实验7指南

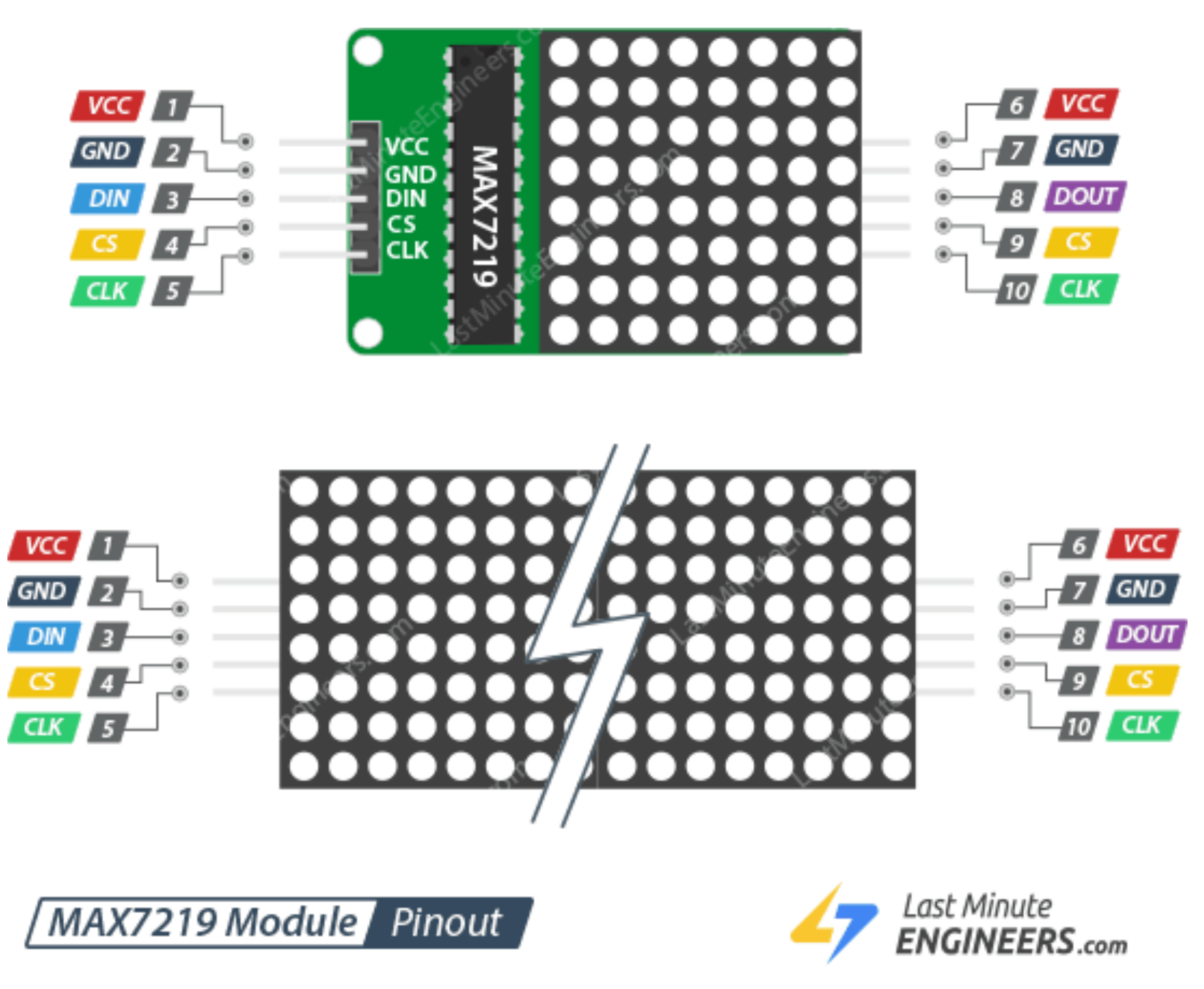
请跟随实验指南完成实验，完成文档中所有的 `TASK` 。 `BONUS` 部分的内容完成可作为加分，但报告的总分不应超过100分。请下载此指南作为实验报告模版，将填充完成的实验报告导出为PDF格式，并命名为“学号_姓名_lab7.pdf”，上传至学在浙大平台。下载请点击 这里 。

1 LED 矩阵

本实验中所使用的 LED 矩阵为 8x8 的点阵，使用 MAX7219 驱动。MAX7219 是一款集成了 8x8 LED 矩阵驱动电路的芯片，可以通过 SPI 接口控制。

1.1 引脚定义

MAX7219 的引脚定义如下图所示：



> 图片来源自 https://lastminuteengineers.com

- 输入端接口
  - **VCC:** 电源。连接到5V。因为显示器需要很大的电流（在最大亮度时高达1A），最好使用外部电源而不是 3568 板的5V电源。如果你想使用 3568 板的 5V 电源，请将亮度保持在 25% 以下，以避免电压调节器过热。
  - **GND:** 地。连接到地。
  - **DIN:** 数据输入。将其连接到 3568 板的任一空闲 GPIO 引脚。DIN的数据仅在CLK处于上升沿的时候有效。
  - **CS/LOAD:** 片选。将其连接到 3568 板的任一空闲 GPIO 引脚。作用为是控制串口通讯，其为低电平时，串行数据才被载入寄存器，高电平时会被锁存。
  - **CLK:** 时钟。将其连接到 3568 板的任一空闲 GPIO 引脚。
- 输出端接口
  - **VCC:** 电源。连接到下一个 MAX7219 的 VCC 引脚。
  - **GND:** 地。连接到下一个 MAX7219 的 GND 引脚。
  - **DOUT:** 数据输出。将其连接到下一个 MAX7219 的 DIN 引脚。
  - **CS/LOAD:** 片选。将其连接到下一个 MAX7219 的 CS/LOAD 引脚。
  - **CLK:** 时钟。将其连接到下一个 MAX7219 的 CLK 引脚。

1.2 内置寄存器功能

> 以下内容摘录自 https://juejin.cn/post/6976559353666994189

通过查阅 max7219 的 **datesheet** 可以发现，max7219芯片内不同的地址存储的值都有相应的作用。

0x01-0x08对应的是led 1到8行的显示通过8位二进制数来控制这一行led的显示状况，0 led熄灭，1 led点亮。

比如说 0x3C 转换成二进制就是 00111100，该行的led的显示模式就是 ○○●●●●●●○○

地址	寄存器名称	作用
0x09	译码模式寄存器	0 为关闭译码模式
0x0b	扫描限制寄存器	设置扫描led的行数 1-8
0x0a	亮度调节寄存器	16位调节亮度
0x0c	关断模式寄存器	0: 关断状态； 1: 正常操作状态
0x0f	显示测试寄存器	0: 正常模式； 1: 测试模式（全部点亮）

1.3 点亮 LED 矩阵

参考资料

1. 第二个树莓派应用——LED点阵
2. Interfacing MAX7219 LED Dot Matrix Display with Arduino
3. 树莓派学习笔记——使用文件IO操作GPIO SysFs方式

`TASK1` 请设计接线方案，使得 3568 板能够在之后的实验中使用 LED 矩阵。画出你的连线示意图，并拍摄你实际连接的板卡照片。（5分）

`TASK2` 编写 C 程序，采用 Arduino-ish 库或虚拟文件系统（或其他你选择的库）访问 GPIO，实现在矩阵上显示文字或图案。请在下方给出你的源代码（需有详细注释），对你选用的方式（库 / 虚拟文件系统）做出阐释，并对代码关键步骤进行解释。同时需附上 LED 矩阵成功显示文字或图案的照片。（20分）

2 字符设备驱动

2.1 Linux内核模块

> 此节内容摘录自 Linux驱动基本知识 - Linux内核模块

Linux是一个跨平台的操作系统，支持众多的设备，在Linux内核源码中有超过50%的代码都与设备驱动相关。 Linux为宏内核架构，如果开启所有的功能，内核就会变得十分臃肿。 内核模块就是实现了某个功能的一段内核代码，在内核运行过程，可以加载这部分代码到内核中， 从而动态地增加了内核的功能。基于这种特性，我们进行设备驱动开发时，以内核模块的形式编写设备驱动，只需要编译相关的驱动代码即可，无需对整个内核进行编译。

模块是具有独立功能的程序，它可以被单独编译，但不能独立运行， 在运行时它被链接到内核作为内核的一部分在内核空间运行，这与运行在用户空间的进程是不一样的。 模块由一组函数和数据结构组成，用来实现一种文件系统、一个驱动程序和其他内核上层功能。 因此内核模块具备如下特点：

- 模块本身不被编译入内核映像，这控制了内核的大小。
- 模块一旦被加载，它就和内核中的其它部分完全一样。

我们编写的内核模块，经过编译，最终形成.ko为后缀的ELF文件。我们可以使用file命令来查看它。

2.2 编译内核

2.3.1 交叉编译环境配置

> 此节内容参考自 https://doc.embedfire.com/linux/imx6/driver/zh/latest/linux_driver/exper_env.html

设备驱动是具有独立功能的程序，它可以被单独编译，但不能独立运行，在运行时它被链接到内核作为内核的一部分在内核空间运行。也因此想要我们写的内核模块在某个版本的内核上运行，那么就必须在该内核版本上编译它，如果我们编译的内核与我们运行的内核具备不相同的特性，设备驱动则可能无法运行。

由于嵌入式 Linux 板卡的性能有限，编译内核需要较长时间，因此我们需要使用交叉编译的方式，在PC上编译内核，然后将编译好的内核模块传输到板卡上。

可以参考 **编译环境搭建** 下载并解压 **Firefly_Linux_SDK源码包**。

> 需要选择和你之前烧录固件相同版本的 Linux 内核，或者也可以选择编译好 SDK 以后重新烧录一次固件。
>
> 这里仍然只有百度网盘，且 SDK 源码很大（十几GB），超过了浙大云盘的限额，所以要做好心理准备。

同时配置好相关的交叉编译环境（推荐使用docker，并参考以上教程使用如下的 dockerfile）：

```
FROM ubuntu:18.04
MAINTAINER firefly "service@t-firefly.com"

ENV DEBIAN_FRONTEND=noninteractive

RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak
RUN sed -i 's@http://.*ubuntu.com@http://repo.huaweicloud.com@g' /etc/apt/sources.li

RUN apt update

RUN apt install -y build-essential crossbuild-essential-arm64 \
    bash-completion vim sudo locales time rsync bc python

RUN apt install -y repo git ssh libssl-dev liblz4-tool lib32stdc++6 \
    expect patchelf chrpath gawk texinfo diffstat binfmt-support \
    qemu-user-static live-build bison flex fakeroot cmake \
    unzip device-tree-compiler python-pip ncurses-dev python-pyelftools \
    subversion asciidoc w3m dblatex graphviz python-matplotlib cpio \
    libparse-yapp-perl default-jre patchutils swig expect-dev u-boot-tools

RUN apt update && apt install -y -f

# language support
RUN locale-gen en_US.UTF-8
ENV LANG en_US.UTF-8

# switch to a no-root user
RUN useradd -c 'firefly user' -m -d /home/firefly -s /bin/bash firefly
RUN sed -i -e '/\%sudo/ c \%sudo ALL=(ALL) NOPASSWD: ALL' /etc/sudoers
RUN usermod -a -G sudo firefly

USER firefly
WORKDIR /home/firefly
```

警告

所给的参考资料使用的上位机架构为 x86_64，如果你使用的上位机架构为 arm64（对说的就是你 Apple 的 M 系列芯片），那么就需要对以上教程中的相关配置进行修改。（实际上是更简单了，因为如果使用 docker 中的 Linux 进行编译，那么就不需要进行交叉编译，直接编译即可）

`TASK3` 请参考以上教程配置 docker 镜像，创建容器并启动。在 docker 中使用交叉编译工具编译 C 语言的 "Hello, world" 程序，并将编译后的二进制文件传输到 3568 板上运行。请给出你在 docker 中成功交叉编译产生二进制文件的截图以及在 3568 板上成功运行此程序的截图。（15分）

2.3.2 开始编译内核

请参考 **Kernel 使用**，在前一节中配置好的环境中编译 Linux 内核。

`TASK4` 请参考以上教程，编译内核。请给出编译过程中的截图，以及编译完成后生成的内核镜像文件的截图。若选择与之前固件内核版本不同需要重新烧录，则需要同时放上烧录成功的截图。（15分）

2.3 驱动编写

2.3.1 第一个内核模块

请先参考 **第一个内核模块** 和 **Linux驱动实践：带你一步一步编译内核驱动程序**，尝试一个最简单 hello module 框架。需要注意 `Kconfig` 和 `Makefile` 的编写。

`TASK5` 将以上框架编译成 `.ko` 文件后，传输到 3568 板上，加载此驱动后再卸载此驱动，并在过程中观察串口输出，并使用如 `lsmod` 、 `dmesg` 等命令查看驱动加载状态。请给出相应的截图和对截图的解释。（10分）

2.3.2 编写字符设备驱动程序

完成以上内容后，请编写字符设备驱动程序，通过内核 GPIO 库访问引脚，能将 `write()` 送来的单个字符在矩阵上显示出来。

> 以下内容摘录自 https://zhuanlan.zhihu.com/p/137636768

Linux 系统将设备分为三大类：字符设备、块设备和网络设备。字符设备是其中较为基础的一类，它的读写操作需要一个字节一个字节的进行，不能随机读取设备中的某一数据，即要按照先后顺序。举例来说，比较常见的字符设备有鼠标、键盘、串口等。

字符设备驱动所做的工作主要是添加、初始化、删除 cdev 结构体，申请、释放设备号，填充 `file_operations` 结构体中的功能函数，比如 `open()` 、 `read()` 、 `write()` 、 `close()` 等。当我们创建一个字符设备时，一般会在 `/dev` 目录下生成一个设备文件，Linux 用户层的程序就可以通过这个设备文件来操作这个字符设备。

参考资料

1. 字符设备驱动
2. Linux驱动篇(五)--字符设备驱动(一)
3. GPIO 使用

`TASK6` 编写字符设备驱动程序，通过内核GPIO库访问引脚，能将 `write()` 送来的单个字符在矩阵上显示出来。请给出源代码（需有详细注释）以及对关键部分的解释。将驱动编译并加载后，请编写C语言程序（或直接使用 shell 脚本）测试此驱动，并给出测试使用的程序、终端中的截图以及 LED 板显示的照片。（20分）

3 驱动的应用

`TASK7` 编写 Linux 应用程序，能通过 MQTT 协议连接自己的 MQTT broker，将订阅收到的文字在 LED 矩阵上流动显示出来。要求给出源代码（有详细注释）以及对关键部分的解释。同时给出 LED 矩阵成功显示的现象照片。此 TASK 要求使用 TASK6 所编写的字符设备驱动程序，如果是通过 TASK2 的方式控制 LED，则此 TASK 分数减半。（10分）

`TASK8` 在 TASK7 的基础上，实现 Lab6 的 DHT-11 的数据发布到自己的 MQTT broker 上后显示在 LED 矩阵上。要求给出源代码（有详细注释）以及对关键部分的解释。同时给出 LED 矩阵成功显示的现象照片。（5分）

4 拓展内容

拓展内容要求对驱动进行进一步细化，满足以下要求：

`BONUS1` 设备驱动程序能将 `write()` 送来的字符串以每个字母停留 500ms 的速度依次显示。（5分 Bonus）

`BONUS2` BONUS1 中的 `write()` 函数是非阻塞类型的，每500ms一个字符的显示是由内核定时器队列实现的。（5分Bonus）

5 讨论和心得

> 请认真填写本模块，若不填写或胡乱填写将酌情扣分，写明白真实情况即可。

请在此处填写实验过程中遇到的问题及相应的解决方式。

由于本实验为新实验，可能存在不足之处，欢迎同学们对本实验提出建议。

个人水平有限，如您发现文档中的疏漏欢迎 Issue!