2 PLL 锁相环实验

2.1. 背景介绍

本章实验将介绍 PLL IP 核的使用以及 Cortex-M3 的时钟结构，并将 PLL 输出的时钟给 Cortex-M3 使用。

2.1.1. PLL 简介

PLL（Phase-Locked Loop）即相位锁定环路，也就是常说的锁相环。锁相环的特点是：利用外部输入的参考信号控制环路内部振荡信号的频率和相位。PLL 由一个 N 位预分频计数器，鉴频鉴相器（PFD,Phase-Frequency Detector），电荷泵（CP,Charge Pump），环路滤波器（LF,Loop Filter），压控振荡器（VCO,Voltage Controlled Oscillator），M 位的反馈乘法计数器以及 K 位和 N 位的后分频计数器构成，图 2. 1 为 PLL 基本结构框图。

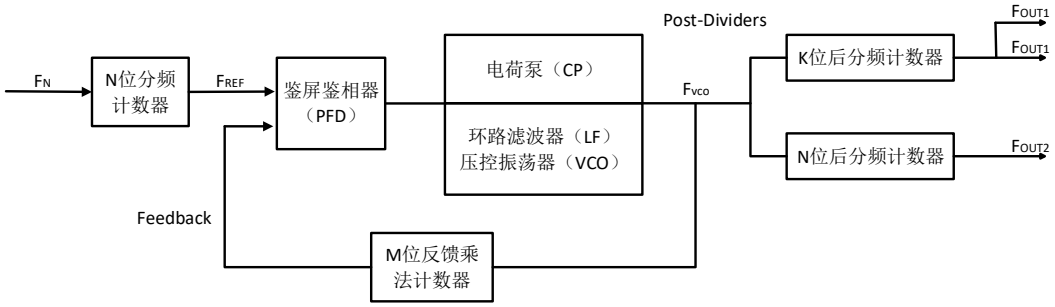

图 2. 1 PLL 基本结构框图

PLL 在工作的时候，鉴频鉴相器检测 F_{REF} 与返回信号（Feedback）之间的相位及频率差异，控制可将相位差异转换成电压的环路滤波器，进而控制压控振荡器产生更高或者更低的频率振荡。当 F_{REF} 与返回信号的频率相位相等时，输出电压与输入电压保持固定的相位差值，即输出电压与输入电压被锁住，这就是锁相环名称的由来。

2.1.2. PLL 应用

在某些对系统时钟频率没有固定要求的系统中，外部晶振输入的时钟可以直接作为逻辑驱动时钟。一些应用中可以通过 PLL 将外部晶振输入的时钟进行降频，以得到较低的工作时钟，在不影响系统功能实现的前提下降低系统功耗。

而另外一些应用，则必须在指定频率的时钟信号下才能正常工作，常见于通信协议类应用，如以太网、USB 以及 PCIE 等。在这些应用中，必须使用指定频率的时钟信号，如果没有刚好满足条件的外部时钟源，此时则可通过片内 PLL 生

成相应的时钟信号来进行驱动。

在某些实时性要求较高的应用中，如数字信号处理，图像处理等等，提高系统工作时钟能够提升系统的性能，这一类应用中，也往往使用 PLL 进行倍频和分频，以得到较高频率的时钟，用以提升系统整体性能。

再有一个常见的应用是对同一 PLL 生成的多个时钟的相位进行控制，以保证两个时钟域的逻辑工作时有确定的时间差。例如生成两路频率相同，相位不同的时钟，供 SDRAM 控制器和 SDRAM 芯片使用。根据 SDRAM 芯片的工作原理，SDRAM 控制器的工作时钟和 SDRAM 芯片的工作时钟需要保持一定的相位差才能保证正确的读写数据。所以这里就可以使用 PLL 的相位控制功能来产生两路相位不同的时钟，以分别供控制器和 SDRAM 芯片使用。

2.1.3. Cortex-M3 时钟结构说明

下图 2. 2 显示了 SEAL Cortex-M3 的时钟结构：

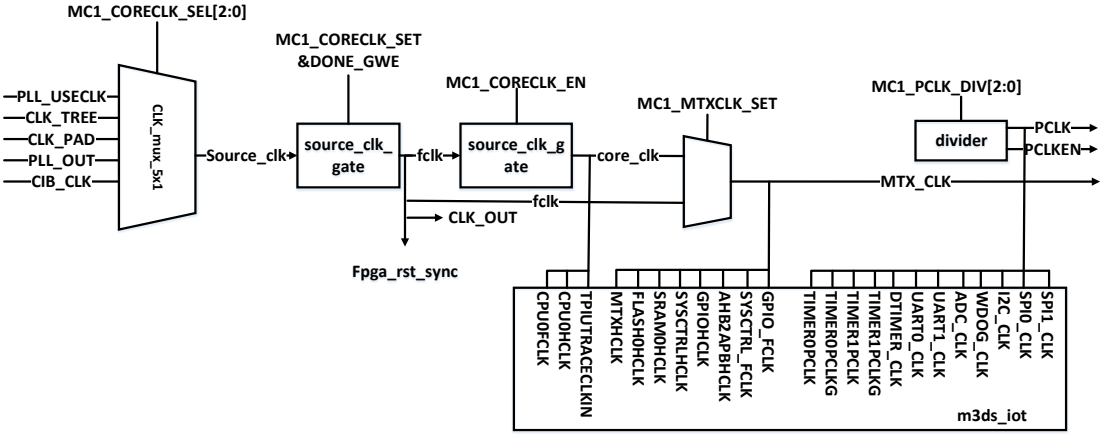

图 2. 2 Cortex_M3 时钟结构

从上图可以看出，Cortex-M3 的时钟来源分为 5 种：PLL_USECLK、CLK_TREE、CLK_PAD、PLL_OUT 和 CIB_CLK。下表 2- 1 对这 5 种时钟来源进行了说明。

表 2- 1 Cortex-M3 时钟来源说明表

时钟名称	类型	宽度	描述
PLL_USECLK	input	1	来自 OSC 分频器的时钟
CLK_TREE	input	1	来自 FPGA 时钟树的时钟
CLK_PAD	input	1	来自于芯片的外部引脚，对应的引脚名称为 DIFF_RX_C10N
PLL_OUT	input	1	来自 PLL 输出的时钟
CIB_CLK	input	1	来自 CIB 块的时钟，CIB 是逻辑内部互连线的缩写，可以用 PLL 过来的时钟，也可以用管脚输入的

对于图 2. 2 中的时钟输出说明如下表 2- 2 所示：

表 2- 2 Cortex-M3 时钟输出说明表

时钟名称	类型	宽度	描述
CLKOUT	output	1	PLL 产生的时钟
MTX_CLK	output	1	AHB 时钟输出
PCLK	output	1	APB 时钟输出
PCLKEN	output	1	APB 时钟输出使能

5 种时钟源通过一个 5 选 1 多路器（MC1_CORECLK_SEL）选择其中一个作为需要使用的时钟，其说明如下表 2- 3 所示。

表 2- 3 MC1_CORECLK_SEL 时钟选取说明表

赋值	时钟选取
000	选择时钟 PLL_USRCLK
001	选择时钟 CLK_TREE
011	选择时钟 CLK_PAD
101	选择时钟 PLL_OUT
111	选择时钟 CIB_CLK

代码中修改时钟来源的参数位于 cm3_system 文件中，如下所示：

```
defparam inst.CORECLK = " CIB_CLK";// 时钟来源选择
```

将选择的时钟 Source_clk 通过 source_clk_gate 模块确定是否使能时钟：TRUE：使能；FALSE：不使能。使能之后输出时钟 Fclk。代码中设置的位置如下所示：

```
defparam inst.CORE_SET = "TRUE";
```

Fclk 通过 source_clk_gate 模块中 MC1_CORECLK_EN 的值确定是否使能 CM3 内核时钟：TRUE：使能；FALSE：不使能。使能之后输出 core_clk。代码中设置的位置如下所示：

```
defparam inst.CORECLK_EN = "TRUE";
```

Fclk 和 core_clk 根据 MC1_CORECLK_EN 的值确定哪一个作为 AHB 总线的时钟输出：1：Fclk（也就是 sorce_clk）；0：core_clk。代码中设置的位置如下所示：

```
defparam inst.MTXCLK = "CORECLK"; // AHB 总线时钟 "CORECLK" "SOURCECLK"
```

APB 总线的时钟是通过 Divide 分频得到的，其取值与其对应的分频比如下表 2- 4 所示。

表 2- 4 APB 时钟与 Divide 关系说明表

Divide 取值	PCLK 取值
0	$f_{PCLK}=f_{HCLK}$
1	$f_{PCLK}=1/2f_{HCLK}$
2	$f_{PCLK}=1/3f_{HCLK}$
3	$f_{PCLK}=1/4f_{HCLK}$
4	$f_{PCLK}=1/5f_{HCLK}$
5	$f_{PCLK}=1/6f_{HCLK}$
6	$f_{PCLK}=1/7f_{HCLK}$
7	$f_{PCLK}=1/8f_{HCLK}$

在代码中可以进行修改的地方如下所示：

```
defparam inst.PCLK_DIV = 0;                    // 分频 PCLK_DIV 0-7
```

2.2. 实验介绍

本章实验使用 PLL 输出不同的时钟频率供 Cortex-M3 使用，并且通过 LED 的闪烁速度判断频率的高低。

2.3. 建立 HQFPGA 工程

本次实验将不再重新建立工程，而是对已有的工程进行修改，修改方式如下。

2.3.1. 复制已有工程到新工程目录

将第一个实验的 CM3_System 文件夹复制到 CM3_PLL 文件夹之下，并且将 GPIO 重新命名为 CM3_PLL，操作步骤如下图 2. 3 所示。

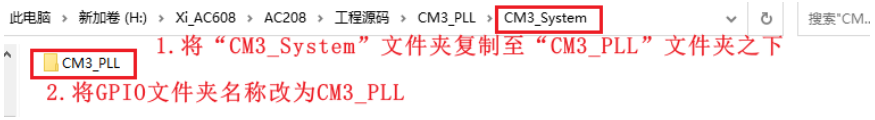

图 2. 3 复制已有工程至新工程目录下

2.3.2. 修改工程名

将第一个实验的工程名“GPIO”修改为“CM3_PLL”,并且将第一个实验生成的扩展文件“GPIO.hqprj_backup”删除，如下图 2. 4 所示。

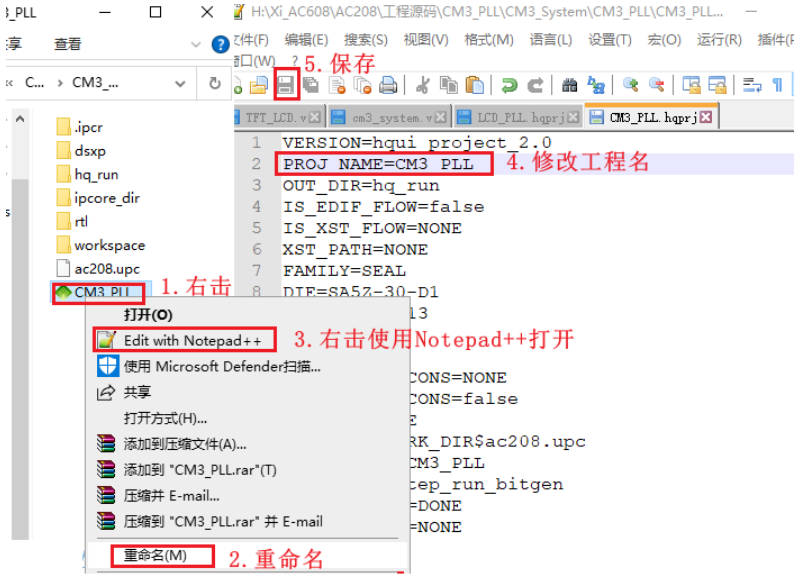

图 2. 4 修改工程名

2.3.3. 修改顶层文件名

2.3.3.1. 修改工程中顶层模块名

在 Notepad++界面中，Ctrl+F 搜索关键词“LED_KEY”，将其替换成 CM3_PLL，如下图 2. 5 所示。

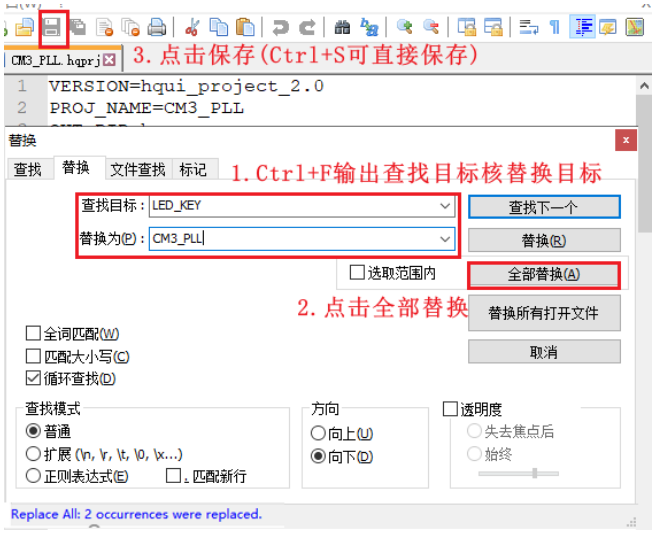

图 2. 5 修改工程中的顶层模块名

2.3.3.2. 修改顶层模块名

修改顶层模块名主要分为两步：

1. 修改文件名：找到 rtl 文件下的顶层文件（LED_KEY），重新命名为 CM3_PLL。

2. 使用 Notepad++打开，将模块名修改为 CM3_PLL。

操作步骤如下图 2. 6 所示。

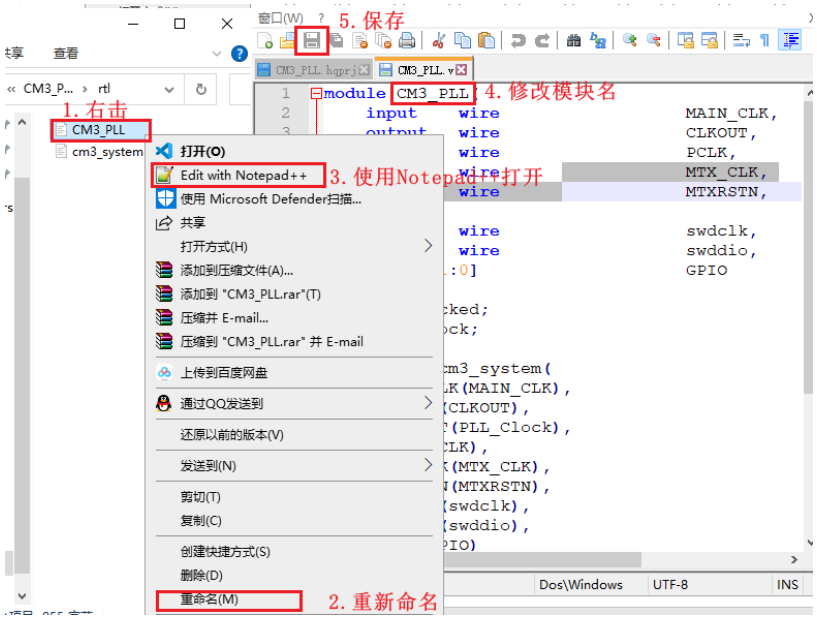

图 2. 6 修改顶层模块名

2.3.4. 删除不需要的文件

打开 hq_run 文件夹，删除上一个实验生成的 bin 文件及其扩展文件，操作方式如下图 2. 7。

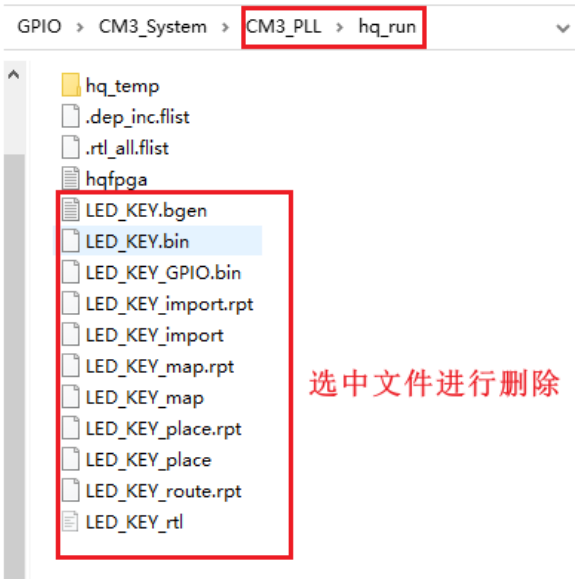

图 2. 7 删除 bin 及其扩展文件操作示意图

2.4. 添加 PLL IP 核

2.4.1. IP 管理中添加 IP 核

打开 HQ 工程，点击 IP 管理，进入 IP Creator 界面，找到时钟管理单元中的 PLL_FREQ_30K 双击或者点击创建进入配置界面，操作如图 2. 8 所示。

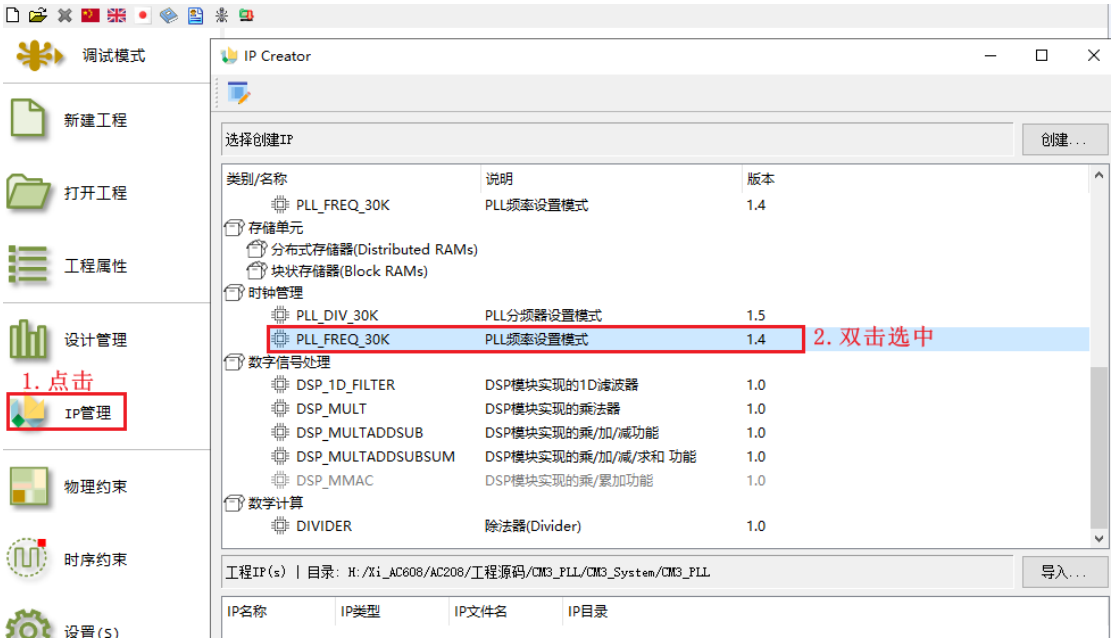

图 2. 8 添加 PLL IP 核操作示意图

PLL 配置界面如下图 2. 9 所示。

图 2. 9 PLL 配置界面示意图

在 PLL 的配置界面中可以看出，PLL 的 IP 核提供有 7 路时钟输出，输出的频率必须与输入时钟有一定分数关系，LOCK 在锁定之后输出高电平，可支持设置输出相移，根据本次实验，配置 PLL 核的步骤如下所示：

1. 修改输入时钟为 25Mhz。
2. 勾选锁定。
3. 设置输出时钟 CLKOP 为 50Mhz。

配置步骤操作示意图如下图 2. 10 所示。

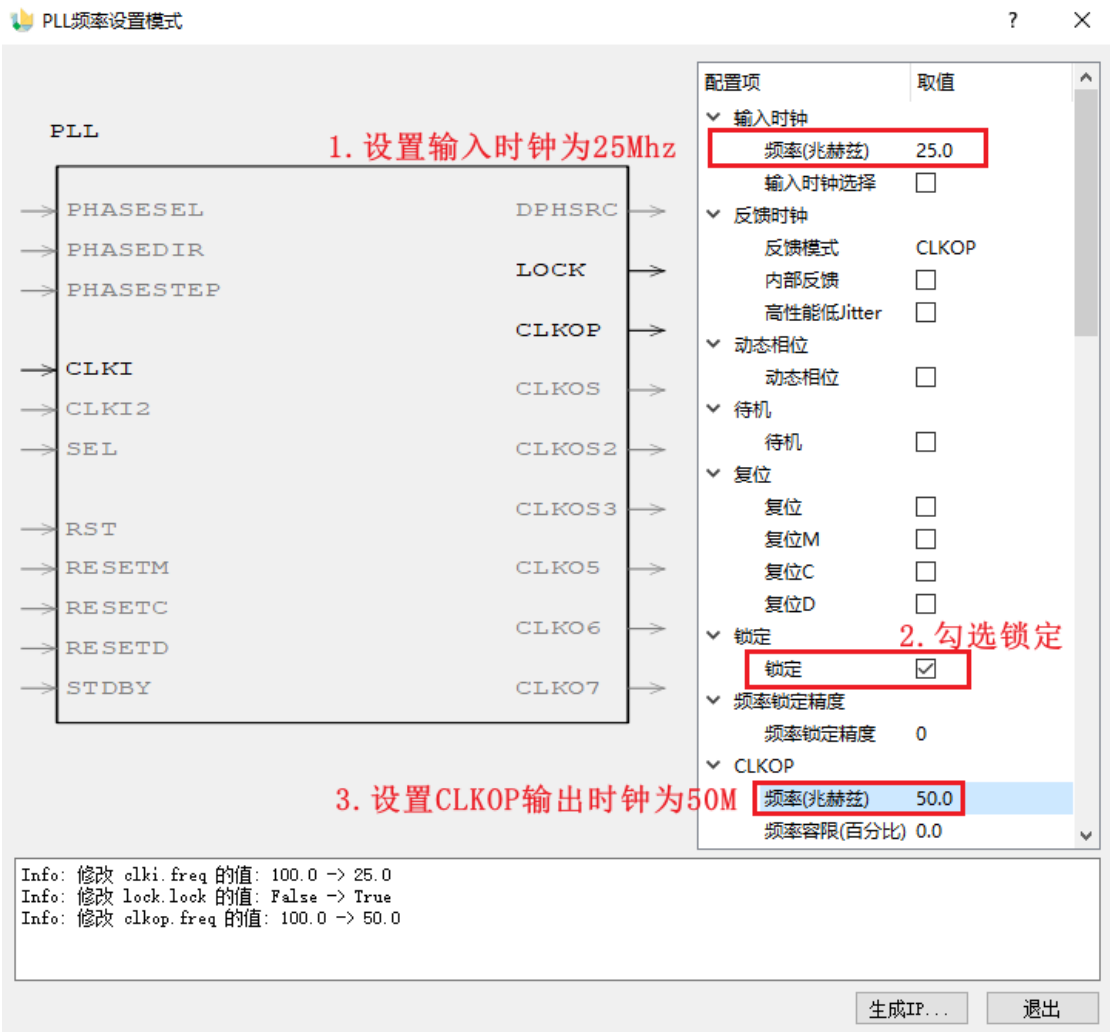

图 2. 10 配置 PLL IP 核

通过上诉步骤配置完成之后点击界面右下角生成 IP，进入保存 IP 界面，可以设置模块名称、文件名、文件格式及存放的 IP 目录，用户可根据自己需求进行设置，也可以保持默认，在这里我们将模块名称修改为 PLL，最后点击保存，操作界面如下图 2. 11 所示。

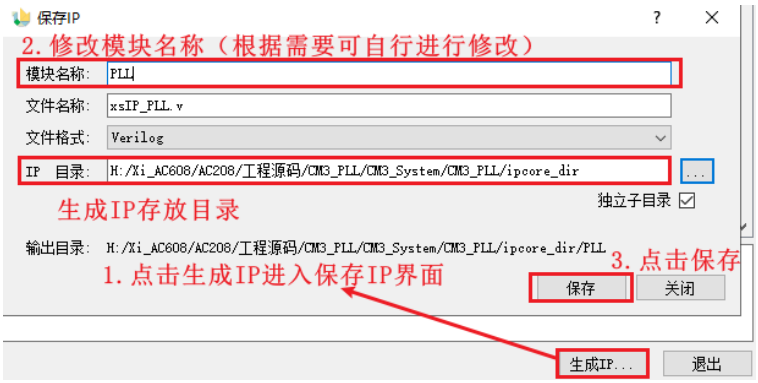

图 2. 11 保存 IP 界面示意图

保存完成之后，在 PLL 设置界面的显示信息窗口会显示完成 IP 生成，然后点击退出，退出 PLL 的设置，如下图 2.12 所示

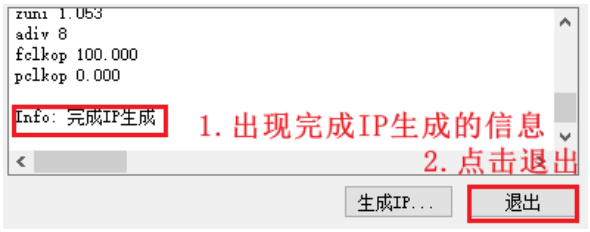

图 2.12 退出 PLL IP 核的设置

退出之后，在 IP Creator 界面下面，会检测到我们添加的 IP 核，如下图 2.13 所示。

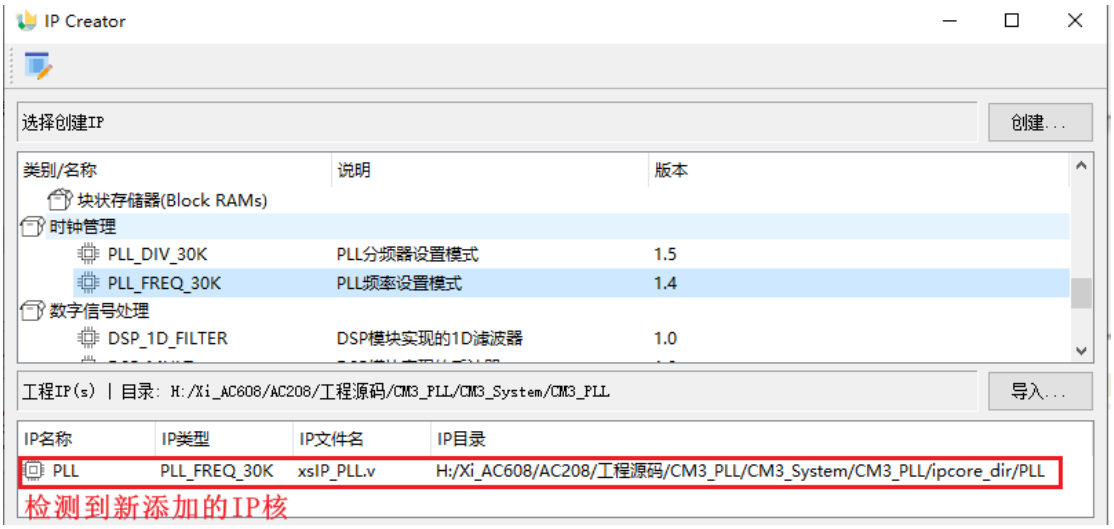

图 2.13 IP Creator 界面检测到新添加的 IP 核示意图

双击上图中检测到的 IP 核，可以再次进入 IP 核配置界面，如果需要更改 IP 设置，在更改后点击“更新 IP”并保存，完成 IP 设置的修改（该种方式进行修改太过于繁琐，不建议使用，可以在设计管理器中进行修改，具体可以参看 2.5.1 一节中的内容），这里需要注意两点：

1. 手动设置 IP 保存路径。在更新 IP 之后，IP 的默认路径与我们之前设置的不一致，如果不设置为相同路径的话，在调用 IP 的时候还是调用的原来未更新的 IP，这里就需要我们手动去设置 IP 的保存路径，确保和之前的路径保持一致。

2. 不勾选独立子目录的生成。勾选独立子目录之后，会在我们选择的 IP 目录之下生成一个新的文件夹（新文件的名称和模块名称一样）。这将会导致 IP 路径保持不一致，这里就需要我们取消独立子目录的生成，操作步骤如下图 2.14 所示。

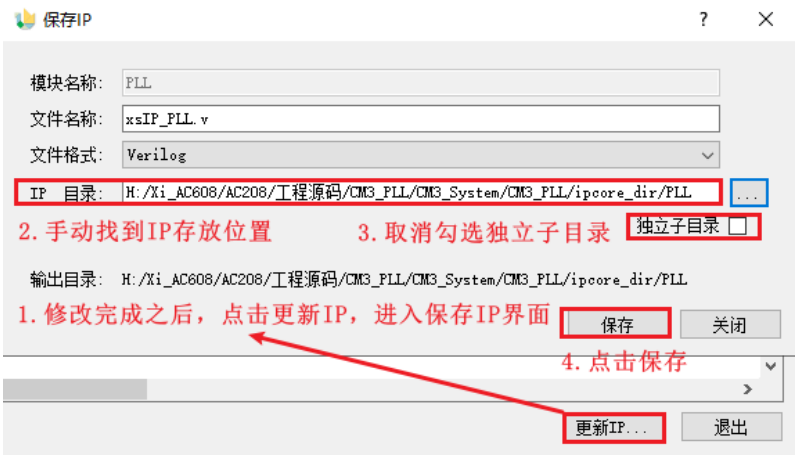

图 2. 14 修改 IP 之后保存 IP 操作示意图

2.4.2. 工程属性中添加 IP 文件

点击工程属性，在源文件一栏中，点击“+”号，添加 IP 的.v 文件，操作步骤如下图 2. 15 所示。

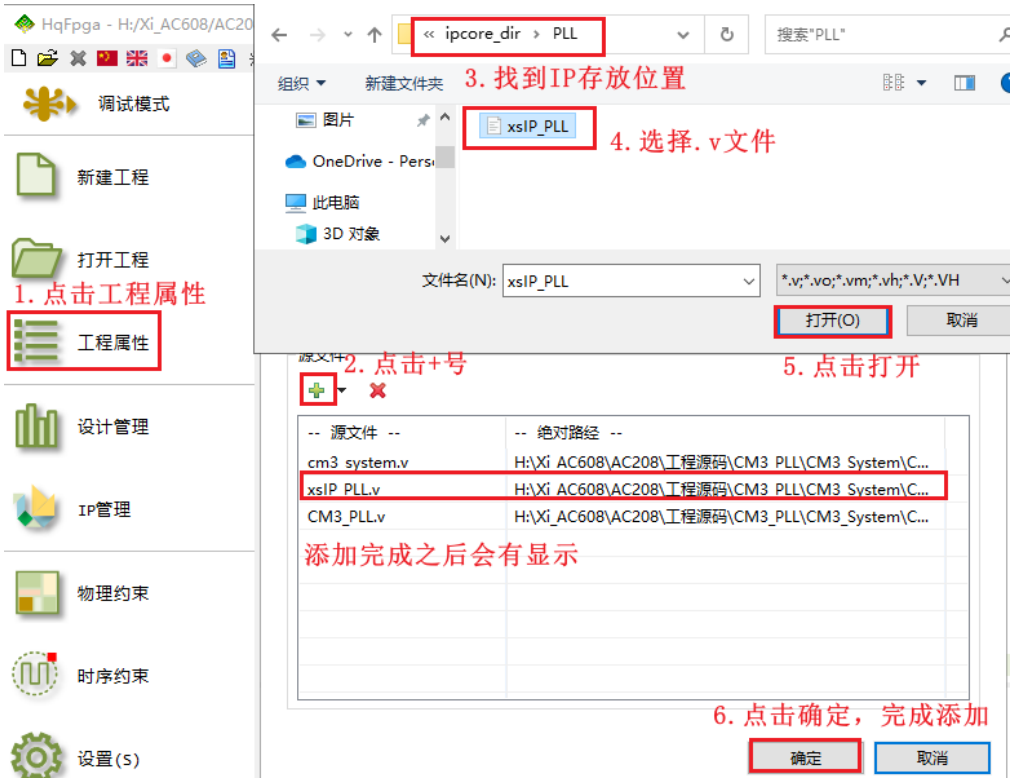

图 2. 15 工程属性中添加 IP 文件

2.5. FPGA 侧代码修改

2.5.1. 例化 PLL 模块

点击全部运行，加载文件，然后点击设计管理进入设计管理器，可以看到我

们新添加的 PLL IP。之前我们介绍过在 IP 管理界面修改 IP 参数（具体参考 2.4.1 一节中的内容），其实在设计管理器中也可以进行修改，通过右击 PLL IP 选择配置 IP 即可进入 IP 配置界面进行修改（建议使用该种方式），操作方式如下图 2.16 所示。

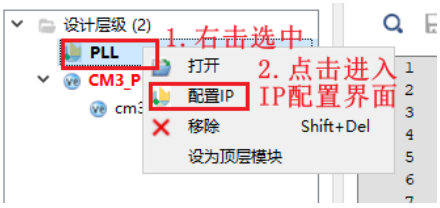

图 2.16 修改 IP 配置参数示意图

右击 PLL IP 点击打开，即可看到 PLL IP 的.v 文件，如下图 2.17 所示。

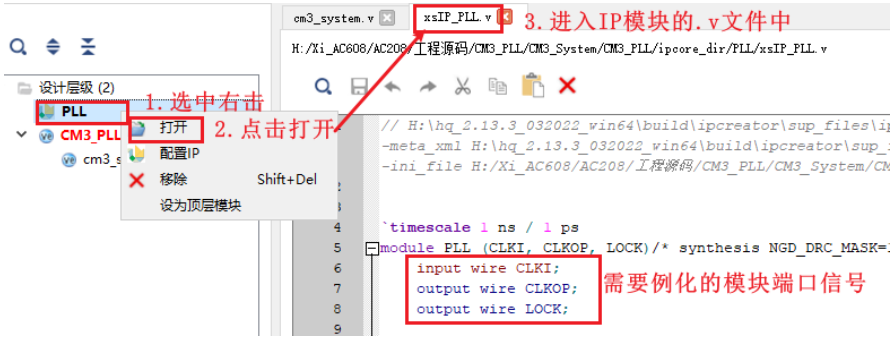

图 2.17 查看 IP 模块.v 文件操作示意图

将上图所示模块的端口信号例化至顶层模块“CM3_PLL”当中，例化代码如下所示：

```
wire PLL_Locked; //PLL lock 信号
wire PLL_Clock; //PLL 输出时钟信号
PLL PLL(
    .CLKI(MAIN_CLK),
    .CLKOP(PLL_Clock),
    .LOCK(PLL_Locked)
);
```

2.5.2. 修改 Coretx-M3 的时钟来源

点击 cm3_system 文件，修改 Cortex-M3 的时钟来源，修改步骤如下所示：

1. 添加 PLL 输出时钟模块端口信号如下所示：

```
input    wire                PLL_OUT,        //PLL 输出时钟
```

2. 将 PLL 输出时钟连接至 Cortex-M3 模块的 PLL_OUT 输入端口（xsCM3 inst），如下所示：

```
.PLL_OUT            (PLL_OUT),
```

3. 修改时钟源参数，根据 2.1.3 一节中的内容得知，当我们选择用 PLL 输

出时钟作为 Cortex-M3 的时钟来源时，其对应的时钟源参数为“PLL_OUT”，需要注意的是，如果时钟源为“PLL_OUT”，只能使用 PLL 的 CLKOP，而不能用 CLKOS。修改如下所示：

```
defparam inst.CORECLK = "PLL_OUT";        // 时钟来源选择
```

4. 在第 1 步的时候添加了 PLL 输出时钟模块，在顶层例化中需要将该信号与 PLL 的输出时钟 PLL_Clock 相连接，如下所示：

```
.PLL_OUT(PLL_Clock),
```

5. 需要注意的是在 FPGA 侧代码中将时钟来源修改之后，还需要在 MDK 软件中修改时钟参数。MDK 软件中在“system_CM3DS.c”文件中修改时钟参数，其中 SYSTEM_CLOCK 就是 Cortex-M3 的时钟频率，如下图 2. 18 所示。

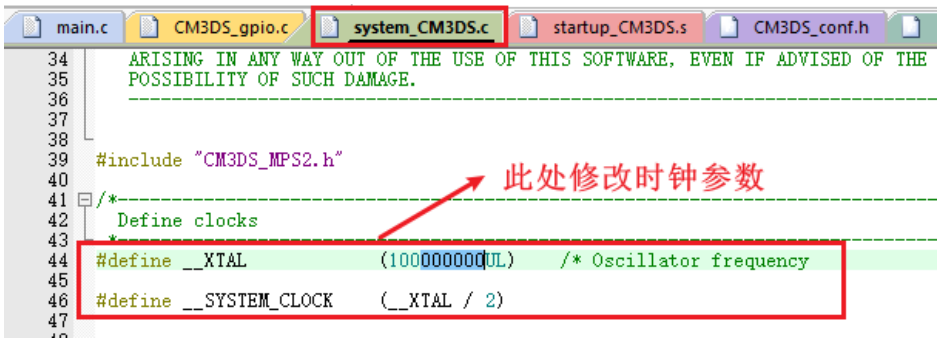

图 2. 18 MDK 软件中修改时钟参数

2.6. 物理管脚约束

打开工程，点击物理约束，可以看到第一个实验添加的引脚约束文件“AC208.upc”，将其修改为 AC208 开发板通用的约束文件，修改方式如下。

第一步：清除已有的约束文件，操作方式如下图 2. 19 所示。

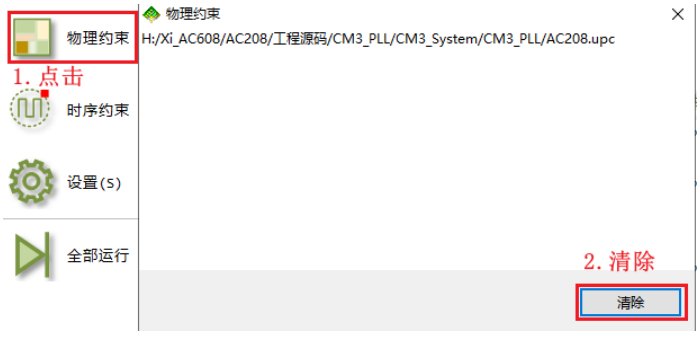

图 2. 19 清除已有的约束文件

第二步：添加新的引脚约束文件。将我们提供的 AC208 通用约束文件“ac208.upc”复制至工程目录下（用户在添加的时候可以在我们提供的例程中进行复制）。操作方式如下图 2. 20 所示。

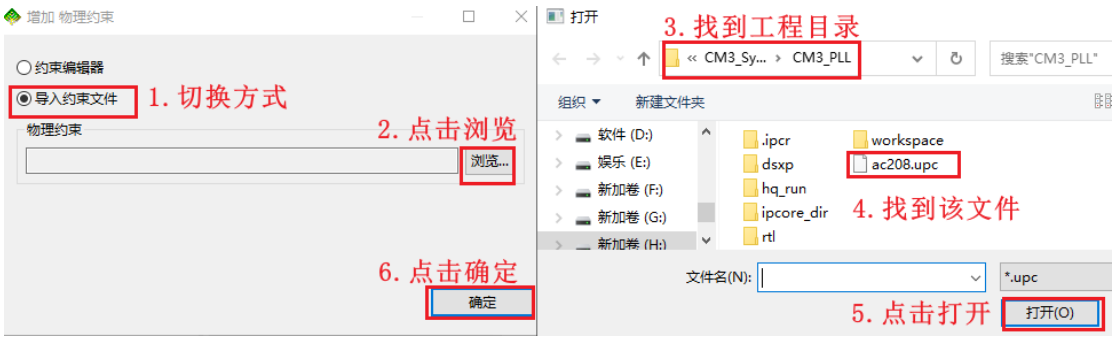

图 2. 20 添加引脚约束文件

2.7. 编译设计

点击“全部运行”按钮，对设计进行全编译并生成 bin 文件。操作步骤如下图 2. 21 所示。

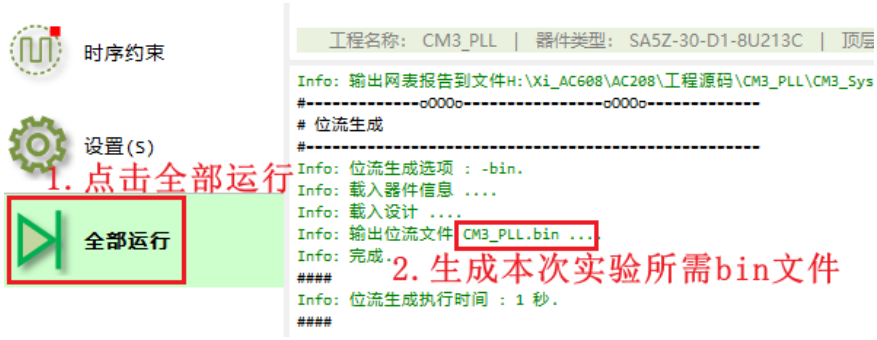

图 2. 21 编译运行整个文件

2.8. 建立 MDK 工程

重新建立一个新的 MDK 工程非常复杂，我们可以通过对第一个实验的 MDK 工程进行修改，从而减少建立工程所需的时间，修改步骤如下。

2.8.1. 复制已有工程至新工程目录

将第一个实验 CM3_Softerware 文件夹复制到 CM3_PLL 文件夹之下。

2.8.2. 修改工程名

将第一个实验的工程名“GPIO”修改为“CM3_PLL”，操作方式如下图 2. 22 所示。

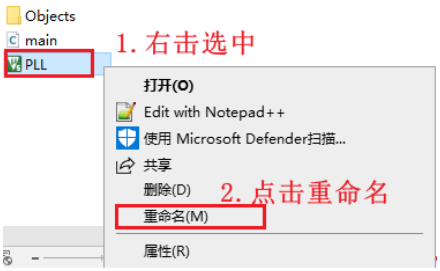

图 2. 22 修改 Keil 工程名

2.8.3. 修改输出文件名

将工程通过 Notepad++打开，查找目标 GPIO，找到需要修改的部分，将其替换成 Timer。需要修改的位置如下图 2. 23 所示。

```
50            </TargetStatus>
51            <OutputDirectory>.\Objects\</OutputDirectory>
52            <OutputName>CM3_PLL</OutputName>
53            <CreateExecutable>1</CreateExecutable>
54            <CreateLib>0</CreateLib>
84              <RunUserProg2>1</RunUserProg2>
85              <UserProg1Name>$K\ARM\ARMCC\BIN\fromelf.exe #L -c -d -e -s --output CM3_PLL.lst
86              <UserProg2Name>$K\ARM\ARMCC\BIN\fromelf.exe #L --bin -o CM3 CM3 PLL.bin</UserPr
87              <UserProg1Dos16Mode>0</UserProg1Dos16Mode>
88              <UserProg2Dos16Mode>0</UserProg2Dos16Mode>
448       <LayerInfo>
449         <Layers>
450           <Layer>
451             <LayName>CM3_PLL</LayName>
452             <LayPrjMark>1</LayPrjMark>
453           </Layer>
454         </Layers>
455       </LayerInfo>
456
```

图 2. 23 修改输出文件名

2.8.4. 删除不需要的文件

将上一个实验生成的 bin 文件和一些扩展文件进行删除，操作方式如下图 2. 24 所示。

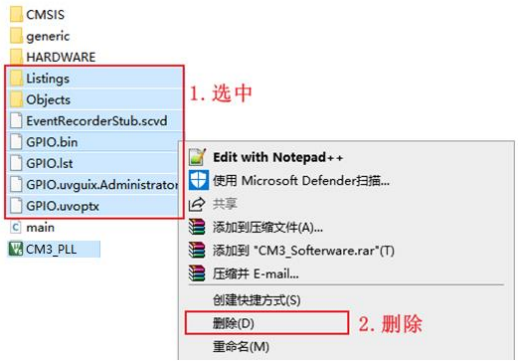

图 2. 24 删除不需要的文件操作示意图

2.8.5. 调试软件设置

打开工程之后，还需要对调试软件设置，否则将无法下载程序，操作步骤参考 1.7.3 节中的内容。

2.9. 软件设计

实验内容：本次实验通过延时的方式，实现 LED 的闪烁，并且在延时相同时间的情况下，通过修改 PLL 的输出时钟使 LED 灯的闪烁频率不一样。

实验代码：在第一个实验中我们介绍了 GPIO 库的使用，了解了如何点亮和熄灭 LED 灯。LED 灯的闪烁是通过延时实现的，延时的实现方式有很多，这里使用的 CM3 内核中的 SysTick 定时器，下面将简单的介绍一下 SysTick 定时器。

2.9.1. SysTick 定时器

Cortex-M3 处理器内部集成了一个 SysTick(系统节拍)的定时器，SysTick 为简单的向下计数的 24 位计数器，可以使用处理器时钟或外部参考时钟。

在现代操作系统中，需要一个周期性的中断来定期触发 OS 内核，如用于任务管理和上下文切换，处理器也可以在不同时间片内处理不同任务。处理器设计还需要确保运行在非特权等级的应用任务无法禁止该定时器，否则任务可能会禁止 SysTick 定时器并锁定整个系统。若应用中不需要使用 OS，SysTick 定时器可用作简单的定时器外设，用以产生周期性中断、延时和时间测量。本次实验中就是使用该定时器实现的延时。

2.9.1.1. SysTick 寄存器简介

SysTick 定时器中存在 4 个寄存器。在“core_cm3.h”文件中定义了一个名为 SysTick 的结构体，方便对这些寄存器的访问。下表 2- 5 中对 SysTick 的寄存器进行了简单了介绍。

表 2- 5 SysTick 寄存器一览表

地址	结构体调用	寄存器
0xE000E010	SysTick->CTRL	SysTick 控制和状态寄存器
0xE000E014	SysTick->LOAD	SysTick 重装载值寄存器
0xE000E018	SysTick->VAL	SysTick 当前值寄存器
0xE000E01C	SysTick->CALIB	SysTick 校准值寄存器

SysTick 寄存器的详细描述如下表 2- 6~表 2- 9 所示。

表 2- 6 SysTick 控制和状态寄存器（0xE000E010）

位	名称	类型	复位值	描述
16	COUNTFLAG	RO	0	当 SYSTICK 定时器计数到 0 时，该位变为 1，读取寄存器或清除寄存器当前值时会被清零。
2	CLKSOURCE	R/W	0	0=外部参考时钟（STCLK）；1=使用内核时钟
1	TICKINT	R/W	0	1=SYSTICK 定时器计数减至 0 时产生异常；1=不产生异常
0	ENABLE	R/W	0	SYSTICK 定时器使能

表 2- 7 SysTick 重装载值寄存器（0xE000E014）

位	名称	类型	复位值	描述
23:0	RELOAD	R/W	未定义	定时器计数为 0 时的重装载值

表 2- 8 SysTick 当前值寄存器（0xE000E018）

位	名称	类型	复位值	描述
23:0	CURRENT	R/W	0	读出值为 SYSTICK 定时器的当前数值。写入任何值都会清除寄存器，SYSTICK 控制和状态寄存器中的 COUNTFLAG 也会清零。

表 2- 9 SysTick 校准值寄存器（0xE000E01C）

位	名称	类型	复位值	描述
31	NOREF	R	-	1=没有外部参考时钟（STCLK 不可用）;0=有外部参考时钟可供使用
30	SKEW	R	-	1=校准值并非精确 10ms;0=校准值准确
23:0	TENMS	R/W	0	10 毫秒校准值。芯片设计者应通过 Cortex-M3 的输入信号提供该数值，若读出为 0，则表示校准值不可用

2.9.1.2. SysTick 定时器延时使用

使用 SysTick 定时器进行延时有两种方式：一种是通过中断的方式；另外一种是直接操作寄存器。本次实验使用第二种方式。

1. 定时器中断

产生周期性的 SysTick 中断，最简单的方法就是使用“core_cm3.h”文件中的 SysTick_Config 函数：

```
uint32_t SysTick_Config(uint32_t ticks)
```

该函数将 SysTick 中断间隔设置为 ticks，使能计数器使用处理器时钟，然后设置 SysTick 异常为最低优先级。

例如,若要在 25MHz 的时钟频率下产生 lkHz 的 SysTick 异常,则可以使用：

```
SysTick_Config(SystemCoreClock/1000);
```

其中 SystemCoreClock 应该存放的正确的时钟频率值（位于 system_CM3DS.c 文件中），也就是 25000000。

通过上诉操作之后，SysTick_Handler(void)的触发频率就变成了 1kHz。若 SysTick_Config 函数的输入参数不满足 24 位重加载数值寄存器(大于 0xFFFFFF)，SysTick_ Config 函数返回 1 ，否则会返回 0。

2. 寄存器操作

许多情况下，可能会使用参考时钟或者不想使能 SysTick 中断，那么就不要使用 SysTick_Config 函数。此时需要直接操作 SysTick 寄存器，操作步骤如下所示：

(1) 设置重装载值。根据时钟频率值进行设置，比如当时钟频率为 50MHz 的时候，需要延时 1ms。重装载寄存器中需要写入的值为 50KHz。

(2) 清空计数器。向当前值寄存器 VAL 中写入 0。

(3) 使能定时器。向 CTRL 寄存中的最低位写入 1。

(4) 等待计时时间到达。读取当前倒计数值，当读取到 CTRL 寄存器的第 16 位为 1 的时候，代表计数时间到达。

(5) 关闭和清空计数器。

根据上述步骤，得到延时函数如下所示（ms 级）：

```
void delay_ms(uint16_t nms)
{
    uint32_t temp;
    SysTick->LOAD = 50000*nms;
    SysTick->VAL=0X00;//清空计数器
    SysTick->CTRL=0X01;//使能定时器
    do{
        temp=SysTick->CTRL;//读取当前倒计数值
    }while((temp&0x01)&&(!(temp&(1<<16))));//等待时间到达
    SysTick->CTRL=0x00; //关闭计数器
    SysTick->VAL =0X00; //清空计数器
}
```

根据上诉实验的描述，得到本次实验 main 函数中的内容如下所示：

```
int main()
{
    while(1){
        GPIO_ResetBit(CM3DS_MPS2_GPIO0,LED0);//点亮 LED0
        delay_ms(1000);
        GPIO_SetBit(CM3DS_MPS2_GPIO0,LED0);//熄灭 LED0
        delay_ms(1000);
    }
}
```

2.10. 板级验证

2.10.1. 实验所需硬件

(1) AC208 开发板

(2) FPGA 下载器：XIST USB Cable

(3) CM3 仿真器：DAP Link

(4) 电源线一根

2.10.2. 硬件连接

硬件连接参考实验一。

2.10.3. 下载文件至目标板

下载文件的方式参考实验一。

2.10.4. 板级调试

本次实验例程中默认使用的时钟频率为 50MHz，将 PLL 输出时钟依次修改为 75M、100M、125M、150M，修改方式参考 2.5.1 一节中的内容，其余部分不用修改，可以观察到的现象是，随着频率不断的增加，LED 灯的闪烁频率不断加快。

2.11. 思考与总结

本章实验主要是讲解了 Cortex-M3 的时钟来源和 PLL 的使用。在接下来所有的实验中将 Cortex-M3 的时钟频率升高至 100M。实验中需要注意的是，在修改 PLL 的参数的时候建议直接从设计管理器中进行修改，而不是从 IP 管理中修改，这是因为在设计管理器中修改时无需关心 IP 存放路径，在 IP 管理中这一点是需要特别注意的。