



山东大学(威海)
SHANDONG UNIVERSITY, WEIHAI

毕业论文（设计）

设计(论文)题目 基于 FPGA 的数字识别系统设计

姓 名: 郭世平

学 号: 201700800144

学 院: 机电与信息工程学院

专 业: 电子信息科学与技术

年 级: 2017

指导教师: 李素梅

2021 年 5 月 12 日

目 录

摘 要	1
关 键 词	1
ABSTRACT	1
1. 绪论	2
1.1 选题背景及研究意义	2
1.2 国内外研究现状	2
1.3 论文主要研究内容	2
1.4 论文组织结构	2
2. Cortex M3 定制	3
2.1 ARM Cortex M3 简介	3
2.2 CMSDK 使用	3
2.3 CMSDK 文件结构	4
2.4 CMSIS 介绍与使用	4
2.5 Cortex M3 定制	5
2.6 Cortex M3 核仿真测试	5
3. 主要外设模块设计	6
3.1 总体架构设计	6
3.2 AHB 总线设计与生成	6
3.3 FIFO 控制器设计与实现	8
3.4 GPIO 控制器设计与实现	8
3.5 UART 控制器设计	11
3.6 LCD 控制器设计	13
3.7 Camera 控制器设计	15
3.8 SDRAM/SRAM 控制器	16
3.9 顶层模块设计	18
4. 基于 FPGA 的图像处理加速方式	18
4.1 流水线、并行加速	18
4.2 指令集加速	19
4.3 压缩数据量	19
4.4 优化数据传输	19
4.5 算法优化	20
4.6 轻量级卷积加速器加速器设计	20
5.2 手写体识别系统实现	21
5. SoC 测试	21
6. 总结与展望	24
参考文献	25
谢 辞	26

摘 要

近些年，图像处理系统已经应用在社会的方方面面，并且图像处理系统正向着体积小、速度快的方向不断发展。本文在 Intel 公司的 FPGA 芯片 Cyclone IV 上设计实现了一个以 Cortex M3 为内核的轻量级图像处理片上系统，包括定制的 Cortex M3 软核、图像处理硬件加速器、摄像头和液晶屏驱动电路等模块。通过并行计算、数据复用、数据压缩等方式实现手写体数字识别算法的硬件加速。硬件化实现了数字手写体算法完成整个系统。采用 Verilog HDL 设计描述各电路模块，并采用 QUARTUS II 和 ModelSim 进行综合、仿真。通过硬件测试实现了手写数字识别功能，经过实际测试识别成功率可达 90% 以上。

关 键 词

手写体数字识别；FPGA；Cortex M3 软核；CNN

ABSTRACT

Image processing system has been applied in all aspects of the society in recent years, and the image processing system is developing towards the direction of small volume and fast speed. In this work, a light image processing system on chip based on Cortex M3 is designed and implemented on Cyclone IV FPGA chip, including Cortex M3 soft core、image processing hardware accelerator、camera、LCD driver circuit and other modules. Parallel computing, data reuse and data compression are used to design the hardware acceleration of image processing. Verilog HDL was used to describe each circuit module. Quartus II and ModelSim were used for compilation and simulation. The function of handwritten digit recognition is realized through hardware test and the recognition success rate can reach 90% after the actual test.

KEY WORDS

Handwriting number recognition；FPGA；Cortex M3 soft core；CNN

1. 绪论

1.1 选题背景及研究意义

随着社会的发展,科技的进步,在图像识别、图像分类等领域对数字图像处理的要求越来越苛刻。GPU (Graphics processing unit) 的出现也催生出一些更加复杂的图像处理算法如 AlexNet^[1]、VGGNet^[2]、GoogleNet^[3]等。这些复杂的算法具有非常高的准确性但也有较高的延时性,因此又出现一些轻量级的卷积网络如: MobileNet^{[5][6][7]}等。高精度和快速一直是图像处理追求的目标,但可惜的是这两个目标是相互矛盾的。利用现有的技术探索更优地均衡方案是现在的科研人员研究的方向。在一些嵌入式设备中需要非常低的延时性,此时,相比于 GPU、CPU、IC 等硬件, FPGA 的优势就会更加明显。FPGA 在图像处理领域具有更大的潜力,尤其是在处理速度方面, FPGA 的并行的、流水线结构可以显著的提高图像处理速度。各种基于 FPGA 的图像处理加速器也可以显著地提高图像处理速度。图像处理加速器也是当前图像处理的热门话题。

1.2 国内外研究现状

剑桥大学的 Yu-Hsin Chen 等人研究的 DCNN 可配置低能耗加速器^[4]通过优化数据传输、数据复用、并行处理等方式对 Alex Net 进行加速,最终吞吐量达到 34.7fps。此外通过跳过卷积运算中的 0 数据将整体运算速度提高 180%。印度的 Harsh Srivastava 等人通过流水线结构对深度可分离卷积运算加速^[8]该方法只加速了深度可分离卷积运算的一层,因此便于移植。哈尔滨工业大学的宋鹏飞等人通过设计新的框架来加速快速数论变换算法 (NTT) ^[9]优化数据存储得到一个更低面积和延迟的 FPGA 图像处理加速器。郑州大学的齐延荣等人设计了一种具有高效流水线内核的 OpenCL 网络加速器。此外还有通过优化运算单元 (PE)、优化指令集、压缩数据量的方式加速卷积运算。以上学者提出的方案的都只是从纯硬件角度实现图像加速或者图像识别等功能。本文设计了一款低功耗、灵活度高、可移植的数字手写体识别硬件系统。

1.3 论文主要研究内容

本文根据开源的资源和现有的理论知识实现了一个基于 FPGA 的数字手写体识别系统,并在 Quartus II 18.0 环境下,采用 verilog 描述语言实现了电路模型,在 ModelSim10.3 中进行仿真,最后在搭载了 Cyclone IV 的 DE-2 开发板上进行硬件测试。系统定制了 ARM 公司开源的 Cortex M3 微控制器,设计了 LCD、SDRAM、Camera、UART 等硬件控制器、软件驱动为该系统提供必要的外设资源。此外将手写体数字识别算法在该系统上硬件化实现。采用并行计算、流水线作业、数据复用,数据传输优化等方式设计硬件图像处理加速器使得智能图像处理算法能在该轻量级系统上运行。最后通过 Keil 控制调试整个系统,并对系统的性能进行测试评估。

1.4 论文组织结构

文章第一部分是绪论,介绍课题的研究背景和意义。第二部分是 Cortex M3 CPU 核的定制,使用 CMSDK (Cortex M System Design Kit)、CMSIS(Microcontroller Software Interface Standard)实现 Cortex M3 核的定制。第三部分是搭建总线和 Cortex M3 核下搭载图像处理所需的外设,设计相关的硬件控制器,连接对应的模块。第四部分介绍图像处理加速器的设计方案。第五部分是实现软硬件适配,完成系统设计并进行验证。第六部分是本系统可改进的地方与未来研究方向。

2. Cortex M3 定制

2.1 ARM Cortex M3 简介

ARM Cortex M 系列处理器目标市场是微控制器（Microcontroller Processors），这类处理器通常设计为具有更低的硅面积和更高的高能量效率。通常，这类处理器具有更短的流水线，最大频率较低(尽管其中一些处理器的运行频率超过 200MHz)。同时，新的 Cortex-M 处理器系列被设计得非常易于使用。因此，它们在微控制器和深度嵌入式系统市场非常受欢迎。

Cortex M 系列价格低，功耗低，适用于嵌入式开发，虽然在性能上逊色于 R 系列和 A 系列，但已经足以用来设计基础的 SoC。Cortex M3 是一种小型但强大的嵌入式处理器，用于低功耗微控制器，具有丰富的指令集，使其能够更快地处理复杂的任务。它有一个硬件除法器 and 乘累加 (MAC) 指令。此外，它还具有全面的调试和跟踪功能，开发人员能够快速开发对应的应用程序。Cortex M3 采用的 AHB（Advanced High performance Bus）总线协议，此外还支持 APB（Advanced Peripheral Bus）等总线协议。

Cortex M3 中包含一个 NVIC（Nested Vectored Interrupt Controller）嵌套向量中断控制器，该部件支持 256 个中断，其中 16 个默认中断，240 个可配置中断。通过分配优先级可以实现抢占机制此外还支持迟到，这些功能为微处理器提供了出色的中断处理机制。

当发生异常时，中断控制器需要定位到中断的位置，此时就需要查找中断向量表。中断向量表中存储的中断的起始地址，中断产生后首先需要保护现场，把 xPSR、PC、LR、R12、R0-R3 压入到堆栈中，响应异常时正在使用的 PSP 或 MSP 也将会被压入栈中，然后通过中断向量表定位到中断的位置执行相应的程序执行中断程序，中断结束后执行出栈操作返回到中断之前的位置。

2.2 CMSDK 使用

表 2.4 ARM 开源的 Cortex M3 软核类型

类别	特点
DesignStart Eval 版	网表级 verilog 代码，免费申请，可读性差。
DesignStart FPGA 版	内部定制，免费，针对特定的开发板进行 FPGA 优化。
DesignStart Pro 版	RTL 级 Verilog 代码，需要 license，不便于学习。
DesignStart Physical 版	定制好的硬件电路。
DesignStart University 版	类似于 Pro 版本，但仅适用于大学教育使用。

Eval 版本容易获取且免费，虽然 Eval 版的 Cortex M3 软核核心代码经过了模糊化处理可读性差，但在本项目开发过程中不需要关心 Cortex M3 核内部的实现方式，只需要调用内部的功能，所以本项目采用（Eval）评估版，足够实现目标 SoC。

2.3 CMSDK 文件结构

表 2.4 CMSDK 文件描述

文件夹	描述
docs/	Cortex-M3 DesignStart Eval 相关文档
cortexm3_model/	Cortex-M3 处理器综合层模型
cmsdk/	包含 ARM Cortex-M 系统工具箱，应用在 Cortex-M3 DesignStart 的开发组件的 RTL 程序
m3designstart/	包含 Cortex-M3 DesignStart 评估版系统样例 在 testbench/execution_tb/下有需要的 Testbench 在 testbench/testcodes/有综合测试程序 ARM Cortex 系列微控制器接口标准支持文件（CMSIS） 搭建 FPGA 镜像的脚本文件
m3designstart_iot/	Cortex-M3 DesignStart ARM CoreLink SSE-050 子系统评估版。
rtc_pl031/	实时时钟外围设备。
smm/	MPS2+ FPGA 不同平台外围设备和对应的程序
shared/	IP-XACT 总线定义。

2.4 CMSIS 介绍与使用

CMSIS 是 ARM Cortex 微控制器/处理器软件接口标准。CMSIS 层属于设计的软件方面，位于硬件层与用户之间，提供了一个硬件抽象层方便用户开发使用，此外还为实时操作系统、外设提供简单的处理器软件接口，屏蔽了硬件差异，可以极大地提高软件的移植性。也就是说我们可以直接调用 CMSIS 的顶层来操作 Cortex M3 内核实现想要的功能。在 CMSDK 的 m3designstart\software\cmsis\CMSIS\Include 文件夹下就包含 CMSIS 最核心的三个头文件。其中包含用于访问内核寄存器的名称、地址定义。设备外设访问层：提供了片上的核外外设的地址核终端定义。在 m3designstart\software\cmsis\Device\ARM\CM3DS\Source 下可以找到驱动文件，驱动文件中给出了大量基础的驱动程序，包括计时器及后面需要用到的 GPIO、UART 等驱动函数。此外在文件夹 m3designstart\software\common\demos 下还还有一些常用功能的例子如 Watch dog、Interrupt、sleep 等。

软件端除了三个重要的头文件还需要一个启动程序，在 m3designstart\software\cmsis\Device\ARM\CM3DS\Source\ARM 下可以找到，该函数是由汇编语言编写的，是整个系统的启动程序，其中包含 __Vectors 定义了中断向量表核中断句柄。当中断产生时便会通过中断向量表定位到对应的中断然后调用对应的句柄执行中断函数。这些给出的例子都极大地降低了开发难度。图 2-1 是通过 CMSIS-DAP 下载器在 keil 开发环境下与系统正常通信的截图。

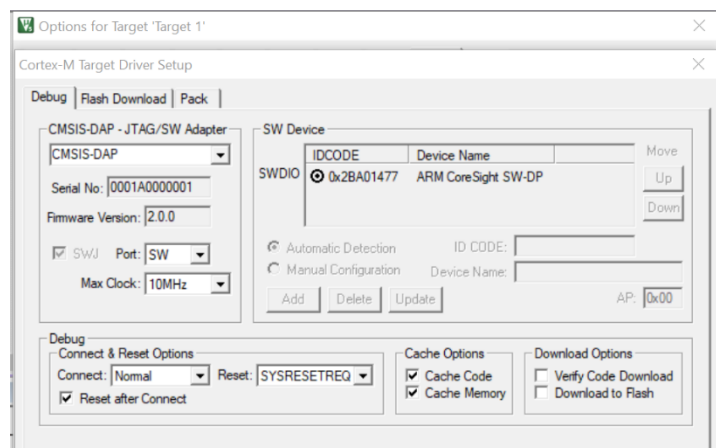


图 2-1 Keil 与系统正常通信

2.5 Cortex M3 定制

利用上述的 CMSDK 和 CMSIS 可以很方便地开发基于 Cortex M3 微控制器的片上系统，如图 2-2 是本项目使用的系统整体架构图，其中 FPGA 芯片内主要搭载 Cortex M3 核、AMBA 总线和图像处理加速器。外围设备通过 AMBA 总线直接或间接地与 Cortex M3 核通信。CMSIS-DAP Debug Port 是用来下载程序、调试程序地接口。

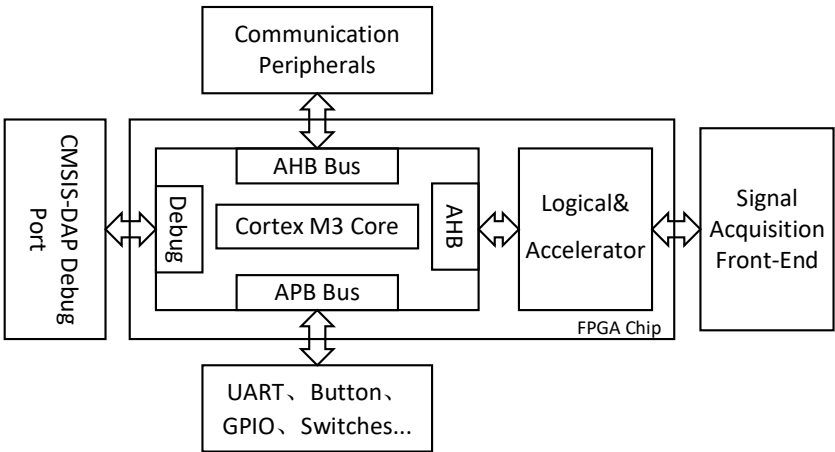


图 2-2 Cortex M3 定制系统整体框图

2.6 Cortex M3 核仿真测试

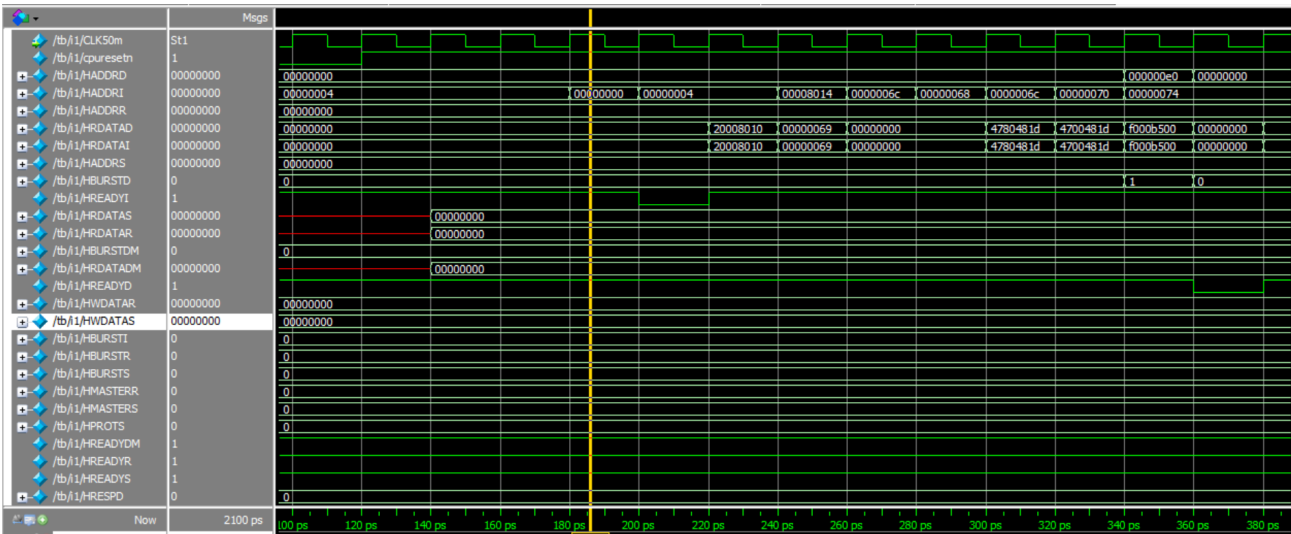


图 2-3 Cortex M3 核仿真测试

图 2-3 是 Cortex M3 内核定制后的仿真结果，从 2-3 是下载到 Cortex M3 内程序的反汇编代码，在图 2-4 中可以看到，0x00000000 地址放的是 20008010 指令，0x00000004 放的是 00000061 指令。在仿真结果中也可以看到在系统复位后地址总线 HADDRI 值为 0x00000000，当 HREADYI 为 1 时 HRDATAI 开始读指令 20008010，然后读 00000069 依次执行。Cortex M3 内核可以正常运行。

```

D: > desk > Cortexm3lite > CortexM3Lite > Keil > image.hex
1  20008010
2  00000069
3  00000071
4  00000079
5  00000081
6  00000089

```

图 2-4 程序反汇编代码

3. 主要外设模块设计

3.1 总体架构设计

在 ARM 官网上可以看到他们给出的 SoC 架构，根据本项目的实际需求并参照 ARM 官网给出的 SoC 结构设计出本项目的 SoC 总体架构如图 3-1。

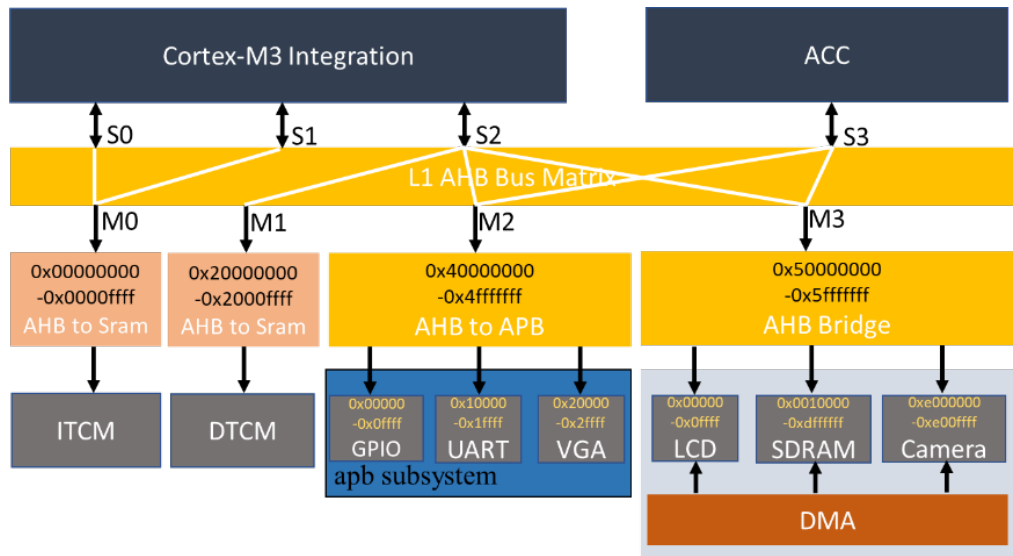


图 3-1 本项目使用的 SoC 架构

架构中有 Cortex M3 核和 ACC 两个主设备，ITCM、DTCM、GPIO 等多个从设备。其中 ITCM 和 DTCM 主要用来存放指令和数据。GPIO、UART、VGA 主要是辅助显示结果。LCD、SDRAM、Camera 是负责获取、存储并显示图像和结果。

3.2 AHB 总线设计与生成

根据架构中有 4 个 master 分别为：DCODE、ICODE、SYS、ACC，第一层 AHB 有四个 Slave 分别为 ITCM、DTCM、AHB_to_APB、AHB Bridge。这里采用了一种稀疏的地址映射（每个 S* 端口并没有与每个 M* 端口连接），同时也采用一种固定的地址映射（无 remap 且 M 端口与任意 S 端口之间的数据传输地址是固定的，例如 S2 与 S3 均通过 0x40000000-0x4ffffff 地址访问 M2 端

3.3 FIFO 控制器设计与实现

FIFO 模块是本项目中非常重要的部分，FIFO 作为缓冲模块可以实现不同工作频率下的数据传输。在 LCD、Camera 模块中都是用了 FIFO 模块。在 Quartus II 的 IP library 中有 FIFO 模块，这里可以直接调用对应的 FIFO 核修改对应的参数生成所需的 FIFO 模块。下面以简单的位宽为 8，深度为 8 的单时钟 fifo 为例进行测试。在 Quartus II 的 IP catalog 中找到 FIFO 模块根据需要配置对应的参数，配置完成后会生成 fifo.v 和 fifo_bb.v 两个 v 文件，这两个 v 文件就是我们需要的 fifo 源程序，为了验证功能还需要编写 testbench 程序，在 testbench 中对各个引脚初始化然后配置时钟控制信号，最后在 ModelSim 中仿真结果如下：

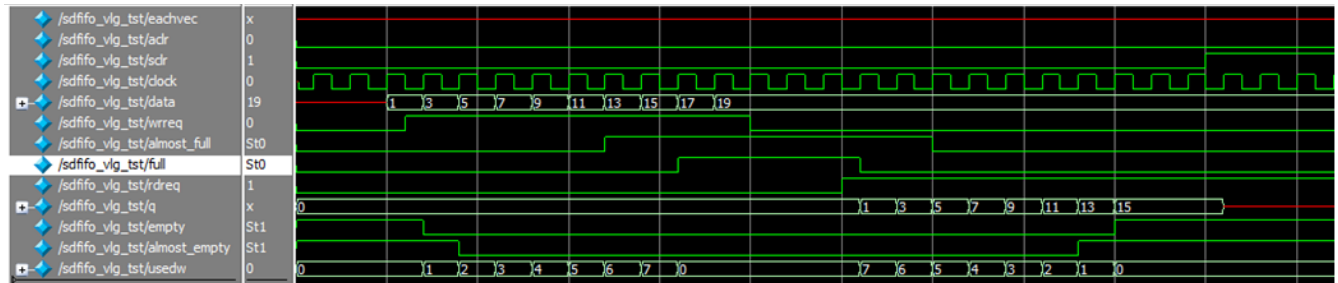


图 3-4 FIFO 功能验证

当写使能 wrreq 为 1 时每遇到一个时钟上升沿就会写入一个数据到 fifo 中，通过 usedw 也可以看出每遇到时钟上升沿就会加一直到写满 8 位从 0 开始。因为 fifo 已经写满所以图中 17，19 两位数据并没有写入 fifo，当读使能 rdreq 为 1 时每遇到一个时钟上升沿就会读一个数据到 q 然后 usedw 递减，当只剩 2 个数据时 almost_empty 拉高。因此 fifo 可以正常工作。

3.4 GPIO 控制器设计与实现

在 CMSDK 中有定义好的 cmsdk_ahb_gpio 和 cmsdk_iop_gpio 模块，这两个模块的 rtl 结构如下图 3-5 所示：

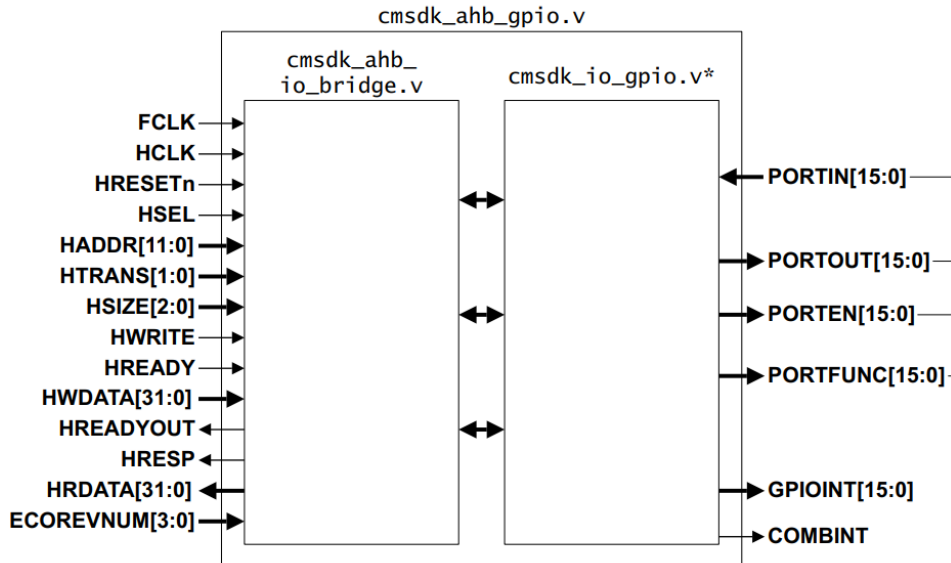


图 3-5 CMSDK 中 GPIO 模块结构图，摘自 Cortex_m_system_design_kit

将该模块通过顶层连接到 AHB 总线上，便将 GPIO 接到了系统中。在 cmsdk_io_gpio 中定义了各个寄存器的功能：

```
// 0x000 RW   Data
// 0x004 RW   Data Output latch
// 0x010 RW   Output Enable Set
// 0x014 RW   Output Enable Clear
// 0x018 RW   Alternate Function Set
// 0x01C RW   Alternate Function Clear
// 0x020 RW   Interrupt Enable Set
```

.....

以上都是偏移地址，在软件端定义 GPIO 结构体与硬件地址对应

```
typedef struct
{
    __IO uint32_t DATA; /* Offset: 0x000 (R/W) DATA Register */
    __IO uint32_t DATAOUT; /* Offset: 0x004 (R/W) Data Output Latch Register */
    uint32_t RESERVED0[2];
    __IO uint32_t OUTENABLESET; /* Offset: 0x010 (R/W) Output Enable Set Register */
    __IO uint32_t OUTENABLECLR; /* Offset: 0x014 (R/W) Output Enable Clear Register */
    __IO uint32_t ALTFUNCSET; /* Offset: 0x018 (R/W) Alternate Function Set Register */
    __IO uint32_t ALTFUNCCLR; /* Offset: 0x01C (R/W) Alternate Function Clear Register */
    __IO uint32_t INTENSET; /* Offset: 0x020 (R/W) Interrupt Enable Set Register */
    __IO uint32_t INTENCLR; /* Offset: 0x024 (R/W) Interrupt Enable Clear Register */
    .....
} CM3DS_MPS2_GPIO_TypeDef;
```

下面是对应的基地址

```
#define AHB2_BASE (0x50000000UL)
#define CM3DS_MPS2_GPIO0_BASE (AHB2_BASE + 0x0000UL)
```

然后通过操纵结构体中的子对象就可以修改对应的硬件寄存器值实现 GPIO 的控制。如可以先将 OUTENSET 寄存器的第一位二进制值置 1，即开启 GPIO0 的输出功能，然后操作 GPIO 的 DATAOUT 寄存器（偏移地址为 0x004），将 DATAOUT 的第一位二进制值置 1 的话 GPIO0 就会输出高电平。（这里对寄存器的操作都是在软件端进行的，通过 CMSIS 协议调用内核的程序修改对应的寄存器）。同理可以通过修改对应的值启用 GPIO 中断，但启用中断时还需要修改软件与硬件程序。在 Verilog 顶层中断向量中添加自定义的中断：

```
assign IRQ = {23'b0, COMBINT, GPIOINT[7], TXOVRINT, RXOVRINT, RXINT, TXINT};
```

从右到左是自己添加的五个中断，左侧填充 0。在软件方面同样需要添加中断，首先也是在中断结构体中添加对应的中断：

```
typedef enum IRQn
{
    /*** Cortex-M3 Processor Exceptions Numbers ***/
    NonMaskableInt_IRQn = -14, /*!<2 Cortex-M3 Non Maskable Interrupt */
    .....
    SysTick_IRQn = -1, /*!< 15 Cortex-M3 System Tick Interrupt*/

    /***CM3DS_MPS2 Specific Interrupt Numbers ***/
    UARTTX_IRQn = 0, /* UART 0 RX and TX Combined Interrupt */
}
```

```

UARTRX_IRQn          = 1,    /* Undefined */
UARTOVR_IRQn         = 2,    /* UART 1 RX and TX Combined Interrupt */
PORT0_7_IRQn         = 3,
PORT0_ALL_IRQn       = 4
} IRQn_Type;

```

此外还需要在系统文件中添加对应的句柄：

```

; External Interrupts
    DCD    UARTRX_Handler      ; UART RX Handler
    DCD    UARTRX_Handler      ; UART TX Handler
    DCD    UARTOVR_Handler     ; UART RX and TX OVERRIDE Handler
    DCD    PORT0_7_Handler     ; GPIO 7 Handler
    DCD    PORT0_ALL_Handler

PORT0_7_Handler PROC
    EXPORT PORT0_7_Handler     [WEAK]
    IMPORT PORT07Handler
    PUSH    {LR}
    BL      PORT07Handler
    POP     {PC}
    ENDP

```

然后在中断句柄中定义中断所需要执行的具体操作：

```

void PORT07Handler(void){
    CM3DS_MPS2_GPIO0->DATAOUT=~((CM3DS_MPS2_GPIO0->DATAOUT)&0x0001);
    CM3DS_MPS2_GPIO0->INTCLEAR=0x00080;
}

```

通过以上操作就实现了中断处理全过程。触发中断可参见 Cortex™-M System Design Kit 中 GPIO 各寄存器的作用，按照手册把对应的寄存器配置后便可以开启中断，当 GPIO 开启中断后，外部中断条件来临时中断就会触发从而执行自己定义的中断程序。在本例中开启了第 8 个 GPIO 即 GPIO7，首先将 interrupt enable[7]（即偏移地址 0x0020 的二进制第 8 位数据）置 1，然后 interrupt polarity[7]（即偏移地址 0x0034 的二进制第 8 位数据）和 interrupt type[7]（即偏移地址 0x0028 的二进制第 8 位）置 1 开启上升沿触发中断。以下是该中断的执行顺序：

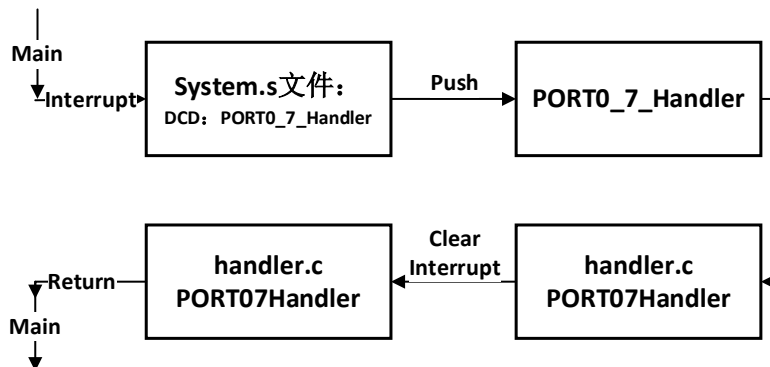


图 3-6 中断执行过程

在添加 UART、LCD 等外设的流程与 GPIO 基本一致，在下面的部分只进行简单描述。

3.5 UART 控制器设计

3.5.1 功能实现

CMSDK 文件中有封装好的 UART 程序，只需要移植对应的程序便能实现一些基本的数据传送功能。在 cmsdk/logical/下有 cmsdk_ahb_to_apb 和 cmsdk_apb_uart 文件夹，前者是将 AHB 转成 APB，后者是接在 APB 下面的 UART 外设。同样按照接口定义在顶层文件（在）中把 cmsdk_ahb_to_apb 接在 AHB 下，然后将 UART 外设接在 APB 下便实现了硬件的连接。

但在使用之前需要完成软件与硬件地址匹配。在 cmsdk_apb_uart 文件中已经给出各个偏移地址的作用，如：

```
// 0x00 R   RXD[7:0]   Received Data
//      W   TXD[7:0]   Transmit data
// 0x04 RW  STAT[3:0]
.....
```

在软件端需要定义对应的结构体：

```
typedef struct
{
    __IO uint32_t DATA;      /*!< Offset: 0x000 Data Register  (R/W) */
    __IO uint32_t STATE;      /*!< Offset: 0x004 Status Register (R/W) */
    __IO uint32_t CTRL;       /*!< Offset: 0x008 Control Register (R/W) */
    .....
} UART_TypeDef;
```

然后定义一个 UART 结构体指针指向 UART 的地址 0x40000000+0x10000 便可以通过操作 UART 直接访问硬件地址从而控制 UART 传送数据。

在软件端如需要编写 UART 的驱动程序，具体的驱动程序可以参照 CMSDK 文件夹中的例子。具体参见 cmsdk\m3designstart\software\cmsis\Device\ARM\CM3DS\Source 下的驱动文件，里面定义了包括 UART 在内的驱动程序，在主程序中可以直接调用这些驱动函数实现数据传送功能。在 CM3DS_MPS2_uart_ReceiveChar 函数可以直接用来接收串口发送的数据（调试时需要串口收发器）。此外还需要在.s 文件中编写 UART 初始化函数，初始化波特率是接收使能发送使能等。然后在主函数中调用对应的函数便能相应的功能。比如实现简单的 UART 接收与发送：

在 main 函数中调用 UART

```
char *a[100];
while(1) {
    scanf("%s",a);
    printf("double string : %s %s\n",a);
}
```

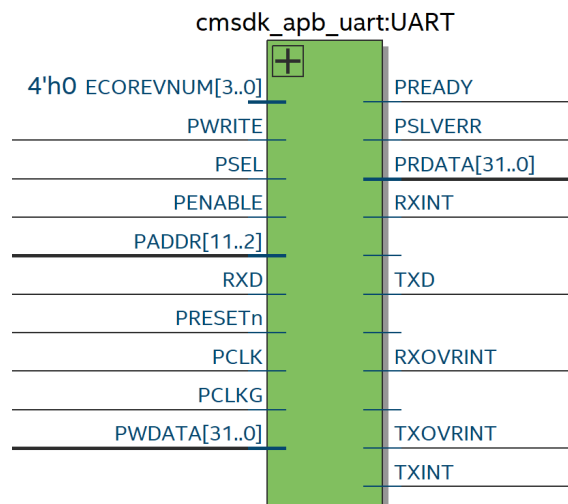


图 3-7 UART 模块的 RTL 图

表 3.1 端口解释

名称	类别	作用
PADDR	输入	APB 地址总线
PSEL	输入	APB 片选信号
RXD	输入	数据接收信号
PRESETn	输入	APB 复位信号
PCLK	输入	APB 时钟
PWDATA	输入	APB 数据总线
READY	输出	APB Slave 准备信号
PSLVERR	输出	APB 传输失败信号
PRDATA	输出	APB 读数据信号
RXINT	输出	接收中断信号
TXD	输出	数据发送信号
RXOVRINT	输出	接收溢出中断信号
TXOVRINT	输出	传输溢出中断信号
TXINT	输出	传输中断信号信号

表中的 APB 总线相关的信号主要是用来完成 APB 通讯，其他信号用来控制 UART。

3.5.2 功能测试

直接通过串口调试助手验证 UART 功能，选好端口和波特率，在 Keil 端运行 C 程序，然后通过串口调试助手发送数据，下图中黑色字体为发送数据，红色字体为接收数据，从而验证了 UART 工作正常如右图 3-8 所示。

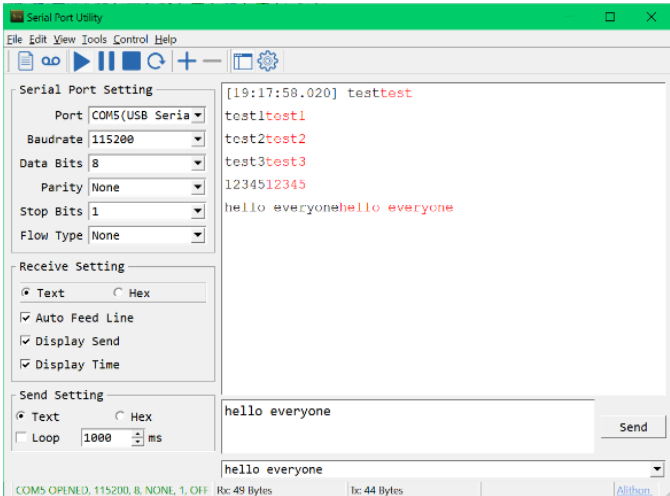


图 3.8 UART 通信测试

3.6 LCD 控制器设计

3.6.1 功能实现

本项目使用的是 2.4 TFT SPI 240X320 的液晶显示屏，首先需要了解该 LCD 显示屏的工作时序，然后根据工作时序和参考文档编写 LCD controller 模块控制 LCD 显示屏，本项目 LCD 模块最终的实现效果如下图 3-9 所示。

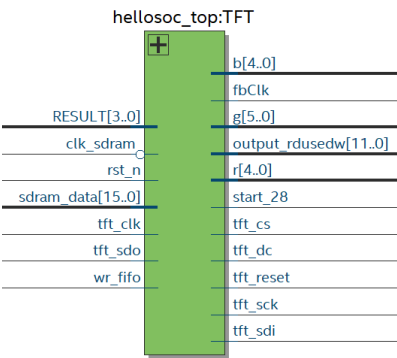


图 3-9 LCD 控制模块

表 3.2 端口解释

名称	类别	作用
RESULT	输入	手写体识别结果
clk_sdrām	输入	sdrām 可读信号与 fifos 时钟信号同步
rst_n	输入	复位
sdrām_data	输入	sdrām 数据线输入到 fifo 中
tft_clk	输出	LCD 时钟信号
tft_sdo	输入	主设备数据输出
wr_fifo	输入	fifo 写信号
b	输出	VGA 接口 blue 信号
fbClk	输出	fifo 取数据时钟
g	输出	VGA 接口 green 信号
output_rdusedw	输出	fifo 中可读 word 数
r	输出	VGA 接口 red 信号
start_28	输出	识别结果显示使能
tft_cs	输出	LCD 片选信号
tft_dc	输出	LCD 寄存器/数据选择信号
tft_reset	输出	LCD 复位信号
tft_sck	输出	SPI 总线时钟信号
tft_sdi	输出	主设备数据输入

该模块主要实现 camera 数据与识别结果显示功能，camera 数据通过 DMA 保存到 SDRAM 中，然后再通过 sdrām_data 数据线传送到 LCD 模块内，因为 SDRAM 与 LCD 工作频率存在差异，所以中间又通过一个 fifo 来缓存数据，其中 wr_fifo、fbClk、output_rdusedw 等信号都是用来操作 fifo。此外就是一些 LCD 外设接口如 tft_cs、tft_dc、tft_reset 等信号与 LCD 硬件直接相连。Camera、LCD、SDRAM 通过 DMA 实现通信，具体的数据传输可以参考下图：

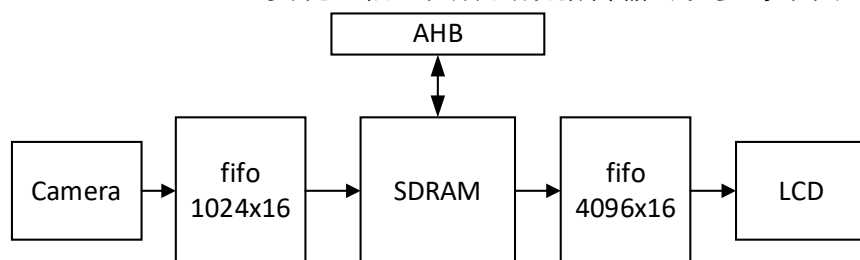


图 3-10 DMA 部分数据传输路径

3.6.2 功能验证

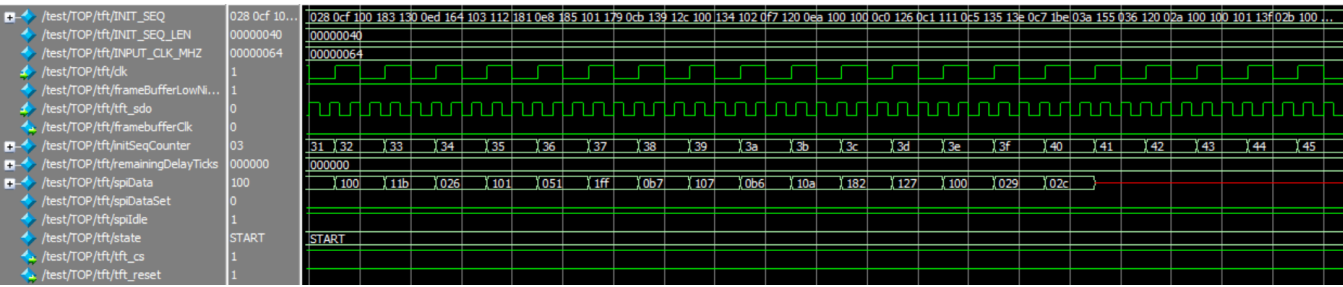


图 3-11 LCD 模块功能验证

由于该 LCD 只支持 SPI 数据传输，所以在 LCD 内部需要通过数据转换将 fifo 中的数据串行输出。在仿真结果中可以看到 spiData 与 spiDataSet 等信号是 LCD 内部分 SPI 数据传输部分控制信号，其中 spiData 与 tft_sdi 相连，直接控制 LCD 显示，tft_sdo 则是由主设备输入的数据。此外在时钟信号的控制下，内部的计数器与数据传输都能正常传输，cs、reset 控制信号也能正常工作，所以 LCD 模块逻辑上没有问题。

3.7 Camera 控制器设计

3.7.1 简介

VSYNC 为高时产生一个帧同步信号，产生两个帧同步信号时，一帧数据传输完成。HSYNC 和 HREF 是一个引脚产生的信号，只是在不同的场合下面，使用的不同信号方式。

3.7.2 实现

OV7670 是非常常见的摄像头模块，在本项目中参照硬件供应商提供的样例，修改部分参数便实现 Camera 的配置和控制。下图是设计出来的 camera 配置与控制 RTL 图。

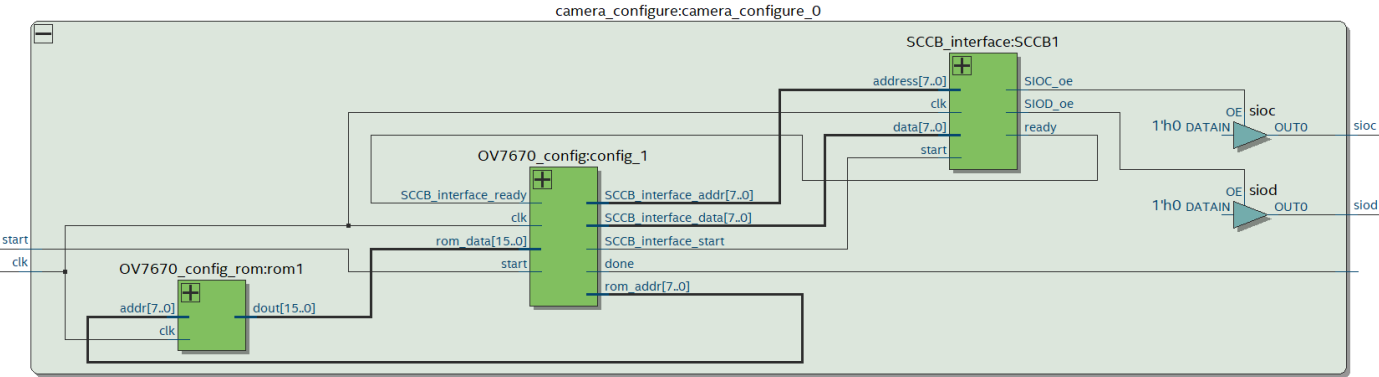


图 3-12 摄像头配置模块

摄像头模块是工作之前需要配置寄存器和工作模式，图 3-12 为配置模块，其中 sioc 与 siod 是 SCCB 通信协议最核心的数据传输线，sioc 是控制信号，siod 是串行输入数据。done 是配置完成信号用来启动数据采集，具体的配置过程是通过图 3-13 所示的有限状态机实现的，默认情况下状态机处于空闲状态（FSM_IDLE），当接收到 start 信号后进入 FSM_SEND_CMD 状态开始

将 ROM 中的数据转换成 16 位宽的数据传输到 SCCB 模块转换成串行输入。当快要到达 ROM 顶时给计数器赋值产生 10ms 延迟。当到达 ROM 顶层时进入 FSM_DONE 状态，完成一次转换。其中 FSM_TIMER 主要用来计数实现延迟，与 FSM_SEND_CMD 状态协同工作完成数据传输。

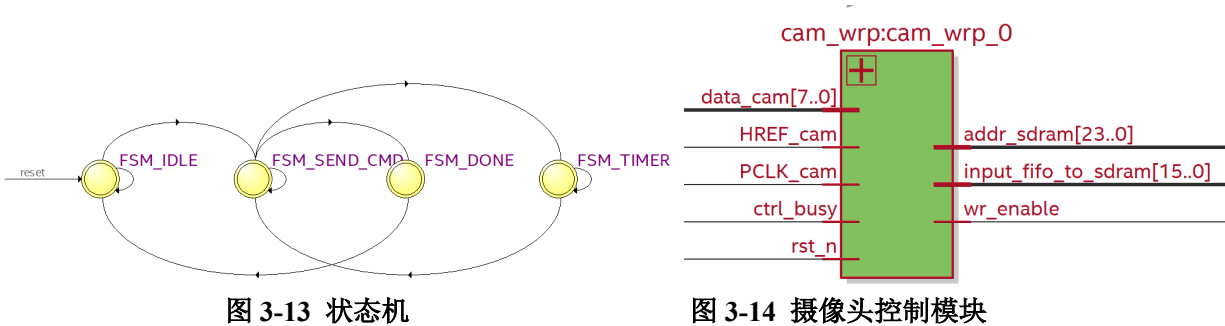


表 3-3 摄像头控制模块引脚说明

名称	类别	作用
data_cam	输入	camera 数据输入
HREF_cam	输入	场同步信号
PCLK_cam	输入	时钟信号
ctrl_busy	输入	判断数据是否忙碌
rst_n	输入	复位信号
addr_sdram	输出	sdram 地址线
wr_enable	输出	写使能信号
input_fifo_to_sdram	输出	输入先写入 fifo

图 3-14 为摄像头控制模块，camera 获取的数据首先写入 fifo 中然后再写入到 sdram 中。其中 data_cam 是摄像头采集到的图像数据，HREF_cam 与 PCLK_cam 则是数据同步信号，PCLK_cam 用与 fifo 同步，ctrl_busy 用来控制读 fifo 时钟，addr_sdram 与 wrenable 是控制 SDRAM 的信号。

3.8 SDRAM/SRAM 控制器

3.8.1 简介

SDRAM 与 SRAM 有相似的使用方法，这里先简单介绍下 SDRAM。SDRAM 时钟需要与 Cortex m3 核前端总线的系统时钟同步，内部数据传输、指令发送以此为标准。动态是指存储器需要不断刷新防止数据丢失。随机是指可以在存储器任何位置进行读写操作。

3.8.2 实现

操作存储器如 SRAM、SDRAM、FIFO 等都需要设计 controller，在 Quartus II 中有对应的 IP 库可以很方便地实现 Controller 设计。具体操作如下，在 Quartus II 地 IP Library 中找到所需要的库然后根据 MegaWizard Plug-In Manager 指引配置对应地参数，包括所使用地芯片类型，SDRAM

大小，时钟频率等参数。下图是配置后自动生成的 SRAM controller 与 SDRAM controller RTL 如图 3-15 和 3-16：

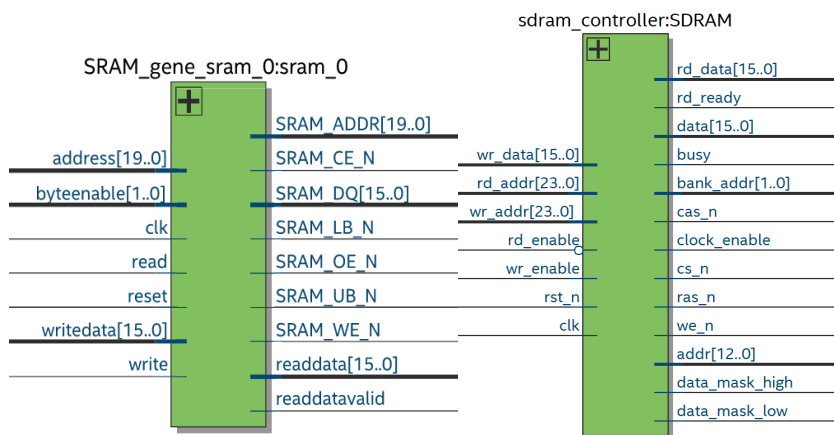


图 3-15 SRAM 控制模块

图 3-16 SDRAM 控制模块

引脚说明（以 SDRAM 为例），cs_n 是片选信号；ras_n 行地址选通信号；cas_n 列地址选通信号；we_n 写使能，低电平有效；dqm 数据掩码，确定 dq（数据 I/O）的有效部分；sd_addr 存储器地址；ba 确定 L-Bank（Logic Bank）地址线；cke 时钟使能。

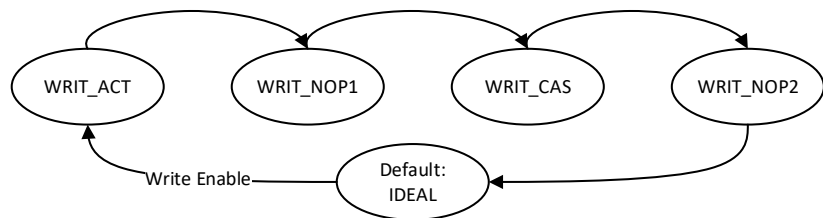


图 3-17 控制 SDRAM 读写的有限状态机

SDRAM 通过多个有限状态机控制读写中断处理等功能，图 X 是写操作的状态机实现，它与状态计数器（非零时每遇到一个时钟上升沿递减 1，当为 0 时所有状态机进入空闲状态）、控制命令一起工作。默认情况是空闲状态，一旦检测到写使能就会开始进入写状态，然后依次进入下一个状态，在第一个状态下会将状态计数器置 1 然后进入第二个状态，在第二个状态会指定控制命令进入 CMD_WRIT（即写命令），然后进入 WRITE_NOP2 状态，该状态类似于 IDEAL 状态，但当 Write Enable 为高时会直接进入下一个写周期。读操作也使用了类似的状态机实现。

3.8.3 功能验证

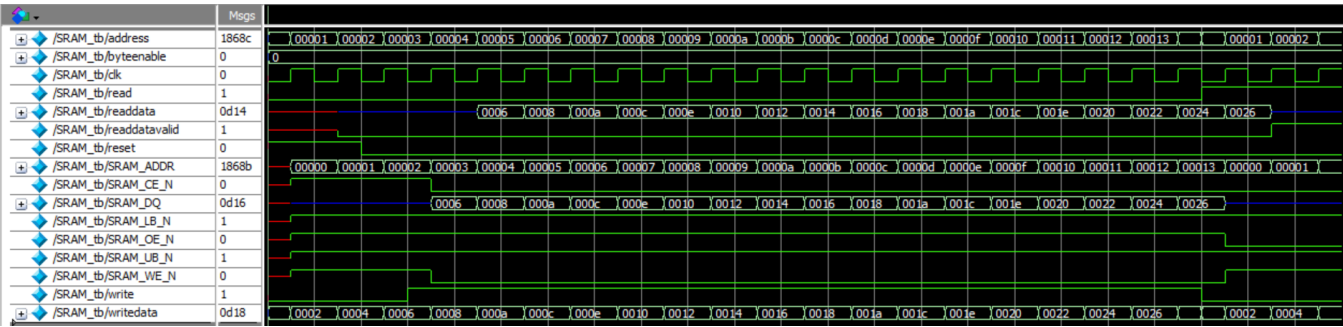


图 3-18 SDRAM 时序验证

图中当 address 地址与 SRAM_ADDR 保持一致，writedata 一直有数据但只有当 write 为 SDRAM_WE_N 使能信号和 SRAM_CE_N 片选信号同时拉低开始写操作，每遇到一个时钟上升沿 DQ 中就载入一个数据写到 SRAM 中，SRAM 功能正常。

3.9 顶层模块设计

以上模块完成后需要设计一个顶层模块将各个子模块连接起来，在上文中已经生成总线模块，在顶层文件中首先例化 AHB 总线模块，定义对应的线网与外设连接起来。比如将 GPIO 挂在 APB 总线的 P0 处就需要先将 AHB_APB 模块挂在第一层 AHB 总线的 P2 位置。其他模块都按照总体架构依次连接。

4. 基于 FPGA 的图像处理加速方式

4.1 流水线、并行加速

Cortex M3 内核自带三级流水线操作，在 4-2 图中可以看出。关于并行性一般会复用多个处理单元（Processing Element）来实现并行加速的目的，这也是通过面积换速度最直接的方式。在胡航天等人的研究中^[10]通过对单个运算单元功能扩展，可以有效的完成卷积神经网络中卷积、池化和全连接等运算，运算单元矩阵可以极大提升图像处理速度。许欣等人^[11]中提出一种基于数据对齐并行处理、多卷积核并行计算的硬件架构设计方法最终将性能提高 20%-60%。

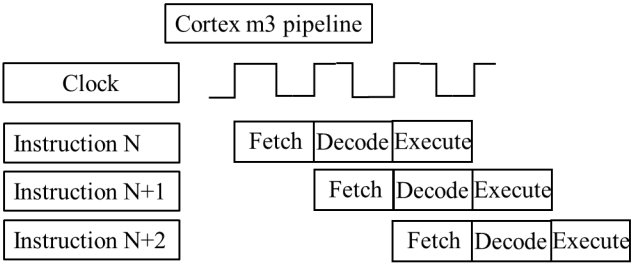


图 4-1 Cortex M3 三级流水线工作模式

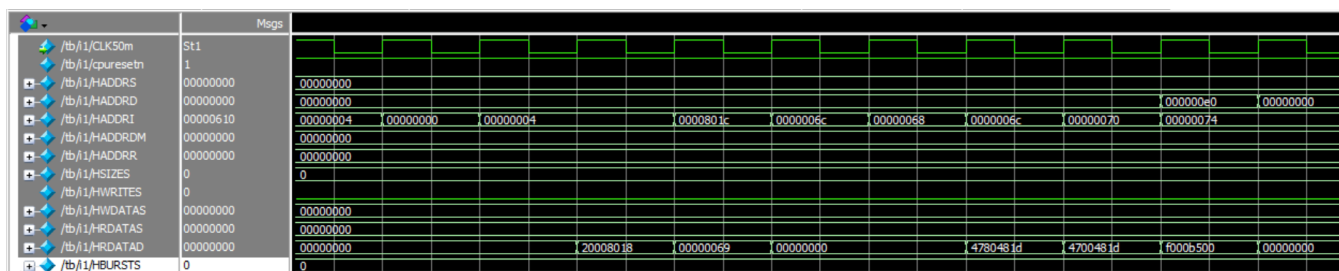


图 4-2 流水线验证

由仿真结果可以看出 Cortex M3 会连续取 3 个地址然后解码执行，这可以极大地提高代码的运行效率。

4.2 指令集加速

指令集加速主要是通过自定义指令集简化操作流程提高代码执行效率，在^[12]文献中设计了一套专用 AI 指令集并应用在了基于 FPGA 的神经网络加速器的设计中。整个指令集由失灵寄存器、指令解释器、指令转发模块、内存管理单元和多个模块构成，最终可以减少约 50%的 FPGA 逻辑资源。

4.3 压缩数据量

压缩数据量是拿精度换速度最直接的方式，比如可以将 32 位宽的数据压缩到 16 位或者更低，实验表明，当数据量压缩一半是，算法的精度只降低了 1%，而速度提升将近 4 倍。此外还可以通过不同编码方式压缩数据量。由于数据中存在大量的 0，可以采用不同的编码压缩数据 0 占用的空间，在运算时再通过译码把 0 复原。

4.4 优化数据传输

实验证明传送数据在运算过程中代价是非常高的。此外，从表 4-1 中可以看出不同类别的存储器的数据传输速度差距非常大。

表 4-1 存储器功耗速度对比

Category	SRAM	SDRAM	FLASH	HDD
Energy per bit(pJ)	0.0005	0.05	0.00002	$1-10 \times 10^9$
Read time(ns)	0.1-0.3	10	100000	$5-8 \times 10^6$
Write time(ns)	0.1-0.3	10	100000	$5-8 \times 10^6$
Endurance(cycles)	10^{16}	10^{16}	10^4	10^4

摘自中国存储网: <http://www.chinastor.com/baike/storage/020162332017.html>

尽可能降低数据传输消耗的时间也是加速的一个重要的方法。因此在设计中在资源允许的情况下应尽量使用高速存储器如 SRAM、FiFo 等。数据复用也可以算是优化数据传输，因为通过复用可以减少数据的传输次数从而缩短时间。数据复用可以分为多种，在本项目中数据复用主要存在卷积运算过程中，如下图 4-3 所示。

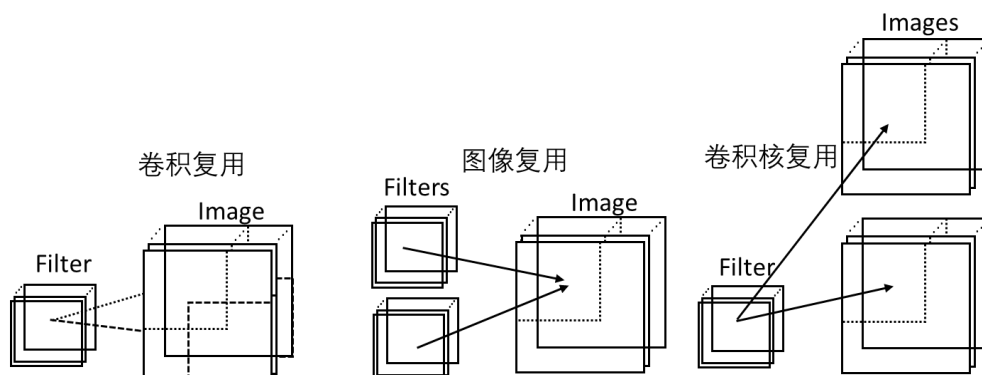


图 4-3 三种数据复用形式

4.5 算法优化

算法优化不输于硬件层面的优化，但是算法的优化可以通过硬件实现，从而大大加速运算速度。比如削减神经网络中的卷积层，将卷积转化成可分离卷积等。例如 Mobile Net^{[5][6][7]}网络通过简化^[13]去掉 6 层网络层后的识别率仅仅降低不到 1%，而参数量降低近 50%，运算量也降低近 50%。但目前通过算法加速图像处理速度的步伐相比于通过设计硬件加速器显的微不足道。这主要是因为硬件加速还有很大的潜力待挖掘。

4.6 轻量级卷积加速器加速器设计

在神经网络中卷积运算占总运算量的 90%-99%，所以本项目针对卷积运算设计加速器，在前人的基础上做一定的改进。在上面小节中已经讨论了主流的加速器设计方案，本项目中也采用经典的并行计算和数据传输优化方案。

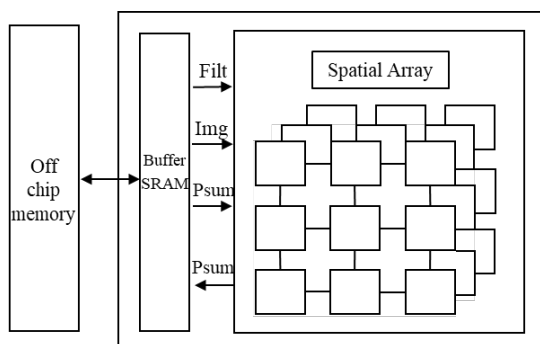


图 4-4 加速器总体架构

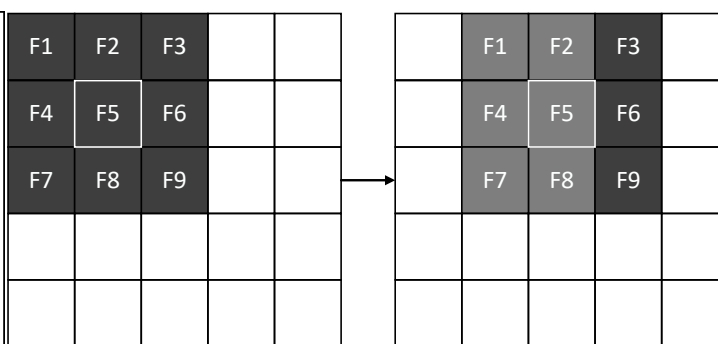


图 4-5 数据复用

上图 4-4 是加速器的顶层结构。通过高效的数据复用最小化数据传输量。通过缓存器输入图像数据使数据传输速度最快。为了优化数据传输，使用了数据复用功能，即单个缓冲区可以被多个 pe 使用。滤波器的权重和部分和共用 108KB 的 SRAM 缓存器实现了数据复用。数据流和数据复用允许系统达到高效的运算效果。运算阵列和滤波器权重之间的内部乘积，生成部分和，从阵列返回到缓冲区，再经过校正(ReLU)压缩到 DRAM。压缩使图像平均带宽减少 2 倍。每个 PE 负责计算输入图像和滤波器权值的内积，为了减少计算量使用了数据复用机制，前一个滤波器卷积结果其中的六个内积和（如图 4-5 右图的灰色部分）在第二个滤波器卷积中重用。数据门控是通过在“零缓冲区”中记录输入的零图像值，跳过过滤器读取、计算这些值，压缩大量 0 数据，避

免不必要的读和运算操作。压缩数据降低功耗减小带宽。数据压缩可以缩小 1.9 倍的读图像带宽，缩小 2.6 倍的写图像带宽从而在 PE 中节省 45%的运算量。

Compression Example
Input: 0 0 12 0 0 0 53 0 0 22
Output: 2 12 3 53 2 22

图 4-6 数据压缩例子

4.7 手写体识别系统实现

本项目手写体识别使用 LeNet 网络实现，并对该网络做了一定的简化，相比于深度卷积网络，该网络具有参数少，结构简单，运算量小等特点。最重要的是该网络适合嵌入式开发，所以我们将该网络嵌入到 SoC。算法的具体实现方式、网络的训练方法、训练量等内容在另一个项目中详细说明，在此不赘述。此外硬件化实现了最大池化，激活函数等模块，这些模块协同工作可以实现手写体识别系统。

5. 系统测试

本项目使用的硬件是 DE-2 开发板，搭载的是 Cyclone IV 芯片，烧录程序之前需要设置引脚约束图 5-1 是对应的引脚约束图，图 5-2 是上板测试结果。表 5-1 是通过使用 SketchBook 软件书写的不同像素宽度的手写体数字识别效果，像素宽度小时识别效果非常差，随着宽度增大识别效果也逐渐变好。表 5-2 实验了不同类型的手写体数字识别效果，从结果可以看出正常的手写体数字基本都能识别，识别效果总体不错。5-3 是对 0-9 数字的识别，从识别结果可以看出，除了 7 和 1 容易混淆，3 需要书写的比较宽才能准确识别外，其他数字都可以很准确被识别出来，所以整个系统识别效果比较理想。

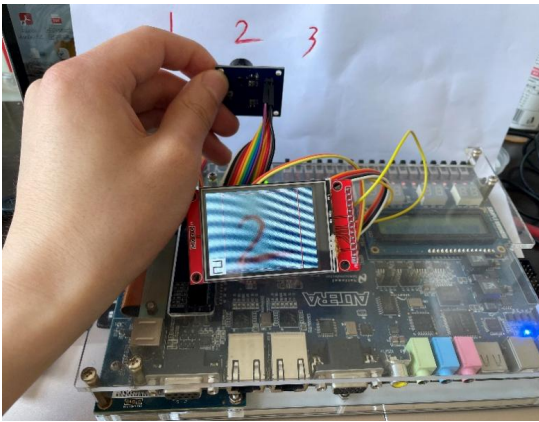


图 5-2 上板测试

表 5-1 不同像素宽度的手写体数字识别效果






	字体宽度/像素	能否识别	效果
	15	×	识别结果非常不稳定
	20	×	识别结果非常不稳定
	25	√	识别结果不稳定，但可以正确识别
	30	√	效果比较好，可以准确识别
	35	√	效果很好，可以准确识别

表 5-2 不同形式的手写体数字识别效果






	字体宽度/像素	能否识别	效果
	35	√	识别结果不稳定，容易错误识别为 7
	35	×	识别结果非常不稳定
	35	√	识别结果略不稳定，但可以正确识别
	35	√	效果比较好，可以准确识别
	35	√	效果比较好，可以准确识别

表 5-3 所有手写体数字识别效果

	字体宽度/像素	能否识别	效果
0	35	√	可以准确识别
1	35	√	可以准确识别
2	35	√	识别结果略不稳定，容易误识别为 7
3	35	√	识别结果不太稳定
3	35	√	可以准确识别
4	35	√	可以准确识别
5	35	√	可以准确识别
6	35	√	可以准确识别
7	35	√	识别结果不太稳定，容易误识别为 1
7	35	√	可以准确识别
8	35	√	可以准确识别
9	35	√	可以准确识别

6. 总结与展望

本项目在 FPGA 平台上定制一个 Cortex M3 微控制器，搭载必要的外设实现了一款体积小、扩展性高、可移植的数字手写体识别硬件系统。在此基础上利用并行计算、流水线作业、数据复用、数据传输优化等措施设计了一个简易的图像处理加速器使得该系统可以运行轻量级的 CNN 智能算法。最后在搭载了 Cyclone IV 的 DE-2 开发板上硬件实现了该系统，手写体数字识别正确率可达到 90% 以上，处理速度可达到 150 帧/s，相比于软件实现，该设计有着效率、功耗、成本等多方面的优势。

在取得以上成果的同时，该研究也存在许多不足。首先，该系统架构没有重点考虑性能问题，仍存在很大的改进空间。第二，图像处理加速器设计采用的是比较传统比较经典的加速方案，没有特别的亮点。目前有一些比较新的且性能不错的加速方案可供参考。此外目前针对某些特定神经网络架构的加速效果比较好，该项目未来会针对 Mobile Net 神经网络进行加速器设计。

参考文献

- [1] Krizhevsky, Alex (University of Toronto, Canada); Sutskever, Ilya; Hinton, Geoffrey E. Source: Advances in Neural Information Processing Systems, v 2, p 1097-1105, 2012, Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012, NIPS 2012
- [2] Simonyan, Karen (Visual Geometry Group, Department of Engineering Science, University of Oxford, United Kingdom); Zisserman, Andrew Source: 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings, 2015, 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings
- [3] Szegedy, Christian (Google Inc., United States); Liu, Wei; Jia, Yangqing; Sermanet, Pierre; Reed, Scott; Anguelov, Dragomir; Erhan, Dumitru; Vanhoucke, Vincent; Rabinovich, Andrew Source: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, v 07-12-June-2015, p 1-9, October 14, 2015, IEEE Conference on Computer Vision and Pattern Recognition, CVPR
- [4] Chen, Yu-Hsin (Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge; MA; 02139, United States); Krishna, Tushar; Emer, Joel S.; Sze, Vivienne Source: IEEE Journal of Solid-State Circuits, v 52, n 1, p 127-138, January 2017
- [5] Andrew G. Howard and Menglong Zhu and Bo Chen and D. Kalenichenko and W. Wang and Tobias Weyand and M. Andreetto and Hartwig Adam, "Efficient Convolutional Neural Networks for Mobile Vision Applications" 2017 ArXiv abs/1704.04861
- [6] M. Sandler and Andrew G. Howard and Menglong Zhu and A. Zhmoginov and Liang-Chieh Chen, "Mobile-NetV2: Inverted Residuals and Linear Bottlenecks", 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition p4510-4520, 2018
- [7] Andrew G. Howard and M. Sandler and Grace Chu and Liang-Chieh Chen and B. Chen and Mingxing Tan and W. Wang and Y. Zhu and Ruoming Pang and Vijay Vasudevan and Quoc V. Le and Hartwig Adam, Searching for MobileNetV3, 2019 IEEE/CVF International Conference on Computer Vision (ICCV), p1314-1324, 2019
- [8] H. Srivastava and K. Sarawadekar, "A Depthwise Separable Convolution Architecture for CNN Accelerator," 2020 IEEE Applied Signal Processing Conference (ASPCON), Kolkata, India, 2020, pp.1-5.
- [9] P. Song, J. Pan, C. Yang and C. Lee, "An efficient FPGA-based accelerator design for convolution," 2017 IEEE 8th International Conference on Awareness Science and Technology (iCAST), Taichung, Taiwan, 2017, pp. 494-500.
- [10] 胡航天,刘凯,马士超,郭子博.专用指令集在基于 FPGA 的神经网络加速器中的应用[J].空间控制技术与应用,2020,46(03):36-41+54.
- [11] 徐欣,刘强,王少军.一种高度并行的卷积神经网络加速器设计方法[J].哈尔滨工业大学学报,2020,52(04):31-37.
- [12] 江凯,刘志哲,修于杰,田映辉,赵晨旭,吕笑松.基于卷积神经网络加速运算单元设计[J].计算机工程与设计,2019,40(12):3620-3624.
- [13] D. Sinha and M. El-Sharkawy, "Thin MobileNet: An Enhanced MobileNet Architecture," 2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York, NY, USA, 2019, pp. 0280-0285.

参考手册: ARM Cortex M white paper

Arm® Cortex® -M System Design Kit

ARM® Cortex®-M3 DesignStart™ Eval RTL and Testbench UserGuide

ARM® Cortex® -M3 DesignStart™ Eval FPGA User Guide

ARM® Cortex® -M3 DesignStart™ Eval RTL and FPGA Quick Start Guide

ARM® Cortex® -M3 DesignStart™ Eval RTL and Testbench User Guide

ARM® Cortex® -M3 DesignStart™ Eval Customization Guide

参考书籍: Verilog+HDL 数字设计与综合 (第二版) [美]Samir Palnitkar 著 夏宇闻 胡燕祥等译

谢 辞

感谢李素梅老师在过去几个月的耐心指导，从选题到开题到中期检查再到撰写论文李老师都给出了宝贵建议。老师对我的开题报告、中期检查等文档从排版到行文思路甚至是标点符号都做了一遍又一遍的修改，直到没有任何问题。此外还为我们找实验室做实验，尽可能为我们提供一个好的研究环境。老师严谨的治学态度、时刻为学生着想的工作精神深深地感染了我，整个过程我受益匪浅，这些都将是未来学习和工作的榜样。

还要感谢与我一同完成项目的张逢同学，张逢同学虽然主要负责手写体识别算法方面，但在我开发过程中给我许多帮助，帮我查找资料，解决疑问，最终我俩合作顺利完成了整个项目。这段经历我俩受益匪浅。还要感谢常桂林、张舒豪、陈宝翔等同学在开发过程中给我的帮助和支持。此外还要感谢陈雪静同学一直以来给予我的帮助与支持，陈雪静同学一丝不苟的学术态度给我极大的感触。

通过完成此项目，我掌握了许多技术和研究方法，包括 Cortex M3 软核的实现、外设的搭建、软件驱动的编写等 FPGA 开发所具备的基础技能。此外我还了解了图像处理加速器领域国内外的状况。掌握了一些基础的图像处理硬件加速方案与实现方法。最重要的是我积累了项目开发经验，我相信这将是宝贵的科研经历。

我衷心地感谢山东大学（威海）电子系的各位老师，从系主任到每一个老师，无论是教授、副教授、讲师都兢兢业业、认真负责，竭尽全力给我们传授知识。此外在与老师频繁接触的过程中老师们的敬业、树人的精神更是让我感触颇深。在老师的高要求下我们不断提升自己，不知不觉中有了长足的进步。真心感谢山东大学（威海）、机电与信息工程学院、电子系的所有老师，他们将是未来学习的榜样。

最后，将特别的感谢送给我的父母、家人和朋友，感谢他们在物质上、生活上和精神上给予我莫大的帮助，没有他们我走不到今天，谢谢！