西安交通大學

MSP430 实验报告

题　目 关于 MSP430 的 DCO 自校准和检测的研究实验

电气　学院　电气工程与自动化　系

学生姓名　陈钱牛　班级　电气 64 班　学号 2160400096

学生姓名　王　玮　班级　电气 64 班　学号 2160400093

学生姓名　梁永回　班级　电气 63 班　学号 2160400075

指导教师　金印彬 、 宁改娣

设计所在单位　西安交通大学电气学院

2018 年 6 月

分工与给分

我们组分工合作完成此次任务。实验之前，我们开小会讨论题目，提了很多也否决了很多。原因是我们觉得我们最后的大作业一来是对我们学 MSP430 半学期以来的总结检验，二来也应该是能够为宁老师、金老师的教学工作做出一定的贡献。于是我们最终决定做“DCO 的自校准”这个题目。这个内容首先很实用，解决了 MSP430 时钟源准度与广度的问题，如果把这个问题研究透了，可以省去很多的麻烦；其次这一块考察到了我们对最基础内容的掌握同时也考察到了对新知识的学习能力，内容的涉及面很广，算是充分调用了 MSP 内部的资源；最后，确实这部分不太引起同学们的重视，大家都想着要做 MP3、内存卡等等高端大气的东西，反而瞧不起对片内资源的研究实验。我们觉得越是在大家忽视的地方越是能发力做出优异的成绩。所以，我们觉得“DCO 的自校准”这个题目是在众多选择中做出的一个理性的选择，从一开始我们就知道这个题目只要认真去做，肯定可以做出来；但是，我们要把它做透彻。

从一开始我们就确定了整体的结构与各种基本的变量定义与底层模块，然后具体能力分到个人，各自实验完成模块设计，最后再综合调试，编写注释。整个**初步的设计过程**持续了两天（总用时一个多礼拜），我们互相帮助，最后高效地完成了设计任务。实验期间，共去过实验室三次，主要是为了使用带晶振的板子和高带宽的示波器。最后写报告的时候我们三人也是通力合作，分工完成，每个人都付出了宝贵的时间和心血。

具体分工细节如下表:

姓名	程序设计与实验	报告撰写内容	建议给分
陈钱牛	整体架构设计 检测模块设计实验 综合调试 DCO 校准实验改进	第一章“概述”部分; 第三章“DCO 自校准和检测”部分; 第五章二、三节部分; 整合、审阅、修改。	50
王玮	写入模块设计实验 主程序注释 DCO 校准实验记录与分析 15MHz 误差研究	第四章“DCO 校验实验; 第五章中第一节“15MHz 误差研究”。 审阅、修改。	50
梁永回	测频模块设计实验 校准模块设计实验 DCO 校准实验改进	第二章“DCO 自校准原理” 审阅、修改。	50

注意：其中实验部分各自负责的部分都有实验验证通过才综合，总实验也就是 DCO 观测实验是三人一起到实验室一边实验一边改进程序完成的。

报告题目：关于 MSP430 的 DCO 自校准和检测的研究实验
学生姓名：陈钱牛　王　玮　梁永回
指导教师：金印彬　宁改娣

摘　要

本文详细介绍了 MSP430G2553 的 DCO 振荡器及其配置校准方法，通过实验校准 DCO 振荡器并检测校准后的频率。实验内容主要包括两部分：一是使用 32.768kHz 时钟晶振对 DCO 振荡器的所有可能的产生的频率进行准确测量，选出与若干目标频率最接近的若干组配置参数,写入 InfoFlash 中，方便后续调用；二是使用频率计测量校准的频率以及观察其输出波形，与理想值、出厂预设值进行比较。实测结果令人满意。在一定误差范围，校准后的 DCO 振荡器在 1MHz 到 16MHz 内精准可调，且配置方便。这为程序开发提供了极大的设计空间。在一定误差范围内，开发人员只需通过配置 DCO 振荡器即可满足其对高频时钟的要求。此外，本次实验过程中我们使用板内资源设计了 16MHz **高频频率计**，并通过实验详细解释了 15MHz 无法精准校准和产生混频抖动的原因。

关 键 词：MSP430；DCO；自校准；频率测量

目　录

1 前　言

对于MSP430来讲，DCO振荡器算是其一大特点。不需要任何晶振我们就可以通过DCO振荡器获得一个比较准确的高频时钟源，而且这个时钟源是可以通过配置时钟控制寄存器在0.06~20MHz间自由选频的。这极大地满足了我们对系统时钟源的需求。但是，由于DCO振荡器本身的结构特点，DCO振荡器的本质是RC振荡器，受环境因素及自身误差的影响较大，常常不能直接用作精准的振荡源。

DCO振荡器内部本身是允许自校准的，在0.06~20MHz范围内，用户可以通过改变RSELx、DCOx、MODx三个寄存器的值得到一个相对准确的频率。事实上，MSP430在出厂前就已经在InfoFlashA中写入了4个频率的校验值，分别是1MHz、8MHz、12MHz、16MHz。这四个值相对较准，但是由于测试环境与应用环境不同，它们并不是DCO振荡器在实验环境下能产生的最优配置方案。而且它们给用户的选择范围有限，用户只能在四个频率中选择其中一个。

那么用户能否自己校准DCO的频率呢？答案是肯定的。用户需要做的是找到DCO振荡器能提供的所有频率中与目标频率最接近的频率，并将它的寄存器配置值存到Flash中。在需要调用这个频率的时候，只需要从Flash中读取相应地址中存储的值并将它赋给寄存器，就可以得到当前环境下最理想的DCO振荡频率。通过实验，我们不仅完成了对DCO频率的校验，而且发现校验结果较为准确，达到了预期的效果。

在本实验中，我们设计了高频频率计，理论上可以测量的最高频率为MSP430可以接受的最大频率。我们在实验过程中观测到了混频抖动现象，通过查阅资料解释了其产生的原因。实验后，我们发现15MHz的频率点误差特别大，并通过实验详细解释了其原因。

在阅读本文的过程中，部分需要注意的点在此提前注明，以免读者产生困惑：

a)　本文中的频率单位均采用kHz。

b)　纯净频率：MODx=0时DCO震荡产生的频率，共有128个（16×8）；

c)　小数频率：MODx≠0时，DCO震荡产生的频率，每两个纯净频率区间内有31个；

d)　出厂预设校准值：MSP430中预存的可以直接调用的DCO设置参数，有1/8/12/16MHz四个频率；

e)　混频抖动：小数频率在用示波器观测时会出现的波形混叠现象；

2 DCO 自校准原理

2.1 自测 DCO 准确频率

2.1.1 DCO 振荡器简介

M430G2553 时钟模块包含有三个时钟源：LFXT1CLK，DCOCLK 和 VLOCLK，在 User's Guide 中，有如下介绍:

a) *LFXT1CLK: 低频/高频振荡器，可用于低频手表晶体或 32768hz 的外部时钟源，或用于标准晶体、谐振器或 400-kHz 至 16-MHz 范围内的外部时钟源。*

b) DCO*CLK: 内部数字控制振荡器(即* DCO*)*。

c) *VLOCLK: 内部低功率、低频振荡器，12-kHz 的典型频率。*

由以上介绍和实验经验可知：通常可以从外部接入 32768Hz 的低频时钟源，较为准确；而内部数字控制的 RC 振荡器(DCO)的频率稳定性较差；内部低功耗低频振荡器典型频率值为 12kHz，精准性也不高。频率的稳定性在某些情况下对实验结果及性能的影响很大。环境一变，振荡频率也跟着变，性能时好时坏，这绝对不行。此外，我们可以发现，想要在 MSP430G 系列单片机上获得高频信号，只能借助于“频率不太稳定”的内部数字振荡器 DCO。

DCO 是个开环控制的系统，没有负反馈，频率不太稳定。但是我们为什么可以使用 CALBC2_1MHZ、CALDCO_1MHZ 这两个时钟设置参数呢？而且使用它们配置后的输出频率比较准确。这是因为厂家在生产产品时，已将 1/8/12/16MH 四个频率进行校准并将其设定参数写在了 Info FlashA 段中。每一块单片机的参数都不一样，但是没关系，如果对精度要求不太高的话，放心大胆的调用就行了。

可是，如果我想要的高频不是上述四个频率，或者我嫌这四个频率不准，又该怎么办呢？我们是否可以自行校准 DCO 的频率，照猫画虎，把校准的参数存在 Info FlashB 段中呢？在实验中，调用自己写的函数，就能获得想要的高频。在 MSP430 中，答案是肯定的。

2.1.2 DCO 自校准原理

我们可以先想想如何自测 DCO 准确频率，设想下面一个场景。现在有两块石英表：表 A 和表 B。它们的秒针都能均匀地转动，只不过转圈圈的速度不同。表 A 很准确，秒针转一圈，时间流逝 60 秒，而表 B 秒针转一圈，时间流逝多少秒是未知的，记为 T 秒。二表约定，

同时从 0 开始转圈，当 A 转一圈的时候给 B 发出信号，B 收到信号后立马记录下自己的位置，也就是转的圈数。假设 B 此时转了 N 圈。则 B 转一圈的用时可由下式得出。

$$\frac{60\text{ 秒}}{\text{圈}} \times 1\text{ 圈} = \frac{\text{T 秒}}{\text{圈}} \times \text{N 圈}\ ;$$

$$\text{T} = \frac{60}{N}\text{秒}\ ;$$

例如，N=2 圈，那么 B 表转一圈用时 30 秒。从此以后，B 表记住了这样一件事情：每转 2 圈，时间流逝 60 秒。理论上，表 B 可以知道自己转任意圈数的用时。

我们把 DCO 看做待校准的表 B，把较为准确的外部低频时钟源（32768Hz）看做表 A。校准完成后，我们把得到的圈数N_1，N_2，N_3……记到本子上，要用的时候翻一翻就行了。

2.2 DCO 校验参数

设置 DCO 时，有三个寄存器变量需要设置：RSELx，DCOx，MODx。

RSELx 共有 16 档，用于粗调。由图 2-1，RSEL=0 时频率在 100kHz 附近，RSEL=7 时候，频率在 1000kHz 附近，RSEL=15 时，频率在 20000kHz 附近。

DCOx 共有 8 档，用于细调，每档频率步进约 10%。由图 2-1 可以看出，DCO 越大，频率越高。

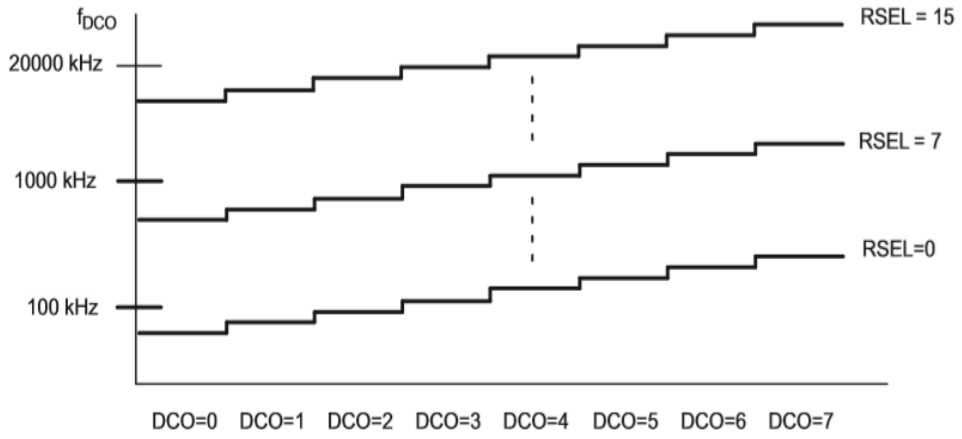


图 2-1 DCOx、RESLx 与 DCO 频率的关系图

MODx 共有 32 档，用于混频，即交替输出 DCOx 和 DCOx+1 两种频率，可以获得更小的频率增幅，见图 2-2。混频后周期由下式计算：

$$T_{\mathrm{x}} = (32 - MOD_{\mathrm{x}}) \times t_{DCO} + MOD_{\mathrm{x}} \times t_{DCO+1};$$

例如，MODx=5 时，

$$T_5 = 27 \times t_{DCO} + 5 \times t_{DCO+1};$$

RSELx，DCOx，MODx 三者组合，共有 16×8×32=4096 种组合。现在给定一个待校验频率 f，我们是不是“比对”4096 次，再找出最接近的那个呢？恐怕有点傻。本实验中，我们先校验 RSELx 和 DCOx 对应的 128 种组合，选出最接近 f 但要求小于 f 的一组，在该种组合下再校验 MODx，逐次增加 MODx 找到最接近（差值绝对值最小）的 MODx。

为什么第一次校验要求最接近但小于频率 f 的那一组呢?如果我们洞悉了 MODx 所代表的的混频器原理，就不难理解。混频器只能使$\mathrm{DCO_x}$和$\mathrm{DCO_{x+1}}$的频率混合输出，也就是说，在$\mathrm{DCO_x}$确定以后，频率只能增加，不能减少。$\mathrm{MOD_x} = 32$时，相当于$\mathrm{DCO_{x+1}}$的频率（高于$\mathrm{DCO_x}$的）。所以，在第一次校验时要求“接近但小于 f ”。第二次校验是最后一次校验，当然是约接近越好，也就是差的绝对值越小越好，不难理解。

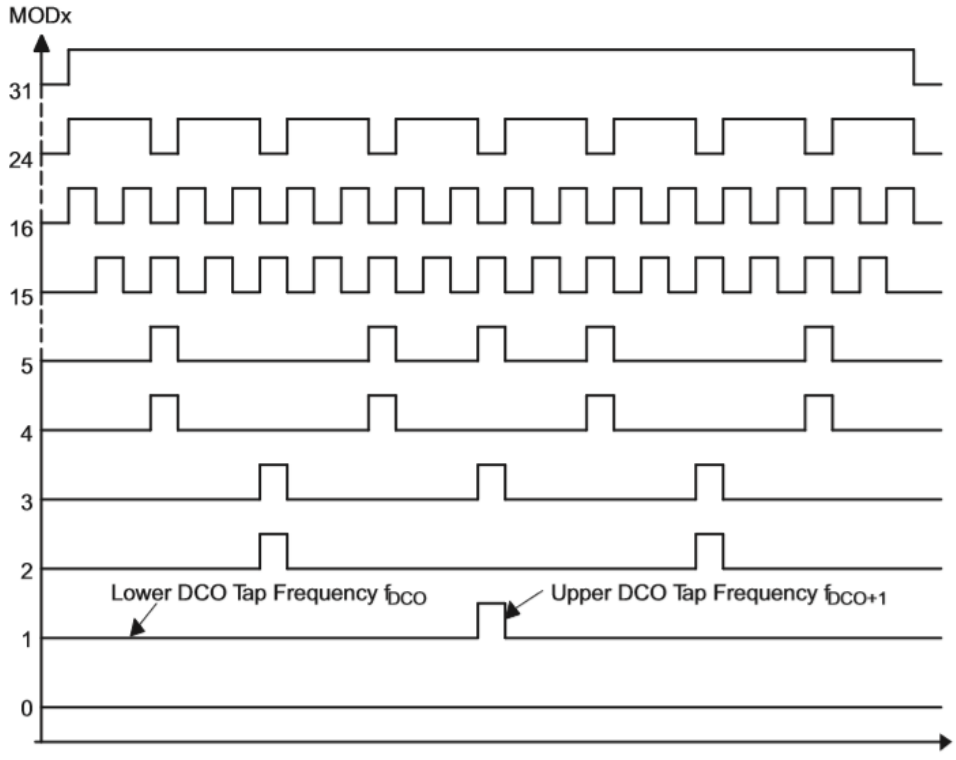


图 2-2 混频原理图

DCOx 和 MODx 由 DCOCTL 控制，如图 2-3。RSELx 由 BCSCTL1 的低 4 位控制，如图 2-4。改变 DCOCTL 和 BCSCTL1 中参数就可以改变 DCOx、MODx 和 RESELx 的值。获得 RSELx，DCOx，MODx 三组校验参数后，整理打包，写到 Info FlashB 中，设置后，断电后数据不丢失。手里拿着较为准确的频率，就可以开开心心做实验了，需要什么频率就调用什么频率。

当然，把校验参数写入 Info FlashB 需要自编函数实现；仿照调用 1/8/12/16MH 四个频率时的方法，调用任意频率也可以通过自编函数实实现，好在这些都不麻烦。

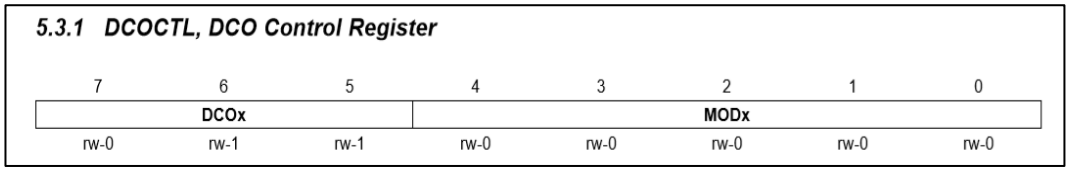


图 2-3

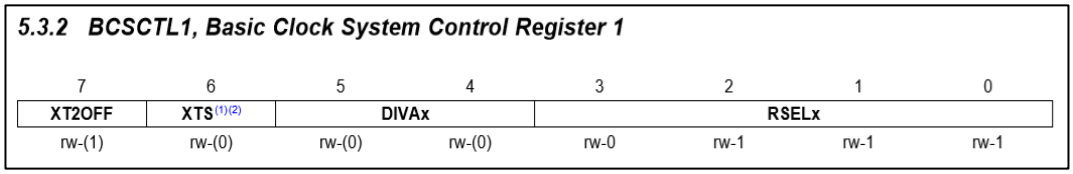


图 2-4

2.3 Info_Flash 的擦除和写入

首先我们来看看这块单片机的内存编址（以 32KB 为例）。

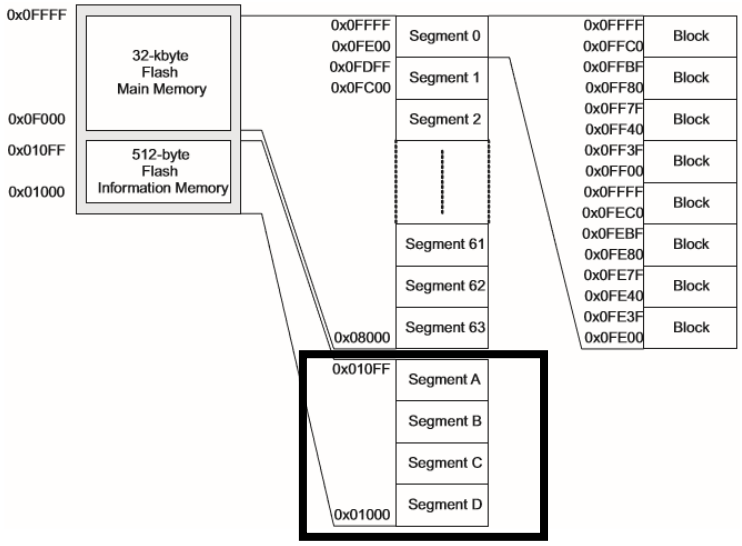


图 2-5 单片机内存编址

其中“512-byte Flash Information Menory”即 Info Flash，地址范围 0x01000~0x010FF，而 Info_Flash 又分成了 A、B、C、D 四个段。Info_Flash A 段存放了出厂校验数据，1/8/12/16MH 四个频率的设定参数就写在这里。

Flash 既然可以存放参数（类似于存放货物），那么就涉及到“存货”和“取货”，也就是“写”和“读”。我们想象一下取货和存货的过程，取货只需要知道货架的地址就行了，而存货呢？好像比较麻烦。下面讲一讲 Info_Flash 的擦除和写入。

存货时不能太快也不能太慢：太快了存货容易出错 (写入不可靠)，太慢了仓库的寿命就得不到保证 (降低 Flash 擦写寿命)，现在库房制定了规定：时钟频率必须在 257kHz~476kHz 之间。这个频率相对来说比较高。正好，DCO 能提供 1MHz 较为准确的的频率（看来出厂时的校准参数很重要，千万别丢了），分频输出给 Flash 时钟即可。图 2-6 为 Flash 时钟控制器。FNx 可进行分频。

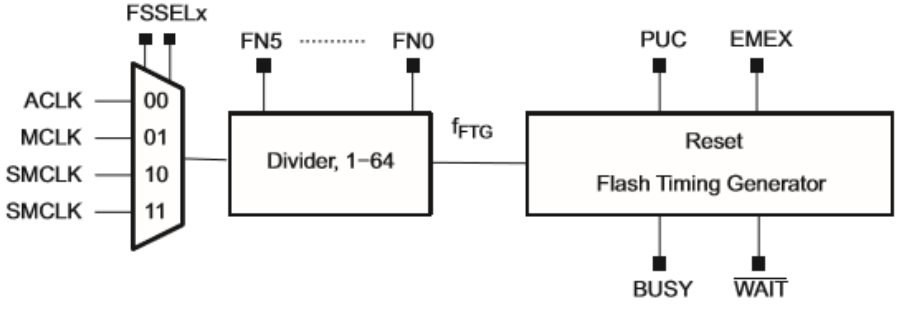


图 2-6Flash 时钟控制器

Flash 的擦除是改写 Flash 的前一步，擦除次数是有限的（典型值为 10 万次），每一次写都要擦除，哪怕改动的数据只有一点点，也得擦，这就是规定。擦除模式的选择由 MERAS 和 ERASE 完成，最常用的是 MERAS=0，ERASE=1，选择单段擦除, 擦出过程的时序如图 2-7。擦除的全程，BUSY=1，打开编程电压发生器，进行擦除操作，最后关闭编程电压发生器。完成 Flash 擦除的配置顺序如图 2-8。

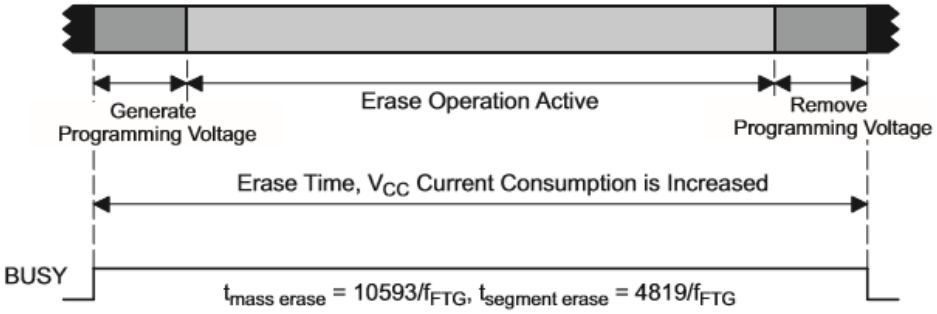


图 2-7 擦除时序

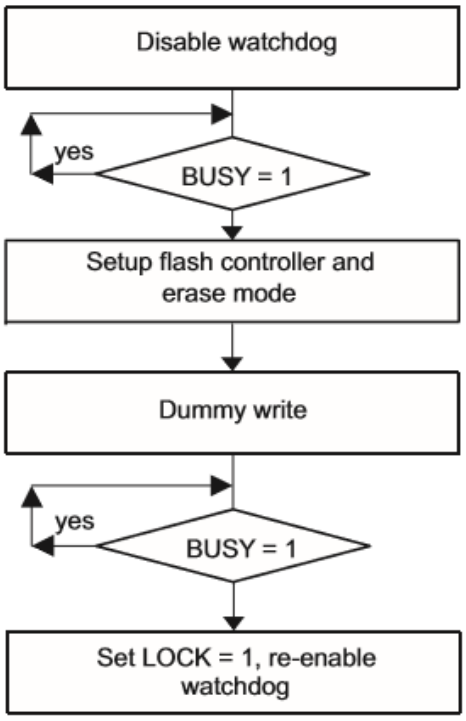


图 2-8 Flash 擦除的配置顺序

仓库没货时，相当于货架上每一位都是 1，写数据就是把 1 改成 0。写入有两种方式，字节/字写入和块写入。

字节/字写入方式，如图 2-9，BUSY 始终为 1，先打开编程电压发生器，然后写入 Flash，最后关闭编程电压发生器。完成这样一个过程，可以写 8 位（1 字节）或者 16 位（1 字），每次都要打开和关闭编程电压发生器。

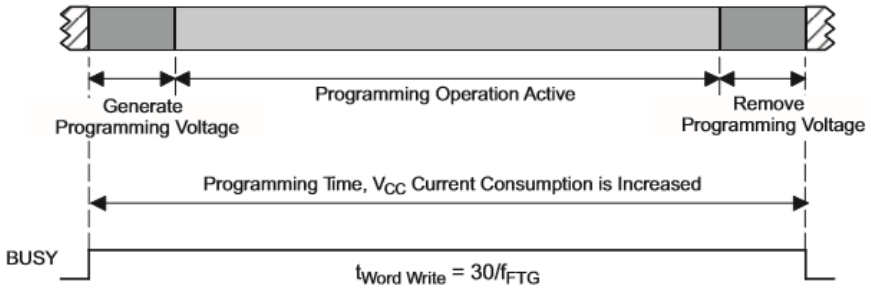


图 2-9 字节/字写入时序

块写入一次可以写 64 字节，整个过程与 BUSY=1，与节/字写入方式相同，但多了一个 WAIT 位，WAIT=0 时，说明正在写某字节，WAIT=1 时，说明某字节写入完成，可以进行下一个字节了。由于完成 64 字节只需要打开/关闭一次编程电压发生器，速度比字节/字写入快了不少。本实验中使用字节/字写入的方式，并且块写入较为复杂，在实现少数特殊功能时才启用，小打小闹就不要兴师动众了，故在此对块写入不做过多介绍。

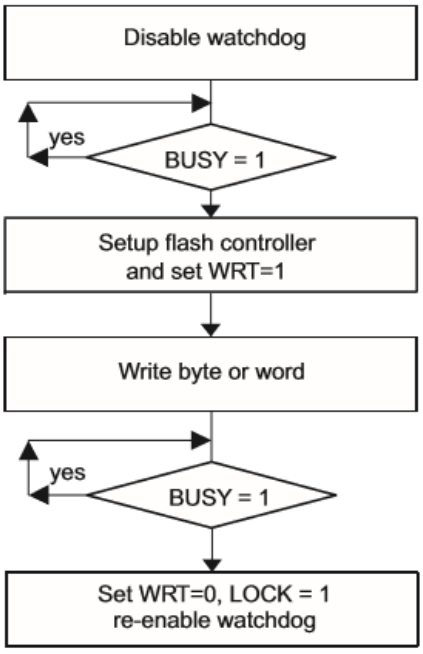


图 2-10 字节/字写入的配置顺序

最后，图 2-11 贴出了 Flash 的控制寄存器（FCTL4 不常用，未列出），便于进行查找设置。主要几位列出如下：

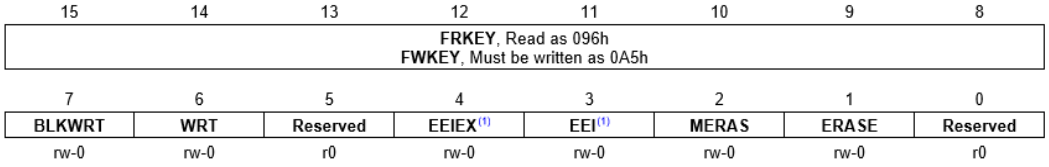
a) *BLKWR 和 WRT：开启/关闭字节/字或块写入*

b) *MERAS、ERASE 和 LOCKA：选择擦除方式*

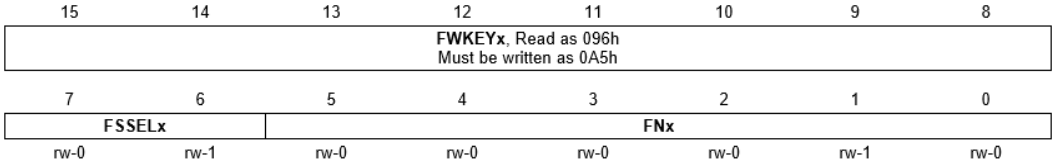
c) *FSSELx、FNx：Flash 时钟以及分频*

d) *LOCK：锁定 Flash 擦写，一旦锁定，不允许擦写*

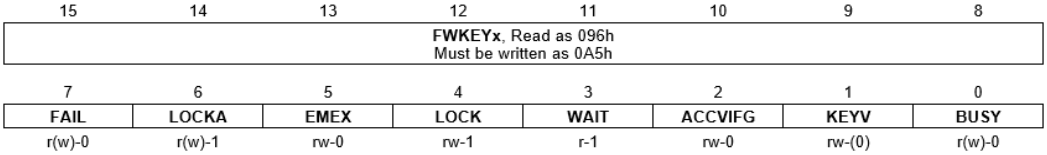
e) *WAIT、BUSY：判断擦写是否结束*



FCTL1



FCTL2



FCTL3

图 2-11 Flash 的控制寄存器

3 “DCO 自校准和检测”程序设计

程序设计过程遵循层次化设计思想，主程序由各子程序模块组成，各子程序模块再调用更次一级的模块，层层嵌套。这样设计的好处是思路清晰，主程序简洁，便于阅读和理解。本章将以**“总—分”的结构**通过系统流程图、程序流程图介绍程序的实现过程。（程序源码请查看附录，已加注详细注释。）

3.1 整体思路

系统设计流程图是是系统实现的中心思想，要想有条不紊地实现程序设计，必须要牢牢把握程序设计的整体思路。本次“DCO 自校准和检测”程序的系统流程图如图 3-1。

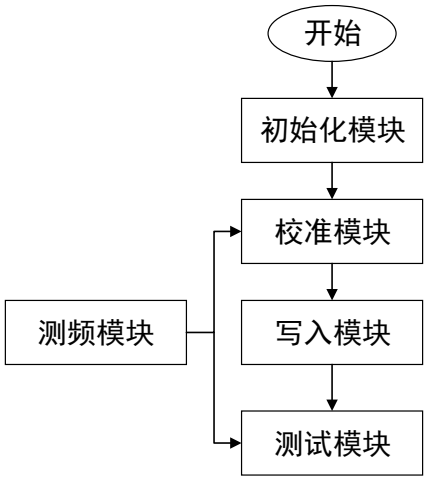


图 3-1 系统流程图

本程序主要由五大模块组成：初始化模块、校准模块、写入模块、测试模块和测频模块。其中校准、写入和测试模块体现了程序的三部分主要功能，串行级联。我们的设计分工也主要是按这个标准来分配的。另外由于测频模块在校准模块和测试模块中被多次调用，发挥了至关重要的作用，因此也独立设块。

五大模块的具体实现流程详见下文，其主要功能概述如下：

a) *初始化模块：配置时钟、I/O 口、中断定时器等参数*

b) *校准模块：测量* DCO *频率，穷举比较得到最优校准频率并保存寄存器参数；*

c) *写入模块：将寄存器参数封装并写入 InfoFlashB 段；*

d) *测试模块，读取 Flash 中寄存器参数测试校准值并循环输出供外部观测；*

e) *测频模块：使用 32.768kHz 时钟晶振作为基准测量当前* DCO *频率。*

3.2 初始化模块

看门狗初始化：看门狗关闭；

定时器 TA0 初始化：定时器设置为连续计数模式，时钟源选择 SMCLK；

I/O 口初始化：红灯亮绿灯灭，表示正在校准；P1.3 设置为输入，接上拉电阻；

中断初始化：总中断、看门狗中断使能。按键中断在测试时使能，此处不设置。

3.3 测频模块

此处需要测量高频 SMCLK 的频率，由于最高频率时钟就是 SMCLK 本身，不能使用捕获器捕获脉冲实现。此处实现思路是以 32.768 时钟晶振作为基准时钟，使用看门狗定时器定时。SMCLK 作为 TA0 时钟源，计数 SMCLK 脉冲数 Temp，通过换算关系：

$$\Delta T = \frac{1}{32768} \times 64s = \frac{1}{512}s \approx 1.95ms\ ;$$

$$Frequency = \frac{Temp}{\Delta T \times 1000} = \frac{Temp \times 512}{\Delta T \times 1000}\ ;$$

得到当前 DCO 振荡频率。因为 TA0 定时器只有 16 位，最大计数范围为 0~65535，所以看门狗定时时间不能过长。配置 32.768kHz 晶振时看门狗定时周期只能选择 1.95ms、15.6ms、0.25s、1s，除了 1.95ms，其他时间周期都会使 TA0 在计数 1~16MHz 高频 DCO 过程中溢出。这就是选择 1.95ms 档的原因。由于 32.768kHz 晶振在误差范围内可视为精准时钟（1 个月产生 1 分钟误差），可以认为测量的 DCO 频率在误差可接受的范围内是精准的。

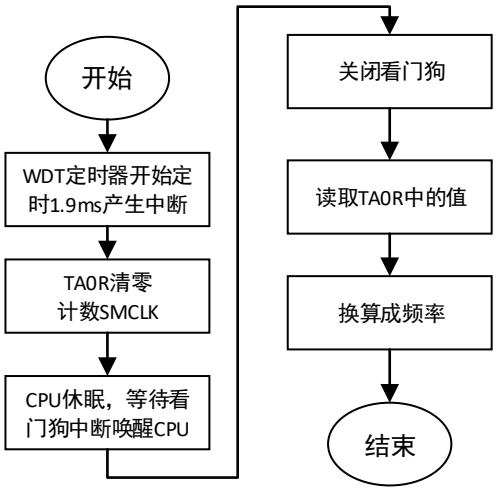


图 3-2 测频模块流程图

测频模块的流程图如图 3-2。此处不同于原先测频实验中采用捕获器数脉冲的方式，主要是运用了“CPU 休眠+看门狗定时唤醒 CPU 休眠”起到程序精准暂停 1/512s 的功能，使用定时器 TA0 设置 SMCLK 作为时钟源来计数脉冲数。模块开始时，看门狗开启，定时器清零。实测发现执行看门狗定时器配置需要 4590 个时钟周期，而配置 TA0 清零只需要 9 个时钟周期，所以，把 TA0 清零放在看门狗配置之后以减小误差。看门狗进入中断后，CPU 进入 LPM0 休眠模式，程序暂停，但是定时器 TA0 仍在工作。等待看门狗中断产生唤醒 CPU 后，读取定时器 TA0R 的值，记为定时周期内计数的 DCO 脉冲数。最后再通过换算关系得到 DCO 频率（单位是 kHz）。

3.4 校准模块

校准模块是本实验的核心模块。校准模块的实现思路是一一测量 DCO 能够提供的所有频率，分别与待校准频率比较，最终找到当前环境下最精准的校准参数。

需要注意的是，DCO 可以提供 112*32+16=3600 个频率。为什么不是 128*32 呢？其中当 DCOx=7 时，因为 MODx 不被采用。其中 32 个小数频率只能在纯净频率[1]的基础上加频率，所以我们应取比带校准频率小且最接近待校准频率的纯净频率作为最优纯净频率。在寻找最优小数频率时只需要找最接近待校准频率的即可。

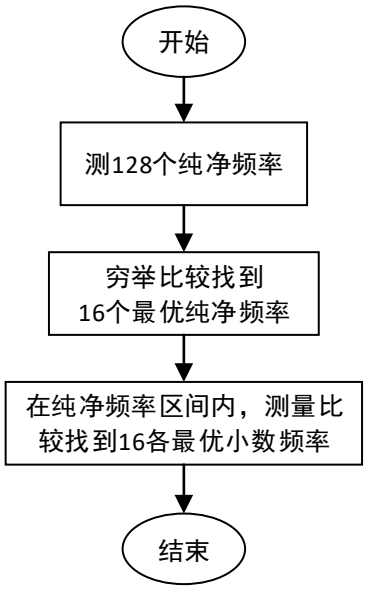


图 3-3 校准模块的总流程框图

校准模块的流程框图 3-3，先穷举测量纯净频率，再将待校准频率与纯净频率一一比较找到最优纯净频率。进而在纯净频率到下一个纯净频率的范围区间寻找最优小数频率。测量小数频率，步骤类似。比较小数频率与待校准频率，保存差值最小纪录，再比较，再保存，直到找到最优小数频率。

[1] 纯净频率：当 MODx=0 时输出的频率称为纯净频率。

3.4.1 寻找最优纯净频率

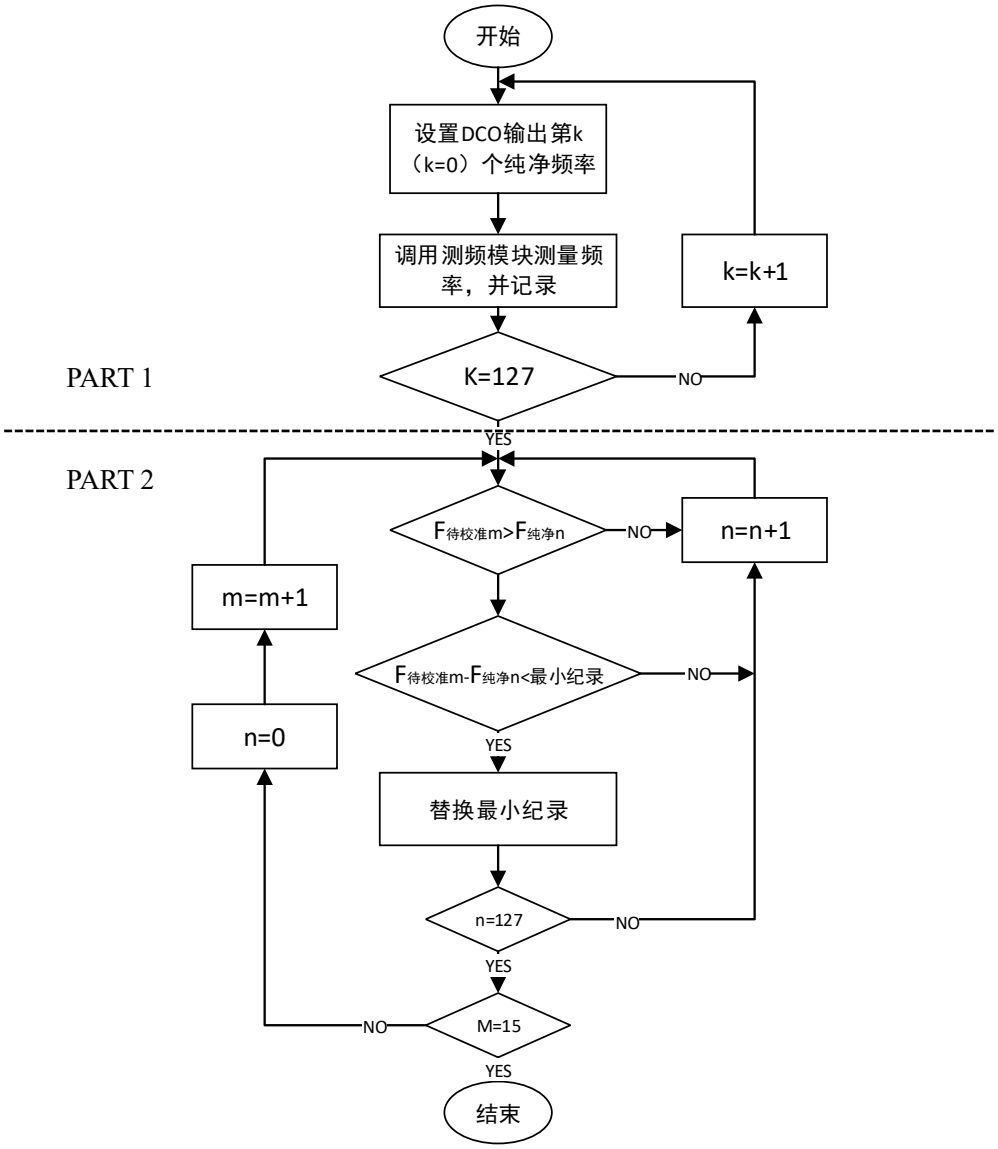


图 3-4 寻找纯净频率模块设计流程图

寻找最优纯净频率流程图如图 3-4。本模块由两部分组成，一是测量 128 个纯净频率的实际频率，二是穷举比较得到最优纯净频率。第一部分循环调用测频模块 128 次，得到所有纯净频率保存到数组中。第二部分按比较规则——最优纯净频率小于但最接近于待校准频率——将 16 个待校准频率依次和 128 个纯净频率比较，得到相应的 16 个最优纯净频率。

3.4.2 寻找最优小数频率

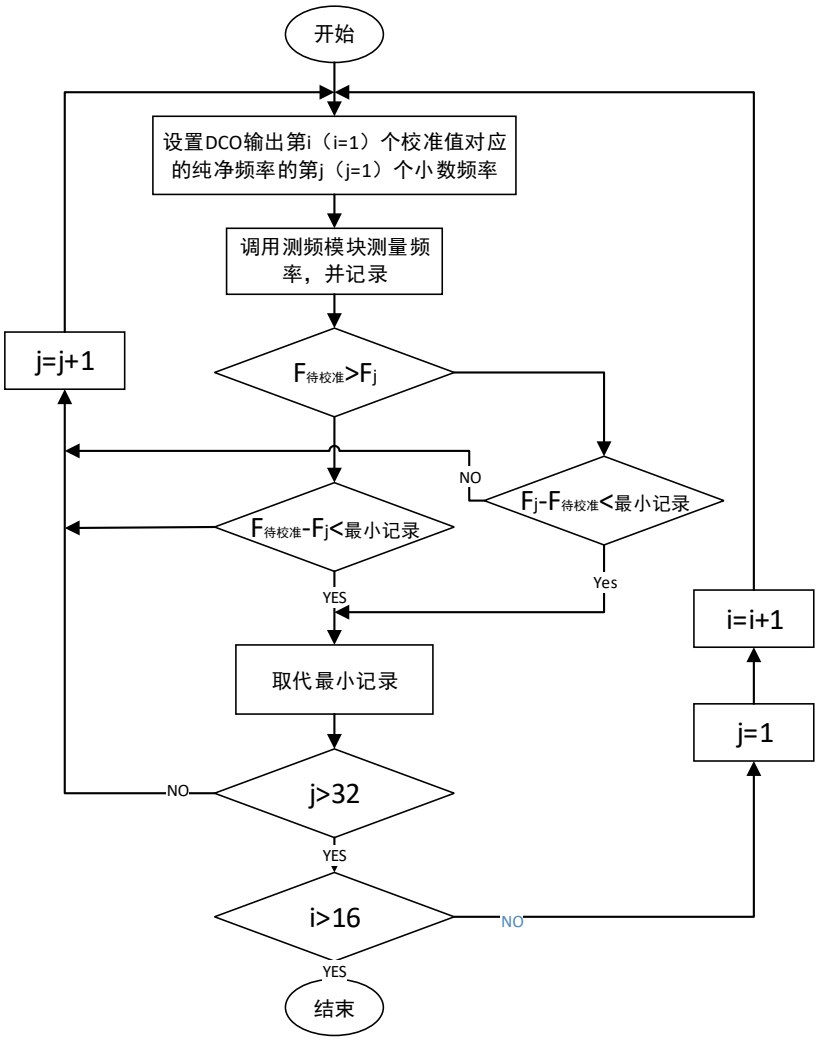


图 3-5 寻找小数频率模块设计流程图

寻找最优小数频率流程图如图 3-5。与寻找最优纯净频率模块有所不同，本模块一次循环完成测频、比较两个功能，一共需要跑 32*16 次循环得到 16 个对应的小数频率。每次比较过程中取小数频率与待校准频率差的绝对值，通过判断大小后调整减数和被减数顺序实现。将差值与中间最小记录比较，如果小于最小记录，则替代最小记录。直到 32 次比较结束，得到最接近带校准频率的小数频率参数。

3.5 写入模块

本模块把之前校验后得到的寄存器配置参数存到 Flash 中，以后在同一实验环境下可以直接通过调用便捷地得到校准后的 DCO 频率。在写入 Flash 前，需要把参数按一定格式封装。这里仿照 InfoFlashA 段(0x10fe)按寄存器顺序将校验参数“1000_RSELx_DCOx_MODx”封装成字,其中高字节“1000_RSELx”为寄存器 BCSCTL1 参数，低字节“DCOx_MODx”为寄存器 DCOCTL 配置参数。

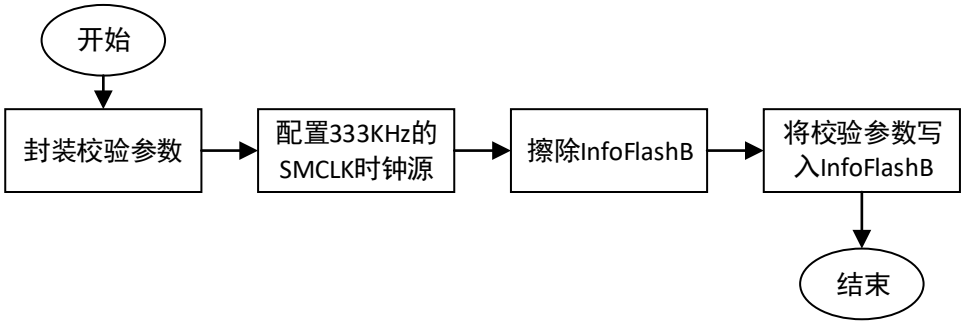


图 3-6 写入模块流程框图

需要注意的是，在擦出和写入 Flash 时，对频率有一定的要求：频率范围在 257kHz 到大约 476kHz 的范围内。本程序通过调用出场校准值配置 DCO 为 1MHz，再对其进行 3 分频（通过系统函数 Flash_Init（）可以自动分频）得到 333kHz 的时钟。另外，由于存储器本身的性质，Flash 只能将 1 写成 0，不能将 0 写成 1，所以在写入之前必须先擦出 Flash，这样才能确保写入的数据准确无误。

3.6 测试模块

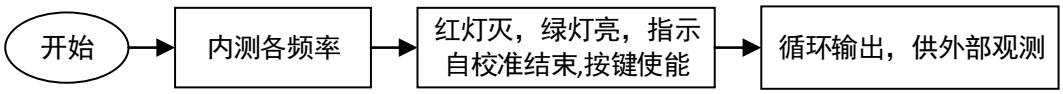


图 3-7 测试模块流程框图

测试模块分为两部分，内测模式和外测模式。在内测模式，程序自动循环读取写入 Flash 中的 16 个校准值并分别测量 16 个校准频率的实际频率，存入数组后暂停程序可查看变量读取校准后的真实频率值当内测模块结束，也就是程序自校准全过程结束，红灯灭，绿灯亮，指示自校准过程结束，按键可以控制循环输出频率。此时在 P1.4 口可以按 1MHz、2MHz…… 16MHz 的顺序依次观测到 16 个校准后的频率。

3.6.1 内测模块流程

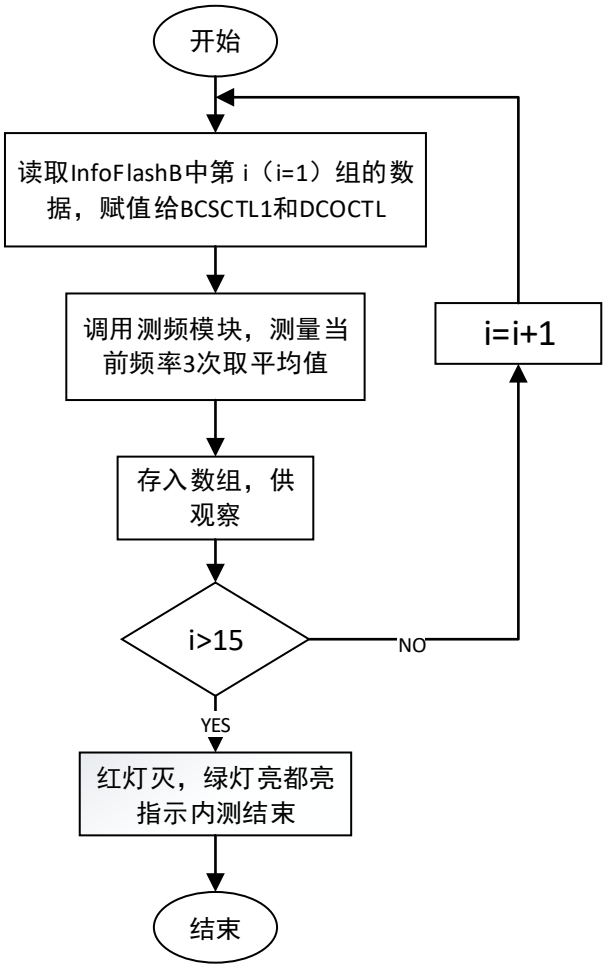


图 3-8 内测模式流程图

自测模式中循环调用测频模块测量 16 组校准频率。调用的方法如下：

```
#define CALDCO_MHz(x) *(unsigned char*)(0x1080+(x-1)*2)
#define CALBC1_MHz(x) *(unsigned char*)(0x1080+(x-1)*2+1)
……
DCOCTL=CALDCO_MHz(j+1);
BCSCTL1=CALBC1_MHz(j+1);
```

CALDCO_MHz(x)表示读取写入数组的低字节，0~4 位存储 MODx 参数，5~7 位存储 DCOx 参数；CALBC1_MHz(x)表示读取写入的高字节，0~3 位存储 RSELx 参数。

当内测结束，也就是系统自动完成的任务已经完全结束，指示系统工作的红灯熄灭，绿灯点亮，指示可以操作按键控制频率切换，进入循环输出模式。

3.6.2 循环输出模式

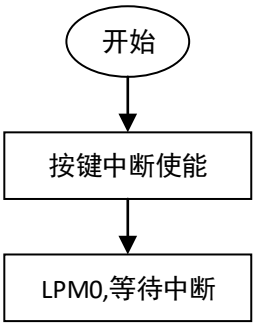


图 3-9 循环输出模式流程图

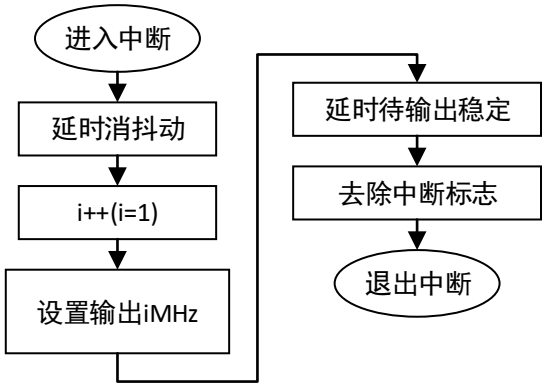


图 3-10 按键中断响应流程图

循环输出模块程序是简单的由按键引起中断唤醒 CPU 执行中断程序。进入中断响应程序后先消抖动，否则会出现按一下改变多次频率的现象。设置 DCO 输出频率的具体过程是读取 Flash 中的参数再赋值给 DCOCTL、BCSCTL1 设置寄存器。CPU 退出中断后再进入休眠模式 LPM0，等待下次按键激活中断转换频率。此时 SMCLK 激活，此时允许输出 DCO 频率，可以从 P1.4 口观测。

4 DCO 校验实验

4.1 实验内容

1) 查看校准后得到的 16 个 DCO 频率，将其与理论值、出厂校验值进行比较。
2) 用示波器观察输出波形，观察混频现象。
3) 查看程序变量与寄存器数据。查看 128 个纯净频率值，绘制 RSELx-DCOx-频率图像；查看 Flash 信息，检验数组是否正确写入 Flash 中，比较写入的 16 位校验信息与出场预设值。

4.2 实验数据与分析

4.2.1 内测自校准频率与理论值的比较

我们设计的内部频率计利用的是外部 32768Hz 时钟源，准确性高，测得的结果可认为是实际频率。程序运行后，在看到绿灯亮后，说明程序校验完成，暂停程序，在 Expression 栏中查看 F_Test 数组的值。如图 4-1 所示，F_Test 记录的值为通过芯片内部频率计测得的校验后实际频率。

Expression	Type	Value	Address
F_Test	unsigned int[16]	[1000,1986,2995,4044,4...	0x0300
[0]	unsigned int	1000	0x0300
[1]	unsigned int	1986	0x0302
[2]	unsigned int	2995	0x0304
[3]	unsigned int	4044	0x0306
[4]	unsigned int	4967	0x0308
[5]	unsigned int	6012	0x030A
[6]	unsigned int	6949	0x030C
[7]	unsigned int	8023	0x030E
[8]	unsigned int	9010	0x0310
[9]	unsigned int	10055	0x0312
[10]	unsigned int	10881	0x0314
[11]	unsigned int	12005	0x0316
[12]	unsigned int	13028	0x0318
[13]	unsigned int	13964	0x031A
[14]	unsigned int	14703	0x031C
[15]	unsigned int	16004	0x031E

图 4-1 F_Test 数组记录的校准后 1-16MHzDCO 的内测频率

整理并计算相对误差结果如下表。

表 4-1 内测频率与理论值比较

理论值/kHz	内测频率/kHz	相对误差
1000	1002	0.20%
2000	1982	0.90%
3000	2996	0.13%
4000	4001	0.03%
5000	4962	0.76%
6000	6067	1.12%
7000	6945	0.79%
8000	8015	0.19%
9000	9002	0.02%
10000	10048	0.48%
11000	10875	1.14%
12000	11999	0.01%
13000	13023	0.18%
14000	13962	0.27%
15000	14697	2.02%
16000	16078	0.49%

由比较结果可见，内测自校准频率较理论值的误差相对较小，在我们所选取的 1-16MHz 的 16 个频率点中平均误差等于 0.54％，最大不超过 2.02％，基本都小于 1.00％，其中 15MHz 的误差特别大，达到了 2.02%（这主要是由于这个频率点正好处于 DCOx=7，MODx 不起做作用导致）；4MHz、9MHz、12MHz 的精度优于 0.1％。说明我们的校准结果理想，一定程度上克服了 DCO 时钟频率不准确的问题。但另一方面，自校准并不能做到使设置的 DCO 频率完全等于理论频率值，DCO 时钟频率的调节从机理上存在不可避免的误差。

4.2.2 内测自校准频率与出厂校准频率的比较

为了检验 DCO 自校准对时钟频率准确性的改善，将其与出厂校准频率相比较。通过设置：

BCSCTL1=CALBC1_xMHZ;

DCO*CTL=CAL*DCO*_xMHZ;*

将 x 设置为 1,8,12,16，通过调用测频模块，测试出厂校准频率的实际大小，计算与理论值的误差，并在各个频率点同自校准频率比较，结果如表 4-2 所示。

表 4-2 内测自校准频率与出厂校准频率比较

频率点	1MHz	8MHz	12MHz	16MHz
理论值/kHz	1000	8000	12000	16000
内测自校准值/kHz	1002	8015	11999	16078
出厂校准值/kHz	1013	7960	11913	15863
内测自校准值误差	0.20%	0.19%	0.01%	0.49%
出厂校准值误差	1.30%	0.50%	0.73%	0.86%

测试结果说明了经过程序自校准后的频率值更为精确，与理论值的误差明显小于出厂校准频率，提高了芯片 DCO 的准确性。可见 DCO 自校准为获取较为精确的时钟信号提供了一条更有效的途径。当 32768Hz 精确外部时钟信号的频率范围不能满足设计需要时，自校准 DCO 频率便为此提供了解决办法。

4.2.3 示波器观测输出波形

用示波器观察到的输出波形如图 4-2 所示，可见除了正中央触发位置波形没有重影，其他位置波形均有重影，且越远离中央位置，重影越明显。

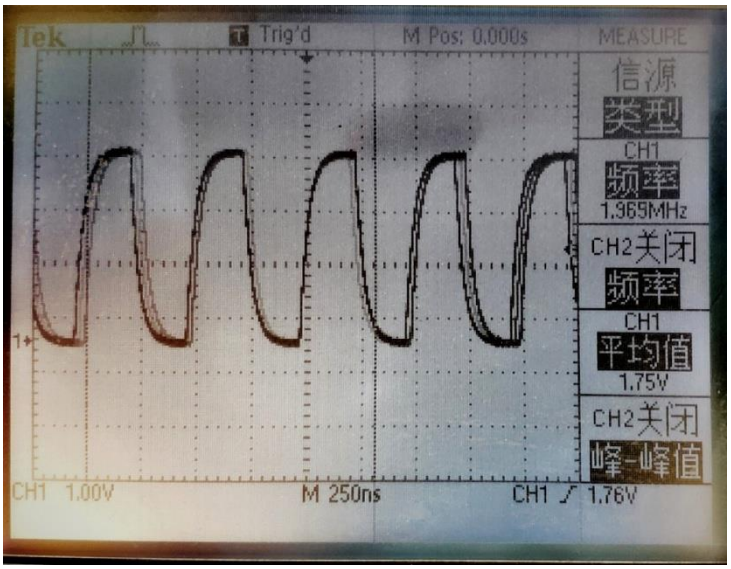


图 4-2 示波器观察混频时钟抖动

查找数据手册及其他相关资料，5 位 MODx 的作用是，在由 DCOX 位选择的频率和下一个由 DCOX+1 设置的更高的频率之间的切换并设置两者的占比，即我们依靠设置 MODx 混频得到期望的精确频率。实际上是两种纯净频率轮流输出来实现微调，原理如图 4-3 所示（MODx 设 16），因此存在频率抖动。

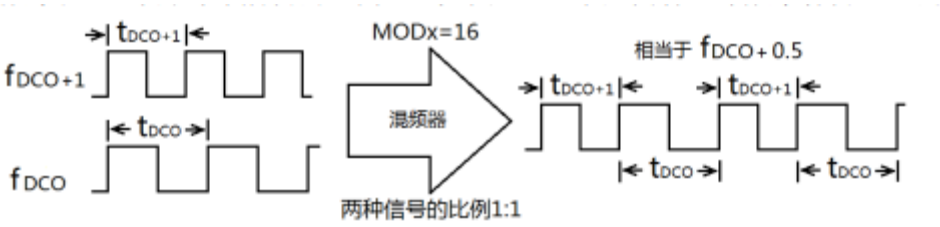


图 4-3 混频抖动原理

4.2.4 查看 CCS 变量、寄存器数据

1）查看 128 个“纯净”频率值

在测频模块，将 MODx 固定设为 0 时，RSELx 和 DCOx128 种组合的 DCO 频率，查看用程序测得的 128 个纯净频率值，截取了部分数据如图 4-4 所示。

Expression	Type	Value	Address
PURE_FREQ	unsigned int[128]	[98,105,111,119,128...]	0x0200
[0 ... 99]			
[0]	unsigned int	98	0x0200
[1]	unsigned int	105	0x0202
[2]	unsigned int	111	0x0204
[3]	unsigned int	119	0x0206
[4]	unsigned int	128	0x0208
[5]	unsigned int	139	0x020A
[6]	unsigned int	152	0x020C
[7]	unsigned int	167	0x020E
[8]	unsigned int	121	0x0210
[9]	unsigned int	128	0x0212
[10]	unsigned int	136	0x0214
[11]	unsigned int	146	0x0216
•••			
[123]	unsigned int	14477	0x02F6
[124]	unsigned int	15510	0x02F8
[125]	unsigned int	16595	0x02FA
[126]	unsigned int	18014	0x02FC
[127]	unsigned int	19944	0x02FE

图 4-4 程序自测的 128 个纯净频率值

由此可见，当 RSELx 和 DCOx 都设为 0 时，DCO 的最低频率为 98kHz；当 RSELx 和 DCOx 都设为最大值时，DCO 的最高频率为 19.944MHz。我们便知道 DCO 的频率范围。

另外根据得到的实验数据，绘制如下图 4-5（RSEL=0,1,2,3,4），观察 RSEL 的阶跃，可见 DCO 的频率由 RSELx、DCOx 的取值共同决定，128 个纯净频率各不相同，而不同 RSELx 取值情况下 DCO 的频率范围存在重叠，并非 RSELx 越大 DCO 频率就越大，实验结果与用户手册及数据手册相符。

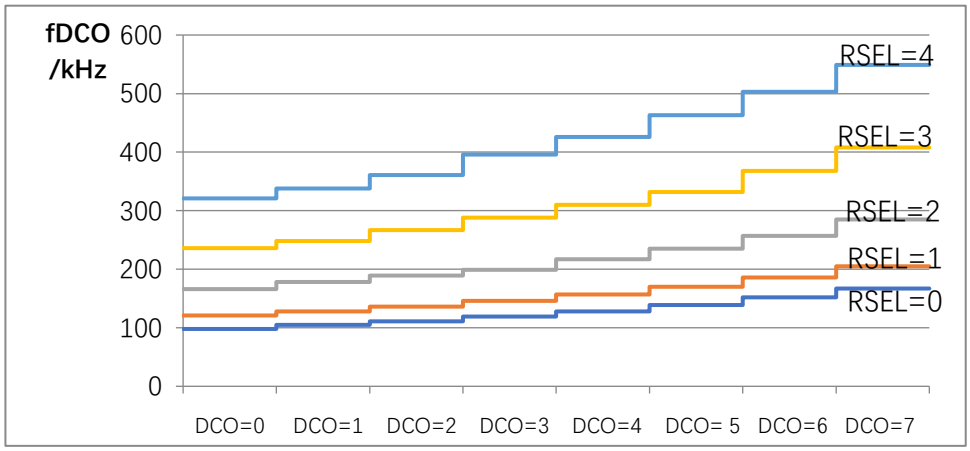


图 4-5 不同 RSELx、DCOx 取值条件下 DCO 频率

2）查看寄存器校准参数数组

在校准模块，查看经程序校准后 PURE_WINNER 数组中对应 RSELx 和 DCOx 的值，以及 MODx_WINNER 数组中得到的 MODx 对应值。如图 4-6、图 4-7 所示，PURE_WINNER 数组中每个值的最高位均为 0，RSELx、DCOx 共占 7 位， 4 位 RSELx 对应 3-7 位，3 位 DCOx 则对应 0-2 位；由 MODx_WINNER 的值可见 MODx 为 5 位数据，与已知相符。可见程序正确实现了校准功能，即选择出最接近待校验频率的 RSELx、DCOx、MODx 参数值。

Variables | Expressions | Registers | Memory Browser

Expression	Type	Value	Address
[100 ... 127]			
PURE_WINNER	unsigned char[16]	[00111010b ':',01000110b 'F',...	0x0350
[0]	unsigned char	00111010b ':'	0x0350
[1]	unsigned char	01000110b 'F'	0x0351
[2]	unsigned char	01010010b 'R'	0x0352
[3]	unsigned char	01010110b 'V'	0x0353
[4]	unsigned char	01100001b 'a'	0x0354
[5]	unsigned char	01100100b 'd'	0x0355
[6]	unsigned char	01101010b 'j'	0x0356
[7]	unsigned char	01101100b 'l'	0x0357
[8]	unsigned char	01110000b 'p'	0x0358
[9]	unsigned char	01110010b 'r'	0x0359
[10]	unsigned char	01110011b 's'	0x035A
[11]	unsigned char	01110100b 't'	0x035B
[12]	unsigned char	01111001b 'y'	0x035C
[13]	unsigned char	01111010b 'z'	0x035D
[14]	unsigned char	01110111b 'w'	0x035E
[15]	unsigned char	01111100b '|'	0x035F

图 4-6 PURE_WINNER 数组值

Variables | Expressions | Registers | Memory Browser

Expression	Type	Value	Address
MODx_WINNER	unsigned char[16]	[00000010b '\x02',00000111b...	0x0340
[0]	unsigned char	00000010b '\x02'	0x0340
[1]	unsigned char	00000111b '\x07'	0x0341
[2]	unsigned char	00001011b '\x0b'	0x0342
[3]	unsigned char	00000011b '\x03'	0x0343
[4]	unsigned char	00010101b '\x15'	0x0344
[5]	unsigned char	00001111b '\x0f'	0x0345
[6]	unsigned char	00000110b '\x06'	0x0346
[7]	unsigned char	00001000b '\x08'	0x0347
[8]	unsigned char	00000111b '\x07'	0x0348
[9]	unsigned char	00000010b '\x02'	0x0349
[10]	unsigned char	00001100b '\x0c'	0x034A
[11]	unsigned char	00010110b '\x16'	0x034B
[12]	unsigned char	00001010b '\x0a'	0x034C
[13]	unsigned char	00010000b '\x10'	0x034D
[14]	unsigned char	00000000b '\x00'	0x034E
[15]	unsigned char	00010011b '\x13'	0x034F

图 4-7 MODx_WINNER 数组值

3）查看校验数组是否正确写入 InfoFlashB 段

在写入模块，程序中我们需要将经过校准得到的 16 位 RSELx_DCOx_MODx 校验数组存入 InfoFlashB 段，这要求正确调节擦写 Flash 的频率，指定正确的改写地址并进行正确的擦写操作等。

为了检验 16 位校验数组是否正确写入，就需要查看准备写入 InfoFlashB 段的数据 InfoFlash_Data[]和实际 InfoFlashB 段的数据是否一致。程序运行后可得 InfoFlash_Data[]数据如图 4-8 所示，FlashB 中存储数据如图 4-9。

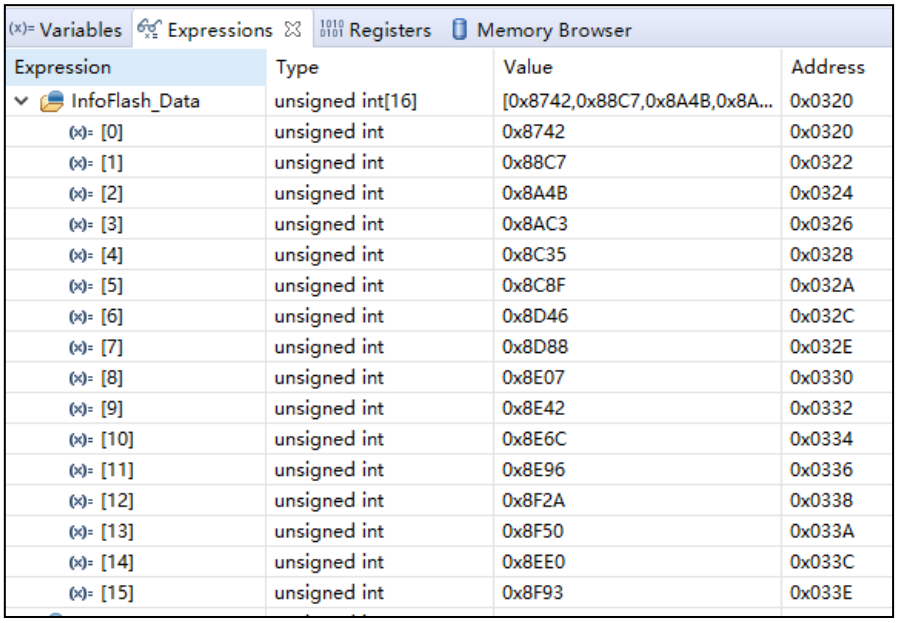


图 4-8 InfoFlash_Data[]值

在 Memory Browser 的输入框中输入 InfoFlashB 的首地址 0x1080，查看到内部存储的数据如图 4-9 所示。

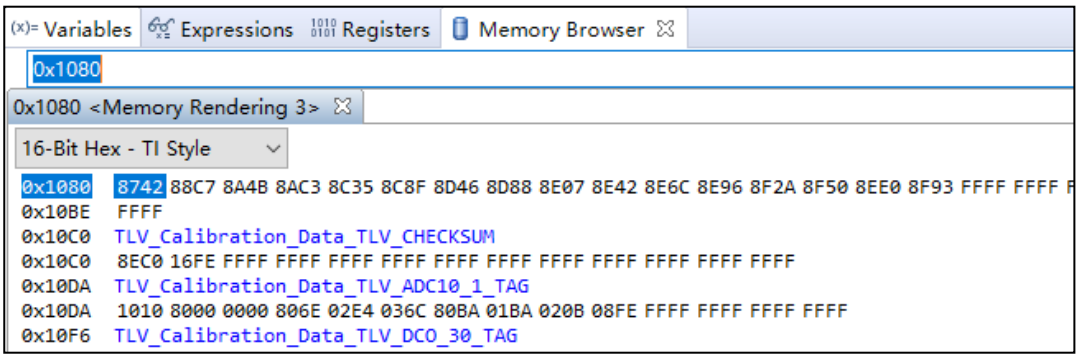


图 4-9 InfoFlashB 内存数据

比较图 4-8 中查看到的 InfoFlash_Data 数组的值与 Flash 中存储的数据一致，说明经过计算得到的 InfoFlash_Data 数组成功存入了 InfoFlashB 前 16 字中，写入程序正确实现，可进行后续的读取及频率输出。

4）查看比较 16 位校验数组与出厂校验值

程序运行后，在 Expression 栏中查看 InfoFlash_Data 数组的值，如图 4-6 所示。在 Register 栏中查看 Calibration_Data 的值如图 4-10，可得到 1/8/12/16MHz 的出厂校验参数，记录并与自校准得到的校验值比较，如表 4-3 所示。

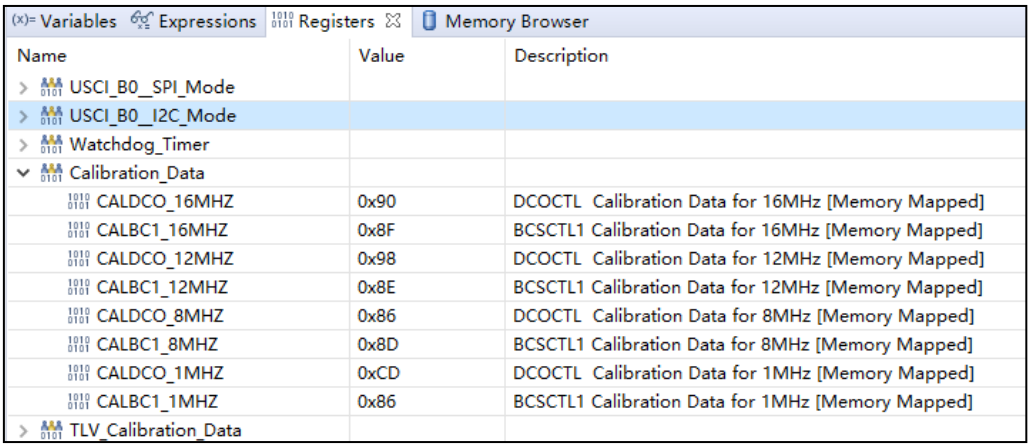


图 4-10 Calibration_Data 中查看到的出厂校验值

表 4-3 自校准得到的校验值与出厂校验值比较

频率点	1MHz	8MHz	12MHz	16MHz
出厂校验值	0x86CD	0x8D86	0x8E98	0x8F90
自校准校验值	0x8742	0x8D88	0x8E96	0x8F93

经过程序自校准的校验值与原出厂校验值并不相同，说明程序对校验值进行了重新的计算、选择，从而得到了更为准确的频率。

4.3 实验结论

1) 实验结果说明了程序成功实现了 DCO 的自校准功能，与设定的期望频率相差极小，与出厂校准频率相比，自校准得到的频率准确度更高。DCO 自校准能在较大的频率范围内实现较为精确的校准。但是 DCO 时钟频率的调节从机理上存在不可避免的误差，自校准不能做到使设置的 DCO 频率完全等于理论频率值。

2) 128 个“纯净”频率由 RSELx、DCOx 的取值共同决定。在 MODx=0 的前提下：当 RSELx 和 DCOx 都设为 0 时，DCO 的最低频率为 98kHz；当 RSELx 和 DCOx 都

设为最大值时，DCO 的最高频率为 19.944MHz。且由 RSELx 阶跃图中可见，在不同 RSELx 取值时 DCO 频率的范围存在重叠，并非 RSELx 越大，输出频率越高。

3) 微调 MODx 通过引入混频得到更精确频率，由于两种纯净频率轮流输出来实现微调所以用示波器观察输出波形会出现重影的现象。

4) 程序通过校准模块能够正确实现校验参数的设置，选择出最接近待校准频率的 RSELx、DCOx、MODx 参数值记入相应的数组。校验参数能够正确写入 FlashB 段，从而可进行后续的读取及频率输出。自校验得到的 16 位校验数组与出厂校验值不同。

5 设计与实验过程中的问题与改进

5.1 15MHz 误差研究

观察内部频率计测得的自校验后 DCO 频率发现 15MHz 频率点的误差明显大于其他频率点，进行多次测试，结果均发现 15MHz 频率点误差要远大于其他频率点，在 2%左右波动，而在其他 15 个频率点，自校验后 DCO 频率误差均能做到极小，这让我们感到疑惑，决定对这一现象进行深入研究。

经过分析，在其他 15 个频率点的自校验频率准确度较为满意，程序功能应该不存在问题，那么就从查看程序变量入手找问题的根源。

首先我们怀疑程序在 15MHz 频率点的校验参数设置是否正确，先检验 RSELx、DCOx 的设置。在 Expression 栏中查看 PURE_WINNER 数组选择的纯净频率值序号，以及在 PURE_FREQ 数组中查看 128 个纯净频率值大小。理论上 PURE_WINNER 数组应该要选择出频率小于且最接近待校准频率的纯净频率值序号，经过我们判断，PURE_WINNER 数组选择正确。排除了 RSELx 和 DCOx 的设置问题。

进一步查看 MODx_WINNER 数组选择的 MODx 值，多次运行自校准程序发现 15MHz 频率点对应的 MODx_WINNER[14]总是为 0，也就意味着 MODx=0，没有进行混频。更换实验板，反复实验，结果都是显示 MODx=0。

于是我们另外编程，单独调用 15MHz 时的纯净频率，通过改变 MODx 的值，发现输出的 32 个频率都是在 MODx=0 前后波动，说明 MODx 没有发挥作用。于是我们意识到了一个之前应该考虑到但是被忽略的问题。分析原理，数据手册中说明了在 DCOx=7 时，MOD 不被采用，也就是不混频直接输出。查看 15MHz 频率点选择的纯净频率（数组 119 位），并与数组中下一位的纯净频率比较，如图 5-1。

[118]	unsigned	13469	0x02EC
[119]	unsigned	14624	0x02EE
[120]	unsigned	12200	0x02F0
[121]	unsigned	12857	0x02F2

图 5-1 第 119、120 位纯净频率值

发现第 120 个纯净频率反而比第 119 个纯净频率小。之后我们回过头来计算， 119 对应 RSEL=14，DCO=7，数组 120 位对应 DCO=0，RSEL=15。即 15MHz 频率点选择的纯净频率恰好对应 DCO=7，MOD 不被采用。实验结果与理论相符。说明了是 DCOx、RSELx 设置的固有误差导致了在 15MHz 频率点出现无法克服的误差。

经过我们的分析及实验，说明了 15MHz 频率点恰好处于 DCOx=7 时，MOD 不被采用，存在由芯片自身造成的不可避免的误差。

5.2 高频频率测量的实现

设计初期，最困扰我们的难题就是如何测量高频 DCO 频率。先前第二次实验的时候我们也测过频率，那时我们测的是低频信号的频率，是通过捕获器或者自己写程序识别上下沿的方式来计数脉冲数。这种方式极大地依赖于高频的系统时钟 MCLK 检测是否产生脉冲，如果系统时钟频率没有远大于待检测频率的话，将会有脉冲没有被检测到，导致无法准确测频。所以，既然我们要测高频信号，最高 16MHz，而根据用户手册，MSP430 可接受的外部时钟频率为 400kHz~16MHz，必然不能用这一种方法。

于是我们另辟佳径，开始想有没有其他实现方式，能不能避开高频时钟捕获低频时钟。于是我们就想到了计数器本身就没有用到捕获，却也实现准确计数的功能。我们开始思考能不能定时器 TA0 来计数 DCO 的脉冲数，用 TA1 或者 WDT 定时器来精准定时。本例因为之前从来没有使用过 WDT 定时器，所以决定使用 WDT 来定时。之后我们查阅了网上对看门狗定时器的使用例程，学习了在主程序中打开看门狗定时和中断，CPU 休眠，再通过产生中断唤醒 CPU 的方法来精准定时，成功实现了高频频率测量的功能。

5.3 中断嵌套无法正常实现

在按键切换频率循环输出的模块，我们最初的想法是在按键中断中嵌套频率测量模块，每次按按键既可以改变频率也可以在屏幕上显示当前频率值，即在按键中断中再嵌套 WDT 中断。编译没问题，但是调试过程中，程序出现了许多不符合逻辑的错误，比如变量 FREQ_NUM 在一次循环中只加 1，但是我们会测到它在第一次循环中就加到了 10。而程序除了循环内给它赋值外，没有再调用过这个变量。除此之外程序会出现跑飞的现象,甚至有几次在按键循环过程中还改变了校准模块中的 PURE_FREQ[]数组的值。我们为这个问题讨论了许久，虽然我们也可以花时间去解决这个问题，但就实验任务而言，我们没必要和它死磕到底。我们将测频模块与按键模块分离设计，回避了这个问题。

参考文献

[1] [作者不详].MSP430G2x53、MSP430G2x13 混合信号微控制器 数据表 (Rev. G) [M]. 出版地不详:出版者不详,2012

[2] [作者不详]. MSP430x2xx 系列用户指南 (Rev. I) [M].出版地不详:出版者不详,2012

[3] 傅强,杨艳 Launchpad 口袋实验平台(指导书)[M].青岛:青岛大学,2013

附录 1 程序源码

```
/*==================DCO 自校准和检测程序描述==============
 * 程序名称：DCO 自校准和检测
 *
 * 程序功能：DCO 自校准 1、2.....16MHz 十六个频率，将结果存入 InfoFlashB 段中。
 *      依次调用十六个校准频率，内测频率后通过按键控制依次输出，供外部观测。
 *
 * 程序说明：
 *  程序分为三大模块：测频模块、校准模块、写入模块、检测模块。
 *  测频模块：重要的小模块，使用 32.768KHz 作为精准时钟基准测量高频 DCO 信号的频率。
 *  校准模块：调用测频模块测量 DCO 真实振荡频率，穷举比较得到最接近理想值的校准值。
 *  写入模块：将校准参数按出厂校准格式封装，存入 InfoFlashB 段中。
 *  测试模块：读取 InfoFlashB 中数据，循环输出 DCO 校准频率，调用测频模块测量其频率。
 *      通过按键控制循环输出各校准频率，供外部观测。
 *  本程序中频率单位为 KHz。
 *
 * 作    者：陈钱牛(2160400096)：主要负责检测模块、终期调试
 *          王  玮 (2160400093)：主要负责写入模块、注释说明
 *          梁永回(2160400075)：主要负责校准模块、测频模块
 ************************************************************/

#include <msp430.h>
#include <flash.h>
//-----读取 InfoFlashB 段第 x 个数据单元中低八位数据-----
#define CALDCO_MHz(x) *(unsigned char*)(0x1080+(x-1)*2)
//-----读取 InfoFlashB 段第 x 个数据单元中高八位数据-----
#define CALBC1_MHz(x) *(unsigned char*)(0x1080+(x-1)*2+1)
#define CAL_NUM 16    //总待校准频率数

const unsigned int CAL_FREQ[CAL_NUM]=  //待校准频率
{
  1000, 2000, 3000, 4000,
  5000, 6000, 7000, 8000,
  9000,10000,11000,12000,
 13000,14000,15000,16000
};

unsigned int PURE_FREQ[128]={0};              //存放 128 个纯净频率
unsigned char PURE_WINNER[CAL_NUM]={0};       //存放小于且最接近校准值的 RSELx、DCOx 参数
unsigned char MODx_WINNER[CAL_NUM]={0};       //存放最接近校准值的 MODx 参数
unsigned int InfoFlash_Data[CAL_NUM]={0};     //存放封装后的校准参数
unsigned int F_TEST[CAL_NUM]={0};             //内测结果，以便观察
unsigned int FREQ_NUM=1;                      //记录当前调用的频率

//-----函数列表，详细说明见函数定义前说明性注释-----

void Inital();                                //初始化模块
unsigned int FREQ_MEASURE();                  //频率测量模块
void FIND_WINNER();                           //校准模块
void WRITE_FLASH();                           //写入模块
void FREQ_TEST();                             //测频模块
void DCO_RANK_SET                             //将纯净频率按寄存器数值顺序排序
   (unsigned char NUM,unsigned char MODtemp);
```

```
53.     void FIND_PURE_WINNER();       //找到最优纯净频率
54.     void FIND_MODx_WINNER();     //找到最优小数频率
55.     void CAL_FLASH_DATA();         //按出厂格式封装校准参数
56.     void WRITE_InfoFlashB();       //将校准参数写入 InfoFlashB 段
57.     void DCO_TEST_MEASURE();     //测量校准后的 DCO 频率
58.
59.     /*****************************************************************************************************
60.      * 名    称：main()
61.      * 功    能：主函数
62.      * 入口参数：无
63.      * 出口参数：无
64.      * 说    明：主函数调用四个模块，具体功能在四个模块中分层实现。
65.      *       本程序设计思路是自上向下分层分模块设计。
66.      * 范    例：无
67.      *****************************************************************************************************/
68.     int main(void)
69.     {
70.       Inital();                    //初始化模块
71.       FIND_WINNER();           //校准模块
72.       WRITE_FLASH();           //写入模块
73.       FREQ_TEST();             //测频模块（死循环）
74.     }
75.
76.     /*****************************************************************************************************
77.      * 名    称：Inital()
78.      * 功    能：程序初始化
79.      * 说    明：包括关闭看门狗、中断初始化、I/O 口初始化、定时器初始化
80.      *       中断初始化：总中断、看门狗定时中断打开。
81.      *       I/O 口初始化：P1.0、P1.6 设置成输出模式，
82.      *         P1.3 设置为输入模式等待，在测试模块中打开才 P1.3 中断。
83.      *       定时器初始化：定时器 TA0 接 SMCLK 即 DCOCLK 时钟源，在准确时间间隔内
84.      *         读取 TA0 计数差，可换算得到 DCOCLK 的频率
85.      *****************************************************************************************************/
86.     void Inital()
87.     {
88.       WDTCTL = WDTPW + WDTHOLD;       //关狗
89.       //-----在 P1.4 上输出 SMCLK，这样可以全程用示波器和频率计观测 DCO----
90.       P1SEL |= BIT4;
91.       P1DIR |= BIT4;
92.       //-----P1.3 按键切换 DCO 频率-----
93.       P1REN|=BIT3;
94.       P1OUT|=BIT3;
95.       //-----红灯亮绿灯灭表示正在校验-----
96.       P1DIR |=BIT0+BIT6;
97.       P1OUT |= BIT0;
98.       P1OUT &=~BIT6;
99.       //-----初始化 TA，开启 WDT、P1.3 中断使能-----
100.      TA0CTL = TASSEL_2+MC_2+TACLR;     //SMCLK 时钟源，增计数开始
101.      IE1|=WDTIE;                                 //开启 WDT 中断
102.      _EINT();                                  //开启总中断
103.    }
104.
105.    /*****************************************************************************************************
106.     * 名    称：FREQ_MEASURE()
107.     * 功    能：测量并返回当前 DCOCLK 的频率。
108.     * 返 回 值：当前参数下的 DCO 频率
109.     * 说    明：看门狗定时器定时间隔选择有限，使用 32.768KHz 晶振最短可以定时 1.9ms。
110.     *       DCO 频率为 20MHz 时，TA0R 计数 38000，没有溢出。若选用大一级时间
```

```
111.     *      间隔 15.2ms，TA0R 计数 304000 次，溢出。
112.     ************************************************************************************************/
113.    unsigned int FREQ_MEASURE()   //频率测量模块
114.    {
115.      unsigned long Freq_Temp;
116.      TA0CTL |=TACLR;                    //计数开始
117.      WDTCTL=WDT_ADLY_1_9;               //看门狗定时器 1.9ms(=1/32768*64=1/512s)后中断
118.      LPM0;                              //CPU 休眠，等待看门狗唤醒
119.      Freq_Temp = TA0R;                  //读取计数，这个值就是 DCO 在中断区间的脉冲数
120.      WDTCTL=WDTPW+WDTHOLD;              //定时结束后看门狗关闭
121.      return ((Freq_Temp)*512)/1000;     //换算得到当前 DCO 频率
122.    }
123.
124.    /************************************************************************************************
125.     * 名    称：DCO_RANK_SET()
126.     * 功    能：给 128 个纯净频率排序并根据传入参数，设定 DCO 频率
127.     * 入口参数：NUM：NUM 的格式为“RSELx_DCOx”,即 3~6 位表示 RSELx,0~2 位表示 DCOx
128.     *     MOD_temp：MODx 的参数
129.     * 说    明：改变参数后需要延时等待 DCO 频率稳定。
130.     * 范    例：DCO_RANK_SET(12,5),表示将 DCO 设定为 RSELx=1，DCOx=4，MODx=5
131.     ************************************************/
132.    void DCO_RANK_SET(unsigned char NUM,unsigned char MODtemp)
133.    {
134.      DCOCTL = ((NUM%8)<<5)+MODtemp;       //数据拆分
135.      BCSCTL1 &=~0x0F;                     //将 RSELx 置零
136.      BCSCTL1 |=NUM/8;                     //整除 8 以后剩下 RSELx
137.      __delay_cycles(15000);               //等待 DCO 频率稳定
138.    }
139.
140.    /************************************************************************************************
141.     * 名    称：FIND_WINNER()
142.     * 功    能：找到最优校准频率的寄存器参数
143.     * 说    明：校准模块分为两部分：找到最优纯净频率、找到最优小数频率
144.     ************************************************/
145.    void FIND_WINNER()      //校准模块
146.    {
147.      FIND_PURE_WINNER();   //纯净频率校验模块
148.      FIND_MODx_WINNER();   //小数频率校验模块
149.    }
150.
151.    /************************************************************************************************
152.     * 名    称：FIND_PURE_WINNER()
153.     * 功    能：找到最优纯净频率
154.     * 说    明：1、由于小数频率只能在纯净频率的基础上加，所以应该选择比理想值较小但最接近理想值的纯净频率
155.     *     2、通过穷举比较，找到最优纯净频率
156.     ************************************************/
157.    void FIND_PURE_WINNER()
158.    {
159.      unsigned char i=0;
160.      unsigned int Delta_Min=0;
161.      unsigned int Delta_Now=0;
162.      unsigned char Calibrate_Num=0;
163.      unsigned char PURE_NUM=0;
164.      //-----测 128 个频率-----
165.      for(PURE_NUM=0;PURE_NUM<128;PURE_NUM++)
166.      {
167.        DCO_RANK_SET(PURE_NUM,0);                  //依次设置 DCO 为 128 个纯净频率
168.        PURE_FREQ[PURE_NUM]=FREQ_MEASURE();        //测量得到纯净频率
```

```
    }
    //-----穷举比较，找到最优频率-----

    for(Calibrate_Num=0;Calibrate_Num<CAL_NUM;Calibrate_Num++)
      {
        Delta_Min=65534;     //第一次比较时，故意定一个大差值
        for(i=0;i<128;i++)
        {
          //-----单方向取最小差值，一定要小于预设值-----
          if(PURE_FREQ[i]<CAL_FREQ[Calibrate_Num])
            {
              Delta_Now=CAL_FREQ[Calibrate_Num]-PURE_FREQ[i];
              if (Delta_Now<Delta_Min)          //如果当前差值比当前最小差值还小时
              {
                PURE_WINNER[Calibrate_Num]=i;     //新 Winner 诞生
                Delta_Min=Delta_Now;          //当前差值取代先前最小差值
              }
            }
        }
      }
  }
  /*********************************************
   * 名    称：FIND_MOD_WINNER()
   * 功    能：找到最优小数频率
   * 说    明：1、比较差值的绝对值的大小，选择最接近理想值的小数频率
   *     2、通过穷举比较找到最优小数频率
   *********************************************/
  void FIND_MODx_WINNER()
  {
    unsigned char MOD_x = 0;
    unsigned long  Freq_Temp = 0;
    unsigned int Delta_Now=0;
    unsigned int Delta_Min=0;
    unsigned char Calibrate_Num=0;
    for(Calibrate_Num=0;Calibrate_Num<CAL_NUM;Calibrate_Num++)
    {
      Delta_Min=65530;                  //第一次比较时，故意定一个大差值
      //-----RSELx 和 DCOx 一定时，测量 MOD_x 改变时的系列频率-----
      for(MOD_x=0;MOD_x<32;MOD_x++)
      {
        DCO_RANK_SET(PURE_WINNER[Calibrate_Num],MOD_x);
        Freq_Temp=FREQ_MEASURE();
        if(Freq_Temp<CAL_FREQ[Calibrate_Num])
        {
          Delta_Now=CAL_FREQ[Calibrate_Num]-(unsigned int )Freq_Temp;
        }
        else
        {
          Delta_Now=(unsigned int )Freq_Temp-CAL_FREQ[Calibrate_Num];
        }
        if (Delta_Now<Delta_Min)              //如果当前差值比记录还小时
        {
          MODx_WINNER[Calibrate_Num]=MOD_x;     //新 Winner 诞生
          Delta_Min=Delta_Now;              //取代前记录
        }
      }
    }
  }
```

```
227.  /*************************************************************************************************
228.  * 名    称：WRITE_FLASH()
229.  * 功    能：将校准参数封装后写入 InfoFlashB 段
230.  * 说    明：写入模块包括两部分，封装校准参数和写入校准参数
231.  *************************************************************************************************/
232.  void WRITE_FLASH()
233.  {
234.    CAL_FLASH_DATA();  //按出厂格式封装校准参数
235.    WRITE_InfoFlashB(); //写入 InfoFlashB 段
236.  }
237.
238.  /*************************************************************************************************
239.   * 名    称：CAL_FLASH_DATA()
240.   * 功    能：基于已校验成功的 RSELx、DCOx 和 MODx 封装待写入 Flash 的数据
241.   * 说    明：16 位数据的格式完全仿照 FlashA 段出厂校验参数 (0x10FE~0x10F8)，
242.   *      “1000_RSELx_DCOx_MODx"
243.   *************************************************************************************************/
244.  void CAL_FLASH_DATA()
245.  {
246.    unsigned char RSELx=0;
247.    unsigned char DCOx=0;
248.    unsigned char i=0;
249.    for(i=0;i<CAL_NUM;i++)
250.    {
251.      RSELx=PURE_WINNER[i]/8;
252.      DCOx=PURE_WINNER[i]%8;
253.      //-----出厂校准参数的 16 位格式为“1000_RSELx_DCOx_MODx"-----
254.      InfoFlash_Data[i]=0x8000+(RSELx<<8)+(DCOx<<5)+MODx_WINNER[i];
255.    }
256.  }
257.
258.  /*************************************************************************************************
259.   * 名    称：WRITE_InfoFlashB()
260.   * 功    能：将封装完的 16 位数组写入 InfoFlashB 段
261.   * 说    明：初始化 Flash,擦除频率范围在 257kHz 到大约 476kHz 的范围内，由于 falsh
262.   *              只能将 1 写为 0，所以需要全部擦出后再写入
263.   *************************************************************************************************/
264.  void WRITE_InfoFlashB()
265.  {
266.    unsigned char i=0;
267.    //-----写 Flash 前 DCO 时钟一定要重新确认一遍-----
268.    BCSCTL1 = CALBC1_1MHZ;                    // 调用系统校准值，设置 DCO 频率为 1MHz
269.    DCOCTL = CALDCO_1MHZ;
270.    Flash_Init(3,'B');                        // 初始化 Flash,擦除频率范围在 257kHz 到大约 476kHz 的范围内
271.    Flash_Erase();                            // 擦除 Info_B
272.    //-----把 InfoFlash_Data[CAL_NUM]存入 InfoFlashB-----
273.    for(i=0;i<CAL_NUM;i++)
274.      Flash_Direct_WriteWord(i*2,InfoFlash_Data[i]);    // 不擦直接写
275.  }
276.
277.  /*************************************************************************************************
278.   * 名    称：FREQ_TEST()
279.   * 功    能：读取已写入 Flash 的校准参数，测量 DCO 频率，通过按键控制循环输出各频率，供外部观测。
280.   * 说    明：测量结束，红灯灭，绿灯亮，指示程序自校验结束，可用按键控制循环输出校准后的频率
281.   *************************************************************************************************/
282.
283.  void FREQ_TEST()
284.  {
```

```
285.      DCO_TEST_MEASURE();   //自测 16 个频率，写入 F_TEST[]
286.      P1OUT|=BIT6;              //绿灯亮，指示可用按键切换输出频率
287.      P1OUT&=~BIT0;              //红灯灭，指示自校验过程完成
288.      P1IE|=BIT3;               //开启按键中断
289.      LPM0;                     //CPU 休眠，等待按键中断，切换频率
290.    }
291.
292.    /*********************************************************************************************
293.     * 名    称：DCO_TEST_MEASURE()
294.     * 功    能：自检验校准后的频率
295.     * 说    明：每组数据测量三次后取平均值存入 F_TEST[]
296.     *********************************************************************************************/
297.    void DCO_TEST_MEASURE()
298.    {
299.      int j;
300.      int k;
301.      unsigned int F_temp[3]={0};
302.      for(j=0;j<CAL_NUM;j++)
303.      {
304.        DCOCTL=CALDCO_MHz(j+1);
305.        BCSCTL1=CALBC1_MHz(j+1);
306.        __delay_cycles(15000);
307.        for(k=0;k<3;k++)
308.        {
309.          F_temp[k]=FREQ_MEASURE(); ;      //计算 DCO 准确频率
310.        }
311.        F_TEST[j]=(F_temp[0]+F_temp[1]+F_temp[2])/3;
312.      }
313.      DCOCTL=CALDCO_MHz(FREQ_NUM);
314.      BCSCTL1=CALBC1_MHz(FREQ_NUM);
315.      __delay_cycles(15000);
316.    }
317.
318.
319.    /*********************************************************************************************
320.     * 名    称：F_CHANGE()
321.     * 功    能：按键中断响应，切换输出频率。
322.     * 说    明：按键小抖动，循环输出 16 个频率。
323.     *********************************************************************************************/
324.
325.    #pragma vector=PORT1_VECTOR
326.    __interrupt void F_CHANGE(void)
327.    {
328.      __delay_cycles(10000);
329.      if(~P1IN&BIT3)
330.      {
331.        FREQ_NUM++;              //每次按下按键，频率+1MHz
332.        if(FREQ_NUM==17)
333.        {
334.          FREQ_NUM=1;
335.        }
336.        DCOCTL=CALDCO_MHz(FREQ_NUM);
337.        BCSCTL1=CALBC1_MHz(FREQ_NUM);
338.        __delay_cycles(15000);
339.      }
340.      P1IFG=0;                  //清除标志位
341.    }
342.    /*********************************************************************************************
```

```
343.     * 名    称：DT_ISR()
344.     * 功    能：唤醒 CPU
345.     * 说    明：按键小抖动，循环输出 16 个频率。
346.     ********************************************************************************************/
347.    #pragma vector=WDT_VECTOR
348.    __interrupt void WDT_ISR(void)
349.    {
350.      LPM0_EXIT;    //退出中断时，唤醒 CPU
351.    }
```

附录 2 总结和心得（陈钱牛）

此次实验历时一个礼拜整。看着报告的完工，我心中一块铅石落地，自豪、喜悦以及淡淡的害怕与紧张。

我觉得我们此次实验是有意义的，而不只是为了满足自我的虚荣或者再低一层次只为了交作业。我认为做 MSP430 的基础实验比做具体的产品更有意义。产品做出来的东西不过是市面上花一点小钱就能买到的东西，即使锻炼了自己的能力，那也只是仅此而已，核心内容无外乎参考例程，看人家怎么做然后照猫画虎搬过来改改。但是做基础研究就不一样，当然，是深入去研究而不是写段代码把功能实现。我们的研究成果把一系列关于 DCO 调频的问题具体解决了，成果可以直接供他人学习使用（当然如果能看到的话）。这样我们也算是为大我尽了一份绵薄之力。我觉得这是至关重要的。

我们不像其他组把重心全放在设计上，我们的重心是放在把 DCO 调频这个问题研究透彻上，从问题出发再回到问题去设计实验，改进实验。所以我们详细分析了实验原理，整理了详细的实验结果，并把不起眼的小问题也深入研究并发现原因，最终解释清楚。我们的目标就是让读者在阅读这篇文章的时候能够尽可能地了解与 DCO 调频相关的所有信息以及会遇到的所有问题，甚至能够把此次成果直接转化到往后的教学工作上。

在此次任务中，我主要担任队长和总设计人的身份。我从全局上把握并负责分工，最后再由我整合。在分工过程中，我对每个人的工作量都有一定的控制，不存在“抱大腿”的现象——即使是编程能力稍弱的王玮也独立完成了 Flash 写入模块的设计实验和总试验的数据记录和处理工作（还有其他工作）。以下分选题、分工、设计检测模块、综合调试、报告撰写、报告综合几个方面细说。

选题。选题不是一件容易事，我们讨论了一天仍悬而未决，想要做万用表、MP3、蓝牙遥控器……但总觉得做这些费时费力而意义不大。最后决定要做 DCO 的想法是我提出的，仔细分析后说服组员并确立了我们的目标——专注于一个点把问题研究透，而不追求花哨地把各种功能杂揉到一起。

分工。考虑到各人能力有差异，以及我本就不赞同平均主义，每个人地分工内容都有差异，但总体还是能够平衡的。对于梁永回，思路清晰且编程能力强，我就让他多负责点编程任务。当然也有试验任务，自己的模块完成之后也是需要实验验证才能综合使用的；对于王玮，编程能力较弱，但是工作认真仔细，我就让她多负责总实验的数据整理分析和代码注释等工作，当然她也独立设计了 Flash 写入模块。至于我自己，我在完成自己的任务之外，基本在各方面都有插手，每个人个工作上交后我都需要进行修改。总的来说，我们的分工是明确的，但在执行的过程中又是相互交叉，互帮互助的，没有人占便宜抱大腿，大家都付出了宝贵的时间和心血。

设计检测模块。这个模块还是相对简单的，用我们已学的知识加梁永回设计的测频模块调用就可以实现。这个模块设计过程中主要是在中断嵌套中出现不可控的错误，而且都是极

不符合逻辑的错误。我没有花大力气去解决它，而是选择了退一步换一种实现方式。这是因为这不是我们研究的关键，我们在这上面花费大量时间精力是不合适的，在考试周还不如把时间花到复习功课上去。问题就留在那里，等以后有时间我再去一探究竟。不使用中断后问题迎刃而解，而且效果十分理想。

综合调试。由于事先把函数名、变量名都敲定了，所以综合过程只需要把各人的代码拷贝到一个.c 文件上去就好。有遇到编译问题，但问题不大。调试也比较顺利，这主要归功于梁永回工作到位，把校准部分提前调试完成。然后，我又把代码一句句精简，统一格式，使其美观整洁。

报告撰写。报告内容我主要写了程序的实现过程。根据老师的要求，我把源码放到了附录 1 中，在文中全部使用方框图予以说明，共引用了方框图 9 幅。采用“总-分”的结构从整体思路到各个模块依次说明其作用和实现过程。由于 visio 使用不熟练，框图制作效率较低。

报告综合。分工撰写前，我先把 word 文档的模板制作完成，将各章标题和小节标题确定，大家按照我给的提纲来写。各人完成自己的专业任务后，我负责审阅，我觉得不符合要求的我会要求重写，最后整合到一块。再分给各人审阅，将错误与不妥之处集中改进。其中最让我们头痛的是页眉和页码，尤其是决定做双面打印后，要分奇偶页给页眉和页码，而且目录以前的格式要和目录以后的格式不同。Word 好像总是和我们过不去，会出现各种错误。这让我们又花费很许久。

总的来说，这次实验让我们付出了很多也收获了很多。我觉得每个人的认真和付出都远超于此次实验任务本身的期望，应该得到最高评价。

附录 3 总结和心得（王玮）

我们的实验切入点很明确，就是充分利用 MSP430G2553 芯片内部资源，实现对 DCO 频率的校准。也许听起来并没有很“华丽”，但是我们认为这个选题还是非常值得深入学习，很有价值，而且将其理解透彻并设计出一套完整的程序也并非易事。

在构建好程序整体思路的基础上，我们三人分工合作，设计不同的功能模块。我负责设计写入模块。通过查阅用户手册、数据手册及其他资料，学习了时钟模块及 Flash 相关内容及设置方法等。首先明确了两大步骤，即第一将上一校准模块得到的参数进行合成、封装，第二就是将第一步得到的数依次存入 InfoFlashB 段，分别对应两个函数。

在参数封装时，规范起见，仿照 FlashA 段 DCO 的出厂校验参数格式合成 16 位的数并记入数组；在写入函数中，由于是字写入，我直接使用库函数，在正确的写入频率下，根据相关地址及数据位数等，可简洁、正确地将参数写入。

在编写程序的过程中我遇到的问题是如何查看程序是否已经根据我的要求，将设置的数组存入 InfoFlashB 段相应的地址。查找资料得知，在 Memory Browser 中，输入 FlashB 段的首地址 0x1080 便可查看其中的值，对程序功能进行检验。这一功能在我们之前的实验中没有接触到，相对比较陌生，但是通过动手反复尝试，我意识到灵活掌握调试和查看功能对编程和改进有非常重要的意义，这也帮助我提高了这一方面的能力。

另外我还负责了总实验的结果记录和分析。在实验中，不仅要查看变量的值，还需要编写一些程序对出厂的设置进行测试，或是对我们的思路想法进行测试检验，由此验证了频率校准的精确性。在实验测试时我遇到的一个问题是多次测量结果中，在 15MHz 频率点的误差始终明显大于其他频率点，于是进行了分析。首先我查看了相关寄存器变量，发现了在该频率点 MODx 值始终为 0，更换板子依旧如此，编写程序进行了进一步试验，发现对 MODx 的赋值不同值并不起作用，于是结合参数调节频率的原理，认为这可能恰是我们之前忽略的在 DCOx=7，MODx 不被采用的情况，于是再查看变量，对相关纯净频率值大小及对应参数设置进行分析（具体见实验报告），发现实验结果的这一现象确实是由于 DCOx=7 时的本身特性导致，与理论相符。深入探究了这一现象并找到了根源。

我们去了 3 次实验室，用示波器对我们的输出频率及波形进行观察，意识到我们没有可以准确测量频率的仪器，因此调整策略，增加了检测模块；但是示波器仍帮助了我们验证按键控制下检测频率的功能；观察示波器输出波形出现的“奇怪”重影现象也启发了我们对 MODx 调节频率原理的深入探究。由此得到严谨、全面的实验分析结果。

除了实验，我们在实验报告的撰写上也付出了大量心血，从中也收获了很多。

附录 4 总结和心得（梁永回）

实验中遇到的问题以及解决办法

我们将 1-16MHz 频率的校准参数写入 Info_Flash B 以后，想用实验室的示波器测一测，与预期值做比较。示波器测量频率，主要有两种方式。一是从图中读取格数，

周期 *T*=扫描速度（t/div）× 水平方向重复一周期所占的格数（div），频率 *f=1/T*

这种方法明显不准确，光是读数的误差就够受的了，更何况还有混频抖动的现象。二是使用示波器的测量频率功能，看起来很好，直接报频率连算都不用算。我们采用的是第二种。问题又来了，随着频率的增高，示波器报出的频率会在某一范围内规律性跳变，10MHz 以上的时候，跳动范围甚至接近 1MHz。如果改变扫描速率，报出的频率范围也会改变。这种方式明显也不合适。主要有以下几点①用示波器测频的时候，要先利用示波器附带的校准信号进行校准，我们没有校准，而且示波器报出的频率会随着扫描速度的改变而改变。②示波器的波形显示的是连续的电压的瞬时值，并非有效值。如想与电压表（或万用表）测量值进行比对，必需按波形折算出对应的有效值。③此示波器带宽为 60MHz，我们所要测量的频率的最大值接近 20MHz，对于数字示波器，如果要观察信号的波形，则频带至少要为被测波形的 3 倍。

因此，第二种办法也被我们放弃了，于是我们另辟他径，利用单片机内部的看门狗定时中断唤醒 CPU 的方法来定时，完成测高频的功能，且无需外接设备。

另外一些问题都是在共同调试程序中遇到的小问题，例如寄存器的名字写错、程序前后顺序有误等等，是辅助陈钱牛解决的。

对课程的想法 建议

这们课程很好，让我们这群大二的学生能够走在数电改革的最前沿，并且能够接触研究生的学习模式。

①或许是我的观念太老旧了，我还是觉得如果老师们在数电实验和处理器实验的初期能够多提供一些指导，比如说所有人完成点灯实验、测量各个时钟源频率、在 LCD 显示屏幕上输出各自的学号……类似指导的目的，绝非老师口中的“死学一块板子”，而是引起大家的兴趣，就如同 c++课程中的 Hello World 程序一样，一下子引起了大家的好奇心，我们在心里告诉自己：这门课好像也是可以学会的嘛。我觉得，大多数人对数电及微处理器实验的最大的感受就是：未知、害怕、抵触。我们很少能从微处理器实验中获得成就感，大多是挫败感和无力感。

②实验报告的书写锻炼了我的书写能力，只不过平时精力有限，还有其他课业要完成，难受呀。

感谢

感谢金印彬老师和宁改娣老师的无私付出和悉心指导。以及本次实验队友的大力帮助。

MSP430 综合实验报告

陈钱牛 / 王玮 / 梁永回

2018 年 6 月