SRAM-Flash 分级存储架构设计

主要参数

- 流水线设计（待完成）
- 片外独立 Flash：2MiB*1
- 主 SRAM：128KiB*1
- 次 SRAM：512Byte*4

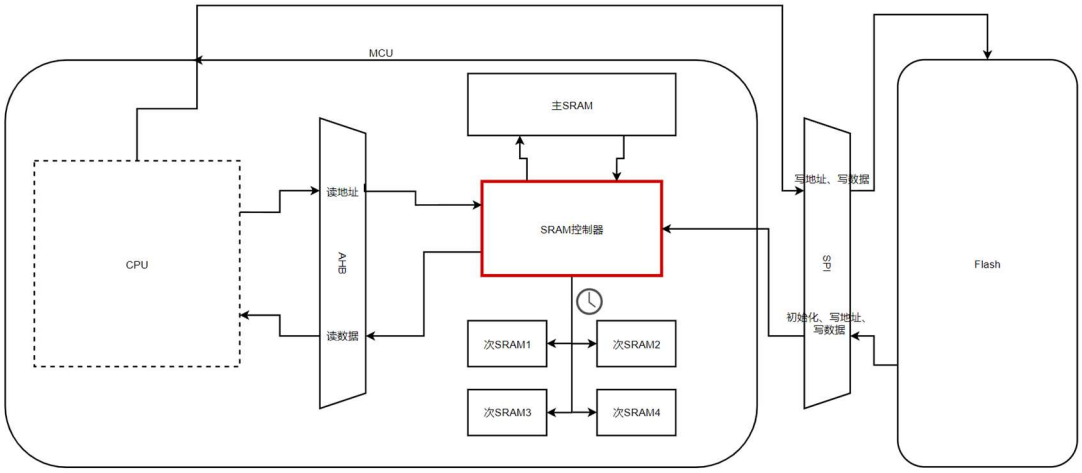
概述

本设计对标 STM32F10x。设计所述 MCU 中，SRAM 作为一个外设挂载在 CPU 的总线上；而大容量 Flash 作为独立的芯片与 MCU 一同被封装，使用 SPI 接口与 MCU 相连。

设计采用了分级存储架构实现大容量片外 Flash 的快速读写。具体而言，这一架构使用了内存页置换中的 Clock 算法（F J Corbató *et al*, 1969）。更进一步地，这一架构在分级存储的基础上增加了流水线设计，以实现更高性能的存储器读写。

流水线设计待定

结构框图与模块功能



- **AHB 总线**：一种主要用于连接高性能低带宽外设的总线规范，是使用本设计的 MCU 采用的片上总线。在本设计中，AHB 总线仅包含从机接口，主机是本设计之外的 CPU 核。
- **SRAM 控制器**：控制主次 SRAM 的读写功能，存有各个 SRAM 块的地址数据与次 SRAM 的使用位（clock bit）数据。对于读操作，其响应 AHB 总线发送的读请求，查找 SRAM 块，返回读取数据或 SRAM 未命中信号；对于写操作，其在 MCU 上电时从 Flash 中读取数据初始化主 SRAM，在运行时响应 Flash 发送的写请求。

• **主 SRAM**：128KiB 的 SRAM，主要用于存储程序代码与静态数据，具体存储内容由编译器指定。MCU 上电时，SRAM 会将 Flash 固定地址段的数据初始化进入主 SRAM 中；运行时，如果对应地址段中的数据被擦除或写入，SRAM 会更新其中数据。

•**次 SRAM**：4 个 512Byte 的 SRAM，用于存储其它数据。其置换策略采用 Clock 算法。

• **SPI 接口**：一种串行外围设备接口，常用于外部存储器的连接中。本设计中，SPI 接口的从机是片外 Flash 芯片，主机是包含了 CPU 核与 SRAM 控制器的 MCU 整体。其中，CPU 核输入读写请求，SRAM 控制器接收 Flash 返回的读数据。

• **Flash**：2MiB 的大容量 Flash 原型，带有简化的 Flash 控制器，存储有所有数据。

介绍

读取流程：

当 MCU 上电时，SRAM 控制器从 Flash 中加载固定地址范围的数据进入主 SRAM 进行初始化，清空 4 个次 SRAM 的地址范围与使用位。

CPU 读数据算法如下：

```
if (address in main)
  read main
else:
  loop:
    if (address in sub[i])
      read sub[i]
      sub[i].clk_bit = 1
      i = i+1
    else:
      sub[i].miss = 1
      i = i+1
  read miss = sub[0].miss & sub[1].miss & sub[2].miss & sub[3].miss
```

当 CPU 需要读数据时，会通过 AHB 总线发送读请求进入 SRAM 控制器。如果数据地址位于主 SRAM 地址范围内，SRAM 控制器直接从对应地址取出数据并送回 CPU 中。

如果数据地址不在主 SRAM 地址范围内，SRAM 控制器会~~（从最近读取过的次 SRAM 开始，）~~将这一地址依次与各次 SRAM 的地址范围进行比对，如果次 SRAM 已加载的数据地址范围包括了需读取的数据地址，SRAM 控制器将从对应的次 SRAM 中取出数据并送回 CPU 中，同时将该次 SRAM 对应的使用位置为 1。

更进一步地，如果所有 SRAM 均没有加载该数据地址，则未命中信号会被置为 1 并返回 CPU。

CPU 接收到未命中信号后，会从 SPI 接口发送读 Flash 请求。Flash 接收到读请求后，将所需数据及其地址一同送入 SRAM 控制器中。SRAM 控制器将使用 Clock 算法对次 SRAM 中数据进行替换，算法如下：

```
pointer = sub[i]
Loop:
```

```
  if sub[i] = sub[3]
    sub[i].next = sub[0]
  else:
    sub[i].next = sub[i+1]
  if sub[i].clk_bit =0
    pointer = sub[i]
    break
  else:
    pointer = sub[i].next
    sub[i].clk_bit = 0
write data from Flash to pointer
read data from pointer to AHB
```

一个指针最初指向最后一次更新过的次 SRAM 块，随后开始判定其使用位是否为 1。如果使用位为 1，将使用位清零，随后指针指向下一个次 SRAM 块；否则，将该 SRAM 块替换为需要读取的数据，随后从该 SRAM 块中读取数据进入 CPU，完成新数据的读取流程。

需要注意的是，4 个次 SRAM 块是首尾相接的，与指针一同构成了一个形似时钟的硬件链表，故而称其为 Clock 算法。

Clock 算法是一种优化过的二次机会算法，同时也是 LRU 算法的一种近似。其复杂程度较二次机会算法低，而工作效率与二次机会算法保持一致。Clock 算法利用了局部性原理，次 SRAM 块的读取流程将对应块的使用位置为 1，在进行替换时给予了最近被读取过的 SRAM 块一次豁免机会，转而替换链表中连续两次未被使用过的块，减小了前者未来再次被读取时的延迟，提高了程序运行效率。

写入流程：

写入流程算法如下：

```
write data from SPI to Flash
output write data and address from Flash to SRAM controller
if (address in main)
  write main
else:
  loop:
    if (address in sub[i])
      write sub[i]
      i = i+1
```

当 CPU 需要写数据时，其直接通过 SPI 接口向 Flash 发送写请求。Flash 在完成写操作后，通过 SPI 接口向 SRAM 控制器返回写地址与写数据。如果写地址包含在主 SRAM 地址范围内，SRAM 控制器将对主 SRAM 对应地址数据进行更新；如果写地址包含在任意一个次 SRAM 块中，SRAM 控制器将更新对应的次 SRAM 块；如果写地址没有在任何一个 SRAM 块中，不进行任何操作。

整体而言，本设计数据通路基本单向，减小了硬件复杂度，同时为流水线的添加提供了

便利。