微码控制器设计与实现

微码是在 CISC 结构下，运行一些功能复杂的指令时，所分解一系列相对简单的指令。相关的概念最早在 1947 年开始出现。微指令的作用是将机器指令与相关的电路实现分离，这样一来机器指令可以更自由的进行设计与修改，而不用考虑到实际的电路架构。与其他方式比较起来，使用微指令架构可以在降低电路复杂度的同时，建构出复杂的多步骤机器指令。撰写微指令一般称为微程序设计，而特定架构下的处理器实做中微指令有时会称为微程序。

（以上内容参考 Wikipedia）

任务简介

您已经成功地在实验中设计了流水灯外设。现在，您需要一个微码控制器来轻松更改流水灯的运行模式。Cortex-M0 通过总线向微码控制器写一个起始地址，微码控制器便从这个起始地址开始执行微程序，微程序存储在一个存储器中。

任务目标

目标系统结构框图如下所示。微码流水灯结构可以分成以下三部分，首先为存放微码数据（程序机器码）的双口 RAM，一个读写端口通过 AHB 总线与 M0 处理器连接，一个读端口连接到微码控制器（controller）。其次是微码控制器（controller），负责根据处理器所给的配置信息，读取对应的微码，配置信息为微码的起始地址 s_addr、结束地址 e_addr 和 GPIO 的输入数据 indata，当 GPIO 配置为输入模式时有效。最后一个部分就为实验三所介绍的 GPIO。

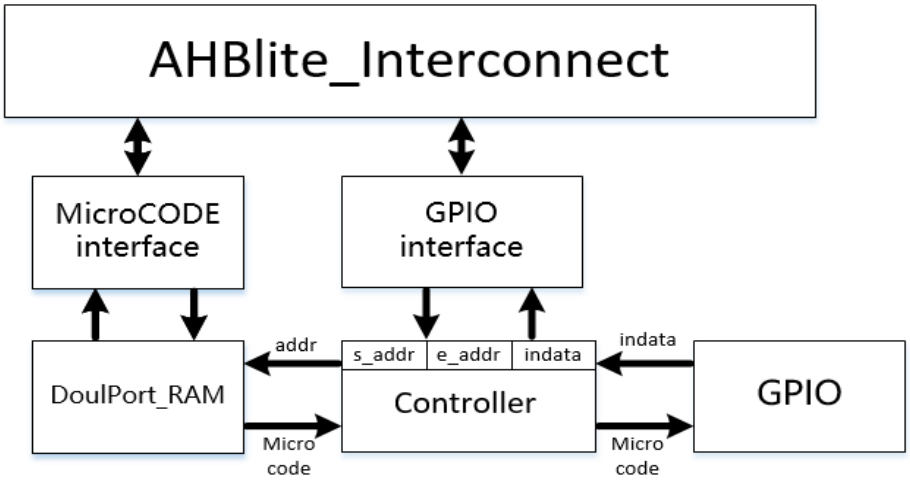

图 1 微码流水灯系统框图

我们本次任务的目的时（是）利用微码结构来控制 GPIO 实现流水灯的功能。那么 GPIO 实现流水灯需要的 3 个数据就必须能在微码的结构中体现，这三个部分就包括 GPIO 输出状态、GPIO 输出使能和 GPIO 输出延时，其在微码中的映射如图 2 所示。可以看到微代码的 32 为（位）中，23 位用来存放输出延时（在 50M 时钟下最多延迟不到 0.2s），第 9 位用来表示 GPIO 的输入输出使能，低 8 位用来表示 GPIO 的输出状态。

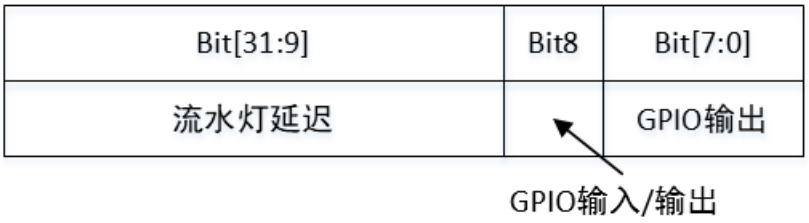

图 2 微码数据分配映射

接下来，根据微代码的结构，我们的微码控制器（controller）应该拥有一个译码功能，即识别 32 位的微代码的数据映射。并且，识别出数据映射以后，我们还应该有一个执行结构，即一个计数器计数等于延迟信息的数值时，将配置信息（微代码的 0 到 8 位）送入 GPIO 让它输出，同时读取下一个微代码。

根据以上提示，利用 GPIO 微码结构设计具有左移、右移、闪烁、全亮模式的流水灯，这几种不同的运行模式可以通过不同的微指令控制，通过微指令的不同组合，流水灯可以工作在各种模式下。因此您需要设计起主要控制作用的微码控制器，以及存储微指令所使用的存储器。

常见疑问

Q：不同微指令对应机器码的映射关系是怎样的

A：可以自行设计

Q：微指令如何存储

A：可以通过存储器初始化的方式，也可以通过总线写入。

如果还有其他的问题，建议在 qq 群内直接提出讨论。

Q：怎么进一步增大流水灯的输出延迟

A：再使用一个计数器，计数满多个延迟的值再经行（进行）下一条，就能成倍增加延迟