Semiconductors

AN10391

Low battery voltage measurement with the LPC900 microcontrollers

Rev. 01 — 3 February 2006 Application note

Document information

Info	Content
Keywords	Microcontrollers, low battery measurement, LPC900, comparators
Abstract	Low battery measurement with the LPC900 using the comparators.

PHILIPS

Revision history

Rev	Date	Description
01	20060203	Initial revision

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**

1. Introduction

The LPC900 microcontrollers can be used in battery applications. Being able to detect a low battery condition might be necessary in these applications. The Brownout detect feature can be used to detect a low battery condition, the brownout has a fixed reference voltage of 2.7 V.

If a lower voltage then 2.7 V needs to be detected the comparator can be used for this. Using a voltage divider hooked up to the comparator can detect a low battery condition.

This application note will talk about the implementation of this circuit and how to minimize power consumption with this circuit.

2. Low battery measurement circuit

Figure 1 shows the test circuit that works with the demo code provided in the Appendix.

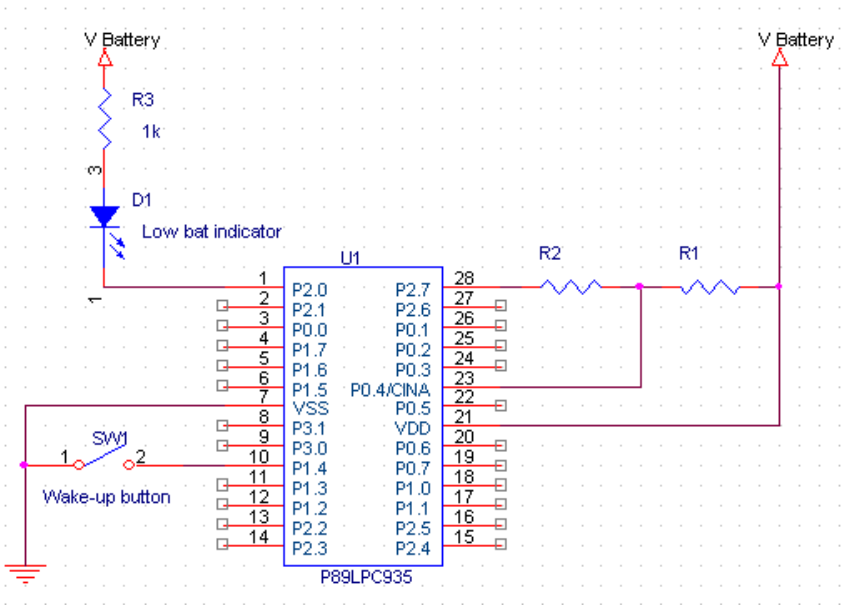

Fig 1. Low battery detect circuit

R1 and R2 are a resistor divider that is fed into the comparator input, as the battery voltage drops the voltage into the comparator drops. Since the comparator has a stable internal reference voltage a ratio of R1 and R2 can be chosen to trip the comparator at a specific battery voltage.

R2 is hooked up to a port pin, which can enable and disable the resistor ladder by either pulling it low or driving it high. When the resistor ladder is driven high there will be no current flowing through the resistor ladder and it will not consume any power. When a measurement needs to be taken the resistor ladder can be turned on by driving P2.0 low.

Switch 1 is hooked up to external interrupt 1 to interrupt the part from power-down. An LED is hooked up to indicate a low battery condition for demonstration purposes. In a real application there might be need to store information on such a low battery condition or another task that would need to be taken care off.

3. Low battery measurement

The trip point of the internal voltage reference of the comparators is 1.23 V. The equation of the resistor ratio between R1 R2.

$$\frac{R2}{R1+R2} VBatterytrip = 1.23\ V \quad (1)$$

$$\frac{R1}{R2} = \frac{VBatterytrip}{1.23} - 1 \quad (2)$$

Table 1 shows the battery trip voltages of the comparator and what resistor ratios to use to get these trip points.

Table 1: Battery trip voltage and resistor ratio

Vbattery	Resistor ratio [R1/R2]
2.4 V	0.95
2.5 V	1.03
2.6 V	1.11
2.7 V	1.20
2.8 V	1.28
2.9 V	1.36
3.0 V	1.44
3.1 V	1.52
3.2 V	1.60
3.3 V	1.68
3.4 V	1.76
3.5 V	1.85
3.6 V	1.93

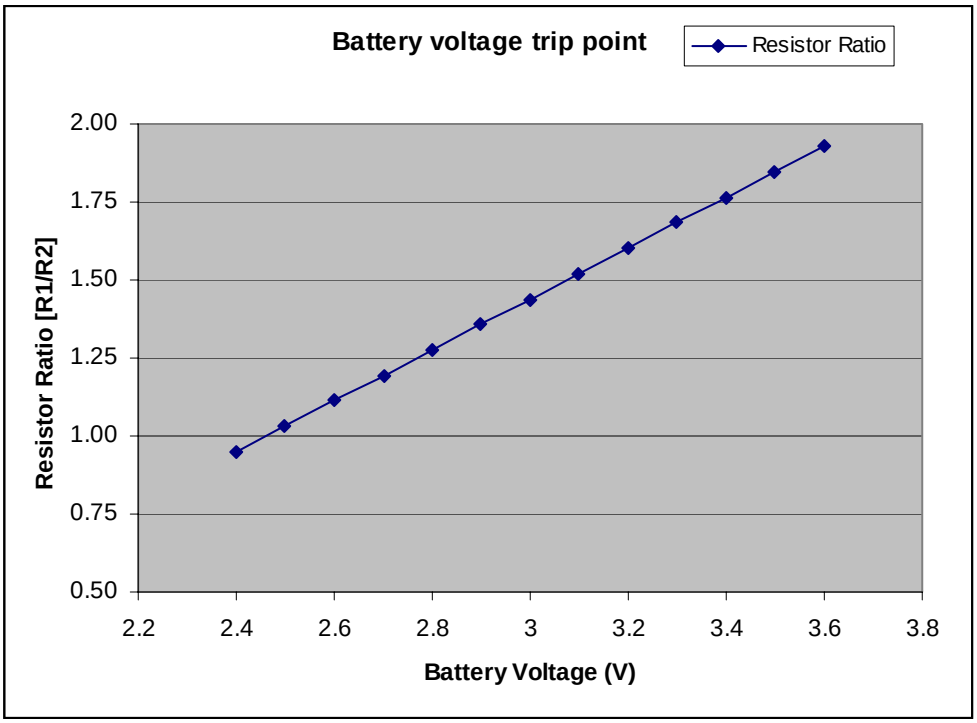

Fig 2. Battery trip points resistor ratio

4. Low battery demo software

The software for the low battery demo was created with the help of the online code generation tool Code Architect, which can be found at the Code Architect website. The code architect generates code functions for different peripherals that can be used.

For this software the comparator is used for the battery voltage measurement, the external interrupt is used for waking up from power-down.

Different power-down configurations can be used, either total Power-down mode or Power-down mode. Total Power-down mode is used in this demo software where the whole application draws less then 1 μA. This means the battery can only be measured when the application wakes up.

It is also possible to have the comparators active in power-down, but this will draw more power in Power-down mode. Also the resistor divider has to stay on for the comparison overall drawing significantly more power then only measuring power on a wake-up.

The difference between the two is that Power-down mode will keep the comparators active while total Power-down mode will turn of the comparators.

The source for the low battery demo software is listed in Section 5 “Appendix”.

5. Appendix

5.1 Appendix A: low_bat_det.c

```
//*************************************************************************
//*  low_bat_det.c
//*Description : low battery detection
//*
//*************************************************************************
//* Versions
//*************************************************************************
//*
//* v1.0 August 2005
//* Initial version.
//*
//*************************************************************************
#include <reg932.h> // SFR definitions for LPC932
#include <comparators.h> // header file for comparators
//*************************************************************************
//* Functions
//*************************************************************************
void init(void);            // part initialization
void main(void);            // main loop
bit low_bat_det(void);
void vdiv_enable(void);
void vdiv_disable(void);
void msec(int msec);
//*************************************************************************
//* init()
//* Input(s) : none.
//* Returns : none.
//* Description : initialization of P89LPC932
//*************************************************************************
void init(void)
{
  P0M1 = 0x00; // P0 quasi bi
  P0M2 = 0x00;
  P1M1 = 0x00; // P1 quasi bi
  P1M2 = 0x00;
  P2M1 = 0x00; // P2 quasi bi
  P2M2 = 0x00;
  EX1 = 1; // enable external interrupt 1
}
//*************************************************************************
//* main()
//* Input(s) : none.
//* Returns : none.
//* Description : main loop
//*************************************************************************
void main ()
{
```

```
  init();
  while(1) // HexFile Loader
  {
    EA = 1; // enable all interrupts
    //PCON |= 0x03; // enter Power-down mode
EA = 0; // disable all interrupts after external interrupt wakeup
if(low_bat_det())
{
  ICB = ~ ICB; // toggle P2.7 to indicate a low battery condition
  msec(500);
  ICB = ~ ICB; // toggle P2.7 to indicate a low battery condition
  msec(500);
}
  }
}
//*************************************************************************
//* low_bat_det()
//* Input(s) : none.
//* Returns : bit, low battery detect,
//* Description : Return bit when low voltage condition is detected
//*************************************************************************
bit low_bat_det(void)
{
  char low_battery;
  PCONA &= ~0x20;                    // power up voltage comparator
  vdiv_enable();                     // enable the voltage divider
  comparators_init(COMP_1,COMP_INPUTA, COMP_INTERNALREF, COMP_OUTPUTDISABLE);
  low_battery = comparators_getoutput(COMP_1);
  comparators_disable(COMP_1);          // comparator number: COMP_1
  vdiv_disable();                    // disable the voltage divider
  PCONA |= 0x20;                     // power down the voltage comparator
  if(!low_battery) // return low baterry bit
  {
    return 1;
  }
  else
  {
    return 0;
  }
}
//*************************************************************************
//* enter_icp()
//* Input(s) : none.
//* Returns : none.
//* Description : enable voltage divider
//*************************************************************************
void vdiv_enable(void)
{
  ICA = 0; // low on P2.7 to enable resistor ladder
}
//*************************************************************************
```

```
//* enter_icp()
//* Input(s) : none.
//* Returns : none.
//* Description : function to pulse reset to enter ICP mode
//*****************************************************************************
void vdiv_disable(void)
{
  ICA = 1; // high on P2.7 to disable resistor ladder
}
//*****************************************************************************
//* msec()
//* Input(s) : none.
//* Returns : none.
//* Description : delay for number of mili seconds
//*****************************************************************************
void msec(int msec)
{
  int delay = 0;
  while(msec) // delay till msec is 0
  {
    for(delay = 0;delay < 680; delay++);// 1 msec delay
    msec--; // decrement msec
  }
}
//*****************************************************************************
//* msec()
//* Input(s) : none.
//* Returns : none.
//* Description : delay for number of mili seconds
//*****************************************************************************
void ext_int1_isr(void) interrupt 6
{

}
```

5.2 Appendix B: comparators.c

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

MODULE:    Comparators
VERSION:   1.02
CONTAINS:  Routines for controlling the comparators on the Philips P89LPC935
COPYRIGHT: Embedded Systems Academy, Inc.
LICENSE:   May be freely used in commercial and non-commercial code without royalties
provided this copyright notice remains in this file and unaltered

WARNING:   IF THIS FILE IS REGENERATED BY CODE ARCHITECT ANY CHANGES MADE WILL BE LOST.
WHERE POSSIBLE USE ONLY CODE ARCHITECT TO CHANGE THE CONTENTS OF THIS FILE

GENERATED: On "Aug 15 2005" at "13:49:35" by Code Architect 2.11
*****************************************************************************/
// SFR description needs to be included
```

```
#include <reg935.h>
#include "comparators.h"

/***********************************************************************
DESC:    Generates a 13 microsecond delay needed to stabilize a
         comparator output after enabling.
         Note that the datasheet mentions a 10 microsecond delay.
         Because the timer may be clocked from the watchdog timer, which
         can be up to 30% faster than stated, 30% has been added to the
         absolute minimum delay of 10us to give 13us.
         Uses timer 0
         Actual delay: 13 us
RETURNS: Nothing
CAUTION: The delay must be an absolute minimum of 10us
***********************************************************************/
void comparators_13usdelay
  (
  void
  )
{
  // ensure timer 0 stopped
  TR0 = 0;
  // configure timer 0 as 16-bit timer
  TMOD &= 0xF0;
  TMOD |= 0x01;
  TAMOD &= 0xFE;
  // set reload value
  TH0 = 0xFF;
  TL0 = 0xB2;
  // disable timer interrupt
  ET0 = 0;
  // run timer and wait for overflow
  TF0 = 0;
  TR0 = 1;
  while (!TF0);
  // stop timer and clean up
  TR0 = 0;
  TF0 = 0;
}

/***********************************************************************
DESC:    Initializes a comparator
         Selects the comparator inputs/reference voltage source, enables
         comparator output, enables comparator, configures I/O pins
         needed, enables interrupts
         If a comparator is being enabled then comparators_13usdelay
         is called to provide a 13us delay to stabilize the comparator
RETURNS: Nothing
CAUTION: Set EA to 1 to enable interrupts after calling
***********************************************************************/
void comparators_init
```

```
  (
  bit compnum,                    // comparator number:     COMP_1 or COMP_2
  unsigned char posinput,         // positive input A or B: COMP_INPUTA or COMP_INPUTB
  unsigned char neginput,         // negative input CMPREF or internal reference:
                                  //                        COMP_INPUTREF, COMP_INTERNALREF
  unsigned char outputenable      // enable or disable output pin:
                                  //                        COMP_OUTPUTDISABLE or
COMP_OUTPUTENABLE
  )
{
  bit currenable;

  if (compnum == COMP_1)
  {
    // initialize port pins according to configuration
    if (posinput == COMP_INPUTA)
    {
      // select CIN1A as analog input
      P0M1 |= 0x10;
      P0M2 &= ~0x10;
      PT0AD |= 0x10;
    }
    else
    {
      // select CIN1B as analog input
      P0M1 |= 0x08;
      P0M2 &= ~0x08;
      PT0AD |= 0x08;
    }
    if (neginput == COMP_INPUTREF)
    {
      // select CMPREF as analog input
      P0M1 |= 0x20;
      P0M2 &= ~0x20;
      PT0AD |= 0x20;
    }
    if (outputenable == COMP_OUTPUTENABLE)
    {
      // select CMP1 as push-pull output
      P0M1 &= ~0x40;
      P0M2 |= 0x40;
    }
    // find out if comparator is currently enabled or not
    currenable = CMP1 & 0x20;
    // configure and enable comparator
    // clear interrupt flag
    CMP1 = posinput | neginput | outputenable | 0x20;
    // if comparator just enabled then we need to call a function
    // so the user can generate a 13us delay
    if (!currenable) comparators_13usdelay();
    // clear comparator interrupt flag to avoid spurious interrupt
```

```
    CMP1 &= ~0x01;
  }
  else
  {
    // initialize port pins according to configuration
    if (posinput == COMP_INPUTA)
    {
      // select CIN2A as analog input
      P0M1 |= 0x04;
      P0M2 &= ~0x04;
      PT0AD |= 0x04;
    }
    else
    {
      // select CIN2B as analog input
      P0M1 |= 0x02;
      P0M2 &= ~0x02;
      PT0AD |= 0x02;
    }
    if (neginput == COMP_INPUTREF)
    {
      // select CMPREF as analog input
      P0M1 |= 0x20;
      P0M2 &= ~0x20;
      PT0AD |= 0x20;
    }
    if (outputenable == COMP_OUTPUTENABLE)
    {
      // select CMP2 as push-pull output
      P0M1 &= ~0x01;
      P0M2 |= 0x01;
    }
    // find out if comparator is currently enabled or not
    currenable = CMP2 & 0x20;
    // configure and enable comparator
    // clear interrupt flag
    CMP2 = posinput | neginput | outputenable | 0x20;
    // if comparator just enabled then we need to call a function
    // so the user can generate a 13us delay
    if (!currenable) comparators_13usdelay();
    // clear comparator interrupt flag to avoid a spurious interrupt
    CMP2 &= ~0x01;
  }

  // set isr priority to 0
  IP1 &= 0xFB;
  IP1H &= 0xFB;

  // enable comparator interrupt
  EC = 1;
}
```

```
/************************************************************************
DESC:    Comparator Interrupt Service Routine
         Uses register bank 1
RETURNS: Nothing
CAUTION: comparators_init must be called and EA set to 1 to enable
         interrupts.
         Called when the output of any enabled comparator changes
************************************************************************/
void comparators_isr
  (
  void
  ) interrupt 8 using 1
{
  // check if comparator 1 caused interrupt
  if (CMP1 & 0x01)
  {
    // clear interrupt flag
    CMP1 &= ~0x01;
  }
  // check if comparator 2 caused interrupt
  if (CMP2 & 0x01)
  {
    // clear interrupt flag
    CMP2 &= ~0x01;
  }
}

/************************************************************************
DESC:    Disables a comparator
RETURNS: Nothing
CAUTION: The port pins used by the comparator are not reconfigured to
         be digital inputs or outputs.
************************************************************************/
void comparators_disable
  (
  bit compnum                     // comparator number:     COMP_1 or COMP_2
  )
{
  // disable comparator 1
  if (compnum == COMP_1)
  {
    CMP1 &= ~0x20;
  }
  // disable comparator 2
  else
  {
    CMP2 &= ~0x20;
  }
}
```

```
/*************************************************************************
DESC:    Gets the current output of a comparator
RETURNS: Current comparator output
CAUTION: comparators_init must be called first
*************************************************************************/
bit comparators_getoutput
  (
  bit compnum                    // comparator number:     COMP_1 or COMP_2
  )
{
  // get output of comparator 1
  if (compnum == COMP_1)
  {
    return (CMP1 >> 1) & 0x01;
  }
  // get output of comparator 2
  else
  {
    return (CMP2 >> 1) & 0x01;
  }
}

/*************************************************************************
DESC:    Selects a positive input source for a comparator
RETURNS: Nothing
CAUTION: comparators_init must be called first.
         The comparator interrupt is disabled while the input is
         changed. This means that the other comparator not being changed
         will also not generate an interrupt.
*************************************************************************/
void comparators_selectposinput
  (
  bit compnum,                   // comparator number:     COMP_1 or COMP_2
  unsigned char posinput         // positive input A or B: COMP_INPUTA or COMP_INPUTB
  )
{
  // disable comparator interrupt
  EC = 0;

  // configure comparator 1
  if (compnum == COMP_1)
  {
    // initialize port pins according to configuration
    if (posinput == COMP_INPUTA)
    {
      // select CIN1A as analog input
      P0M1 |= 0x10;
      P0M2 &= ~0x10;
      PT0AD |= 0x10;
    }
    else
```

```
    {
      // select CIN1B as analog input
      P0M1 |= 0x08;
      P0M2 &= ~0x08;
      PT0AD |= 0x08;
    }
    // clear input selection
    CMP1 &= ~0x10;
    // select new input
    CMP1 |= posinput;
  }
  // configure comparator 2
  else
  {
    // initialize port pins according to configuration
    if (posinput == COMP_INPUTA)
    {
      // select CIN2A as analog input
      P0M1 |= 0x04;
      P0M2 &= ~0x04;
      PT0AD |= 0x04;
    }
    else
    {
      // select CIN2B as analog input
      P0M1 |= 0x02;
      P0M2 &= ~0x02;
      PT0AD |= 0x02;
    }
    // clear input selection
    CMP2 &= ~0x10;
    // select new input
    CMP2 |= posinput;
  }

  // enable comparator interrupt
  EC = 1;
}
```

5.3 Appendix C: comparators.h

```
/*************************************************************************
MODULE:    Comparators
VERSION:   1.02
CONTAINS:  Routines for controlling the comparators on the Philips
           P89LPC935
COPYRIGHT: Embedded Systems Academy, Inc.
LICENSE:   May be freely used in commercial and non-commercial code
           without royalties provided this copyright notice remains
           in this file and unaltered
WARNING:   IF THIS FILE IS REGENERATED BY CODE ARCHITECT ANY CHANGES
```

```
          MADE WILL BE LOST. WHERE POSSIBLE USE ONLY CODE ARCHITECT
          TO CHANGE THE CONTENTS OF THIS FILE
GENERATED: On "Aug 15 2005" at "13:49:35" by Code Architect 2.11
*****************************************************************************/

#ifndef _COMPARATORSH_
#define _COMPARATORSH_

// values passed to the comparator functions
#define COMP_1             0      // comparator 1
#define COMP_2             1      // comparator 2
#define COMP_INPUTA        0x00   // comparator input CINnA
#define COMP_INPUTB        0x10   // comparator input CINnB
#define COMP_INPUTREF      0x00   // reference voltage input CMPREF
#define COMP_INTERNALREF   0x08   // internal reference voltage Vref
#define COMP_OUTPUTDISABLE 0x00   // disable comparator output pin
#define COMP_OUTPUTENABLE  0x04   // enable comparator output pin

/*****************************************************************************
DESC:    Initializes a comparator
         Selects the comparator inputs/reference voltage source, enables
         comparator output, enables comparator, configures I/O pins
         needed, enables interrupts
         If a comparator is being enabled then comparators_13usdelay
         is called to provide a 13us delay to stabilize the comparator
RETURNS: Nothing
CAUTION: Set EA to 1 to enable interrupts after calling
*****************************************************************************/
extern void comparators_init
  (
  bit compnum,                    // comparator number:     COMP_1 or COMP_2
  unsigned char posinput,         // positive input A or B: COMP_INPUTA or COMP_INPUTB
  unsigned char neginput,         // negative input CMPREF or internal reference:
                                  //                        COMP_INPUTREF, COMP_INTERNALREF
  unsigned char outputenable      // enable or disable output pin:
                                  //                        COMP_OUTPUTDISABLE or
COMP_OUTPUTENABLE
);

/*****************************************************************************
DESC:    Disables a comparator
RETURNS: Nothing
CAUTION: The port pins used by the comparator are not reconfigured to
         be digital inputs or outputs.
*****************************************************************************/
extern void comparators_disable
  (
  bit compnum                     // comparator number:     COMP_1 or COMP_2
  );

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

```
DESC:    Gets the current output of a comparator
RETURNS: Current comparator output
CAUTION: comparators_init must be called first
*****************************************************************************/
extern bit comparators_getoutput
  (
  bit compnum                    // comparator number:     COMP_1 or COMP_2
  );

/*****************************************************************************
DESC:    Selects a positive input source for a comparator
RETURNS: Nothing
CAUTION: comparators_init must be called first.
         The comparator interrupt is disabled while the input is
         changed. This means that the other comparator not being changed
         will also not generate an interrupt.
*****************************************************************************/
extern void comparators_selectposinput
  (
  bit compnum,                   // comparator number:     COMP_1 or COMP_2
  unsigned char posinput         // positive input A or B: COMP_INPUTA or COMP_INPUTB
  );

#endif // _COMPARATORSH_
```

6. 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.

7. Trademarks

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

8. Contents

Date of release: 3 February 2006
Document number: AN10391_1

Published in The Netherlands