Solutions v4.1. 2018.

[5] XILINX. AXI Ethernet Lite MAC v3.0. 2019.