电子创新实验 I 报告

刘禹麟 12211351

2024 年 11 月 30 日

目录

1 摘要

本次实验报告总结梳理了前三周 stm32 课程学习的内容和问题。

2 STM32 介绍

STM32 是意法半导体（STMicroelectronics）推出的一系列基于 ARM Cortex-M 内核的 32 位微控制器，具有高性能、低功耗、可靠性强等特点。STM32 系列微控制器结合了实时功能、数字信号处理、低功耗/低电压操作和出色的连接性，同时保持高度集成和易于开发的特点。它们提供了从稳定的低成本 8 位 MCU 到拥有多种外设选项的基于 32 位 ARM Cortex-M Flash 内核的微控制器，产品线极为丰富，确保满足设计工程师们对其应用所需的性能、功效与安全性的多方位要求。

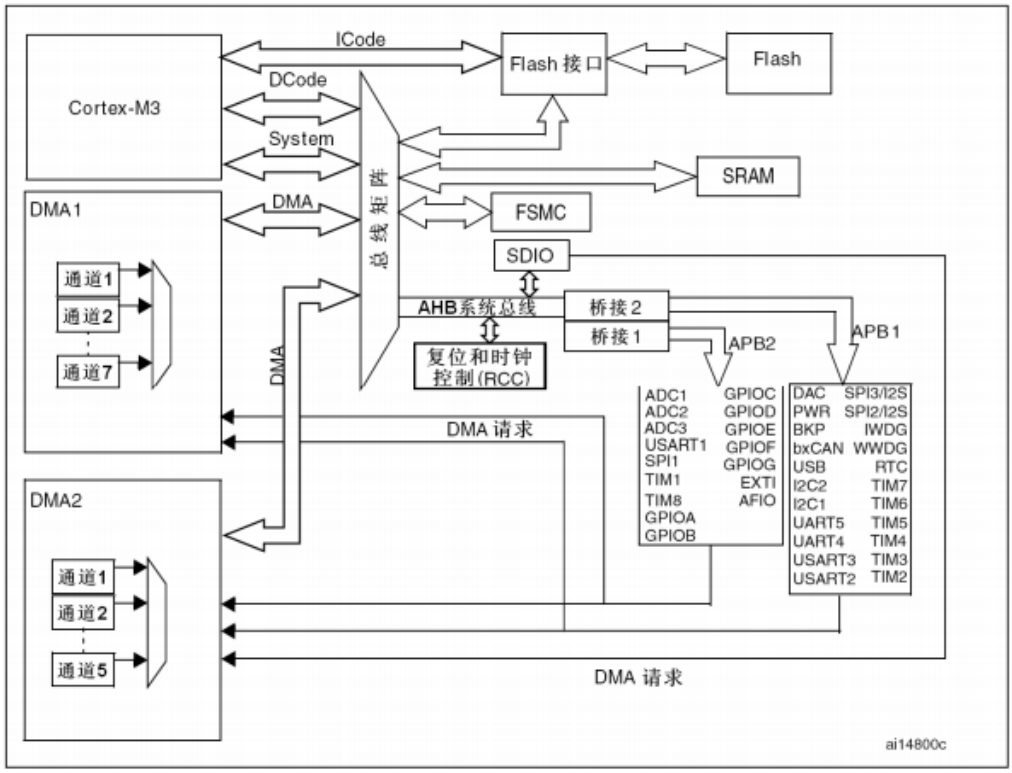

图 1: 系统结构

2.1 系统架构

通过查看参考手册，我们可以了解到 F1 系列芯片的系统机构。

1. 四个驱动单元
   (a) Cortex-M3 内核 Dcode 总线和系统总线 S-bus
   (b) 通用 DMA1 和通用 DMA2
2. 四个被动单元
   (a) 内部 SRAM

(b) 内部闪存存储器

(c) FSMC

(d) AHB 到 APB 的桥 (AHB2APBx)，它连接所有的 APB 设备

这些都是通过一个多级的 AHB 总线构架相互连接的，如图 1 所示。

2.2 总线

1. ICode 总线

   该总线将 Cortex™-M3 内核的指令总线与闪存指令接口相连接。指令预取在此总线上完成。

2. DCode 总线

   该总线将 Cortex™-M3 内核的 DCode 总线与闪存存储器的数据接口相连接 (常量加载和调试访问)。

3. 系统总线

   此总线连接 Cortex™-M3 内核的系统总线 (外设总线) 到总线矩阵，总线矩阵协调着内核和 DMA 间的访问。

4. DMA 总线

   此总线将 DMA 的 AHB 主控接口与总线矩阵相联，总线矩阵协调着 CPU 的 DCode 和 DMA 到 SRAM、闪存和外设的访问。

2.3 存储器

程序存储器、数据存储器、寄存器和输入输出端口被组织在同一个 4GB 的线性地址空间内。

数据字节以小端格式存放在存储器中。一个字里的最低地址字节被认为是该字的最低有效字节，而最高地址字节是最高有效字节。

可访问的存储器空间被分成 8 个主要块，每个块为 512MB。

3 STM32 编程

3.1 寄存器模式

寄存器本质上就是存储数据的硬件单元，其赋值需要有具体的操作或者指令。

1. 打开对应端口的时钟 STM32 在使用外设前，必须在配置之前就打开时钟，是因为时钟有同步操作、计时、节省功耗等作用。STM32 的外设需要时钟来驱动。

2. 配置输出，确定输出模式。

3. 操作对应的端口。

```
int main (void){
// 打开GPIOB端口的时钟
*( unsigned int * )0x40021018 |= ( (1) << 3 );
```

```
// 配置IO口PB5为输出，推挽输出，速率为10M
*( unsigned int * )0x40010C00 &= 0xFF0FFFFF;
*( unsigned int * )0x40010C00 |= 0X00100000;
// 控制ODR寄存器
*( unsigned int * )0x40010C0C &= ~(1<<5); //LED on
//*( unsigned int * )0x40010C0C |= (1<<5); //LED off
}
```

3.2 固件库模式

```
#include "led.h"

void LED_GPIO_Config(void){
GPIO_InitTypeDef GPIO_InitStruct;
// 打开GPIOB端口的时钟
RCC_APB2PeriphClockCmd(LED_GPIO_CLK, ENABLE);
// 配置IO口PB5为输出，推挽输出，速率为10M
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_Out_PP;
GPIO_InitStruct.GPIO_Pin=LED_GPIO_Pin;
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_10MHz;
GPIO_Init(LED_GPIO_PORT, &GPIO_InitStruct);
}
void LED_ON(void){// 输出电平
GPIO_ResetBits(LED_GPIO_PORT, LED_GPIO_Pin);//LED on
}

void LED_OFF(void){// 输出电平
GPIO_SetBits(LED_GPIO_PORT, LED_GPIO_Pin); //LED on
}
```

3.3 Start up 文件

start up 文件是运行工程的必要文件，也是工程的真正入口。创建工程时需要根据芯片的 Flask 内存在官方固件库中选择对应的 start up 文件。本课程所用芯片所用的 start up 文件为 *startup_stm32f10x_hd.s*

3.4 点亮 PC13 端口 LED 灯

现将 LED 改为接在 PC13 端口上，代码修改如下：

```
int main(void){
//打开GPIOC端口的时钟
*(unsigned int*)0x40021018 |= (1<<4);
//配置IO口PC13为输出，模式为推挽输出，速率为10M
*(unsigned int*)0x40011004 &= 0xFF0FFFFF;
*(unsigned int*)0x40011004 |= 0x00100000;
//控制ODR寄存器
*(unsigned int*)0x4001100C &= ~(1<<13); //LED on
*(unsigned int*)0x4001100C |= (1<<13); //LED off
}
```

3.5 GPIO Init 方法实现过程

GPIO_Init(GPIOB,&GPIO_InitStructure) 的每一步实现过程如下。

声明变量:

```
uint32_t currentmode = 0x00, currentpin = 0x00, pinpos = 0x00, pos = 0x00
    ;
uint32_t tmpreg = 0x00, pinmask = 0x00;
```

查看输入变量合法性:

```
assert_param(IS_GPIO_ALL_PERIPH(GPIOx));
assert_param(IS_GPIO_MODE(GPIO_InitStruct->GPIO_Mode));
assert_param(IS_GPIO_PIN(GPIO_InitStruct->GPIO_Pin));
```

配置 GPIO 模式:

```
// 将输入参数GPPIO_Mode的低四位暂存在currntmode
// currentmode = 0b00001000
currentmode = ((uint32_t)GPIO_InitStruct->GPIO_Mode) & ((uint32_t)0x0F);
// 判断是否为输出，若是输出，则配置速度；若为输出，则跳过这一步，
// 这里0x08&0x10 = 0x00,为输入模式，故跳过配置速度。
if ((((uint32_t)GPIO_InitStruct->GPIO_Mode) & ((uint32_t)0x10)) != 0x00){
/* Check the parameters */
assert_param(IS_GPIO_SPEED(GPIO_InitStruct->GPIO_Speed));
/* Output mode */
currentmode |= (uint32_t)GPIO_InitStruct->GPIO_Speed;
}
```

配置 CRL 为 pin2, 下拉输入模式。

```
/* 配置低八位寄存器 */
// 判断是否为低八位: 0x0010 & 0x00FF = 0x0010 != 0x00,故为低八位。
if (((uint32_t)GPIO_InitStruct->GPIO_Pin & ((uint32_t)0x00FF)) != 0x00){
// 通过一系列赋值，配置好CRL寄存器，且模式为IPD
// 备份寄存器的值
tmpreg = GPIOx->CRL;
// 循环，从Pin0开始，到Pin8结束，找到对应的Pin
for (pinpos = 0x00; pinpos < 0x08; pinpos++){
  pos = ((uint32_t)0x01) << pinpos;
  /* Get the port pins position */
  currentpin = (GPIO_InitStruct->GPIO_Pin) & pos;
  // 输入的GPIO_Pin = 0x04，所以对应pin2，此时pinpos = (0x04)H = (2)D
  if (currentpin == pos){
    // pos为pinpos的值左移两位，或乘4
    pos = pinpos << 2; // pos = (8)D
    /* Clear the corresponding low control register bits */
    // 将Pin2对应寄存器清零
    // pinmask = 0b0000 1111 << 8 = 0b0000 0000 0000 0000 0000 1111 0000
       0000
    pinmask = ((uint32_t)0x0F) << pos;
    // tmpreg &= 0b1111 1111 1111 1111 1111 0000 1111 1111，pin2对应的寄
       存器被清零
    tmpreg &= ~pinmask;
    /* Write the mode configuration in the corresponding bits */
    // 再给寄存器写入将要配置的引脚模式
    // tmpreg |= 0b0000 0000 0000 0000 0000 1000 0000 0000
    tmpreg |= (currentmode << pos);
    /* Reset the corresponding ODR bit */
    // 输入模式为下拉，则BRR寄存器写1对引脚置0
    if (GPIO_InitStruct->GPIO_Mode == GPIO_Mode_IPD){
      GPIOx->BRR = (((uint32_t)0x01) << pinpos);
    }else{
      /* Set the corresponding ODR bit */
      if (GPIO_InitStruct->GPIO_Mode == GPIO_Mode_IPU){
        GPIOx->BSRR = (((uint32_t)0x01) << pinpos);
      }
    }
```

```
  }
}
GPIOx->CRL = tmpreg;
}
```

将配置好的 GPIO_InitStructure 赋值给 GPIOB。

```
currentmode = 0x0000;
currentmode = 0x0008;
tmpreg = GPIOx->CRL
for to find currentpos = 0x04; pinpos=5;
pos = 8;
pinmask = 0x00000F00;
tmpreg &= 0xFFFFF0FF;
tmpreg |=0x0008 << 8;
GPIOx->CRL = tmpreg;
```

4 STM 官方固件库文件结构梳理

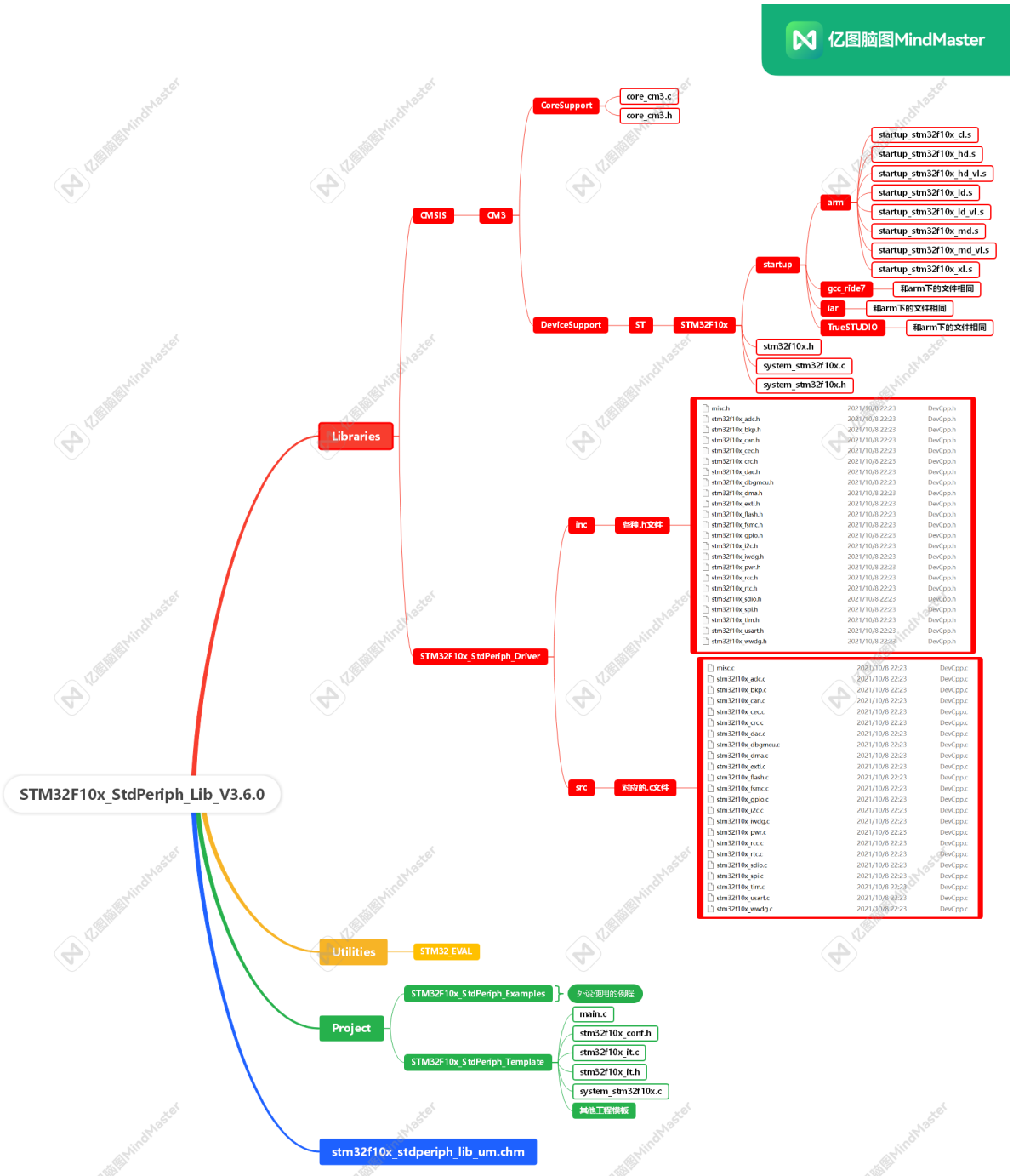
亿图脑图MindMaster
STM32F10x_StdPeriph_Lib_V3.6.0
Libraries
CMSIS
CM3
CoreSupport
core_cm3.c
core_cm3.h
DeviceSupport
ST
STM32F10x
startup
arm
startup_stm32f10x_cl.s
startup_stm32f10x_hd.s
startup_stm32f10x_hd_vl.s
startup_stm32f10x_ld.s
startup_stm32f10x_ld_vl.s
startup_stm32f10x_md.s
startup_stm32f10x_md_vl.s
startup_stm32f10x_xl.s
gcc_ride7
和arm下的文件相同
iar
和arm下的文件相同
TrueSTUDIO
和arm下的文件相同
stm32f10x.h
system_stm32f10x.c
system_stm32f10x.h
STM32F10x_StdPeriph_Driver
inc
各种.h文件
src
对应的.c文件
Utilities
STM32_EVAL
Project
STM32F10x_StdPeriph_Examples
外设使用的例程
STM32F10x_StdPeriph_Template
main.c
stm32f10x_conf.h
stm32f10x_it.c
stm32f10x_it.h
system_stm32f10x.c
其他工程模板
stm32f10x_stdperiph_lib_um.chm

图 2: 文件结构梳理