Semiconductors

AN10421

Power management for LPC2138

Rev. 01 — 06 January 2006 Application note

Document information

Info	Content
Keywords	Core and peripherals power consumption measurement, code examples
Abstract	This application note contains power consumption data for the LPC2138 MCUs. Code examples used when making the room temperature measurements are also included.

PHILIPS

Revision history

Rev	Date	Description
01	20060106	Initial version.

Contact information

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

For sales office addresses, please send an email to: sales.addresses@www.semiconductors.philips.com

© Koninklijke Philips Electronics N.V. 2006. All rights reserved.

1. Introduction

The LPC2000 family of ARM7 MCUs has a wide range of operation. This application note shows the power consumption of the LPC2138 on a MCB2130 board. Measurements are focused on the different possible power modes of the device. Software code used is included at the end of this application note.

2. LPC2138 power management

Power consumption of an ARM7 core and its peripherals depend basically on V_{DD} and the clock rate. All used abbreviations are listed in the user manual UM10120 for the LPC2131/2/4/8 devices.

2.1 LPC2138 power measurement

The power consumption measurement for LPC2138 was done on the Keil MBC2130 evaluation board. All measurements have been done on the V_{DD} = 3.3 V rail of the board by an ampere meter shown in Fig 1 at room temperature. Please note that this application note provides only typical numbers and values.

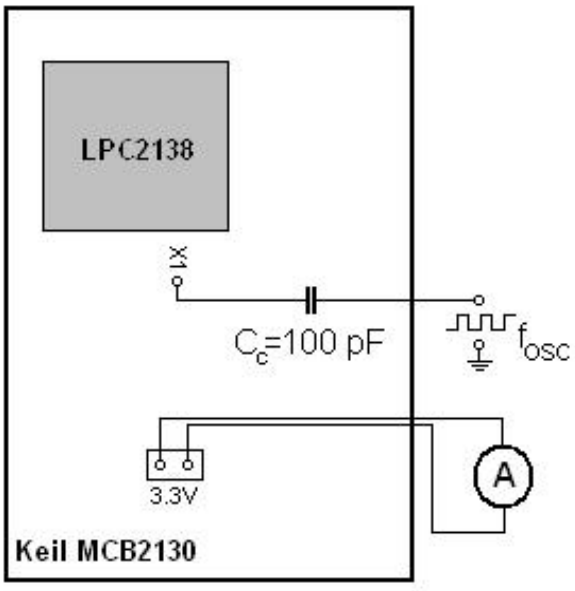

Fig 1. Test assembly, pin X1 and capacitance C_c (refer to LPC2138 user manual (UM10120) chapter 3.4 Crystal oscillator Fig. 7)

© Koninklijke Philips Electronics N.V. 2006. All rights reserved.

2.1.1 Clocks and PLL

In general there are two reference clocks inside the LPC2000 family ARM7 MCUs - the CPU clock or CCLK and the peripheral clock or PCLK.

To generate CCLK the external clock source on the X1 pin can be used directly or the internal PLL (Phase Lock Loop) can be used to provide a certain target CCLK above the external frequency. The application note AN10331 describes how to implement PLL operation.

Equation (1) describes the derivation of PCLK. VPBDIV can have values of {1,2,4}.

$$PCLK = \frac{CCLK}{VPBDIV} \quad (1)$$

UM10120 Chapter 3.10 describes the relationship between CCLK and PCLK. The register VPBDIV can be used to lower power consumption, mentioned as the second purpose of VPBDIV (see Table 1). The relationship between power consumption and PCLK is linear.

Table 1: Using PLL in combination with VPBDIV

If required core performance is higher than peripheral required speed, user can save power by using the VPBDIV register

VPBDIV	CCLK [MHz]	PCLK [MHz]	Relative peripheral power consumption
1	12	12	100%
	24	24	200%
	48	48	400%
2	12	6	50%
	24	12	100%
	48	24	200%
4	12	3	25%
	24	6	50%
	48	12	100%

2.1.2 Core power consumption

The power consumption of an ARM7 MCU core depends on hardware and software factors. UM10120 chapter 4 describes the Memory Accelerator Module (MAM). Running with the MAM on is a power saving feature in the LPC2138.

Table 2 shows how the memory was mapped in this applications note. On a side note, the difference in performance is negligible, since Flash has a Memory Accelerator Module (MAM), so performance is close to SRAM.

© Koninklijke Philips Electronics N.V. 2006. All rights reserved.

Table 2: Flash or SRAM
Requirements: code, data and const have to fit in memory size

Mapping	Type	Memory
Flash like	Code	Flash
	Data	SRAM
	Const	Flash
SRAM like[1]	Code	SRAM
	Data	SRAM
	Const	SRAM

[1] Reference to Keil online article: ARM: Debug programs in on-chip RAM of Philips LPC2000

Besides the acceleration effect, MAM, if fully enabled, offers a way to save power, since enabling various buffers reduces the number of Flash fetches. The MAM Timing register determines how many CCLK cycles are used to access the Flash. For power optimization, a MAM Timing value of two or more is recommended.

The behavior of the ARM7 three-stage pipeline is an indicator of the core power consumption. In order to keep power consumption low, optimal pipelining is needed. That means all core operations are single cycle operations. Therefore software code should be very iterative. Loop structures, like `while (1)`, or strong recursive code will increase power consumption, since the core keeps fetching all the time. A work around could be idle-mode mentioned later in this application note.

In addition, software decides whether the instruction set used is ARM or Thumb. Since ARM state uses 32-bit instructions, it has by definition the higher power consumption. In Thumb state, eight instructions are fetched by the MAM versus four in ARM state, and some instructions cannot execute in a single cycle. A comparison of ARM and Thumb encoding is shown in Table 3.

Table 3: The instruction pipeline
The program counter (PC) points to the instruction being fetched, not executed

Pipeline	ARM	Thumb
FETCH	PC	PC
DECODE	PC - 4	PC – 2
EXECUTE	PC - 8	PC – 4

So the lowest power consumption results from Thumb state, but time critical code, like an interrupt handler, should be written in ARM. A summary of useful values is shown in Table 4. The core power consumption can be described as a linear function with an offset.

© Koninklijke Philips Electronics N.V. 2006. All rights reserved.

Table 4: Core power consumption
Typical numbers of LPC2138 core power consumption

Core mode	Memory - State	Offset @ 1 MHz [mA]	Offset @ 10 MHz [mA]	Additional power consumption [mA/MHz]	Power consumption @ 60 MHz [mA]
Active[1]	Flash -Thumb	~5.6	~10.3	~0.52	~36.3
Active[1]	Flash - ARM	~6.4	~11.6	~0.58	~40.6
Active[1]	SRAM[2]	~5.5	~11.9	~0.71	~47.4
Idle	-	~1.3	-	~0.17	~11.3

[1] See code section chapter 3.2

[2] SRAM doesn't care about ARM or Thumb state, since measured while Debugger was running

2.1.3 Peripheral power consumption

The second focus point of this application note concerns the power consumption of the peripherals. This application note summarizes typical peripheral power consumption in a specific configuration in Table 5. Note that the internal PCLK divider column applies to the prescaler built into the indicated peripheral.

Table 5: Peripheral power consumption
Typical numbers of LPC2138 peripheral power consumption

Peripheral	Configuration	Internal PCLK divider	1 MHz < PCLK < 10 MHz [uA/MHz]	10 MHz < PCLK < 20 MHz [uA/MHz]	PCLK > 20 MHz [uA/MHz]
Timer/PWM[1]	Continuously counting	1	24.0	24.0	24.0
ADC[2][3][4]	10 channel burst mode	-	15.8	10.4	8.2
UART	Receiver mode	16	30.0	30.0	30.0
I2C	Master transmitter mode	-	13.0	10.2	6.3
SPI	Slave receiver mode	8	14.9	8.5	5.0
SSP	Slave receiver mode	2	14.9	8.5	5.0

[1] Pulse-width-modulator (PWM): Only timer registers are active

[2] 10 channels ADC with a sample rate of 400 kHz

[3] Value for only PCLK at 4Mhz and 8 MHz

[4] Running on maximum output clock rate (400 kHz)

Please note that on default all peripherals are active, so unused peripherals may be deactivated by the PCONP register to lower power consumption.

2.2 Power saving modes

The LPC2138 supports two reduced power modes: idle and power-down mode. In Idle mode, execution of instructions is suspended until either a reset or an interrupt occurs. Bits in the PCONP register, described in section 3.8.3 of UM10120, enable or disable specific peripherals in the LPC2138.

Peripheral functions continue operation during idle mode and may generate interrupts to cause the processor to resume execution. Idle mode eliminates power used by the processor, internal buses and memory systems. So entering idle mode instead of a `while`

© Koninklijke Philips Electronics N.V. 2006. All rights reserved.

(1) loop in the main program keeps peripherals fully functional with less power consumption.

In Power-down mode, the oscillator is shut down and the chip receives no internal clocks. The processor state and registers, peripheral registers, and internal SRAM values are preserved throughout power-down mode and the logic levels of chip pins remain static. The power-down mode can be terminated and normal operation resumed by either a reset or certain specific interrupts that are able to function without clocks because all dynamic operation of the chip is suspended.

Since the SRAM controller incorporates a write-back buffer holding the last data sent by software to the SRAM, a dummy write is needed before entering Idle or Power-down mode. Therefore, a dummy write to data is necessary if the same data is updated before entering either power saving mode.

For more detailed information see UM10120 Chapter 1.

2.2.1 Idle mode

The Idle mode, as used in this applications note, is when the CPU clock is inactive and all peripheral modules are turned off. In this application note, the offset value represents the power consumed with all peripherals turned off the PCONP register allows power consumption measurement of the specific peripheral. So the measurements results for idle mode itself are used as an offset to calculate single peripheral power consumption.

2.2.2 Power-down mode

Power Consumption in power-down mode depends on the pin configuration. For detailed information see LPC2131/2/4/8 data sheet table 10 and also the application note AN10404 chapter 4, which includes some hints for lowering power consumption in power-down mode. Using the information in AN10404 and the LPC2131/2/4/8 data sheet allows the defining of an optimized pin configuration (see Table 6). Also note, that input pins should be pulled high before entering power-down mode, since each pin, when low, will consume about ~50uA (typical).

© Koninklijke Philips Electronics N.V. 2006. All rights reserved.

Table 6: Optimized pin configuration in power-down mode

Optimized pin configuration	Description	Power consumption on V_{DD} [µA]	Power consumption on V_{BAT} [3] [µA]
Input[1]	Pins on port 0 are connected to V_{DD} Pins on port 1[2] are connected to V_{DD}	~60	~18

[1] Default setting

[2] Internal pull-ups resistors on port 1, cannot be deactivated

[3] V_{BAT} = 3.3 V

2.3 Calculation examples for LPC2138

As an example, once the CCLK is chosen for a certain application need, let's assume 60 MHz is required, the next step in design is to maximize the value of VPB divider or in other words to minimize the PCLK rate, which just perform above application's required peripheral speed. For instance a clock rate of 15 MHz is adequate for user's need, so VPBDIV can be set to 4. Therefore the reference clock rate for the core is 60 MHz (lookup Table 4) and for the peripherals it's 15 MHz (lookup Table 5). Now it's possible to calculate the total power consumption for a certain application by assuming the core uses Flash (see Table 2), the instructions are encoded in Thumb and only the vectored interrupts use ARM instructions. The results for this example are summarized in Table 7.

Table 7: Calculation example

Consumer load	Number	Peripheral power consumption @ 15 MHz [mA] (PCLK)	Core power consumption @ 60 MHz [mA] (CCLK)
Core	-	-	~36.3
Timer	2 x	0.720	-
Timer (PWM)	1 x	0.360	-
ADC	2 x	0.312	-
UART	2 x	0.900	-
I2C	2 x	0.306	-
SPI	1 x	0.128	-
SSP	1 x	0.128	-
Total [mA]		**~39.2**	

Values refer to chapter 2.1.2 and 2.1.3 of this application note

3. Software code

3.1 Idle and power-down mode

```
/*****************************************/
/*                                       */
/* Power Measurements: Idle/Power-Down   */
/*                                       */
```


© Koninklijke Philips Electronics N.V. 2006. All rights reserved.

```
/*****************************************/

#include <stdio.h>        // standard I/O .h-file
#include <LPC213x.H>      // LPC213x definitions

int main (void)
{
// Optimized pin configuration
// Output configuration: (~40uA)
   // Port0:
      // output configuration no additional resistors are needed
      IODIR0 = 0xFFFFFFFF;        // all output
      IOSET0 = 0xFFFFFFFF;        // set to high
      // Additional to that, all pins on Port0 are shortcut to 3.3V!!!
   // Port1: has internal pull-ups, cannot switched off!
      IODIR1 = 0xFFFF0000;        // all output (Pins P1.0-15 not available)
      IOSET1 = 0xFFFF0000;        // set to high
      // Additional to that, all pins on Port1 are shortcut to 3.3V!!!
/*
// Input configuration: (~65uA)
   // Port0:
      // input configuration
      IODIR0 = 0x00000000;        // all input
      // Additional to that, all pins on Port0 are shortcut to 3.3V!!!
   // Port1: has internal pull-ups, cannot switched off!
      IODIR1 = 0x00000000;        // all intput (Pins P1.0-15 not available)
      // Additional to that, all pins on Port1 are shortcut to each other!!!
*/
   PCONP = 0x00000000;       // deactivate all peripherals
   while(1)                  // loop forever
```


© Koninklijke Philips Electronics N.V. 2006. All rights reserved.

```
   {
   PCON |= 0x02;            // set power-down mode or
// PCON |= 0x01;            // set idle mode
   }
}
```

3.2 Core active mode

```
/*************************************/
/*                                   */
/* Power Measurements: Core Active Mode */
/*                                   */
/*************************************/


#include <stdio.h>     // standard I/O .h-file
#include <LPC213x.H>   // LPC213x definitions


//#pragma arm          // compile in ARM mode
//#pragma thumb        // compile in Thumb mode


int main (void)
{
   long i = 0;
   PCONP = 0x0000000;  // deactivate all peripherals
   while(1)            // loop forever
   {
   i++;…               // … e.g. 300 times an iterative command (i++; i++; i++;…)
   i = 0;
   }
}
```


© Koninklijke Philips Electronics N.V. 2006. All rights reserved.

3.3 Peripherals

```
/*************************************/
/*                                   */
/* Power Measurements: Peripherals   */
/*                                   */
/* Code example for CCLK = 60MHz     */
/*                                   */
/*************************************/


#include <stdio.h>                        /* standard I/O .h-file */
#include <LPC213x.H>                      /* LPC213x definitions  */


void I2C0_ISR_Handler(void) __irq
{
   // NOTE: SI bit is set after each STAT register change exept the 0xF8 status
   // STATUS:  a START condition has been transmitted (Bus was free)
   // NEXT: load SLA and write bit
   if (I20STAT == 0x08)
      {
      I20DAT    = 0x0000000E;     // 7 bit Slave Adress plus LSB = 0 for write
      I20CONCLR = 0x00000020;     // Reset the START bit
      }


   // STATUS:  SLA and Write has been transmitted, but not Acknowledge bit received
   // NEXT: repeat START
   if (I20STAT == 0x20)
      {
      I20CONSET = 0x00000020;     // Set the STA bit
      I20CONCLR = 0x00000020;     // Reset the START bit
      }
```


© Koninklijke Philips Electronics N.V. 2006. All rights reserved.

```
   // STATUS: A repeated START has bee transmitted
   // NEXT: load SLA and write bit
   if (I20STAT == 0x10)
      {
      I20DAT    = 0x0000000E;      // repeat Slave Adress and write
      I20CONCLR = 0x00000020;      // Reset the START bit
      }
   // STAUS:  SLA and Write has been transmitted and Acknowledge bit received
   // NEXT: load first Data byte
   if (I20STAT == 0x18)
      {
      I20DAT    = 0x000000AA;        // Data master transmits
      }
   // STATUS: Data byte has been transmitted and Acknowledged bit received
   // NEXTE:  load next Data byte
   if (I20STAT == 0x28)
      {
      I20DAT    = 0x000000AA;        // Data master transmits
      }
   // STATUS: Data byte has been transmitted, but no Acknowledged bit received
   // NEXT: repeat START
   if (I20STAT == 0x30)
      {
      I20CONSET = 0x00000020;      // Set the STA bit
      }
   I20CONCLR   = 0x00000008;    // Reset the SI bit to continue
   VICVectAddr = 0x00000000;    // Acknowledge register
}

void I2C1_ISR_Handler(void) __irq
```


© Koninklijke Philips Electronics N.V. 2006. All rights reserved.

```
{
   // NOTE: SI bit is set after each STAT register change exept the 0xF8 status
   // STATUS:  a START condition has been transmitted (Bus was free)
   // NEXT: load SLA and write bit
   if (I21STAT == 0x08)
      {
      I21DAT    = 0x0000000E;      // 7 bit Slave Adress plus LSB = 0 for write
      I21CONCLR = 0x00000020;      // Reset the START bit
      }

   // STATUS:  SLA and Write has been transmitted, but not Acknowledge bit received
   // NEXT: repeat START
   if (I21STAT == 0x20)
      {
      I21CONSET = 0x00000020;      // Set the STA bit
      I21CONCLR = 0x00000020;      // Reset the START bit
      }
   // STATUS:  A repeated START has bee transmitted
   // NEXT: load SLA and write bit
   if (I21STAT == 0x10)
      {
      I21DAT    = 0x0000000E;      // repeat Slave Adress and write
      I21CONCLR = 0x00000020;      // Reset the START bit
      }
   // STAUS:   SLA and Write has been transmitted and Acknowledge bit received
   // NEXT: load first Data byte
   if (I21STAT == 0x18)
      {
      I21DAT    = 0x000000AA;        // Data master transmits
      }
```


© Koninklijke Philips Electronics N.V. 2006. All rights reserved.

```
   // STATUS: Data byte has been transmitted and Acknowledged bit received
   // NEXTE:  load next Data byte
   if (I21STAT == 0x28)
      {
      I21DAT    = 0x000000AA;        // Data master transmits
      }
   // STATUS: Data byte has been transmitted, but no Acknowledged bit received
   // NEXT: repeat START
   if (I21STAT == 0x30)
      {
      I21CONSET = 0x00000020;     // Set the STA bit
      }
   I21CONCLR   = 0x00000008;   // Reset the SI bit to continue
   VICVectAddr = 0x00000000;   // Acknowledge register
}

int main (void)
{
          /*
          Turn on Peripherals:     PCONP
          bit 0 - reserved         0x00000001
          bit 1 - Timer0           0x00000002
          bit 2 - Timer1           0x00000004
          bit 3 - UART0            0x00000008
          bit 4 - UART1            0x00000010
          bit 5 - PWM0             0x00000020
          bit 6 - reserved         0x00000040
          bit 7 - I2C zero         0x00000080
          bit 8 - SPI0             0x00000100
          bit 9 - Real Time Clock  0x00000200
```


© Koninklijke Philips Electronics N.V. 2006. All rights reserved.

```
                bit 10- SSP/SPI1          0x00000400
                bit 11- reserved          0x00000800
                bit 12- ADC0              0x00001000
                bit 13- reserved          0x00002000
                bit 14- reserved          0x00004000
                bit 15- reserved          0x00008000
                bit 16- reserved          0x00010000
                bit 17- reserved          0x00020000
                bit 18- reserved          0x00040000
                bit 19- I2C one           0x00080000
                bit 20- ADC1              0x00100000
                bit 21-31 reserved
                */

PCONP  = 0x00000080; // I2C0      -> master transmitter mode (400kHz)
PCONP |= 0x00080000; // I2C1      -> master transmitter mode (400kHz)
PCONP |= 0x00000002; // Timer0    -> count continuously with P_CLK
PCONP |= 0x00000004; // Timer1    -> count continuously with P_CLK
PCONP |= 0x00000020; // PWM       -> use PWM as timer and counting with P_CLK
PCONP |= 0x00000008; // UART0     -> receiver mode (Divider = 16, max speed)
PCONP |= 0x00000010; // UART1     -> receiver mode (Divider = 16, max speed)
PCONP |= 0x00001000; // ADC0      -> burst mode conversion 10bit/11clock (400k s_rate)
PCONP |= 0x00100000; // ADC1      -> burst mode conversion 10bit/11clock (400k s_rate)
PCONP |= 0x00000100; // SPI0      -> slave receiver mode (Divider = 8, max speed)
PCONP |= 0x00000400; // SSP, SPI1 -> slave receiver mode (Divider = 2, max speed)


  // ISR for I2C0 transmitter
  VICIntEnable = 0x00000200; // set bit 9
  VICVectCntl0 = 0x00000029;
  VICVectAddr0 = (unsigned long) I2C0_ISR_Handler;
```


© Koninklijke Philips Electronics N.V. 2006. All rights reserved.

```
// ISR for I2C1 transmitter
VICIntEnable |= 0x00080000; // set bit 19
VICVectCntl1 = 0x00000033;
VICVectAddr1 = (unsigned long) I2C1_ISR_Handler;

 // Initialize I2C0
PINSEL0   = 0x00000050;     // Enable SCL0 and SDA0 on P0.2 and P0.3
I20CONSET = 0x00000040;     // Master mode configuration, I2EN bit set
I20SCLH   = 0x00000000;     // f_bit = f_clk / (I2SCLH + I2SCLL) <= 400kHz
I20SCLL   = 0x00000096;     // Init Master data counter = 400kHz @ 60MHz

 // Initialize I2C1
PINSEL0  |= 0x30C00000;     // Enable SCL1 and SDA1 on P0.11 and P0.14
I21CONSET = 0x00000040;     // Master mode configuration, I2EN bit set
I21SCLH   = 0x00000000;     // f_bit = f_clk / (I2SCLH + I2SCLL) <= 400kHz
I21SCLL   = 0x00000096;     // Init Master data counter = 400kHz @ 60MHz

// Timer0 Settings
T0TC     = 0x00000000;    // set Timer0 to 0
T0MCR    = 0x00000000;    // continuous operation

// Timer1 Settings
T1TC     = 0x00000000;    // set Timer1 to 0
T1MCR    = 0x00000000;    // continuous operation

// PWM Settings
PWMTC    = 0x00000000;    // set PWM Timer to 0
PWMMCR   = 0x00000000;    // continuous operation
```


© Koninklijke Philips Electronics N.V. 2006. All rights reserved.

```
 // Initialize the UART0
PINSEL0 |= 0x00000005;    // Enable RxD0 and TxD0   on P0.1  and P0.0
U0LCR   = 0x00000083;     // 8 bits, no Parity, 1 Stop bit
U0DLL   = 0x00000000;     // on Maximum Speed
U0DLM   = 0x00000000;     // Divider is const = 16
U0LCR   = 0x00000003;     // DLAB = 0
U0FCR   = 0x00000007;     // Enable FIFO and receiver water mark

 // Initialize the UART1
PINSEL0 |= 0x00050000;    // Enable RxD0 and TxD0   on P0.9  and P0.8
U0LCR   = 0x00000083;     // 8 bits, no Parity, 1 Stop bit
U0DLL   = 0x00000000;     // on Maximum Speed
U0DLM   = 0x00000000;     // Divider is const = 16
U0LCR   = 0x00000003;     // DLAB = 0
U0FCR   = 0x00000007;     // Enable FIFO and receiver water mark

// Initializes the ADC0
// Using Channel 5, which is reserved on P0.26
AD0CR    = 0x00210E20;    // CLKDIV = 15 @ 60 MHz -> 400k sample rate

// Initializes the ADC1
// Using Channel 5, which have to be selected on P0.15
PINSEL0 |= 0xC0000000;    // Enable AD1.5 on P0.15
AD1CR    = 0x00210E20;    // CLKDIV = 15 @ 60 MHz -> 400k sample rate

// Initializes the SPI0
PINSEL0 = 0x00005500;     // Select SCK, MISO, MOSI, SSEL on P0.4,...P0.7
S0SPCR  = 0x18;           // CPOL = 1, CPHA = 1, slave mode
S0SPCCR = 0x08;           // min. CLK Divider
```


© Koninklijke Philips Electronics N.V. 2006. All rights reserved.

```
// Initializes the SSP, SPI1
PINSEL1 = 0x000002C8;     // Select SCK, MISO, MOSI, SSEL on P0.17,...P0.20
SSPCR0  = 0x00C7;      // SCR = 1, CPOL = 1, CPHA = 1, SPI mode, 8 bits
SSPCR1  = 0x04;           // slave mode, MISO active
SSPCPSR = 0x02;           // min. CLK Divider

// Start I2C0 transmission
I20CONSET = 0x00000020;
// Start I2C1 transmission
I21CONSET = 0x00000020;
// Start Timer0
T0TCR     = 0x00000001;
// Start Timer1
T1TCR     = 0x00000001;
// Start PWM counter
PWMTCR    = 0x00000001;
// Start ADC0 conversion (burst mode)
AD0CR    |= 0x01000000;
// Start ADC1 conversion (burst mode)
AD1CR    |= 0x01000000;

while (1)             // Loop forever
{
PCON |= 0x01;         // set idle mode
}
```


© Koninklijke Philips Electronics N.V. 2006. All rights reserved.

4. Disclaimers

Life support — These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips Semiconductors customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors for any damages resulting from such application.

Right to make changes — Philips Semiconductors reserves the right to make changes in the products - including circuits, standard cells, and/or software - described or contained herein in order to improve design and/or performance. When the product is in full production (status 'Production'), relevant changes will be communicated via a Customer Product/Process Change Notification (CPCN). Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no licence or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

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

5. Trademarks

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

© Koninklijke Philips Electronics N.V. 2006. All rights reserved.

6. Contents

© Koninklijke Philips Electronics N.V. 2006

All rights are reserved. Reproduction in whole or in part is prohibited without the prior written consent of the copyright owner. The information presented in this document does not form part of any quotation or contract, is believed to be accurate and reliable and may be changed without notice. No liability will be accepted by the publisher for any consequence of its use. Publication thereof does not convey nor imply any license under patent- or other industrial or intellectual property rights.

Date of release:06 January 2006
Document number: AN10421_1

Published in The Netherlands