AN10795

LPC29xx Power-down mode explained

Rev. 02 — 25 February 2009 Application note

Document information

Info	Content
Keywords	LPC29xx, PMU, CGU, Power-down mode
Abstract	This Application Note describes the techniques necessary to place the LPC29xx device into Power-down mode

Revision history

Rev	Date	Description
02	20090225	Made various edits throughout.
01	20090210	First release

Contact information

For additional information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

The LPC29xx incorporates a sophisticated and complex power management system. In this application note we will describe how to use the power management unit to place the MCU into a power down (PD) state while maintaining the ability to respond to its environment.

2. Hardware

The Keil MCB2900 version 1 evaluation board with a LPC2919/01 is used in this application note. The board switches connected to external interrupt 4 and 5 are used as inputs. The external interrupts can be changed to other ports by modifying a ‘define’ in the code, so other board can also be used. Software configuration of the hardware will be discussed in detail in the software chapter.

By removing 0 Ω resistor R6 from the MCB2900 it is possible to measure the supply current of the VDD (1v8) domain. The VDDE (3v3) domain isn’t taken in account because this primarily depends on the environment of the device. The current readings mentioned in this application note are numbers for a single device at room temperature and do not represent minimum or maximum guarantees. For detailed specification refer to the datasheet.

3. Software

3.1 IDE

The Code supplied was developed with Keil’s uVision MDK V3.24. This tool has a convenient feature to place functions in volatile memory. It is necessary to place some code in volatile memory, as will be explained in paragraph 4.1.6.

3.2 Configuring the code

The program uses two external interrupts to exit from wakeup and sleep modes. The ‘extint.h’file configures which pins are connected to the external interrupts and which interrupt is used. External interrupts zero to seven can be defined. For the MCB2900, the push buttons P1.30 and P1.31 are connected to external interrupt 4 and 5.

```
#define EXINT_sleep 4
#define EXINT_wake 5
```

The user also has the option of lowering power consumption at the cost of a slightly longer recovery time. Details about this are presented in paragraph 4.3.

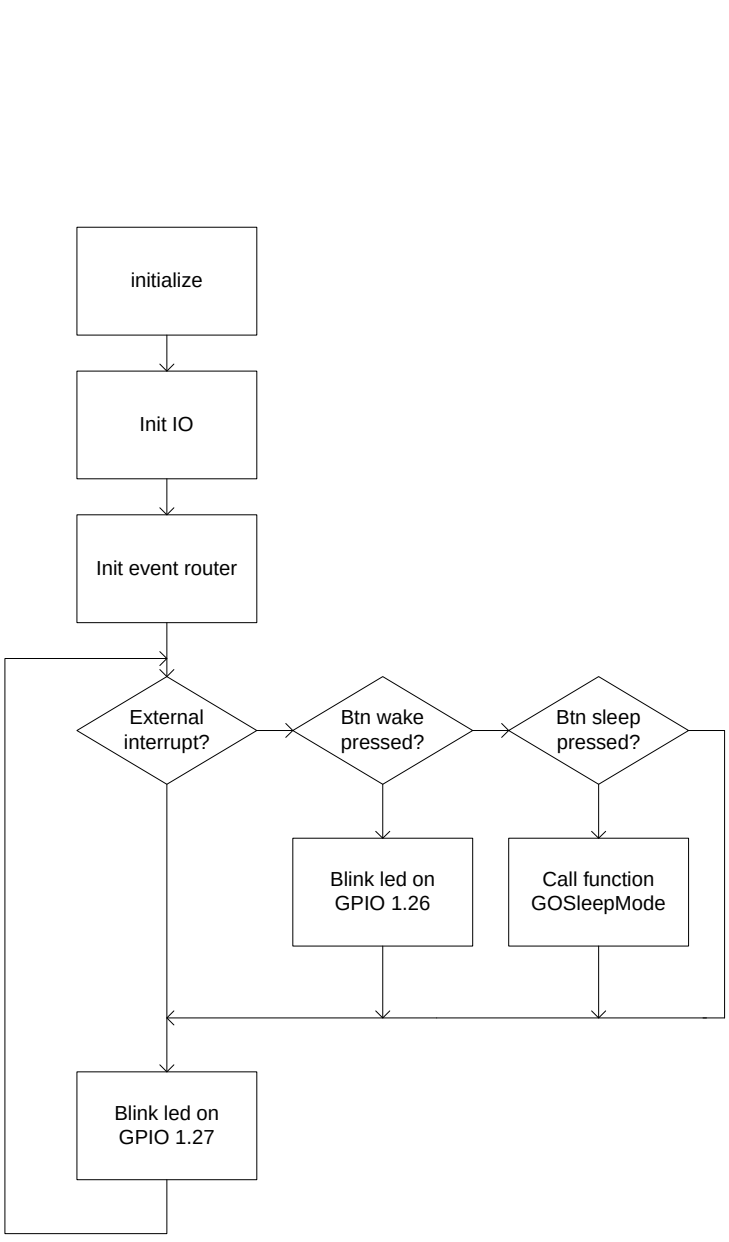
```
#define SWITCH_CLOCKS_TO_XTAL 0
```

3.3 Compiling and flashing the code

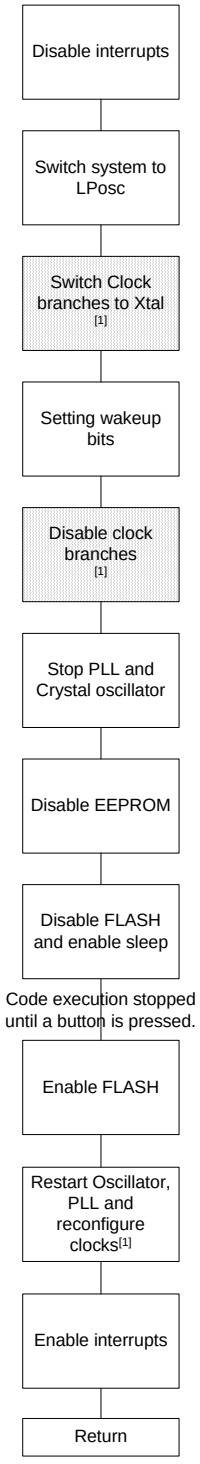
The project is setup to be run from the LPC2900 internal FLASH. After compiling, the code can be downloaded (flashed) and run. Debugging is only possible before entering the powerdown (PD) mode since all clocks, including the JTAG clock, are disabled in PD mode. The connection between the MCU and debugger is lost in PD mode and can only be restored by a reset.

4. Program description

The main program blinks an LED connected to P1.27 while waiting for a button press on P1.30 (defined as the wakeup button) or P1.31 (defined as the sleep button). The external interrupt handler sets a flag when the wakeup or sleep button is pressed and this flag is polled in the main program loop. If the wakeup button (P1.30) was pressed, an LED is toggled and the MCU will wake up, if in Power Down mode. If the sleep button (P1.31) was pressed, the software calls the function '`GOSleepMode()`' which handles the clock switching and PMU settings to cause the MCU to enter PD state. The status of the system is indicated on the LCD display.



a. Main program



b. Function GOSleepMode

[1] Implementation depends on configuration see §3.2

Fig 1. Program flow diagram

4.1 Entering sleep mode

Some preparation is needed to setup the wakeup source and prepare the MCU to enter PD. In this paragraph each action required is explained in some detail.

4.1.1 Disable interrupts

Before disabling any part of the MCU it is necessary to disable the interrupts in the core. This requirement ensures there are no read or writes in an interrupt handler to peripherals or memory that is disabled. If this happens, an exception is raised. The interrupt is re-enabled when the wakeup sequence is completed.

```
__disable_irq();
__disable_fiq();
```

4.1.2 Switch system to LPosc

After disabling the interrupts, the system clock should be switched to the LPosc (Low Power Oscillator) which runs at 475 KHz nominal. The LPosc cannot be switched off, thereby ensuring that the main system will have a clock when we are waking up. The LPosc could be engaged later, but care must be taken to not disable the clock to the core while running from it. Although the core will automatically switch to another clock source if the clock is disabled, it is preferable for the code to be in full control of the core clock.

Switching to the LPosc is performed by setting the corresponding bits in the 'SYS_CLK_CONF' register.

```
SYS_CLK_CONF = AUTOBLK | DIV1;
```

4.1.3 Setting the wakeup bits

By enabling the Wakeup bit in the 'PMU_CLK_CFG_xxx' register, the clock to the specified clock leaf will be disabled when the power down bit in the 'PMU_PM' register is set. By setting the PD bit, all clocks with the wakeup bit set will be turned off at the same time. When a wakeup event occurs, the power down bit is cleared and all peripherals will be clocked again. To access the 'PMU_CLK_CFG_xxx' register, it is necessary to have a clock in the specific clock leaf. Checking if a clock is available can be done with the 'CGU_xxx_STAT' register.

The process described above is the only method to disable the clock to the core, because the RUN bit in the clock configuration register cannot be cleared. Setting the PD bit will be the final action before entering sleep mode. We will describe how and when to do this later.

4.1.4 Stop PLL and crystal oscillator

The PLL's and the main crystal oscillator no longer needed, so it is now safe to stop them.

```
CGU1_PLL_CTRL = 1; //Power Down CGU1 PLL
CGU_PLL_CTRL = 1;  //Power Down CGU PLL
// Power down OSC pads, still leave HF bit as default.
CGU_OSC_CTRL = (0x1<<2) | (0x0<<0);
```

4.1.5 Disable the EEPROM

The EEPROM needs to be disabled next, however, the EEPROM controller is a part of the FLASH controller and uses the same clock. The EEPROM memory can be seen as a separate device and needs to be disabled.

```
EEPWRDWN =1;  // Power down the EEPROM controller
```

4.1.6 Enable Sleep mode

The last actions needed to go to PD are disabling the flash and finally setting the PD bit in the PMU.

Since the FLASH is being disabled, the code to disable flash cannot run from FLASH. Hence the 'disabling' code is placed in TCM. To achieve this some special features of the Keil uVision tool are used. This will copy the code from FLASH to a specified memory location just before branching to 'main'. The function '`sleepFlashOff()`' which does the work is placed in a separate file 'flashoff.c'. In 'Options for File' the TCM memory area is selected, uVision will handle the copying of the code.

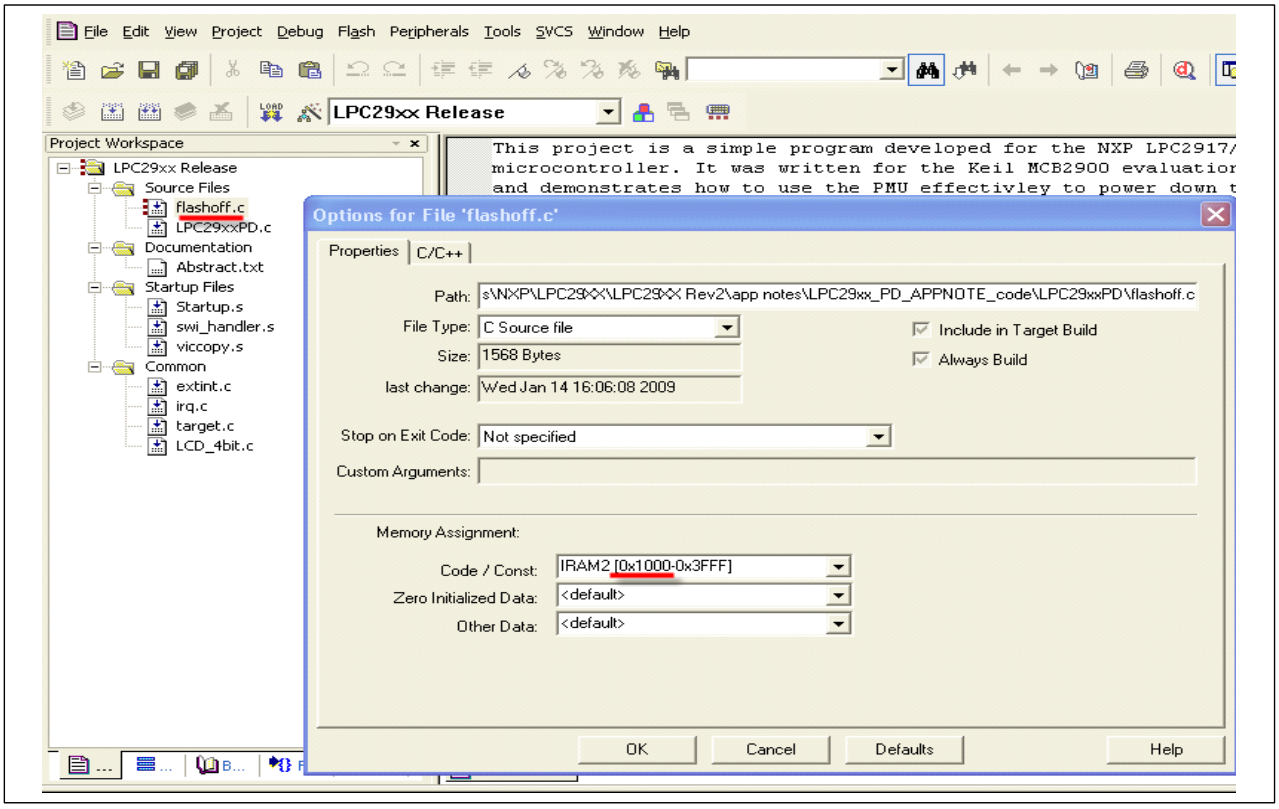


Fig 2. Disabling flash

In '`sleepFlashOff()`' firstly the flash controller is disabled and then the PD bit in the PMU_PM register is set. At the moment the PD bit is set the clock to all clock leafs with wakeup set are disabled, including the core clock, so no more code is executed.

```
FCTR |=0x200; // Power down FLASH
PMU_PM =0x1; // set the PD bit in the PMU
```

A few blocks of the device are still working; the LPosc cannot be disabled and thus is still running. If the Watchdog timer is enabled it will continue to do so, because it is running from the LPosc.

4.2 Waking up

The wakeup switch can be pressed to wake the device,. which will trigger the wakeup circuit through the event router. This clears the PMU PD bit to re-enable all devices with

wakeup enabled. Some actions are required to get the controller fully up and running again..

4.2.1 Re-enabling FLASH

The code had stopped execution with the 'sleepFlashOff()' function and will restart execution after the PMU PD instruction. The code that puts the device to sleep is located in the TCM, and will execute as soon as the PD bit is cleared. The remainder of the code is still located in flash, so before we can proceed we need to re-enable the FLASH.

```
FCTR &=~0x200;
```

Now that the FLASH is accessible, we can branch back and wakeup the other parts of the MCU.

4.2.2 Starting the clock source

All peripherals with the wakeup bit set are now active again. If the clock source for the peripheral is the LPosc, then the peripheral will be working properly. . If the peripheral clock source is the Xtal or the PLL, then those sources must be reactivated before the peripherals will resume operation.

First we need to restart the crystal oscillator and wait for it to startup before starting the PLL, as it is the clock source for the PLL.

We will then wait for the PLL to lock and output clock to stabilize before switching over the system clock from the LPosc to the PLL.

```
CGU_OSC_CTRL |= (0x1<<2) | (0x1<<0); // re-enable the osc
while ( !(CGU_RDET & XTAL_PRESENT) ){ NOP; } // wait for osc to start up

// Restart PLL
CGU_PLL_CTRL = PLL_XTAL_SEL | (PLL_M_VALUE<<MSEL_SHIFT) | P23EN;

// Check lock bit, if unlocked, PLL_LOCK is always 0
while ( !(CGU_PLL_STAT & PLL_LOCK) ) { NOP; }
// Check clock detection register to make sure PLL is present now.
while ( !(CGU_RDET & PLL_PRESENT) ) { NOP; }

CGU_PLL_CTRL = PLL_XTAL_SEL | (PLL_M_VALUE<<MSEL_SHIFT) | P23EN;

// PLL is 250Mhz, SYS_CLK and TMR_CLK is 125Mhz. This line depends on the board
// and configuration of clocks used see TargetResetInit() in target.c
SYS_CLK_CONF = CLK_SEL_PLL | AUTOBLK | DIV2;
```

4.2.3 Enabling interrupts

Now that all systems and subsystems are working again , it's safe to re-enable the interrupts without causing any exceptions. Any pending interrupts in the VIC will be handled now and normal operation will continue.

```
__enable_fiq();
__enable_irq();
```

4.3 Additional power saving

By understanding the architecture of the clock tree and specifically that of the CGU, the user can save an additional 20 – 30 uA.. The clock to each peripheral is disconnected by the clock switches of the PMU and are located at the output of the CGU . Depending on the selected clock source, the logic in the CGU is still working even when the crystal oscillator and the PLL's are disabled. If the LPosc was selected it will continue to run and keep clocking the clock path inside the CGU through to the clock switch of the PMU. To save power, the peripheral clock needs to be switched to a clock source that can be disabled, namely the crystal oscillator..

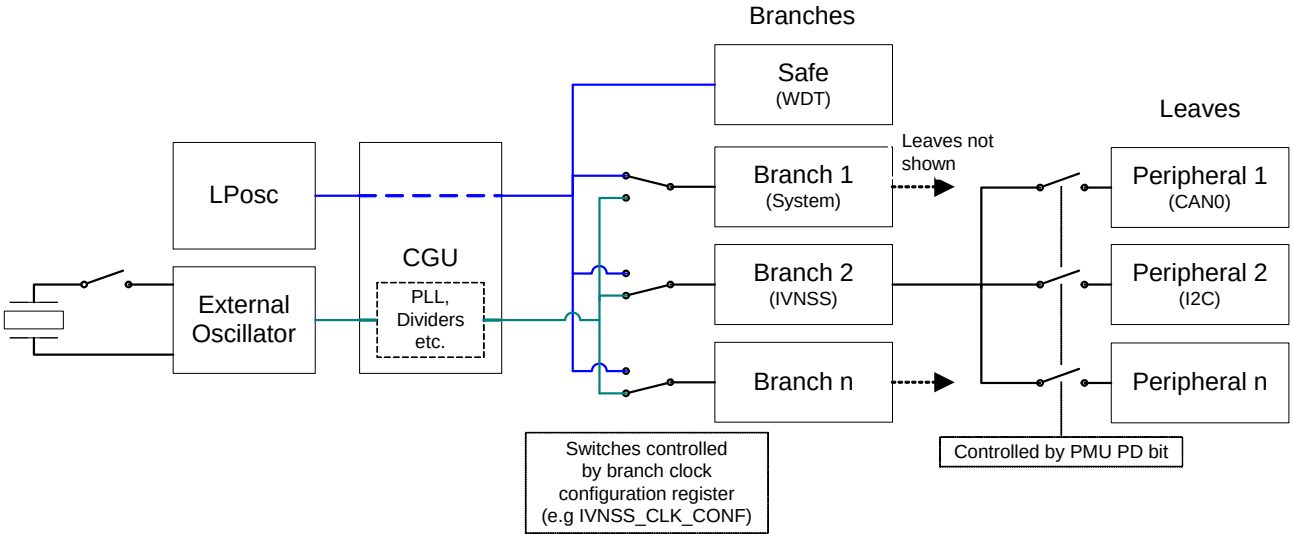


Fig 3. Simplified clock structure

To save power it is essential that as little logic is working as possible. Figure 2 illustrates a simplified clock structure that makes it easy to see how the switches need to be configured.

In the code a define '`SWITCH_CLOCKS_TO_XTAL`' is used to switch all clocks to the crystal oscillator prior to PD.

4.3.1 Changing the clock source

All clock branches need to be switched to work directly from the crystal oscillator. Setting the XXX_CLK_CONF register does this.

```
XXXX_CLK_CONF = CLK_SEL_XTAL | AUTOBLK | DIV1;
```

This code is located directly behind the line which switches the system clock to the LPosc. The system clock is not set to work from the crystal oscillator.

4.3.2 Disable the clock branches

After setting the wakeup bits, the clock branches can be disabled. Although the main clock source for these branches will be disabled, switching them off gives a predictable behavior when the clock source is restored.

```
xxxx_CLK_CONF |= 0x01;
```

4.3.3 Re-configuring the clock branches

For the reconfiguration of the clocks, the function '`TargetResetInit()`' is called again.. The restarting of the crystal oscillator and PLL's as described in paragraph 4.2.2. is skipped because this was already carried out by '`TargetResetInit()`'.

4.4 Measurement

If the MCU is in PD, the current consumption can be measured over the pads of resistor R6. Table 1 shows some typical values.

Table 1. VDD current consumption

Shown values are typical

Power state	Active	PD normal	PD clocks switching
VDD (1.8V) Current consumption	51 mA	85 uA	55 uA

5. Program code

```
/*****************************************************************************
 *   pmuIPD.c:  PMUIPD demo module file for NXP LPC29xx Family Microprocessors
 *
 *   Copyright(C) 2007, NXP Semiconductor
 *   All rights reserved.
 *
 *   History
 *   2009.01.14  ver 1.00    First Release
 *
******************************************************************************/
#include "LPC29xx.h"                        /* LPC29xx definitions */
#include "type.h"
#include "target.h"
#include "irq.h"
#include "LCD.h"
#include "extint.h"
#include "flashoff.h"

#define SWITCH_CLOCKS_TO_XTAL 0          // 1: Switch all clocks to Xtal before PD to save some additional power
                                         // 0: only use the PMU PD features
void GOSleepMode( void );

extern volatile DWORD eint_flag;

/*****************************************************************************
** Function name:      main
**
** Descriptions:       main routine for PMU module test
**
** parameters:         None
** Returned value:     int
**
******************************************************************************/
int main( void )
{
    DWORD cnt=25;
    unsigned long i;

    LCD_init ();
    LCD_cls ();
    LCD_print (0, 0, "LPC2900 PD DEMO ");
    LCD_print (0, 1, "  www.NXP.com   ");

    // Delay is necessary to allow reflashing, else the clocks could be disabled before
    // the JTAG communication can initialeze and you brick you MCU.
    for (i=0;i<=0x2FFFFF;i++){}

    GPIO1_DR = (1<<24)|(1<<25)|(1<<26)|(1<<27)|(1<<28)|(1<<29);
```

```
49
50          // external interrupts are used as a wakeup event, select the event in exint.h
51          EventRouter_Init();
52
53          while ( 1 )
54          {
55              // external interrupt occurs.
56              if ( eint_flag > 0 )
57              {
58                  if(eint_flag & (1<<EXINT_sleep))
59                  {
60                      GOSleepMode(); // got to sleep
61                  }
62                  if(eint_flag & (1<<EXINT_wake))
63                  {
64                      GPIO1_OR ^= (1<<26); // toggle led
65                  }
66                  eint_flag = 0;
67              }
68
69              // blink led on port 1.27 to indicate we are running the main loop waiting for a switch
70              GPIO1_OR ^= (1<<27);
71              LCD_bargraph (0, 1, 16, cnt&0x7F); /* Display bargraph according to cnt  */
72              cnt++;
73          }
74      }
75
76
77      /*****************************************************************************
78      ** Function name:      GOSleepMode
79      **
80      ** Descriptions:       Set clocks and go to sleep mode. The sleep mode
81      **                  turns off all the clocks, then, puts the
82      **                  the micro into power down mode which turns off
83      **                  the clocks with wakeup enabled. The wakeup source
84                         is configured before going into sleep mode (power down).
85      **
86      ** parameters:         None
87      ** Returned value:     None
88      **
89      *****************************************************************************/
90      void GOSleepMode( void )
91      {
92          LCD_print (0, 0, "Sleeping..ZZzzz");
93
94          // any pending interrupts in the VICwill prevent the device from going into power down mode,
95          // so all pending interrupts need to be handled before we can proceed.
96      Interrupts in the core do not
97          // prevent the device from going to sleep.
98
```

```
99        // Before we proceed, we will disable the interrupt in the core because any core interrupt will cause problems when waking up. If we wakeup by a
      interrupt
100       // the CPU will try to branch to an interrupt handler and approach unavailable hardware     // , causing an undesired jump to the exeption handlers.
101       __disable_irq();
102       __disable_fiq();
103
104        // If LP_OSC and PLL are not present, no need to go further
105       while (!(CGU_RDET & (0x01|PLL_PRESENT)));
106
107        // Set the system clock to run from LPosc
108       SYS_CLK_CONF = AUTOBLK | DIV1;
109
110   #if SWITCH_CLOCKS_TO_XTAL
111          // All the subsystem switched to run from X_tal.
112       IVNSS_CLK_CONF = CLK_SEL_XTAL | AUTOBLK | DIV1;
113       MSCSS_CLK_CONF = CLK_SEL_XTAL | AUTOBLK | DIV1;
114       UART_CLK_CONF = CLK_SEL_XTAL | AUTOBLK | DIV1;
115       SPI_CLK_CONF = CLK_SEL_XTAL | AUTOBLK | DIV1;
116       ADC_CLK_CONF = CLK_SEL_XTAL | AUTOBLK | DIV1;
117
118       ICLK0_CLK_CONF = CLK_SEL_XTAL | AUTOBLK | DIV1;
119       ICLK1_CLK_CONF = CLK_SEL_XTAL | AUTOBLK | DIV1;
120
121       USB_CLK_CONF = CLK_SEL_XTAL | AUTOBLK | DIV1;
122       USB_I2C_CLK_CONF = CLK_SEL_XTAL | AUTOBLK | DIV1;
123       OUT_CLK_CONF = CLK_SEL_XTAL | AUTOBLK | DIV1;
124       CTEST_CLK_CONF = CLK_SEL_XTAL | AUTOBLK | DIV1;
125   #endif
126
127        // All the branch clocks are set as "wake-up enabled".
128        // The CPU, SYS, PCR clocks can  only be disabled with the PMU_PD bit.
129        // bit2    bit1   bit0
130        // WAKEUP | AUTO | RUN //
131       PMU_CLK_CFG_CPU = (0x1<<2)|(0x1<<1)|(0x1<<0);
132       PMU_CLK_CFG_SYS = (0x1<<2)|(0x1<<1)|(0x1<<0);
133       PMU_CLK_CFG_PCR = (0x1<<2)|(0x1<<1)|(0x1<<0);
134       PMU_CLK_CFG_FMC = (0x1<<2)|(0x1<<1)|(0x1<<0);
135       PMU_CLK_CFG_RAM0 = (0x1<<2)|(0x1<<1)|(0x1<<0);
136       PMU_CLK_CFG_RAM1 = (0x1<<2)|(0x1<<1)|(0x1<<0);
137
138        // The following clocks cannot be accessed unless XX_CLK_CONF are set,
139        // LP_OSC, ext. OSC, or PLL, or FDIVx
140        // set all clocks to run, auto and wakeup enabled.
141       PMU_CLK_CFG_SMC = (0x1<<2)|(0x1<<1)|(0x1<<0);
142       PMU_CLK_CFG_GESS = (0x1<<2)|(0x1<<1)|(0x1<<0);
143       PMU_CLK_CFG_VIC = (0x1<<2)|(0x1<<1)|(0x1<<0);
144       PMU_CLK_CFG_PESS = (0x1<<2)|(0x1<<1)|(0x1<<0);
145       PMU_CLK_CFG_GPIO0 = (0x1<<2)|(0x1<<1)|(0x1<<0);
146       PMU_CLK_CFG_GPIO1 = (0x1<<2)|(0x1<<1)|(0x1<<0);
147       PMU_CLK_CFG_GPIO2 = (0x1<<2)|(0x1<<1)|(0x1<<0);
148       PMU_CLK_CFG_GPIO3 = (0x1<<2)|(0x1<<1)|(0x1<<0);
```

```
149       PMU_CLK_CFG_IVNSSA = (0x1<<2)|(0x1<<1)|(0x1<<0);
150       PMU_CLK_CFG_MSCSSA = (0x1<<2)|(0x1<<1)|(0x1<<0);
151       PMU_CLK_CFG_GPIO4 = (0x1<<2)|(0x1<<1)|(0x1<<0);
152       PMU_CLK_CFG_GPIO5 = (0x1<<2)|(0x1<<1)|(0x1<<0);
153       PMU_CLK_CFG_DMA = (0x1<<2)|(0x1<<1)|(0x1<<0);
154       PMU_CLK_CFG_USB = (0x1<<2)|(0x1<<1)|(0x1<<0);
155       PMU_CLK_CFG_PCR_IP = (0x1<<2)|(0x1<<1)|(0x1<<0);
156       PMU_CLK_CFG_IVNSS_VPB = (0x1<<2)|(0x1<<1)|(0x1<<0);
157       PMU_CLK_CFG_CANCA = (0x1<<2)|(0x1<<1)|(0x1<<0);
158       PMU_CLK_CFG_CANC0 = (0x1<<2)|(0x1<<1)|(0x1<<0);
159       PMU_CLK_CFG_CANC1 = (0x1<<2)|(0x1<<1)|(0x1<<0);
160       PMU_CLK_CFG_I2C0 = (0x1<<2)|(0x1<<1)|(0x1<<0);
161       PMU_CLK_CFG_I2C1 = (0x1<<2)|(0x1<<1)|(0x1<<0);
162       PMU_CLK_CFG_LIN0 = (0x1<<2)|(0x1<<1)|(0x1<<0);
163       PMU_CLK_CFG_LIN1 = (0x1<<2)|(0x1<<1)|(0x1<<0);
164       PMU_CLK_CFG_MSCSS_VPB = (0x1<<2)|(0x1<<1)|(0x1<<0);
165       PMU_CLK_CFG_MTMR0 = (0x1<<2)|(0x1<<1)|(0x1<<0);
166       PMU_CLK_CFG_MTMR1 = (0x1<<2)|(0x1<<1)|(0x1<<0);
167       PMU_CLK_CFG_PWM0 = (0x1<<2)|(0x1<<1)|(0x1<<0);
168       PMU_CLK_CFG_PWM1 = (0x1<<2)|(0x1<<1)|(0x1<<0);
169       PMU_CLK_CFG_PWM2 = (0x1<<2)|(0x1<<1)|(0x1<<0);
170       PMU_CLK_CFG_PWM3 = (0x1<<2)|(0x1<<1)|(0x1<<0);
171       PMU_CLK_CFG_ADC0_VPB = (0x1<<2)|(0x1<<1)|(0x1<<0);
172       PMU_CLK_CFG_ADC1_VPB = (0x1<<2)|(0x1<<1)|(0x1<<0);
173       PMU_CLK_CFG_ADC2_VPB = (0x1<<2)|(0x1<<1)|(0x1<<0);
174       PMU_CLK_CFG_QEI = (0x1<<2)|(0x1<<1)|(0x1<<0);
175       PMU_CLK_CFG_OUT_CLK = (0x1<<2)|(0x1<<1)|(0x1<<0);
176       PMU_CLK_CFG_UART0 = (0x1<<2)|(0x1<<1)|(0x1<<0);
177       PMU_CLK_CFG_UART1 = (0x1<<2)|(0x1<<1)|(0x1<<0);
178       PMU_CLK_CFG_SPI0 = (0x1<<2)|(0x1<<1)|(0x1<<0);
179       PMU_CLK_CFG_SPI1 = (0x1<<2)|(0x1<<1)|(0x1<<0);
180       PMU_CLK_CFG_SPI2 = (0x1<<2)|(0x1<<1)|(0x1<<0);
181       PMU_CLK_CFG_TMR0 = (0x1<<2)|(0x1<<1)|(0x1<<0);
182       PMU_CLK_CFG_TMR1 = (0x1<<2)|(0x1<<1)|(0x1<<0);
183       PMU_CLK_CFG_TMR2 = (0x1<<2)|(0x1<<1)|(0x1<<0);
184       PMU_CLK_CFG_TMR3 = (0x1<<2)|(0x1<<1)|(0x1<<0);
185       PMU_CLK_CFG_ADC0 = (0x1<<2)|(0x1<<1)|(0x1<<0);
186       PMU_CLK_CFG_ADC1 = (0x1<<2)|(0x1<<1)|(0x1<<0);
187       PMU_CLK_CFG_ADC2 = (0x1<<2)|(0x1<<1)|(0x1<<0);
188       PMU_CLK_CFG_TSSHELL = (0x1<<2)|(0x1<<1)|(0x1<<0);
189       PMU_CLK_CFG_USB_I2C = (0x1<<2)|(0x1<<1)|(0x1<<0);
190       PMU_CLK_CFG_USB_CLK = (0x1<<2)|(0x1<<1)|(0x1<<0);
191
192   #if SWITCH_CLOCKS_TO_XTAL
193       // power down clock slices on the subsystem.
194       USB_CLK_CONF |= 0x01;
195       USB_I2C_CLK_CONF |= 0x01;
196       OUT_CLK_CONF |= 0x01;
197
198       IVNSS_CLK_CONF |= 0x01;
199       MSCSS_CLK_CONF |= 0x01;
```

```
200       UART_CLK_CONF |= 0x01;
201       SPI_CLK_CONF |= 0x01;
202       ADC_CLK_CONF |= 0x01;
203       TMR_CLK_CONF |= 0x01;
204       ICLK0_CLK_CONF |= 0X01;
205       ICLK1_CLK_CONF |= 0x01;
206   #endif
207
208       CGU1_PLL_CTRL = 1; //Power Down CGU1 PLL
209       CGU_PLL_CTRL = 1;   //Power Down CGU PLL
210
211       // Power down OSC pads, still leave HF bit as default.
212       CGU_OSC_CTRL = (0x1<<2) | (0x0<<0);
213
214       EEPWRDWN =1;   // Power Down the EEPROM controller
215
216       sleepFlashOff();// SleepflashOff is placed in TCM so flash can be disabled. see "flashoff.c"
217                       // To locate the code in TCM the uVision feature is used.
218
219                       //void sleepFlashOff( void )
220                       //{
221                       //   /* Disable flash (Power Down) */
222                       //   FCTR |=0x200;
223                       //
224                       //   /* Switch selected clocks off (resumed at event) */
225                       //   PMU_PM =0x1;
226                       //
227                       //   /* Enable flash */
228                       //   FCTR &=~0x200;
229                       //
230                       //  return;
231                       //}
232
233   #if SWITCH_CLOCKS_TO_XTAL
234
235       TargetResetInit();   // a quick way to get all clock configured again after wakeup
236       EventRouter_Init();
237   #else
238       CGU_OSC_CTRL |= (0x1<<2) | (0x1<<0);          // re-enable the osc
239       while ( !(CGU_RDET & XTAL_PRESENT) ){ NOP; }     // wait for osc to start up
240
241       // Restart PLL
242       CGU_PLL_CTRL = PLL_XTAL_SEL | (PLL_M_VALUE<<MSEL_SHIFT) | P23EN;
243
244       // Check lock bit, if unlocked, PLL_LOCK is always 0
245       while ( !(CGU_PLL_STAT & PLL_LOCK) ) { NOP; }
246       // Check clock detection register to make sure PLL is present now.
247       while ( !(CGU_RDET & PLL_PRESENT) ) { NOP; }
248
249       CGU_PLL_CTRL = PLL_XTAL_SEL | (PLL_M_VALUE<<MSEL_SHIFT) | P23EN;
250
```

```
251         // PLL is 250Mhz, SYS_CLK and TMR_CLK is 125Mhz. This line depends on the board
252         // and configuration of clocks used see TargetResetInit() in target.c
253         SYS_CLK_CONF = CLK_SEL_PLL | AUTOBLK | DIV2;
254     #endif
255
256         // At this point all interrupts are still disabled and it's now safe to
257         // re-enable them as all the hardware is up and running again.
258         __enable_fiq();
259         __enable_irq();
260
261         LCD_print (0, 0, "      Awake       ");
262
263         return;
264     }
265     /*****************************************************************************
266     **                End Of File
267     *****************************************************************************/
```

6. Legal information

6.1 Disclaimers

General — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of a NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is for the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

6.2 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

7. Contents

1. Introduction 3
2. Hardware 3
3. Software 3
3.1 IDE 3
3.2 Configuring the code 3
3.3 Compiling and flashing the code 3
4. Program description 4
4.1 Entering sleep mode 6
4.1.1 Disable interrupts 6
4.1.2 Switch system to LPosc 6
4.1.3 Setting the wakeup bits 6
4.1.4 Stop PLL and crystal oscillator 6
4.1.5 Disable the EEPROM 6
4.1.6 Enable Sleep mode 7
4.2 Waking up 7
4.2.1 Re-enabling FLASH 8
4.2.2 Starting the clock source 8
4.2.3 Enabling interrupts 8
4.3 Additional power saving 9
4.3.1 Changing the clock source 9
4.3.2 Disable the clock branches 10
4.3.3 Re-configuring the clock branches 10
4.4 Measurement 10
5. Program code 11
6. Legal information 17
6.1 Disclaimers 17
6.2 Trademarks 17
7. Contents 18

For more information, please visit: http://www.nxp.com
For sales office addresses, email to: salesaddresses@nxp.com

Date of release: 25 February 2009
Document identifier: AN10795_2