郑州大学毕业设计（论文）

题　　目：基于 FPGA 的信息显示滚动屏

指导教师：范文兵　　职称：教授

学生姓名：郭广昊　　学号：20132410105

专　　业：电子信息工程

院（系）：信息工程学院

完成时间：2017 年 5 月 15 日

2017 年　5 月 15 日

基于 FPGA 的 LED 信息显示滚动屏

摘要：本篇论文介绍了数字温度计集成芯片 DS18B20 的特点和使用方法，并概述了温度测量模块硬件和软件的设计。除此之外，还介绍了环境光传感器 BH1750FVI 的原理和使用方法，以及亮度显示模块的硬件和软件设计。本次设计把 FPGA 作为主要控制芯片，驱动包括环境光传感器，温度传感器在内的两个传感器。把采集的数据通过 FPGA 处理，并把处理后的数据显示到 LED 点阵上。环境光传感器 BH1750FVI 采用 I2C 通信方式，温度传感器 ds18b20 则采用单总线驱动方式，需要根据不同的通信协议来设计程序驱动这两种传感器。

本次设计由三部分组成：FPGA 核心板、底座和显示功能板。使用的核心板 MXO2 二代是以拥有 40 个管脚的 fpga 芯片作为主控芯片的开发板。核心 FPGA 芯片选用了 Lattice 公司 MXO2 系列的 4000HC 产品。同时核心板上也包含有很多外设资源，包括 8 个 LED 灯、4 个拨码开关，2 个触摸按键，2 个数码管，2 个三色灯和 1 个下载接口。板上有 36 个可供使用的通用 IO 接口。板卡尺寸为 52mm X 18mm，通过两排插针嵌入到底座。底座可实现把 40 脚的核心板转换成 52 脚正反两面的插槽形式，方便与显示功能板的连接。显示功能板作为 FPGA 核心板的扩展板卡，其中包含有 16*8 的 LED 点阵，flash 芯片，红外对管和多种传感器。

整个设计以及各个模块都在 lattice 公司的开发环境 Diamond 中进行，可以编写测试软件进行仿真并下载到硬件实物上观察代码运行情况。本次设计的系统不仅测量精度高误差较小，而且抗干扰能力强、工作状态稳定，能在大多数环境下保持正常工作。

关键词：FPGA；数字传感器;DS18B20；BH1750FVI；温度；亮度

Abstract： This paper introduces the characteristics and basic principles of the 1-wire digital thermometer DS18B20 and describes the hardware and software design of the FPGA and DS18B20 temperature measurement system. In addition, the principle and use of ambient light sensor BH1750FVI and the hardware and software design of brightness display module are introduced. FPGA is designed as the main control chip, driving two sensors including the ambient light sensor, temperature sensor. The collected data from sensors can be processed by FPGA, and the processed data is displayed on the LED dot matrix. BH1750FVI is an digital ambient light sensor IC for I2C bus interface,while the temperature sensor ds18b20 is a unique 1-wire interface requiring only one pin for communication. Different communication protocol should be taken into consideration when designing the program to drive these sensors.

This design consists of three parts: FPGA core board, base and display function board. The core board which is STEP-MXO2 of second generation is a compact 40-pin DIP structure of the FPGA development board. Core FPGA chip select 4000HC products of MXO2 series produced by Lattice.At the same time the core board also contains a lot of peripheral resources, including 8 LED lights, 4 DIP switch, 2 touch button, 2 digital tube, 2 three-color lights and a download interface.There are 36 available general IO interfaces on the board. Board size is 52mm X 18mm, embedded in the base through the two rows of pins. The base can be achieved to 40 feet of the core board into 52 feet both sides of the form of slot,The function of the base is to converse 40 feet into 52 feet in the form of the slot, which is easy to connect with the display function board. Display function board is an expansion of FPGA core board , which contains 16 * 8 LED dot matrix, flash chip, infrared tube and a variety of sensors.

The entire design and the various modules are programmed in Diamond which is a software designed by Lattice.Test software can be wrote to simulate the code and it is also an option to download the code into the FPGA to observe the code operation in specific products. The entire system is characterized by its precise measurement and anti-interference ability.Apart from that, working condition is stable, in most circumstances to maintain normal work.

Key words:FPGA; digital sensor; DS18B20; BH1750FVI; temperature; brightness

目录

1 绪论

1.1 课题背景及意义

FPGA，是英文 Field Programmable Gate Array 的缩写, 即现场可编程门阵列。它是作为专用集成电路领域中的一种半定制电路而出现的，一方面解决了全定制电路局限性太强、设计周期较长的不足，又克服了原有可编程逻辑器件门电路数有限，无法完成复杂需求的缺点。FPGA 的特点主要有编译完的程序是以门电路的形式存在在芯片上，随时可以修改程序来纠正错误，因此可以降低成本。擅长并行高速数字信号处理，也可以嵌入各类软核，执行软件代码。FPGA 相应的也有一些缺点，比 ASIC（专用集成芯片）的速度慢，无法完成复杂的设计，而且消耗更多的电力。

FPGA 主要有四大供货商商，分别是 Xilinx 、Altera、Lattice 和 Microsemi。本次设计所用的芯片是 lattice 厂商生产的 MXO2 系列芯片，lattice 公司提供范围广泛的现场可编程门阵列(FPGA)，芯片门类齐全而且成本相比 xilinx，altera 较低。Lattice 公司的软件开发环境为 Diamond，可以去 Lattice 公司的官网下载，除此之外软件比较容易操作，上手简单，适合于新手入门学习。

温度是最常测量的模拟参数之一，这很好理解，因为大多数电气，化学，机械和环境系统直接受温度影响或者，把温度作为参考量去控制其他相关过程。测量温度就不可避免地会用到温度传感器，温度传感器有很多种，比如 RTD、热电偶、热敏电阻、恒温器、固态传感器、IC 传感器，需要根据不同的应用场景和要求选择不同种类的传感器。最终决定选择 IC 传感器，因为 IC 传感器芯片资料很多，而且成熟的设计方案较多。市面上广泛使用的主要有两种芯片，一种是 Sensirion 公司生产的 sht20 芯片；另一种是美信公司生产的 ds18b20 芯片。Sht20 是 I2C 的通信方式，会占用两个芯片管脚；相对应的 ds18b20 是单总线的通信方式，只会占用一个芯片管脚。因为 FPGA 有 36 个可供使用的通用 I/O 口，选用 ds18b20 会占用相对少的芯片管脚资源。除此之外，sht20 为 QFN 的封装，而 ds18b20 为 SOP8 的封装，容易焊接且芯片较小，综上所述，选用 ds18b20 最为合适。

光也是本次设计的重要参数值之一，测量光的常见方法是利用光敏器件采集光学信号，并把模拟量转换成数字量，再把数字量传到 mcu 中进行处理。光落在传感器上，其中光子的能量转换成电荷。撞击表面的光线越多，电荷越多。一般来说，两者是相关的。光敏器件的作用就是把外界光强信号转换成电信号。感光模块也同样有两种设计方案，一种是采用光敏电阻配合 AD 转换器实现，另一种是直接采用集成芯片 BH1750FVI 实现，可以直接实现光强信号向数字信号的转换。最终选择使用集成芯片 BH1750FVI, 一是成本较少，而是因为

BH1750FVI 受红外线影响较小，光源在产生光线的同时也会产生少量的红外线可能会对亮度的测量产生一定影响。BH1750FVI 是 I2C 的通信方式，I2C（Inter－Integrated Circuit）总线是由 PHILIPS 公司开发的两线式串行总线，用于连接微控制器及其外围设备。是微电子通信控制领域广泛采用的一种总线标准。它是同步通信的一种特殊形式，具有接口线少，控制方式简单，器件封装形式小，通信速率较高等优点。I2C 总线支持任何 IC 生产工艺(CMOS、双极型）。通过串行数据（SDA）线和串行时钟 （SCL）线在连接到总线的器件间传递信息。每个器件都有一个唯一的地址识别（无论是微控制器——MCU、LCD 驱动器、存储器或键盘接口），而且都可以作为一个发送器或接收器（由器件的功能决定）。由此可知，需要两个 I/O 总线来驱动这个芯片。

1.2 实现功能概述

1、模式一为温度显示模式，利用温度传感器 DS18b20 采集环境温度。温度不仅会显示在 FPGA 核心板的数码管上，还会显示在扩展板的 16*8LED 点阵上，可检测的温度范围满足日常生活环境的需要。

2、模式二为亮度显示模式，将所处环境光亮度量化分级为 64 级，用 1 到 64 共 64 个数字代表环境光的强度，1 代表光线最弱，64 代表光线最强。这个亮度值也有三种显示方式，同样可以显示在 FPGA 核心板的数码管上；FPGA 核心板上的三色灯也会随外界亮度的变化进行同步的变化，外界光线越强三色灯亮度越强；在 LED 点阵上则以箭头流动快慢的形式表现出来，亮度越高箭头流动越快，反之亦反。

3、模式一的功能可类比成温度计，用于检测周围环境的温度。模式二相当于一个环境光强检测仪，与手机中自动调节屏幕亮度的功能相似。

2 系统主要器件介绍

2.1 FPGA 核心开发板

2.1.1 概述

FPGA 核心开发板 STEP-MXO2 二代是一款超小巧 40 脚 DIP 结构的 FPGA 开发板。核心 FPGA 芯片 选用了 Lattice 公司 MXO2 系列的 4000HC 产品，相比于第一代 STEP-MXO2，板上的 FPGA 芯片资源提升了 4 倍。同时板上集成了 FT232 编程器和按键、拨码开关、数码管、LED 等多种外设资源。板上的 36 个 FPGA IO 接口都通过 2.54mm 通孔焊盘引出，可以和面包板 配合使用。板卡尺寸为 52mm X 18mm，能够灵活的嵌入到插座或者其他的系统中。 STEP-MXO2 二代板上集成的编程器能够完美支持 Lattice 工具 Diamond，只需要一根 USB 链接线就能够完成 FPGA 的编程仿真和下载，使用更加方便。

核心芯片型号：Lattice LCMXO2-4000HC-4MG132

2.1.2 芯片的主要特点

1、4320 个 LUT（查找表）资源；

2、有可供使用的 96Kbit User Flash，92Kbit RAM；

3、2+2 路 PLL+DLL；PLL 即 phase lock loop 主要根据输入时钟产生一个与输入时钟信号 in phase 的倍/除频时钟。DLL 即 delay lock loop 主要用于产生一个精准的时间延时，且这个不随外界条件如温度、电压的变化而变化；

4、嵌入式功能块（硬核）：一路 SPI、一路定时器、2 路 I2C 。扩展板上的 flash 是 SPI 的通信方式，环境光传感器 BH1750fvi 和陀螺仪 mpu6050 是 I2C 的通信方式，刚好能满足本次设计所需要的硬件资源；

5、支持 DDR/DDR2/LPDDR 存储器；

6、上电瞬时启动，启动时间<1ms；

7、板载资源（核心板上其他器件）包括：

1 路 Micro USB 接口；

2 位 7 段数码管；

2 个 RGB 三色 LED；

4 路拨码开关；

4 路按键；

8 路用户 LED；

36 个用户可扩展 I/O 口（其中包括一路 SPI 硬核接口和一路 I2C 硬核接

口）；

集成 FT232 编程器；

2.2 温度传感器

2.2.1 概述

DS18B20 数字温度计提供 9 到 12 位（可配置）温度读数，表明器件的温度。信息通过 1-Wire 接口发送到 DS18B20，从而只需要一根线（和地）从中央微处理器连接到 DS18B20。读取，写入和执行温度转换的功能可以从数据线本身导出，无需外部电源。它的测温范围为-55～＋125℃，并且在-10～＋85℃精度为±0.5℃。可以直接从数据线上汲取能量，供电电压为 3v 到 5.5v。

DS18B20 的核心功能是其直接数字温度传感器。DS18B20 的分辨率可配置（9,10,11 或 12 位），12 位读数为出厂默认状态。9、10、11 或 12 位的数据位等效于 0.5℃，0.25℃，0.125℃或 0.0625℃的温度分辨率。在发出 Convert T [44h]命令之后，执行温度转换，数据以 16 位，符号扩展的二进制格式存储在暂存器中。一旦执行了转换，就可以通过发出 Read Scratchpad [BEh]命令，通过 1-Wire 接口检索温度信息。 数据通过 1-Wire 总线传输，LSB 优先。温度寄存器的 MSB 包含“符号”（S）位，表示温度是正还是负。

FPGA 通过 1 线端口与 DS18B20 通讯。使用 1-Wire 端口，在 ROM 功能协议建立之前，所有的存储器和控制功能将不可用。主机必须首先提供五个 ROM 功能命令之一：1）读 ROM，2）匹配 ROM，3）搜索 ROM，4）跳过 ROM 或 5）报警搜索。这些命令在每个器件的 64 位激光 ROM 部分上运行，如果有多个存在于 1-Wire 线路上，并且可以向总线主机指示存在多少种类型的器件，则可以将特定器件单独使用。 在 ROM 功能序列成功执行之后，可以访问存储器和控制功能，然后主机可以提供六个存储器和控制功能命令中的任何一个。

2.2.2 温度与输出数据的对应关系

下表描述了输出数据与测量温度的确切关系。该表假设为 12 位分辨率。如果 DS18B20 配置为较低分辨率，则无效位将包含零。对于华氏度使用，必须使用查找表或转换程序。

2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	LSB
MSb			(unit = °C)				LSb	
S	S	S	S	S	2^6	2^5	2^4	MSB

TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+125°C	0000 0111 1101 0000	07D0h
+85°C	0000 0101 0101 0000	0550h*
+25.0625°C	0000 0001 1001 0001	0191h
+10.125°C	0000 0000 1010 0010	00A2h
+0.5°C	0000 0000 0000 1000	0008h
0°C	0000 0000 0000 0000	0000h
-0.5°C	1111 1111 1111 1000	FFF8h
-10.125°C	1111 1111 0101 1110	FF5Eh
-25.0625°C	1111 1110 0110 1111	FF6Fh
-55°C	1111 1100 1001 0000	FC90h

*The power on reset register value is +85°C.

图 2. 2. 1 ds18b20 温度和二进制数据的对应关系

2.2.3 DS18B20 程序指令

SKIP ROM [CCh] (忽略 ROM 指令) :

该命令可以通过允许总线主机在不提供 64 位 ROM 代码的情况下访问存储器功能来节省单个总线系统中的时间。如果总线上有多个从器件，并且在 Skip ROM 命令之后发出 Read 命令，则当多个从器件同时发送时，总线上将发生数据冲突（开漏下拉将产生有线 AND 结果）。

CONVERT T [44h] (温度转换指令) :

此命令开始温度转换，不需要进一步的数据。将执行温度转换，然后 DS18B20 将保持空闲状态。如果总线主机根据该命令发出读取时隙，只要正忙于进行温度转换，DS18B20 将在总线上输出 0; 当温度转换完成时，它将返回 1。如果处在寄生模式下，总线主控必须在紧随其后的一段时间内启用强大的上拉电压发出此命令。

READ SCRATCHPAD [BEh] （读暂存器指令）:

此命令可以读取暂存器的内容。 读取将从字节 0 开始，并将继续存储在暂存器中，直到读取第九个（字节 8，CRC）字节。 如果不是所有的位置被读取，主人可以随时发出重置以终止读取。

2. 3 环境光传感器

2.3.1 综述

BH1750FVI 是用于 I2C 总线接口的数字环境光传感器 IC。该 IC 最适用于获取手机 LCD 和键盘背光功率的环境光数据。可以在高分辨率下检测宽范围(1 - 65535 lx）。BH1750FVI 支持 I2C 的通信方式，有接近视觉灵敏度的光谱灵敏度并且对应广泛的输入光范围(相当于 1-65535lx)，并可以把光照的强度的模拟量转换成数字量。精确度较高，最小误差变动在 20%，且受红外线的影响较小，可以减少周围光源发出的红外线对测量结果的影响。而且 BH1750FVI 的光源依赖性弱，可以分辨包括白炽灯，荧光灯，卤素灯，白光 LED，日光灯等多种光源发出的光线。

2.3.2 芯片各管脚的功能及描述

1、Vcc：电源端口。

2、ADDR：I2C 地址端口，ADDR 端口是为了缓冲内部测试的三种状态而设计的，所以注意 Vcc 和 DVI 支持程序。

3、GND：接地端口。

4、SDA：I2C 接口的 SDA 端口。

5、DVI：SDA、SCL 端口的参考电压，DVI 端口为内部寄存器的异步重置端口，所以在内部供电以后设置“L”（至少 1us，DVI<0.4V）。当 DVI=“L”时，BH1750 被 150K 的电阻下拉。

6、SCL：I2C 接口的 SCL 端口。

2.3.3 I2C 总线通路（编程时可参考）

1、I2C 总线的接口时序图

写测量指令和读测量结果指令都是由 I2C 总线接口完成的

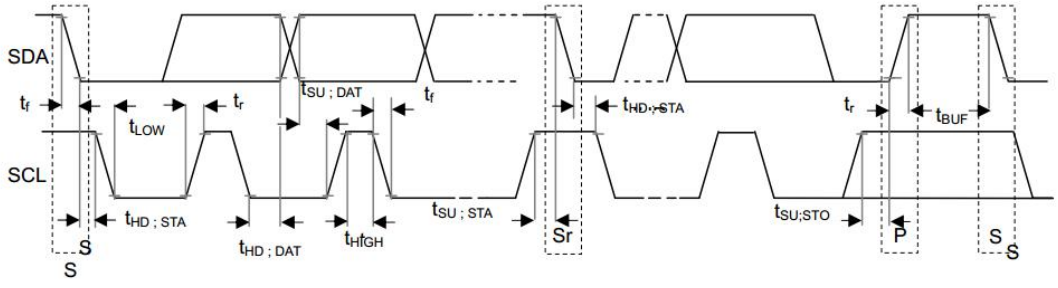


图 2.3.1 BH1750FVI 的通信时序图

2、从属地址

从属地址有 2 中形式，由 ADDR 端口决定。

ADDR=“H”（ ADDR ≧ 0.7VCC ）→“1011100”

ADDR=“L”（ ADDR ≦ 0.3VCC ）→“0100011”

3、写格式

BH1750FVI 不能在停机状态下接受复数指令，请在每个 opecode 后插入 SP。

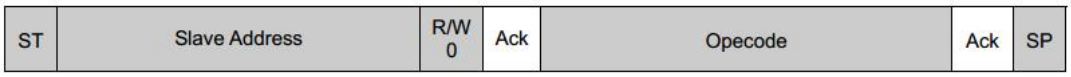
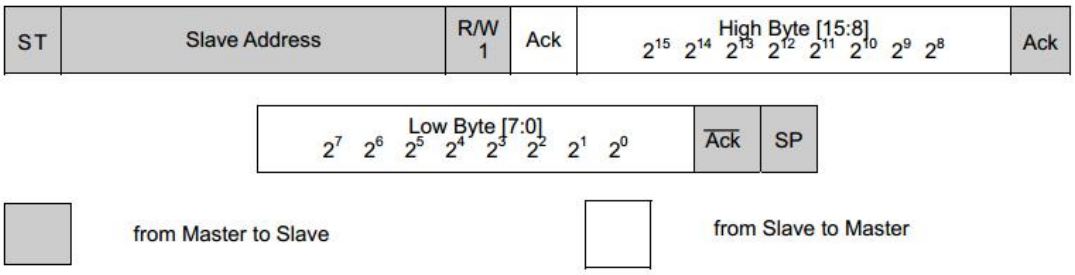


图 2.3.2 BH1750FVI 的写格式

4、读格式



（注释：阴影部分代表从主到从，白色部分代表从到主的通信）

图 2.3.3 BH1750FVI 的读格式

2.4 flash

2.4.1 综述

该 W25Q64BV（64M 位）串行 Flash 存储器用有限的空间，引脚和电源提供了系统存储解决方案。25Q 系列提供了超越普通串行闪存设备的灵活性和性能。 该器件工作于单个 2.7V 至 3.6V 电源，电流消耗低至 4mA 有效，1μA 用于掉电。 所有设备均采用节省空间的封装。

该 W25Q64BV 阵列是由可编程的 32,768 区域组成，每个区域包含 256 字节。最多 256 个字节可以在一个时间被编程。存储区域可以以 16 个为一组擦除（即每次擦除 4kb），存储区域也可以以 128 个为一组擦除（即每次擦除 32kb），存储区域还可以以 256 个为一组擦除（即每次擦除 64kb），甚至可以整个芯片一次性擦除。该 W25Q64BV 分别有 2,048 可擦除扇区和 128 可擦除块。小的 4KB 扇区在需要的数据和参数的存储的应用程序中允许更大的灵活性。

该 W25Q64BV 支持标准串行外设接口（SPI）。可以支持高达 80MHz 的 SPI 时钟频率，当使用双 I/O 口时，允许 160MHz 的等效时钟频率；当使用四 I/O 口时，允许 320MHz 的等效时钟频率。这些传输速率比传统的片上存储器速率要快的多。连续的读数据模式读取 24 位的地址时，通常只需要 8 个时钟周

期，非常快捷和高效。

2.4.2 芯片管脚定义

管脚名称	输入/输出	功能定义
CLK	输入	连续时钟输入
GND		接地
VCC		接电源
/CS	输入	片选输入
/WP	输入/输出	写保护输入
DO	输入/输出	数据输出
DI	输入/输出	数据输入
/HOLD	输入/输出	输入保持

表 2.4 W25Q64 引脚定义

2.4.3 部分芯片管脚详细描述

1.芯片选择（/ CS）

该 SPI 片选（/ CS）引脚可以控制设备的启用和禁用。因为/CS 端是低电平有效，所以当/CS 在高电平的状态下，器件不会运行，同时串行输入的通用 I/O 口处于高阻抗的状态。并且当/CS 为高电平时，设备的功率消耗维持在较低的水平下，设备处在编程或状态寄存器的周期之中。当/CS 有效即为低电平时，该设备会被选中。电力的消耗会加大至活跃水平。可以从设备中读出或写入数据。上电后，/CS 指令从高过渡到低，然后才能接受新的指令。该/CS 输入必须与上电时的电源电平保持一致。如果需要，/ CS 上的上拉电阻可以被用来实现此目的。

2.串行数据输入，输出和 IO（DI，DO 和 IO0，IO1，IO2，IO3）

该 W25Q64BV 支持标准的 SPI，双 SPI 和四路 SPI 操作。在时钟（CLK）上升沿，标准的 SPI 通信方式在写指令和数据到设备上时使用 单向 DI（输入）引脚，并且采用串行的方式。在 CLK 下降沿期间，SPI 协议还使用了单一方向 DO（输出）用来读取设备的状态或数据。在 CLK 上升沿期间，双路和四路 SPI 指令通过串行的方式使用双向 IO 引脚把数据、指令和地址写到装置之中；在 CLK 下降沿，从器件中读取数据或状态。四通道 SPI 指令需要对在状态寄存器 2 中的 QE 使能位进行配置。当 QE=1 时，/ WP 引脚变为 IO2 并且/ HOLD 引脚变为 IO3。

3.写保护（/ WP）

写保护（/ WP）引脚可实现阻止状态寄存器的写入的功能。 / WP 引脚为低电平有效。 但是，当状态寄存器 2 的 QE 使能位被配置为四通道 I / O，/ WP 引脚功能不能使用，因为该引脚已经被用于 IO2。

4.锁存（/ HOLD）

当设备被选中时，/HOLD 引脚可以允许设备暂停。当/ HOLD 变为低电平， 并且/ CS 也为低电平期间，DO 引脚会处于高阻抗的状态，主机会自动忽略 DI 和 CLK 引脚上的信号。当/ HOLD 置高，就可以恢复设备的操作。当多个设备共享同一 SPI 信号，/ HOLD 功能可以很有用。/ HOLD 是低电平有效的管脚。当状态的 QE 位寄存器-2 设置为四 I/ O 的/ HOLD 引脚功能不可用，因为该引脚为 用于 IO3。当状态寄存器 2 的 QE 位被配置成四 I/O 口时，因为该管脚被当做通用 I/O 口 3 来使用，所以/HOLD 引脚功能将不能被使用。

5.串行时钟（CLK）

CLK 即为 SPI 提供时钟的引脚，具有为串行的输入以及输出的操作提供时序的功能 （“见 SPI 操作“）

2.5 陀螺仪

2.5.1 综述

MPU 设备提供世界上第一个集成的 6 轴运动处理器解决方案，集成了 3 轴 MEMS 陀螺仪还有 3 轴 MEMS 加速度计，与以前的分立解决方案相对比，可解决陀螺仪和加速度计交叉轴没有对准的问题。MPU-6050 将 3 轴的陀螺仪和 3 轴的加速器集成在了同一个芯片上，并且搭配有复杂的板载数字运动处理器（DMP）。

为了精确捕捉到快速运动和慢速运动的物体，传感器 mpu6050 以用户可编程的陀螺仪作为主要优势，并且测量范围较大，在±250，±500，±1000，±2000°/秒（dps），加速度计也具备同样的优点，可测范围为±2，±4，±8，±16g。MPU6050 支持 I2C 通信，并且 MPU6050 有一个 VLOGIC 引脚，用于定义其接口电压水平。

2.5.2 引脚定义

引脚编号	引脚名称	描述
1	CLKIN	可选的外部时钟输入如果不用则连到 GND
6	AUX_DA	I2C 主串行数据，用于外接传感器
7	AUX_CL	I2C 主串行数据，用于外接传感器
8	VLOGIC	数字 I/O 供电电压
9	ADO/SDO	I2C Slave 地址 LSB
10	REGOUT	校准滤波电容连线
11	FSYNC	帧同步数字输入
12	INT	中断数字输出（推挽或升漏）
13	VDD	电源电压及数字 I/O 供电电压
18	GND	电源地
19,21,22	RESV	预留，不接
20	CPOUT	电荷泵电容连线
24	SCL	I2C 串行时钟
2,3,4,5,14,15,16,17	NC	不接

表 2.5 mpu6050 引脚定义

2.5.3 I2C 通信协议

1、开始（S）和结束（P）标志

传输开始通常表现为 SCL 线为高电平并且 SDA 由高电平变为低电平，结束信号的形式正好相反，SCL 还是保持高电平信号不变，在此期间 SDA 信号由低电平变成高电平。

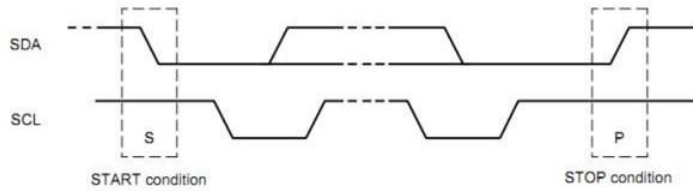


图 2.5.1 I2C 通信起始信号和结束信号时序图

2、数据格式和应答

I2C 单次传送 8 位的数据包，并且不限制一次通信过程传送的总字节数。并且每传送完一个数据包，从设备需要发出一个 ACK 信号表示已经接受到传

送的数据包，通过拉低 SDA 总线表示应答。SDA 仍为高电平，没有被拉低，表示并没有收到传送的数据。

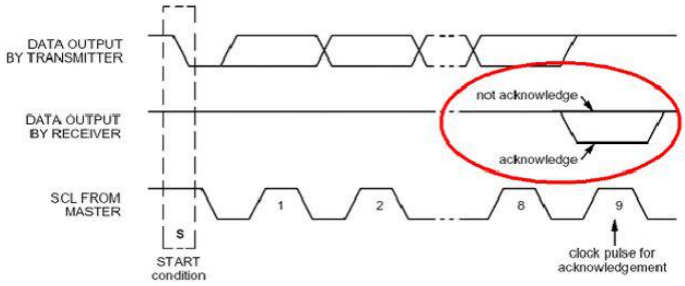


图 2.5.2 I2C 通信应答信号时序图

3、通信过程

开始标志（S）发出后，主设备会传送一个 7 位的 Slave 地址，并且后面跟着一个第 8 位，称为 Read/Write 位。R/W 位表示主设备是在接受从设备的数据还是在向其写数据。然后，主设备释放 SDA 线，等待从设备的应答信号（ACK）。每个字节的传输都要跟随有一个应答位。应答产生时，从设备将 SDA 线拉低并且在 SCL 为高电平时保持低。数据传输总是以停止标志（P）结束，然后释放通信线路。然而，主设备也可以产生重复的开始信号去操作另一台从设备，而不发出结束标志。综上可知，所有的 SDA 信号变化都要在 SCL 时钟为低电平时进行，除了开始和结束标志 。

通信开始后，主设备会发出一个从设备的 7 位地址来查找将要通信的从设备，并接 1 位读写信号，1 代表读设备，0 代表写设备。这个字节传送完成后等待从设备的应答信号。如果从设备给出应答信号，表示通信已经建立，下一步就是传送数据了，每传送 1 个字节的数据信号，同样需要从设备给出应答，以确定是否接受到信号。当传送完所有数据之后，以停止信号表示通信结束。需要注意的是，除了起始和应答信号，所有 SDA 信号状态的变化都是在 SCL 信号为低电平时进行。

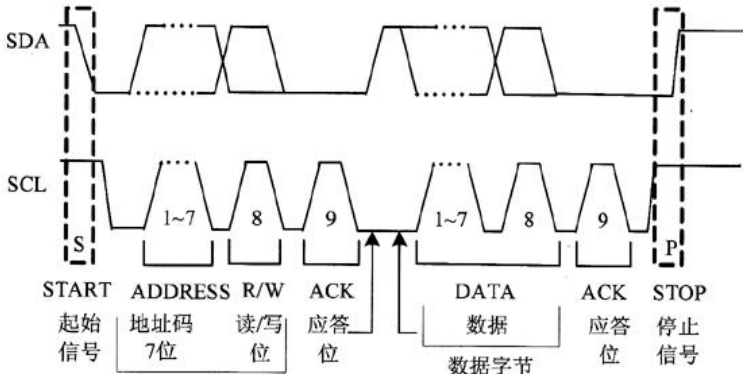


图 2.5.3 一次完整数据的传送过程时序图

3 系统组成及电路设计

3.1 系统组成部分

3.1.1 框图

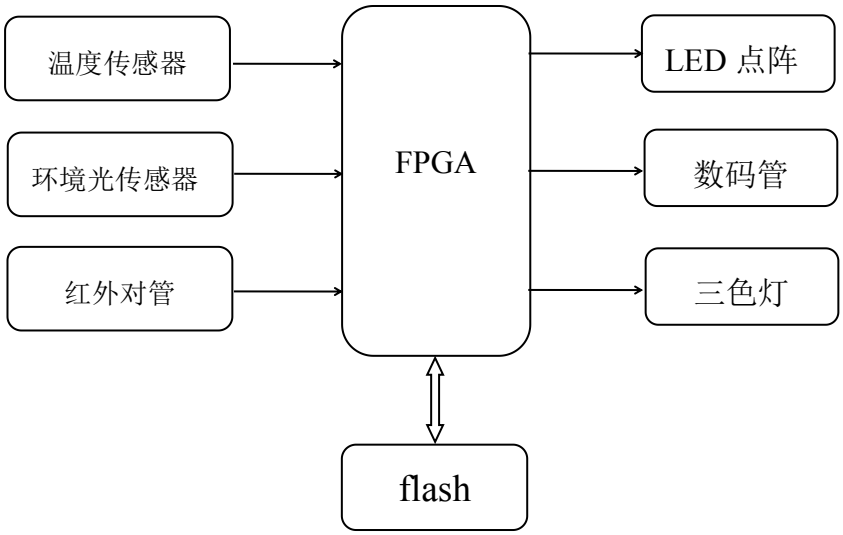


图 3.1 系统总框图

3.1.2 框图说明

本系统组成如图所示，主要由四部分组成：

1. 传感器部分：温度传感器 DS18B20，环境光传感器 BH1750FVI，将周围环境的模拟信号转换成数字信号，并传递给 FPGA 芯片等待处理；

2. 以 FPGA STEP-MXO2 芯片组成的中央处理单元：处理信号并发出控制命令；

3. 信息显示部分：包括 LED 点阵、数码管、三色灯，用于显示 FPGA 处理过的数据信息；

4. 存储部分：指在扩展板上的 flash W25Q64 芯片，可以作为程序存储区，把程序下载到扩展板的 flash 上。

3.2 电路详细设计

3.2.1 测温模块

测温模块由 DS18B20 作为主芯片，前文已概述了芯片的基本功能。DS18B20 芯片的电源和地之间需要接一个 100nF 的电容。这个电容主要有两个作用，分别是储能和旁路。储能指的是电路的耗电不确定，时大时小，当耗电突然增大的时候，假如没有电容，则会拉低电源电压，产生噪声，严重会导致 CPU 重启，如果存在电容，这时候 100nF 电容可以在短时间内把储存的电能给

释放出来一部分，起到保持电源电压稳定的作用。旁路指的是:电路电流在大多数情况下有脉动的现象，例如数字电路同步的频率，会造成电源电压的脉动，这种脉动是一种交流的噪声，而 100nF 的电容正好可以把这种噪声旁路到地，起到去除噪声的作用（电容可以通交流并且阻直流，同时小容量电容的通频带比大电容的同频带要高很多），也是为了提高系统的稳定性。

DS18B20 是单总线的通信方式，唯一的通信总线 DS DATA 与 FPGA 相连。DQ 管脚通过接一个上拉电阻到电源 Vcc 上，这种连接方式可以起到将不确定的输入输出信号通过一个上拉电阻嵌位到高电平上。

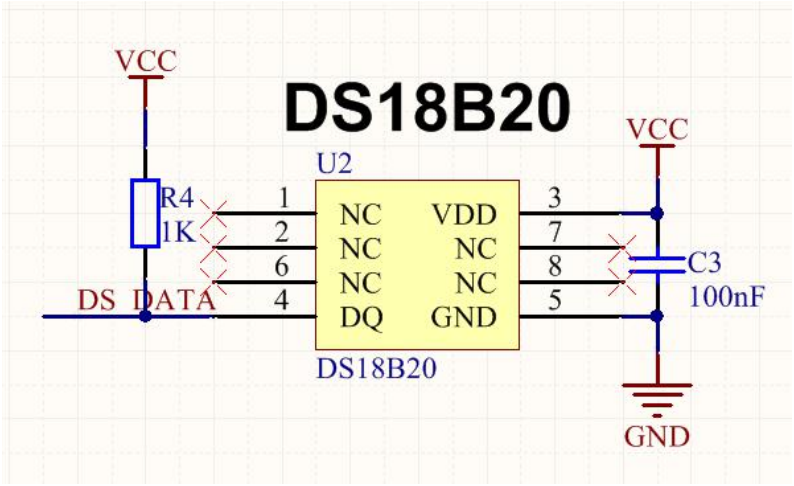


图 3.2.1 温度显示模块电路图

3.2.2 环境光模块

环境光模块由 bh1750fvi 作为主控芯片，其主要功能已经在前文有提到过。I2C 总线 SCL / SDA 线需要通过开漏结构进行驱动和连接，即由驱动低电压输出逻辑 0；通过驱动高阻的状态可以输出逻辑 1，并通过外部的上拉电阻拉至高电压。SCL 和 SDA 通信总线直接连接到主芯片 FPGA 配置好的 I2C 总线上的 SCL 和 SDA 管脚上去，因为主控芯片 fpga 内部已经接有上拉电阻，并且在软件调试中可以把 SCL 和 SDA 配置成上拉模式，所以并不需要再加上额外的上拉电阻。Vcc 和 GND 之间接了一个 100nF 的滤波电容，上文已经有描述这个电容的作用，这里不再赘述。

DVI 终端是一个可以实现异步重置的终端，如果在启动连接 VCC 完毕后忘记设置重置区，可能导致集成电路不正常工作，所以 DVI 管脚需要通过一个 100nF 的滤波电容接地。图下所示电路，按照芯片手册中参考电路设计，经过调试后可以正常运行，能从 BH1750FVI 中读出数据。

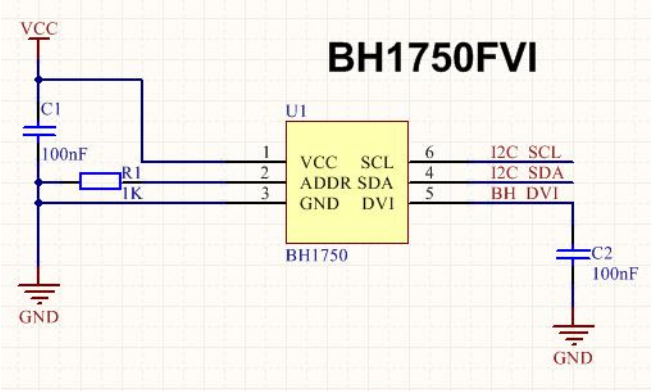


图 3.2.2 环境光亮度检测模块电路图

3.2.3 红外对管电路

红外收发对管是一种可以分辨红外线的开关管，包含红外发射管和红外接收管。红外接受管在接受较多红外线和接受很少红外线时电阻会发生非常明显的改变，利用外围电路可以使输出产生明显的高低电平的变化，高低电平的变化输入主控芯片 FPGA 就可使之识别，从而达到智能控制的目的。

左边是红外线发射电路，需要加一个 150 欧姆的限流电阻，用来防止红外发射管被击穿；右侧电路是红外线接收电路，R8 同样起到限流的作用，以防红外接收管被击穿。lm393 是电压比较器，IN-是参考电压，通过滑动变阻器 R6 调节参考电压，参考电压可以从 0 到 5V 之间变化，IN+的电压会随着红外接收管接收红外线的多少的变化而变化。当 IN+大于 IN-时，IR OUTPUT 会输出高电平；反之，当 IN+小于 IN-时，IR OUTPUT 会输出低电平。IR OUTPUT 需要接一个上拉电阻到高电平上。

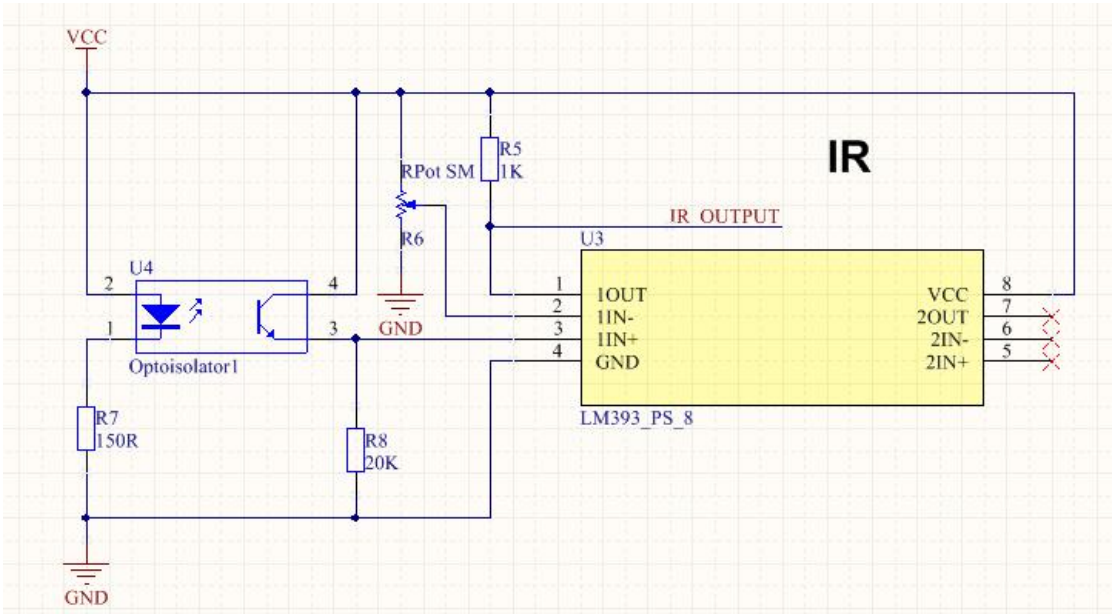


图 3.2.3 红外对管模块电路图

3.2.4 flash 电路设计

w25q64fv 是一种 flash 芯片，它的电路设计如下图所示。根据芯片资料可知，这个芯片是 SPI 的通信方式。时钟信号 CLK 由主设备产生接 FPGA 上 SPI 的 SCLK 管脚；CS 则是从设备的使能信号，由主设备来发出命令，控制从设备是否工作，所以需要与 FPGA 上 SPI 的 CS 管脚连接；DO 是 flash 的数据输出管脚，需要接 FPGA 的 MISO 管脚，即数据从主设备数据输入，从设备输出；DI 是 flash 的数据输入管脚，需要接 FPGA 的 MOSI 管脚，即数据从主设备输出，从设备输入。电源 Vcc 和地之间同样需要接一个 100nF 的滤波电容。WP 为写保护信号，为了防止状态寄存器被写入数据，这个引脚是低电平有效。电路连接时需要将管脚拉高，使写保护信号无效。HOLD 引脚可同时积极选择的设备被暂停。根据芯片手册可知，当 HOLD 变为低电平，而 CS 为低电平期间，DO 引脚将处于高阻抗的状态，从而会自动忽略在 DI 和 CLK 引脚上的信号。当 HOLD 拉高，设备操作就可以恢复。所以应当直接将 hold 信号置高。

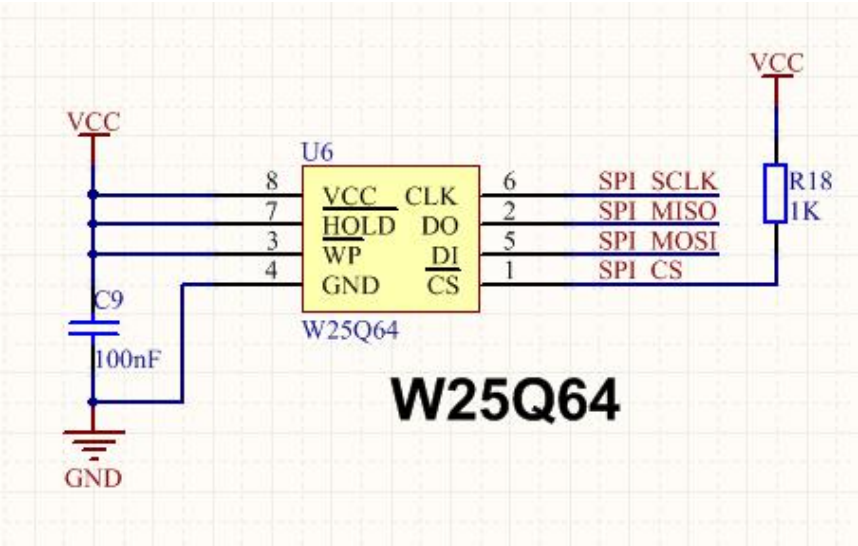


图 3.2.4 flash 存储模块电路图

3.2.5 LED 点阵的电路设计

LED 点阵由 8*16 的 LED 灯组成，LED 灯采用 0603 封装的贴片器件，较容易焊接。只有当列电平为高，行电平为低时，LED 灯会亮，其他条件下 LED 不会亮，并且每一路会接 470 欧姆的限流电阻，以防止电流过大把 LED 灯烧坏。

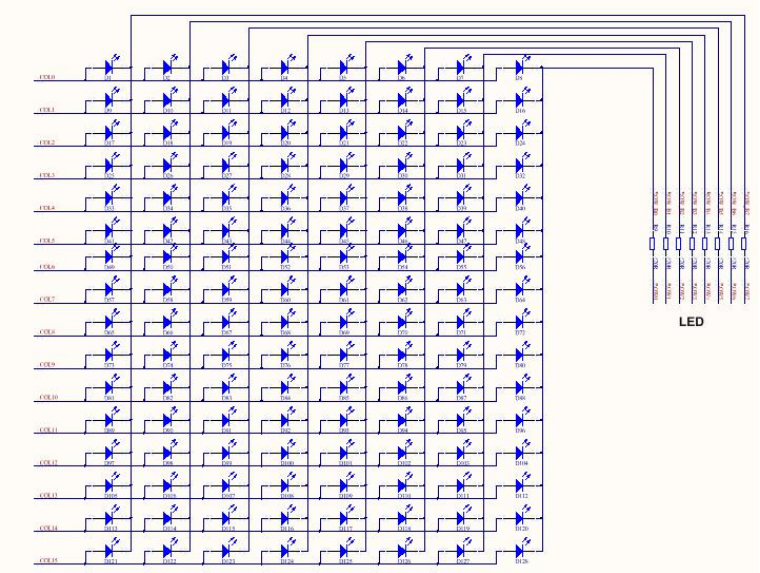


图 3.2.5 LED 点阵驱动电路图

3.2.6 核心板与扩展板的连接

在设计 FPGA 的管脚连接时，需要考虑有限的 I/O 接口资源。比如在设计 FPGA 主控芯片与 LED 点阵的连接时，首先要考虑 I/O 口是否够用。如果 I/O 足够使用，可以采用直接连接的方式，即 LED 点阵一共有 16+8=24 个端口，需要 FPGA 抽出 24 个管脚与之相连;否则需要额外添加 74hc595 这样的芯片来驱动 LED 灯。因为这款 FPGA 芯片资源足够，所以可以使用直接连接的方式。

另外一个需要注意的地方是 I2C 需要接上拉电阻到 Vcc 电源上。由于 I2C 接口采用开漏机制，器件本身只可以输出低电平信号，而无法输出高电平信号，只能依靠外部的上拉电阻将其拉至高电平。因此 I2C 总线上的上拉电阻是必须的!

上拉电阻 R 的阻值不能太小，一般来说不会小于 1K 欧姆。因为通常情况下通用 IO 口的驱动电流在 2 毫安与 4 毫安之间。如果上拉电阻过小，Vdd 的灌电流将加大，会使 mos 管不全导通，从而使其由饱和的状态退化为放大的状态，这样的端口输出的低电平值将增大（根据 I2C 的协议规定，端口允许输出的低电平大小最高为 0.4V）；灌电流过大还很有可能会使端口损坏。以上即为选用电阻须超过 1K 欧姆的原因。

但是上拉电阻 R 也不可以太大，一般来说不会大于 10K 欧姆。由于端口输出的高电平必须通过上拉电阻 R 来实现，那么电平从低到高发生变化时，电源会通过上拉电阻 R 对线上的负载电容 CL 进行充电，这个过程势必需要一定的

时间，这个时间被称作上升时间。端口信号的上升时间可以近似地用充电时间常数的乘积来表示。如果 RC 的充电时间常数过大，势必使得信号的上升沿变化得非常缓慢，从而不能达到数据传输的基本要求。

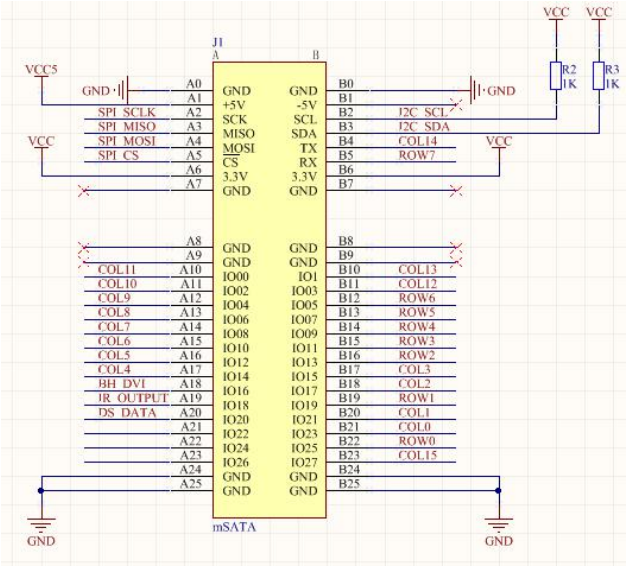


图 3. 2. 2 FPGA 主芯片连接图

4 PCB 设计

4. 1 PCB 布局

4. 1. 1 正面布局

U1 为环境光传感器 BH1750FVI，U2 为温度传感器 DS18B20，U3 为电压比较器 lm393，U4 是红外对管，U5 是陀螺仪 mpu6050，U6 是 flash 芯片。因为板卡空间有限，需要设计成 60mm*40mm 的扩展板，所以 led 点阵以 8 行 16 列的方式排列，达到节省空间的目的。右上方和右下方的定位孔起到固定板卡的作用。不同芯片分列在 LED 的两侧，方便与 FPGA 的引脚相连接。Flash 为 SPI 的通信方式，陀螺仪 mpu6050 为 I2C 的通信方式，均摆在 led 点阵的上方，因为底板的上半部分是配置好的 SPI 和 I2C 的接口，方便连接。剩余芯片摆在 LED 点阵的下方。

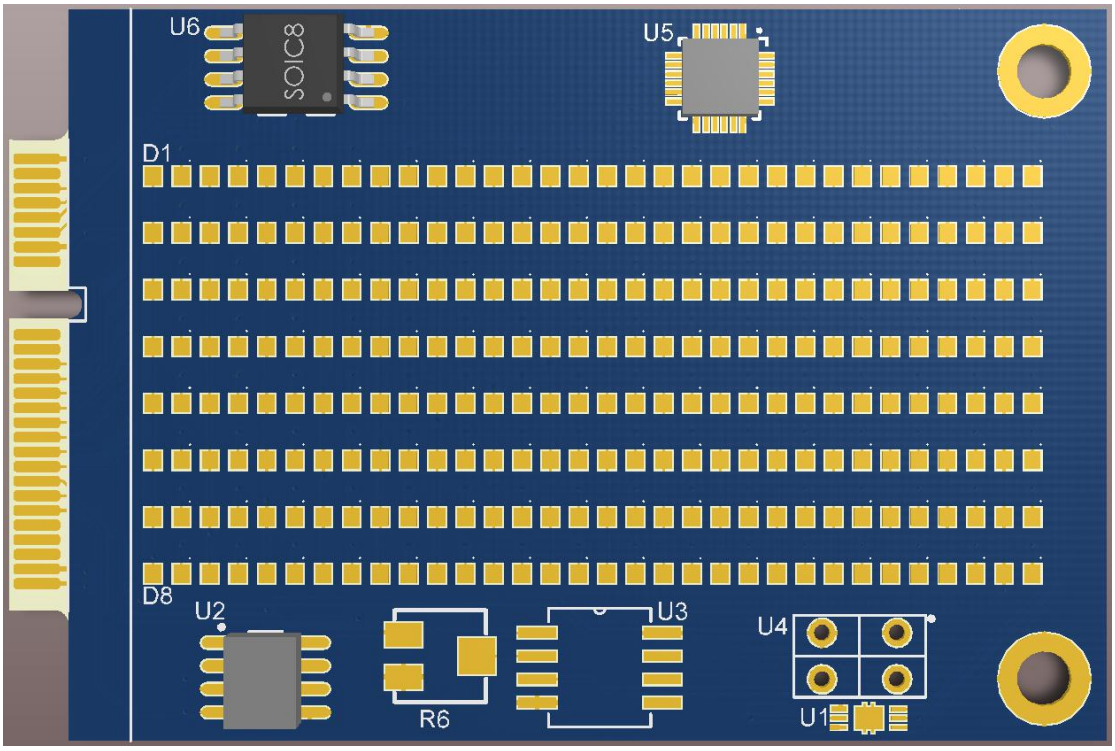


图 4.1.1 PCB 板正面布局

4.1.2 反面布局

正面空间有限，可以把一些电阻电容放在板卡的反面。R9~R16 为限流电阻，摆在反面的左侧。方便与每行的第一个 LED 灯相连。其余电阻电容根据不同所属的模块，放在对应芯片的旁边，使整个系统稳定准确地工作。

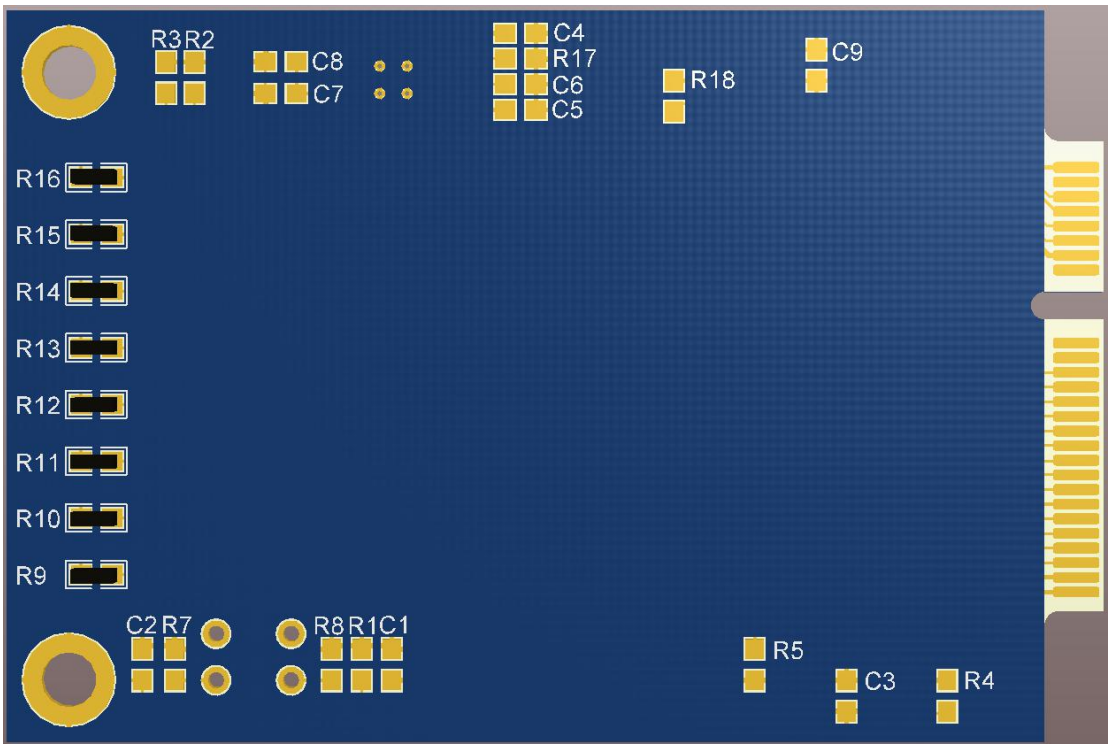


图 4.1.2 PCB 板反面布局

5 程序设计

5.1 温度显示模式

5.1.1 程序实现框图

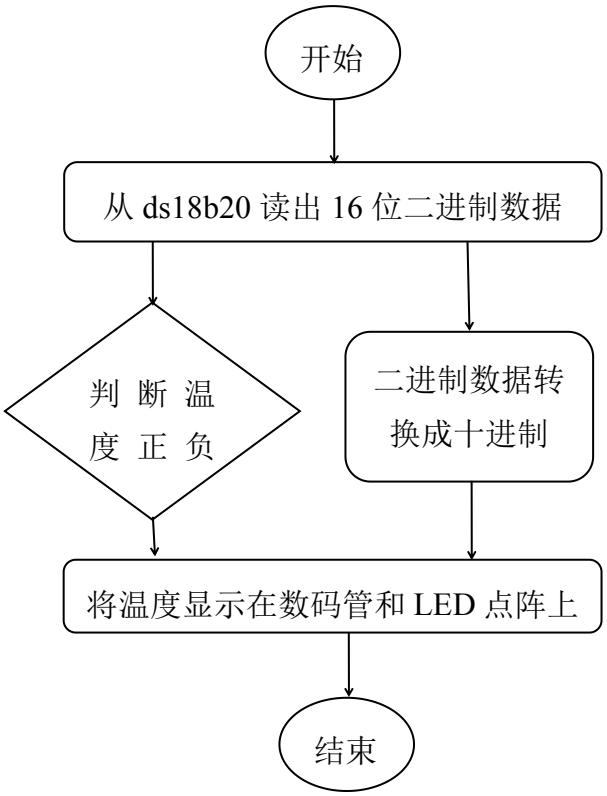


图 5.1.1 温度显示模块程序框图

程序框图说明：

FPGA 编程时都需要设计一个顶层文件，类似于 C 语言的 main 函数，主要功能是定义整个系统输入和输出的参数，方便配置管脚，输入使用 input，输出使用 output，输入输出使用 inout 定义。顶层文件的主要功能是例化调用各个子程序，完成系统的基本操作。

数据采集：读取传感器数据时，一般以状态机的形式，依据传感器不同的通信方式进行编程。比如 ds18b20 是单总线通信方式，需要依据总线协议写出程序，每次读出 16 位的二进制数据。

数据处理：即把二进制数据转换成对应的十进制，由 ds18b20 的数据手册

可知，16 位数据的前 5 位是符号位，需要根据全零或全一判定温度的正负号。中间 7 位是温度的整数部分，后四位是温度的小数部分，本次设计对温度精度要求不高，所以只需要对温度的整数部分进行处理即可。

7 位二进制数可以转换成 2 位 bcd 码，采用左移加 3 的算法实现，具体描述如下：左移需要转换的二进制码 1 位；左移之后，二进制码分别置于百位、十位、个位；如果移位后所在的 bcd 码列大于或等于 5，则对该值加 3；继续左移的过程，直到全部移位完成。如下图所示

Operation	Hundreds	Tens	Units	Binary	
HEX				F	F
Start				1 1 1 1	1 1 1 1
Shift 1			1	1 1 1 1	1 1 1
Shift 2			1 1	1 1 1 1	1 1
Shift 3			1 1 1	1 1 1 1	1
Add 3			1 0 1 0	1 1 1 1	1
Shift 4		1	0 1 0 1	1 1 1 1	
Add 3		1	1 0 0 0	1 1 1 1	
Shift 5		1 1	0 0 0 1	1 1 1	
Shift 6		1 1 0	0 0 1 1	1 1	
Add 3		1 0 0 1	0 0 1 1	1 1	
Shift 7	1	0 0 1 0	0 1 1 1	1	
Add 3	1	0 0 1 0	1 0 1 0	1	
Shift 8	1 0	0 1 0 1	0 1 0 1		
BCD	2	5	5		

图 5.1.2 二进制转十进制流程

数据显示：温度显示在 LED 点阵上是利用扫描显示的方法，按照固定频率进行扫描，每次扫描 1 列，利用人体的视觉暂留效应，把需要显示的温度数据显示在扩展板卡上的 LED 点阵上。显示在核心板的数码管上则较容易实现，只需要配置 seg 寄存器即可。

详细的程序代码可见附录一

5.2 光照强度显示模式

5.2.1 程序实现框图

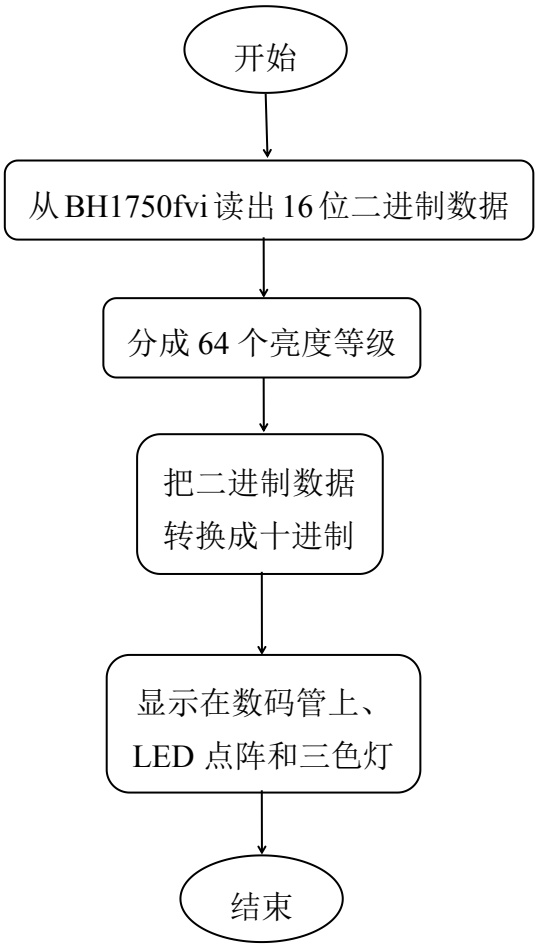


图 5. 2. 1 光照强度显示模块程序框图

程序框图说明：

Data collect：和 ds18b20 读取数据的方法类似，都是通过写一个状态机把传感器中的数据读出来。唯一不同的是 bh1750fvi 是 I2C 的通信方式，需要根据 I2C 的协议标准把 16 位的二进制数据读到 FPGA 当中。

Data process：16 位二级制数据即把亮度分成 2^16=65536 个亮度等级，实际上人眼是很难区分这么细微的亮度差异的，所以我们可以通过把数据向右移动十位，相当于除以 2^10，可以把亮度等级缩小为 64 个，然后将这六位二进制码运用算法转换成 BCD 码，转换方法与温度测量系统类似，即使用左移加三的算法。

Data display：运用 pwm 波的方式使三色灯的亮度随外界环境光亮度的变化而变化，pwm 波的产生方式为使用一个计数器产生锯齿波，与参照值作对比，大于参照值时亮，小于参照值时不亮，从而调节三色灯的亮度。显示在扩展板的 LED 点阵上的方法与温度测量显示系统类似，均采用扫描显示的方法，不过

实现的难度更大，因为 LED 点阵上箭头流动的快慢需要与环境光的亮度相匹配，所以需要配置两个计数器以达到这个目的。显示在核心板数码管上的数字较容易实现，只需要按照芯片手册分配管脚即可，不再赘述。

详细的程序代码可见附录二

6 结论

本次毕业设计主要可以实现两大功能，一是测量周围环境的亮度，二是测量环境光的强度。系统具有传输速度快，转换精度高的特点，整个系统使用 FPGA 作为主控芯片，因此具有良好的兼容性，与处理器接口简单，可节省硬件资源。该系统还具有设计简单，抗干扰能力较强，精度较高等优点。DS18B20 采用达拉斯专有的 1-Wire 总线协议，可将多个 DS18B20 连接到总线上。 万一发生故障，可立即更换损坏的 DS18B20，系统无需再次进行调节。 BH1750fv 使用 I2C 通信方式，正好与 FPGA 配置好的 I2C 总线相连接，巧妙地利用了芯片资源。并且 led 点阵会随着环境光亮度的不同相应地做出变化，与手机自动调节亮度的功能类似，具有很强的应用价值。

除此以外，本次毕业设计也可以供教学时使用，核心板上采用的 lattice 公司的芯片较简单，并且软件开发环境 Diamond 较容易上手，便于新手学习 FPGA。核心板上存在数码管、按键、LED 灯、拨码开关、三色灯等简单的外设，在初期学习时可以练习驱动这些简单外设，比如练习编写跑马灯等小程序。等熟悉 verilog 语言的语法和结构时，可以尝试较难的编程，这时就可以用到扩展板了。因为扩展板上存在多种传感器，并且传感器种类较多，属于不同的通信方式。比如 ds18b20 是单总线的驱动方式，BH1750FVI 是 I2C 的通信方式，而 flash 芯片 w25q64 采用 spi 的通信方式，可以让学生能练习驱动不同的传感器，并在这个过程中了解不同的通信协议。

致谢

时光飞逝，四年的本科生涯即将结束，在完成这篇毕业论文之际，我要向这四年成长和学习的道路上给了我帮助与支持的所有人表达我最真挚的感谢。

首先，我要感谢我毕业设计的指导老师范文兵老师，作为系主任和教授，他担负着教学和科研的重任，能在百忙之中抽出很多时间指导我毕业设计，我非常感激。范老师在论文选题，中期答辩以及论文修改上给予了我很大的帮助和支持，他严谨的治学态度、一丝不苟的精神将指引我在以后的学习与工作不断进步。在此次的毕业设计中，范老师在各方面都付出了很多，再次向敬爱的范老师表示感谢！

此次毕业设计，也得到了苏州摩尔吧硬禾实战营各位老师的帮助与支持。尤其是陈强和张泽老师在 PCB 设计上和软件代码的调试上给予了我全面的指导，帮助我改正了许多设计电路图时的常识性错误，以及传授了我许多 debug 的小技巧，我从中获益颇多，摩尔吧还提供了很多硬件设施比如万用表、示波器和焊台等等，借助这些我得以顺利完成毕业设计。

最后感谢各位专家在论文评审过程中提出的批评和建议。

参考文献

[1] Zhou Ruanjing and Xu Hongwei.Design of Temperature Measurement System Consisted of FPGA and DS18B20.IEEE.

[2] datasheet of DS18B20 by Dallas Semiconductor.Programmable Resolution 1-Wire® Digital Thermometer.

[3] datasheet of BH1750FVI by Rohm Semiconductor.Digital 16bit Serial Output Type Ambient Light Sensor IC .

[4] datasheet of w25q64 by Winbond Electronics Corp.3V 64M-BIT SERIAL FLASH MEMORY WITH DUAL/QUAD SPI & QPI.

[5] datasheet of mpu6050 by InvenSense.MPU-6000 and MPU-6050 Product Specification Revision3.2.

[6] 夏宇闻.Verilog 数字系统设计教程【第三版】.北京航空航天大学出版社

附录 1 温度显示程序

1-1 顶层文件（主程序）

```
module main
(
input clk_in,
input rst_n_in,
inout one_wire,
output   [15:0]    row,
output   [7:0]      line,

output [8:0] segment_led_1,    //MSB~LSB = SEG,DP,G,F,E,D,C,B,A
output [8:0] segment_led_2      //MSB~LSB = SEG,DP,G,F,E,D,C,B,A
);
wire [15:0] data_out;
wire[3:0] tens;
wire[3:0] ones;
DS18B20Z u1
(
.clk_in(clk_in),              // system clock
.rst_n_in(rst_n_in),          // system reset, active low
.one_wire(one_wire),              // ds18b20z one-wire-bus
.data_out(data_out)          // ds18b20z data_out
);
wire temperature_flag = data_out[15]? 1'b0:1'b1;
wire [11:4]data_in = temperature_flag? data_out[11:4]:(~data_out[11:4]);
conversion u2
(
.number(data_in[11:4]),
.tens(tens),
.ones(ones)
);
led_scan u3
(
```

```
.clk_in  (clk_in)    ,
.rst_n_in(rst_n_in)    ,
.data_out(data_out),
.tens(tens),
.ones(ones),
.row(row),
.line(line)
);
reg[8:0] seg [9:0];
initial
      begin
            seg[0] = 9'h3f;      //    0
            seg[1] = 9'h06;      //    1
            seg[2] = 9'h5b;      //    2
            seg[3] = 9'h4f;      //    3
            seg[4] = 9'h66;      //    4
            seg[5] = 9'h6d;      //    5
            seg[6] = 9'h7d;      //    6
            seg[7] = 9'h07;      //    7
            seg[8] = 9'h7f;      //    8
            seg[9] = 9'h6f;      //    9
      end

assign segment_led_1 = seg[tens];
assign segment_led_2 = seg[ones];
endmodule
```

1-2 从 ds18b20 读取数据（子程序）

```
module DS18B20Z
(
    input                clk_in,          // system clock
    input                rst_n_in,        // system reset, active low
    inout                one_wire,        // ds18b20z one-wire-bus
    output   reg  [15:0] data_out      // ds18b20z data_out
```

```
);
    localparam  IDLE    =   3'd0;
    localparam  MAIN    =   3'd1;
    localparam  INIT    =   3'd2;
    localparam  WRITE   =   3'd3;
    localparam  READ    =   3'd4;
    localparam  DELAY   =   3'd5;

    //generate clk_1mhz clock
    reg                 clk_1mhz;
    reg     [2:0]       cnt_1mhz;
    always@(posedge clk_in or negedge rst_n_in) begin
        if(!rst_n_in) begin
            cnt_1mhz <= 3'd0;
            clk_1mhz <= 1'b0;
        end else if(cnt_1mhz >= 3'd5) begin
            cnt_1mhz <= 3'd0;
            clk_1mhz <= ~clk_1mhz;
        end else begin
            cnt_1mhz <= cnt_1mhz + 1'b1;
        end
    end

    reg                 one_wire_buffer;
    reg     [3:0]       cnt_main;
    reg     [7:0]       data_wr;
    reg     [7:0]       data_wr_buffer;
    reg     [2:0]       cnt_init;
    reg     [19:0]      cnt_delay;
    reg     [19:0]      num_delay;
    reg     [5:0]       cnt_write;
    reg     [5:0]       cnt_read;
    reg     [15:0]      temperature;
    reg     [7:0]       temperature_buffer;
```

```
reg     [2:0]       state = IDLE;
reg     [2:0]       state_back = IDLE;
always@(posedge clk_1mhz or negedge rst_n_in) begin
    if(!rst_n_in) begin
        state <= IDLE;
        state_back <= IDLE;
        cnt_main <= 4'd0;
        cnt_init <= 3'd0;
        cnt_write <= 6'd0;
        cnt_read <= 6'd0;
        cnt_delay <= 20'd0;
        one_wire_buffer <= 1'bz;
        temperature <= 16'h0;
    end else begin
        case(state)
            IDLE:begin
                    state <= MAIN;
                    state_back <= MAIN;
                    cnt_main <= 4'd0;
                    cnt_init <= 3'd0;
                    cnt_write <= 6'd0;
                    cnt_read <= 6'd0;
                    cnt_delay <= 20'd0;
                    one_wire_buffer <= 1'bz;
                end
            MAIN:begin
                    if(cnt_main >= 4'd11) cnt_main <= 1'b0;
                    else cnt_main <= cnt_main + 1'b1;
                    case(cnt_main)
                        4'd0: begin state <= INIT; end
                        4'd1: begin data_wr <= 8'hcc;state <= WRITE; end
                        4'd2: begin data_wr <= 8'h44;state <= WRITE; end
                        4'd3: begin num_delay <= 20'd750000;state <=
DELAY;state_back <= MAIN; end
```

```
                    4'd4: begin state <= INIT; end
                    4'd5: begin data_wr <= 8'hcc;state <= WRITE; end
                    4'd6: begin data_wr <= 8'hbe;state <= WRITE; end

                    4'd7: begin state <= READ; end
                    4'd8:  begin  temperature[7:0]  <=  temperature_buffer;
end

                    4'd9: begin state <= READ; end
                    4'd10: begin temperature[15:8] <= temperature_buffer;
end

                    4'd11: begin state <= IDLE;data_out <= temperature;
end
                    default: state <= IDLE;
                endcase
            end
        INIT:begin
                if(cnt_init >= 3'd6) cnt_init <= 1'b0;
                else cnt_init <= cnt_init + 1'b1;
                case(cnt_init)
                    3'd0: begin one_wire_buffer <= 1'b0; end
                    3'd1:   begin   num_delay   <=   20'd500;state   <=
DELAY;state_back <= INIT; end
                    3'd2: begin one_wire_buffer <= 1'bz; end
                    3'd3:   begin   num_delay   <=   20'd100;state   <=
DELAY;state_back <= INIT; end
                    3'd4: begin if(one_wire) state <= IDLE; else state <=
INIT; end
                    3'd5:   begin   num_delay   <=   20'd400;state   <=
DELAY;state_back <= INIT; end
                    3'd6: begin state <= MAIN; end
                    default: state <= IDLE;
```

```
                endcase
            end
        WRITE:begin
                if(cnt_write >= 6'd50) cnt_write <= 1'b0;
                else cnt_write <= cnt_write + 1'b1;
                case(cnt_write)
                    //lock data_wr
                    6'd0: begin data_wr_buffer <= data_wr; end
                    //write bit 0
                    6'd1: begin one_wire_buffer <= 1'b0; end
                    6'd2: begin num_delay <= 20'd2;state <= DELAY;state_back <= WRITE; end
                    6'd3: begin one_wire_buffer <= data_wr_buffer[0]; end
                    6'd4: begin num_delay <= 20'd80;state <= DELAY;state_back <= WRITE; end
                    6'd5: begin one_wire_buffer <= 1'bz; end
                    6'd6: begin num_delay <= 20'd2;state <= DELAY;state_back <= WRITE; end
                    //write bit 1
                    6'd7: begin one_wire_buffer <= 1'b0; end
                    6'd8: begin num_delay <= 20'd2;state <= DELAY;state_back <= WRITE; end
                    6'd9: begin one_wire_buffer <= data_wr_buffer[1]; end
                    6'd10: begin num_delay <= 20'd80;state <= DELAY;state_back <= WRITE; end
                    6'd11: begin one_wire_buffer <= 1'bz; end
                    6'd12: begin num_delay <= 20'd2;state <= DELAY;state_back <= WRITE; end
                    //write bit 2
                    6'd13: begin one_wire_buffer <= 1'b0; end
                    6'd14: begin num_delay <= 20'd2;state <= DELAY;state_back <= WRITE; end
```

```
                6'd15: begin one_wire_buffer <= data_wr_buffer[2];
end
                6'd16: begin num_delay <= 20'd80;state <=
DELAY;state_back <= WRITE; end
                6'd17: begin one_wire_buffer <= 1'bz; end
                6'd18: begin num_delay <= 20'd2;state <=
DELAY;state_back <= WRITE; end
                //write bit 3
                6'd19: begin one_wire_buffer <= 1'b0; end
                6'd20: begin num_delay <= 20'd2;state <=
DELAY;state_back <= WRITE; end
                6'd21: begin one_wire_buffer <= data_wr_buffer[3];
end
                6'd22: begin num_delay <= 20'd80;state <=
DELAY;state_back <= WRITE; end
                6'd23: begin one_wire_buffer <= 1'bz; end
                6'd24: begin num_delay <= 20'd2;state <=
DELAY;state_back <= WRITE; end
                //write bit 4
                6'd25: begin one_wire_buffer <= 1'b0; end
                6'd26: begin num_delay <= 20'd2;state <=
DELAY;state_back <= WRITE; end
                6'd27: begin one_wire_buffer <= data_wr_buffer[4];
end
                6'd28: begin num_delay <= 20'd80;state <=
DELAY;state_back <= WRITE; end
                6'd29: begin one_wire_buffer <= 1'bz; end
                6'd30: begin num_delay <= 20'd2;state <=
DELAY;state_back <= WRITE; end
                //write bit 5
                6'd31: begin one_wire_buffer <= 1'b0; end
                6'd32: begin num_delay <= 20'd2;state <=
DELAY;state_back <= WRITE; end
                6'd33: begin one_wire_buffer <= data_wr_buffer[5];
```

```
end
                6'd34: begin num_delay <= 20'd80;state <=
DELAY;state_back <= WRITE; end
                6'd35: begin one_wire_buffer <= 1'bz; end
                6'd36: begin num_delay <= 20'd2;state <=
DELAY;state_back <= WRITE; end
                //write bit 6
                6'd37: begin one_wire_buffer <= 1'b0; end
                6'd38: begin num_delay <= 20'd2;state <=
DELAY;state_back <= WRITE; end
                6'd39: begin one_wire_buffer <= data_wr_buffer[6];
end
                6'd40: begin num_delay <= 20'd80;state <=
DELAY;state_back <= WRITE; end
                6'd41: begin one_wire_buffer <= 1'bz; end
                6'd42: begin num_delay <= 20'd2;state <=
DELAY;state_back <= WRITE; end
                //write bit 7
                6'd43: begin one_wire_buffer <= 1'b0; end
                6'd44: begin num_delay <= 20'd2;state <=
DELAY;state_back <= WRITE; end
                6'd45: begin one_wire_buffer <= data_wr_buffer[7];
end
                6'd46: begin num_delay <= 20'd80;state <=
DELAY;state_back <= WRITE; end
                6'd47: begin one_wire_buffer <= 1'bz; end
                6'd48: begin num_delay <= 20'd2;state <=
DELAY;state_back <= WRITE; end
                //back to main
                6'd49: begin num_delay <= 20'd80;state <=
DELAY;state_back <= WRITE; end
                6'd50: begin state <= MAIN; end
                default: state <= IDLE;
            endcase
```

```
            end
        READ:begin
                if(cnt_read >= 6'd48) cnt_read <= 1'b0;
                else cnt_read <= cnt_read + 1'b1;
                case(cnt_read)
                    //read bit 0
                    6'd0: begin one_wire_buffer <= 1'b0; end
                    6'd1:  begin  num_delay  <=  20'd2;state  <=
DELAY;state_back <= READ; end
                    6'd2: begin one_wire_buffer <= 1'bz; end
                    6'd3:  begin  num_delay  <=  20'd10;state  <=
DELAY;state_back <= READ; end
                    6'd4: begin temperature_buffer[0] <= one_wire; end
                    6'd5:  begin  num_delay  <=  20'd55;state  <=
DELAY;state_back <= READ; end
                    //read bit 1
                    6'd6: begin one_wire_buffer <= 1'b0; end
                    6'd7:  begin  num_delay  <=  20'd2;state  <=
DELAY;state_back <= READ; end
                    6'd8: begin one_wire_buffer <= 1'bz; end
                    6'd9:  begin  num_delay  <=  20'd10;state  <=
DELAY;state_back <= READ; end
                    6'd10: begin temperature_buffer[1] <= one_wire; end
                    6'd11:  begin  num_delay  <=  20'd55;state  <=
DELAY;state_back <= READ; end
                    //read bit 2
                    6'd12: begin one_wire_buffer <= 1'b0; end
                    6'd13:  begin  num_delay  <=  20'd2;state  <=
DELAY;state_back <= READ; end
                    6'd14: begin one_wire_buffer <= 1'bz; end
                    6'd15:  begin  num_delay  <=  20'd10;state  <=
DELAY;state_back <= READ; end
                    6'd16: begin temperature_buffer[2] <= one_wire; end
                    6'd17:  begin  num_delay  <=  20'd55;state  <=
```

```
DELAY;state_back <= READ; end
                    //read bit 3
                    6'd18: begin one_wire_buffer <= 1'b0; end
                    6'd19: begin num_delay <= 20'd2;state <=
DELAY;state_back <= READ; end
                    6'd20: begin one_wire_buffer <= 1'bz; end
                    6'd21: begin num_delay <= 20'd10;state <=
DELAY;state_back <= READ; end
                    6'd22: begin temperature_buffer[3] <= one_wire; end
                    6'd23: begin num_delay <= 20'd55;state <=
DELAY;state_back <= READ; end
                    //read bit 4
                    6'd24: begin one_wire_buffer <= 1'b0; end
                    6'd25: begin num_delay <= 20'd2;state <=
DELAY;state_back <= READ; end
                    6'd26: begin one_wire_buffer <= 1'bz; end
                    6'd27: begin num_delay <= 20'd10;state <=
DELAY;state_back <= READ; end
                    6'd28: begin temperature_buffer[4] <= one_wire; end
                    6'd29: begin num_delay <= 20'd55;state <=
DELAY;state_back <= READ; end
                    //read bit 5
                    6'd30: begin one_wire_buffer <= 1'b0; end
                    6'd31: begin num_delay <= 20'd2;state <=
DELAY;state_back <= READ; end
                    6'd32: begin one_wire_buffer <= 1'bz; end
                    6'd33: begin num_delay <= 20'd10;state <=
DELAY;state_back <= READ; end
                    6'd34: begin temperature_buffer[5] <= one_wire; end
                    6'd35: begin num_delay <= 20'd55;state <=
DELAY;state_back <= READ; end
                    //read bit 6
                    6'd36: begin one_wire_buffer <= 1'b0; end
                    6'd37: begin num_delay <= 20'd2;state <=
```

```
DELAY;state_back <= READ; end
                                6'd38: begin one_wire_buffer <= 1'bz; end
                                6'd39: begin num_delay <= 20'd10;state <=
DELAY;state_back <= READ; end
                                6'd40: begin temperature_buffer[6] <= one_wire; end
                                6'd41: begin num_delay <= 20'd55;state <=
DELAY;state_back <= READ; end
                                //read bit 7
                                6'd42: begin one_wire_buffer <= 1'b0; end
                                6'd43: begin num_delay <= 20'd2;state <=
DELAY;state_back <= READ; end
                                6'd44: begin one_wire_buffer <= 1'bz; end
                                6'd45: begin num_delay <= 20'd10;state <=
DELAY;state_back <= READ; end
                                6'd46: begin temperature_buffer[7] <= one_wire; end
                                6'd47: begin num_delay <= 20'd55;state <=
DELAY;state_back <= READ; end
                                //back to main
                                6'd48: begin state <= MAIN; end
                                default: state <= IDLE;
                            endcase
                        end
                    DELAY:begin
                            if(cnt_delay >= num_delay) begin
                                cnt_delay <= 1'b0;
                                state <= state_back;
                            end else cnt_delay <= cnt_delay + 1'b1;
                        end
                endcase
            end
        end

        assign    one_wire = one_wire_buffer;
```

endmodule

1-3 十进制转换成二进制（子程序）

```
module conversion (number, tens, ones);
    // I/O Signal Definitions
    input    [7:0] number;
    reg [3:0] hundreds;
    output reg [3:0] tens;
    output reg [3:0] ones;

    // Internal variable for storing bits
    reg [19:0] shift;
    reg [10:0] numb;
    integer i;

    always @(number)
    begin
        // Clear previous number and store new number in shift register
        shift[19:8] = 0;
        numb=(number*5)>>3;
       shift[7:0] = numb[7:0];
        // Loop eight times
        for (i=0; i<8; i=i+1) begin
            if (shift[11:8] >= 5)
                shift[11:8] = shift[11:8] + 3;

            if (shift[15:12] >= 5)
                shift[15:12] = shift[15:12] + 3;

            if (shift[19:16] >= 5)
                shift[19:16] = shift[19:16] + 3;

            // Shift entire register left once
            shift = shift << 1;
```

```
        end

        // Push decimal numbers to output
        hundreds = shift[19:16];
        tens     = shift[15:12];
        ones     = shift[11:8];
    end

endmodule
```

1-4 温度显示程序（子程序）

```
module led_scan
(
    input    clk_in      ,
    input    rst_n_in    ,
    input    [15:11]data_out,
     input    [3:0] tens,
     input    [3:0] ones,
    output   reg  [15:0]   row ,
    output   reg  [7:0]    line
);
    reg      [31:0]   mem     [12:0];
    reg      [7:0]    cache   [15:0];
     wire flag = data_out[15:11]? 1'b0:1'b1;

    wire     clk10KHz;
    counter #
    (
    .COUNTER_NUM (1200)
    )
    clk10KHz_uut
    (
    .clk(clk_in),       //system clk
    .rst(rst_n_in),          //system rst
```

```
.invert(clk10KHz)     //invert signal
);

always @ (posedge clk10KHz)
    begin
        if (!rst_n_in == 1)
            row <= 16'h0001;
        else
            row <= {row[14:0],row[15]};
    end

initial
    begin
        mem[0]      = {~8'h3E,~8'h41,~8'h41,~8'h3E};
        mem[1]      = {~8'h11,~8'h31,~8'h7f,~8'h01};
        mem[2]      = {~8'h23,~8'h45,~8'h49,~8'h31};
        mem[3]      = {~8'h22,~8'h49,~8'h49,~8'h36};
        mem[4]      = {~8'h0C,~8'h14,~8'h24,~8'h7f};
        mem[5]      = {~8'h7A,~8'h49,~8'h49,~8'h46};
        mem[6]      = {~8'h3E,~8'h49,~8'h49,~8'h26};
        mem[7]      = {~8'h40,~8'h47,~8'h48,~8'h70};
        mem[8]      = {~8'h36,~8'h49,~8'h49,~8'h36};
        mem[9]      = {~8'h32,~8'h49,~8'h49,~8'h3E};
        mem[10] = {~8'h00,~8'h00,~8'h00,~8'h00};
        mem[11] = {~8'h08,~8'h08,~8'h08,~8'h08};
        mem[12] = {~8'hC0,~8'hDE,~8'h21,~8'h21};
    end
always@(flag)
    if(!flag)
        begin
            cache[0]<=mem[10][31:24];
            cache[1]<=mem[10][23:16];
            cache[2]<=mem[10][15:8];
            cache[3]<=mem[10][7:0];
```

```
            end
        else
            begin
                cache[0]<=mem[10][31:24];
                cache[1]<=mem[10][23:16];
                cache[2]<=mem[10][15:8];
                cache[3]<=mem[10][7:0];
            end
    always@(tens or ones)
        begin
            cache[4]<=mem[tens][31:24];
            cache[5]<=mem[tens][23:16];
            cache[6]<=mem[tens][15:8];
            cache[7]<=mem[tens][7:0];
            cache[8]<=mem[ones][31:24];
            cache[9]<=mem[ones][23:16];
            cache[10]<=mem[ones][15:8];
            cache[11]<=mem[ones][7:0];
            cache[12]<=mem[12][31:24];
            cache[13]<=mem[12][23:16];
            cache[14]<=mem[12][15:8];
            cache[15]<=mem[12][7:0];
        end

    always @ (row)
        begin
            case (row)
                16'h0001 : line <= cache[0];
                16'h0002 : line <= cache[1];
                16'h0004 : line <= cache[2];
                16'h0008 : line <= cache[3];
                16'h0010 : line <= cache[4];
                16'h0020 : line <= cache[5];
                16'h0040 : line <= cache[6];
```

```
                16'h0080 : line <= cache[7];
                16'h0100 : line <= cache[8];
                16'h0200 : line <= cache[9];
                16'h0400 : line <= cache[10];
                16'h0800 : line <= cache[11];
                16'h1000 : line <= cache[12];
                16'h2000 : line <= cache[13];
                16'h4000 : line <= cache[14];
                16'h8000 : line <= cache[15];

            endcase

        end
/*  initial
        begin
    cache[0] = {8'b00000000,8'b01111110,8'b01111110,8'b00000000};
    cache[1] = {8'b11111111,8'b11111111,8'b00000000,8'b11111111};
    cache[2] = {8'b01100000,8'b01100001,8'b01101110,8'b00001110};
    cache[3] = {8'b01101110,8'b01101110,8'b01101110,8'b00000000};
    cache[4] = {8'b00001111,8'b11100000,8'b11101111,8'b00000000};
    cache[5] = {8'b00001110,8'b01101110,8'b01101110,8'b01100000};
    cache[6] = {8'b00000000,8'b01101110,8'b01101110,8'b01100000};
    cache[7] = {8'b01111111,8'b01111111,8'b01111111,8'b00000000};
    cache[8] = {8'b00000000,8'b01101110,8'b01101110,8'b00000000};
    cache[9] = {8'b00001110,8'b01101110,8'b01101110,8'b00000000};
    mem[15]  = 8'b11111111;
    mem[14]  = 8'b11111111;
    mem[13]  = 8'b11111111;
    mem[12]  = 8'b11111111;
    mem[3]     = 8'b11111111;
    mem[2]     = 8'b11111111;
    mem[1]     = 8'b11111111;
    mem[0]     = 8'b11111111;
        end
```

```
always@(tens or ones)
    begin
        mem[4]<=cache[tens][7:0];
        mem[5]<=cache[tens][15:8];
        mem[6]<=cache[tens][23:16];
        mem[7]<=cache[tens][31:24];
        mem[8]<=cache[ones][7:0];
        mem[9]<=cache[ones][15:8];
        mem[10]<=cache[ones][23:16];
        mem[11]<=cache[ones][31:24];
    end
always @ (row)
    begin
        case (row)
            16'h0001 : line <= mem[0];
            16'h0002 : line <= mem[1];
            16'h0004 : line <= mem[2];
            16'h0008 : line <= mem[3];
            16'h0010 : line <= mem[4];
            16'h0020 : line <= mem[5];
            16'h0040 : line <= mem[6];
            16'h0080 : line <= mem[7];
            16'h0100 : line <= mem[8];
            16'h0200 : line <= mem[9];
            16'h0400 : line <= mem[10];
            16'h0800 : line <= mem[11];
            16'h1000 : line <= mem[12];
            16'h2000 : line <= mem[13];
            16'h4000 : line <= mem[14];
            16'h8000 : line <= mem[15];
            default  : line <= ~8'b00011000;
        endcase

    end*/
```

```
endmodule
```

1-5 10khz 时钟信号的产生（子程序）

```
module counter #
(
    parameter    COUNTER_NUM = 3464      //period      =      (3464**2)*2      =
24_000_000 = 2s
)
(
    //INPUT
    input          clk,       //system clk
    input          rst,       //system rst
    output   reg  invert                    //invert signal
);
    reg       [31:0]          cnt;
    //generate invert signal);
    always @ (posedge clk or negedge rst)
        begin
            if(!rst == 1)
                begin
                    cnt <= 32'd0;
                    invert <= 0;
                end
            else
                begin
                    if(cnt >= COUNTER_NUM - 1)
                        begin
                            invert <= ~invert;
                            cnt <= 1'b0;
                        end
                    else
                        begin
                            cnt <= cnt + 1'b1;
                        end
```

```
            end
        end
endmodule
```

附录二 亮度显示程序

2-1 顶层文件（主程序）

```
Module
Interface_BH1750(clk,rst,scl,sda,dvi,seg_led1,seg_led2,row,line,rgb_led1,rgb_led2);
        //BH1750 Interface//
        input           clk;
        input           rst;
        output          scl;
        inout           sda;
        output          dvi;

        //Signal Output//
        output  [15:0]  row;
        output  [7:0]   line;
        output  [8:0]   seg_led1;
        output  [8:0]   seg_led2;
        output  [2:0]   rgb_led1;
        output  [2:0]   rgb_led2;

        //Module Connection//
        wire    [15:0]  BH1750_data_out;
        wire    [7:0]   BH1750_processed_data;
        wire    [3:0]   tens;
        wire    [3:0]   ones;

        //BH1750 I2C Read Data
        BH1750 u1
        (
        .clk(clk),
```

```
.rst(rst),
.scl(scl),
.sda(sda),
.dvi(dvi),
.data_out(BH1750_data_out)
);

//Process Data: Divide 1024//
Data_Process u2
(
.data_input(BH1750_data_out),
.data_output(BH1750_processed_data)
);

//BIN_To_BCD//
BIN_To_BCD u3
(
.number(BH1750_processed_data),
.tens(tens),
.ones(ones)
);

//Segment_LED Display//
seg_led u4
(
.tens(tens),
.ones(ones),
.segment_led_1(seg_led1),    //MSB~LSB = SEG,DP,G,F,E,D,C,B,A
.segment_led_2(seg_led2)      //MSB~LSB = SEG,DP,G,F,E,D,C,B,A
);

//Matrix_LED Display//
led_scan u5
(
```

```
        .clk(clk),
        .rst(rst),
        .number(BH1750_processed_data),
        .tens(tens),
        .ones(ones),
        .row(row),
        .line(line)
        );

        //RGB_LED Display//
        rgb_led u6
        (
        .clk(clk),
        .rst(rst),
        .pwm_duty(BH1750_processed_data),
        .led1(rgb_led1),
        .led2(rgb_led2)
        );
    endmodule
```

2-2 BH1750 读数据（子程序）

```
module BH1750(clk,rst,scl,sda,dvi,data_out);

    input           clk;
    input           rst;
    output          scl;
    inout           sda;
    output          dvi;
    output  [15:0]  data_out;


    wire    [7:0]   led;
    reg     [7:0]   r_led;
```

```
assign led = r_led;

reg                 r_scl;
reg                 r_sda;
reg                 ctl_sda;
assign sda = ctl_sda ? r_sda : 1'bz;
assign scl = r_scl;
assign dvi = 1;

reg       [15:0]    data;
assign data_out = data;

reg  [7:0]    cnt_100khz;
always@(posedge clk or negedge rst)
    begin
        if(!rst)
            cnt_100khz<=1'b0;
        else if(cnt_100khz==59)
            cnt_100khz<=1'b0;
        else
            cnt_100khz<=cnt_100khz+1'b1;
    end

reg clk_100khz;
always@(posedge clk or negedge rst)
    begin
        if(!rst)
            clk_100khz<=1'b0;
        else if(cnt_100khz==59)
            clk_100khz<=~clk_100khz;
        else
            clk_100khz<=clk_100khz;
    end
```

```
    parameter [7:0] DEVICE_ADDR_W      = 8'h46;
    parameter [7:0] DEVICE_ADDR_R      = 8'h47;
    parameter [7:0] DEVICE_POWER_OFF = 8'h00;
    parameter [7:0] DEVICE_POWER_ON    = 8'h01;
    parameter [7:0] DEVICE_MEASURE     = 8'h13;


    reg [11:0] cnt;
    always@(posedge clk_100khz or negedge rst) begin
        if(!rst) begin
            ctl_sda<=1'b1;r_sda<=1'b1;
            r_scl<=1'b1;
            cnt<=0;
            r_led<=8'b00000000;
        end else begin
            cnt<=cnt+1'b1;
            case(cnt)
//外部地址+写+断电-----------------------------------------------------------//
                0:   begin ctl_sda<=1'b1; r_led<=8'b00000000; end
                //START
                1:   begin r_scl<=1'b1;          end
                2:   begin r_sda<=1'b1;          end
                3:   begin r_sda<=1'b0;          end
                //ADDRESS+WRITE 8'H46
                4:   begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_W[7]; end
                5:   begin r_scl<=1'b1;                                 end

                6:   begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_W[6]; end
                7:   begin r_scl<=1'b1;      end

                8:   begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_W[5]; end
                9:   begin r_scl<=1'b1;      end

                10:  begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_W[4]; end
```

```
11:  begin r_scl<=1'b1;       end

12:  begin r_scl<=1'b0;     r_sda<=DEVICE_ADDR_W[3]; end
13:  begin r_scl<=1'b1;       end

14:  begin r_scl<=1'b0;     r_sda<=DEVICE_ADDR_W[2]; end
15:  begin r_scl<=1'b1;       end

16:  begin r_scl<=1'b0;     r_sda<=DEVICE_ADDR_W[1]; end
17:  begin r_scl<=1'b1;       end

18:  begin r_scl<=1'b0;     r_sda<=DEVICE_ADDR_W[0]; end
19:  begin r_scl<=1'b1;       end
20:  begin r_scl<=1'b0;       end

21:  begin ctl_sda<=1'b0; end
22:  begin r_scl<=1'b1;       end
23:  begin
          r_scl<=1'b0;
          if(sda==1'b0) begin
               cnt<=24;r_led[0]<=0;
          end else
               cnt<=22;
          end
//POWER OFF 8'H00
24:  begin ctl_sda<=1'b1; end
25:  begin r_scl<=1'b0 ;    r_sda<=DEVICE_POWER_OFF[7]; end
26:  begin r_scl<=1'b1 ;      end

27:  begin r_scl<=1'b0 ;    r_sda<=DEVICE_POWER_OFF[6]; end
28:  begin r_scl<=1'b1 ;      end

29:  begin r_scl<=1'b0 ;    r_sda<=DEVICE_POWER_OFF[5]; end
30:  begin r_scl<=1'b1 ;      end
```

```
31:  begin r_scl<=1'b0 ;   r_sda<=DEVICE_POWER_OFF[4]; end
32:  begin r_scl<=1'b1 ;     end

33:  begin r_scl<=1'b0 ;   r_sda<=DEVICE_POWER_OFF[3]; end
34:  begin r_scl<=1'b1 ;     end

35:  begin r_scl<=1'b0 ;   r_sda<=DEVICE_POWER_OFF[2]; end
36:  begin r_scl<=1'b1 ;     end

37:  begin r_scl<=1'b0 ;   r_sda<=DEVICE_POWER_OFF[1]; end
38:  begin r_scl<=1'b1 ;     end

39:  begin r_scl<=1'b0 ;   r_sda<=DEVICE_POWER_OFF[0]; end
40:  begin r_scl<=1'b1 ;     end
41:   begin r_scl<=1'b0;       end

42:  begin ctl_sda<=1'b0;   end
43:  begin r_scl<=1'b1;       end
44:  begin
          r_scl<=1'b0;
          if(sda==1'b0) begin
               cnt<=45;r_led[1]<=0;
          end else
               cnt<=43;
          end
45:  begin ctl_sda<=1'b1; end
     //STOP
46:  begin r_scl<=1'b1;         end
47:   begin r_sda<=1'b0;        end
48:  begin r_sda<=1'b1;         end
//外部地址+写+上电----------------------------------------------------------//
 49: begin ctl_sda<=1'b1; end
 //START
```

```
50:  begin r_scl<=1'b1;           end
51:  begin r_sda<=1'b1;           end
52:  begin r_sda<=1'b0;           end
//ADDRESS+WRITE 8'H46
53:  begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_W[7]; end
54:  begin r_scl<=1'b1;                                  end

55:  begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_W[6]; end
56:  begin r_scl<=1'b1;      end

57:  begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_W[5]; end
58:  begin r_scl<=1'b1;      end

59: begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_W[4]; end
60: begin r_scl<=1'b1;      end

61: begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_W[3]; end
62: begin r_scl<=1'b1;      end

63: begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_W[2]; end
64: begin r_scl<=1'b1;      end

65: begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_W[1]; end
66: begin r_scl<=1'b1;      end

67: begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_W[0]; end
68: begin r_scl<=1'b1;      end
69: begin r_scl<=1'b0;      end

70: begin ctl_sda<=1'b0; end
71: begin r_scl<=1'b1;      end
72: begin
        r_scl<=1'b0;
        if(sda==1'b0) begin
```

```
                cnt<=73;r_led[2]<=0;
            end else
                cnt<=71;
            end
//POWER OFF 8'H00
  73: begin ctl_sda<=1'b1; end
  74: begin r_scl<=1'b0 ;    r_sda<=DEVICE_POWER_ON[7]; end
  75: begin r_scl<=1'b1 ;      end

  76: begin r_scl<=1'b0 ;    r_sda<=DEVICE_POWER_ON[6]; end
  77: begin r_scl<=1'b1 ;      end

  78: begin r_scl<=1'b0 ;    r_sda<=DEVICE_POWER_ON[5]; end
  79: begin r_scl<=1'b1 ;      end

  80: begin r_scl<=1'b0 ;    r_sda<=DEVICE_POWER_ON[4]; end
  81: begin r_scl<=1'b1 ;      end

  82: begin r_scl<=1'b0 ;    r_sda<=DEVICE_POWER_ON[3]; end
  83: begin r_scl<=1'b1 ;      end

  84: begin r_scl<=1'b0 ;    r_sda<=DEVICE_POWER_ON[2]; end
  85: begin r_scl<=1'b1 ;      end

  86: begin r_scl<=1'b0 ;    r_sda<=DEVICE_POWER_ON[1]; end
  87: begin r_scl<=1'b1 ;      end

  88: begin r_scl<=1'b0 ;    r_sda<=DEVICE_POWER_ON[0]; end
  89: begin r_scl<=1'b1 ;      end
  90:    begin r_scl<=1'b0;      end

  91: begin ctl_sda<=1'b0;    end
  92: begin r_scl<=1'b1;        end
  93: begin
```

```
                r_scl<=1'b0;
                if(sda==1'b0) begin
                    cnt<=94;r_led[3]<=0;
                end else
                    cnt<=92;
                end
            94: begin ctl_sda<=1'b1; end
            //STOP
            95: begin r_scl<=1'b1;          end
            96: begin r_sda<=1'b0;          end
            97: begin r_sda<=1'b1;          end
////外部地址+写+读取单次-------------------------------------------------------//
            98:     begin ctl_sda<=1'b1; end
            //START
            99:     begin r_scl<=1'b1;          end
            100:    begin r_sda<=1'b1;          end
            101:    begin r_sda<=1'b0;          end
            //ADDRESS+WRITE 8'H46
            102:    begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_W[7];
end
            103:    begin r_scl<=1'b1;                                  end

            104:    begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_W[6];
end
            105:    begin r_scl<=1'b1;      end

            106:    begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_W[5];
end
            107:    begin r_scl<=1'b1;      end

            108:    begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_W[4];
end
            109:    begin r_scl<=1'b1;      end
```

```
                110:    begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_W[3];
end
                111:    begin r_scl<=1'b1;      end

                112:    begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_W[2];
end
                113:    begin r_scl<=1'b1;      end

                114:    begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_W[1];
end
                115:    begin r_scl<=1'b1;      end

                116:    begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_W[0];
end
                117:    begin r_scl<=1'b1;      end
                118:    begin r_scl<=1'b0;      end

                119:    begin ctl_sda<=1'b0; end
                120:    begin r_scl<=1'b1;      end
                121:    begin
                        r_scl<=1'b0;
                        if(sda==1'b0) begin
                            cnt<=122;r_led[4]<=0;
                        end else
                            cnt<=120;
                        end
               //POWER OFF 8'H23
                122:    begin ctl_sda<=1'b1; end
                123:    begin  r_scl<=1'b0  ;     r_sda<=DEVICE_MEASURE[7];
end
                124:    begin r_scl<=1'b1 ;     end

                125:    begin  r_scl<=1'b0  ;     r_sda<=DEVICE_MEASURE[6];
end
```

```
126:    begin r_scl<=1'b1 ;     end

127:    begin  r_scl<=1'b0  ;     r_sda<=DEVICE_MEASURE[5];
end
128:    begin r_scl<=1'b1 ;     end

129:    begin  r_scl<=1'b0  ;     r_sda<=DEVICE_MEASURE[4];
end
130:    begin r_scl<=1'b1 ;     end

131:    begin  r_scl<=1'b0  ;     r_sda<=DEVICE_MEASURE[3];
end
132:    begin r_scl<=1'b1 ;     end

133:    begin  r_scl<=1'b0  ;     r_sda<=DEVICE_MEASURE[2];
end
134:    begin r_scl<=1'b1 ;     end

135:    begin  r_scl<=1'b0  ;     r_sda<=DEVICE_MEASURE[1];
end
136:    begin r_scl<=1'b1 ;     end

137:    begin  r_scl<=1'b0  ;     r_sda<=DEVICE_MEASURE[0];
end
138:    begin r_scl<=1'b1 ;     end
139:      begin r_scl<=1'b0;      end

140:    begin ctl_sda<=1'b0;    end
141:    begin r_scl<=1'b1;        end
142:    begin
        r_scl<=1'b0;
        if(sda==1'b0) begin
            cnt<=143;r_led[5]<=0;
        end else
```

```
                cnt<=141;
            end
 143:     begin ctl_sda<=1'b1;     end
//STOP
 144:    begin r_scl<=1'b1;         end
 145:    begin r_sda<=1'b0;         end
 146:    begin r_sda<=1'b1;         end
////读取测量值----------------------------------------------------------//
 2147:   begin ctl_sda<=1'b1; end
 //START
 2148:   begin r_scl<=1'b1;          end
 2149:   begin r_sda<=1'b1;          end
 2150:   begin r_sda<=1'b0;          end
 //ADDRESS+WRITE 8'H46
 2151:   begin r_scl<=1'b0;   r_sda<=DEVICE_ADDR_R[7]; end
 2152:   begin r_scl<=1'b1;                                  end

 2153:   begin r_scl<=1'b0;   r_sda<=DEVICE_ADDR_R[6]; end
 2154:   begin r_scl<=1'b1;     end

 2155:   begin r_scl<=1'b0;   r_sda<=DEVICE_ADDR_R[5]; end
 2156:   begin r_scl<=1'b1;     end

 2157:   begin r_scl<=1'b0;   r_sda<=DEVICE_ADDR_R[4]; end
 2158:   begin r_scl<=1'b1;     end

 2159:   begin r_scl<=1'b0;   r_sda<=DEVICE_ADDR_R[3]; end
 2160:   begin r_scl<=1'b1;     end

 2161:   begin r_scl<=1'b0;   r_sda<=DEVICE_ADDR_R[2]; end
 2162:   begin r_scl<=1'b1;     end

 2163:   begin r_scl<=1'b0;   r_sda<=DEVICE_ADDR_R[1]; end
 2164:   begin r_scl<=1'b1;     end
```

```
2165:   begin r_scl<=1'b0;    r_sda<=DEVICE_ADDR_R[0]; end
2166:   begin r_scl<=1'b1;      end
2167:   begin r_scl<=1'b0;    r_sda<=1; end

2168:   begin ctl_sda<=1'b0; end
2169:   begin r_scl<=1'b1;      end
2170:   begin r_scl<=1'b0;
        if(sda==1'b0)
            begin
                cnt<=2171;        end
            else
                cnt<=2169;
        end

//READ
2171:   begin r_scl<=1'b1 ;      end
2172:   begin data[15]=sda ;     end
2173:   begin r_scl<=1'b0 ;     end

2174:   begin r_scl<=1'b1 ;     end
2175:   begin data[14]=sda ;    end
2176:   begin r_scl<=1'b0 ;     end

2177:   begin r_scl<=1'b1 ;     end
2178:   begin data[13]=sda ;    end
2179:   begin r_scl<=1'b0 ;     end

2180:   begin r_scl<=1'b1 ;     end
2181:   begin data[12]=sda ;    end
2182:   begin r_scl<=1'b0 ;     end

2183:   begin r_scl<=1'b1 ;     end
2184:   begin data[11]=sda ;    end
```

```
2185:   begin r_scl<=1'b0 ;      end

2186:   begin r_scl<=1'b1 ;      end
2187:   begin data[10]=sda ;    end
2188:     begin r_scl<=1'b0;       end

2189:   begin r_scl<=1'b1;    end
2190:   begin data[9]=sda ;    end
2191:   begin r_scl<=1'b0;    end

2192:     begin r_scl<=1'b1;   end
2193:   begin data[8]=sda ;    end
2194:   begin r_scl<=1'b0;         end

2195:   begin r_scl<=1'b1;ctl_sda<=1'b1;r_sda<=1'b0;end

2196:   begin r_scl<=1'b1; end

2197:   begin r_scl<=1'b0; ctl_sda<=1'b1; r_sda<=1'b1;end

2198:   begin ctl_sda<=0;end

2199:   begin r_scl<=1'b1 ;      end
2200:   begin data[7]=sda ;      end
2201:   begin r_scl<=1'b0 ;      end

2202:   begin r_scl<=1'b1 ;      end
2203:   begin data[6]=sda ;      end
2204:   begin r_scl<=1'b0 ;      end

2205:   begin r_scl<=1'b1 ;      end
2206:   begin data[5]=sda ;      end
2207:   begin r_scl<=1'b0 ;      end
```

```
                2208:    begin r_scl<=1'b1 ;      end
                2209:    begin data[4]=sda ;      end
                2210:    begin r_scl<=1'b0 ;      end

                2211:    begin r_scl<=1'b1 ;      end
                2212:    begin data[3]=sda ;      end
                2213:    begin r_scl<=1'b0 ;      end

                2214:    begin r_scl<=1'b1 ;      end
                2215:    begin data[2]=sda ;      end
                2216:     begin r_scl<=1'b0;       end

                2217:    begin r_scl<=1'b1;       end
                2218:    begin data[1]=sda ;      end
                2219:    begin r_scl<=1'b0;       end

                2220:     begin r_scl<=1'b1;       end
                2221:    begin data[0]=sda ;      end
                2222:    begin r_scl<=1'b0;       end

                2223:     begin ctl_sda<=1'b1; r_sda<=1;end

              default:begin r_scl<=1'b1; ctl_sda<=1'b0;    end
            endcase
        end
    end

endmodule
```

2-3 数据处理，分成 64 个亮度等级（子程序）

```
module Data_Process(data_input,data_output);

    input          [15:0]   data_input;
    output   reg   [7:0]    data_output;
```

```
    always@(data_input)
        begin
            data_output<=(data_input>>10);
        end
endmodule
```

2-4 十进制转换成二进制（子程序）

```
module BIN_To_BCD(number, tens, ones);
    // I/O Signal Definitions
    input           [7:0]   number;
    output  reg     [3:0]   tens;
    output  reg     [3:0]   ones;

    // Internal variable for storing bits
    reg             [19:0]  shift;
    integer i;

    always@(number)
    begin
        // Clear previous number and store new number in shift register
        shift[19:8] = 0;
       shift[7:0] = number[7:0];
        // Loop eight times
        for (i=0; i<8; i=i+1) begin
            if (shift[11:8] >= 5)
                shift[11:8] = shift[11:8] + 3;

            if (shift[15:12] >= 5)
                shift[15:12] = shift[15:12] + 3;

            if (shift[19:16] >= 5)
                shift[19:16] = shift[19:16] + 3;
```

```
            // Shift entire register left once
            shift = shift << 1;
        end

        // Push decimal numbers to output
        tens        = shift[15:12];
        ones        = shift[11:8];
    end

endmodule
```

2-5 光照显示程序，显示在数码管上（子程序）

```
module seg_led(tens,ones,segment_led_1,segment_led_2);

    input   [3:0]   tens;
    input   [3:0]   ones;
    output  [8:0]   segment_led_1;   //MSB~LSB = SEG,DP,G,F,E,D,C,B,A
    output  [8:0]   segment_led_2;   //MSB~LSB = SEG,DP,G,F,E,D,C,B,A

    reg     [8:0]   seg  [9:0];
    initial
        begin
            seg[0] = 9'h3f;     //   0
            seg[1] = 9'h06;     //   1
            seg[2] = 9'h5b;     //   2
            seg[3] = 9'h4f;     //   3
            seg[4] = 9'h66;     //   4
            seg[5] = 9'h6d;     //   5
            seg[6] = 9'h7d;     //   6
            seg[7] = 9'h07;     //   7
            seg[8] = 9'h7f;     //   8
            seg[9] = 9'h6f;     //   9
        end
    assign segment_led_1 = seg[tens];
```

```
    assign segment_led_2 = seg[ones];
endmodule
```

2-6 光照显示程序，显示在 led 点阵上（子程序）

```
module led_scan(clk,rst,number,tens,ones,row,line);

    input                clk;
    input                rst;
    input        [7:0]   number;
     input       [3:0]   tens;
     input       [3:0]   ones;
    output   reg [15:0]  row;
    output   reg [7:0]   line;

    wire      clk16Hz;
    clk_div #
    (
    .COUNTER_NUM (100000)
    )
    clk16Hz_uut
    (
    .clk(clk),           //system clk
    .rst(rst),       //system rst
    .invert(clk16Hz)       //invert signal
    );

    reg       [3:0] temp_num;
    reg       [3:0] temp_cnt=0;
    reg             get_num_flag =1;
    always@(posedge clk16Hz)
        begin
            if (get_num_flag == 1)
                begin
                    temp_num <= number[5:2];
```

```
                get_num_flag <= 0;
            end
        else if (temp_num == 15)
            begin
                temp_num <= 0;
                get_num_flag <= 1;
                temp_cnt <= temp_cnt + 1;
            end
        else
            temp_num <= temp_num + 1;
    end

wire    clk10KHz;
clk_div #
(
.COUNTER_NUM (1200)
)
clk10KHz_uut
(
.clk(clk),          //system clk
.rst(rst),      //system rst
.invert(clk10KHz)    //invert signal
);

always@(posedge clk10KHz)
    begin
        if (!rst == 1)
            row <= 16'h0001;
        else
            row <= {row[14:0],row[15]};
    end


reg     [31:0]  mem     [18:0];
```

```
reg     [7:0]    cache    [15:0];
initial
    begin
        mem[0]      = {~8'h3E,~8'h41,~8'h41,~8'h3E};// 0
        mem[1]      = {~8'h11,~8'h31,~8'h7f,~8'h01};  // 1
        mem[2]      = {~8'h23,~8'h45,~8'h49,~8'h31}; // 2
        mem[3]      = {~8'h22,~8'h49,~8'h49,~8'h36}; // 3
        mem[4]      = {~8'h0C,~8'h14,~8'h24,~8'h7f}; // 4
        mem[5]      = {~8'h7A,~8'h49,~8'h49,~8'h46};// 5
        mem[6]      = {~8'h3E,~8'h49,~8'h49,~8'h26}; // 6
        mem[7]      = {~8'h40,~8'h45,~8'h49,~8'h70}; // 7
        mem[8]      = {~8'h36,~8'h49,~8'h49,~8'h36}; // 8
        mem[9]      = {~8'h32,~8'h49,~8'h49,~8'h3E}; // 9
        mem[10] = {~8'h00,~8'h00,~8'h00,~8'h00};     // 全灭
        mem[11] = {~8'h08,~8'h08,~8'h08,~8'h08};     // 负号
        mem[12] = {~8'hC0,~8'hDE,~8'h21,~8'h21};   // 摄氏度符号
        mem[13] = {~8'h38,~8'h7C,~8'hC7,~8'hD7};   // 灯泡左边 4 列
        mem[14] = {~8'hC7,~8'h7C,~8'h38,~8'h00};   // 灯泡右边 4 列
        mem[15] = {~8'h00,~8'h18,~8'h3C,~8'h7E};   // 箭头左边 4 列
        mem[16] = {~8'hDB,~8'h99,~8'h18,~8'h18};   // 箭头右边 4 列

        cache[0]<=mem[15][31:24];
        cache[1]<=mem[15][23:16];
        cache[2]<=mem[15][15:8];
        cache[3]<=mem[15][7:0];
        cache[4]<=mem[16][31:24];
        cache[5]<=mem[16][23:16];
        cache[6]<=mem[16][15:8];
        cache[7]<=mem[16][7:0];
        cache[8]<=mem[15][31:24];
        cache[9]<=mem[15][23:16];
        cache[10]<=mem[15][15:8];
        cache[11]<=mem[15][7:0];
        cache[12]<=mem[16][31:24];
```

```
            cache[13]<=mem[16][23:16];
            cache[14]<=mem[16][15:8];
            cache[15]<=mem[16][7:0];
        end

    always @ (row)
        begin
            case (row)
                16'h0001 : line <= cache[temp_cnt+0];
                16'h0002 : line <= cache[temp_cnt+1];
                16'h0004 : line <= cache[temp_cnt+2];
                16'h0008 : line <= cache[temp_cnt+3];
                16'h0010 : line <= cache[temp_cnt+4];
                16'h0020 : line <= cache[temp_cnt+5];
                16'h0040 : line <= cache[temp_cnt+6];
                16'h0080 : line <= cache[temp_cnt+7];
                16'h0100 : line <= cache[temp_cnt+8];
                16'h0200 : line <= cache[temp_cnt+9];
                16'h0400 : line <= cache[temp_cnt+10];
                16'h0800 : line <= cache[temp_cnt+11];
                16'h1000 : line <= cache[temp_cnt+12];
                16'h2000 : line <= cache[temp_cnt+13];
                16'h4000 : line <= cache[temp_cnt+14];
                16'h8000 : line <= cache[temp_cnt+15];

            endcase

        end

endmodule
```

2-7 光照显示程序，显示在三色灯上（子程序）

```
module rgb_led(clk,rst,pwm_duty,led1,led2);
```

```
    input                   clk;
    input                   rst;
    input          [7:0]    pwm_duty;
    output   reg   [2:0]    led1;
    output   reg   [2:0]    led2;

    reg        [5:0]     count;
    localparam   cycle=6'd63;
        always@(posedge clk)
            if (!rst)
                count<=0;
            else if(count==cycle-1)
                count<=0;
            else
                count<=count+1;
        always@(*)
            begin
            if(count<pwm_duty)
                begin
                    led1<=3'b000;
                    led2<=3'b000;
                end
            else
                begin
                    led1<=3'b111;
                    led2<=3'b111;
                end
            end

endmodule
```

2-8 时钟信号的产生（子程序）

```
module clk_div #
(
```

```
    parameter    COUNTER_NUM = 3464      //period      =      (3464**2)*2      =
24_000_000 = 2s
)
(
    //INPUT
    input          clk,      //system clk
    input          rst,      //system rst
    output   reg  invert                    //invert signal
);
    reg       [31:0]         cnt;
    //generate invert signal);
    always @ (posedge clk or negedge rst)
        begin
            if(!rst == 1)
                begin
                    cnt <= 32'd0;
                    invert <= 0;
                end
            else
                begin
                    if(cnt >= COUNTER_NUM - 1)
                        begin
                            invert <= ~invert;
                            cnt <= 1'b0;
                        end
                    else
                        begin
                            cnt <= cnt + 1'b1;
                        end
                end
        end
endmodule
```