

AN11043

LPC1100L 电源配置

Rev. 1.1-1/3/2011

应用手册

文档信息

| 信息 | 内容 |
|-----|---|
| 关键字 | 电源 API、ROM、LPC110L、低电流、效率、性能、set_pll、set_power、LPCXpresso |
| 摘要 | 本文档主要用于描述如何使用 LPC1100L 电源配置，以及它们的优点。 |

1、 简介

LPC1100L 电源配置为用户提供了现成的电源管理模板。电源配置适用于任一低功耗应用，设计人员使用最简单的操作就能达到理想的电源等级。电源配置具有类似于无配置低功耗模式的功能，可以进行动态电源管理，且能在不同的应用状态下优化 CPU 操作。此项特性能在低电压下使电流最小，功耗达最小化。为了优化 CPU 性能、CPU 效率和最小有效电流，电源配置可以在整个电压范围 1.8V-3.6V 最大化操作频率，而不用牺牲速率或功能。

Typical Flow of Power Profiles

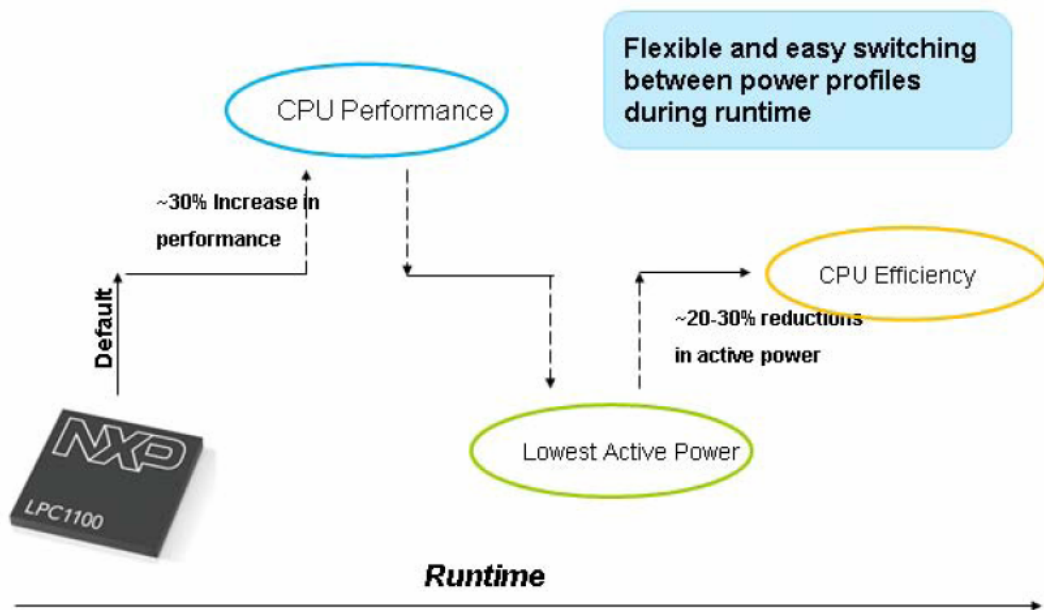


图 1 典型电源配置的流程

低电流模式用于注重低有效电流且保持所需的 CPU 高有效性的应用。在此模式下，CoreMark benchmarks 表现出了功耗降低了 20%-30%的效果。

在 CPU 性能模式，微控制器增加了 CPU 的吞吐量，提供更多的处理性能。CoreMark benchmark 的结果证明了，与常规操作相比，增长了 35%。

效率模式用于平衡 CPU 的代码执行、数据处理、低电流损耗之间的关系。

无论使用哪种电源模式（低电流模式、CPU 性能模式、CPU 效率模式），所消耗的电流

都比默认模式要少。

本文档帮助用户在 LPC1100L 上何时、如何使用电源配置。且，本文档仅仅是个参考。用户自己选择适合自己应用需求的电源配置。

深圳市伟博创科技有限公司

2、 电源配置 API

2. 1 需求

电源配置例程只适用于 LPC1100L (LPC111x/x02) 器件。LPC1100 或 LPC11C00 器件不支持电源配置 API。

2. 2 电源配置 API

电源配置 API 例程是一个简单易用的方式，来配置 LPC1100L 的时钟频率和电源模式。电源配置 API 由两个函数组成：

```
set_pll()
set_power()
```

set_pll API 函数决定并选择 LPC1100L 理想系统 PLL 以及达到所需的系统时钟频率的系统时钟除法器设置。set_pll API 函数免除了用户计算配置合适系统 PLL 所需的乘法和除法器的设置；有效地简化了配置系统时钟的过程。

set_power API 函数用来配置 LPC1100L 4 种不同模式的电源设置：低电流模式、CPU 性能模式、效率模式、默认模式。用户可以通过改变器件的电源配置以满足他们的利益。

本文档有一个配套的示例代码包，在 NXP 网站上。这个代码包同时调用了 set_pll 和 set_power API 例程。相关代码在 power_profiles.c 和 power_profiles.h 文件里，可以作为参考。

这个包的代码利用了电源配置的头文件定义，可以在 power_api.h 头文件找到。power_api.h 头文件在 “..\Common\inc” 目录里。

2. 2. 1 执行电源配置 API

LPC1100L 电源配置 API 函数保存在器件的 ROM 上。调用 API 到 ROM，通过一系列指针，如图 2 所示。

ROM Driver Table 的指针位于 0x1FFF1FF8。ROM Driver Table 中第 4 项包含一个指向电源配置 API 函数表格的指针。电源配置函数表格包含 set_pll 和 set_power 函数。

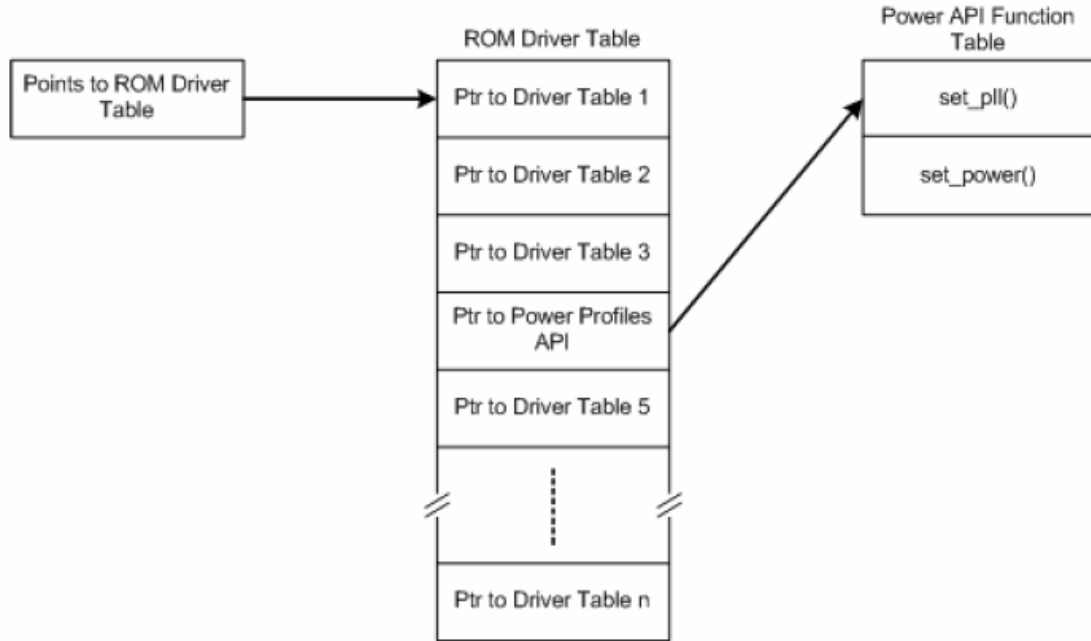


图 2 API 调用 ROM 指针结构

执行电源配置 API 所需的头部定义包含在 power_api.h 头部文件中。这些定义包含 ROM Driver table 和电源配置 API 函数表格，如下所示。

```

1  typedef struct _PWRD {
2      void (*set_pll)(unsigned int cmd[], unsigned int resp[]);
3      void (*set_power)(unsigned int cmd[], unsigned int resp[]);
4  } PWRD;
5
6  typedef struct _ROM {
7      const unsigned p_dev1;
8      const unsigned p_dev2;
9      const unsigned p_dev3;
10     const PWRD * pPWRD;    //Pointer to Power Profiles API function table
11     const unsigned p_dev5;
12     const unsigned p_dev6;
13     const unsigned p_dev7;
14     const unsigned p_dev8;
15 } ROM;
16
17 ROM ** rom = (ROM **) 0x1FFF1FF8;
18 unsigned int command[4], result[2];
    
```

通过一组 command[] 和 result[] 数组简单的调用所需的函数，以操作电源配置 API 函数。这些 32 位无符号整型数组用来提供命令和返回错误代码。

示例调用 set_pll 电源配置函数如下:

```
1  command[0] = 12000;  
2  command[1] = 16500;  
3  command[2] = CPU_FREQ_APPROX;  
4  command[3] = 0;  
5  (*rom)->pWRD->set_pll(command, result);
```

类似的, 调用 set_power 函数如下:

```
1  command[0] = 24;  
2  command[1] = PWR_CPU_EFFICIENCY;  
3  command[2] = 12;  
4  (*rom)->pWRD->set_power(command, result);
```

set_pll 和 set_power 示例, 电源配置 API 调用结果保存在 result[] 数组中。

2. 2. 2 set_pll API 函数

set_pll 函数设置 LPC1100L 的系统时钟。这个函数最大的优点在于用户无需设置系统 PLL。

使用这个函数须确保系统时钟正确设置。本文档中示例软件在 config_pll_power() 函数中演示了这些步骤。

调用 set_pll 之前, 注意 (见图 3):

1. 设置 IRC 或系统振荡器为 sys_pllclk
2. 设置 MAINCLKSEL 为 (MAINCLKSEL = 0x1), 作为主时钟 (main clock)。
 - a. 注意: 你必须在 MAINCLKUEN 寄存器中更新 MAINCLKSEL。
3. 设置系统时钟除法器 (System Clock Divider) 除数为 1 (SYSAHBCLKDIV = 0x1)。

```

1  /**** Required by the Power Profiles set_pll API ****/
2  /*  Switch to the sys_pllclk to the main clock */
3  LPC_SYSCON->MAINCLKSEL = 0x1;
4  LPC_SYSCON->MAINCLKUEN = 0x0;
5  LPC_SYSCON->MAINCLKUEN = 0x1;
6  while ( !(LPC_SYSCON->MAINCLKUEN & 0x01) ); /* Wait until updated */
7
8  /*  Specify AHBCLKDIV = 1 */
9  LPC_SYSCON->SYSAHBCLKDIV = 1;
10 /**** END Required by the Power Profiles API ****/
    
```

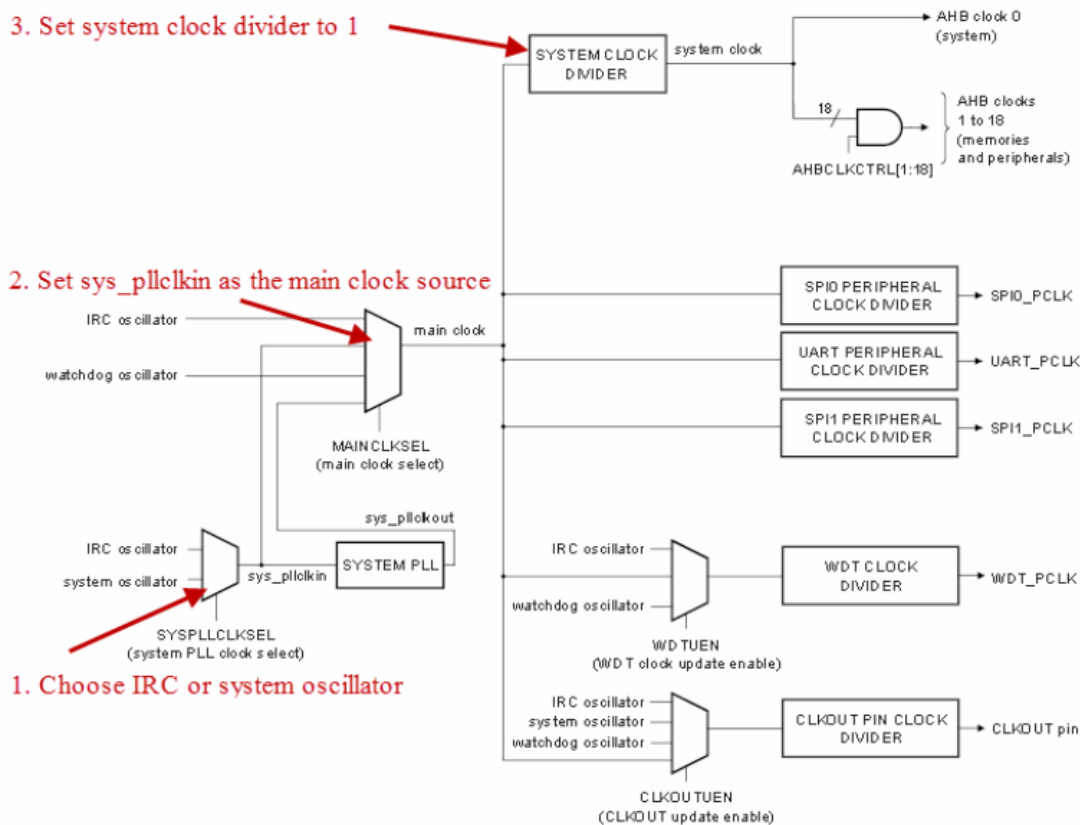


图 3 LPC111x/LPC11C1x 时钟单元

set_pll 的参数如下所示，以及表 1。

```
1  /* set_pll mode options */
2  #define CPU_FREQ_EQU      0
3  #define CPU_FREQ_LTE     1
4  #define CPU_FREQ_GTE     2
5  #define CPU_FREQ_APPROX  3
6
7  /* set_pll result options */
8  #define PLL_CMD_SUCCESS   0
9  #define PLL_INVALID_FREQ 1
10 #define PLL_INVALID_MODE  2
11 #define PLL_FREQ_NOT_FOUND 3
12 #define PLL_NOT_LOCKED    4
```

Table 1. set_pll routine

| Routine | set_pll | Description |
|---------|-------------|---|
| Input | command[0]: | System PLL input frequency in kHz This argument tells the API function at what frequency the IRC/system oscillator is currently clocked at. |
| | command[1]: | Expected system clock in kHz This argument specifies the desired system clock frequency the API should apply. |
| | command[2]: | Mode (CPU_FREQ_EQU CPU_FREQ_LTE CPU_FREQ_GTE CPU_FREQ_APPROX) This mode determines the desired outcome if an exact match of the clock frequency requested cannot be found. A call specifying CPU_FREQ_EQU will only succeed if the PLL can output exactly the frequency requested in command[1]. CPU_FREQ_LTE can be used if the requested frequency should not be exceeded (such as overall current consumption and/or power budget reasons). CPU_FREQ_GTE helps applications that need a minimum level of CPU processing capabilities. CPU_FREQ_APPROX results in a system clock that is as close as possible to the requested value (it may be greater than or less than the requested value). |
| | command[3]: | System PLL lock timeout This parameter specifies a timeout on how many polling cycles the API function will wait until a PLL has been achieved. This assumes of course that the desired system clock frequency should require the PLL. Setting this parameter to the desired system frequency in Hz divided by 10000 should be enough time for PLL lock to take place. |
| Result | result[0]: | PLL_CMD_SUCCESS PLL_INVALID_FREQ PLL_INVALID_MODE PLL_FREQ_NOT_FOUND PLL_NOT_LOCKED A call to the set_pll routine will result with a status code, indicating a successful routine call or some type of error. |
| | result[1]: | System clock in kHz If the set_pll function doesn't return any errors, this return value indicates at what frequency the system clock was set to. |

表 1

2. 2. 3 set_power API 函数

set_power 函数设置内部电源控制的选项以匹配所需要的类型。注意，这个函数并不改变系统时钟。它只需要当前以及新系统时钟参数以决定器件继续安全运行所需要等待的时间。调用这个函数，用户可以选择 LPC1100L 的 4 种不同运行模式：

1. 默认模式
2. 低电流模式
3. 性能模式
4. 效率模式

每一种不同的配置都是为了优化器件以满足不同应用的需求。无论如何，低电流模式、性能模式、效率模式总比默认模式的损耗更低。每一种电源配置模式如表 2 所示。

Table 2. Power profile modes

| Power profile | Description |
|------------------|---|
| Low Current mode | Low Current mode is designed for applications where minimizing current consumption has higher priority than CPU execution time. |
| Performance mode | Performance mode is intended for applications where it is desired to have the fastest CPU execution; regardless of the current consumption. |
| Efficiency mode | Efficiency mode is intended as a general purpose mode where a balance between performance and low current consumption is needed. |
| Default mode | The device is in Default mode after the device performs a reset. |

表 2

set_power 的参数如下，以及表 3 所示。

```

1  /* set_power mode options */
2  #define PWR_DEFAULT          0
3  #define PWR_CPU_PERFORMANCE 1
4  #define PWR_EFFICIENCY      2
5  #define PWR_LOW_CURRENT     3
6
7  /* set_power result options */
8  #define PWR_CMD_SUCCESS      0
9  #define PWR_INVALID_FREQ    1
10 #define PWR_INVALID_MODE    2

```

Table 3. set_power routine

| Routine | set_power | Description |
|---------|-------------|---|
| Input | command[0]: | New system clock in MHz This argument is used to inform the API function to what system clock frequency the LPC1100L should be configured for. |
| | command[1]: | Mode (PWR_DEFAULT PWR_CPU_PERFORMANCE PWR_EFFICIENCY PWR_LOW_CURRENT) This mode indicates which power profile should be enabled. PWR_LOW_CURRENT is intended for those solutions that focus on lowering power consumption rather than CPU performance. PWR_CPU_PERFORMANCE configures the microcontroller so that it can provide more processing capability to the application. CPU performance is 30% better than the default option. PWR_EFFICIENCY setting was designed as a general purpose mode with a focus of providing a balance between execution time and low current consumption. PWR_DEFAULT keeps the device in a baseline power setting similar to its reset state. This mode must be used for In-Application-Programming (IAP) API routines. |
| | command[2]: | Current system clock in MHz This parameter specifies the current clock rate of the LPC1100L when set_power is called. If set_pll was executed prior to calling set_power, this argument should be set to the system clock frequency used before the set_pll call. |
| Result | result[0]: | PWR_CMD_SUCCESS PWR_INVALID_FREQ PWR_INVALID_MODE A call to the set_power routine will result with a status code, indicating a successful routine call or some type of error. |

表 3

LPC1100L 用户手册中有一个框图，描述了如何使用 set_pll 和 set_power API 函数，见图 4。这个框图很有帮助，如果有人只想调用应用中所用到的一部分函数。值得注意的是，如果调用 set_power 之后，立即调用 set_pll，那么，只需要 50 μs 的延时。

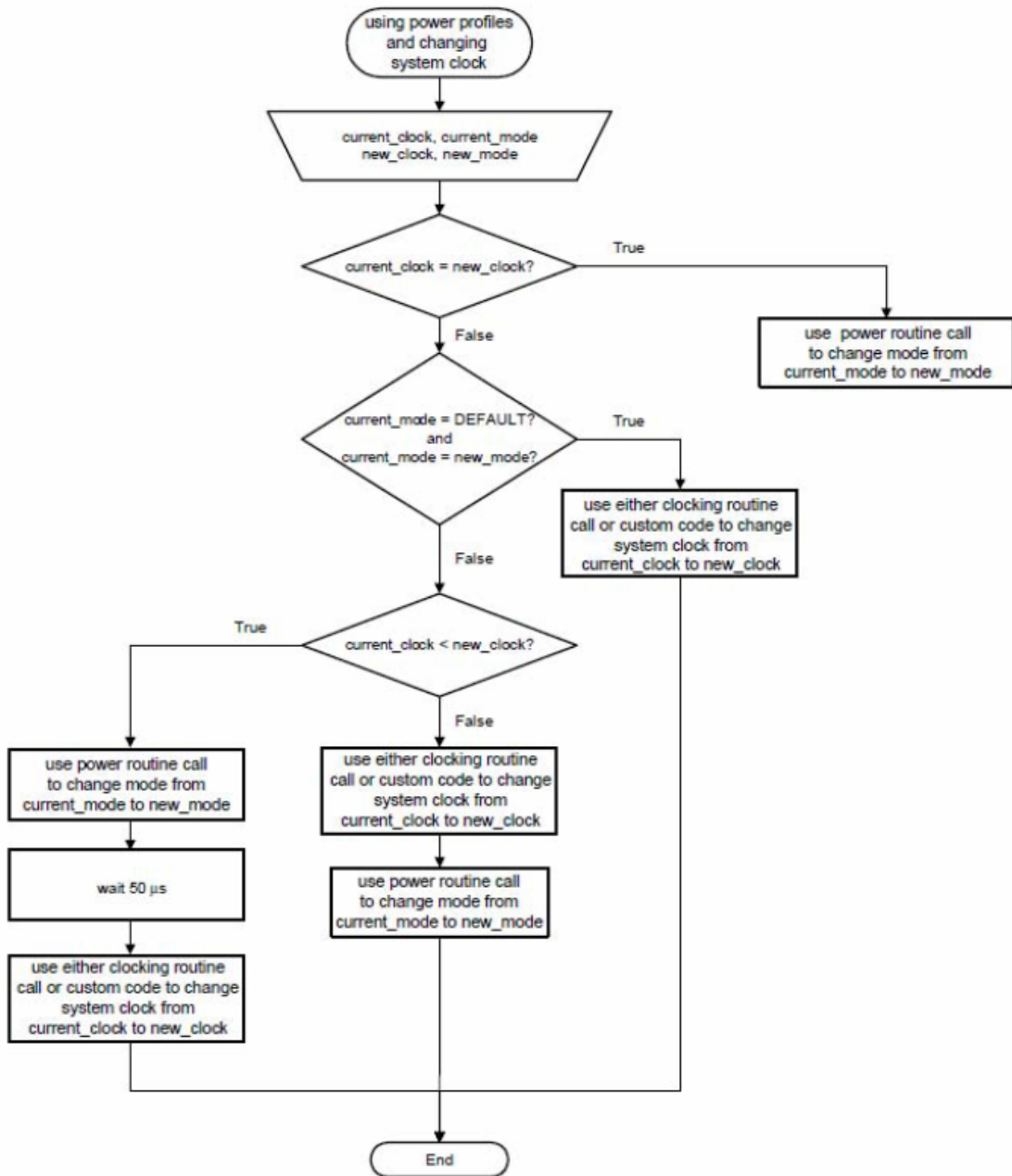


图 4 电源配置流程图（来自 LPC1100L 用户手册）

图 4 较为详细，有另一种更简单的方法。图 5 为简化的流程图。这个版本可用于一般的多用途电源配置函数调用，同时处理了 set_pll 和 set_power。

本文档提供的示例工程使用了简化的版本。这个方法总是先让 LPC1100L 进入默认状态，设置新的系统时钟和电源模式。

简化的流程：

1. 使用 set_power 设置器件为默认模式，50MHz。
2. 调用 set_pll 设置所要的系统时钟频率。
3. 再次使用 set_power 来配置器件的电源模式。

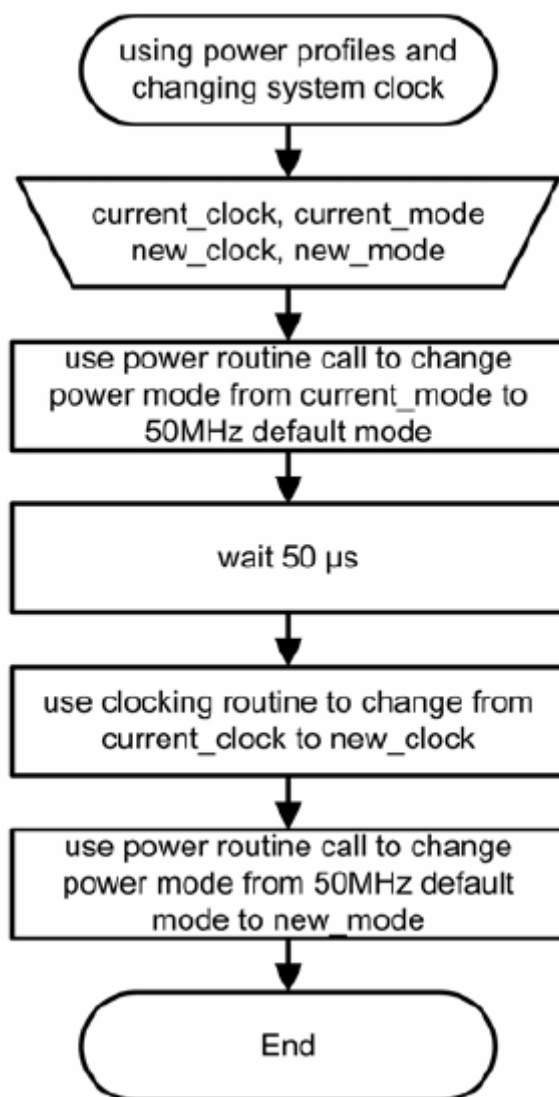


图 5 简化的电源配置使用流程框图

3、 应用示例

这个章节用程序示例演示了电源配置的特性。这个应用示例分别在两个不同的工程中演示 set_pll 和 set_power 的特性。两个演示都利用了一个 UART 用户接口和 LED。只有 set_power 演示有电流测试。

3. 1 开发板

本文档的软件（代码）可用于 LPCXpresso 和 μ Vision4 工具链和开发板。

3. 1. 1 LPCXpresso

LPCXpresso开发软件可在这里找到：<http://lpcxpresso.code-red-tech.com>
LPCXpresso 板（图 6a），建议配合 Embedded Artists' LPCXpresso 底板（图 6b）进行 UART 通信。

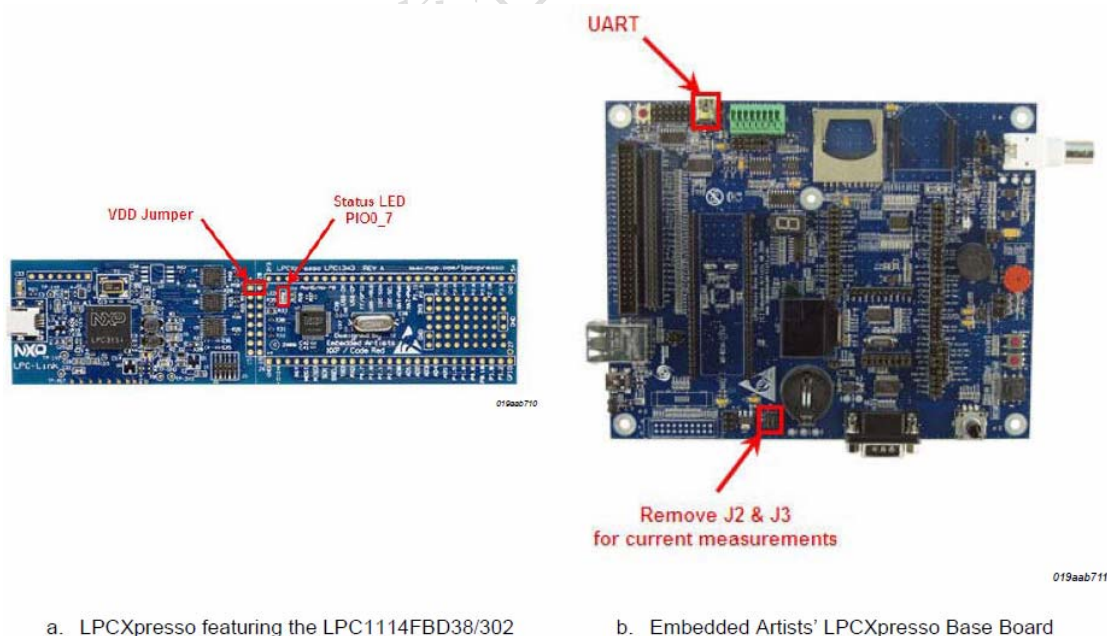


图 6

3. 1. 1. 1 LPCXpresso UART 用户接口

本文档的所有示例应用都利用 LPC1100L 的 UART 作为用户接口 (UI) 来控制器件功能。UART 配置为 57600-N-1。推荐使用 Tera Term Pro 系列终端 (支持 VT100)。

Tera Term Pro 的下载地址: <http://en.sourceforge.jp/projects/ttssh2/releases/>

LPCXpresso 板本身不包含 UART-RS232 转换。因此, 推荐使用 Embedded Artists' LPCXpresso 底板。这块板使用 UART-USB 转换器进行 UART 通信。

3. 1. 1. 3 LPCXpresso 状态 LED

一个 LED 连接到 PIO0_7, 作为状态指示灯。它在软件计数器内每 0x10000 触发一次。这个 LED 在 set_pll 和 set_power 示例演示中都会用到。

在 set_pll 演示中, 这个 LED 给用户相关系统时钟频率简单的视觉批示。系统时钟频率越高, LED 闪烁越快。

在 set_power 演示中, 这个 LED 指示, LPC1100L 处于“就绪”作态, 等待 UART UI 的用户输入。

3. 1. 1. 3 LPCXpresso 电流测试

应用示例, 演示了 set_power API 功能, 可以测量电流, 在 V_{DD} 跳线处插入安培计, 见图 6a。

测量微控制器电流, 在 Embedded Artists' LPCXpresso 底板上移去 J2 和 J3 (见图 6b), 使用一个外部 3.3V 电源与安培计串联。

set_power 示例软件, 改变了 I/O 引脚的状态, 最小化电流的损耗。

3. 1. 2 Keil's MCB1000

Keil's μ Vision4 评估版可以用来编译电源配置应用示例。Keil 也提供 MCB1000 开发板, 见图 7。烧写 LCP1114/302 可以使用 Keil's uLink JTAG/SW debugger 或 Flash Magic。

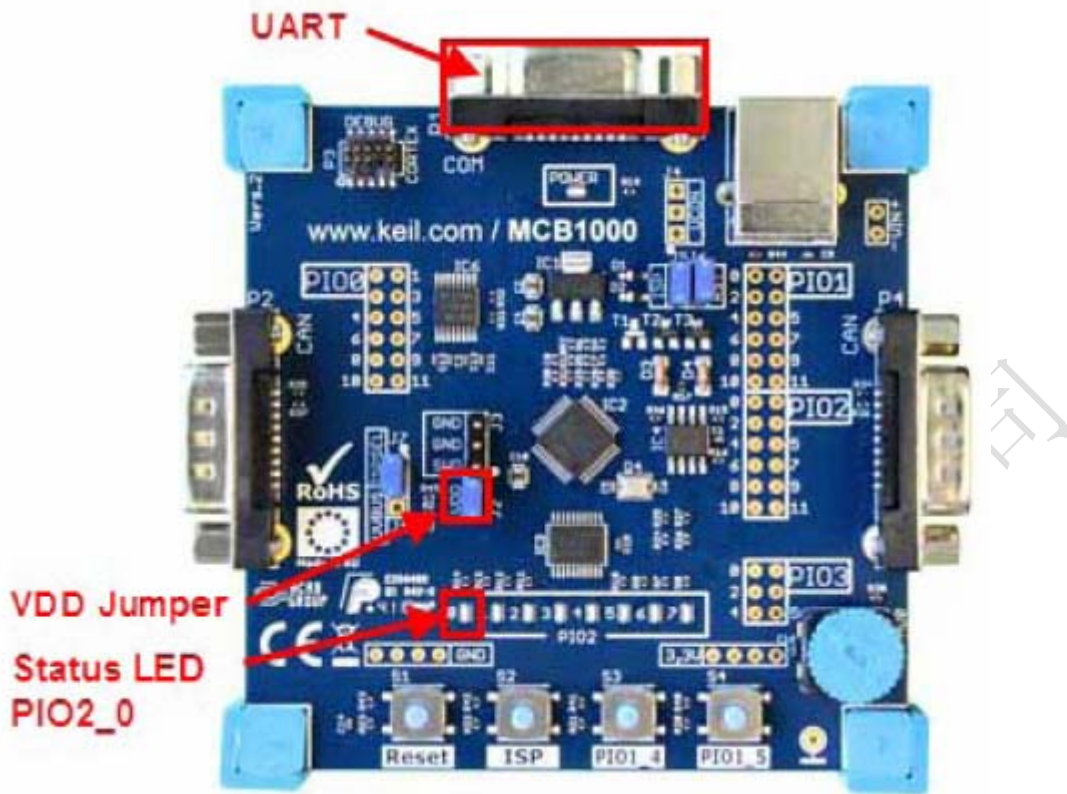


图 7

3. 1. 2. 1 MCB1000 UART 用户接口

本文档的示例应用利用 LPC1100L 的 UART，作为用户接口 (UI)，来控制器件的功能。UART 配置为 57600-N-1。推荐使用 Tera Term Pro 系列终端 (带 VT100)。

Tera Term Pro 下载地址: <http://en.sourceforge.jp/projects/ttssh2/releases/>

3. 1. 2. 2 MCB1000 状态 LED

一个 LED 连接到 PIO2_0，作为状态指示灯。它在软件计数器内每 0x10000 触发一次。LED 在 set_pll 和 set_power 示例演示中都用到。

在 set_pll 演示中，LED 给用户相关系统时钟速率的简单指示。系统时钟频率越高，LED 闪烁越快。

在 set_power 演示中，LDE 指示 LPC1100L 处于“就绪”状态，等待 UART UI 用户输入。

3. 1. 2. 3 MCB1000 电流测量

演示 set_power API 功能的应用示例，可以测量电流，在 V_{DD} 跳线 (J2) 插入安培计，见图 7。V_{DD} 的头部引脚位于 J3 旁边，可以用来测量电流。

测量微控制器的电流，推荐使用外部 3.3V 电源串联安培计。

set_power 软件示例，改变了 I/O 引脚的状态，以最小化电流损耗。

3. 2 电源配置演示

本文档提供的软件分别演示了 set_pll 和 set_power API 例程。在 LPCXpresso 和 Keil's μ Vision 环境下，这两个演示效果不一样。

3. 2. 1 LPCXPRESSO 工程

使用 LPCXPRESSO，高亮想要的演示工程，点击“Debug”，如图 8。

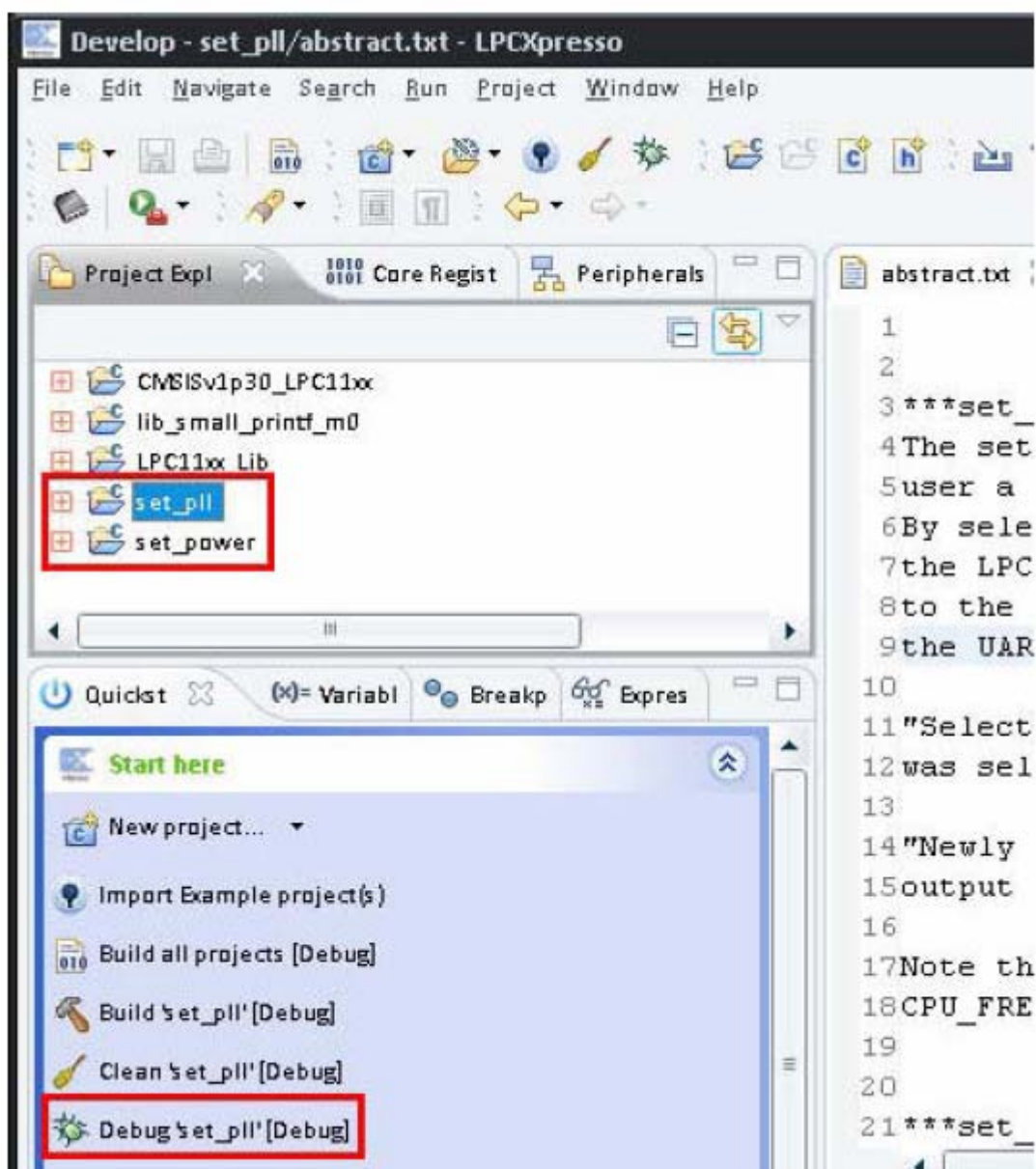


图 8 建立/调试演示工程

3. 2. 2 Keil's μ Vision 工程

在 μ Vision 环境下，就容易得多了。见图 9。

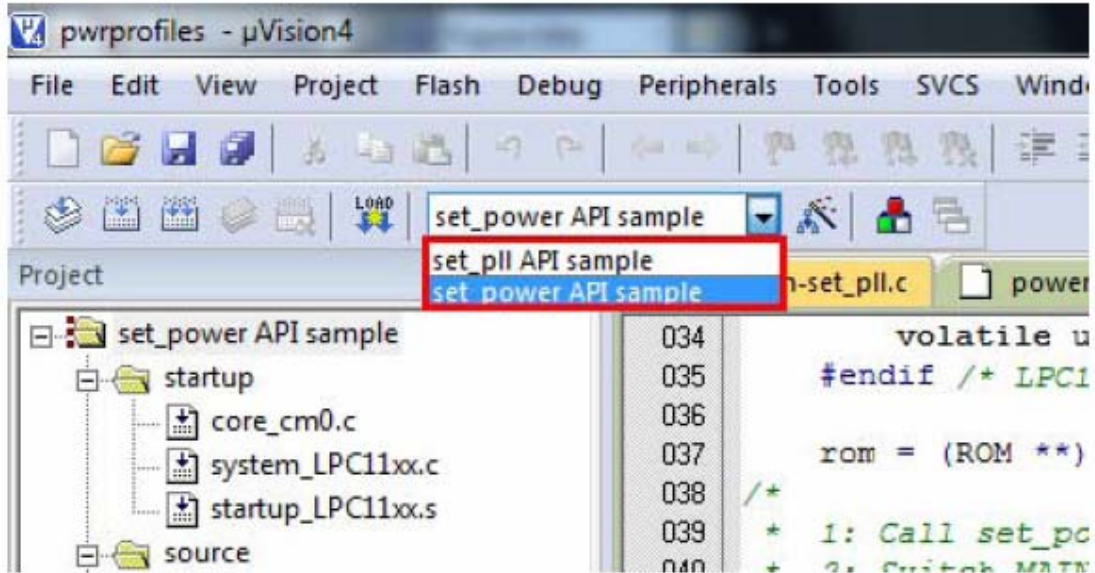


图 9

选择好目标文件后，“Rebuild”所有的目标文件。
Project → Rebuild all target files

3. 2. 3 set_pll 演示

set_pll 电源配置 API 演示，展现了通过调用 API 例程来配置 LPC1100L 的系统时钟是如何的简单。

系统时钟的改变可以在 UART UI 和状态指示 LED 中观察到。

3. 2. 4 set_power 演示

set_power 配置 API 演示，展现了以下几个特性：

1. 易用
2. 降低电流损耗
3. 改进了 CPU 性能

为了演示电流损耗的降低以及性能的改进，这个应用示例重复从静态库调用了常量运行时间除法例程（constant runtime divide routine）。当 LPC1100L 调用了 5000000 次除法库，器件的电流损耗可以从外部的安培计测量得到。而且，使用 32 位 Time 0，我们也可以测量完成 5000000 次除法操作所需要的时间，作为一个性能指标。

获得这些电流损耗以及时间，我们就可以利用这些指标比较 4 种不同的模式。
表 4 简单概述了低电流模式、效率模式、性能模式与默认模式比较的结果。

表 4 与默认模式的比较结果

Calling 5000000 library division calls

| Profile Comparison vs. Default Mode | Current Consumption | Execution Time |
|-------------------------------------|---------------------|--------------------|
| Low Current mode | Reduced | Increased |
| Efficiency mode ¹ | Reduced | Equal or Increased |
| Performance mode | Reduced | Reduced |

set_power 演示的结果可以参看 3.4.1 章节的图。

3. 2. 5 set_pll 和 set_power 示例的配置选项

以下选项可以灵活配置：

```

1  /* Configuration options */
2  #define UART_BAUD          57600
3  #define ENABLE_CLKOUT     0
4  #define MAIN_OSC          1
    
```

“UART_BAUD” 定义了 UART UI 的波特率。

“ENABLE_CLKOUT” 使能或禁止主时钟信号进入 CLKOUT pin (PIO0_1)。为保持电流损耗最小，这个需为 ‘0’。

“MAIN_OSC” 让用户选择主振荡器还是 IRC 振荡器作为 sys_pllclk 的时钟源。见图 3。

3. 3 set_pll 应用示例

set_pll 应用示例为用户展示了 UART UI 从列表中选择系统时钟频率，见图 10。

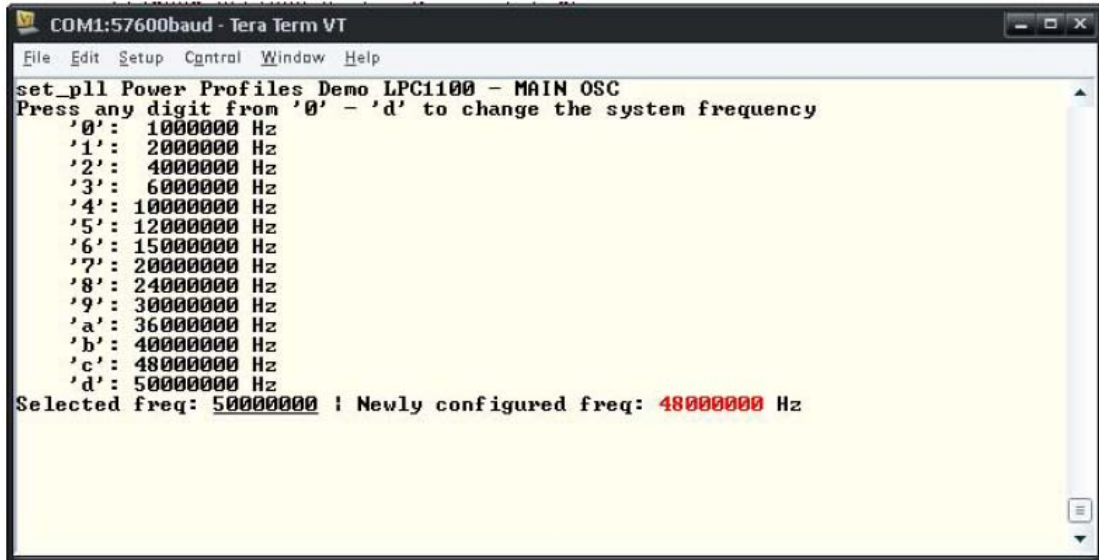


图 10 set_pll 工程菜单

选择菜单上显示的任一选项('0' - 'd')，LPC1100L 将调用 set_pll API 函数，改变系统时钟频率，重新配置 UART。

“Selected freq:” 显示从菜单中选择的频率选项。

“Newly configured freq:” 显示 set_pll 产生的实际频率。

注意：set_pll 已用 CPU_FREQ_APPROX 参数编码。

3. 4 set_power 应用示例

set_power 应用示例也为用户演示了 UART UI。在这里，用户只能选择哪一种 set_power 电源配置测试。图 11 为用户做了几个电源配置选项的屏幕。

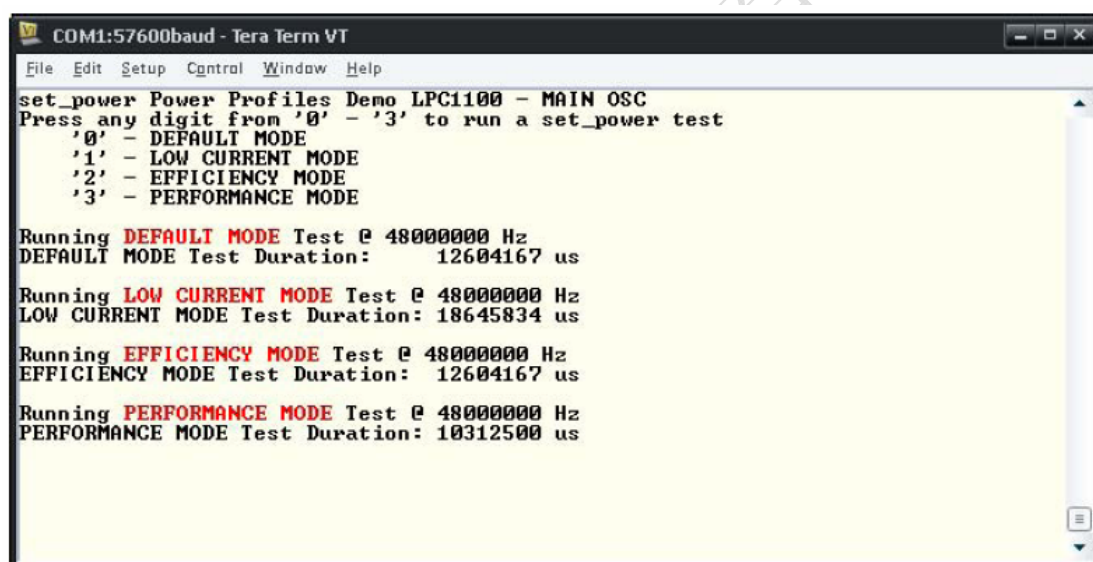
电源配置测试将：

1. 配置 LPC1100L 所选的电源模式
2. 禁止 UART 中断，防止测试期间的不可预知中断
3. MCB1000 所有的 I/O 都设为低电流损耗状态
4. 启动 32-bit Timer 0
5. 使用除法库执行 5000000 次除法
在完成 5000000 次除法之前，从安培计获取电源损耗
6. 完成 5000000 次除法之后，停止 Timer 0
7. 使能 I/O 的正常操作

8. 使能 UART 中断
9. 显示 5000000 次除法的时间 (μ s)

为保持工程的简洁性，main-set_power.c 文件中系统频率只在编译过程中才能改变。系统时钟频率取决于 SYS_FREQ 预处理器的定义。只要简单的选择 FREQ 的定义，就可以选择系统频率。

```
1  #define FREQ          5000000UL
2  //#define FREQ       3600000UL
3  //#define FREQ       3000000UL
4  //#define FREQ       1200000UL
5  //#define FREQ       600000UL
6
7  #define SYS_FREQ      FREQ
```



```
COM1:57600baud - Tera Term VT
File Edit Setup Control Window Help
set_power Power Profiles Demo LPC1100 - MAIN OSC
Press any digit from '0' - '3' to run a set_power test
'0' - DEFAULT MODE
'1' - LOW CURRENT MODE
'2' - EFFICIENCY MODE
'3' - PERFORMANCE MODE

Running DEFAULT MODE Test @ 48000000 Hz
DEFAULT MODE Test Duration: 12604167 us

Running LOW CURRENT MODE Test @ 48000000 Hz
LOW CURRENT MODE Test Duration: 18645834 us

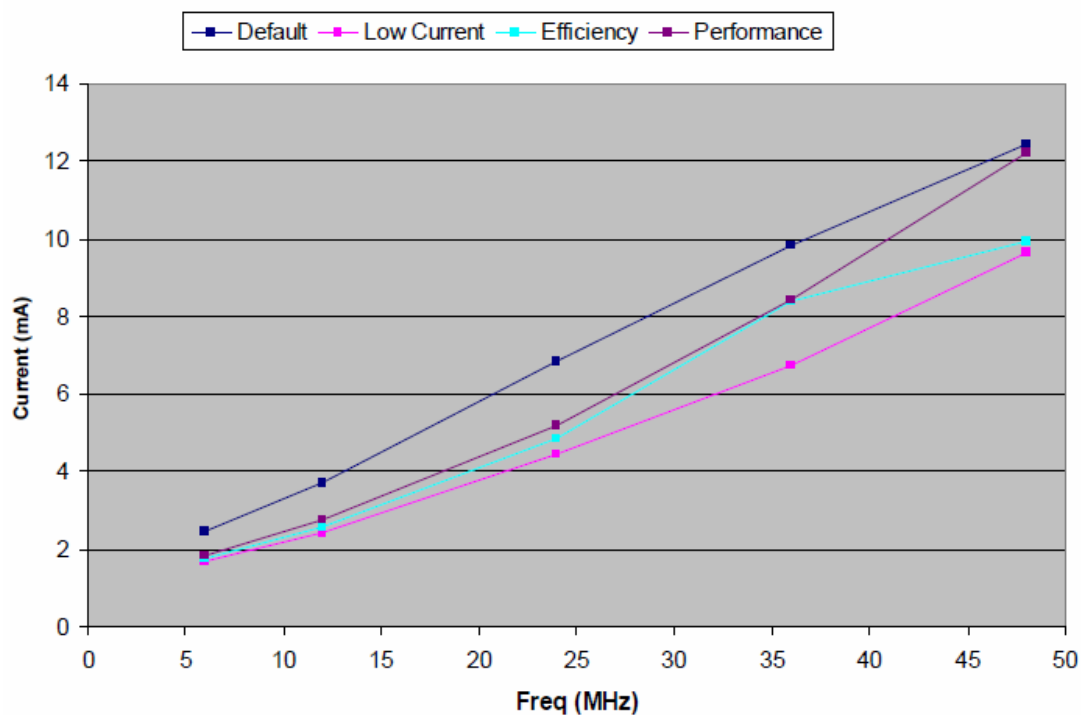
Running EFFICIENCY MODE Test @ 48000000 Hz
EFFICIENCY MODE Test Duration: 12604167 us

Running PERFORMANCE MODE Test @ 48000000 Hz
PERFORMANCE MODE Test Duration: 10312500 us
```

图 11 set_power 工程菜单

3. 4. 1 示例电流损耗和性能比较结果

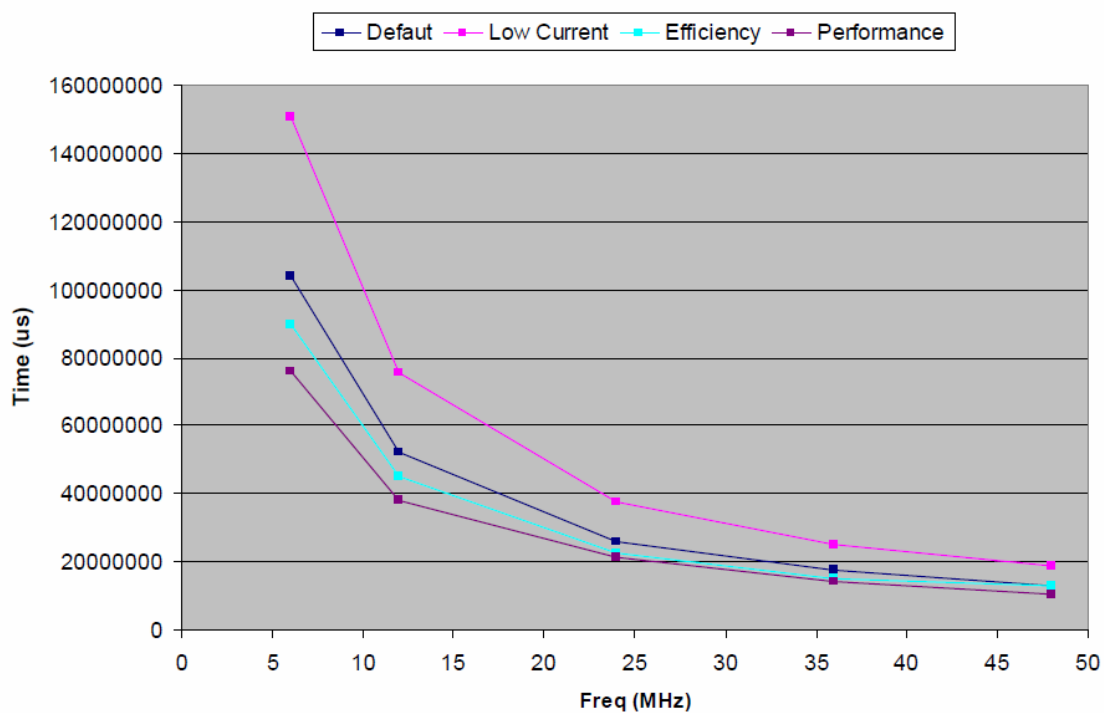
5000000 Divisions: Current consumption (mA) vs Freq (MHz)



- (1) The result depicted in this chart applies to 5000000 "division" function calls from a sample library.
- (2) Results will vary depending on the application.
- (3) Clock source: system oscillator
- (4) No code optimization.

图 12 使用 set_power 减小电流损耗的结果

5000000 Divisions: Process duration (us) vs Freq (MHz)



- (1) The result depicted in this chart applies to 5000000 "division" function calls from a sample library.
- (5) Results will vary depending on the application.
- (6) Clock source: system oscillator
- (7) No code optimization.

图 13 使用 set_power 来提高性能的结果

深圳市伟博创科技

4、结论

电源配置让用户可以轻易地动态改变系统时钟频率，而无需重新配置系统 PLL。它同时提供一种机制，来优化 LPC1100L，最小化电流、最大化 CPU 性能，或平衡这两者。

低电流模式是为了最小化电流损耗，而性能模式是为了达到最短的执行时间。效率模式是为了达到电流损耗和执行时间之间的平衡。无论如何，这三种模式都比默认模式更节省电流。