Research of Machine Vision Based on SOC

Abstract

Embedding eye and brain in machine is a dream for human beings. Machine vision, as the extension of human vision and brain, is a measure of the degree of automation of modern industry. With the rapid development of semiconductor technology, emerging technology will gradually make the dreams come true. Products with visual function have been applied in various domains, and it will promote and facilitate the intelligent development of automation. In this thesis, the implement of embedded machine vision will be researched.

The SOC technology, including IP reuse and S/H co-design ,is firstly introduced, and especially, Soft core regarding SOPC technology will be concretely analyzed in this paper, such as NiosII and MicroBlaze.

And then, I mainly concentrate on the introduction of CCD and CMOS image sensor, the comparison of their differences, the performance analysis of MT9V032 image sensor with global shutter. MT9V032 with a view to dynamic quality, which is more powerful than common image sensor, will be firstly used in this design.

In the following paper, the mechanism of hardware will be described, including the function of all schematic module, the selection standards of controller, SRAM and power chips and their typical configurations, especially the prevention of EMI considering signal integrity and power integrity.

Finally, the scheme of software will be introduced. I proved the feasibility of approach trough Matlab. The image processing IP core is designed to be compatible with the protocol of Avalon bus. It achieves the function of target identification when using ModelSim to simulate. The NiosII system with I^2C, PIO, UART and image processing IP are customized to connect. According to parallel port transfer agreement, the FPGA and PC achieve the communication. The software is compiled under the Visual Studio 2005, and the image results with high quality are given.

KEY WORD: MT9V032; SOPC; NIOS-II; FPGA; Machine Vision

大连海事大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明：本论文是在导师的指导下，独立进行研究工作所取得的成果，撰写成博士/硕士学位论文 “基于 SOC 的机器视觉研究”。除论文中已经注明引用的内容外，对论文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本论文中不包含任何未加明确注明的其他个人或集体已经公开发表或未公开发表的成果。

本声明的法律责任由本人承担。

论文作者签名：彭国华 2008年 3月23日

学位论文版权使用授权书

本学位论文作者及指导教师完全了解“大连海事大学研究生学位论文提交、版权使用管理办法”，同意大连海事大学保留并向国家有关部门或机构送交学位论文的复印件和电子版，允许论文被查阅和借阅。本人授权大连海事大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，也可采用影印、缩印或扫描等复制手段保存和汇编学位论文。

保密□，在_____年解密后适用本授权书。

本学位论文属于：　保密□

不保密□（请在以上方框内打“√”）

论文作者签名：彭国华　导师签名：

日期：2008年3月18日

第1章 绪论

1.1 课题来源及背景

广袤的大地上种植着大量的蔬菜等农作物，在这些农作物生长的过程当中，在早春播种过后有很长的一段时间需要给未发芽的种子盖上塑料薄膜，以使这些种子可以在很好的温度、湿度和光照条件下发芽生长。当种子发出绿色的新芽以后，如果不尽快去掉这些覆盖在上面的塑料薄膜，那么所发的新芽会因为薄膜覆盖而不透气最终枯萎掉。以前都是用人工的方法，在看到新芽发出以后组织大量的人用利器将薄膜捅破使新芽可以得到透气，由于这些农作物的土地种植面积庞大，就需要大量的人力物力进行这项工作，浪费了很多人力物力。由此我们想到了能不能制造一种可以通过自动识别新发绿芽的位置并且可以自动打孔的装置。本课题就是在此背景之下进行的研究。

1.2 国内外研究现状

随着半导体技术的发展，具备视觉功能的控制系统越来越多。很多领域中都需要通过对当前目标的图像信息进行分析，从而控制电力传动系统的动作，完成相应的任务。

机器视觉是用计算机模拟人的视觉功能从客观事物的图像中提取信息，进行处理并加以理解，最终用于检测与控制[1][2]。图示1-1所示为一个典型的工业机器视觉应用系统：首先利用摄像机获得被测目标的图像，图像采集卡负责图像存储和传送，计算机再根据像素分布，亮度和颜色等信息，进行各种运算来抽取目标的特征，然后再根据预设的判别准则输出判断结果，控制驱动执行机构进行相应处理。

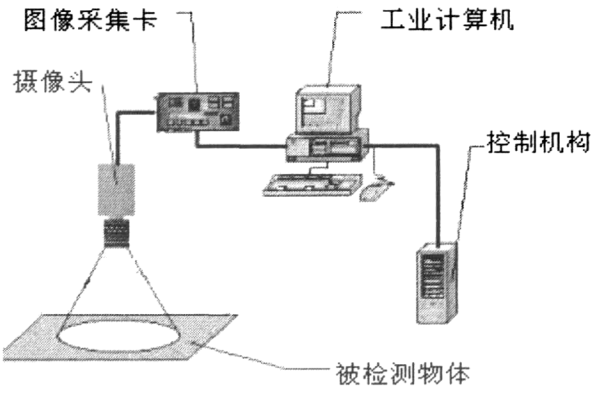


图 1-1 工业机器视觉系统

Figure1-1 Industrial Machine Vision System

上述基于视觉的系统代替了人眼的功能，是对人眼的功能模拟，有着比人眼更高的精度和速度[3]。机器视觉在探测不可视物体和高危险场景时，更具有其突出的优点。机器视觉技术现已得到广泛的应用。下面一一列举说明：

机器视觉在工业检测领域的应用：机器视觉在工业检测领域应用，大幅度地提高了产品的质量和可靠性，保证了生产的速度。例如产品包装印刷质量的检测、半导体集成电路封装质量检测、卷钢质量检测和水果分级检测等。如图 1-2 所示的是在芯片安装到印刷电路板上之前，检测芯片的管脚是否符合要求。

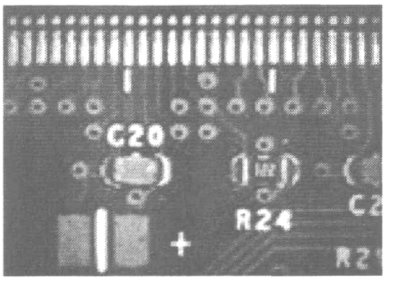


图 1-2 PCB 检测

Figure1-2 PCB Test

机器视觉在机器人导航及视觉伺服系统的应用：赋予机器人视觉是机器人研究的重点之一，其目的是要通过图像定位、图像理解，向机器人运动控制系统反馈目标或自身的状态与位置信息[4]。如图 1-3 所示:

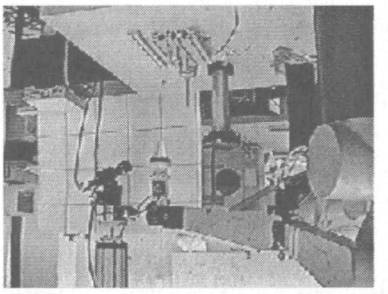


图 1-3 机器视觉伺服系统

Figure1-3 Serve System Based on Machine Vision

图中的摄像机被固定在云台下，一个机械手在一定范围内抓取和移动工件，摄像机利用动态图像识别与跟踪算法，跟踪被移动工件，始终保持其处于视野的正中位置。

以机器视觉为导向的机械手技术：上世纪 80 年代初，在实验室开发的“拣选”系统无法转化为工厂的实际应用。不过，现在有越来越多的迹象表明以机器视觉为导向的机械手“拣选”越来越接近现实。如图 1-4 所示：

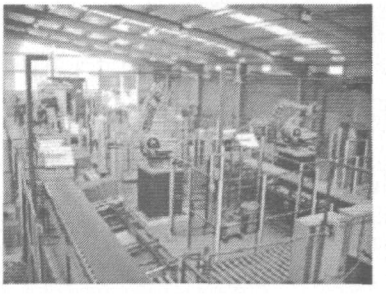


图 1-4 机器视觉机械手

Figure1-4 Manipulator Based on Machine Vision

图中是丰田汽车制造厂已经在其发动机零件加工线上使用了5个机械手的“拣选”系统。该系统采用ABB公司的机械手，配有三维图像技术，大大提高了工作效率。

在国外，机器视觉在工业、农业、国防领域获得了广泛的应用；而在国内，机器视觉本身就属于新兴的领域，加之产品技术的普及不够，该领域在各行业的应用才刚刚起步，随着的制造业发展，国内对于机器视觉的需求将呈上升趋势。

1.3 本课题意义

本课题的主要目标是将植物幼苗所呈现的绿色从图像中辨别出来，并确定其在图像中所处的实际位置，微处理器根据该位置坐标控制电机动作。

传统的机器视觉方案如图1-5所示[5]：这种图像控制系统的成本高，组成的系统庞大，要求运行中必须配备工业计算机，难以适合嵌入式的应用。

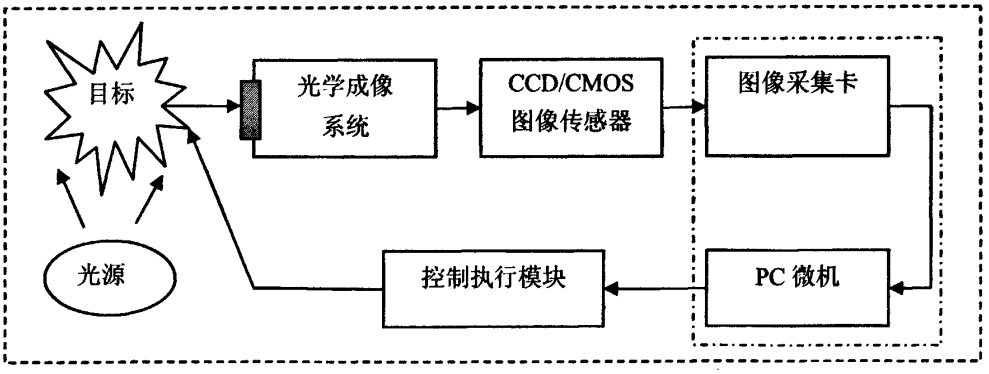


图1-5 传统图像控制系统

Figure1-5 Traditional Imaging Control System

随着半导体技术的飞速发展，特别是具备高速运算能力的DSP、FPGA的出现[6][7]，基于嵌入式的产品将取代板卡式产品。甚至在单个芯片上完成采集、处理、决策的任务，替代了传统视觉控制系统中的图像采集卡和工业PC机，使机器视觉朝着小型化、低功耗、低成本的方向发展。如图1-6所示：此图像控制系统在单芯片上实现图像采集、图像处理的功能，甚至可以将控制决策部分也集成在内。这

样大大降低了成本，真正实现了一个片上系统（SOC）[8]。因此，对嵌入式机器视觉方案的研究具有重要的现实意义[9]。

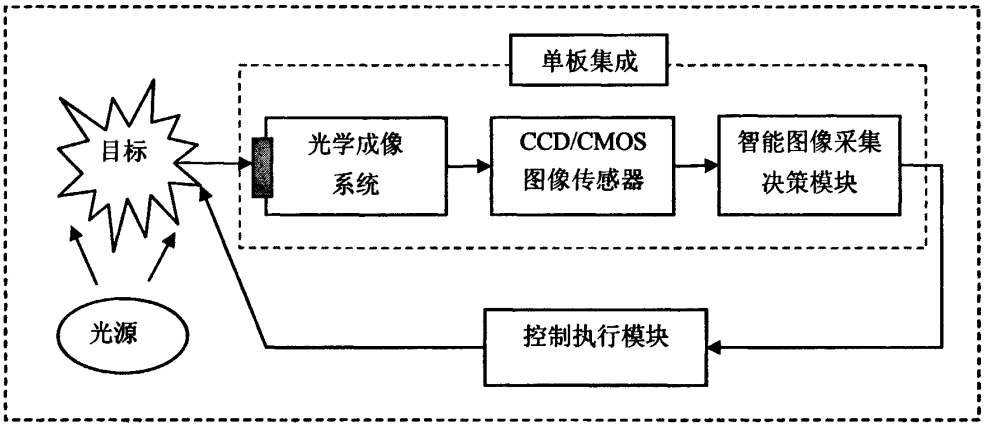


图 1-6 嵌入式图像控制系统

Figure1-6 Embeded imaging Control System

第2章 SOC技术

2.1 设计重用与SOC设计

SOC（System on Chip）是指在单个芯片上实现信号采集、转换、存储、处理和I/O等功能，从而实现一个系统的功能[10]。SOC是在ASIC的基础上发展起来的，具有很多不同于ASIC的独特的优点。SOC实现了一个系统的功能，速度快，集成度高，功耗低。同时由于SOC集成多个功能，使整机成本和体积大大降低，加快了整机系统更新换代的速度。SOC的这些优点正好顺应了通讯产品、消费类、工业控制类电子产品向体积小、重量轻、低功耗发展的趋势。

当SOC的概念盛行的时候，许多专用芯片公司纷纷把微处理器内核嵌入到自己的ASIC中，构建自己的片上系统。其中用户最多的要算ARM这种处理器内核[11]。ARM不仅提供了嵌入式CPU，同时还提供了SOC的解决方案，包括内部总线，外设等。两大PLD供应商Altera和Xilinx也分别把ARM和PowerPC硬核嵌入到自己的FPGA中，同时研发了外围与内核均可灵活定制的软核。这种模块可定制的片上系统在客户中取得了不错的口碑，并被称之为SOPC[12][13]（System on Programmable Chip）。

随着芯片设计进入SOC时代，利用IP内核进行复用变得日益重要。IP复用（IP Reuse）是指在集成电路设计过程中，通过继承、共享或购买所需的智力知识产权内核，然后利用EDA工具进行设计、综合与验证[14]。它可以加速芯片设计过程，降低开发风险。IP Reuse已逐渐成为现代集成电路设计的重要手段。可重用IP核大量应用在SOC的设计之时，基于应用需求、规范协议和行业标准的不同，IP核的内容也是千差万别的。除了购买现有的IP资源外，设计者可能还需要自己进行IP设计，为了使IP核易于访问和集成，并具有良好的复用性，其设计必须严格按照“设计复用方法学（Reuse Methodology）”的要求，按照一定的规范和准则进行设计。

2.2 软硬件协同设计

SOC 是微电子设计领域的一场革命，SOC 主要有 3 个关键支持技术[15]：

- 软、硬件的协同设计技术，面向不同系统的软件和硬件的功能划分理论
- IP 模块库技术
- 模块界面间的综合分析技术

其中软硬件协同设计技术不仅是 SOC 的重要特点，也是 21 世纪 IT 业发展一大趋势。软件硬件协同设计的流程如图 2-1 所示：第一步，用 HDL 语言和 C 语言进行系统描述并进行模拟仿真与功能验证；第二步，对软硬件进行功能划分，然后分别用程序语言进行设计，再综合起来进行功能验证；第三步，进行软件和硬件详细设计；第四步，最后进行系统测试。

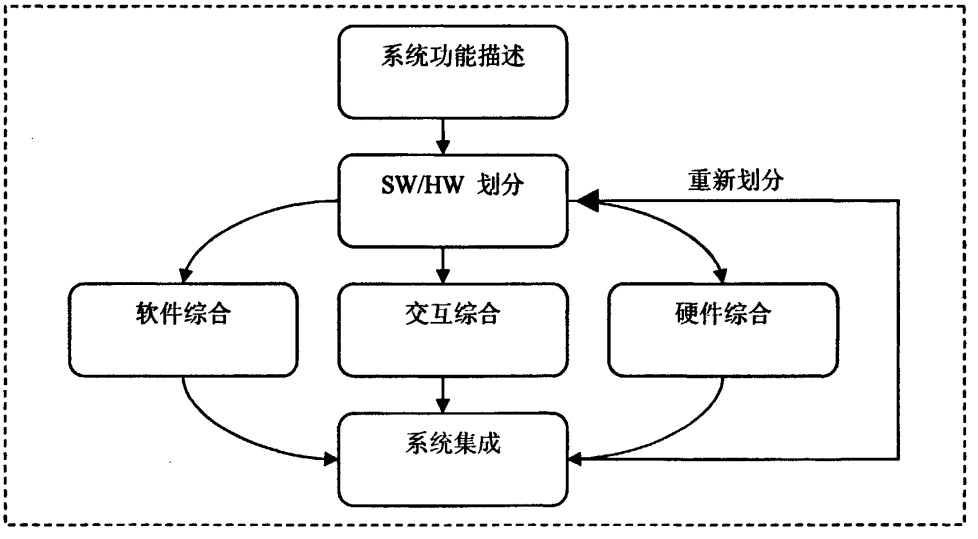


图 2-1 软硬件协同设计流程图

Figure2-1 HW-SW Co-design Diagram

软硬件划分是 SOC 设计中的一个重要课题。其基本原则遵循高速、低功耗由硬件实现；多品种、小批量由软件实现；处理器和专用硬件并用以提高处理速度和降低功耗。划分的方法应该从两个方面着手：1）面向 SW：从 SW 到 HW 满足时序要求；2）面向 HW：从 HW 到 SW 降低成本[16]。

2.3 嵌入式微处理器IP核

在SOPC中最核心的部分就是嵌入式微处理器IP核的设计和使用，其中比较著名的有Altera公司的NiosII和Xilinx公司的MicroBlaze等[17][18]。下面对这两种著名的微处理器软核进行简单的介绍。

1) NiosII

NiosII处理器是一个通用的32位RISC处理器内核，支持Avalon总线，各个模块可以方便实现互联。其内核结构如图2-2所示，其主要特点如下：

- 32位指令集，独立的数据空间与地址空间，可配置的指令和数据Cache
- 32个通用寄存器，32个优良的外部中断源
- 单指令32*32乘除法，产生32位结果，专用指令用来计算64位或128位乘积
- 可以访问多种片上外设，可以和片内存储器和外设接口
- 具有硬件协助的调试模块，可以使处理器在IDE中做各种调试工作

NiosII处理器内核有3种类型，用来满足不同设计的要求。它们分别是快速型、经济型和标准型。快速型NiosII内核具有最高的性能，性能相当于ARM9；经济型NiosII内核具有最低的资源占用，性能相当于51单片机；而标准型NiosII在性能和面积之间做了一个平衡，性能相当于ARM7。

2) MicroBlaze

Xilinx公司的MicroBlaze处理器核是业界最快的软处理解决方案，支持CoreConnect总线的标准，为MicroBlaze设计人员提供了兼容性和重用的方便性。MicroBlaze处理器运行在150MHz时钟下，可提供125 D-MIPS的性能，非常适合设计针对网络、电信、数据通信、嵌入式和消费市场的复杂系统。

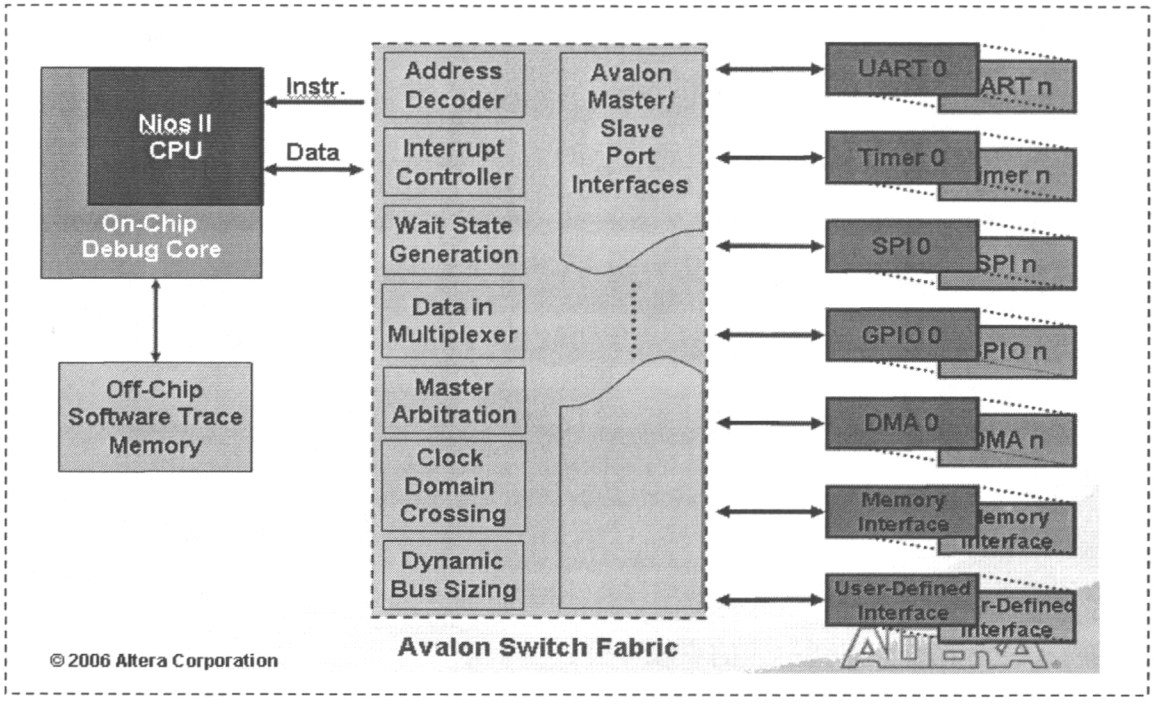


图 2-2 NIOS-II 结构

Figure2-2 NIOS-II Structure

综合上述特点，考虑到在获取开发软件上由于 Altera 公司提供了 QuartusII 7.1、NiosII EDS 7.1、DSPBuilder 7.1、Windows IP 核和其在国内优良的服务。选用 Altera 的 FPGA 作为核心数据采集处理芯片，选用 NiosII 微处理器作为嵌入式 IP 核。而且在后续的实验中，再次证明了 Altera 公司的软件在易用上确实优于 Xilinx 公司的软件。

第3章 CCD与CMOS图像传感器

3.1 CCD与CMOS图像传感器结构

CCD图像传感器最基本的单元是MOS（金属-氧化物半导体）电容器，这些电容器用同一半导体衬底制成，衬底上面生长均匀、连续的氧化层。各个金属化电极互相绝缘，保证电荷能够顺利转移。CCD图像传感器半导体绝缘表面上紧密排列着许多电容器，它可以用来储存和转移以电荷包形势出现的信号。硅半导体有光电效应，当它受光照射时，产生的自由电子与光强成正比，故由许多光敏单元组成的CCD可作为传感器元件。CCD感光元件的表面为透光部分，位移寄存器的表面为遮光部分。每一光敏单元产生的电荷正比于光强和光积分时间[19][20]。

CMOS图像传感器由光敏元阵列及辅助电路构成，光敏元阵列是由光电二极管和MOS场效应管阵列构成的集成电路，每个光敏元内集成有一个放大器，如图3-1中右边所示。CMOS图像传感器由像素感光阵列、行选通逻辑、列选通逻辑、定时和控制电路、模/数转换器（ADC）以及数字处理电路和接口电路构成，CMOS图像传感器的时序电路主要产生各种驱动与控制脉冲；模拟信号处理电路集成了自动增益控制、自动曝光控制、自动白平衡等功能，片上功能可通过I^2C接口控制。实际上CMOS图像传感器是一个较完整的单片图像系统（Camera on Chip），如图3-2所示。

3.2 CCD与CMOS图像传感器比较

由图3-1和图3-2的比较可以看到CCD与CMOS图像传感器的不同之处在于：CCD作为目前主要的实用化固态图像传感器，具有噪声低、动态范围大、响应灵敏高，以及成像质量好等优点。但由于CCD图像传感器中的定时产生、驱动放大、自动曝光控制、模数转换以及信号处理等电路很难与像素阵列集成在同一芯片上，以CCD为基础的图像传感器难以实现单片一体化，具有体积大、功耗高等缺点。而CMOS图像传感器作为近几年发展较快的新型图像传感器，由于采用了CMOS

技术，可以将像素阵列与外围电路集成在同一块芯片上，因而在速度、成本、集成度、体积、功耗以及电路结构设计等方面都有着比较大的优势。随着技术的发展，CMOS 图像传感器有赶超 CCD 之势[21][22][23]。鉴于以上优势，本设计图像采集部分选用 CMOS 图像传感器。

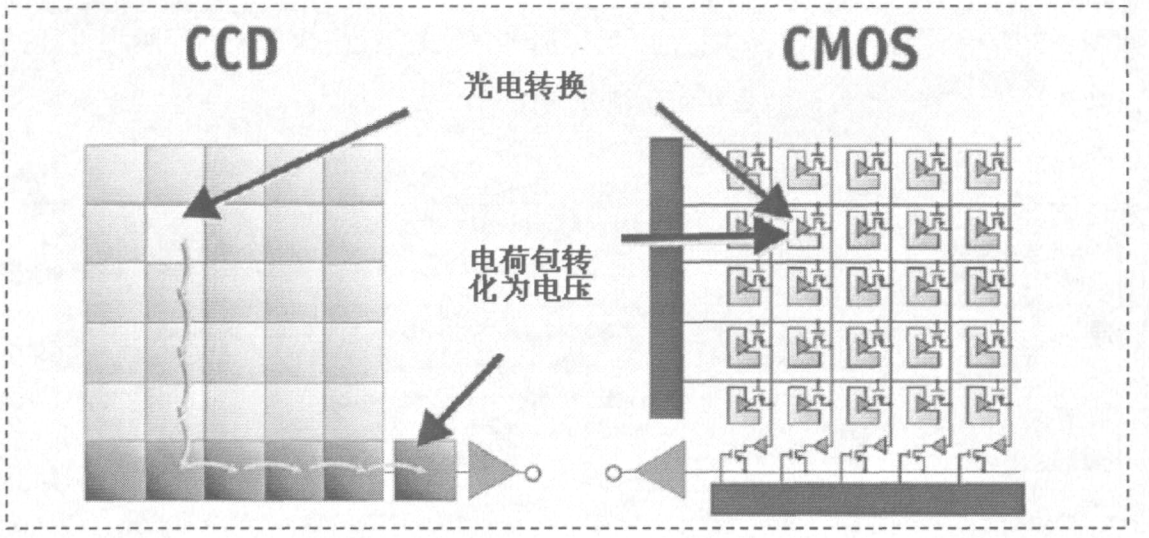


图 3-1 CMOS 与 CCD 结构差异

Figure3-1 Architecture Difference between CMOS & CCD

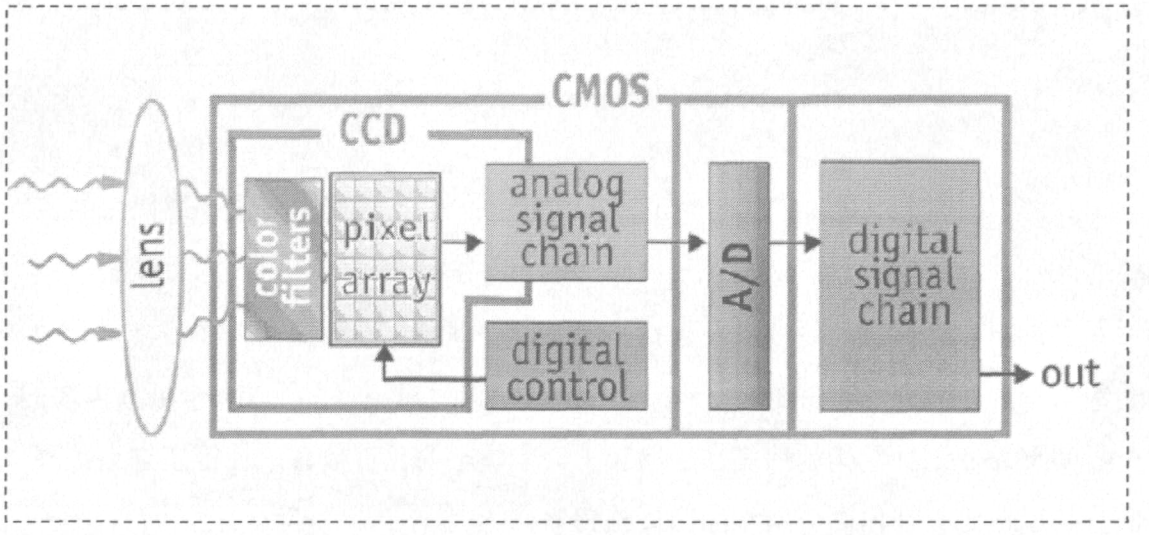


图 3-2 CMOS 与 CCD 相机结构对比

Figure3-2 Camera Architecture Comparison Using CMOS&CCD

3.3 电子曝光方式

目前，CMOS 图像传感器有两种电子曝光方式[24]，分别对应两种不同的快门。

1）电子卷帘快门（Electronic Rolling Shutter）。大多数 CMOS 传感器采用这种快门。对任一像素，在曝光开始时将其清零，等待曝光时间过后，将信号值读出。如图 3-3 所示，数据的读出是串行的，所以清零、曝光、读出也只能逐行顺序进行，通常是从上至下，和机械的焦平面快门非常像。

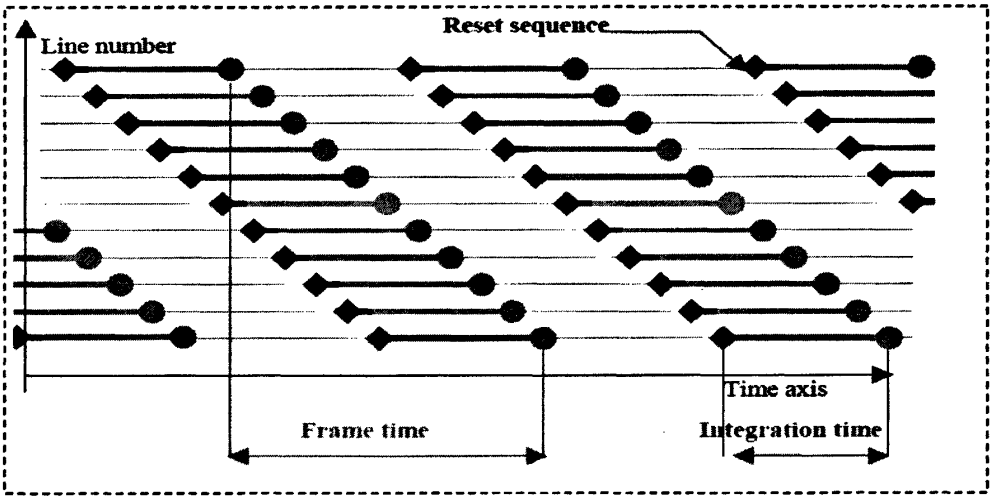


图 3-3 电子卷帘工作时序

Figure3-3 Electronic Rolling Operation Timing

2）全局快门（Global Shutter/Snapshot Shutter）。该快门与电子卷帘快门的主要的区别是：每个像素点增加了采样保持单元，在指定时间内对数据进行采样然后顺序读出，这样虽然后读出的像素仍然在进行曝光，但存储在采样保持单元中的数据却并未改变。如图 3-4 所示，因图像的积分时间相等，所以每个像素点在同一瞬间曝光。这种结构的瑕疵在于增加了每个像素的元件数目，使得填充系数降低，所以很难设计出高分辨率的图像传感器。

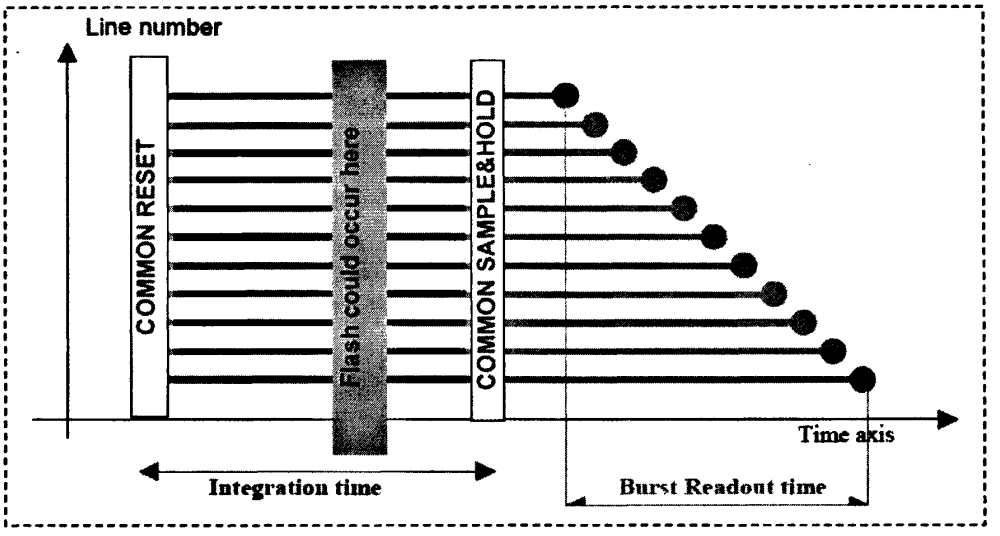


图 3-4 全局快门工作时序

Figure3-4 Global/snapshot Shutter Operation Timing

目前只有 Micron 和 Cypress 生产具有全局快门的 CMOS 传感器[25][26]，主要用于机器视觉和超高速摄影，最高分辨率为 400 万像素，MT9V032 就是 Mircon 基于全局快门的图像传感器。采用全局快门的相机即使在物体高速运动时也不会产生拖影与畸变。如图 3-5 所示，与一般的图像传感器相比，具备全局快门的图像传感器能够拍到运动物体的“静态”效果。

本课题要求在运动的过程中，能拍到清晰的图像，并且没有拖影和畸变，正是由于 MT9V032 这样的特性，因而采用此 CMOS 图像传感器作为图像采集芯片。下面对 MT9V032 的特性作简单介绍。

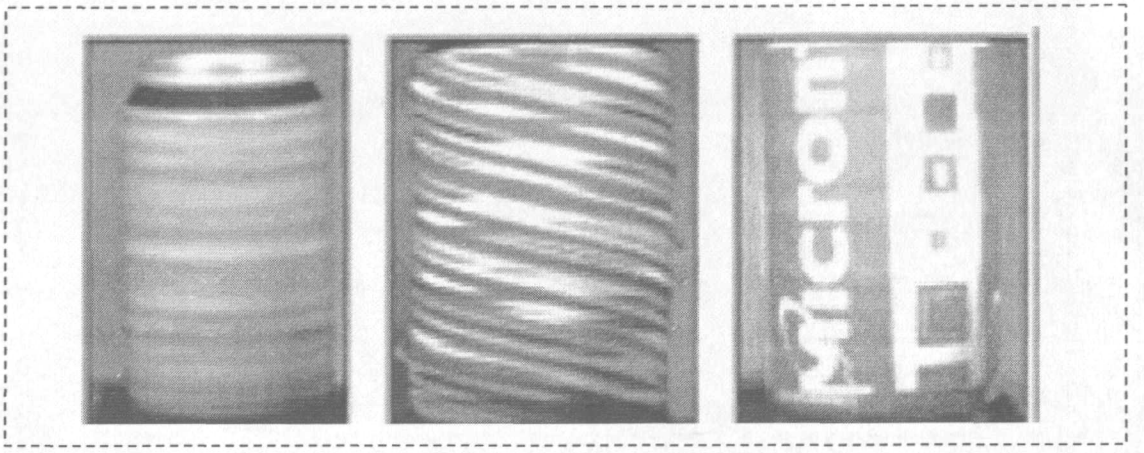


图 3-5 全局快门特点对比

Figure3-5 Characteristics Comparison of Global/snapshot Shutter

3.4 MT9V032 介绍

Micron 的 1/3 英寸、宽 VGA 图像传感器 MT9V032 能让相机在同一地点看清黑暗和明亮区域。大多数图像传感器的动态范围在 45db 左右，而这款图像传感器动态范围达到 100db，接近红外的敏感度[27]。其主要特点如下：

- DigitalClarity™CMOS 图像技术
- 宽屏 VGA 752H*480V 输出
- 1/3 英寸光学格式
- 6um*6um 像素尺寸
- 可供选择彩色和单色模式
- 逐行扫描，隔行扫描模式
- 全局快门光电像素；瞬间积分读出
- 二线串行接口
- 片内 10 位模拟-数字转换（ADC）
- 四个独立串行控制寄存器 ID 支持多传感器共用总线

第 4 章 系统硬件设计

4.1 硬件设计原理

为实现本机器视觉系统中的软硬件功能，设计如下 FPGA 采集控制板。在充分考虑性价比和设计功能的基础上，决定采用 Altera 公司 Cyclone 系列的 FPGA[28]，它是 Altera 公司一款低成本 FPGA，主要定位在终端市场，如消费类电子、计算机、工业和汽车等领域。该系列产品中的 EP1C6T144C8 芯片包括 5980 个逻辑单元（LE）、20 个 M4K RAM、2 个输入锁相环、98 个可编程 IO 口，能满足本设计要求。

开发板要完成的任务具体包括：

1）FPGA 控制 CMOS 图像传感器 MT9V032 的图像采集处理

2）FPGA 对电机位置反馈信号的采集

3）FPGA 对电机的控制

4）FPGA 与 PC 机之间的通讯

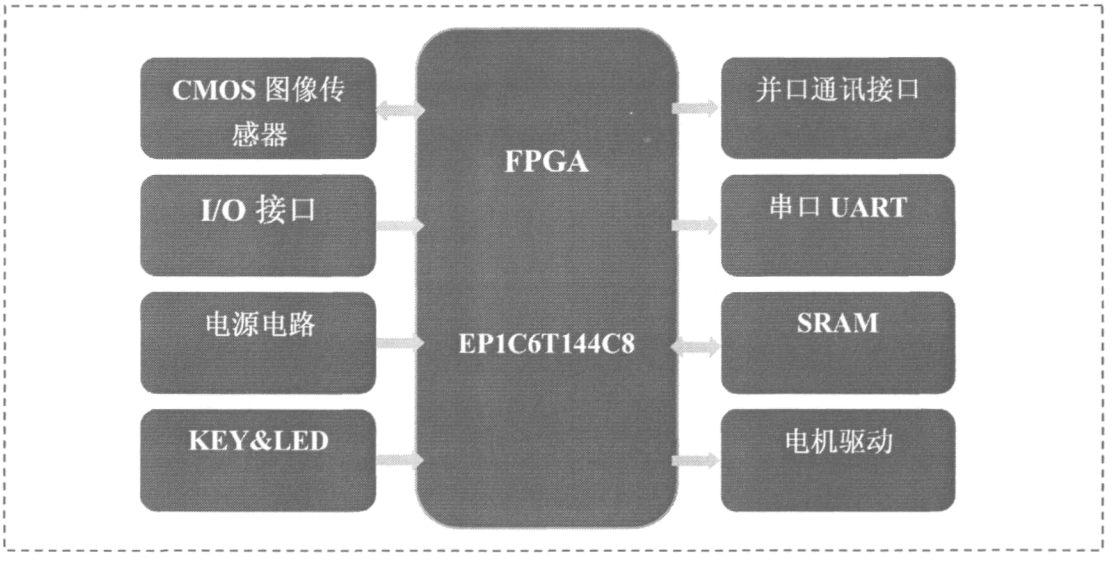


图 4-1 系统功能图

Figure4-1 System Function Diagram

如图 4-1 所示，系统以 EP1C6T144C8 为控制核心，实现 CMOS 图像传感器与 FPGA、FPGA 与 PC 机、FPGA 与电机控制部分的数据通讯。功能的实现过程如下：当 FPGA 接收到同步信号，开始一帧图像采集，其内部图像处理单元对数据进行处理。FPGA 根据处理结果给电机一个启动脉冲，将接收到的电机位置反馈信号与处理结果进行比较，适时地给出停止信号，完成一次控制。

图 4-2 描述的是数据在 FPGA 控制模块与其它模块之间的流动状况。其中，I^2C 总线信号负责对 MT9V032 的工作方式进行设置，UART 串口信号用于实时程序调试，位置反馈信号用于电机位置的检测。

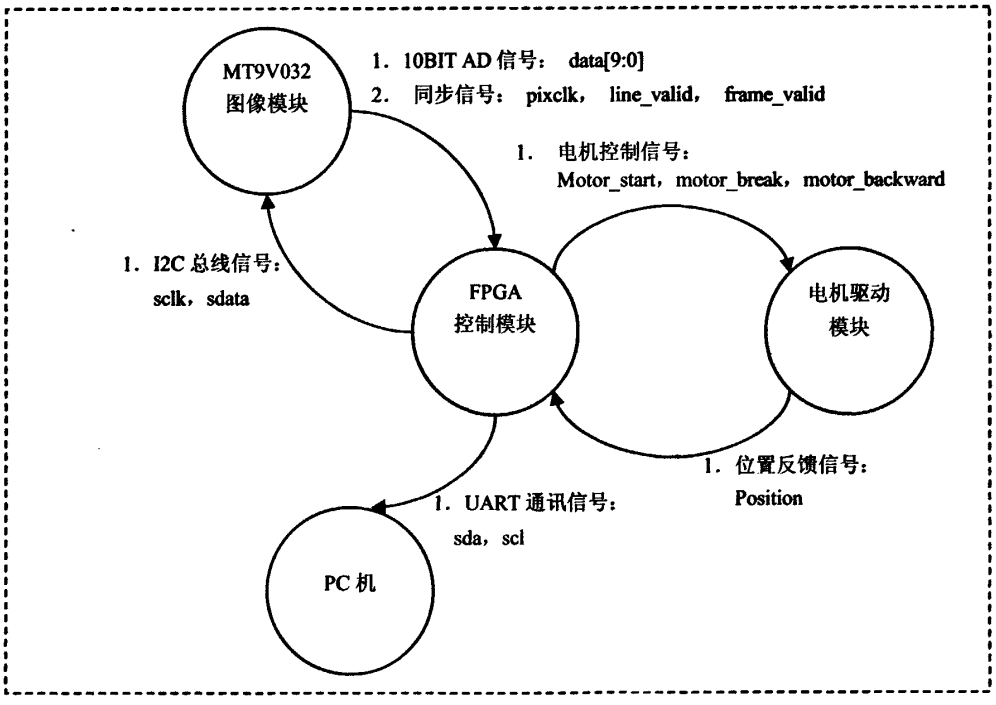


图 4-2 数据流动图

Figure4-2 Illustration of \Data Flow

4.2 模块结构描述

整个电路板可以分为以下 6 个模块：

4.2.1 电源供给模块

图 4-3 为电源模块结构图，系统要求有 3.3V，1.5V 的电压输入。表 4-1 中给出了核心器件的工作电流和预估最大工作电流，加上其它器件，最大工作电流 I 预计在 500mA 左右，为留两倍的电流裕量，电源至少能提供 1.5A 的电流。这里选用凌特公司[29]的线性电压调节器 LT1086-3.3 将 5V 电压转化为 3.3V，其电流输出能力达 1.5A；选用 AMS1117-1.5 将 5V 电压转化为 1.5V，其电流输出能力达 1A，满足系统对电源的要求。图 4-4 是 LT1086 的典型应用图，10uF 的电容用于电源滤波。

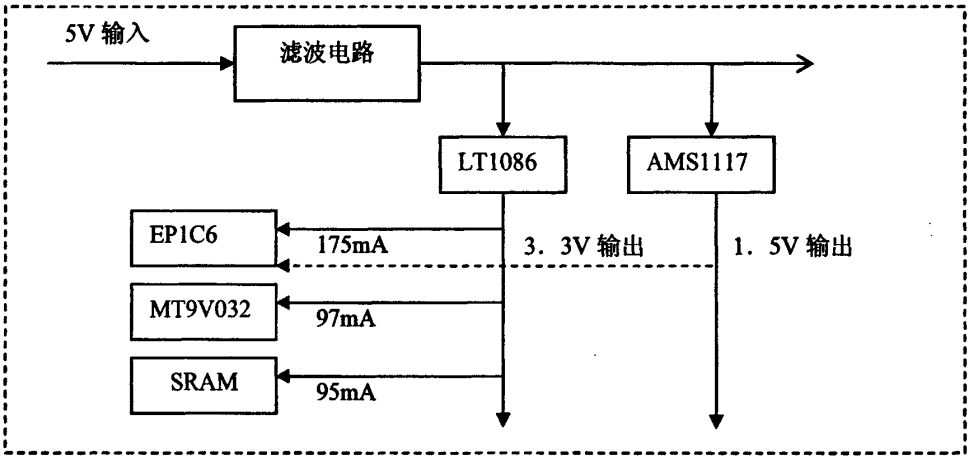


图 4-3 电源模块结构图

Figure4-3 Power Part Diagram

表 4-1 工作电流

Table4-1 Working Current

器件	工作电压	典型工作电流	最大工作电流
EP1C6	3．3V	175mA	200mA
MT9V032	3．3V	97mA	100mA
SRAM	3．3V	95mA	100mA

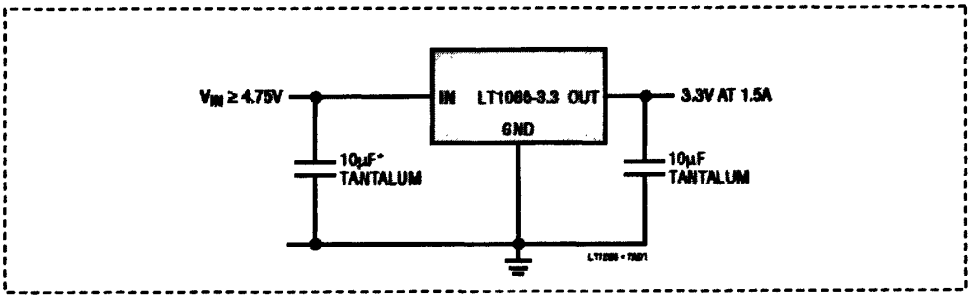


图 4-4 LT1086 典型应用图

Figure4-4 Typical Configuration of LT1086

4.2.2 图像传感器模块

本模块采用 Mircon 公司新一代基于 DigitalClarity™CMOS 图像技术和全局快门（Global Shutter）技术的专用机器视觉产品 MT9V032。其典型应用如图 4-5 所示，MT9V032 由有源晶振提供 27M 时钟输入，FPGA 通过 I^2C 总线对图像传感器的工作方式进行设置，图像数据通过 Dout[9:0]数据端口，在时序同步信号 Pixclk、Line_valid、Frame_valid 的控制下，传送给 FPGA。

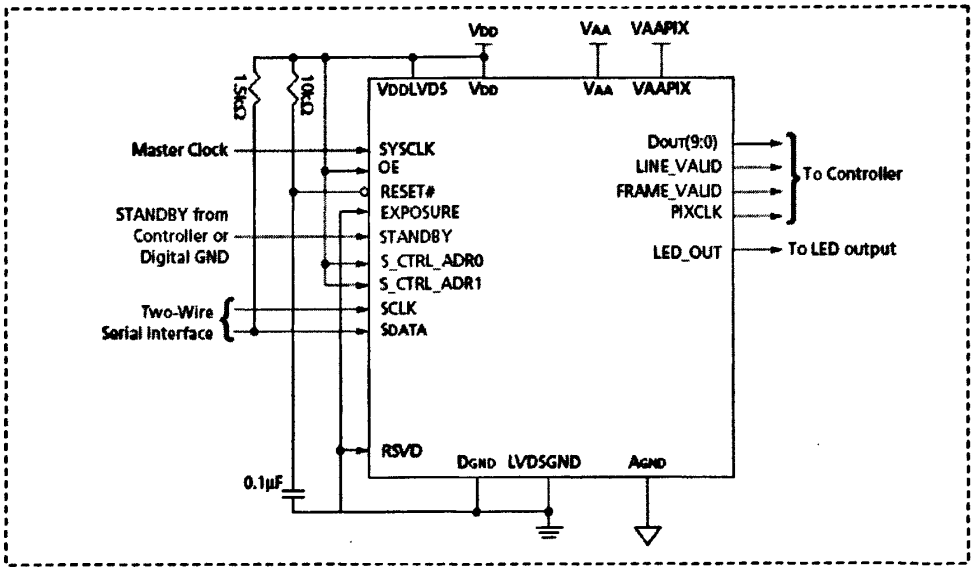


图 4-5 MT9V032 典型应用图

Figure4-5 Typical Configuration of MT9V032

4.2.3 存储器模块

存储器模块扩展了一片 512KByte SRAM 用于图像存储，图像的分辨率为 752*480，对应存储空间大小为 360960 字节，故 512KByte 的容量已经能满足系统要求。这里采用 ISSI 公司的 512K*8Bit 的 IS61LV5128AL[30][31]SRAM，它是一款高速 CMOS 静态 RAM，其数据访问时间高达 10ns。存储器结构如图 4-6 所示，地址线为 19 根，数据宽度为 8 位，对应的容量为 512K。

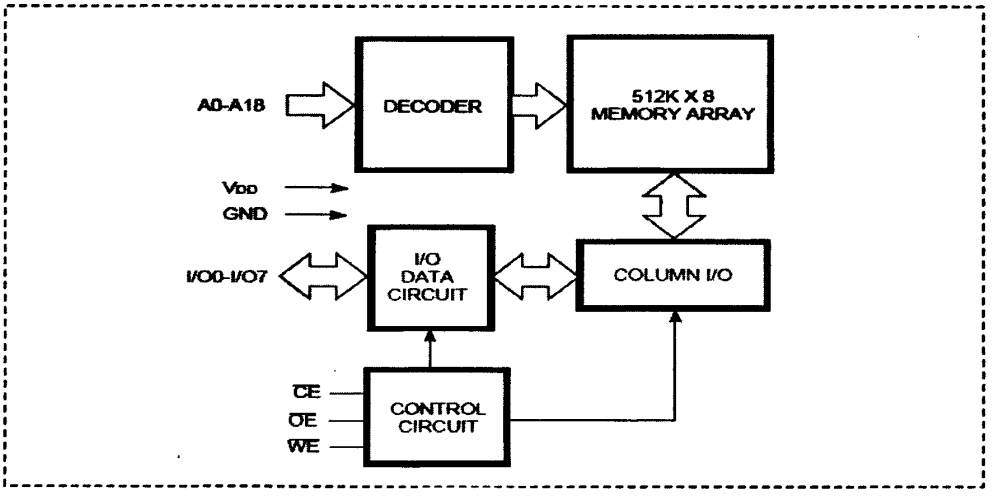


图 4-6 SRAM 结构图

Figure4-6 Architecture of SRAM

4.2.4 电机驱动模块

项目的电机驱动部分已有现成的模块，原理是采用一个专用集成芯片 L6235 控制直流无刷电机[32]。现简要说明该集成芯片的特点：

- 霍尔传感器输入，内置解码逻辑
- 内置的固定间歇（off）时间 PWM 电流控制
- 频率/电压变换器
- 工作频率达到 100KHZ
- 高侧开关管过电流保护

- 交叉导通保护
- 过热关机保护
- CMOS/TTL 输入的正/反转控制
- 内置快速恢复二极管
- 欠电压保护

其典型应用如图 4-7 所示。其中，FPGA 控制 L6235 的 ENABLE、FWD/REV、BRAKE 三个信号端，从而控制电机的停止，正转，反转。为了减小电机驱动部分对 FPGA 信号的干扰，将 FPGA 控制板和驱动板分开设计。

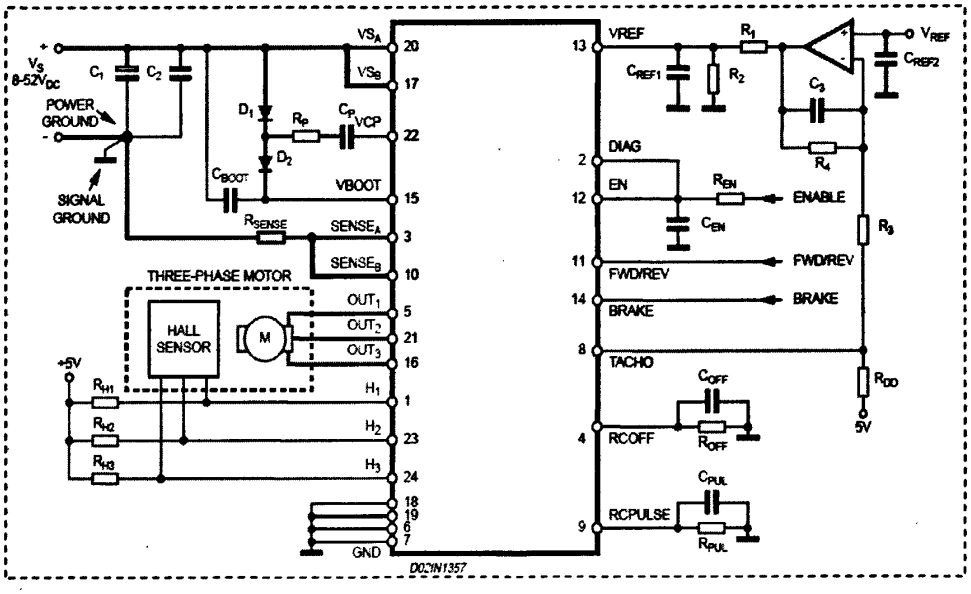


图 4-7 L6235 典型应用图

Figure4-7 Typical Configuration of L6235

4. 2. 5 串口通讯模块

FPGA 的电压输出及板级工作电压均为 3.3V，故要采用 3.3V 电压兼容的串口通讯芯片。这里采用 MAXIM 公司的通用串口芯片 MAX3232。图 4-8 为典型应用图。

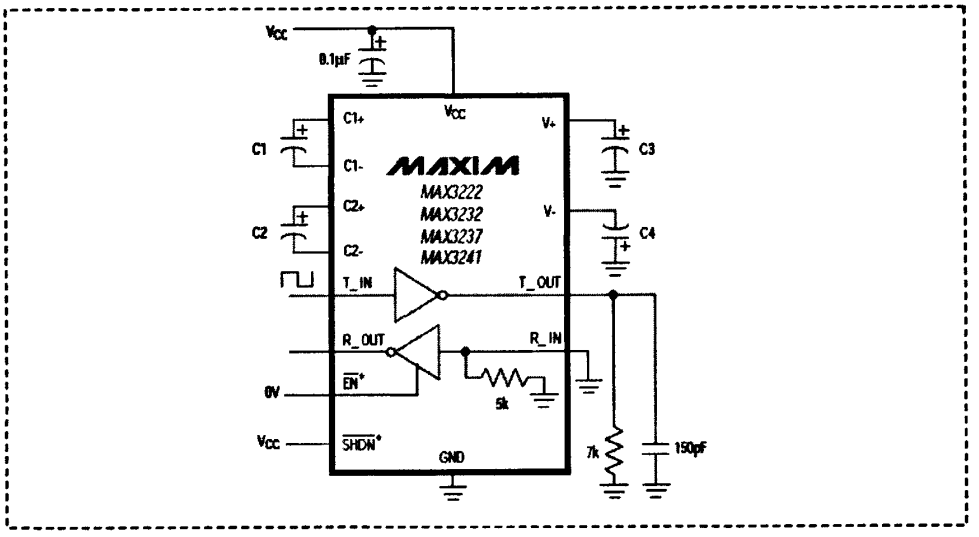


图 4-8 MAX3232 应用图

Figure4-8 Typical Configuration of MAX3232

4.2.6 FPGA 核心控制模块

EP1C6T144C8 引脚定义及与外围器件的连接方式如表 4-2 所示。为了增加电路调试的灵活性，SRAM、串口模块、并口通讯接口、CMOS 图像传感器模块均可选择性使用。

表 4-2 引脚锁定

Table4-2 Pin Lock

器件引脚	FPGA 引脚
SRAM 引脚接口定义	
A0	142
A1	141
A2	140
A3	139
A4	134
A5	125
A6	124

A7	123
A8	122
A9	121
A10	120
A11	119
A12	114
A13	113
A14	112
A15	106
A16	105
A17	104
A18	103
IO0	132
IO1	131
IO2	130
IO3	129
IO4	111
IO5	110
IO6	109
IO7	108
nWE	128
nCE	133
nOE	107
MT9V032 引脚接口定义	
D0	28
D1	31
D2	32
D3	33
D4	11
D5	10
D6	7
D7	6
D8	5

D9	4
LINE_VALID	3
FRAME_VALID	2
PCLK	16
并口引脚接口定义	
EPP_WAIT	100
EPP_AS	99
EPP_RST	98
EPP_DS	97
EPP_nWE	96
EPP_DIR	85
EPP_D0	84
EPP_D1	83
EPP_D2	82
EPP_D3	79
EPP_D4	78
EPP_D5	77
EPP_D6	76
EPP_D7	75

4.3 电磁兼容及抗干扰技术

对于高速数字系统设计，必须考虑电子系统的电磁兼容性[33]。如果设计中不遵循一定的原则，必然会带来干扰，严重时直接影响电路的正常工作。系统设计的问题主要体现在信号完整性、电源完整性和高速系统设计等方面[34][35]，下面就信号完整性和电源完整性内容加以说明。

4.3.1 信号完整性

所谓的信号完整性（Signal Integrity），就是要使信号具有良好的物理特性，防止其产生信号畸变。可以把信号完整性问题归纳成三类：传输线效应；不同信号线之间的串扰；电源地噪声。

1) 传输线效应

在一个信号传输的过程中，如果信号的边沿时间足够快，短于六倍的信号传导延时，那么在信号传输过程中，传输媒介就会表现传输线效应。这样，信号在媒介上传输就像波浪在水中传输一样，会产生波动与反射现象。

信号在传输的过程中，传输线等效一个电阻，这个电阻值称为传输线的特征阻抗（Characteristic Impedance）。如果线路的特征阻抗不连续，就会使得入射信号产生反射现象。导致阻抗不连续的原因很多。因此在设计 PCB 时，必须保持传输线由始至终阻抗尽量连续。

2) 串扰

所谓的串扰（Cross Talk）是指在两个相邻的信号走线之间，其中一条走线上的电流发生改变，同时会在另外一条走线上耦合出一定的噪声电流。如图 4-9 所示，串扰只有在相邻走线上的电流发生改变时才会发生，一般为信号的上升沿或下降沿。

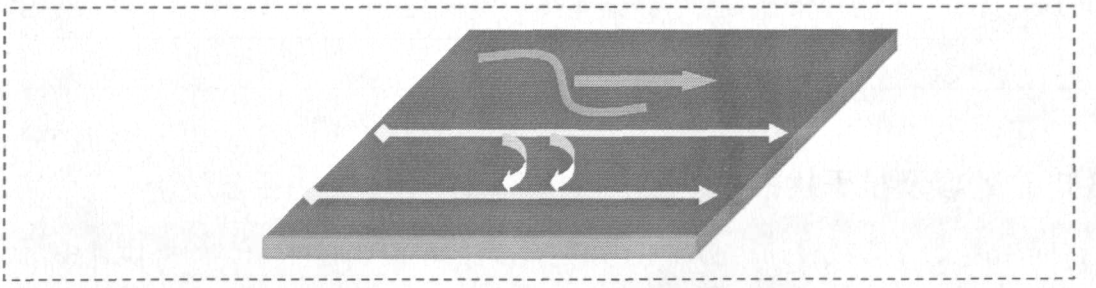


图 4-9 串扰示意图

Figure4-9 Cross Talk Diagram

要减小串扰的影响，方法有很多，从根本一点就是减小信号网络之间的耦合效应。例如，把同一层走线尽量远离一点，在相邻信号层的走线尽量不要平行，以减少信号线之间的耦合。另外，把信号走线与参考平面的间距减小，也有助于减小走线之间的耦合，这样信号线会更多的耦合到参考平面上，而不是相邻的信号线上。

3) 电源地噪声

理想情况下，认为供电电源都是恒定不变的。但是实际并非如此。在系统中的电源和地经常受到外界和内部噪声的干扰，使得电源表现出一定的波动，这种波动又是会严重影响系统的可靠性。设计电源最主要的目的就是使得电源分配系统在相当宽的噪声频率范围内表现出较低的交流（AC）阻抗，使得因各种噪声造成的电源地平面的波动幅度尽量减小，以保持稳定的供电。

减小电源分配系统之间的阻抗有以下办法可以采用：使电源平面和地平面之间的间距尽量减小；尽量采用薄的介质；采用低电感的去耦电容。

4.3.2 电源完整性

由于芯片工艺的不断改进，从 0.35um、0.18um、0.13um 到目前的 90nm 甚至 65nm，芯片的内核电压也在不断降低，从 3.3V、1.8V、1.5V 到 90nm 的 1.2V，芯片对电源的波动越来越敏感。保持电源的完整性，就是保持电源的稳定供电。

由于系统中总是存在着不同频率的噪声，首先我们需要把电源分配系统与外界很好的隔离开来，电源系统与外界主要的连接途径的电压调整模块（VRM），通常我们总是需要使用 T 型或 II 型滤波网络，以阻止低频噪声窜入，同时大电容也提供了一个电荷的蓄水池，以及时提供 VRM 所不能供给的电流；另外，系统内部的一些元件会产生高频的电源噪声，例如：数字逻辑门在翻转的时候会产生翻转噪声，如果 PCB 去耦处理不当，这个电源噪声就会波及整个电源平面。电源分配系统设计的根本目的，就是如何利用电源平面和地平面之间的平面电容，以及分立的各种容值的去耦电容来使得整个电源分配系统在尽可能宽的范围内达到低阻抗。

本人在设计上遵循了上述原则，每个电源与地之间加了 0.1uF 的去耦合电容，并在设计时注意模拟地与数字地的隔离，信号走线从左到右，尽量减小电磁干扰的产生。

4.4 硬件调试结果与分析

图 4-10 所示为本实验所设计的 PCB 板，板上所有芯片由本人自己亲手焊接。其中最难焊的是 144 个引脚的 FPGA 芯片，（每两个引脚之间距离仅约 0.2mm），在掌握焊接技巧后，也被成功焊接。第一次上电时，FPGA 程序即可正常下载。经简单程序测试，FPGA 工作稳定，电路提供的电源质量良好。CMOS 图像传感器在默认工作模式下，像素数据端口有明显的数据变化，示波器测得图像的帧时钟周期为 16ms，行时钟周期为 32us，像素时钟周期为 37ns，与 MT9V032 的默认工作方式相同，如图 4-11 所示。因此，硬件设计在整体上达到要求。系统硬件如图 4-12 所示。

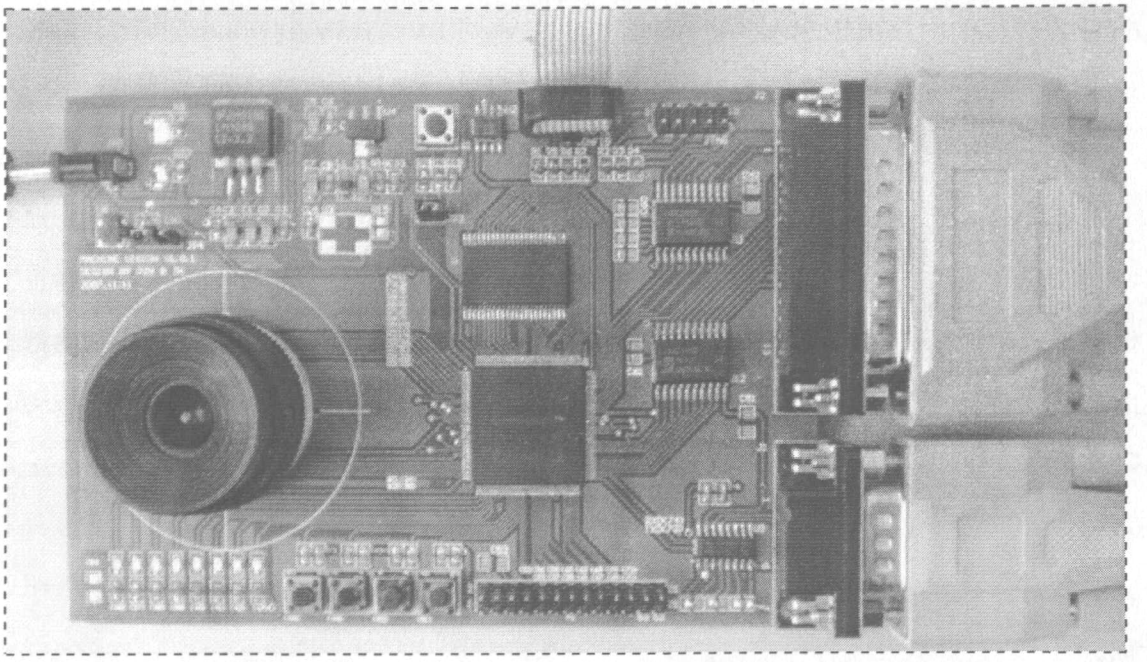


图 4-10 PCB 硬件电路

Figure4-10 PCB Hardware

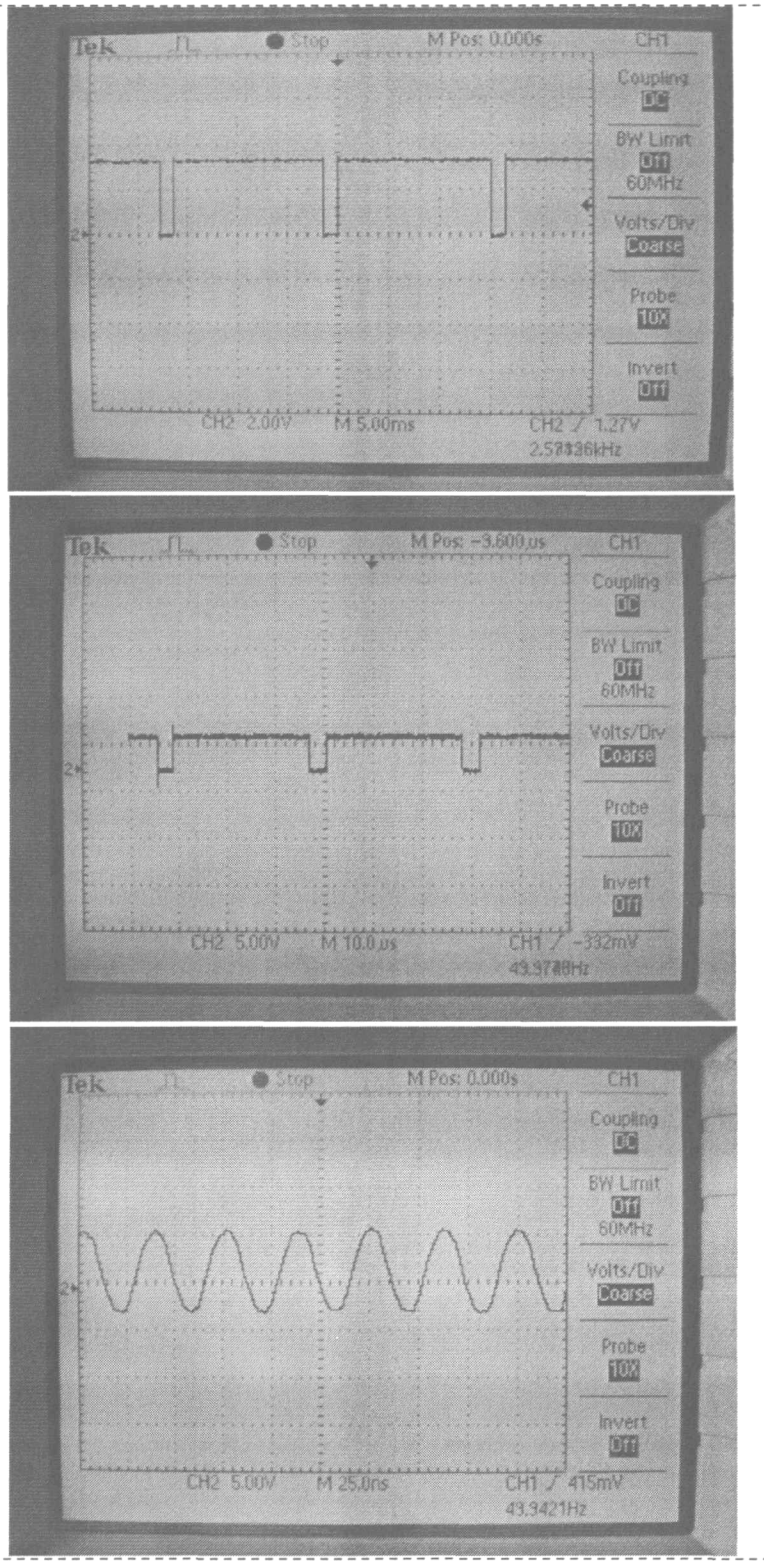


图 4-11 示波器测试波形图

Figure4-11 Testing Wave by Toscillograph

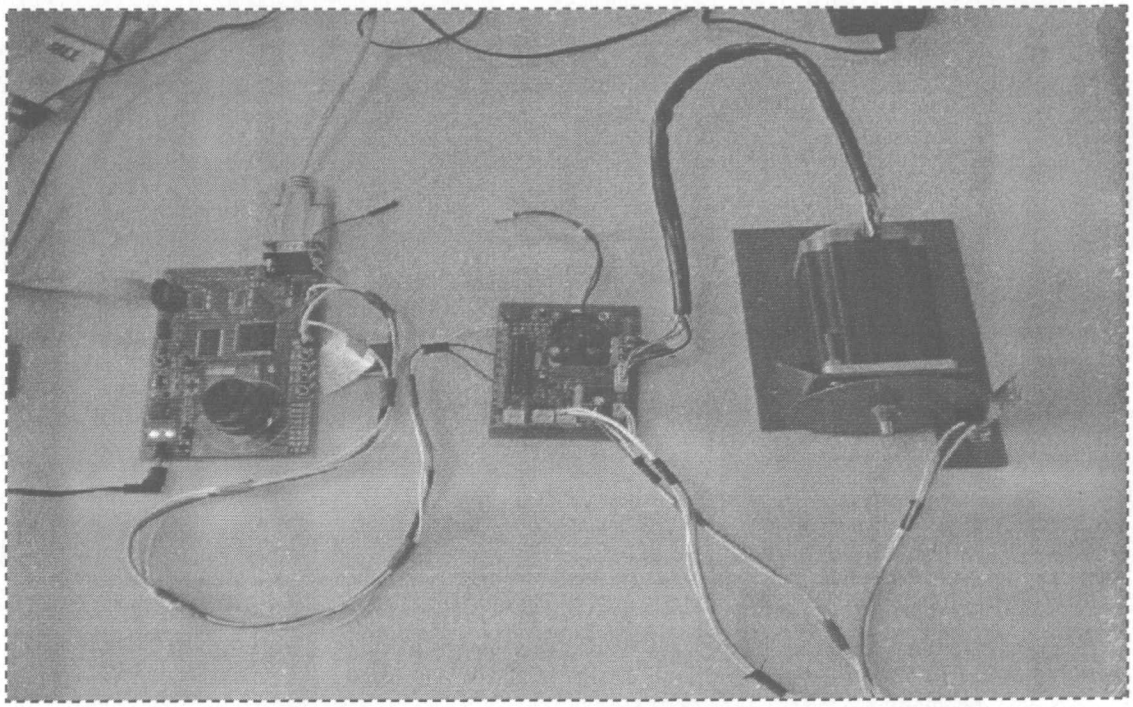


图 4-12 系统硬件图

Figure4-12 System Hardware

第 5 章 系统软件设计

5.1 软件设计原理

5.1.1 任务描述与层次划分

软硬件协同设计在设计之初，需要对系统的任务进行描述，然后考虑各子任务的具体实现方法，对任务进行软硬件划分。本系统需要实现的功能如图 5-1 所示。

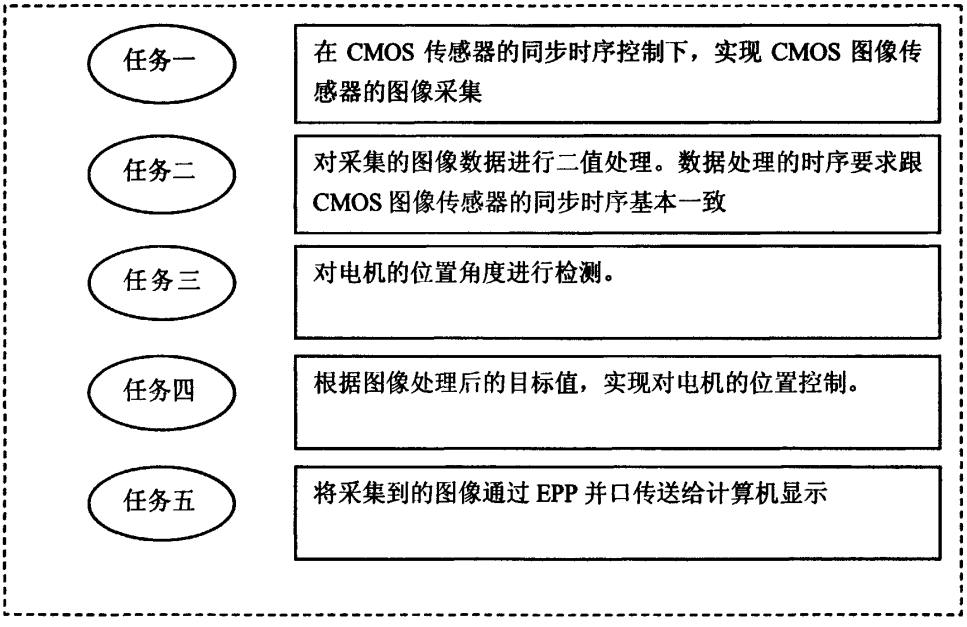


图 5-1 系统任务图

Figure5-1 System Task Diagram

图中，任务一与任务二对时序均有严格的要求，可采用硬件来实现，这也正是 FPGA 所擅长的地方。任务三，任务四对性能没有太大要求，但要求编程灵活多变，可以实时调试，因此可以在 NiosII 软核上实现。这两部分通过 Avalon 总线实现互连，共同完成系统的任务。图 5-2 所示为系统功能分配结果，在这里，图像

处理部分相当于 NiosII 的一个协处理器，可以看到，SOPC 的最大特点，就是可以把一些对性能要求苛刻的算法用硬件电路来实现，提高系统的实时性。

对电机的控制部分已有现成模块，只要在软件上作一下移植即可，因此这部分内容不是本设计的重点，基于篇幅不再做介绍。而图像的采集、目标的识别与定位是本设计的重点和难点。在以下篇幅里将详细介绍图像处理的算法，图像处理的 IP 核设计，IP 模块之间的互联机制和 SOPC 系统生成。在硬件方面采用 VHDL 进行设计[36]。

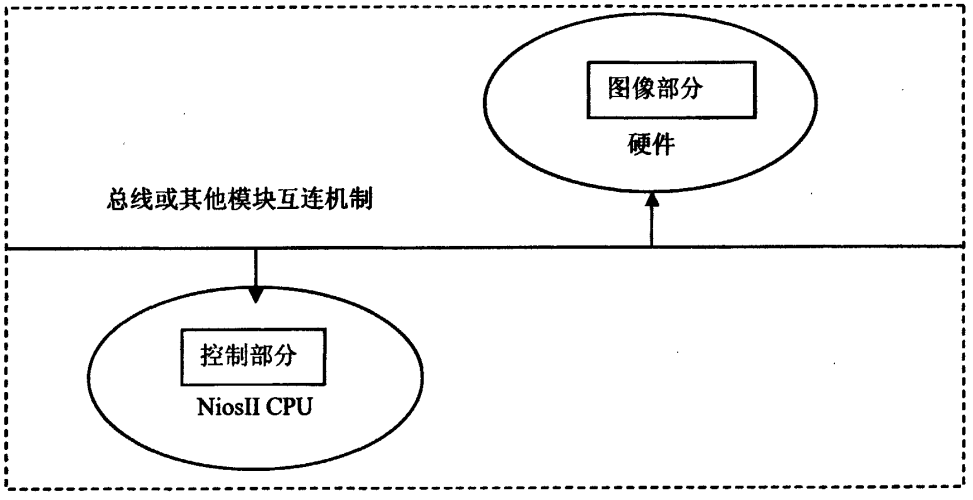


图 5-2 功能分配图

Figure5-2 Function Dispatch Diagram

5.1.2 算法可行性验证

1） Bayer RGB pattern

图像传感器采用一个颜色过滤阵列[37][38]（Color Filter Array）Bayer RGB 采集图像信号。如图 5-3 所示，在这种阵列排列方式下，图像传感器的每个像素点只检测 RGB 三种颜色分量中的一种。所以对应第一行的颜色分量为 BGBGB...第二行的格式是 GRGRGR...这样通过适当的插值算法就能重建出整幅图像。

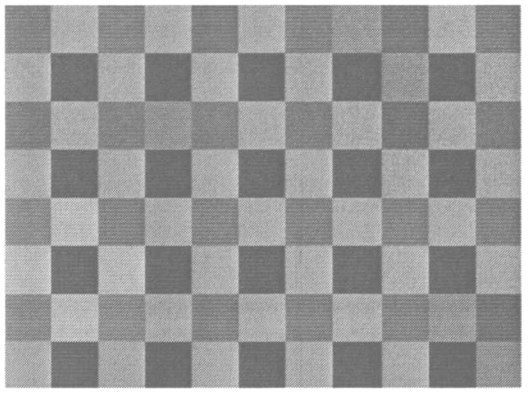


图 5-3 RGB bayer pattern

Figure5-3 RGB Bayer Pattern

本机器视觉系统的图像处理任务是要在自然光照条件下，将农业薄膜中刚发芽的叶子从背景中识别出来。针对农业中目标与背景呈现的颜色差别明显，本文选择性研究了基于（G-B）和（G-R）色差特征的阈值分割方法[39][40]，并进行了对比分析。首先提取 RGB 彩色图像的 R、G、B 分量，并做代数减运算得到（G-B）、（G-R）立体色差图，再对图像进行阈值分割，结果发现，幼苗的（G－B）色差值呈峰状分布。其处理方法的算法描述如下：

If　G－B＞T　then

　Optional Objects

Else

　Backgroud

End if

Matlab 的分析结果如图 5-4 所示，图 A 是薄膜里刚发芽的幼苗，为了模拟 CMOS 图像传感器图像数据产生方式，对原始图像进行减色处理。方法为：第一个像素点去掉绿色（G）分量与红色（R）分量，得到蓝色（B）分量，第二个像素点去掉红色（R）与蓝色（B）分量，得到 G 分量...如此循环就得到第一行的 BGBG...图像数据，同理得到第二行的 GRGR...图像数据。整幅图的原始 BayerPattern 效果如图 B 所示；图 D 是图 B 的放大效果，每个方格代表 RGB 三基色的一种；图 C 是对图 B 进行颜色分割后的图像，可以清晰地看到中间的绿色叶

子被分割出来（彩色模式显示为绿色）；图 E 是图 C 的放大效果；图 F 是颜色分割后的峰状图。仿真结果验证了颜色处理方法的可行性。图像处理 Matlab 仿真程序见附件二。

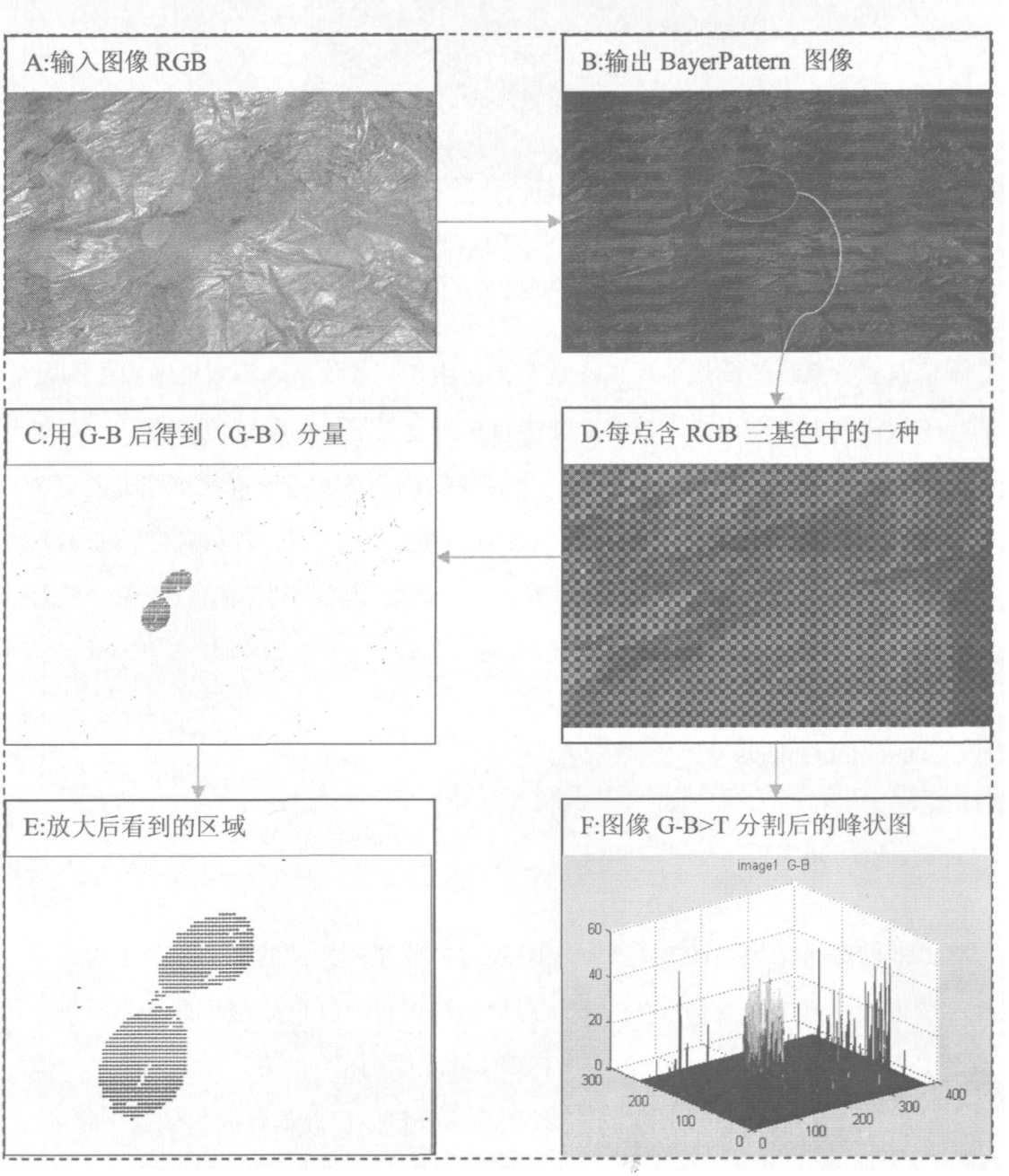


图 5-4 颜色分割图

Figure5-4 Illustration of Color Segmentation

5.2 RTL 描述

5.2.1 图像采集处理 RTL 描述

上一节可行性分析表明：通过（G-B）阈值分割方法，即可以把绿色区域从背景中分割开来。这一节将对算法进行语言描述。Altera 公司提供的 DSP-Builder 开发工具可将 Matlab 程序直接转化为 VHDL 或 Verilog 程序，此方法偏重对算法的研究，忽略硬件实现的细节，由于编译效率低，对 FPGA 容量有较高的要求。为了节约硬件资源，本设计直接采用 VHDL 语言对上述分割方法进行 RTL 描述。如图 5-5 所示，图像采集处理 IP 核划分为图像采集、图像处理、目标定位、位置检测四个子模块，采用至上而下的设计方法，遵循 Avalon 总线协议规范[41]。

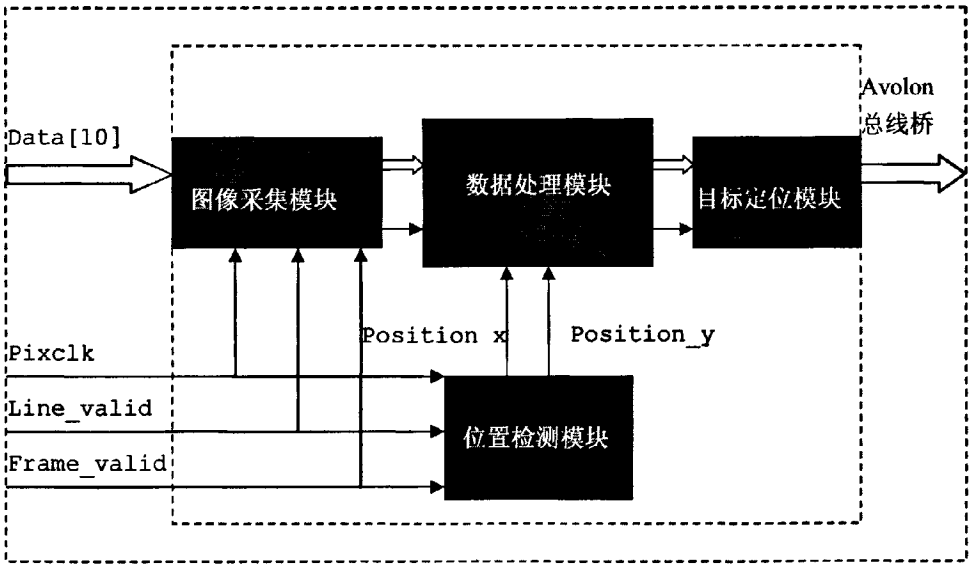


图 5-5 图像采集 IP 核结构

Figure5-5 Architecture of Image Sample IP Core

对上述结构的顶层元件定义：

```
entity cmos_color_processor is
port
(
```

```
    pclk              : in  std_logic;
    line_valid        : in  std_logic;
    frame_valid       : in  std_logic;
    data              in  std_logic_vector （PIXEL_WIDTH -1 downto 0）;
    Avalon_clk        : in  std_logic;
    Avalon_reset_n    : in  std_logic;
    Avalon_address    : in  std_logic_vector （1 downto 0）;
    Avalon_en         : in  std_logic;
    Avalon_rd         : in  std_logic;
    Avalon_data_out   : out std_logic_vector （15 downto 0）;
    Avalon_irq_out    : out std_logic --interrupt flag
);
end cmos_color_processor;
```

1) 图像采集模块

如图 5-6 所示，图像采集模块在像素时钟的驱动下对像素值进行缓存，为下一级处理模块提供数据。模块接口详细描述见表 5-1。

图 5-6 图像采集结构图

Figure5-6 Architecture of Image Sample

表5-1 图像采集接口信号定义

Table5-1 Interface Signal Definition

信号 Signal	描述 Description	方向 Direction	数据类型 Data type
pixclk	时钟信号由MT9V032的Pixclk引脚提供	输入	Std_logic
Line_valid	行同步信号，在每个时钟上升沿后开始新的一行数据传输，共有752行像素数据。	输入	Std_logic
Frame_valid	帧同步信号，在每个时钟上升边沿后开始新的一帧数据传输，每帧共有752*480个像素数据。	输入	Std_logic
Pdata[10]	从MT9V032接受到的像素数据，在每个行同步信号与帧同步信号为高，像素时钟Pixclk的上升沿，像素数据有效。	输入	Std_logic_vector（9 downto 0）
Pixeldata	把采集到的像素数据传送给下一个模块	输出	Std_logic_vector（9 downto 0）

根据上表其实体定义如下：

```
entity  color_sample is
port
 (
    pclk，  frame_valid，  line_valid :  in     std_logic;
    pdata     : out std_logic_vector （BIT_WIDTH_X -1 downto 0）；
    pixel_data: out std_logic_vector （BIT_WIDTH_Y -1 downto 0）
）；
end color_sample;
```

2) 位置检测模块

如图5-7所示，位置检测模块生成像素点的坐标值，供其它模块调用。模块接口详细描述见表5-2。

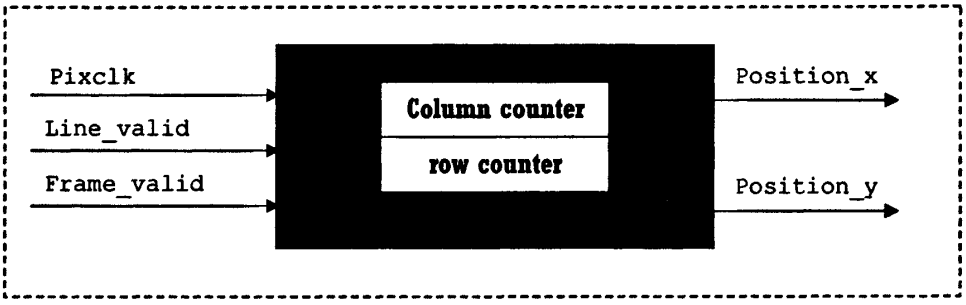


图 5-7 位置检测结构图

Figure5-7 Illustration of Position Detector

表 5-2 位置检测接口信号定义

Table5-2 Interface Signal Definition of Position Detector

信号 Signal	描述 Description	方向 Direction	数据类型 Data type
pixclk	时钟信号由MT9V032的Pixclk引脚提供	输入	Std_logic
Line_valid	行同步信号，在每个时钟上升边沿后开始新的一行数据传输，共有 752 行像素数据。	输入	Std_logic
Frame_valid	帧同步信号	输入	Std_logic
Position_x	代表正在被计算像素数据位置的 x 坐标	输出	Std_logic_vector
Position_y	代表正在被计算像素数据位置的 y 坐标	输出	Std_logic_vector

根据上表其实体定义如下：

```
entity position_determination is
port
(
    pclk,  frame_valid,  line_valid :    in    std_logic;
```

```
    position_x : out std_logic_vector （BIT_WIDTH_X -1 downto 0）；
    position_y: out std_logic_vector （BIT_WIDTH_Y -1 downto 0）
）；
end position_determination;
```

3) 数据处理模块

如图 5-8 所示，数据处理模块根据可行性分析的原理分别计算出符合条件的总像素 X 坐标、Y 坐标之和。模块接口详细描述见表 5-3。

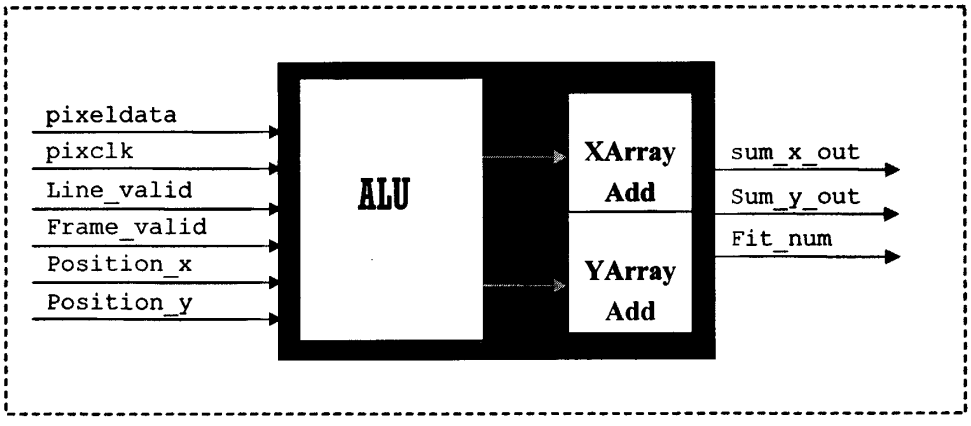


图 5-8 数据处理结构图

Figure5-8 Illustration of Data Process

表 5-3 数据处理接口信号定义

Table5-3 Interface Signal Definition of Data Process

信号 Signal	描述 Description	方向 Direction	数据类型 Data type
position_x	来自位置检测模块的位置 x 数据	输入	Std_logic_vector
Position_y	来自位置检测模块的位置 y 数据	输入	Std_logic_vector
pixclk	像素时钟	输入	Std_logic
Line_valid	行同步信号	输入	Std_logic

Frame_valid	帧同步信号	输入	Std_logic
pixeldata	来自采集模块的像数数据	输入	Std_logic_vector
Sum_x_out	满足 R-B>阈值 T 所有点 x 坐标之和	输出	Std_logic_vector
Sum_y_out	满足 R-B>阈值 T 所有点 y 坐标之和	输出	Std_logic_vector
Fit_num	满足 R-B>阈值所有像素点之和	输出	Std_logic_vector

根据上表其实体定义如下：

```
entity color_process is
port
 (
    frame_valid        :in   STD_LOGIC;
    line_valid         :in   STD_LOGIC;
    pclk               :in   STD_LOGIC;
    position_x         :in   STD_LOGIC_VECTOR(BIT_WIDTH_X   -1 downto 0);
    position_y         :in   STD_LOGIC_VECTOR(BIT_WIDTH_Y   -1 downto 0);
    pixel_data         :in   std_logic_vector（PIXEL_WIDTH -1 downto 0）;
    sum_x_out          :out std_logic_vector（DIVISION_WIDTH -1 downto 0）;
    sum_y_out          :out std_logic_vector（DIVISION_WIDTH -1 downto 0）;
    fit_num            :out std_logic_vector（DIVISOR_WIDTH   -1 downto 0）
）;
end color_process;
```

4) 目标定位模块

如图 5-9 所示，目标定位模块由减法实现除法运算。计算公式为：

Target_x=Sum_x/Fit_num

Target_y=Sum_y/Fit_num

模块接口详细描述见表 5-4。

图 5-9 目标定位结构图

Figure5-9 Illustration of Target Identification

表 5-4 目标定位接口信号定义

Table5-4 Interface Signal Definition of Target Identification

信号 Signal	描述 Description	方向 Direction	数据类型 Data type
position_x	来自位置检测模块的位置 x 数据	输入	Std_logic_vector
Position_y	来自位置检测模块的位置 y 数据	输入	Std_logic_vector
pclk	像素时钟	输入	Std_logic_vector
Sum_x	满足 G-B>阈值 T 所有点 x 坐标之和	输入	Std_logic_vector
Sum_y	满足 G-B>阈值 T 所有点 y 坐标之和	输入	Std_logic_vector
Fit_num	满足 G-B>阈值所有像素点之和	输入	Std_logic_vector
Target_x	绿色目标位置的 x 坐标	输出	Std_logic_vector
Target_y	绿色目标位置的 y 坐标	输出	Std_logic_vector
flag	目标确定后的标志位，高有效	输出	Std_logic

根据上表其实体定义如下：

```
entity target_position_confirm is
port
 (
     pclk          : in   std_logic;
     position_x    : in   std_logic_vector （BIT_WIDTH_X-1   downto 0）;
```

```
    position_y   : in  std_logic_vector (BIT_WIDTH_Y -1   downto 0);
    sum_x        : in  std_logic_vector (DIVISION_WIDTH-1downto 0);
    sum_y        : in  std_logic_vector (DIVISION_WIDTH-1 downto 0);
    fit_num : in  std_logic_vector (DIVISOR_WIDTH-1      downto 0);
    target_x     : out std_logic_vector (BIT_WIDTH_X-2   downto 0);
    target_y     : out std_logic_vector (BIT_WIDTH_Y-2   downto 0);
    flag         : out std_logic
);
end target_position_confirm;
```

5) Simplicity 综合与 ModelSim 仿真

随着 EDA 技术的发展，EDA 工具的种类日益丰富，功能日趋强大，易学易用。Simplicity 是世界上最优秀的 VHDL/Verilog 综合软件之一，ModelSim 是最具代表性与影响力的仿真工具。

表 5-5 是采用 Synplify Pro 7.7 软件对上述 VHDL 程序进行硬件综合的结果，逻辑资源仅占用了 5%，最高工作频率为 120MHz，足见其高效的编译效率。

图 5-10 所示的是在输入激励为 Matlab 产生的 BayerPattern 数据下，ModelSim 的仿真结果。当 Frame_Valid 发生 1 到 0 的跳变，即一帧结束时，幼苗相对背景的坐标就被计算出来，可以看到 target_x 为 154，target_y 为 113，跟在 Matlab 中的仿真结果一样，证明该硬件描述程序能够完成图像处理的功能。

图 5-11 为 Avalon 总线协议 ModelSim 仿真结果，在 NiosII 发起总线读信号时，该模块根据地址总线传送的数据向数据总线提供对应的数值，由图可以看到，当相对地址 avalon_address 为 0 时，avalon_data_out 端口出现幼苗的 X 坐标值 154，当 avalon_address 为 1 时，avalon_data_out 端口出现幼苗的 Y 坐标值 113,这样，图像处理的结果就通过 avalon 总线传送给 NiosII 处理器。NiosII 再根据这个坐标值作出相应的控制。

图像处理 RTL 顶层描述见附录三。

表 5-5 Synplify 综合结果

Table5-5 Synthesis Result using Synplify

综合软件版本	Synplify pro 7.7
所用逻辑资源	314 of 5980（5%）
最大工作频率	120.7 MHz

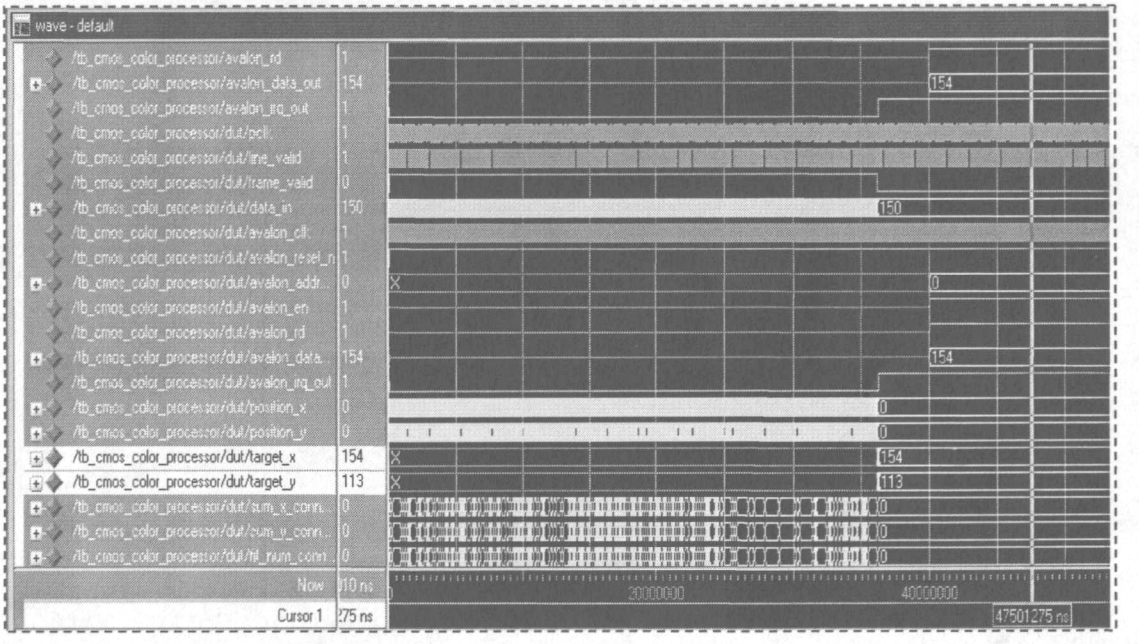


图 5-10 数据处理时序图

Figure5-10 Timing of Data Process

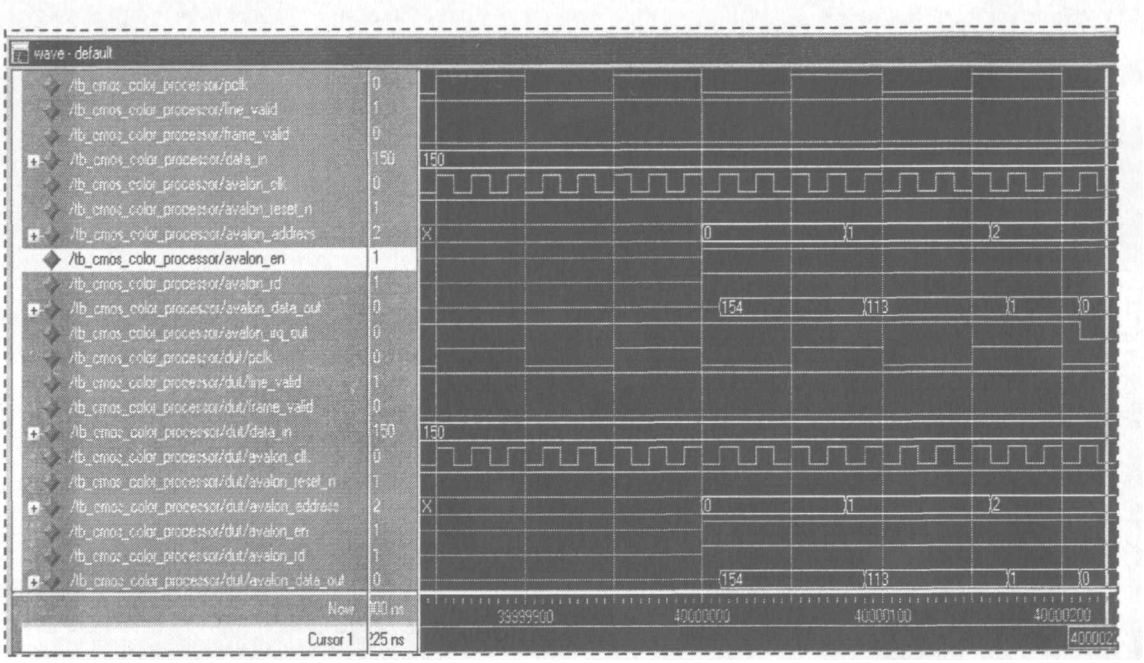


图 5-11 Avalon 总线时序图

Figure5-11 Timing of Avalon Bus

5.2.2 图像采集显示 RTL 描述

1) EPP 并口传输协议

用 FPGA 做图像处理算法时，经常要传输大量数据到 PC 机，这样就需要简单快速的传输方法。单片机常用串口，但速度有限。并口是 PC 机基本配置，传输速度相对较快。它具有 SPP、EPP、ECP 三种工作模式：SPP 支持单向数据传送；EPP 协议支持双向数据传送，且数据传输速度为 500KB/s～2MB/s；ECP 对硬件要求较高。这里采用 EPP 协议来完成图像数据的传输。其接口标准如表 5-6 所示。

表5-6 EPP接口标准

Table5-6 Interface Standards of EPP

Pin	EPP信号	In/Out	功能
1	Write	Out	引脚为低时，写信号；引脚为高时，读信号
2—9	Data	In Out	双向数据总线
11	Wait	In	数据传输握手信号
14	DataStrobe	Out	数据传送，低有效

16	Reset	Out	复位信号
17	AddressStrobe	Out	地址传送，低有效
18-25	Ground	Out	接地

2) 状态机设计

如图 5-12 所示，整个图像采集传输分为等待帧状态、采集一帧状态、帧数据传送状态和空闲状态四个状态。状态变化由按键控制，方便调试。

如图 5-13 所示，并口的状态可分为等待写、等待读、空闲三个状态。状态变化由 PC 软件控制。

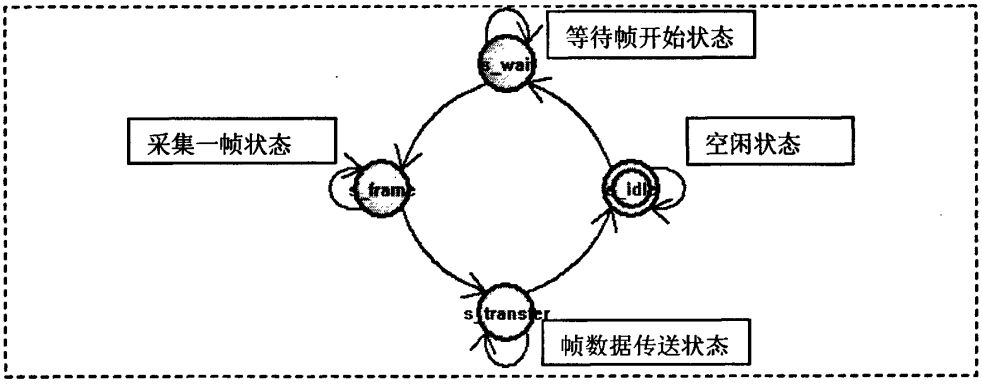


图 5-12 状态转化图

Figure5-12 Illustration of State Changing

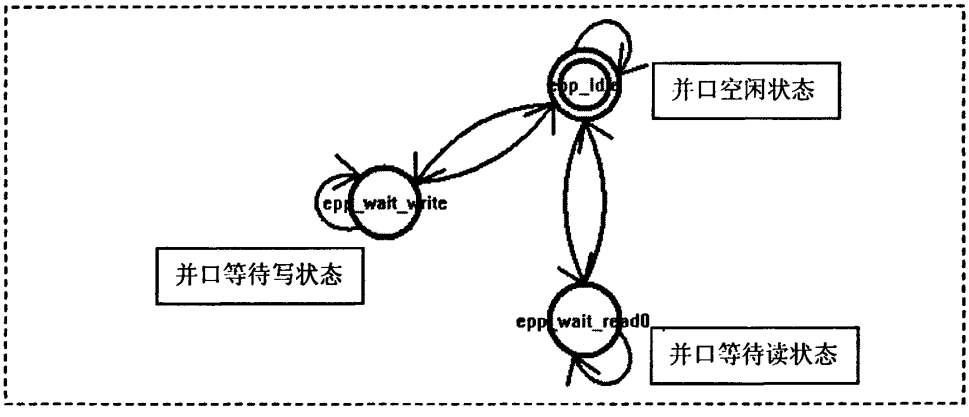


图 5-13 EPP 状态图

Figure5-13 EPP State Diagram

3) 图像采集显示结构

如图 5-14 所示，I2C 控制器对 MT9V032 的工作方式进行设置。时序控制器将采集到的图像存到 SRAM，当一帧数据采集完成，等待上位机读信号。一旦信号到来，EPP 控制器就负责把 SRAM 中的图像数据按照 EPP 时序标准传送给 PC 机。图像采集显示结构的综合结果表 5-7 所示。

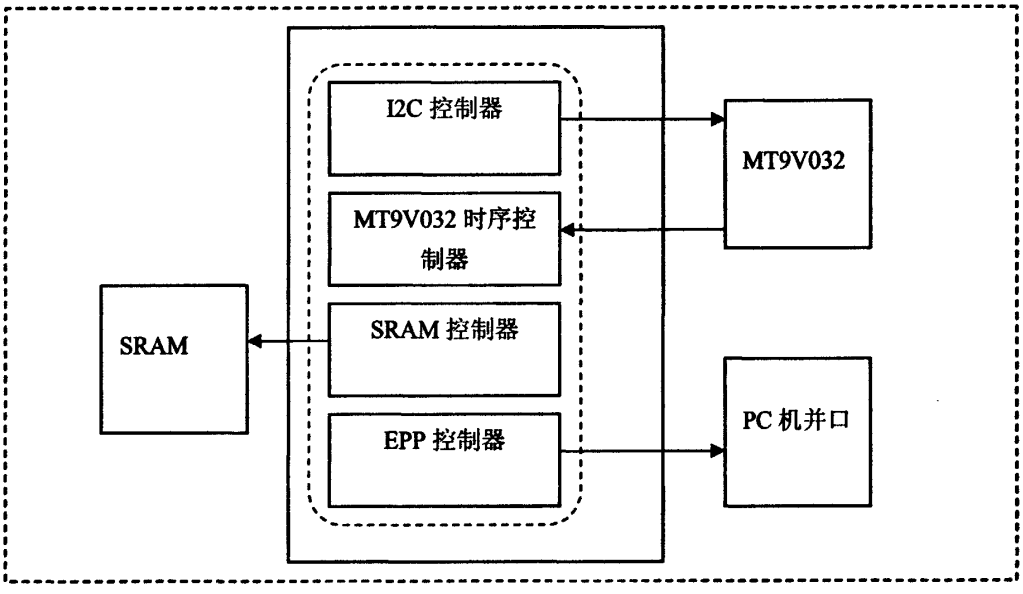


图 5-14 图像采集显示结构图

Figure5-14 Architecture of Image Sample and Display

表 5-7 Synplify report

Table5-7 Synplify report

综合软件版本	Synplify pro 7.7
所用逻辑资源	246 of 5980 （ 4%）
最大工作频率	151.29 MHz

图 5-15 为图像采集存储时的时序图。Matlab 生成的 BayerPattern 数据在时钟的驱动下进入图像采集存储模块。由图可以看出，pixel_data 端口发生明显的数据变化，当 nWe 由 0 变 1 时，图像数据被存储到 SRAM 中。

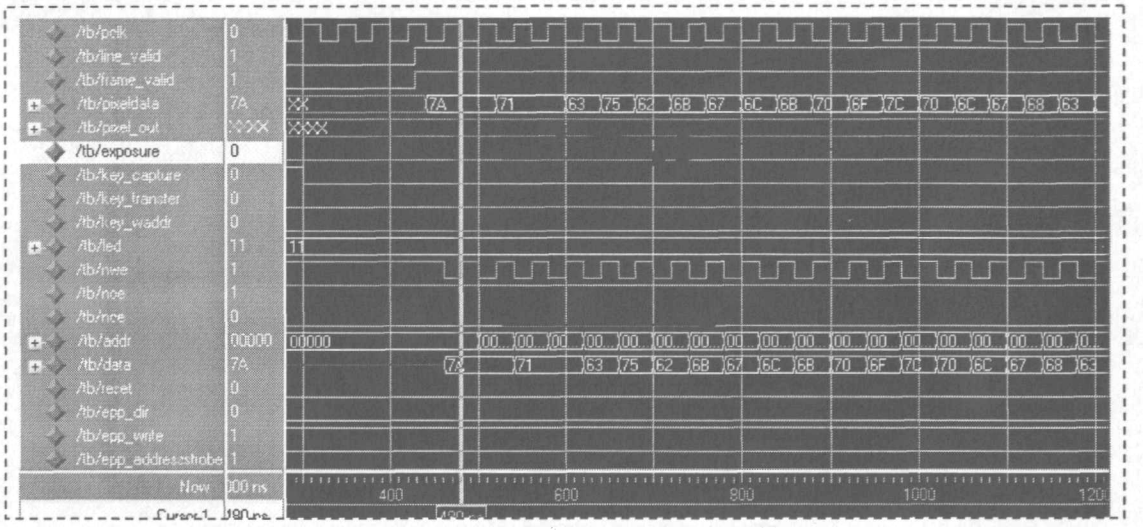


图 5-15 采集存储时序图

Figure5-15 Timing of Sample and Store

图 5-16 为 I^2C 控制器的时序图，其 I^2C 输出为三态，所以引脚上外接 1.5K 的上拉电阻。

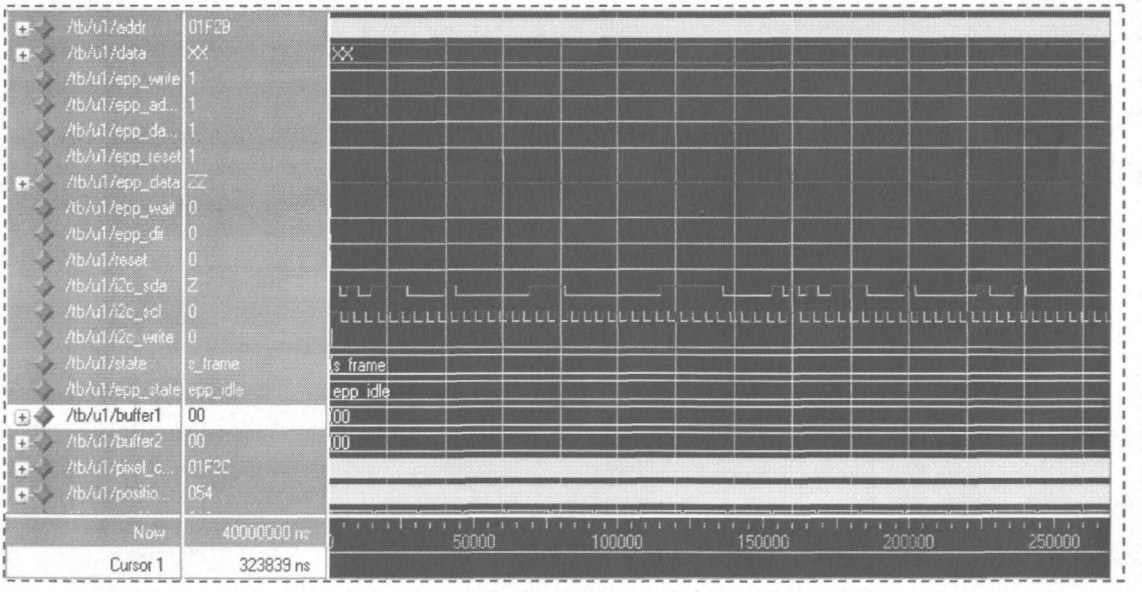


图 5-16 I^2C 时序图

Figure5-16 Timing of I^2C

5.2.3 开源 I²C IP 核移植

软件开源已经走了很久，如大名鼎鼎的 linux 操作系统就是就是开源典范。源码开放软件库 sourceforge.net 有数不清的范例可供学习使用。在硬件开源方面，也存在这样一个组织——OpenCores，它致力于发展公开 IP 核，向工程师提供一些免费 IP 核，应用工程师可以很方便的把这些免费的 IP 核嵌入到自己的系统里，实现一个片上系统。

本设计需要通过 I²C 总线对 MT9V032 的工作方式进行设置，而 SOPC 没有提供 I²C 的 IP 核，因此可以将开源 OpenCores 中的基于 Wishbone 总线结构的 I²CMaster IP 核修改后，移植到 Avalon 总线之上[42][43]。该 I²C IP 核可以从以下连接地址下载得到：http://www.opencores.com/projects.cgi/web/i2c/overview

I²C 结构如图 5-17 所示，它由字节控制单元、位控制单元、时钟产生单元几个部分构成。I²C IP 核遵循 Wishibone 总线规范，而 Wishbone 总线与 Avalon 在时序上基本是兼容的，只要在引脚稍加改动就可以移植到 Avalon 总线之上。

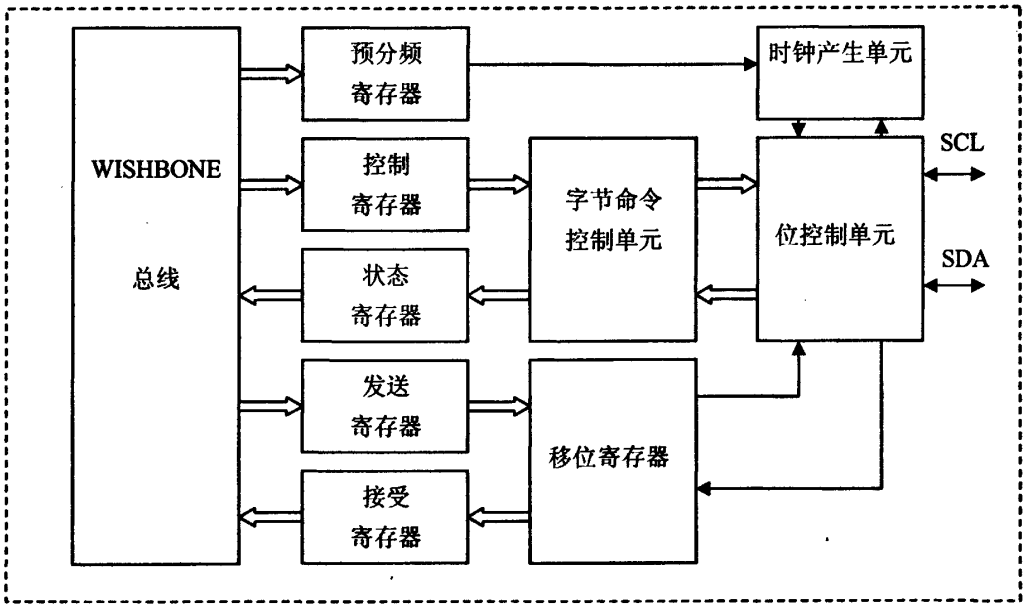


图 5-17 I²C 结构图

Figure5-17 I²C Architecture

5.3 SOPC 下的硬件与软件

5.3.1 NIOS-II 微处理器内核定制

在完成系统的软硬件划分后，就可以根据设计要求定制自己的微处理器系统。如图 5-18 所示，本系统需要定制的 IP 模块有图像采集处理模块、I^2C 模块、UART 模块、PIO 模块、SRAM 控制器、EPP 模块和一个标准型的 NiosII 内核，各个模块通过 Avalon 交换结构实现互联[44]。其中，图像采集处理模块已在 5.2.1 节中作详细介绍，I^2C 控制器在 5.2.3 节中已介绍移植方法，UART 模块、PIO 模块、SRAM 控制器、EPP 模块由 Quartus II 软件提供。下面对系统中最重要的 Avalon 交换结构加以说明。

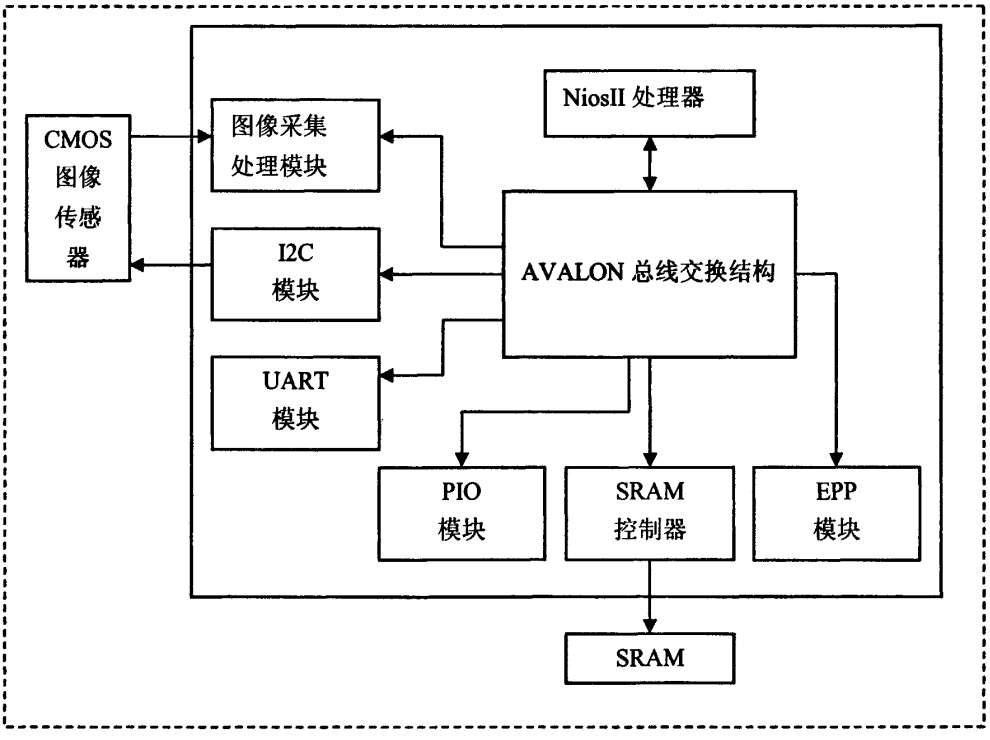


图 5-18 系统功能结构图

Figure5-18 System Function Architecture

1) Avalon 交换结构

IP 核通过总线互联机制和微处理器实现片上系统。在把自己设计的 IP 核和其他商用 IP 嵌入到微处理器时，必须满足相应的总线规范。如 ARM 微处理器一般采用 AMBA 总线。Altera 根据可编程片上系统的特点，提出了 Avalon 总线机制。Avalon 交换结构是一种在可编程片上系统（SOPC）中连接片上处理器和各种外设的互连结构，它定义了结点之间通讯的信号类型和时序关系，使得用户可以非常方便的把自己选定或设计的外设模块通过 Avalon 总线连接到 NiosII 系统上。

Avalon 总线设计的原则就是操作简单，占用的逻辑资源经过了优化。其接口信号全部和 Avalon 总线时钟同步。Avalon 交换总线的特性包括：

- 简单的图形界面配置方式
- 同时多个主设备的操作
- 最大支持 4GB 的寻址空间
- 同步接口
- 内嵌地址译码功能
- 带延时的读写操作
- 流传输
- 动态外设总线宽度调整

Avalon 总线不同于传统的 CPU 外部总线。在 FPGA 内部，信号走线数量不再是设计的瓶颈，Avalon 采用一种全交换功能的内嵌总线形式。如图 5-19 所示：Avalon 在结构上完全不同于传统共享式总线（PCI），它在需要连接的每一个主从对之间都有点到点的连接关系。也就是说，不同的主从对之间可以同时进行通讯，而不会发生任何冲突，这样大大提高了总线的性能。

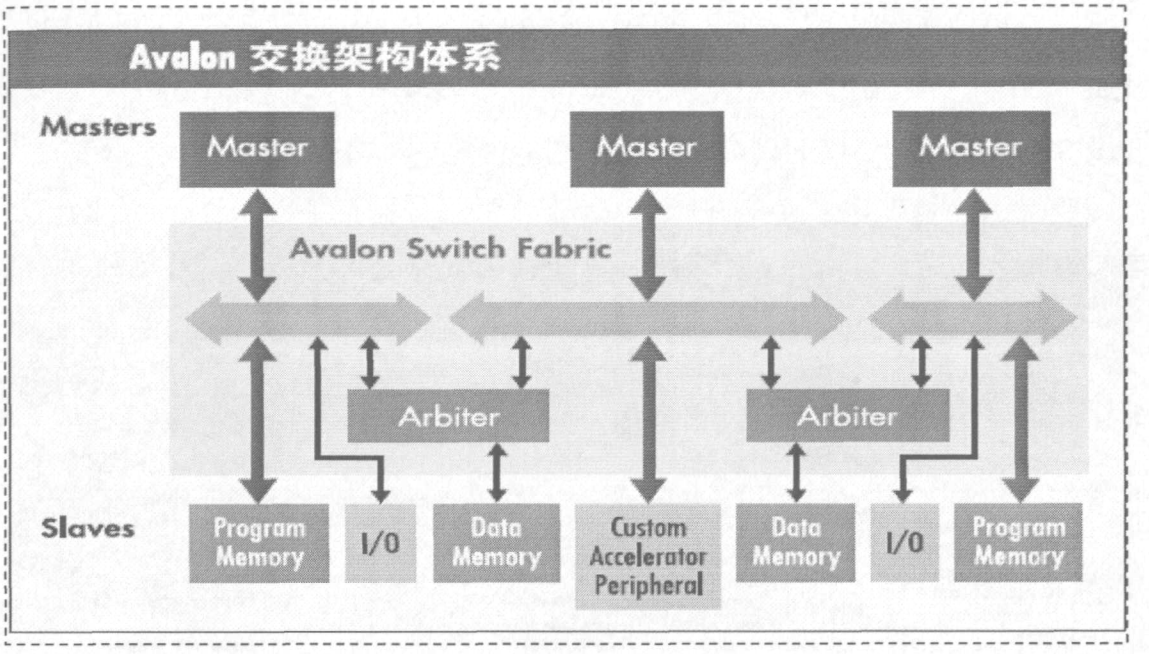


图 5-19 Avalon 交换架构体系

Figure5-19 Avalon Switch Fabric

2) 使用 SOPC 构建系统硬件

传统的 SOC 设计方法，需要用户把处理器以及外设一个一个手动地连接起来，还需要用户手动去分配地址空间资源，这样既耗时又容易出错。Altera 公司针对这种情况，开发了一种智能工具，帮助用户快速的产生一个系统，这个工具就是 SOPC Builder。SOPC Builder 是内嵌在 Quartus II 设计工具中的，用户可以非常方便地在用 SOPC Bulider 产生完一个系统后，在 Quartus II 中对其进行编译，并实现在目标器件中。

SOPC Builder 最大的好处是使得系统设计过程自动化，这也是 EDA 业界追求的目标。在 SOPC Builder 中，设计者只需要选择自己需要的处理器和外设类型，SOPC Bulilder 工具将自动根据 Avalon 总线的标准产生一些互连资源，将各个模块连接起来。这些互连资源功能包括数据通道复用、等待状态产生、中断控制和数据宽度匹配。

图 5-20 为系统的定制界面，在 System Contents 项目中可以看到，SOPC 不仅提供了 NiosII 处理器内核，还提供了 Ethernet、PCI、UART、SPI 等各种协议接口，以及 SDRAM、SRAM、Flash 存储器接口[45]。

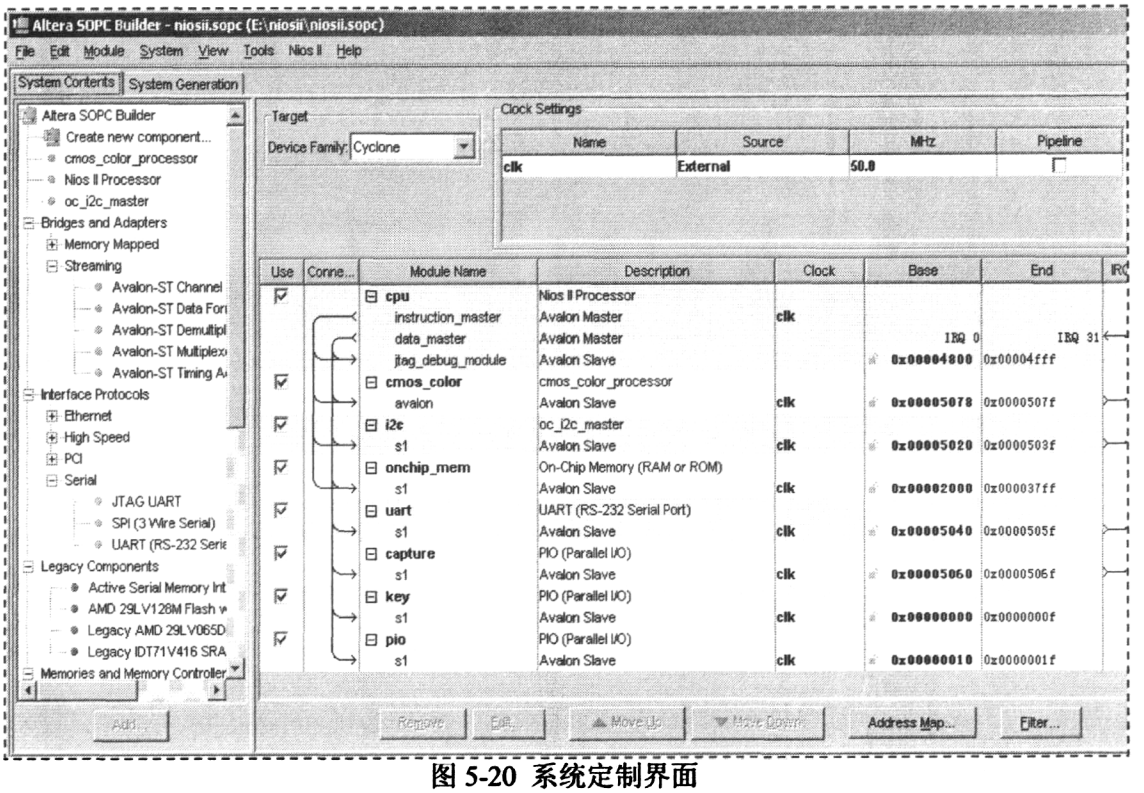


图 5-20 系统定制界面

Figure5-20 System Custom Interface

图 5-21 为 NiosII 的定制结果，调用该元件后可直接编译生成下载文件。表 5-8 为系统编译结果，整个 NiosII 系统用了 1734 个逻辑单元，占系统资源的 46%。时序分析结果如表 5-9 所示，系统最高运行频率为 63MHz。

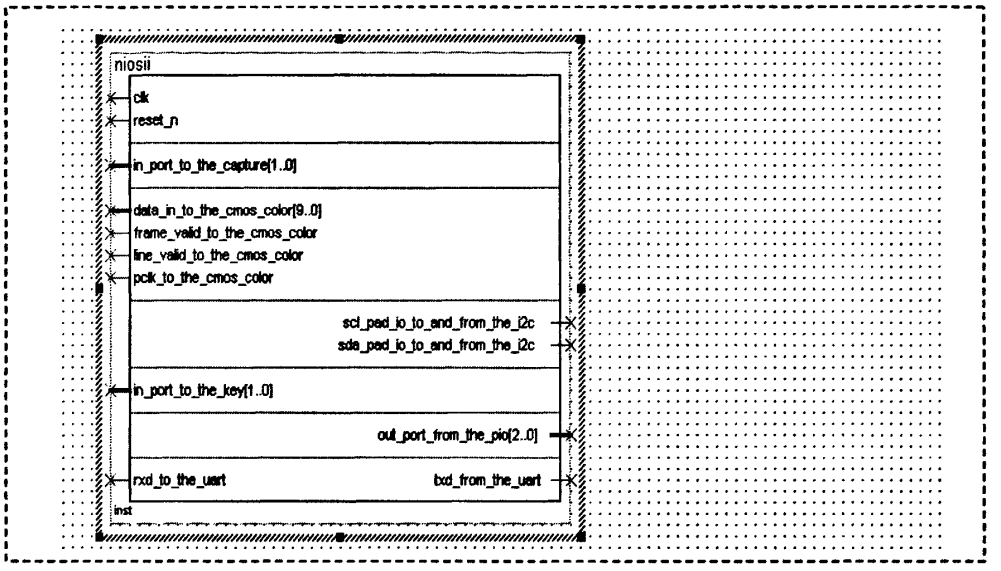


图 5-21 NiosII 定制结果

Figure5-21 NiosII Custom Result

表 5-8 编译结果

Table5-8 Compile Result

编译状态及时间	成功编译 2007-9-14 星期五
Quartus II 版本	7．1 BUILDER FULL VERSION
版本号	NIOS-II
顶层实体名	NiosII
器件系列	CYCLONE
器件	EP1C6T144C8
逻辑单元（LE）	1734/5980 （46%）
IO 引脚	30/98 （31%）
虚拟引脚	0
RAM（BIT）	63712/92160（69%）
PLL	0/2（0%）

表 5-9 时序约束

Table5-9 Time Constraint

Type	Actual Time
Worst-case tsu	8.972 ns
Worst-case tpd	2.124 ns
Worst-case th	3.503 ns
Worst-case tco	8.088 ns
Total number of failed paths	
Clock Setup: 'pclk_to_the_cmos_color'	76.03 MHz （ period = 13.152 ns ）
Clock Setup: 'line_valid_to_the_cmos_color'	Restricted to 275.03 MHz （ period = 3.636 ns ）
Clock Setup: 'clk'	77.58 MHz （ period = 12.890 ns ）
Clock Setup: 'altera_internal_jtag~TCKUTAP'	161.60 MHz （ period = 6.188 ns ）
Clock Hold: 'pclk_to_the_cmos_color'	N/A

5.3.2 NIOS-II 处理器下的软件

1) 系统软件开发流程

图 5-22 所示为 NiosII 嵌入式处理器的开发流程，SOPC Builder 完成 NiosII 系统的定制，QuartusII 用于将定制结果综合，编译，下载，NiosII IDE 用于微处理器下的软件开发。这充分体现了 SOPC 的思想——即硬件可灵活裁剪，软件可灵活编程。

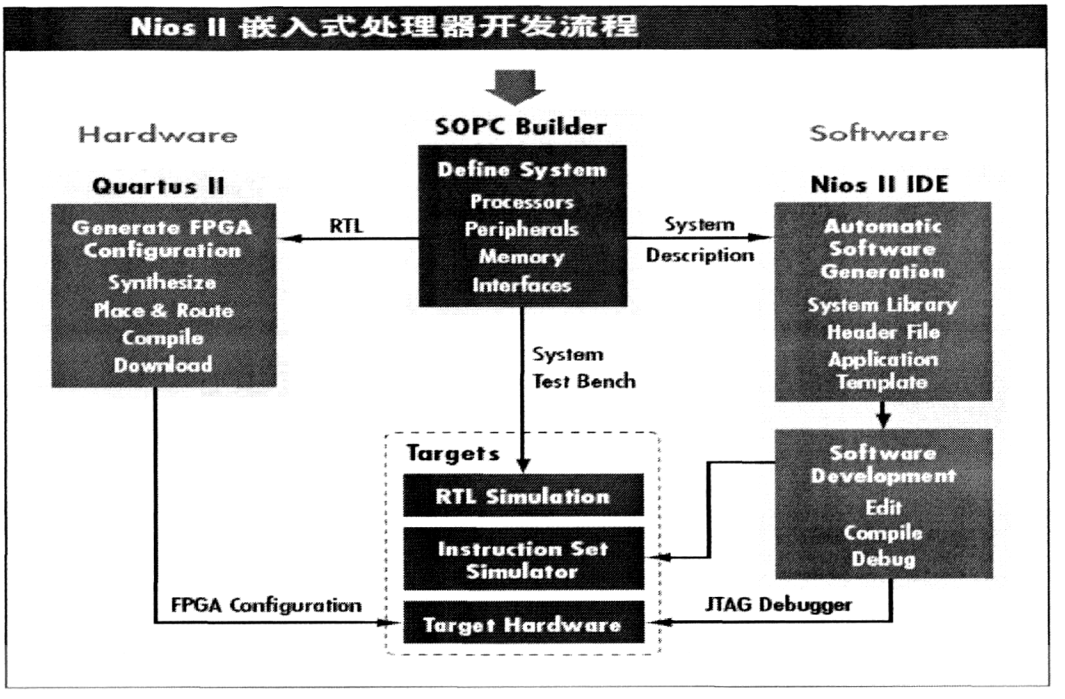


图 5-22 开发流程

Figure5-22 Development Flow

2) 主程序流程图

图 5-23 所示为整个图像采集处理系统的的流程图。程序的开始，对系统初始化，当同步脉冲到来，开始一幅图像的采集处理，图像处理单元完成数据处理后向 NiosII 发出一个中断信号，NiosII 产生中断读取坐标值，然后作出相应的控制。

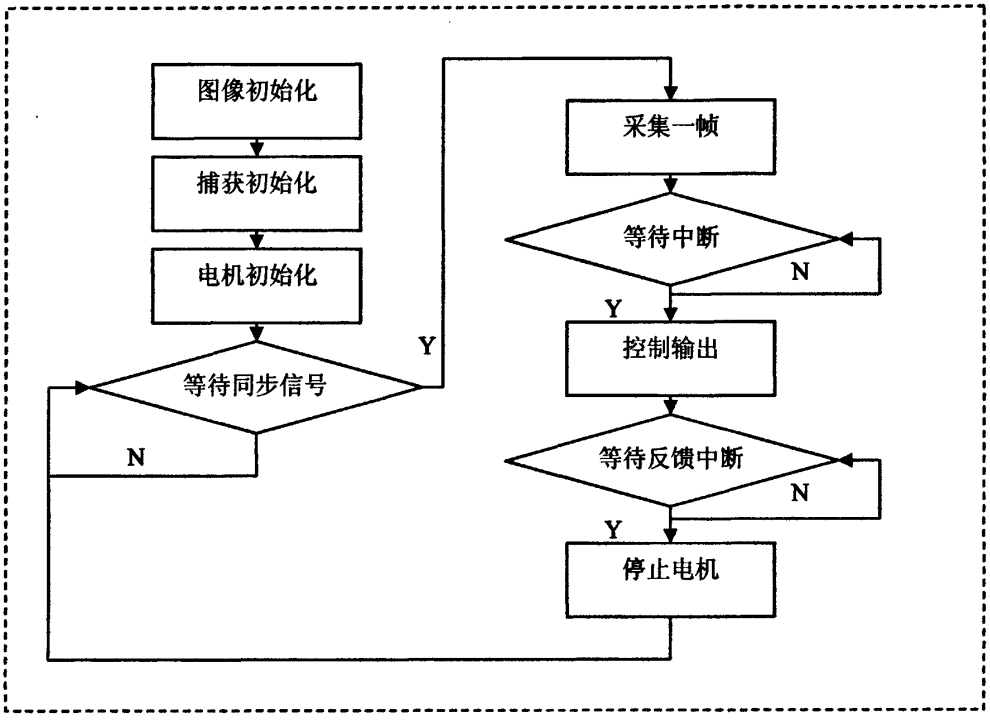


图 5-23 主程序流程图

Figure5-23 Main Program Flow Diagram

5.3.3 并口 PC 上位机通讯程序

为了检验图像效果，这里在 Visual Studio 2005 环境下开发了一个上位机程序。Windows XP 工作在保护模式下，不能直接对 IO 口进行访问，这里从网上免费下载了一个 DriverLINX 并口驱动，通过调用函数实现对并口的读写操作。相关的读写函数定义如下：

UCHAR DLPORT_API DlPortReadPortUchar（IN ULONG Port ）；//从并口读一个字节

VOID DLPORT_API DlPortWritePortUchar（IN ULONG Port， IN UCHAR Value）；//往并口写一个字节

图像显示程序见附录五。

PC 机接受数据显示图像的界面如图 5-24 所示。

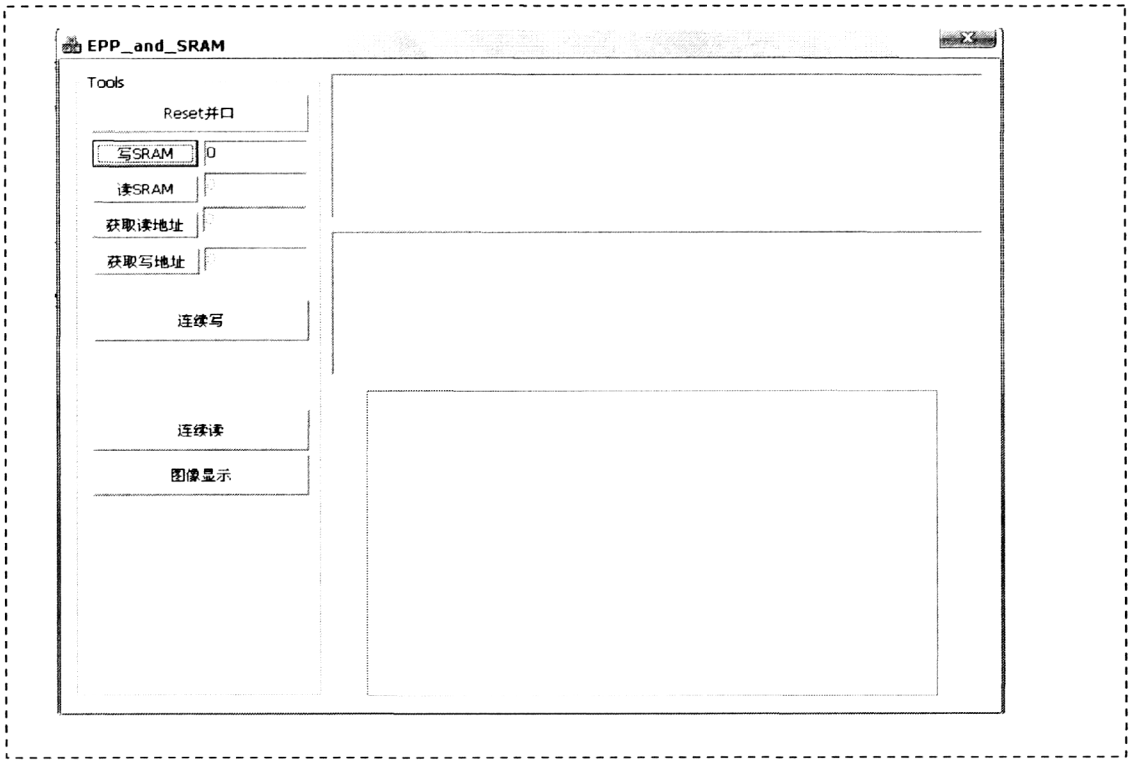


图 5-24 并口通讯界面

Figure5-24 Parallel Communication Interface

5.4 软件调试结果与分析

图 5-25 所示的三幅图是实验所拍到的图像。图像数据经 FPGA 传送给 PC 机，通过适当程序处理，在 372*240 分辨率下显示。可以看到：在没有经过任何去噪处理的情况下，就可以得到清晰的图像，显示效果良好。

图 5-25 测试图像的显示

Figure5-25 Display of Testing Image

图 5-25 为测试所拍到的幼苗目标图像。NisoII 系统上的图像处理单元，计算出了目标的位置，处理器读取位置坐标打印出位置坐标，图 5-26 显示，串口输出的 X 坐标为 383，Y 坐标为 252，跟实际图像的位置相近。

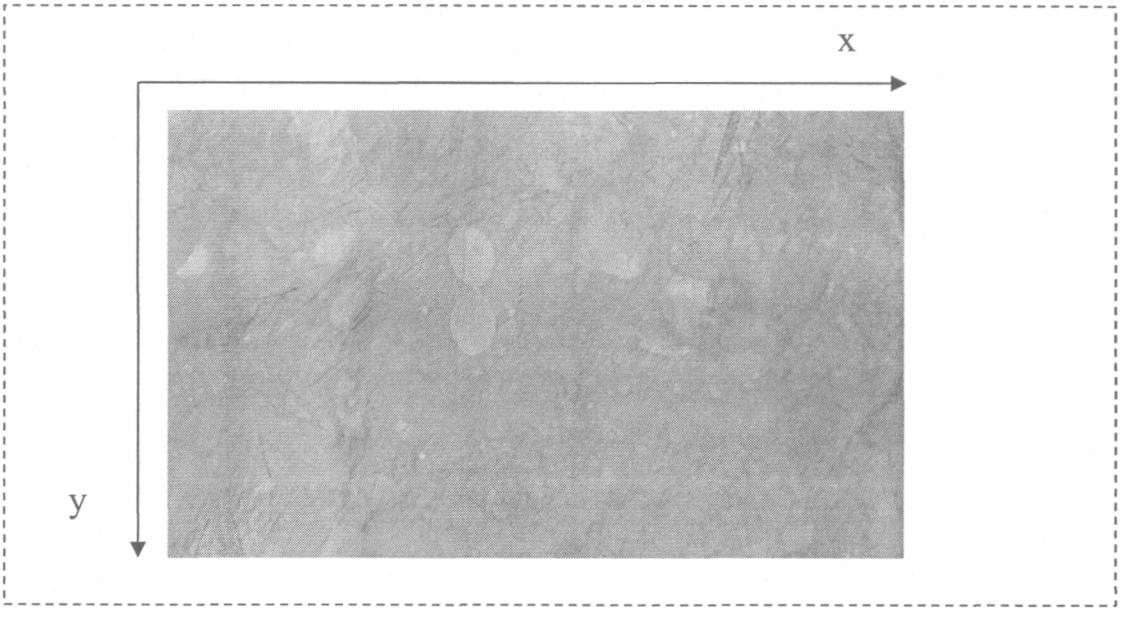


图 5-25 检测图像

Figure5-25 Testing Image

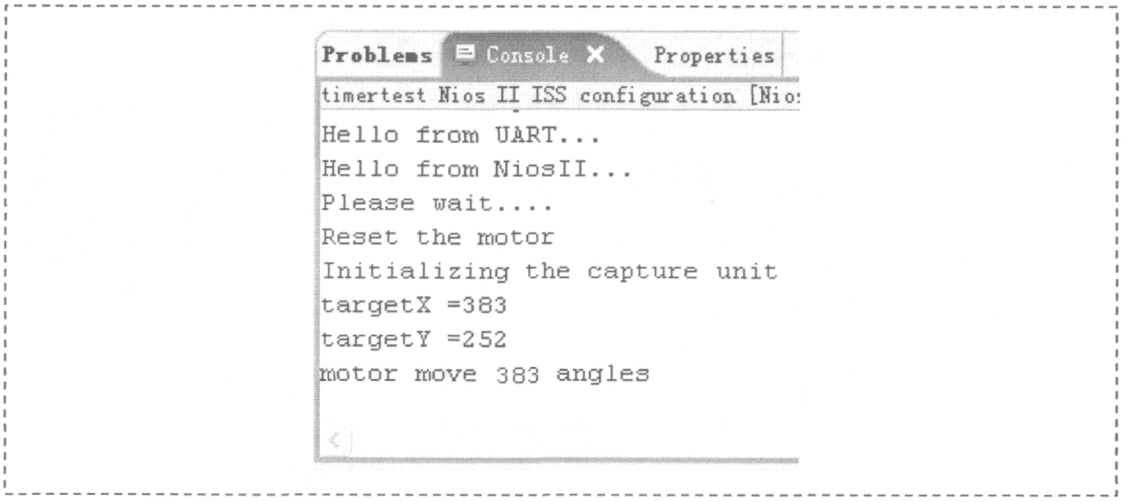


图 5-26 NiosII IDE 串口调试界面

Figure5-26 Serial Debug Interface of NiosII IDE

结　论

通过本文的分析和研究可以看出，基于数字图像处理技术、微处理器技术和嵌入式技术的控制系统设计方案能够以较小的系统实现机器视觉系统设计。本课题选用 FPGA 作为核心控制器和 Micron 公司高性能图像传感器芯片 MT9V032 完成整个图像采集控制系统，充分利用 FPGA 灵活的硬件与软件设计方式，在软硬件协同设计的方法学指导下，真正实现了一个采集控制片上系统，并且硬件电路工作稳定。本课题提出了一种新颖的、低成本的嵌入式机器视觉方案，具有一定的参考价值。

本论文研究过程中的主要工作有：

1) 掌握了嵌入式系统的设计方法、开发流程、硬件与软件系统设计原理，对由 FPGA ，ARM，DSP 组成的嵌入式系统作了深入的研究。
2) 系统了解了图像传感器的成像原理。对各个知名公司的图像传感器作了调查，分析比较了 CCD 与 CMOS 图像传感器，特别对赛普拉斯（Cypress），美光（Micron）公司具备全局快门的 CMOS 图像传感器作了具体研究。
3) 对世界两大 FPGA 生产厂商 Xilinx 和 Altera 公司的产品作了详细的研究，了解了其公司的各类产品，掌握了 SOPC 的系统开发方法。
4) 根据要求对系统的可行性进行了分析，在 Matlab 下对算法进行了验证。
5) 针对方案设计了相应的硬件与软件，制作的电路板工作稳定。
6) 对系统进行了系统的调试，虽然调试过程一度出现问题，但是经过努力，最终在电脑上显示出了清晰的图像，图像处理部分能实时采集到目标的坐标，达到了设计的要求！

当然，整个系统还有待于深入研究，使其组成一个更完善的系统，这将是以后的努力方向。

参 考 文 献

[1] 唐向阳，杨勇.机器视觉关键技术的现状及应用展望.云南:云南昆船设计研究院，2003

[2] 王巧华，文友先，刘俭英.我国机器视觉技术的发展前沿.武汉:华中农业大学，2002

[3] Wei Zhenzhong, Zhang Guangjun, Li Xin.The. Application of machine vision in inspecting position-control accuracy of motor control systems IEEE CNFVolume 2, 18-20, 2001

[4] Gedeon D.V, Gedeon T.D.Applying machine vision in electrical component manufacturing.IEEE CNF:Electrical Electronics Insulation Conference and Electrical Manufacturing Page (s) :737-749

[5] 段峰，王耀南，雷晓峰. 机器视觉技术及其应用综述，北京：自动化博览，2007

[6] Stephan Hengstler, Hamid Aghajan.A Smart Camera Mote Architecture for Distributed Intelligent Surveillance. Stanford University, 2006

[7] Daniel Crispell.Implementation of a Streaming Camera using an FPGA and CMOS Image Sensor. Brown University

[8] Xfest.Implementing Video over Ethernet Solutions On Xilinx FPGAS.2007

[9] http://www.vtest.com.cn/index.htm

[10] 唐杉，徐强，王莉微.数字IC设方法，技巧与实践.北京：机械工业出版社.2006.01

[11] http://www.arm.com/products，2007

[12] 刘皖.FPGA设计与应用.北京：清华大学出版社.2005

[13] 黄智伟.FPGA系统设计与实践.北京:电子工业出版社，2005.01

[14] 王诚.Altera FPGA/CPLD设计基础篇.北京:人民邮电出版社，2005.1

[15] 汤磊.SOC设计领域的核心技术—软/硬件协同设计，北京：信息产业中心

[16] Tang Lei, Wei Shaojun, Qiu Yulin, CFG in sub-graph matching based HW/SW co-design. The 4th International Conference on ASIC Proceedings, 2001; pp. 171-174.

[17] 杨刚.32位嵌入式系统与SoC设计导论.北京:电子工业出版社，2006

[18] 郑亚民.可编程逻辑器件开发软件Quartus II.北京:国防工业出版社，2006.9

[19] 米本和也.CCD/CMOS图像传感器基础与应用.北京：科学出版社，2006

[20] 王元庆.新型传感器原理及应用.北京：机械工业出版社，2002.5

[21] Luster LightVision Corp.Imaging&Vision图像与机器视觉产品手册 北京.2007

[22] Dalsa.CMOS vs CCD Maturing Technologies, Maturing Markets.www.dalsa.com

[23] A Micron White Paper. The Evolution of Digital Imaging From CCD to CMOS. 2006

[24] Kodak. Shutter Operations for CCD & CMOS. www. kodak. com/go/images

[25] http://www. cypress. com/products/

[26] Micron. Imaging Solution Guides. 2007

[27] Mircon. MT9V032C12STC Datasheet. 2006

[28] Altera. Cyclone Device Handbook Volume . 2007

[29] www. linear. com Product Selection Guide. 2007

[30] www. issi. com. SRAM Selection Guide. 2007

[31] ISSI. IS61LV5128AL Datasheet, 2006

[32] Linear Technology. L6235 Datesheet. 2003

[33] 汪思敏. PCB 与电磁兼容设计. 北京:机械工业出版社, 2006

[34] 吴继华. Altera FPGA/CPLD 设计高级篇. 北京: 人民邮电出版社 2005. 7

[35] 刘光斌. 单片机系统实用抗干扰技术. 北京:人民邮电出版社, 2003. 9

[36] 雷伏容. VHDL 电路设计. 北京:清华大学出版社, 2006. 12

[37] CMUcam2. Vision Sensor User Guide. 2003

[38] www. micron. com/innovations/imaging/pixel

[39] 尹建军, 王新忠, 毛罕平, 陈树人, 张际先. RGB 与 HSI 颜色空间下番茄图像分割的对比研究. 2006. 11

[40] Daniel Wolf, Hendrik Hanff . Color Processing for a CMOS Image Sensor Project Report . 2005

[41] Altera Avalon Memory-Mapped Interface Specification 2007. 3

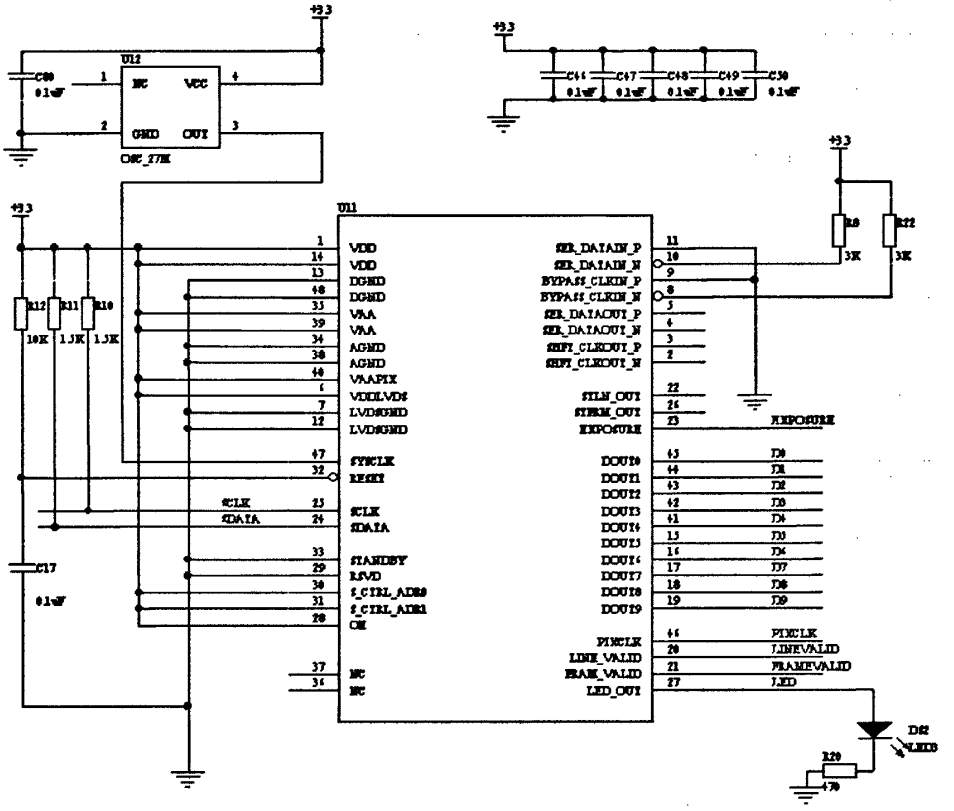
[42] Richard Herveille. I2C-Master Core Specification. www. opencores. com, 2000

[43] Wishbone System on Chip Interconnect Architecture for Portable Ip cores. www. opencores. com, 2002. 7

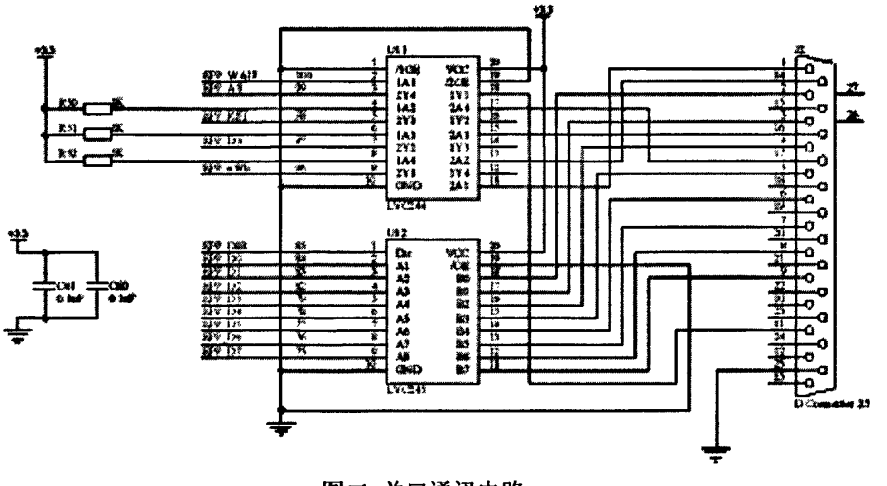
[44] Arrow Corp. Arrow Electronics Portable Reference Platform 2005. 11

[45] 马柯. 基于 NiosII 的 IP 可视终端. 西安: 西安交通大学, 2003

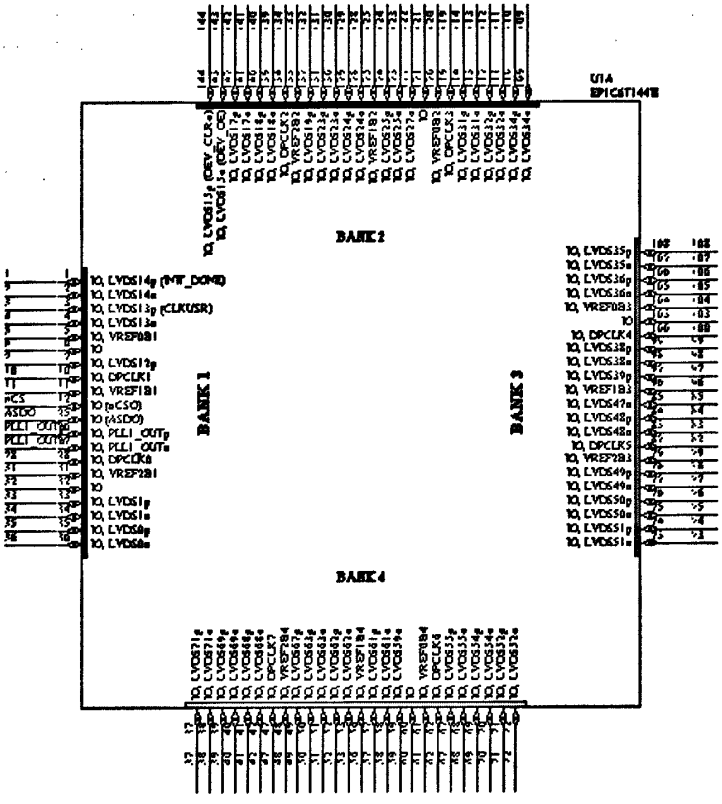
附录一 电路原理图



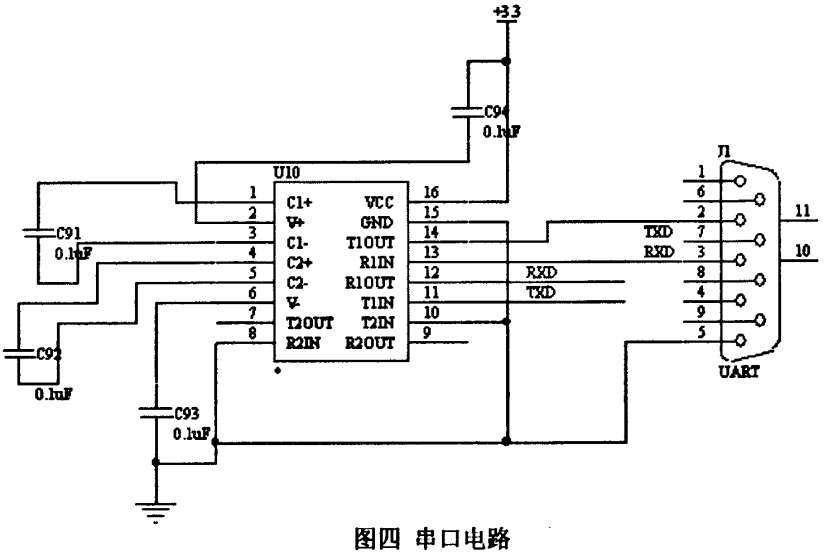
图一 MT9V032 电路图



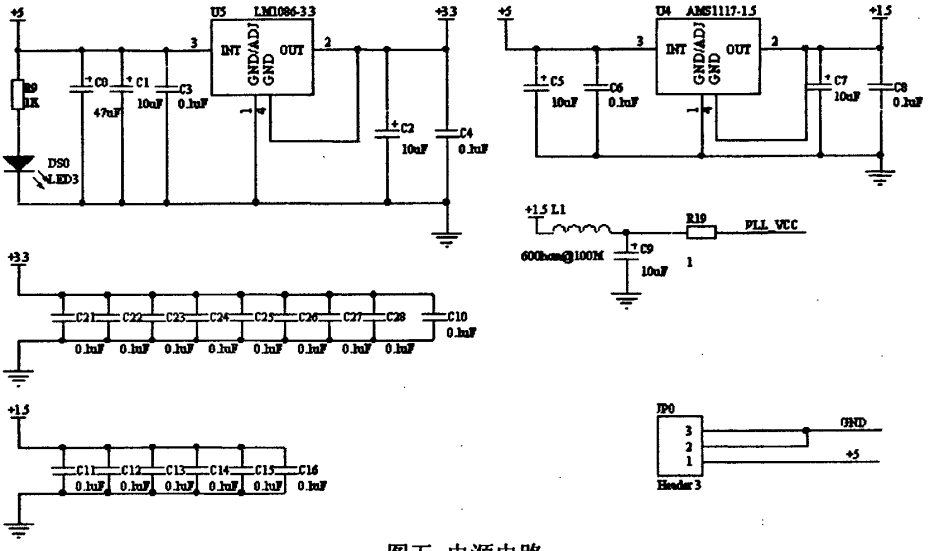
图二 并口通讯电路



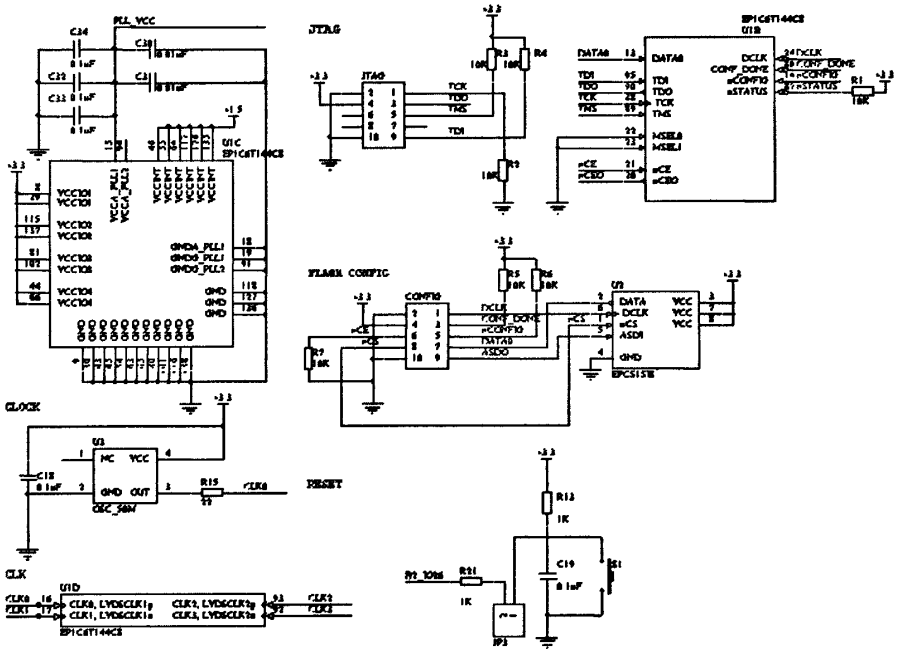
图三 I/O 接口电路图



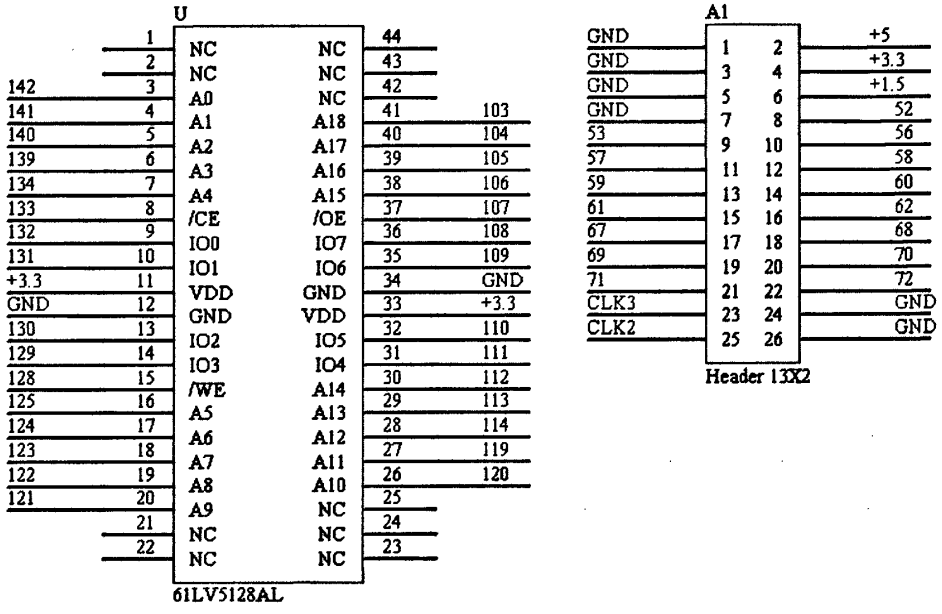
图四 串口电路



图五 电源电路



图六 配置接口电路


U
1 NC NC 44
2 NC NC 43
142 3 A0 NC 42
141 4 A1 A18 41 103
140 5 A2 A17 40 104
139 6 A3 A16 39 105
134 7 A4 A15 38 106
133 8 /CE /OE 37 107
132 9 IO0 IO7 36 108
131 10 IO1 IO6 35 109
+3.3 11 VDD GND 34 GND
GND 12 GND VDD 33 +3.3
130 13 IO2 IO5 32 110
129 14 IO3 IO4 31 111
128 15 /WE A14 30 112
125 16 A5 A13 29 113
124 17 A6 A12 28 114
123 18 A7 A11 27 119
122 19 A8 A10 26 120
121 20 A9 NC 25
21 NC NC 24
22 NC NC 23
61LV5128AL
A1
GND 1 2 +5
GND 3 4 +3.3
GND 5 6 +1.5
GND 7 8 52
53 9 10 56
57 11 12 58
59 13 14 60
61 15 16 62
67 17 18 68
69 19 20 70
71 21 22 72
CLK3 23 24 GND
CLK2 25 26 GND
Header 13X2

图七 SRAM 接口电路

附录二 Matlab 仿真程序

```
%-- 07-6-2 9:59 --%
origimg=imread（'RGB.bmp', 'bmp'）;
[r, g, b]=imread（'RGB.bmp', 'bmp'）;

[NumOfRow, NumOfColumn, clor]=size（origimg）;
noInfo=0;    % This is used as a indicator
N1=round（NumOfRow/2）;
N2=round（NumOfColumn/2）;
F1=floor（NumOfRow/2）;
F2=floor（NumOfColumn/2）;
%=========================================
% Drop r and b info at every other pixel
% Note that even and odd rows are different
%=========================================
%subplot（2, 2, 1）;
%imshow（origimg）;
a=0;
b=0;
sum_x=0;
sum_y=0;
sum_x_2=0;
sum_y_2=0;
for i=1:N1,
  for j=1:N2,
    r（2*i-1, 2*j-1, 1）=noInfo; %  r=0 odd line
    r（2*i-1, 2*j-1, 2）=noInfo; %  g=0 odd line odd column
    r（2*i-1, 2*j, 1）=noInfo;   %  r=0 odd line even column
    r（2*i-1, 2*j, 3）=noInfo;   %  b=0 odd line even column

    r（2*i, 2*j-1, 1）=noInfo;   %  r=0 even line odd column g stay
    r（2*i, 2*j-1, 3）=noInfo;   %  b=0 even line odd column
```

```
r (2*i, 2*j, 2) =noInfo;        %  r=0 even line even column
r (2*i, 2*j, 3) =noInfo;        %  b=0 even line even column
out (i, j) =0;
img (2*i-1, 2*j-1, 1) =255;
img (2*i-1, 2*j, 1) =255;
img (2*i, 2*j-1, 1) =255;     % even line odd column g
img (2*i, 2*j, 1) =255;       % even line even column b
img (2*i-1, 2*j-1, 2) =255;
img (2*i-1, 2*j, 2) =255;
img (2*i, 2*j-1, 2) =255;
img (2*i, 2*j, 2) =255;
img (2*i-1, 2*j-1, 3) =255;
img (2*i-1, 2*j, 3) =255;
img (2*i, 2*j-1, 3) =255;
img (2*i, 2*j, 3) =255;
img2 (2*i-1, 2*j-1, 1) =255;
img2 (2*i-1, 2*j, 1) =255;
img2 (2*i, 2*j-1, 1) =255;   % even line odd column g
img2 (2*i, 2*j, 1) =255;      % even line even column b
img2 (2*i-1, 2*j-1, 2) =255;
img2 (2*i-1, 2*j, 2) =255;
img2 (2*i, 2*j-1, 2) =255;
img2 (2*i, 2*j, 2) =255;
img2 (2*i-1, 2*j-1, 3) =255;
img2 (2*i-1, 2*j, 3) =255;
img2 (2*i, 2*j-1, 3) =255;
img2 (2*i, 2*j, 3) =255;
if (r (2*i-1, 2*j, 2)  -r (2*i-1, 2*j-1, 3) >60)  % old line G-B
      img (2*i-1, 2*j, 1) =r (2*i-1, 2*j, 1) ;
      img (2*i-1, 2*j-1, 1) =r (2*i-1, 2*j-1, 1) ;
      img (2*i-1, 2*j, 2) =r (2*i-1, 2*j, 2) ;
      img (2*i-1, 2*j-1, 3) =r (2*i-1, 2*j-1, 3) ;
```

```
            img (2*i-1, 2*j, 3) =r (2*i-1, 2*j, 3) ;
            img (2*i-1, 2*j-1, 2) =r (2*i-1, 2*j-1, 2) ;
            out (i, j) =r (2*i-1, 2*j, 2)  -r (2*i-1, 2*j-1, 3) -60;
            a=a+1;
            sum_x=sum_x+j;
            sum_y=sum_y+i;
        end;
        if (r (2*i, 2*j-1, 2) -r (2*i, 2*j, 1) >15)       % even line G-R
            img2 (2*i, 2*j, 1) =r (2*i, 2*j, 1) ;
            img2 (2*i, 2*j-1, 1) =r (2*i, 2*j-1, 1) ;
            img2 (2*i, 2*j, 2) =r (2*i, 2*j, 2) ;
            img2 (2*i, 2*j-1, 2) =r (2*i, 2*j-1, 2) ;
            img2 (2*i, 2*j-1, 3) =r (2*i, 2*j-1, 3) ;
            img2 (2*i, 2*j, 3) =r (2*i, 2*j, 3) ;
           out2 (i, j) =r (2*i, 2*j-1, 2) -r (2*i, 2*j, 1) -30;
           b=b+1;
           sum_x_2=sum_x_2+j;
           sum_y_2=sum_y_2+i;
        end;
    end;
end;

target_x_2=floor (sum_x_2/b) ;
target_y_2=floor (sum_y_2/b) ;
target_x=floor (sum_x/a) ;
target_y=floor (sum_y/a) ;
position_x=2*floor (sum_x/a) ;
position_y=2*floor (sum_y/a) ;
subplot (1, 2, 1) ;
imshow (img) ;
title ('image1  G-B') ;
subplot (1, 2, 2) ;
```

```
imshow（img2）；
title（'image2　G-R'）；
dlmwrite（'image2_new_red_Int.txt'，double（r（:，:，1）），'\t'，0，0）；
dlmwrite（'image2_new_green_Int.txt'，double（r（:，:，2）），'\t'，0，0）；
dlmwrite（'image2_new_blue_Int.txt'，double（r（:，:，3）），'\t'，0，0）；
dlmwrite（'image2_bayer.txt'，double（r（:，:，3）+r（:，:，1）+r（:，:，2）），
'\t'，0，0）；
[X，Y]=meshgrid（1:1:N2，1:1:N1）；% N2 x-axis N1 y-axis
out_tmp=double（out）；
```

附录三 图像采集处理 IP 核顶层 RTL 描述

```
--------------------------------------------------------------------------------
-- Title      : Color Process Block
-- Project    : Machine Vision Project
--------------------------------------------------------------------------------
-- File       : cmos_color_processor.vhd
-- Author     : Omnihua
-- Company    : Department of Electrical and Automation
-- Last update : 2007/07/09
-- Platform   : modesim se plus 6.2b synplify 7.7 quartus II 7.1
--------------------------------------------------------------------------------
-- Description : This file is part of the project 'Machine Vision Project'
--------------------------------------------------------------------------------
-- Revisions  : BETA
-- Date       Version   Author      Description
-- 2007/07/09  0.9      Omnihua     Created
-- 2007/09/13  0.9.1    Omnihua     modify Add some timing description
--------------------------------------------------------------------------------

LIBRARY IEEE ;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

package my_pkg is
    TYPE ram_array IS ARRAY （0 TO 23） OF STD_LOGIC_VECTOR（7 downto
0）;
    SUBTYPE unsigned_char  is integer range 0 to 255;
    SUBTYPE unsigned_8bit  is integer range 0 to 255;
    SUBTYPE unsigned_9bit  is integer range 0 to 511;
    SUBTYPE unsigned_10bit is integer range 0 to 1023;
    constant PIXEL_WIDTH      : integer       := 10;
    constant DIVISOR_WIDTH    : integer       := 19;
    constant DIVISION_WIDTH   : integer       := 32;
    constant BIT_WIDTH_X      : integer       := 10;
    constant BIT_WIDTH_Y      : integer       := 9;
    constant HEIGHT_BIN       : std_logic_vector  := "111100000";
    constant WIDTH_BIN        : std_logic_vector  := "1011110000";
    constant HEIGHT           : integer       := 480;
    constant WIDTH            : integer       := 752;
    constant VALUE_ONE        : std_logic_vector  := "0000000001";
```

```
    constant DELAY              : integer           := WIDTH +4;
end my_pkg;

LIBRARY IEEE ;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.my_pkg.all;

entity cmos_color_processor is
port
 (
    pclk              : in  STD_LOGIC;
    line_valid        : in  STD_LOGIC;
    frame_valid       : in  STD_LOGIC;
    data_in           : in  std_logic_vector  (PIXEL_WIDTH     -1 downto 0) ;
    Avalon_clk        : in  std_logic;
    Avalon_reset_n    : in  std_logic;
    Avalon_address    : in  std_logic_vector  (1 downto 0) ;
    Avalon_en         : in  std_logic;
    Avalon_read       : in  std_logic;
    Avalon_data_out   : out std_logic_vector  (31 downto 0) ;
    Avalon_irq_out    : out std_logic --interrupt flag
) ;
end cmos_color_processor;

architecture behavior of cmos_color_processor is
    signal position_x       :std_logic_vector  (BIT_WIDTH_X     -1 downto 0) ;
    signal position_y       :std_logic_vector  (BIT_WIDTH_Y     -1 downto 0) ;
    signal target_x         :std_logic_vector  (BIT_WIDTH_X     -2 downto 0) ;
    signal target_y         :std_logic_vector  (BIT_WIDTH_Y     -2 downto 0) ;
    signal sum_x_connect    :std_logic_vector  (DIVISION_WIDTH -1 downto 0) ;
    signal sum_y_connect    :std_logic_vector  (DIVISION_WIDTH -1 downto 0) ;
    signal fit_num_connect:std_logic_vector  (DIVISOR_WIDTH   -1 downto 0) ;
    signal error            : std_logic;
    signal flag             : std_logic;
    signal flag_temp        : std_logic;
    signal irq_flag         : std_logic;
    signal irq_ack          : std_logic;
    signal Avalon_wacc      : std_logic;
    signal Avalon_racc      : std_logic;
```

```
    -- registers for Avalon read
    signal Avalon_status_reg      : std_logic_vector (31 downto 0) ;
    signal Avalon_target_x_reg    : std_logic_vector (31 downto 0) ;
    signal Avalon_target_y_reg    : std_logic_vector (31 downto 0) ;
--component color_process
component color_process is
port
 (
   pclk         :  in  STD_LOGIC;
    line_valid  :  in  STD_LOGIC;
    frame_valid :  in  STD_LOGIC;
    position_x  :  in  STD_LOGIC_VECTOR (BIT_WIDTH_X-1 downto 0) ;
    position_y  :  in  STD_LOGIC_VECTOR (BIT_WIDTH_Y-1 downto 0) ;
    data_in     :  in  std_logic_vector (PIXEL_WIDTH-1 downto 0) ;
    sum_x_out   :  out std_logic_vector (DIVISION_WIDTH-1 downto 0) ;
    sum_y_out   :  out std_logic_vector (DIVISION_WIDTH-1 downto 0) ;
    fit_add_out :  out std_logic_vector (DIVISOR_WIDTH-1 downto 0)
) ;
end component ;

component position_determination is
port
 (
    pclk       : in std_logic;
    line_valid : in std_logic;
    frame_valid: in std_logic;
    position_x : out std_logic_vector (BIT_WIDTH_X-1 downto 0) ;
    position_y : out std_logic_vector (BIT_WIDTH_Y-1 downto 0)
) ;
end component;
component target_position_confirm is
port
 (
    pclk        : in  std_logic;
    position_x  : in  std_logic_vector (BIT_WIDTH_X-1   downto 0) ;
    position_y  : in  std_logic_vector (BIT_WIDTH_Y-1   downto 0) ;
    sum_x       : in  std_logic_vector (DIVISION_WIDTH-1 downto 0) ;
    sum_y       : in  std_logic_vector (DIVISION_WIDTH-1 downto 0) ;
    fit_num     : in  std_logic_vector (DIVISOR_WIDTH-1  downto 0) ;
    target_x    : out std_logic_vector (BIT_WIDTH_X-2    downto 0) ;
    target_y    : out std_logic_vector (BIT_WIDTH_Y-2    downto 0) ;
```

```
    flag          : out std_logic    -- give a flag after division complete
) ;
end component;
begin
--singal connection
connect_position_determination:
position_determination
port map (pclk => pclk, frame_valid => frame_valid, line_valid =>line_valid,
          position_x=>position_x, position_y=>position_y) ;
connect_color_process:
color_process
port map (pclk => pclk, frame_valid => frame_valid, line_valid =>line_valid,
position_x=>position_x, position_y=>position_y, data_in => data_in,
sum_x_out=>sum_x_connect , sum_y_out=>sum_y_connect , fit_add_out
=>fit_num_connect) ;

connect_target_position_confirm:
target_position_confirm
port map (pclk=>pclk, position_x=>position_x, position_y=>position_y,
sum_x=>sum_x_connect, sum_y=>sum_y_connect, fit_num=>fit_num_connect,
target_x=>target_x, target_y=>target_y, flag=>flag) ;
--generate write access signal
--Avalon_wacc<= Avalon_en and Avalon_we;
--generate read  access signal
Avalon_racc<= Avalon_en and Avalon_read;
process (Avalon_clk)
begin
    if (Avalon_clk'event and Avalon_clk ='1') then
       irq_ack<='1';
    if Avalon_racc ='1' then
       irq_ack<='0';
   case Avalon_address is
        when "00" =>Avalon_data_out<=Avalon_target_x_reg;
        when "01" =>Avalon_data_out<=Avalon_target_y_reg;
        when "10" =>Avalon_data_out<=Avalon_status_reg;
        when "11" =>Avalon_data_out<= (others=>'0') ;
        when others => Avalon_data_out<= (others=>'X') ;
   end case;
   end if;
   end if;
end process;
--generate interrupt flag automatic clear flag when read;
```

```
process（Avalon_clk）
begin
     if Avalon_clk'event and Avalon_clk='1' then
          if（Avalon_reset_n = '0' or irq_ack='0'） then
     irq_flag<='0';
     else
          flag_temp<=flag;
     if（flag='1' and flag_temp='0'） then
     irq_flag<='1';
     end if;
     end if;
    end if;
end process;
process（Avalon_clk）
begin
     if Avalon_clk'event and Avalon_clk='1' then
          if（Avalon_reset_n = '0'） then     --synchronous clear
     Avalon_irq_out<='0';
     else
          Avalon_irq_out<=irq_flag;
     end if;
    end if;
end process;
--assign the registers
Avalon_status_reg（0）<=irq_flag;
Avalon_status_reg（31 downto 1）<=（others=>'0'）;
Avalon_target_x_reg（BIT_WIDTH_X-2 downto 0）<=target_x（BIT_WIDTH_X-2
downto 0）;
Avalon_target_x_reg（31 downto BIT_WIDTH_X-1）<=（others =>'0'）;
Avalon_target_y_reg（BIT_WIDTH_Y-2 downto 0）<=target_y（BIT_WIDTH_Y-2
downto 0）;
Avalon_target_y_reg（31 downto BIT_WIDTH_Y-1）<=（others =>'0'）;
end behavior; --architecture
```

附录四 图像并口通讯顶层 RTL 描述

```
--------------------------------------------------------------------------------
-- Title      : Image EPP Test Program
-- Project    : Machine Vision Project
--------------------------------------------------------------------------------
-- File       : Epp2Sram.vhd
-- Author     : Omnihua
-- Company    : Department of Electrical and Automation
-- Last update : 2007/09/09
-- Platform   : modesim se plus 6.2b synplify 7.7 quartus II 7.1
--------------------------------------------------------------------------------
-- Description : This file is part of the project 'Machine Vision Project'
--------------------------------------------------------------------------------
-- Revisions  : BETA
-- Date       Version   Author      Description
-- 2007/09/09  0.9      Omnihua     Created
--------------------------------------------------------------------------------
LIBRARY IEEE ;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.my_pkg.all;

entity image2sram is
port
 (
    --MT9V032 interfaces
    pclk,  frame_valid,  line_valid : in std_logic;
    pixeldata                       : in std_logic_vector（7 downto 0）;
    exposure                        : out std_logic;
    -- key control
    key_capture                      :in    std_logic;
    led                              :out   std_logic_vector（7 downto 0）;
    -- sram control signals;
    nwe                               :out   std_logic;
    noe                               :out   std_logic;
    nce                               :out   std_logic;
    addr                              :out   std_logic_vector（18 downto 0）;
    data                              :inout std_logic_vector（7 downto 0）;
    -- EPP interfaces;
    EPP_Write                         :in    std_logic;
```

```
    EPP_AddressStrobe          :in     std_logic;
    EPP_DataStrobe             :in     std_logic;
    EPP_Reset                  :in     std_logic;
    EPP_Data                   :inout  std_logic_vector (7 downto 0) ;
    EPP_Wait                   :out    std_logic;
    EPP_Dir                    :out    std_logic;
    -- system reset signal
    reset                      :in     std_logic;   --high active
    --i2c interface
    I2C_Sda                    :inout  std_logic;
    I2C_Scl                    :inout  std_logic;
    I2C_Write                  :in     std_logic
) ;
end image2sram;

architecture behavior of image2sram is
    type states is (S_IDLE, S_FRAME, S_TRANSFER, S_WAIT) ;
    type epp_states is (EPP_IDLE, EPP_WAIT_WRITE, EPP_WAIT_READ0,
EPP_READ1, EPP_WAIT_READ1) ;
     signal state: states;
     signal epp_state:epp_states;
    signal buffer1, buffer2: std_logic_vector (7  downto 0) ;
    signal pixel_counter          : std_logic_vector (18 downto 0) ;
    signal position_x             : std_logic_vector (BIT_WIDTH_X-1 downto
0) ;
     signal position_y            : std_logic_vector (BIT_WIDTH_Y-1 downto
0) ;
    signal frame_valid_tmp        : std_logic;
    -- sram signals
    signal sram_nwe               : std_logic;
    signal sram_noe               : std_logic;
    signal SRAM_Data              : std_logic_vector (7 downto 0) ;
    signal SRAM_DataIn            : std_logic_vector (7 downto 0) ;
    signal SRAM_RAddr             : std_logic_vector (18 downto 0) ; --256K 32bit
    signal SRAM_WAddr             : std_logic_vector (18 downto 0) ;
    signal SRAM_RAddrTmp          : std_logic_vector (18 downto 0) ;
    --EPP siganls
    signal EPP_Wr                 : std_logic;
    signal EPP_AS                 : std_logic;
    signal EPP_DS                 : std_logic;
    signal EPP_Rst                : std_logic;
```

```
signal EPP_DataOut                : std_logic_vector (7 downto 0) ;
signal EPP_DataIn                 : std_logic_vector (7 downto 0) ;
signal EPP_DataTmp                : std_logic_vector (7 downto 0) ;
--I2C signals
signal I2C_Led                    : std_logic_vector (7 downto 0) ;
signal error                      : std_logic;


begin
u2:     image_position
port map (pclk => pclk, frame_valid => frame_valid, line_valid =>line_valid,
      position_x=>position_x, position_y=>position_y) ;

u3:     i2c_interface
port map (clk=>pclk, Reset =>Reset, led=>I2C_led, key_start=>'0',
key_write=>I2C_Write, key_disp=>'0', key_nxt=>'0', error=>error,
SCL=>I2C_Scl, SDA=>I2C_Sda) ;
 --i2c can work with sample concurrently

process (pclk, reset)
     begin
     if reset ='1' then
     EPP_Wr<='1';
     elsif pclk'event and pclk='1' then
          EPP_Wr<=EPP_Write;
     end if;
end process;

process (pclk, reset)
     begin
     if reset ='1' then
     EPP_AS<='1';
     elsif pclk'event and pclk='1' then
          EPP_AS<=EPP_AddressStrobe;
     end if;
end process;

process (pclk, reset)
     begin
     if reset ='1' then
     EPP_DS<='1';
     elsif pclk'event and pclk='1' then
```

```
            EPP_DS<=EPP_DataStrobe;
        end if;
end process;

process (pclk, reset)
        begin
        if reset ='1' then
        EPP_Rst<='1';
        elsif pclk'event and pclk='1' then
            EPP_Rst<=EPP_Reset;
        end if;
end process;

--Main state shift program
process (reset, pclk, line_valid, frame_valid)
begin
if reset ='1' then
    state<= S_IDLE;
    epp_state<=EPP_IDLE;
    led<=X"11";       --state dictator
    sram_nwe<='1'; -- disable sram
    sram_noe<='1';
     SRAM_RAddr<= (others=>'0') ;
     SRAM_WAddr<= (others=>'0') ;
     SRAM_RAddrTmp<= (others =>'0') ;
     SRAM_DataIn<= (others =>'0') ;
     SRAM_Data<= (others =>'0') ;
     pixel_counter<= (others =>'0') ;
     epp_dataout<= (others =>'0') ;
     EPP_DataIn<= (others =>'0') ;
     EPP_DataTmp<= (others=>'0') ;
     buffer1<= (others=>'0') ;
     frame_valid_tmp<='0';
     epp_wait<='0';
     EPP_Dir<='0';   -- data from epp to fpga

elsif pclk'event and pclk ='1' then
        if line_valid='0' then
            buffer1<= (others=>'0') ;
           buffer2<= (others=>'0') ;
       end if;
```

```
        if frame_valid='0' then
           pixel_counter<= (others=>'0') ;
        end if;
        case state  is
        when S_IDLE  =>
              state<=S_IDLE;
              EPP_Dir<='0';
              if key_capture ='1' then
                  state<=S_WAIT;
                  SRAM_RAddr<= (others =>'0') ; --clear Read Register
               end if;
        when S_WAIT => --wait for a new frame
              frame_valid_tmp<=frame_valid;
              if frame_valid ='1' and frame_valid_tmp ='0' then

                  state<=S_FRAME;
                  SRAM_Data (7 downto 0) <=pixeldata;
                  pixel_counter<=pixel_counter+'1';
              end if;
-- save the pixel data to sram
        when S_FRAME => --frame time
              if line_valid ='1'  then
                 sram_noe<='1';
                  sram_nwe<='1';
                 SRAM_WAddr<=pixel_counter (18 downto 0) ;
                 SRAM_Data (7 downto 0) <=pixeldata;
                 pixel_counter<=pixel_counter+'1';
              end if;
              if pixel_counter =X"58200" then   --40000 =256k --1067F ==
                  state<=S_IDLE;
                sram_noe<='0';  -- enable sram output data
                  sram_nwe<='1';
                  state<=S_TRANSFER;
                  led<=X"33";
                EPP_Dir<='1';
              end if;
        -- transfer the pixel data from sram to epp
        when S_TRANSFER =>
              --EPP_Dir<='1';   -- read sram
              state<=S_TRANSFER;
              if  (SRAM_RAddr =X"58200")  then
```

```
                state<= S_IDLE;
                led<=X"44";
            end if;
            case（epp_state） is
            when EPP_IDLE =>
                epp_state<=EPP_IDLE;
                EPP_Wait<='0';
               if EPP_DS ='0' then
                if EPP_Wr ='1' then -- read sram
                    EPP_Wait<='1';
                    EPP_DataOut<=data（7 downto 0）;
                    epp_state<=EPP_WAIT_READ0;
                else                        -- write sram
                    EPP_Wait<='1';
                    EPP_DataIn<=EPP_Data;
                    epp_state<=EPP_WAIT_WRITE;
                end if;
                end if;
            when EPP_WAIT_WRITE => -- write sram
                if EPP_DS ='1' then
                    EPP_Wait<='0';
                    epp_state<=EPP_IDLE;
                end if;
             -- Begin transfering data from sram to epp
             when EPP_WAIT_READ0 =>
                if EPP_DS = '1' then
                    EPP_Wait<='0';
                    epp_state<= EPP_IDLE;
                    SRAM_RAddr<=SRAM_RAddr +'1';
                 led<=SRAM_RAddr（7 downto 0）;
                end if;
            when others =>
            epp_state<=EPP_IDLE;
            end case;
when others =>
            state<=S_IDLE;
 end case;
 end if;
end process;
-- assign epp  signals ,  data from sram to epp
EPP_Data<=EPP_Dataout when（epp_state =EPP_WAIT_READ0 or epp_state
```

```
=EPP_WAIT_READ1)  else  (others =>'Z') ;
     -- assign mt9v032 signals
     exposure<='0';
     -- assign sram signals
     nwe<=  (not pclk)  when (state = S_FRAME )  else '1';
     noe<=sram_noe;
     --addr<=SRAM_WAddr when  (sram_nwe ='0')  else  (SRAM_RAddr) ;
     addr<=SRAM_WAddr when  (state=S_FRAME )  else  (SRAM_RAddr) ;
     data<=SRAM_Data when  (state=S_FRAME ) else  (others=>'Z') ;
     nce<='0';
 end behavior;
```

附录五 Visual Studio 下图像显示程序

```
void CEPPandLEDDlg::OnBnClickedButton3（）//显示图像
{
    CString text_data;
    int i，j;
    BYTE addr=0;
    BYTE tmp=0;
    BYTE buf[240][376]={0，};
    unsigned int count_times=0;
     //reset epp
    DlPortWritePortUchar（PORTBASE+2，addmask（PORTBASE+2，0x0F））；
    for（i=0；i<240；i++）
     for（j=0；j<376；j++）
     {
         buf[i][j]=DlPortReadPortUchar（PORTBASE+4）；
         ount_times+=1;
     }
    // display the image
    CWnd*    pPic=this->GetDlgItem（IDC_IMG）；
    ASSERT（pPic）；
    CClientDC    dc（pPic）；
    CRect    rect;
    pPic->GetClientRect（rect）；
    CBitmap bitmap;
    bitmap.CreateCompatibleBitmap（&dc，rect.Width（），rect.Height（））；
    CDC m_dc;
    m_dc.CreateCompatibleDC（&dc）；
    m_dc.SelectObject（&bitmap）；
    //draw rect
    m_dc.Draw3dRect（rect，RGB（0，0，255），RGB（255，255，0））；
    CPen GreenPen（PS_SOLID，0，RGB（0，255，0））；
    for（i=0；i<240；i++）//bayer display
```

```
    for (j=0; j<376; j++)
    {
        if ( (i%2==0) && (j%2==1) ) //奇数行，偶数列 G
        {
            m_dc.SetPixel (j, i, RGB (0x0, buf[i][j], 0x0) );
        }
        else if ( (i%2 ==1) && (j%2==1) ) //偶数行，偶数列 R
        {
            m_dc.SetPixel (j, i, RGB (buf[i][j], 0x0, 0x0) );
        }
        else if ( (i%2==0) && (j%2==0) ) //奇数行，奇数列 B
        {
            m_dc.SetPixel (j, i, RGB (0x0, 0x0, buf[i][j]) );
        }
        else//偶数行，奇数列 G
        {
            m_dc.SetPixel (j, i, RGB (0x0, buf[i][j], 0x0) );
        }
    }
    dc.BitBlt(0, 0, rect.right-rect.left, rect.bottom-rect.top, &m_dc, 0, 0, SRCCOPY);
    UpdateData (false);
}
```

攻读学位期间公开发表论文

[1]彭周华，樊印海.<<Three-Phase SPWM Wave Generator for Frequency Convertor Based On DSP >>.电气技术（增刊），ISSN 1673-3800，CN11-5255/TM，2006，第六届国际船舶电工技术学术会议。

致　谢

感谢我的导师樊印海教授，在学习和生活中给予我启发和照顾！

感谢审阅老师腾出时间审阅我的论文！

感谢我的父母和朋友，这些年来给予我的一贯支持和关心！

感谢所有我的同学和师弟师妹们，感谢他们对我的鼓励和帮助！

最后，感谢所有答辩老师的辛勤劳动！

研究生履历

姓　　名	彭周华
性　　别	男
出生日期	1982年2月23日
获学士学位专业及门类	工业自动化 工学学士
获学士学位单位	大连海事大学
获硕士学位专业及门类	电力电子与电力传动 工学硕士
获硕士学位单位	大连海事大学
通信地址	辽宁省大连市凌海路1号
邮政编码	116026
电子邮箱	omnihua@163.com