同济大学
TONGJI UNIVERSITY

操作系统移植与应用程序开发——

计算机系统实验申优答辩

2251206 冯羽芯

2025.06.20

实验概述

实验任务总述

在上次实现的89 条MIPS CPU 的基础上增加以下内容

> 增加总线与控制器

- 增加Wishbone 总线
- 增加GPIO
- 增加UART
- 增加Flash 控制器
- 增加SDRAM 控制器
- 实现完整SOPC

> 操作系统移植

- Ubuntu 上建立交叉编译环境
- 对μC/OS-II 系统进行改写、编译

> 应用程序开发

- 编写用户程序
- 通过串口进行交互

同济大学
TONGJI UNIVERSITY

PART1

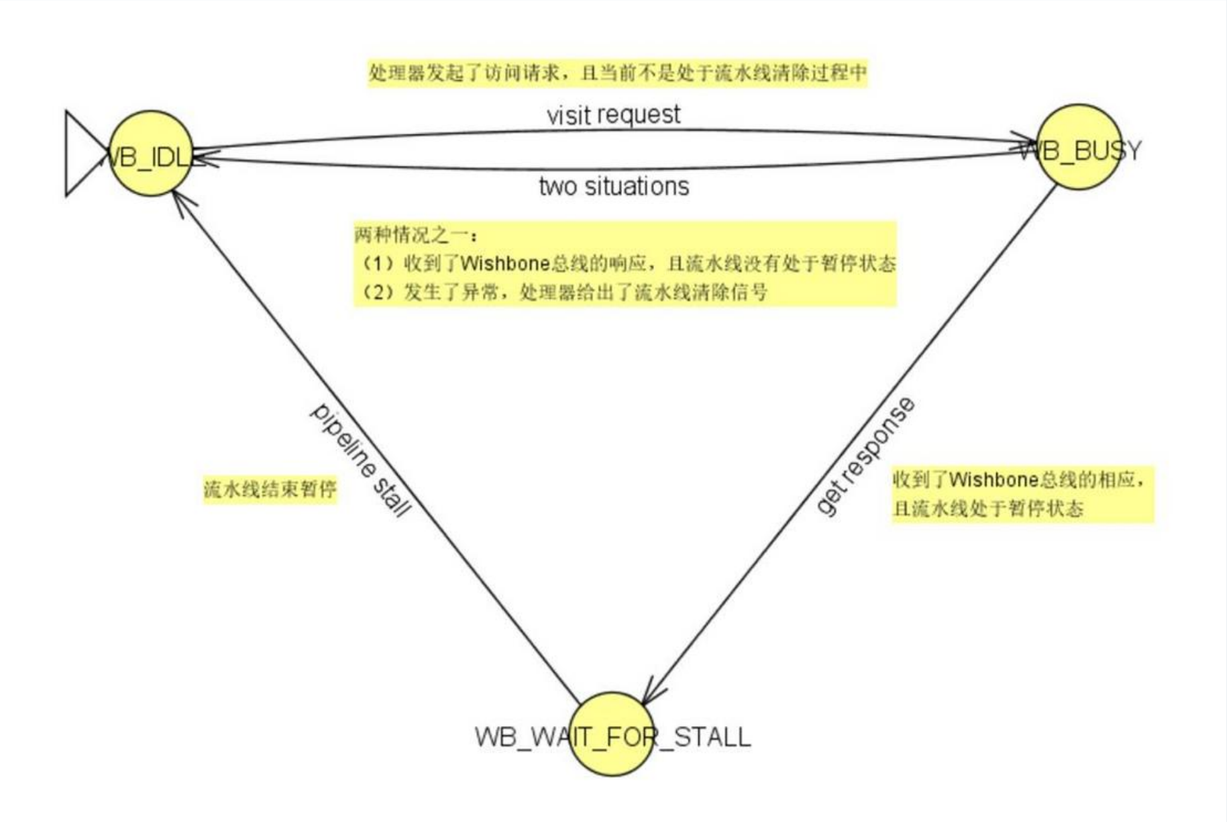
实验2——操作系统移植

Experiment 2 - Operating System Porting

总线接口、时序与各类控制器

WishBone 总线接口

总线状态机



> 状态与转移

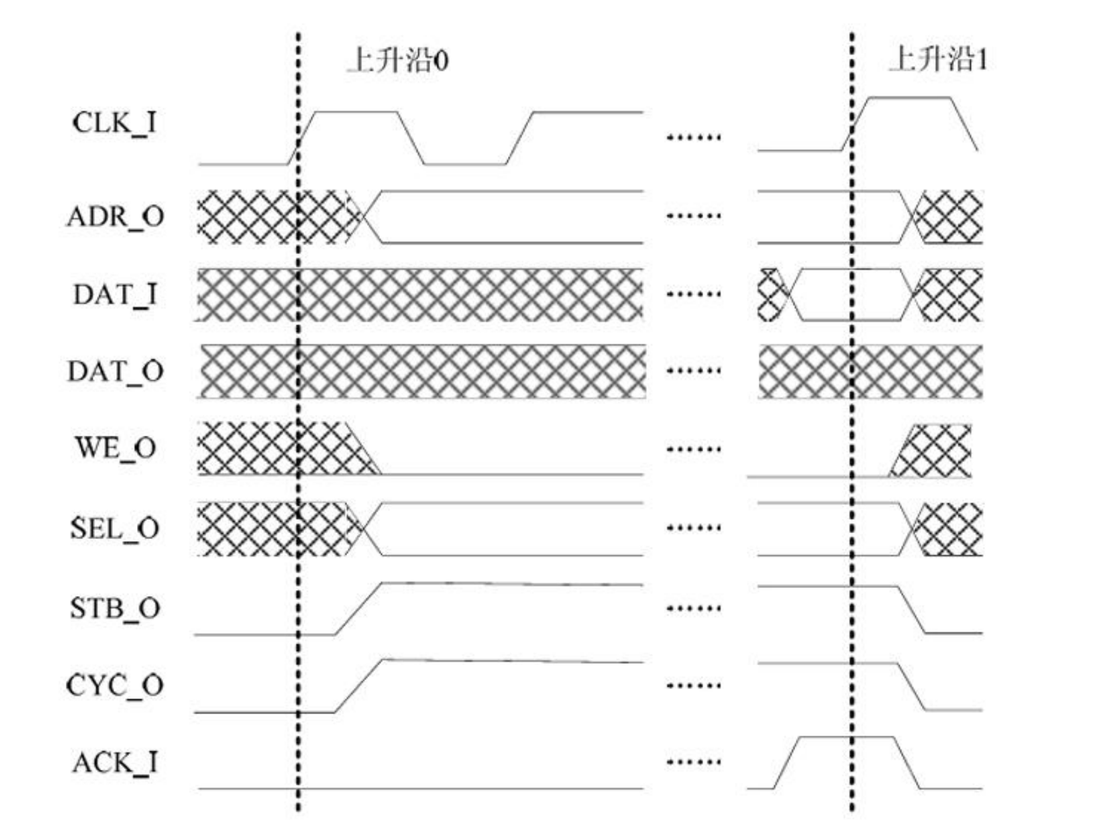
空闲状态 `WB_IDLE`
总线忙状态 `WB_BUSY`
等待暂停结束状态 `WB_WAIT_FOR_STALL`

> 时序与组合

时序电路：控制状态转化
组合电路：给处理器接口信号赋值

总线读写时序

单次读



主设备

> 上升沿 0

ADR_O、SEL_O 放置
WE_O 置低
CYC_O、STB_O 置高

> 上升沿 1

DAT_I 采样
CYC_O、STB_O 置低

从设备

> 中间

DAT_I 放置

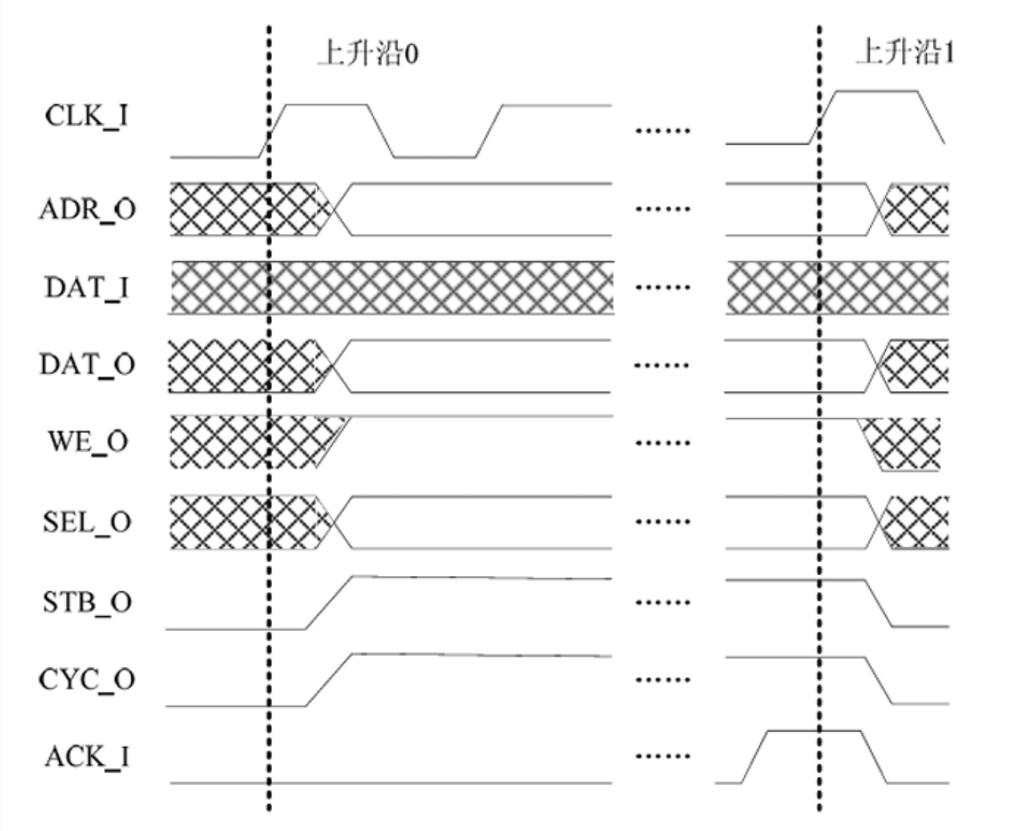
ACK_I 置高

> 中间

ACK_I 置低

总线读写时序

单次写



主设备

> 上升沿 0

ADR_O、DAT_O、SEL_O 放置
WE_O 置高
CYC_O、STB_O 置高

> 上升沿 1

CYC_O、STB_O 置低

从设备

> 中间

DAT_O 锁存

ACK_I 置高

> 中间

ACK_I 置低

GPIO 控制器——从设备接口2

地址：0x20000000-0x2FFFFFFF

功能描述

> I/O

开关与按钮　　七段数码管

通用输入/输出：**输入**时从外部读取输入信号，**输出**时将写入的值输出到外部

> 寄存器说明

寄存器名称	地址	宽度	访问方式	作用描述
RGPIO_IN	Base + 0x0	1~32	只读	输入到 GPIO 的信号
RGPIO_OUT	Base + 0x4	1~32	可读可写	GPIO 输出的信号
RGPIO_OE	Base + 0x8	1~32	可读可写	GPIO 输出接口使能信号
RGPIO_INTE	Base + 0xC	1~32	可读可写	中断使能信号

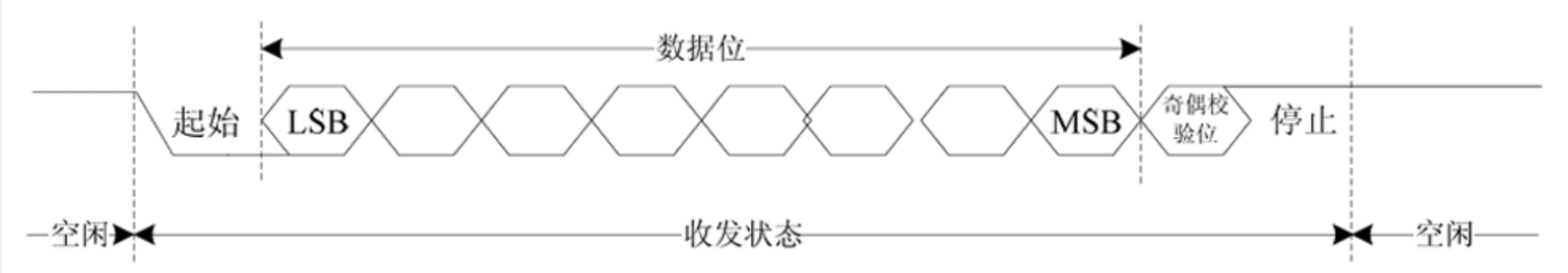
UART 控制器——从设备接口1

地址：0x10000000-0x1FFFFFFF

功能描述

起始位、数据位、奇偶校验位、停止位

波特率 9600 baud



> 数据传输

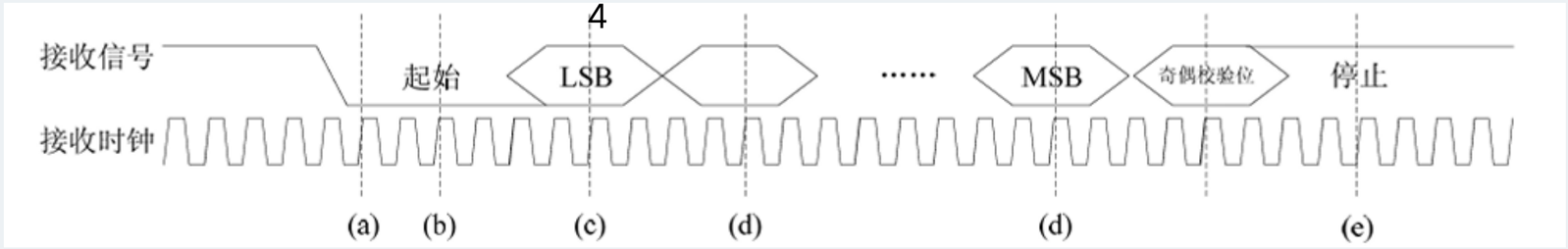
> 数据接收

比波特率高 16 倍的接收时钟

2 4 4 4

分频系数寄存器

分频系数 = 系统时钟 /（16倍的波特率）

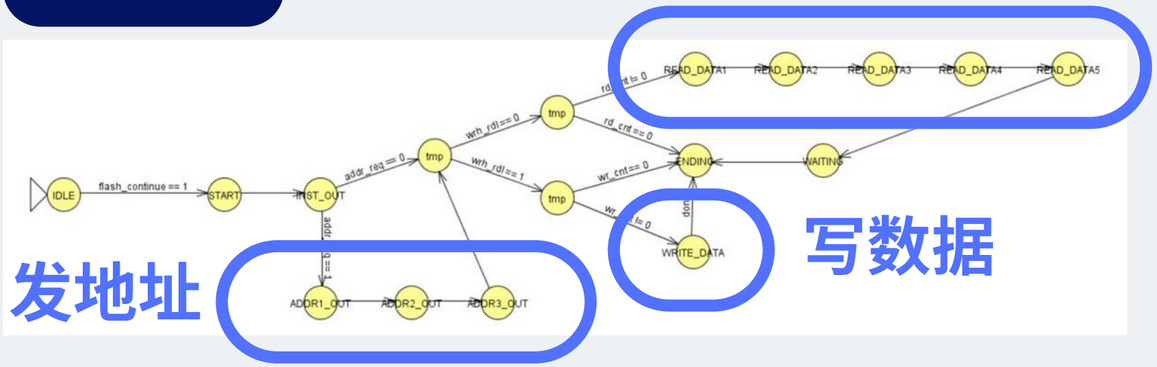


Flash 控制器——从设备接口3

地址：0x30000000-0x3FFFFFFF

功能描述

> 状态机



读数据

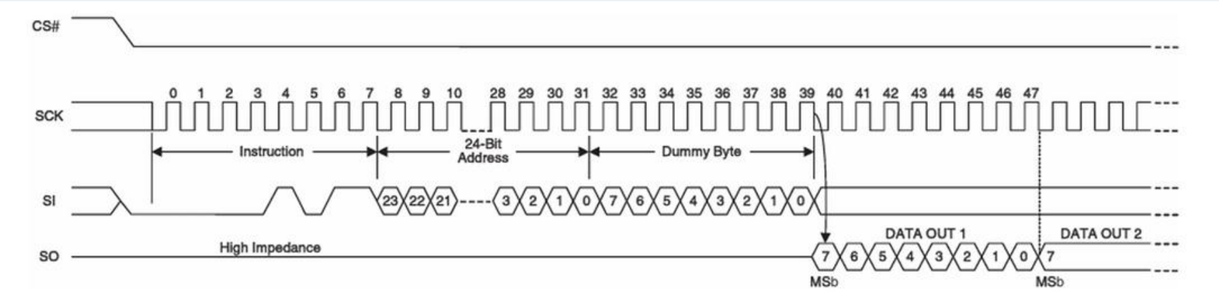
写数据

发地址

若 addr_req（地址请求）标志为高，
表示操作需要地址，则状态转换到 ADDR1_OUT

> 时序

SPI协议



先将 cs_n 拉低，再保持 sck 在工作范围内
然后通过si信号线一位一位的输入指令 READ(03H)，读取地址（24位），最后Flash芯片就会通过 so 信号线一位一位的输出数据

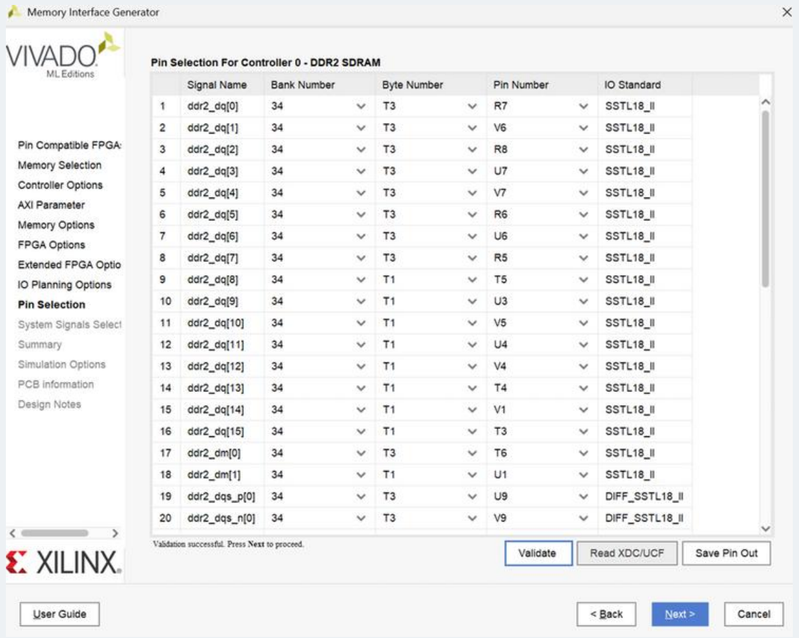
SDRAM 控制器——从设备接口0

地址：0x00000000-0x0FFFFFFF

功能描述

> IP核

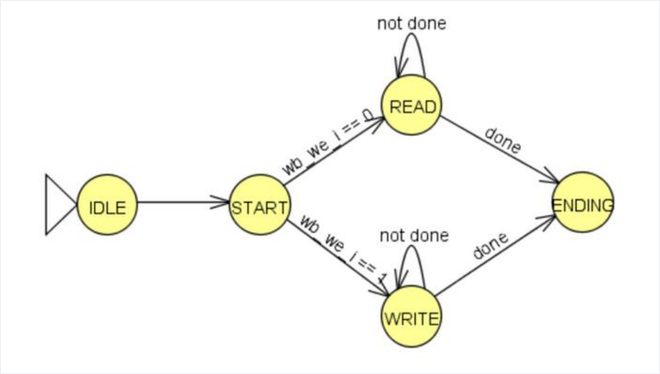
使用的是Nexys4DDR板载的DDR2模块，调用IP核 Memory Interface Generator(MIG)



使用 [Nexys4DDR_DMA_controller/PROJECT/Nexys4DDRmemorypinout.ucf at master · HackLinux/Nexys4DDR_DMA_controller](#) 中下载的文件配置管脚

使用Nexys4官方的Ram2Ddr的VHDL库文件

> 状态机与时序

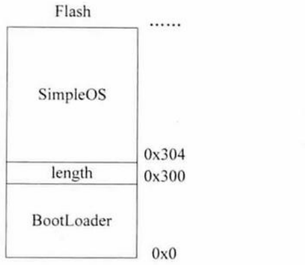


空闲状态 IDLE
开始状态 START
写状态 WRITE
读状态 READ
结束状态 ENDING

同济大学 TONGJI UNIVERSITY

操作系统移植

交叉编译



编写BootLoader

将操作系统读取到 SDRAM

将控制权交给操作系统

1

2

构建文件结构

交叉编译 make all

得到最后的OS.bin

3

Windows配置WSL

构建Linux环境

建立MIPS编译环境

安装GNU工具链

```
chenovo@LAPTOP-MGRUHPKO:/opt$ mips-sde-elf-
mips-sde-elf-addr2line  mips-sde-elf-cpp
mips-sde-elf-ar         mips-sde-elf-g++
mips-sde-elf-as         mips-sde-elf-gcc
mips-sde-elf-c++        mips-sde-elf-gcc-4.3.2
mips-sde-elf-c++filt    mips-sde-elf-gcov
mips-sde-elf-conv       mips-sde-elf-gdb
```

同济大学
TONGJI UNIVERSITY

PART2

实验3——应用程序开发

Experiment 3 - Application Development

2048小游戏逻辑实现

游戏逻辑

数据结构

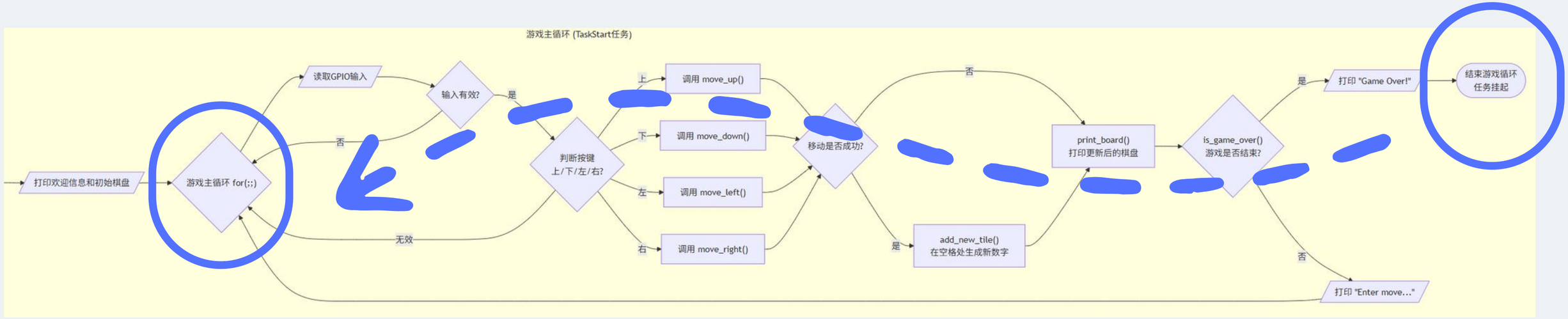
```
#define BOARD_SIZE 4
int board[BOARD_SIZE][BOARD_SIZE];
```

全局变量

直接映射 4x4 的游戏棋盘

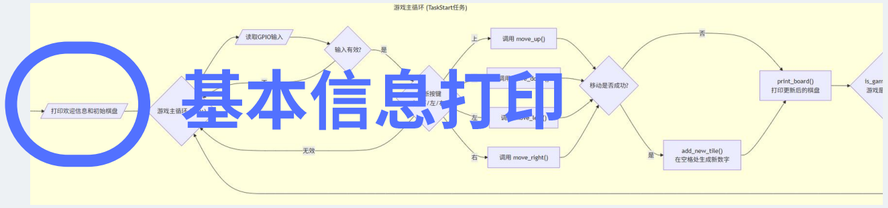
每个位置的整数值代表了棋盘上对应格子的数字，0代表该格子为空

游戏循环



输出

打印与初始化



< 效果图

```
Welcome to 2048!
|--------|--------|--------|--------|
|        |        |        |        |
|--------|--------|--------|--------|
|        |        |        |    4   |
|--------|--------|--------|--------|
|        |        |    2   |        |
|--------|--------|--------|--------|
|        |        |        |        |
|--------|--------|--------|--------|
|        |        |        |        |
|--------|--------|--------|--------|
Use switches to move left, right, up,
```

> 代码

```
void print_board(void) {
    int i, j, k, num;
    char buf[8]; // 足够存储一个 5 位整数 + 空格 + 字符串结束符

    for (i = 0; i < BOARD_SIZE; i++) {
        // 输出上边框
        uart_print_str("|--------|--------|--------|--------|\n");
        // 输出数字和左右边框
        uart_print_str("|");
        for (j = 0; j < BOARD_SIZE; j++) {
            [illegible]
            void add_new_tile(void) {
                int i, j;
                int empty_tiles = 0;
                for (i = 0; i < BOARD_SIZE; i++) {
                    for (j = 0; j < BOARD_SIZE; j++) {
                        if (board[i][j] == 0) {
                            empty_tiles++;
                        }
                    }
                }
                if (empty_tiles == 0) return;

                int pos = my_rand() % empty_tiles;
                int count = 0;
                for (i = 0; i < BOARD_SIZE; i++) {
                    for (j = 0; j < BOARD_SIZE; j++) {
                        if (board[i][j] == 0) {
                            if (count == pos) {
                                board[i][j] = (my_rand() % 10 == 0) ? 4 : 2;
                                return;
                            }
                            count++;
                        }
                    }
                }
            }
        }
        uart_print_str("\n");
    }
    uart_print_str("|--------|--------|--------|--------|\n");
}
```

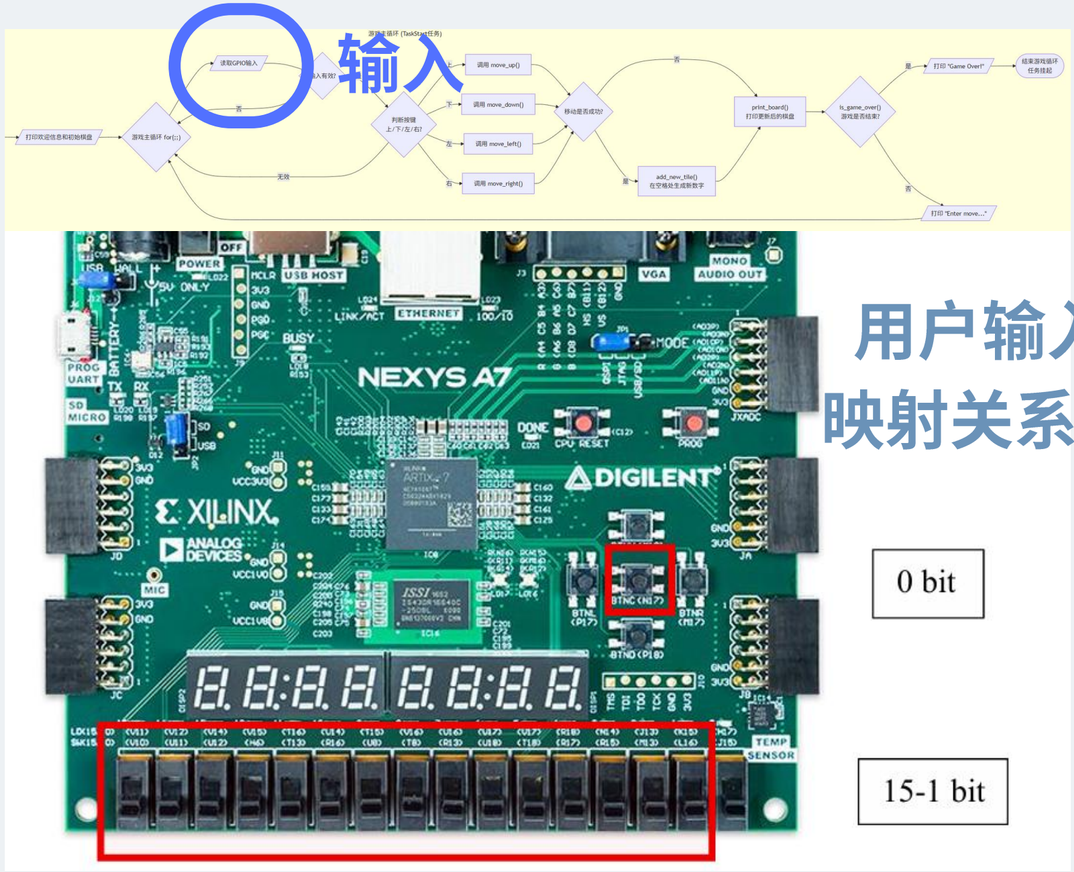
在随机位置
放置两个2或4的初始数字块

如果单元格为空，直接将 buf 填充为5个空格；

如果单元格中含有数字，则从右到左填充buf数组，从而实现右对齐。

用户输入

gpio_in()



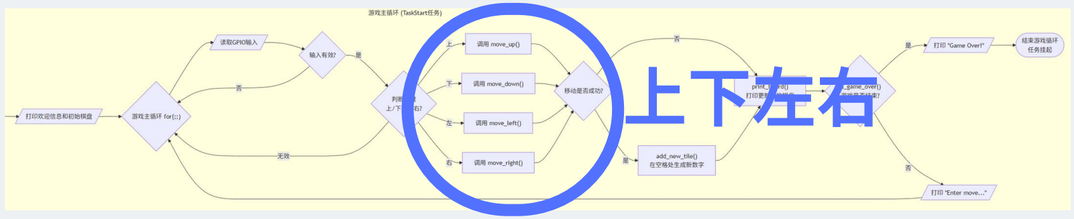
用户的输入其实存储在开发板中的**寄存器**中，我们需要从中实时读取内容即可获取输入

将这组开关状态作为一个**整数**返回，每一位都对应一个 GPIO 引脚的当前电平状态

```
INT32U gpio_in()                    /* 读取 GPIO 模块输入的函数 */
{
    INT32U temp = 0;
    temp = REG32(GPIO_BASE + GPIO_IN_REG);
    return temp;
}
```

核心逻辑

移动合并函数



- 遇到边界
- 遇到另一个不同的数字块
- 遇到一个相同的数字块

外层循环逐行处理棋盘；

中层循环从左到右依次“捡起”每一个非零数字块。

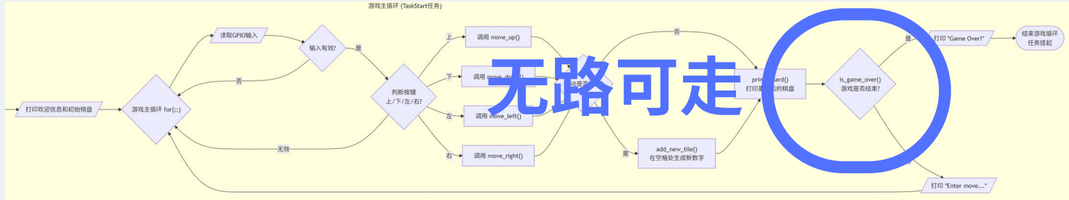
内层循环

- 如果左边是空格，就交换位置
- 如果左边是相同数字，则合并
- 如果左边是不同数字，则被挡住

```c
int move_left(void) {
    int i, j, k;
    int moved = 0;
    for (i = 0; i < BOARD_SIZE; i++) {
        for (j = 1; j < BOARD_SIZE; j++) {
            if (board[i][j] != 0) {
                k = j; // Start from the current position
                while (k > 0) {
                    if (board[i][k - 1] == 0) {
                        // Move to the left
                        board[i][k - 1] = board[i][k];
                        board[i][k] = 0;
                        moved = 1;
                        k--; // Continue moving to the left
                    } else if (board[i][k - 1] == board[i][k]) {
                        // Merge with the left tile
                        board[i][k - 1] *= 2;
                        board[i][k] = 0;
                        moved = 1;
                        break; // Stop moving this tile
                    } else {
                        // Blocked by a different tile
                        break; // Stop moving this tile
                    }
                }
            }
        }
    }
    return moved;
}
```

核心逻辑

游戏结束判断



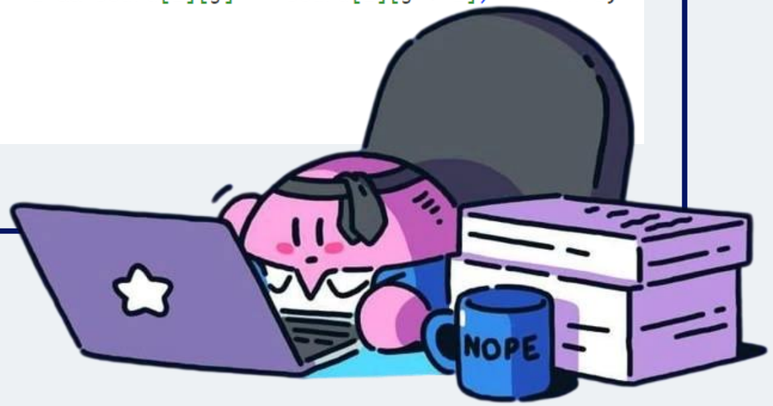

```
Your choice is: down
```

2	4	8	2
4	8	128	4
8	16	4	16
16	2	8	4

```
Game Over!
```

- 当前格子为空
- 当前格子和它下方的格子数字相同
- 当前格子和它右方的格子数字相同

```c
int is_game_over(void) {
    int i, j;
    for (i = 0; i < BOARD_SIZE; i++) {
        for (j = 0; j < BOARD_SIZE; j++) {
            if (board[i][j] == 0) return 0;
            if (i < BOARD_SIZE - 1 && board[i][j] == board[i + 1][j]) return 0;
            if (j < BOARD_SIZE - 1 && board[i][j] == board[i][j + 1]) return 0;
        }
    }
    return 1;
}
```



同济大学
TONGJI UNIVERSITY

PART3

下板演示

Acceptance of experiments

串口测试与下板

同济大学
TONGJI UNIVERSITY

操作系统移植与应用程序开发——

计算机系统实验申优答辩

谢谢

2251206 冯羽芯

2025.06.20