华东师范大学软件学院课程项目报告

课程名称：嵌入式 Linux 编程与应用　　**成绩**：

项目名称：基于 ZYBO 板、MINI12864 液晶屏、8*8 LED 点阵屏的 GAMMER 游戏设备

姓名：林涵菲　　**学号**：10122510319　　**班级**：软件 3 班　　**同组成员**：梁雨霏

基于 ZYBO 板、MINI12864 液晶屏、8*8 点阵屏的 GAMMER 游戏设备

目录

一.项目背景

（一）项目简介

我们的作品“GAMMER”是一个基于 ZYBO 板的小型多模式游戏机，由一个 74HC595 级联的 8*8 LED 点阵外设、一个 MINI12864 液晶显示屏外设和一个面包板上的外接按钮连接 ZYBO 板组成。

“GAMMER”游戏机的模式一共分为三种，可以由外接按钮自由切换。第一种模式是欢迎模式，为上电默认模式，在液晶屏上出现“WELCOME”和我们的信息；第二种模式是在 LED 点阵上出现一个包含闪烁和移动效果的“ I love U”字样，同时在液晶屏上显示”I LOVE U”字样；第三种模式是用 ZYBO 板上自带的四个按钮玩在 8*8 LED 点阵上玩贪吃蛇，并且在液晶屏上显示游戏状态，包括当前玩家得分、玩家最高分和游戏结束时的“GAME OVER”字样，并且当玩家超过五分时贪吃蛇的速度会加快。

我们的代码中没有加入任何外设封装好的库，全部手动编写驱动程序，因此我们的代码已经超过了 1000 行。以下是我们作品的部分照片：

模式一：欢迎模式（如图 1.1）

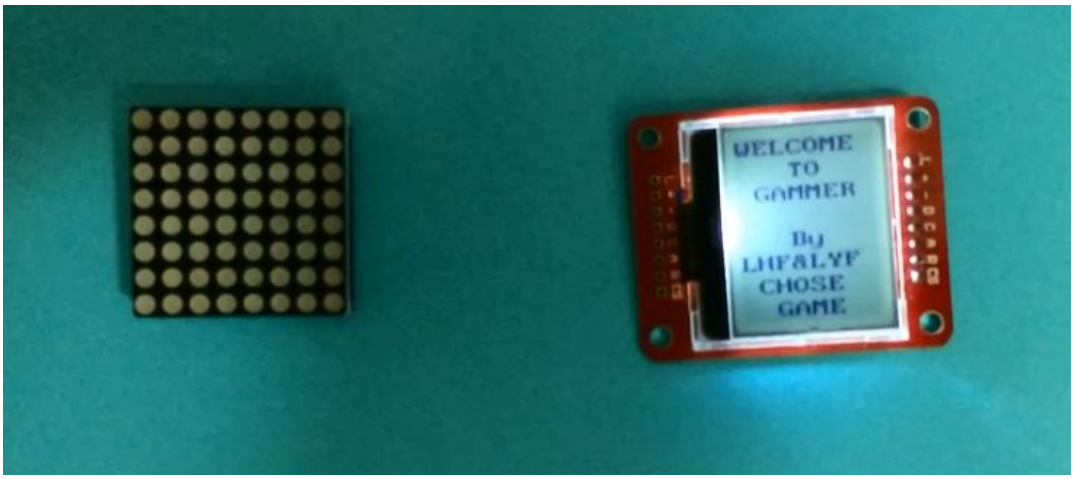


图 1.1 欢迎模式

模式二：闪烁 I LOVE U 字样和移动 I LOVE U 字样模式（如图 1.2）

图 1.2 文字模式

模式三：贪吃蛇游戏模式（如图 1.3-1.5）

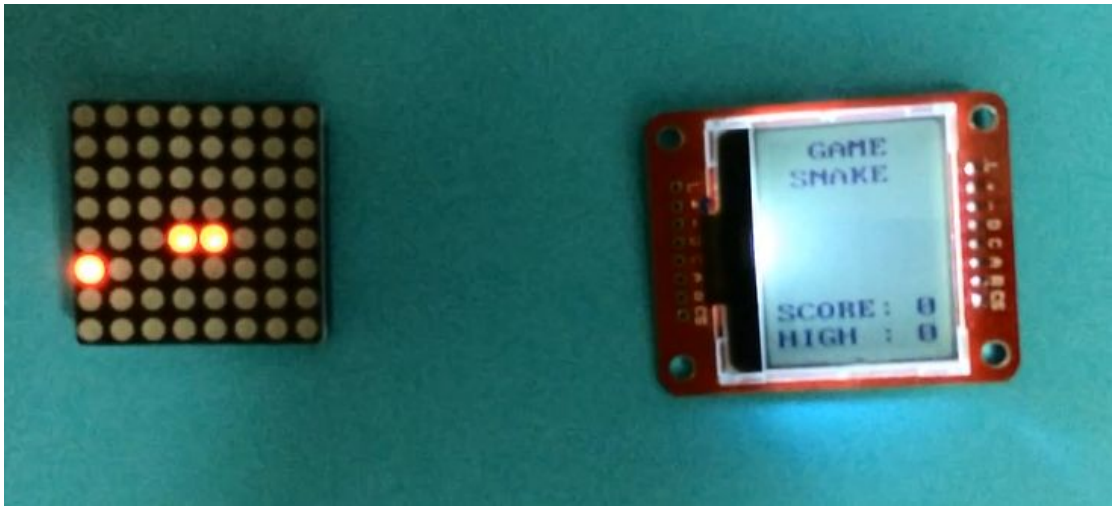


图 1.3 贪吃蛇游戏模式（初始界面）

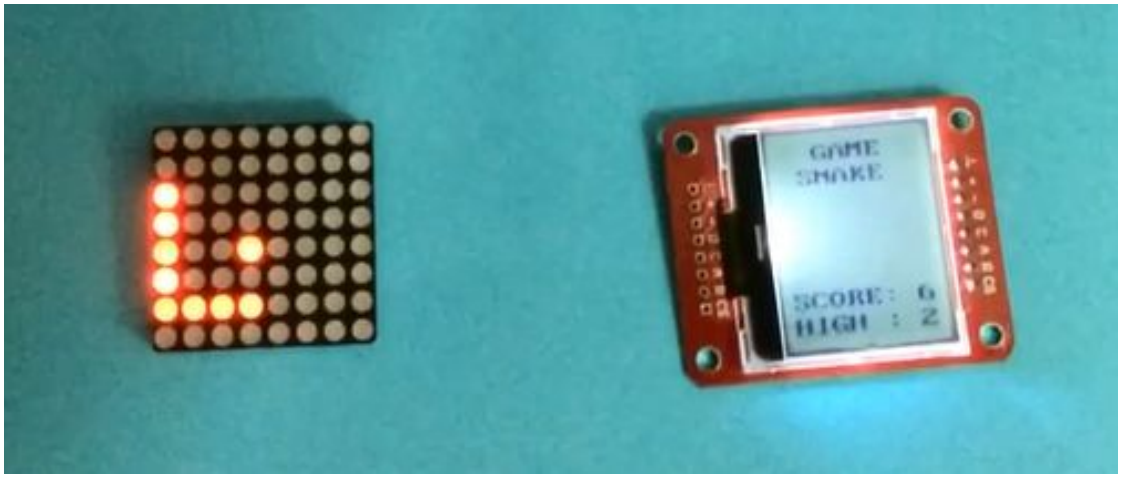


图 1.4 贪吃蛇游戏模式（游戏运行界面）

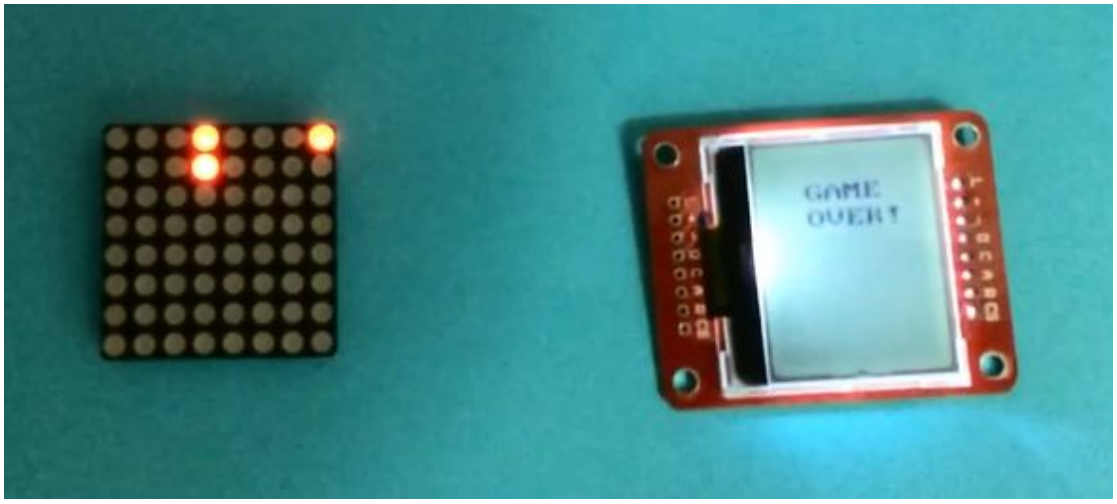


图 1.5 贪吃蛇游戏模式（游戏结束界面）

外接 OMRON 按钮（如图 1.6）控制模式转换：

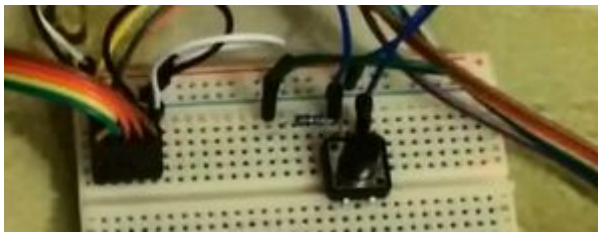


图 1.6 外接 OMRON 按钮

ZYBO 板内置按钮控（如图 1.7）制上下左右：

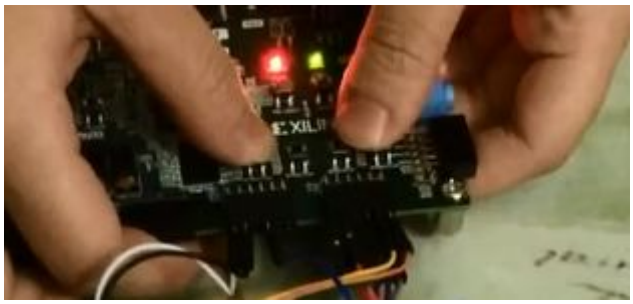


图 1.7 ZYBO 板内置按钮

（二）选题背景

1.ZYBO 板背景

ZYBO (Zynq Board) 是一款基于 Z-7010 而构建的功能丰富入门级嵌入式即用型软件和数字电路开发平台。 具有板上存储器、视频与音频 I/O、双模 USB、以太网和 SD 插槽等丰富配置，无需添加任何硬件即可使设计运行就绪。此外，还提供五个 Pmod 连接器，便于对任何设计进行升级。 ZYBO 可用于设计各种复杂程度的系统，例如运行多个服务器串联应用的完整操作系统，或者用来控制 LED 的简单裸机程序。

本项目就是在 ZYBO 板上进行的，通过 Pmod 口对 ZYBO 板进行扩展，实现我们想要实现的功能。

2.选用器件及其背景

本项目中我们选用的主要外设器件为 74HC595 级联的 8*8 LED 点阵和 MINI12864 液晶显示屏。

LED 显示屏与 PC 的数据传输方式有串行和并行两种，而使用串行传输可有效减少硬件设计的复杂程度，同时传输率也能得到保证。我们采用的 8*8 LED 点阵显示屏，运用的就是串行动态显示技术，它将引脚数量减少到 16 根。再配以两个 74HC595 芯片，采用译码技术，最终将引脚减少到 5 根。最终我们要通过时序逻辑，配合 ZYBO 板内置的按钮，共同控制 8*8 LED 点阵显示屏，动态显示我们想要图案。

12864 液晶显示屏是一种 128 像素*64 像素构成的显示屏的统称，我们所使

用的 MINI12864 液晶显示屏则是一套由 OpenJumper 开发的 SPI 接口的图形显示器，相对于并行接口的显示器，它使用更为方便，占用引脚资源更少。

3.为什么我们想要编写 GAMMER

通过查阅资料我们发现，用 ZYBO 板外接 LED 点阵和液晶屏的例子几乎没有，基本上都是用 Arduino 板外接这两个器件。然而 ZYBO 板与 Arduino 板相比还是有不少优点的，比如功能更加丰富。因此我们希望使用 Arduino 的配套外设，通过自己编写驱动程序，让它们能够连接 ZYBO 板并实现基本的功能。

而说到 GAMMER，是因为我们在最开始的时候是希望按照教程写一遍乒乓球游戏的，因为这样可以大大节约我们的工作量。但是后来我们对 LED 点阵产生了浓厚的兴趣，因此开始不满足于教程，而是考虑将自己平常编写的 C 语言贪吃蛇结合到嵌入式中来，更是一种创新。在我们的 GAMMER 中，贪吃蛇的速度是会变化的，因此可以高度模拟软件界面的贪吃蛇，并且配合液晶显示屏显示分数等信息，效果非常好。

（三）GAMMER 实现的主要功能

1.用 Vivado 设计 ZYBO 模块，并进行综合、实现和生成流文件，导出为 SDK
2.在 SDK 上编写程序，包括如下功能：

（1）控制 8*8 LED 点阵的图形显示
（2）控制 MINI12864 上的文字和数字显示
（3）接收 ZYBO 板内置按钮的输入信号
（4）接收 OMRON 外接按钮的输入信号

二.技术原理

（一）综述

本章节将重点介绍“GAMMER”所用到的硬件的原理，包括 ZYBO 板原理、8*8 LED 点阵原理、MINI12864 原理和外接按钮的原理。 其中 8*8 LED 点阵原理、MINI12864 原理都包含了详细的电路图或时序逻辑图，以及相应驱动的伪代码。 由于我们在编写程序之前做了非常充分的准备工作，对即将用到的外设查找了大量的相关资料甚至精确到逻辑门的电路图， 因此我们在没有加入任何外设支持库的前提下手动编写除了两个外设的驱动程序。

然而由于我们的时间和能力有限，不能保证本章节所述的所有内容完全正确，因为本章节所有内容都是基于已有资料通过我们自己的理解和推测写出来的，可能并不是那么完美。

（二）实现逻辑与电路图

1. 整体实现逻辑

“GAMMER”由一个 74HC595 级联的 8*8 LED 点阵外设、一个 MINI12864 液晶显示屏外设和一个面包板上的外接按钮连接 ZYBO 板组成。抽象的连线图如图 2.1 所示：

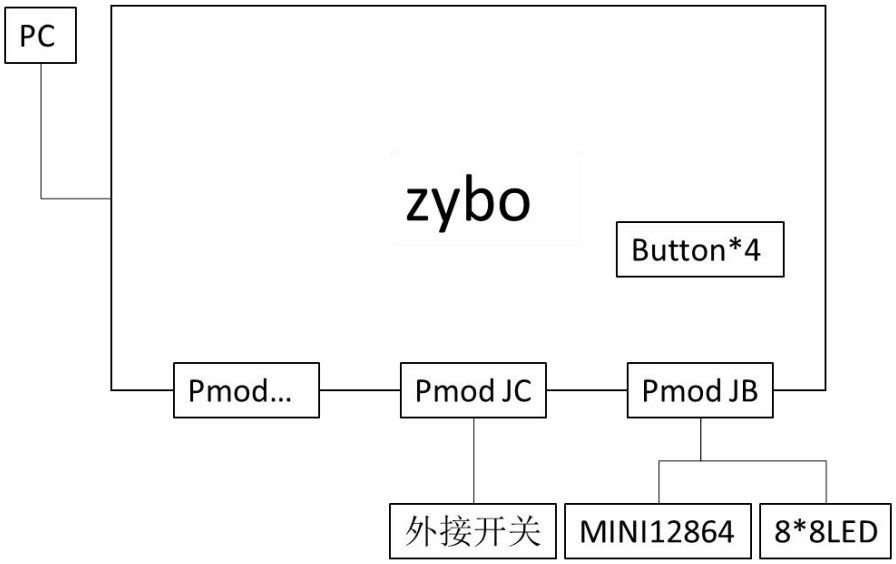


图 2.1 ZYBO 板抽象的连线图

3. ZYBO 板控制系统

尽管我们使用的外设都是基于 ARDUINO 单片机的，但是我们可以通过更改引脚状态和自己编写对应外设的驱动程序使外设连在 ZYBO 板上也能运作。要做到这一点，我们首先需要了解 ZYBO 版的控制系统。

（1）ZYBO 板电路原理

ZYBO 板是一款 ZYNQ 板，ZYNQ 中包含了两个部分，双核的 ARM 和 FPGA。根据 Xilinx 提供的手册，ARM 模块被称为 PS，而 FPGA 模块被称为 PL。这有点像 Xilinx 以前推出的 powerPC+FPGA 平台。ZYNQ 的内部结构如图 2.2 所示。

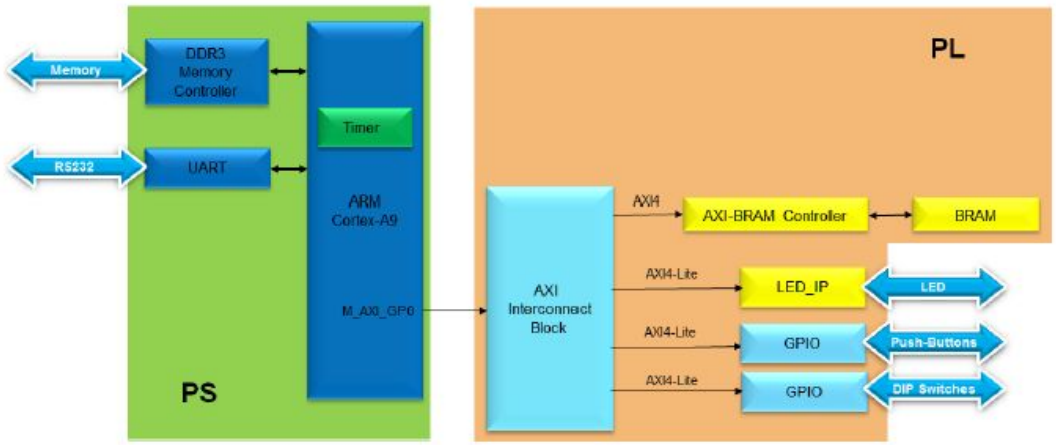


图 2.2 ZYBO 板内部结构图

从图中可以看到，ZYNQ 的绝大多数外设都是 PL 逻辑部分相连，比如说 GPIO，IIS，XADC 等等，所以如果我们要使用这些外设的话必须在 PL 逻辑部分对其进行配置。在本次实验中，我们需要对 PL 的 GPIO 口进行配置，我们在 PL 上增加了两个 8 引脚的 GPIO 模块，其中一个用于接收输入，另一个用于输出。

我们将在 Vivado 中对引脚进行配置，之后导出 SDK 文件，新建板级支持包并更新默认代码，将程序烧入 ZYBO 板的 FPGA 中，对其进行控制。

（2）利用 VIVADO 设计基于 ARM Cortex-A9 的处理器

首先我们要利用 VIVADO 设计 ZYNQ 处理器系统（lab1 中已经做过一次，这里不再赘述）。在此基础上，我们需要向 PL 中添加 IP 核，并且在 PS 中配置相应的主从端口。

我们需要在 lab1 的基础模块设计图上添加两个 GPIO 模块，先在 PS 中使能 M_AXI_GP0 等接口，之后添加 AXI_GPIO 模块，两个模块一个设置为输出一个设计为输入，并且都为 8 位模块，并自动进行连接，连接之后的模块设计如图 2.3 所示：

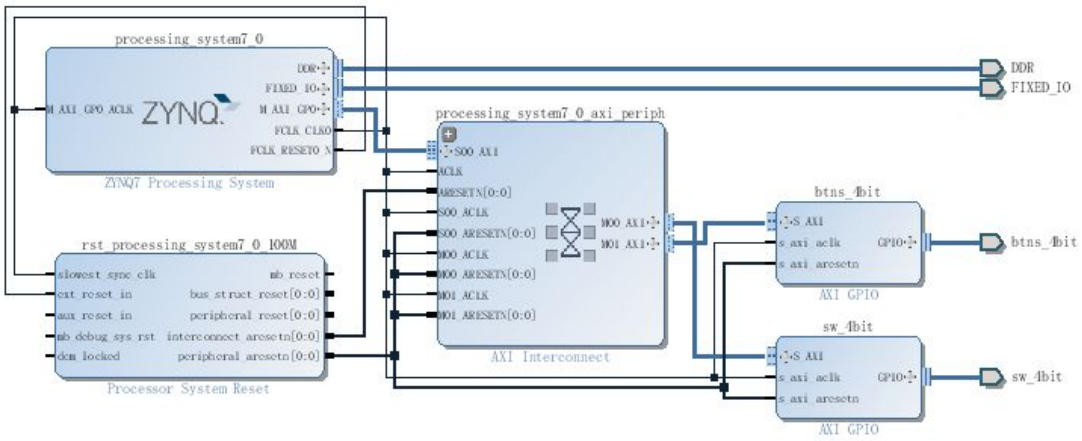


图 2.3 ZYBO 板 GPIO 模块设计图

之后我们对设计进行综合，得到综合之后的网表如图 2.4 所示：

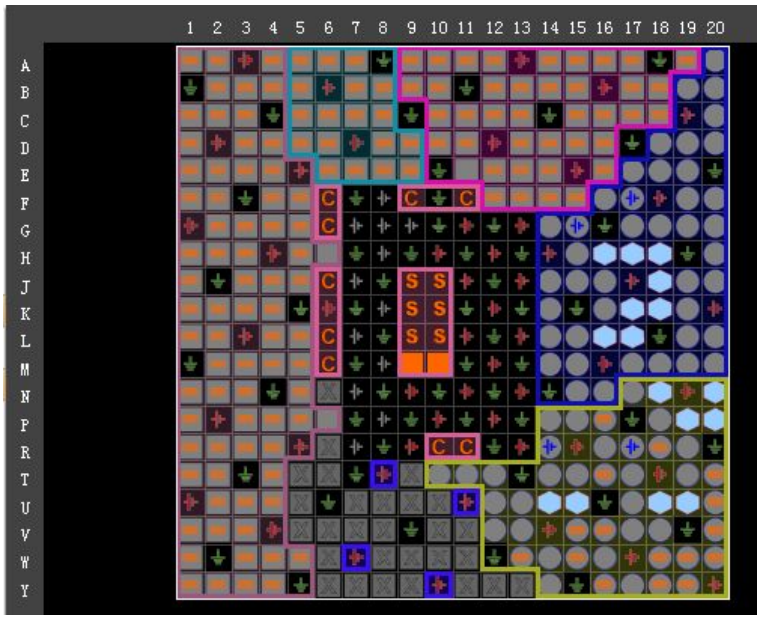


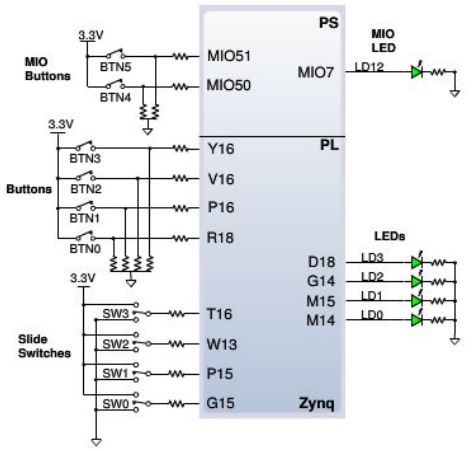
图 2.4 ZYBO 板网表图

我们对 IO 端口进行配置，使一个 GPIO 端口对应一个物理引脚，引脚对应关系如表 2.1：

表 2.1 ZYBO 板 IO 端口对应关系

SW[7]	SW[6]	SW[5]	SW[4]	SW[3]	SW[2]	SW[1]	SW[0]
V15	W15	W13	T16	Y16	V16	P16	R18
BT[7]	BT[6]	BT[5]	BT[4]	BT[3]	BT[2]	BT[1]	BT[0]
Y18	Y19	W18	W19	T20	U20	V20	W20

对应引脚如图 2.5 所示：



Pmod JB (Hi-Speed)	Pmod JC (Hi-Speed)
JB1: T20	JC1: V15
JB2: U20	JC2: W15
JB3: V20	JC3: T11
JB4: W20	JC4: T10
JB7: Y18	JC7: W14
JB8: Y19	JC8: Y14
JB9: W18	JC9: T12
JB10: W19	JC10: U12

图 2.5 ZYBO 板引脚对应图

之后我们就可以对设计进行实现、生成和导出 SDK 了，具体步骤这里不再赘述。

3. 8*8 LED 点阵逻辑

（1）初探 8*8 LED 点阵

我们使用的 8*8 LED 点阵采用的是串行动态显示技术，配以两个 74HC595 芯片，采用译码技术，将原本需要的 16 根引脚减少到 5 根，极大地节约了引脚，但是换来的是更为复杂的内部逻辑。因此我们如果需要在 8*8 LED 显示屏上显示我们想要的图案，就必须非常了解它的内部逻辑和驱动方法。在查阅了大量资料之后，我们整理出了一套关于 8*8 点阵的资料，下面从最底层的逻辑门开始讲起。

（2）艰辛的探索过程——8*8 LED 点阵的工作原理

①从 D 触发器谈起

想要了解要了解 8*8 LED 点阵的控制逻辑，首先需要明白 74HC595 芯片的工作原理。而了解 74HC595 芯片工作原理的前提是了解 D 触发器的逻辑。因此本小节将从 D 触发器谈起，详细介绍 8*8 LED 点阵的逻辑和控制方法，结合后文的逻辑代码方便读者理解如何用 C 语言直接编写 8*8 LED 点阵模块的驱动函数。

D 触发器是基于基本触发器的，在 8*8 LED 点阵逻辑中需要用到的基本触发器包括 SR 触发器和 LR 触发器，这里我们首先介绍 SR 触发器（如图 2.6）。

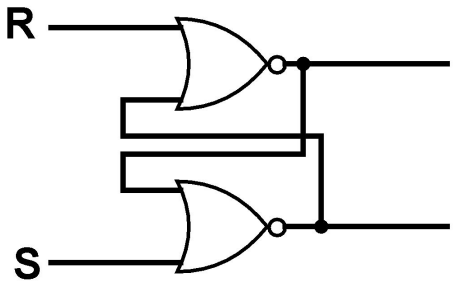


图 2.6 SR 触发器

SR 触发器是一种最基本的触发器， 由图 XXX 可以看到 SR 触发器是由两个与非门耦合成的。与非门是与门和非门相结合的逻辑单元，具有两个输入端和一个输出端，对两个输入端先做与操作再做非操作。SR 触发器中两个与非门的两个输出端作为另一个与非门的输入端输入，两个输出端在正常情况下一个为低电平一个为高电平，当两个输入端 S 和 R 都为低电平时，触发器会进入不稳定的震荡状态。尽管 SR 触发器有两个输出端 Q 和 Q’，但一般规定输出端 Q 输出的状态为触发器的状态。

现在简要地介绍一下 SR 触发器的工作原理。假设 SR 触发器原来的状态为 Q=0，我们在 R 端加上一个低电平，S 端仍保持高电平不变，触发器输出端将会出现 Q=1，此时触发器的状态由 0 变成了 1。当 R 端的高电平消失之后，由于 Q 端的输出被直接耦合到了下面那个与非门的输入端，Q 端可以保持输出为 1，此时触发器处于稳定状态。这样 SR 触发器就成为了一个拥有记忆存储功能的单元器件，可以存储上一次读入的数据。

尽管 SR 触发器能够存储数据，然而在 8*8 LED 点阵中，我们需要用时序逻辑电路来控制点阵的驱动芯片，因此需要在 SR 触发器上加上一个时钟控制信号（如图 2.7）。

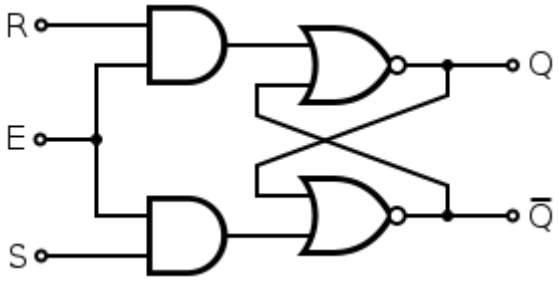


图 2.7 加上时钟控制信号 E 的 SR 触发器

此时新加入了一个信号 E，当 E 端的脉冲到达触发器时，触发器的输入端信号才能接受 R 端和 S 端的输入信息，当 E 端为低电平时，数据被存储在触发器中。

在这一步的基础之上，在 R 端前面增加一个非门（如图 2.8），这样只需要一个输入端就能对触发器进行输入，并且保证 S 端和 R 端的输入数据不会相同，即触发器不会发生震荡现象。此时的触发器就是我们在 8*8 LED 点阵中要使用到的 D 触发器。

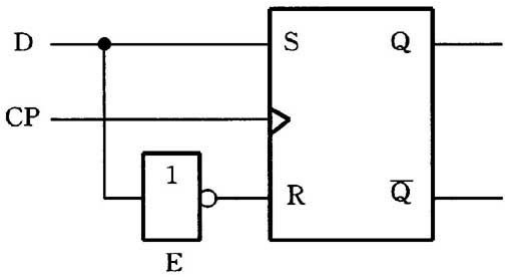


图 2.8 加上与非门的 SR 触发器

②74HC595 芯片

74HC595 芯片是一个具有三层逻辑结构的驱动芯片，其中第一层为由 8 个逻辑 D 触发器组成的移位层，第二层为由 8 个锁存 D 触发器组成的锁存层，第三层为由八个三态门组成的控制层，其电路图如如图 2.9 所示：

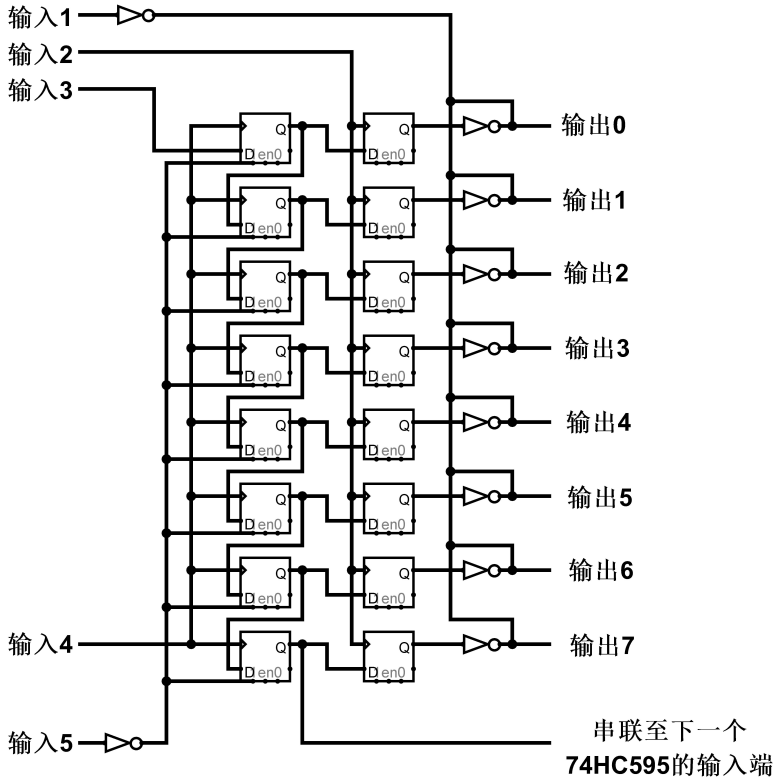


图 2.9 74HC595 芯片电路图

从图中可以看出，74HC595 芯片具有五个输入口和八个输出口，其中输入口 1 是输出使能端，在 8*8 LED 点阵中默认为 0。输入口 2 是第二层锁存 D 触发器的时钟信号，输入口 3 是串行数据输入端，通过这个输入口可以输入一个八位的比特流，74HC595 会将比特流转变成八个一位信号输出。输入口 4 是第一层移位 D 触发器的时钟信号，输入口 5 是第一层移位 D 触发器的复位信号，当输入为 0 时八个移位 D 触发器将被清零。

下面介绍一下 74HC595 的工作过程，假设要从输入口 3 输入一个八位数据，经过 74HC595 后从八个输出口同时输出每一位，74HC595 的工作流程用文字描述可以表示为：

A. 将输入口 2，即第二层的锁存 D 触发器时钟信号置为低电平。

B. 将输入口 4，即第一层移位 D 触发器的时钟信号置为低电平。

C. 设置一个计数器 count，初始化为 7。

D. 获取需要输出的信号的第 count 位，将其发送到输入口 3，即串行数据输入端。

E. 将输入口 4，即第一层移位 D 触发器的时钟信号拉高，并将 count 减 1。

F. 如果 count>=0，回到步骤 3，否则到步骤⑦。

G. 将输入口 2，即第二层的锁存 D 触发器时钟信号拉高，此时八位数据将从八个输出口传出。

③8*8 LED 点阵模块电路

一个 8*8 LED 点阵模块由两个 74HC595 芯片级联组成，两个芯片的八个输出口分别接 8*8 LED 点阵的横排输入口和纵排输入口。电路图如图 2.10 所示。

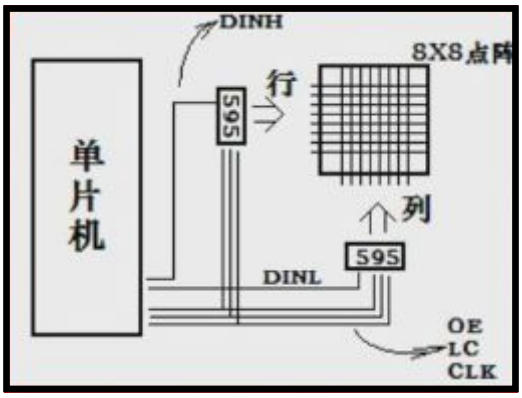


图 2.10 8*8 LED 点阵电路图

注意到图上两片 74HC595 芯片的锁存器时钟信号是连在一起的，所以编写程序的时候只需要给这个时钟信号一次脉冲就够了。两个 74HC595 芯片分别控制 8*8 点阵的行和列，所以在编程的时候如果需要在点阵上显示图像，需要轮流给行和列赋值。

现在假设我们需要在点阵屏幕上一直显示一个图像，需要编写的驱动程序伪代码如代码段 2.1：

```
WHILE(1){--------------------------------不断交替显示每一行,视觉上就是一
    INT I = 8;                           幅图
    WHILE( I-- ){---------------------------针对每一行进行显示
        DOWN( CLK_LATCH );-----------------锁存器时钟置为低电平
        SHIFTOUT( 74HC595_1 ); ------------- 用 74HC595 进行行 1-8 译码
        SHIFTOUT( 74HC595_2 ); -------------用 74HC595 进行列 1-8 译码
        UP( CLK_LATCH );}}-----------------拉高锁存器时钟
```

代码段 2.1 8*8 LED 点阵驱动

其中，SHIFTOUT 函数的伪代码如代码段 2.2：

```
SHIFTOUT( DIGIT ) {-----------------------对单个 74HC595 进行移位
    DOWN(CLK_SHIFT); ---------------------移位时钟置为低电平
```

```
INT J = 8;
WHILE( J-- ) {-------------------------一共要移位八次
    WRITE( DIGITPIN, DIGIT 第 I 位 ); -----将对应位写入输入引脚
    UP(CLK_SHIFT); }}-----------------拉高移位时钟
```

代码段 2.2 8*8 LED 点阵 SHIFTOUT 函数

当想要更新点阵上的图案时，我们只需要为每个图案设置一个循环时间，循环时间结束之后切换到下一个图案的循环时间即可。

最后，附上 8*8 LED 点阵的引脚功能表（如表 2.2），由于除了电源和地之外只有两个时钟引脚和一个控制引脚，所以相对简单，这里不再赘述。

表 2.2 8*8 LED 点阵引脚功能表

名称	功能
VCC	接电源
HND	接地
SER	串行输入数据
RCK	锁存 D 触发器时钟
SRCK	移位 D 触发器时钟

3. MINI12864LCD 液晶显示屏

（1）初探 MINI12864LCD——与 ZYBO 板连接的困难

MINI12864 LCD 模块是一款基于 12864 液晶显示器开发的显示模块。这款模块 SPI 为接口的显示模块，配合 12864LCD 库文件，便可轻松显示汉字，字符和图形。并有背光 LED 控制，可使显示效果更美观。

MINI12864 LCD 一共有八个外接引脚，其中每个引脚的功能如表 2.3：

表 2.3 MINI12864 LCD 引脚功能表

标识符号	名称	功能	
R	RESET	低电平复位，复位完成后回到高电平，液晶模块开始工作，如果不需要软件给芯片复位，可以不连接。	
A	A0	数据和命令选择。L：命令。H：数据。	
CS	CS	SPI	片选，低有效
C	SCK		串行时钟
D	MOSI(SID)		数据传输
-	GND	电源地	
+	VCC	DC 3.3V~5.5V	
L	LED	背光 LED 使能，低有效	

然而我们需要面对的问题是：

①怎么将字库或图形库导入 SDK？

②怎么使用 ZYBO 板子上的 Pmod 口代替 Arduino 板子上的 SPI？

首先是第一个问题，我们将字库与图形库导入 SDK 后发生了汇编级别的系

统错误，这个错误目前以我们的能力是很难解决的。因此后来我们上网查找了相关资料，找到了一份按照 ASC 可见字符的顺序排列的取模数组，我们将取模数组导入 SDK，设计了相关算法使 MINI12864 LCD 板在没有引入任何字库和图形库的情况下显示了文字和数字。

接着是第二个问题，由于我们没办法使用 Arduino 板子封装好的驱动程序，必须手动编写驱动，因此我们必须非常清楚地了解 MINI12864 LCD 的底层构造。

（2）艰辛的探索过程------MINI12864LCD 的工作原理

①MINI12864LCD 的基本原理

要了解 MINI12864LCD 的基本原理，首先需要了解液晶屏显字的原理。简单的来说，液晶屏的每个点都有对应的一个寄存器空间，就像一个个小格子，只要这个格子里面存了 1，屏幕上对应的点就是黑的，存的是 0，这个点就是白的，所以想要点亮 LCD，我们要做的就是在对应的寄存器空间里写进 1。

很巧的是，MINI12864LCD 与 8*8 LED 点阵的原理非常相似，这为我们的研究省去了不少麻烦。两者都是将串行的数据输入转换为并行的数据输出，最后同时输入到点阵或寄存器阵列中。

MINI12864LCD 采用 UC1701 控制器进行控制的，而 UC1707 的资料在网上非常的少，并且 MINI12864LCD 的电路图十分复杂，因此我们转换了研究思路，从搞清楚每个引脚输入信号入手，先去研究 Arduino 封装好的 MINI12864LCD 类库和时序逻辑图，根据类库中的函数和时序逻辑图进行一一对应，之后将对我们有用的函数经过一定的转换移植到 ZYBO 板中。

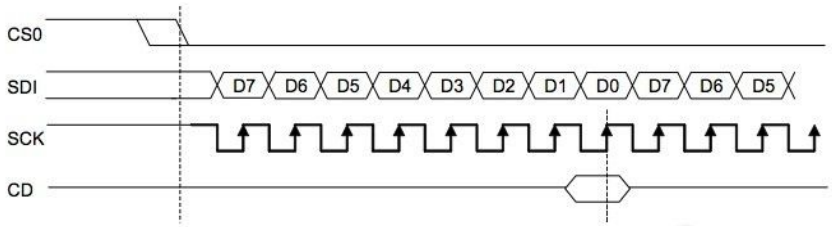


图 2.11 8*8 12864 的时序逻辑图

图 2.11 是 12864 的时序逻辑图，图中两根竖着的虚线之间代表一个通讯周期，在周期最开始的时候先将 CS0 片选信号置为低电平，表示 ZYBO 板准备与 MINI12864 进行通讯。而数据信号和时钟信号与 74HC595 非常相似，当第一个时钟信号上升沿到来时，数据的最高位被传入了数据信号，当第二个时钟信号上升沿到来时，数据的第二位被传入了数据信号，如此循环 8 次，数据的 8 位都被输入到了 MINI12864 中。在最后一个时钟的上升沿，MINI12864 会检查 CD 寄存器选择线，以确定这 8 个数据是发送给自己的执行命令还是要显示的数据。如果是命令那 MINI12864 会执行相应的八位命令，如果是数据则将数据输出到屏幕上。当不需要使用 MINI12864 时，将片选信号置为高电平。

而 MINI12864 在显示的时候与 8*8 LED 点阵稍有不同，MINI12864 显示屏被分成了 8 页（行），和 127 列，在发送地址的时候，页地址以 16 进制进行发送，列地址以 2 个 16 进制分高四位和低四位进行发送。

②自己动手写驱动程序

由于无法使用 Arduino 封装好的驱动程序，我们需要根据 MINI12864 的时序逻辑图自己编写驱动程序。当理解了时序逻辑图之后，编写驱动并不是非常困难。MINI12864 驱动程序的伪代码如代码段 2.3 所示：

```
INT MAIN() {
    INITIALIZE();------------------------------初始化引脚参数和 12864 参数
    WHILE(1){
        …
        SHOWSOMETHING();--------------------显示字母或者数字
        … } }

INITIALIZE() {
    …
    DOWN(CS);--------------------------------选中片选信号
    DOWN(RST);
    UP(RST); ---------------------------------进行复位
    WRITECMD()-------------------------------写入初始化命令
    … }

WRITECMD() {
    DOWN(RS); --------------------------------将 12864LCD 置为写入命令状态
    SHIFTOUT(DIGIT); } ------------------------将 8 位数写入 12864LCD

WRITEDIGIT() {
    DOWN(RS); --------------------------------将 12864LCD 置为写入数据状态
    SHIFTOUT(DIGIT); }------------------------将 8 位数写入 12864LCD

SHOWSOMETHING(SOMETHING) {
    …
    INT I = SOMETHING.LENGTH; ----------------I 为打印数据的长度
    WHILE(I ){
        CALCULATE(SOMETHING);----------------取模算法
        FOR(0 TO 16) {-----------------------  一行一行读入 12864LCD
            …
            WRITEDIGIT（SOMETHING[I]）; --------写入数据
            I--;
            … } } }
```

代码段 2.3 MINI12864 驱动函数

其中 SHIFTOUT 函数和 8*8LCD 板的 SHIFTOUT 函数伪代码基本相同，这里不再赘述。

5. OMRON 外接按钮逻辑

OMRON 外接按钮的逻辑要比上述两个外接设备简单很多。按钮在按下之前，两段的两组引脚（1、2 引脚一组，3、4 引脚一组）是断开的，按下之后就联通了。因此按钮的连接方法（如图 2.12）是一个引脚（1、2 中的任意一个引脚）接电源正极，另一引脚（3、4 中的任意一个引脚）接地，输入口（8 号线）连在与电源正极相连的按钮的引脚（1、2 其中一个）处，然后再串如一个电阻，降低输入口的电压。这样的话，在按钮没有按下时，输入口一直都是高电平，而按钮按下时，相当于联通了地面，因此电被导入到地面，输入口此时就变成了低电平。

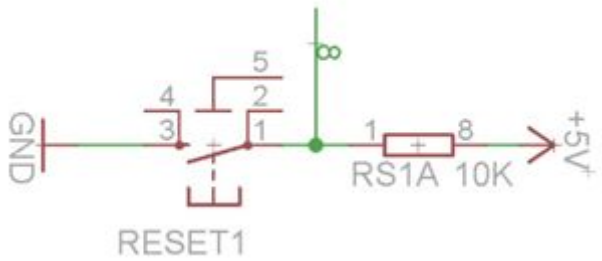


图 2.12 OMRON 按钮连接电路图

了解了 OMRON 外接按钮的逻辑之后，按钮的驱动程序就很好写了。在输入端口接收到低电平的时候，触发按钮按下的事件即可。而在输入端口接收到高电平时，保持程序。同时还要注意的是，按钮需要做去抖动处理，否则紊乱的电平会导致某些函数被多次触发。

（三）软件功能设计

6. 整体

我们一共设计了三种模式，使用外接按钮切换模式。模式 0 是欢迎模式，8*8 的 LED 板所有灯全部亮起，检查有无坏点，然后液晶板显示“Welcome”和我们小组组员的名字缩写。模式 1 是文字模式，8*8 的 LED 板闪烁“I LOVE Y”文字，接着滚动显示“I LOVE YOU”，同时液晶板显示当前文字信息。模式 2 是贪吃蛇游戏模式，通过 ZYBO 内置的四个按钮控制贪吃蛇的上下左右游戏，同时液晶板显示得分、最高得分、和游戏结束等信息。

7. 模式切换伪代码

通过监听外接按钮来切换模式。

```
Int main(void){
    while(1){----------------------------------主程序永远循环
        if (key ==  外接按钮){
            mode++;
            if (mode == 3){--------------------到模式 2 切回模式 1,不回到欢迎模式
                mode = 1;}}
```

```
        if (mode == 0){...}----------------------模式 0 的代码
        if (mode == 1){...}----------------------模式 1 的代码
        if (mode == 2){...}----------------------模式 2 的代码
    return 0;}
```

代码段 2.4 模式切换伪代码

8. 欢迎模式伪代码

欢迎模式（如代码段 2.5）调用液晶板驱动程序，显示欢迎界面，同时不断监听 OMRON 外接按钮。

```
    if (mode == 0){----------------------------模式 0
        while(1){--------------------------------永远循环,除非外接按钮按下
            清空液晶板-----------------------不清空液晶板会出现图案覆盖
            液晶板显示欢迎界面
            key = 按钮去抖动------------------按钮不去抖动会妨碍后续判断
            if (key == 外接按钮){
                break;}}}}
```

代码段 2.5 欢迎模式伪代码

9. 文字模式伪代码

文字模式（如代码段 2.6）在 8*8 LED 点阵屏显示 ILOVEU 的内容，在液晶屏显示文字信息，同时不断监听 OMRON 外接按钮。

```
    if (mode == 1){----------------------------模式 1
        while(1){--------------------------------永远循环,除非外接按钮按下
            清空液晶板-----------------------不清空液晶板会出现图案覆盖
            液晶板显示文字内容
            8*8 LED 灯显示文字----------------函数内部有监听按键按下事件
            if (key == 外接按钮){
                break;}}}}
```

代码段 2.6 文字模式伪代码

8*8 LED 灯显示文字的函数（如代码段 2.7）是利用了之前的 8*8 LED 灯驱动程序，逻辑十分简单，因此只展示一下文字闪烁的函数，移位函数只是在闪烁函数的基础上，对文字矩阵进行了移位计算，然后逐帧显示，逻辑比较简答，就不予罗列了。

```
void flashILOVEU(XGpio *Gpio 输出) {----------------ILOVEU 闪烁
    for ( d=0; d<999;  d++){-------------------如果没有循环函数 LED 灯会一闪而过
        display_once(“I”字木的十六进制数组, Gpio 输出);
        key =按键去抖动 }
    for ( d=0; d<999;  d++){
        display_once(“LOVE”字木的十六进制数组, Gpio 输出);
        key =按键去抖动 }
```

```
for ( d=0; d<999;  d++){
    display_once(“U”字木的十六进制数组, Gpio 输出);
    key =按键去抖动 }}
```

代码段 2.7 flashILOVEU 函数伪代码

10. 贪吃蛇游戏模式伪代码

```
if (mode == 2){----------------------------模式 2
    while(1){-----------------------------永远循环,除非外接按钮按下
        贪吃蛇初始化函数-----------------液晶板的显示和按钮监听都在函数内
        贪吃蛇运行函数                    部
            break;}}}}
```

代码段 2.8 贪吃蛇模式伪代码

贪吃蛇游戏（如代码段 2.8）是思路是利用 C 的二维数组，构建矩阵。矩阵中，0 代表空，1 代表蛇，2 代表食物。对应到 8*8 的 LED 灯就是非 0 的地方全部亮灯。因此只需要对矩阵非零的地方记录一下，利用移位函数，就可以将整个二维数组转换成 16 进制数据组。我们之前自己设计的 8*8 LED 灯的驱动，可以直接处理我们转换而成的 16 进制数组。

对贪吃蛇二维数组操作的函数有许多，比如判断按下的按钮是什么、根据按钮进行移动、移动之后的蛇有没有吃到食物等等，功能比较简单，就不一一罗列了，比较重要的是贪吃蛇的初始化（如代码段 2.9）、和游戏进行（如代码段 2.10）函数。第一次进入游戏之后执行初始化函数，之后就一直执行游戏进行函数，游戏进行函数一旦发现游戏结束，就调用一次游戏初始化函数。

```
void Initial(){-----------------------------------贪吃蛇初始化函数
    初始化蛇身长度、分数、游戏速度等数据
    液晶板显示游戏信息
    初始化蛇的位置
    随机产生食物
    将游戏的 01 矩阵转换成 16 进制数据----------我们自己编写的 8*8 LED 的驱动程序
                                                可以接收 16 进制数据
    while (key ==  空){-------------------------一直循环,直到有按钮按下
        for (d = 0; d < timeloop; d++) {----------如果没有循环函数 LED 灯会一闪而过
            key =  按钮去抖动
            if (key =  空) continue;--------------没有输入按钮的情况下一直停留在游
                                                戏初始化界面
            else return;------------------------有输入任何按钮都退出初始化
        8*8 LED 灯显示游戏初始界面}}
```

代码段 2.9 贪吃蛇初始化伪代码

```
void Show(){--------------------------------------贪吃蛇运行函数
    if (key ==  外接按钮) return;-----------------在初始化函数中的按钮按键如果是外
    while(1) {一直循环直到外接按钮按下              接按钮则直接退出
```

```
判断 ZYBO 内部按钮按
根据按键移动贪吃蛇
if(吃到自己 || 撞墙){--------------------游戏结束
    液晶板显示游戏结束
    if (Score > HighScore){
        HighScore = Score;
        液晶板显示最高分}
    key = 空;
    while (key == 空){
        for (d = 0; d<999; d++){
            key = 按钮去抖动
            if (key == 外接按钮) return;
            else if (key == ZYBO 内置按钮) break;
            8*8 LED 灯显示游戏结束时的蛇图}}
    Initial();---------------------------ZYBO 内置按钮按下后会进入游戏初
}                                       始化函数
将游戏的 01 矩阵转换成 16 进制数据
key = 空;
for (d = 0; d < timeloop; d++){
    key = 按钮去抖动
    if (key == 外接按钮) return;----------其余情况只需等待下一次循环重新判
    8*8 LED 灯显示游戏当前的蛇图}}}        断按键,然后再根据案件进行蛇的移
                                        动
```

代码段 2.10 贪吃蛇游戏进行伪代码

三.实现方法与步骤

（一）硬件电路连接

GAMMER 的实物连接图如图 3.1 所示：

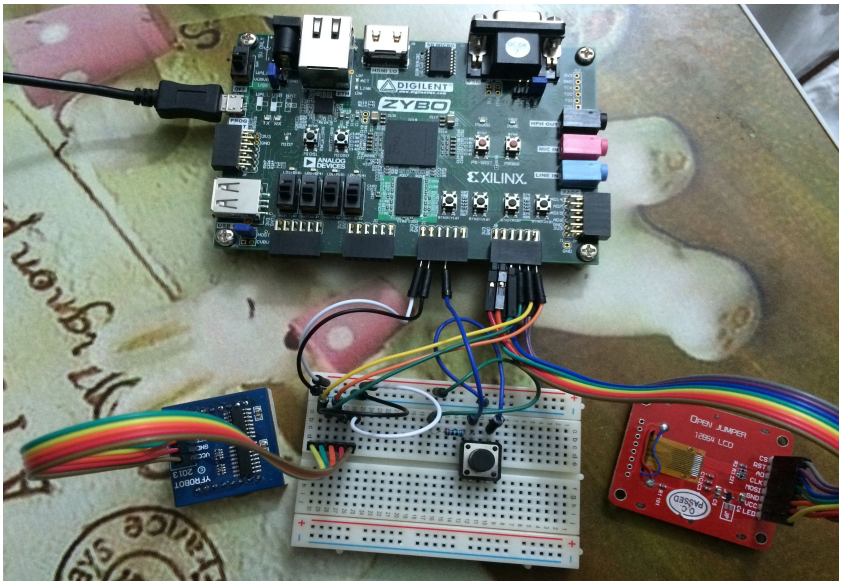


图 3.1 GAMMER 的实物连接图

（二）驱动程序设计

1. 8*8 LED 点阵的驱动程序

8*8 LED 点阵的驱动程序主要由以下两个函数构成，下面一个一个介绍。

首先是 display_once 函数（如代码段 3.1），这个函数的作用是将图像在 LED 点阵上显示一次，当需要持续显示图像的时候，根据需要显示的时间长短把这个函数放在一个 while 循环中即可。

display_once 中的 for 循环将数据一行一行输出，即一行一行将点阵上的灯点亮。在点亮每一行灯时，首先将锁存器时钟置为低电平，然后进行两个 74HC595 芯片的移位，第一个芯片控制行，第二个芯片控制列，最后将锁存器时钟拉高，使得两个芯片同时输出到点镇上，控制一行灯的亮灭。

```
void display_once(u32 mypicture[], XGpio *nn){
    int j;
    for (j = 0; j< 8; j++) {//输出每一行
        XGpio_DiscreteWrite(nn, 1, 0xfffffffd);//将锁存器时钟置为低电平
        laststate = 0xfffffffd;//保存 GPIO 状态
        shiftOut(nn,3,2,~(1<<j));//对第一个 74HC595 进行移位操作
        shiftOut(nn,3,2,mypicture[j]); //对第二个 74HC595 进行移位操作
        int n = 1<<1;
        XGpio_DiscreteWrite(nn, 1, n|laststate);//将锁存器时钟置为高电平
        laststate = n|laststate; }}//保存 GPIO 状态
```

代码段 3.1 display_once 函数

接下来介绍 shiftOut 函数（如代码段 3.2），这个函数的作用是将一个八位数据移到八个移位 D 触发器中。首先将移位时钟和锁存时钟置为低电平，之后进行 for 循环，一共循环 8 次将 8 位数据依次存入寄存器中。在每次循环中，取出数据的最高位，将其移入第一个寄存器，然后将寄存器时钟置为高电平进行移位。

```
void shiftOut(XGpio *mm, int  a,int  b, u32 c) {
    XGpio_DiscreteWrite(mm, 1, laststate & ZhiLinga);//将锁存器时钟置为低电平
    laststate = laststate & ZhiLinga;
    XGpio_DiscreteWrite(mm, 1, laststate & ZhiLingb);//将寄存器时钟置为低电平
    laststate = laststate & ZhiLingb;
    for( i=7; i>=0; i--) {//进行移位
        XGpio_DiscreteWrite(mm, 1, laststate & ZhiLingb);//将寄存器时钟置为低电平
        laststate = laststate & ZhiLingb;//保存寄存器状态
        if(c & (1<<i))//将数据的第 i 为保存在 PinState 中
            PinState = 1;
        else
            PinState = 0;
        if (PinState == 1) {//将 PinState 作为一位数据输入 74HC595 芯片
            XGpio_DiscreteWrite(mm, 1, ZhiYia | laststate);
            laststate = ZhiYia | laststate; }
        else {
            XGpio_DiscreteWrite(mm, 1, ZhiLinga & laststate);
            laststate = ZhiLinga & laststate;}
        XGpio_DiscreteWrite(mm, 1, ZhiYib | laststate);//将寄存器时钟置为高电平
        laststate = ZhiYib | laststate;
        XGpio_DiscreteWrite(mm, 1, ZhiLinga & laststate);//将锁存器时钟置为高电平
        laststate = ZhiLinga & laststate; }
    XGpio_DiscreteWrite(mm, 1, ZhiLingb & laststate);//将寄存器时钟置为低电平
    laststate = ZhiLingb & laststate; }
```

代码段 3.2 shiftOut 函数

2. MINI12864 液晶屏的驱动程序

MINI12864 液晶屏的驱动程序与 8*8 LED 点阵的驱动程序相比较为复杂，但是大同小异。分为以下几个函数，现在一个一个介绍。

首先是 send_8bit_mini12864 函数（如代码段 3.3），这个函数的作用与 8*8 LED 点阵的 shiftOut 函数基本相同，都是将一个八位的数据串行输入，转换为八个一位的数据并行输出。

```
void send_8bit_mini12864(unsigned char d, XGpio *b, XGpio *s) {//传 8 位数据,高位先传
    int i;
    for(i=0;i<8;i++) {//循环 8 次,每次发送 1 个最高位
        XGpio_DiscreteWrite(s, 1, lastsw&(~1));
        lastsw = lastsw&(~1);
        if(d&0x80) {//与上 10000000,最高位如果是 1,就数据位就高
            XGpio_DiscreteWrite(s, 1, lastsw|(1<<4));
            lastsw = lastsw|(1<<4);}
        else {//与上 10000000,最高位如果是 0,就数据位就低
            XGpio_DiscreteWrite(s, 1, lastsw&(~(1<<4)));
```

```
            lastsw = lastsw&(~(1<<4));}
        XGpio_DiscreteWrite(s, 1, lastsw|1); //一位数据发送完毕,时钟拉高
        lastsw = lastsw|1;
        d<<=1; }} //d|ddddddd0 抛掉最高位其余数据左移一位最低位填 0
```

代码段 3.3 send_8bit_mini12864 函数

下面这两个函数分别是对 MINI12864 写入命令（如代码段 3.4）和数据（如代码段 3.5）的函数，逻辑上是相似的，首先将片选拉低选中芯片，之后设置 RS 输入的值判断是数据输入还是命令输入，最后向 MINI12864 写入相应的数据或者命令。

```
void write_cmd_mini12864(unsigned char cmd, XGpio *b, XGpio *s) {//写命令
    XGpio_DiscreteWrite( s, 1, ~(1<<7)&lastsw); //片选拉低,选中芯片
    lastsw = ~(1<<7)&lastsw;
    XGpio_DiscreteWrite( s, 1, ~(1<<5)&lastsw); //rs 为低,写入命令
    lastsw = ~(1<<5)&lastsw;
    send_8bit_mini12864(cmd,b,s); }//写入 8 位命令
```

代码段 3.4 write_cmd_mini12864 函数（对 MINI12864 写入命令）

```
void write_dat_mini12864(unsigned char dat, XGpio *b,XGpio *s) {//写数据
    XGpio_DiscreteWrite( s, 1, ~(1<<7)&lastsw); //片选拉低,选中芯片
    lastsw = ~(1<<7)&lastsw;
    XGpio_DiscreteWrite( s, 1, (1<<5)|lastsw); //rs 为高,写入数据
    lastsw = (1<<5)|lastsw;
    send_8bit_mini12864(dat, b, s); }          //写入 8 位数据
```

代码段 3.5 write_dat_mini12864 函数（对 MINI12864 写入数据）

接下来这个函数是对 MINI12864 进行初始化的函数（如代码段 3.6），对一些必要输入进行初始化，这里不再赘述。

```
void mini12864_Init( XGpio *b, XGpio *s ) {//初始化
    XGpio_DiscreteWrite( s, 1, ~(1<<7)&lastsw);
    lastsw = ~(1<<7)&lastsw;
    XGpio_DiscreteWrite( s, 1, ~(1<<6)&lastsw);
    lastsw = ~(1<<6)&lastsw;
    delay(200);
    XGpio_DiscreteWrite( s, 1, (1<<6)|lastsw);
    lastsw = (1<<6)|lastsw;
    delay(1000);
    write_cmd_mini12864(0xe2, b, s); //软件重置
    delay(200);
    ……
     mini12864_cl(b, s); }//清屏
```

代码段 3.6 mini12864_Init 函数（对 MINI12864 初始化）

接下来这个函数是页列转换函数，这里的页其实就是我们通常所说的行，这个函数被用于在液晶屏上生成不同的文字和图案。

```
void write_add(unsigned char p, unsigned char l, XGpio *b, XGpio *s) { //页列转换
    write_cmd_mini12864(0xb0+p,b, s); //先给页数据 0-7,从上到下,页码是直接读取 8 位数据作为地址
    write_cmd_mini12864(0x10+(8*l/16),b, s); //再是列 0-127,从左到右,列是先读取高四位,后读取低四位
    write_cmd_mini12864(0x00+(8*l%16),b, s); }//低四位
```

代码段 3.7 write_add 函数（在 MINI12864 生成不同的文字和图案）

最后这两个函数分别是在液晶屏上显示相应的字母（如代码段 3.8）和数字（如代码段 3.9）的函数，利用了可见 ASC 码取模表。

```
void show_string(unsigned char p, unsigned char l,char  *string, XGpio *b,XGpio *s) {//在第 p 页的 l 列,显示一个字符串(数组中的元素)
    unsigned int X,i=0;
    write_add(p,l,b, s); //页列转换函数
    int j;
    while(string[i++]) {//判断是否为空
        X=(string[i-1]-32)*16; //将 string 中的元素(ASC 码)匹配到预先写好的 ASC 数组之中
        for(j=0;j<16;j++)
        write_dat_mini12864(ASCII8_16[X++],b, s); } } //发显示数据
```

代码段 3.8 show_string 函数（在 MINI12864 显示文字）

```
void show_number(unsigned char p, unsigned char l,unsigned int d, XGpio *b,XGpio *s) { //在第 p 页的 l 列显示一个字数字
    char ss[8];
    unsigned char m;
    m=0;
    do {//将 d 分离到数组 s 中
        ss[m]=d%10;
        m++;
        d=d/10;
    } while(d);//此时 m 就是 d 的位数
    unsigned int X;
    write_add(p,l,b, s); //页列转换函数
    int z;
    for(z=m;z>0;z--) {//循环 m 次,没就是数字的位数
    X=(ss[z-1]+48-32)*16; //由于此时数组中的并非 ASC 码,而是真真的数字,所以转换时候系数不同
    int j;
    for(j=0;j<16;j++) write_dat_mini12864(ASCII8_16[X++],b, s); } }//发显示数据
```

代码段 3.9 show_number 函数（在 MINI12864 显示数字）

（三）软件编程实现

4. main 函数

main 函数（如代码段 3.10）首先要定义 Gpio 的输入输出口，为后续所有程序使用输入输出口做准备。然后我们在主函数中首先点亮所有 8*8 LED 灯，检查是否有坏点。然后进入欢迎模式，并时刻监听 OMRON 外接按钮，一旦监听到按钮按下，就触发模式转换。

```
int main(void){
    //定义 Gpio 输入输出口
    XGpio_Initialize(&b, XPAR_BTNS_4BIT_DEVICE_ID);
    XGpio_SetDataDirection(&b, 1, 0xffffffff);
    XGpio_Initialize(&s, XPAR_SW_4BIT_DEVICE_ID);
    XGpio_SetDataDirection(&s, 1, 0x00000000);
    mini12864_Init(&b,&s);

    //8*8 板全亮,检查是否有坏点
    for (loop = 0; loop < 9999; loop++) display_once(Check1Date, &s);
    for (loop = 0; loop < 1999; loop++) display_once(Check2Date, &s);

    while(1) {
        if (key == 0xD0) { //模式转换
            mode++;
            if (mode == 4) mode = 1; }

        key = 0xF0;
        if (mode == 0) { //欢迎模式
            while(1) {
                //液晶板显示欢迎界面
                show_string(7,0,"W       L    ",&b,&s);
                show_string(6,0,"E G    HC ",&b,&s);
                show_string(5,0,"L A    FHG",&b,&s);
                show_string(4,0,"CTM B&OA",&b,&s);
                show_string(3,0,"OOM yLSM",&b,&s);
                show_string(2,0,"M E    YEE",&b,&s);
                show_string(1,0,"E R    F    ",&b,&s);
                show_string(0,0,"              ",&b,&s);

                //按钮监听
                psb_check = XGpio_DiscreteRead(&b, 1);
                if (psb_check != 0xF0) {
                    key = XGpio_DiscreteRead(&b, 1);
                    if (key != psb_check) key = 0xF0;
                    if (key == 0xD0) break; } } }
```

```
        else if (mode == 1) { //闪烁 I LOVE U 模式
            while(1) {
                flashILOVEU(&s); //闪烁 I LOVE U 函数

                //液晶板显示 I LOVE U
                show_string(7,0,"   I       ",&b,&s);
                show_string(6,0,"F          ",&b,&s);
                show_string(5,0,"L L        ",&b,&s);
                show_string(4,0,"A O        ",&b,&s);
                show_string(3,0,"S V        ",&b,&s);
                show_string(2,0,"H E        ",&b,&s);
                show_string(1,0,"           ",&b,&s);
                show_string(0,0,"   U       ",&b,&s);

                if (key == 0xD0) break; } }
        else if (mode == 2) { //滚动 I LOVE U 模式
            while(1) {
                moveILOVEU(&s); //滚动 I LOVE U 函数

                //液晶板显示 I LOVE U
                show_string(7,0,"   I       ",&b,&s);
                show_string(6,0,"           ",&b,&s);
                show_string(5,0,"M L        ",&b,&s);
                show_string(4,0,"O O        ",&b,&s);
                show_string(3,0,"V V        ",&b,&s);
                show_string(2,0,"E E        ",&b,&s);
                show_string(1,0,"           ",&b,&s);
                show_string(0,0,"   U       ",&b,&s);

                if (key == 0xD0) break; } }
        else if (mode == 3) { //贪吃蛇游戏模式
            while(1) {
                Initial(); //贪吃蛇初始化函数
                Show(); //贪吃蛇进行函数
                if (key == 0xD0) break; } } }
```

代码段 3.10 main 函数

5. I LOVE U 闪烁函数

flashILOVEU 函数（如代码段 3.11）利用循环使得 8*8 LED 灯持续显示我们想要的文字画面，因为 8*8 LED 灯使用的是时序逻辑，必须持续执行 display_once 函数，否则画面就会一闪而过。与此同时，还要不断监听 OMRON 外接按钮。

```
void flashILOVEU(XGpio *ll){//ILOVEU 闪烁
    for ( d=0; d<999;   d++){
```

```
        display_once(iDate,ll); //8*8 LED 灯驱动函数
        psb_check = XGpio_DiscreteRead(&b, 1);
        if (psb_check != 0xF0){
            key = XGpio_DiscreteRead(&b, 1);
            if (key != psb_check) key = 0xF0;}}
    for ( d=0; d<999;   d++){
        display_once(loveDate,ll);
        psb_check = XGpio_DiscreteRead(&b, 1);
        if (psb_check != 0xF0){
            key = XGpio_DiscreteRead(&b, 1);
            if (key != psb_check) key = 0xF0;}}
    for ( d=0; d<999;   d++){
        display_once(uDate,ll);
        psb_check = XGpio_DiscreteRead(&b, 1);
        if (psb_check != 0xF0){
            key = XGpio_DiscreteRead(&b, 1);
            if (key != psb_check) key = 0xF0;}}}
```

代码段 3.11 flashILOVEU 函数

6. 贪吃蛇相关函数

（3）贪吃蛇初始化函数

贪吃蛇初始化函数（如代码段 3.12）需要将游戏数据初始化，然后在 8*8 LED 板生成游戏初始图案，，还要在液晶板上显示游戏的初始信息。与此同时，还需要不断监听 OMRON 外接按钮。

```
void Initial(){ //地图的初始化
    //初始化数据
    int i, j;
    sum = 2;
    Score = 0;
    over = 0;
    timeloop = 1999;
    memset(GameMap, 0, sizeof(GameMap));   //初始化地图全部为空'0'

    //初始液晶板
    show_string(7,0,"         SH",&b,&s);
    show_string(6,0," S       CI",&b,&s);
    show_string(5,0,"GN       OG",&b,&s);
    show_string(4,0,"AA       RH",&b,&s);
    show_string(3,0,"MK       E ",&b,&s);
    show_string(2,0,"EE       ::",&b,&s);
    show_string(1,0,"            ",&b,&s);
    show_string(0,0,"            ",&b,&s);
```

```
    show_number(0,12,Score,&b,&s);
    show_number(0,14,HighScore,&b,&s);

    //初始化蛇的位置
    GameMap[4][4] = Shead;   GameMap[4][3] = Sbody;
    Snake[0].x = 4;   Snake[0].y = 4;
    Snake[1].x = 4;   Snake[1].y = 3;
    Snake[0].now = -1;

    //随机产生食物
    Create_Food();

    //二维数组转换成 16 进制数组
    memset(GameMapBit, 0, sizeof(GameMapBit));
    for(i = 0; i < 8; i++)
        for(j = 0; j < 8; j++)
            if (GameMap[i][j] != 0) GameMapBit[i] += 1<<j;

    //监听按钮按下
    key = 0xF0;
    while (key == 0xF0) {
        for (d = 0; d < timeloop; d++) {
            psb_check = XGpio_DiscreteRead(&b, 1);
            if (psb_check != 0xF0) {
                key = XGpio_DiscreteRead(&b, 1);
                if (key != psb_check) {
                    key = 0xF0;
                    continue; }
                else return; }

            //8*8 LED 灯显示游戏初始蛇图
            display_once(GameMapBit,&s); } } }
```

代码段 3.12 Initial 函数（贪吃蛇初始化函数）

（4）贪吃蛇运行函数

贪吃蛇初运行函数（如代码段 3.13）需要监听 ZYBO 板内置的按钮的输入，并让游戏蛇作出相应的移动，然后在 8*8 LED 板显示游戏图案，还要在液晶板上更新游戏的信息。与此同时，还需要不断监听 OMRON 外接按钮。如果蛇吃到了自己或者碰到墙壁，则要在 8*8 LED 板显示游戏结束的画面，还要再液晶板显示游戏结束的信息，并再次调用贪吃蛇初始化函数。

```
void Show()   //贪吃蛇进行函数
{
    int i, j; //临时变量
```

```
    if (key == 0xD0) return;
    while(1){
        Button(); //先判断按键再移动
        Move();   //移动

        if(over){   //自吃或碰墙即游戏结束
            //液晶板显示更新
            show_string(7,0,"            ",&b,&s);
            show_string(6,0,"            ",&b,&s);
            show_string(5,0," GO         ",&b,&s);
            show_string(4,0," AV         ",&b,&s);
            show_string(3,0," ME         ",&b,&s);
            show_string(2,0," ER         ",&b,&s);
            show_string(1,0,"   !        ",&b,&s);
            show_string(0,0,"            ",&b,&s);

            //本次游戏分数高于最高分
            if (Score > HighScore){
                HighScore = Score;
                show_string(5,8,"NH",&b,&s);
                show_string(4,8,"EI",&b,&s);
                show_string(3,8,"WG",&b,&s);
                show_string(2,8," H",&b,&s);
                show_string(1,8," :",&b,&s);
                show_number(0,10,HighScore,&b,&s);}

            //按钮监听
            key = 0xF0;
            while (key == 0xF0) {
                for (d = 0; d<999; d++) {
                    psb_check = XGpio_DiscreteRead(&b, 1);
                    if (psb_check != 0xF0) {
                        key = XGpio_DiscreteRead(&b, 1);
                        if (key != psb_check) key = 0xF0;
                        else if (key == 0xD0) return;
                        else break; }
                    display_once(GameMapBit,&s);}}//8*8 LED 灯驱动函数
            for(d = 0; d < 1999; d++); //延时一段时间,否则按钮的值会立马传入,导致游戏迅速重新开始
            Initial();}

        //二维数组转换成 16 进制数组
        memset(GameMapBit, 0, sizeof(GameMapBit));
        for(i = 0; i < 8; i++)
```

```
        for(j = 0; j < 8; j++)
            if (GameMap[i][j] != 0) GameMapBit[i] += 1<<j;

    //按钮监听
    key = 0xF0;
    for (d = 0; d < timeloop; d++) {
        psb_check = XGpio_DiscreteRead(&b, 1);
        if (psb_check != 0xF0) {
            key = XGpio_DiscreteRead(&b, 1);
            if (key != psb_check) key = 0xF0;
            else if (key == 0xD0) return; }
        display_once(GameMapBit,&s);}}}//8*8 LED 灯驱动函数
```

代码段 3.13 show 函数（贪吃蛇运行函数）

六.个人贡献

（一）技术贡献

1. 硬件连接

（1）使用 VIVADO 软件生成项目的约束文件：为 ZYNQ 添加两个 GPIO 模块，设置每个 GPIO 引脚的电压、对应关系和输入输出状态。

（2）连线：将 8*8LED 点阵和 MINI12864 液晶屏连接到正确的 Pmod 口上。

2. 软件编程

（1）8*8LED 点阵驱动程序：通过查找资料，彻底理解了 8*8LED 点阵的内部电路图和时序逻辑图，并在此基础上手动编写了 8*8LED 点阵的驱动程序。编写过程中虽然参考了 PDF 上的示例代码，但是对代码进行深入理解之后我改进了代码逻辑，并将其移植到 ZYNQ 平台。在液晶屏上显示出笑脸之后将封装好的驱动函数接口交给小伙伴，让小伙伴实现逻辑层面代码。

（2）MINI12864 液晶屏驱动程序：由于无法使用支持 Arduino 的 U8glib 图形库，因此所有官方文档和书上的例程都无法参考。因此我查阅了大量 12864 液晶屏的资料，查找到了 12864 液晶屏的时序逻辑，根据时序逻辑图和网上查到的 ASCII 取模数组手动编写了 MINI12864 的驱动程序，程序中没有包含任何封装好的类库。在液晶屏上显示出 HELLO WORLD 之后将封装好的驱动函数接口交给小伙伴，让小伙伴实现逻辑层面代码。

（3）“I LOVE U”闪动和移动效果：编写代码，让 8*8 点阵出现”I LOVE U”的闪动和移动效果。

（二）文档贡献

1. 项目背景部分

2. 技术原理章节的实现逻辑与电路图部分

3. 实现方法与步骤章节的硬件连接与驱动程序设计部分

七.总结

尽管这个项目做了只有不到一个星期的时间，但是整个过程对我而言意义非常重大。由于在刚进入大学的时候是通过调剂来到的软件学院嵌入式系，对嵌入式实在没有什么了解， 更别提好感了。两年多的时间嵌入式类的课程也就是听听课，做做作业，最后期末考完知识也差不多忘干净了。这个项目是我的第一个真正意义上的嵌入式项目，因为之前的所有项目都是有步骤非常详细的教程指导的，所以就算做出来了也并不是非常了解，但是这个项目做完之后，我对整个项目过程乃至非常细微的细节都能够非常了解，对嵌入式也有了比以前更加深刻的理解，完成项目的过程更是充满了欢乐，就像在细细雕琢一个作品一样。

在做项目的过程中，我发现了自己的潜能——能够通过查阅大量资料在没有教程的情况下编写出驱动程序。硬件、驱动这些之前对我而言完全陌生的词，现在都变得非常友好。我们的项目尽管只实现了比较简单的功能，对我而言却是一个全新的开始，我第一次发现嵌入式竟然这么有趣。

再说说我们的 GAMMER，这个小小的游戏机承载了我们很多努力和创新。在最开始的时候，我们的计划是按照教程做一个乒乓球游戏，因为这个游戏有现成的代码，能够大大减轻我们的工作量。然而后来我们不满足于此了，希望能自己有所创新，因此在理解完示例代码之后，我们开始了贪吃蛇的编写工作。在做出最后成果的一瞬间，我们都非常地开心，因为这是真正靠自己动手做出来的东西，从这一刻起，我们不再需要依靠傻瓜式教程了。

总地来说，这是一次非常有意义的项目实践，不仅让我增长了很多有关嵌入式的知识，发掘了自己的潜能，和小伙伴完美合作，更是让我发现了嵌入式的魅力。