AN10535

LPC2138 extreme power down application note

Rev. 01 — 6 December 2006 **Application note**

Document information

Info	Content
Keywords	LPC2138, extreme power down
Abstract	This document describes a method to power down the LPC2138 so the power down current is less than 1 μA. This method requires a few simple external components

Revision history

Rev	Date	Description
01	20061206	Initial version.

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 LPC2138 is a high performance single supply ARM7 microcontroller, which has several power-down modes that are used to conserve power when the microcontroller is waiting for something to do. In power-down mode, the LPC2138 consumes about 60 μA from the 3.3 V supply at room temperature with the brown out enabled and around 30 μA when the brown out is disabled. This is relatively good considering that this part is constructed using a deep sub micron process. However, at high temperatures the leakage current increases significantly.

The purpose of this app note is to describe a low cost method to have extremely low leakage currents over temperature when using an LPC2138. This method requires a few external components, but it provides significant leakage current reduction.

This app note will discuss two methods that restore the microcontroller's state previous to power down. One method uses an inexpensive external EEPROM, and the other uses an existing sector of the internal flash that can be reserved for EEPROM emulation. Both methods will store the microcontroller's state into the non-volatile memory before shutdown and will restore the information back into the internal RAM, so the microcontroller may resume processing were it left off before power down.

2. Description

As stated in the introduction, the lowest possible leakage current of the LPC2138 is about 30 μA at room temperature with the brown out disabled. By itself, there is not much else the user can do to improve this without external components. One solution is to disconnect the power source from the LPC2138 using an external switch, which the LPC2138 controls. The concept would be to have an inexpensive PNP transistor control the power to the LPC2138 and flip-flop that the LPC2138 can control to turn the power off.

2.1 Block diagram

Fig 1 shows a simplified diagram of the circuit concept. When power is first applied the flip-flop is reset via an RC time constant on the reset pin. This insures that the LPC2138 comes up with the power applied. The LPC2138 can shut itself off through a port pin. This example uses port P0.23. Note that the port pins of the LPC2138 come up in a high impedance state. Therefore, the set of the flip-flop is pulled high by a pull up resistor, so there is no conflict at power up with the reset pin. Once the LPC2138 shuts itself down, an external event can wake the part back up. In this case it is a push of a switch.

The concept is to have the microcontroller turn off its own power and then have an external event reapply power. One key requirement is to have the microcontroller store its state in non-volatile memory before power is removed and have the microcontroller restore its state after power is reapplied and continue on where it was before it went into deep power-down.

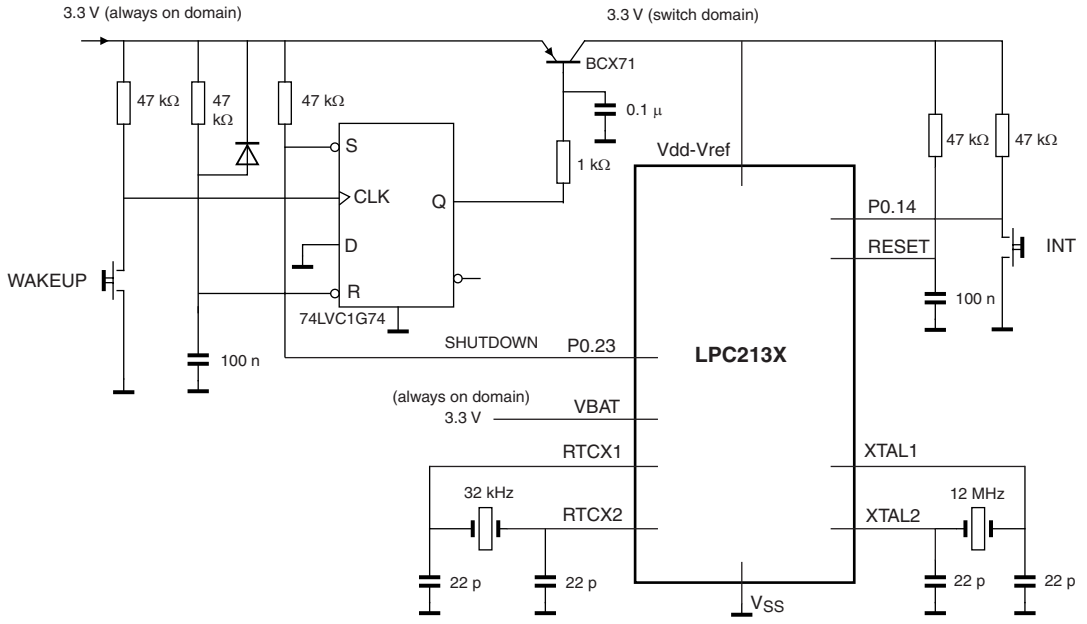

Fig 1. Block diagram

For detailed schematics, see Section 11, appendix C.

3. Measured data

A board was produced according to the schematics in Section 11 and was used to take the following measurements.

3.1 Off current data

When the LPC2138 is switched off using the external control circuit, the following currents were measured.

Room temp current < 100 nA

125 °C data < 200 nA

3.2 Startup time

As shown in the Fig 2 below, the start up time is 856 µs +/– 2 %. The power signal is the top signal shown (labeled POWER) followed by the RESET, which rises with an RC constant. As a sufficient number of cycles are counted from the crystal, the microcontroller will start the execution of a stored code as shown by the P0.0 output pin set high.

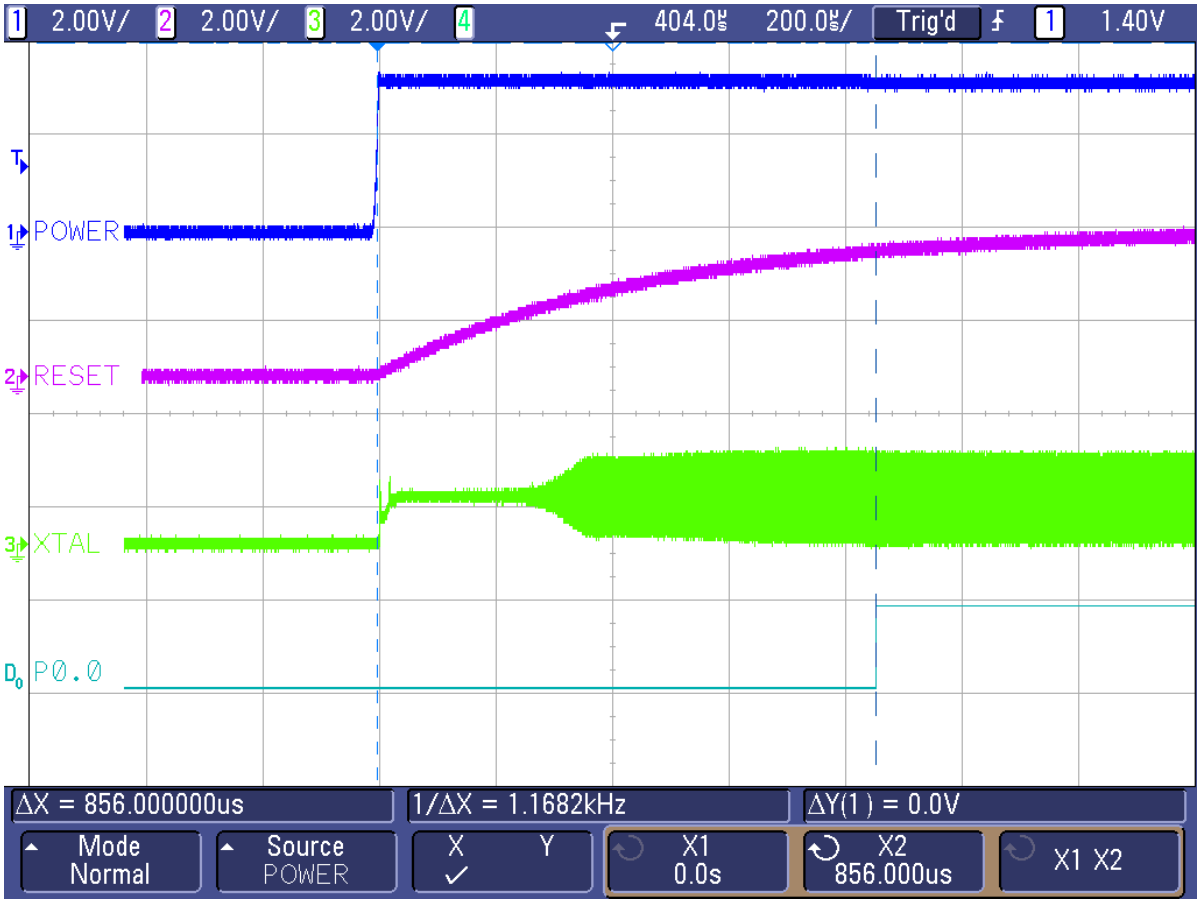

Fig 2. Time to start up

3.3 Current drawn

The measured current into the system during normal operation, with the MCU working at 60 MHz, is between 60 mA to 68 mA. During extreme power-down mode, the system draws less than 100 nA.

3.4 Transistor voltage drop

The voltage drop between the emitter and collector of the PNP transistor is 42 mV during normal operation, while 2.9 V during extreme power down.

3.5 GPIO pins

When using this method to conserve power it is important to make sure the always-on domains and the switched domains are isolated properly. If they are not then current can flow from the always-on domain to the switched domain and increase power consumption. The LPC2000 port pins do not have diodes to VDD so the always-on domain can drive them with the LPC2138 powered off without drawing any current from the always-on domain. However, the port pins are not 5 V tolerant when VDD is not present.

To determine the characteristics of the GPIO port pins during extreme power down, a current meter is used in between the GPIO port pin and a short to 3.3 V to determine sinking current. Also, the current meter checks the source current by shoring to ground as shown in Fig 3.

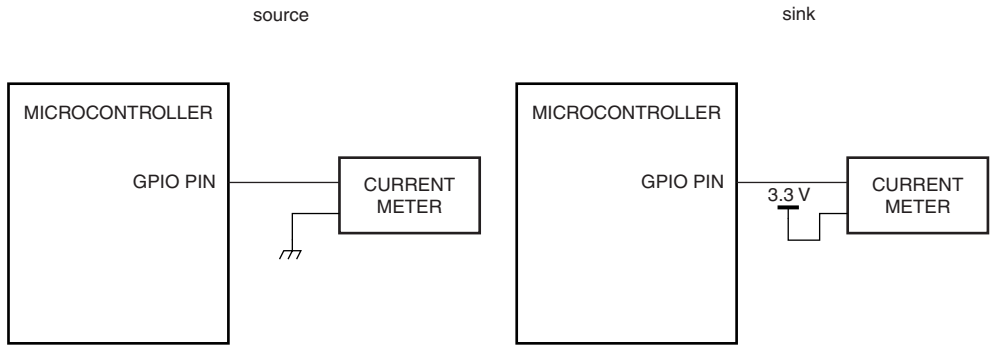

Fig 3. Testing for sourcing and sinking at GPIO port pins

During power-down mode, the GPIO port pins should not source or sink any current, except for the I^2C pins on the DAC pin. Pins p0.2, p0.3, p0.11 are I^2C pins, which will sink roughly 11 µA of current due to the open drain nature of the pins. P0.25, noted by *, is the DAC output and cannot be driven when the part is turned off.

Table 1. Checking GPIO pins for current in power-down mode

	To VDD (µA)	To GND (µA)		To VDD (µA)	To GND (µA)
p0.0	0	0	p0.16	0	0
p0.1	0	0	p0.17	0	0
p0.2	−11	0	p0.18	0	0
p0.3	−11	0	p0.19	0	0
p0.4	0	0	p0.20	0	0
p0.5	0	0	p0.21	0	0
p0.6	0	0	p0.22	0	0
p0.7	0	0	p0.23	0	0
p0.8	0	0	p0.25	*	*
p0.9	0	0	p0.26	0	0
p0.10	0	0	p0.27	0	0
p0.11	−11	0	p0.28	0	0
p0.12	0	0	p0.29	0	0
p0.13	0	0	p0.30	0	0
p0.14	0	0	p0.31	0	0
p0.15	0	0	p0.32	0	0

4. Example programs and types of storage

4.1 External EEPROM

An external EEPROM using the SPI interface may be used to store state information in the case of an extreme power-down mode. Essential information that is stored into the EEPROM before shut down and at startup, will be recalled and written into the internal RAM in order to continue where it left off.

The setup connection is as shown in the detailed schematics, see Section 11.

4.1.1 Size

The EEPROM used for this application note is 16 kbit. However, common sizes on the market can range from 1 kbit to 256 kbit. An appropriate size of the external EEPROM should be used to store essential data for resuming normal operation.

4.1.2 Speed

The writing and reading speed to the external EEPROM is important because information needs to be transferred quickly to reduce turn off and start-up times. The EEPROM in this application note has a maximum operating frequency of 10 MHz. Fig 4 shows some operating waveforms.

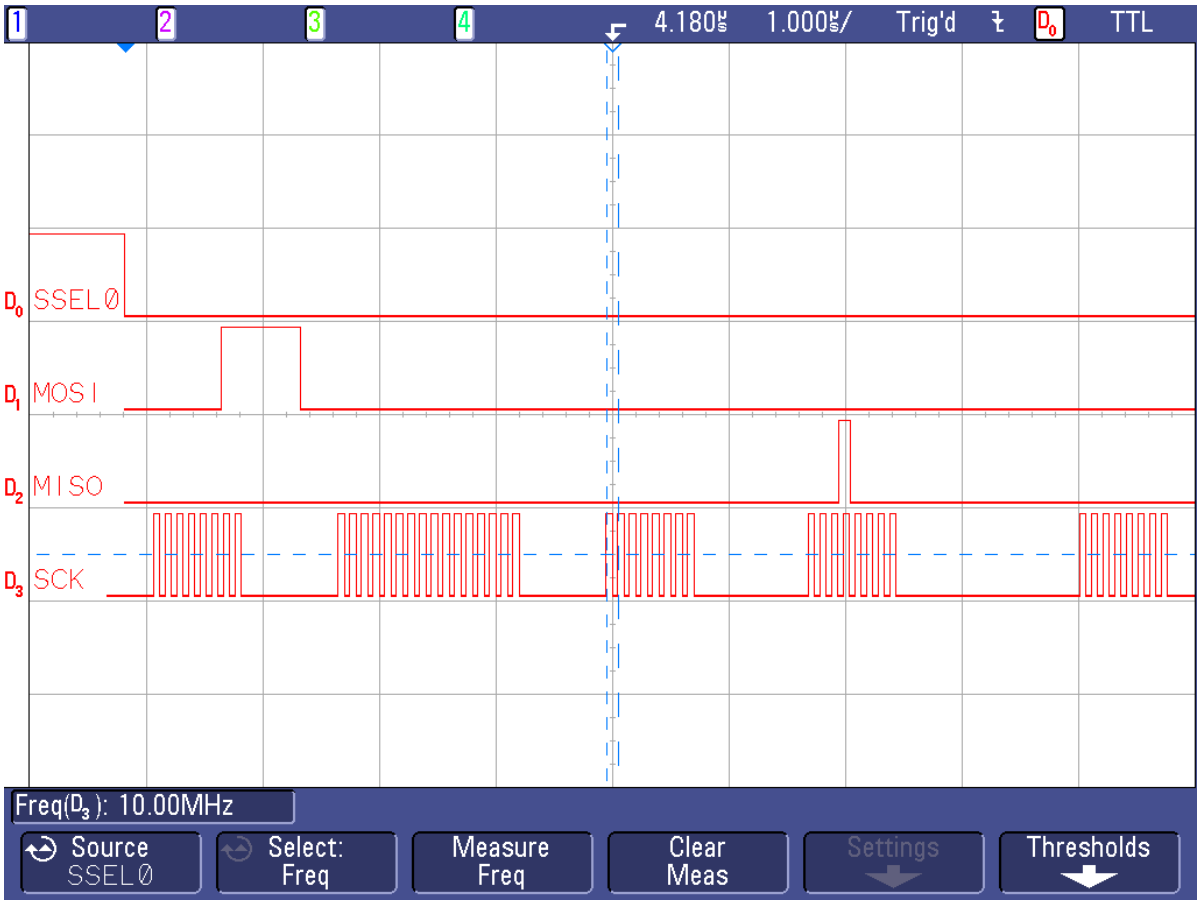

Fig 4. Speed of the SPI SCK signal

4.1.3 Example C program

The example C program in Section 9 shows the basic steps to use the SPI interface for storing essential information to an external EEPROM on demand. During the next start up, the previous conditions are restored into the internal RAM, so the microcontroller can continue processing.

The simple C program blinks and increments eight LEDs continuously in a loop. If the interrupt button is pressed, the current LED, which corresponds to a bit, is stored into the external EEPROM. During startup, this bit, which determined which LED the program was blinking last, is restored into the internal RAM and the program will continue where it left off.

This small program demonstrates the basic procedures for storing and loading conditions that would work for even larger programs with more variables and conditions.

4.2 Internal EEPROM emulation

EEPROM emulation can be used to store startup information into the FLASH before the part is put into the powered down mode with the existing flash memory. This will allow the part to start-up in a known state after power is applied.

4.2.1.1 Example C program

An example C program for the internal Flash storage is shown in Appendix B, Section 9. The program is initialized by the reorganization of the Flash to emulate an EEPROM.

4.2.1.2 Time/settings

The program shows that after the initialization of the real time clock, information such as seconds, minutes, and hours can be stored in to the portion of the internal flash and can be recalled and stored into ram when necessary. This shows how registers can be stored and recalled using this method.

4.2.2 Log information

A counter is used to increment each time the system is shutdown and restarted, which keeps a log of the number of times the cycle occurs.

5. BOM costs for power down components

Shown in Table 2 are the components needed to create an external circuitry for extreme power-down mode. The total additional cost is about 13 cents.

Table 2. Component costs

Quantity	Component	Value	Device	Package	Price
1	D-type flip flop	--	74LVC1G74	SOT765-1	0.05
1	Diode	--	MMSD4148T1	SOD123	0.03
3	Resistor	47 k	R-US_R0805	R0805	0.001
1	Resistor	1 k	R-US_R0805	R0805	0.001
1	Capacitor	0.01 uF	C-USC0805	C0805	0.01
1	Capacitor	0.1 uF	C-USC0805	C0805	0.01
1	PNP Transistor	--	BCX71SMD	SOT23	0.03
				Total cost	0.132

6. Board area required

6.1 Estimated square surface area for components alone

Table 3 shows the estimated square surface area for components only, to which a small margin needs to be added to meet the bare minimum area required to comply with DRC rules.

Table 3. Estimated square surface area for components only

Component	Surface Area	Quantity	Package
D-type flip flop	3.2 x 2.4 mm	1	SOT765-1
Diode	3.85 x 1.8 mm	1	SOD123
Resistor	4.0 x 2.02 mm	3	R0805
Resistor	4.0 x 2.02 mm	1	R0805
Capacitor	4.0 x 2.02 mm	1	C0805
Capacitor	4.0 x 2.02 mm	1	C0805
PNP Transistor	3.61 x 3 mm	1	SOT23

6.2 Surface area for demo board including support components

An example using the support components for extreme power-down mode is shown in Fig 5. However, a current sensing circuit is also included, along with a LCD and LEDs. The actual circuitry for the extreme power-down mode consists of D1, R8, C12, R9, R7, C11, R13, C6, R17, R23, C16, T1, the DFF, and the two switches, which all take up a relatively small area considering the area of the total number of components on this demo board.

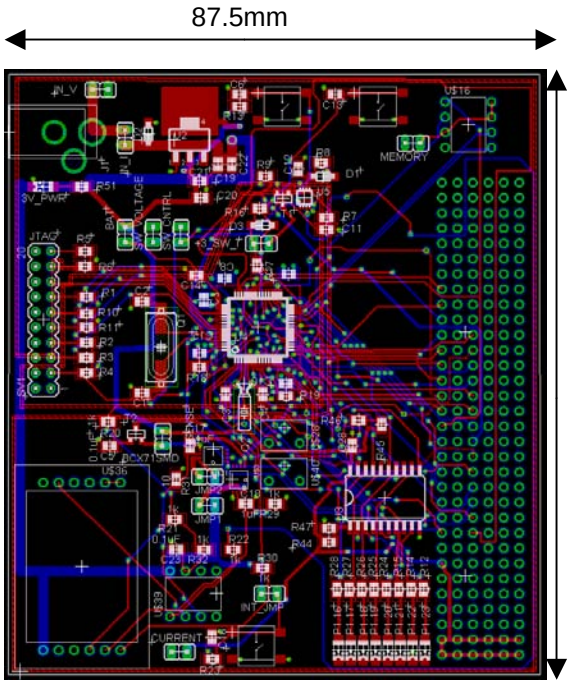

Fig 5. Example board surface area including support components

7. Stress analysis

In order to prevent damage to the components, the stress with respect to the given ranges are checked to be reasonable, as shown in Table 4.

Table 4. Stress analysis

Id	Component	Value	Device	Package	Description	Range	Actual
	D-type flip flop	--	74LVC1G74	SOT765-1	Vcc- supply	1.65 V to 5.5 V	3.29 V
					D input	0 V to 5.5 V	3.28 V
D1	Diode	--	MMSD41T1	SOD123	ReverseVolt	100 V	20 mV
R8	Resistor	47 k	R-US_R0805	R0805	Voltage drop	--	20 mV
					Current	--	0.425 uA
					Power	100 mW	8.5 nW
R9	Resistor	47 k	R-US_R0805	R0805	Voltage drop	--	3.28 V
					Current	--	69 uA
					Power	100 mW	0.23 mW
C12	Capacitor	0.01 uF	C-USC0805	C0805	Voltage	0 V to 50 V	3.29 V

Id	Component	Value	Device	Package	Description	Range	Actual
					rating		
	PNP Transistor	--	BCX71SMD	SOT23	collector current	–500 mA	–68 mA
R7	Resistor	1 k	R-US_R0805	R0805	Voltage drop	--	2.482 V
					Current	--	2.48 mA
					Power	100 mW	6.2 mW
C11	Capacitor	0.1 uF	C-USC0805	C0805	Voltage rating	0 V to 50 V	3.29 V

8. Tips to achieve lowest power down currents

When mixing an always-on domain with a switched domain, it is important to look at all signals that cross the boundaries. For signals that drive into the switch domain make sure there are no current paths into the switched domain that can increase power consumption when the switched domain is turned off. For signals that drive out of the switched domain make sure that the signals to the on domain are controlled. When the switched domain is turned off these signals will be left floating. As an example, if an always on domain SPI ram is being driven by the always off domain, make sure that the chip select signals are pulled high to the always on domain or this pin will be left floating when the switched domain is turned off causing the SPI ram to draw current.

9. Appendix A

```
1   /*********************************************************************************************/
2   /*         shut_down.c - Program for 213x: Cycles the blinking of the 8 LEDS and             */
3   /*         if interrupt is detected, stores the current bit into external EEPROM             */
4   /*         and goes into extreme power-down mode. During the next startup,                    */
5   /*         the EEPROM data is restored into RAM and continues to where the                   */
6   /*         program left off.                                                                 */
7   /*                                                                                           */
8   /*********************************************************************************************/
9
10  #include <LPC213x.H>                            //LPC213x definitions
11  #define S0SPIF (1<<7)
12
13  void Initialize(void);                          //Initialize SPI EEPROM
14  void write(int, int);                       //Write function
15  void read(int);                             //Read function
16  void initread(int);                         //Initialize read
17  void memcheck(void);                        //Memory check function counts number of bits
18                                          //in location 16-23 of the first four bytes
19  int byte, var, z, init, writevar, currentloc, loc, limit;   //initialize variables
20  int shiftvar = 0xFF000000;                    //  "            "
21  int inc,initial,m,p;                          //  "            "
22  int k,l;                                  //  "            "
23  unsigned int j;                               //  "            "
```

```
int cnt, n=300000;                              //  "            "
int mask, set, READ, recall, Var, readloc;           //  "            "

void main (void)
{
     IODIR0  = 0x00000081;           //P0.1 & P0.7 are output pins
     IOSET0  = 0x00000001;           //Testing pin, when data will first come out
     IODIR1  = 0x00FF0000;           //Initializing LEDs

     initread(0x0);                  //Calling function to initialize
                                  //EEPROM for reading
     read(0x0);                  //Reading current data in external EEPROM

     memcheck();                     //memory check
if (inc==1)                          //checks if memory has valid data
     {
          mask = Var;                //Restores into internal RAM data from external EEPROM
          while (1)
          {
               if ((IOPIN0 & 0x00004000) == 0x00000000)  //check p0.14 is pushed
                                        // Looks for the interrupt button to be pushed
                    {
                    IOCLR0 = 0x00000001;       //Turning off test pin to show in shutdown
                    write(mask, 0x0);          //Store data to external EEPROM
                    IODIR0 = 0x00800000;       //set port 0.23 as an out.
                    IOCLR0 = 0x00800000;  //Sets extreme power-down mode
                    }
               IOSET1 = mask;                  //Continue blinking where it left off
               for (cnt = 0; cnt < n; cnt++);          //Delay
               IOCLR1 = 0x00FF0000;            //Turns off LED
               for (cnt = 0; cnt < n; cnt++);          //Delay
               mask = mask << 1;               //Shifts to next LED bit
                    if (mask != 0x01000000)         //checks if last location
                    {
                    IOSET1 = mask;             //blinks next LED
                    for (cnt = 0; cnt < n; cnt++);   //Delay
                    }
                    else
                    mask=0x00010000;      //If LED was last bit, this will set to first
          }
     }
else
     {
          mask = 0x00010000;         //If external EEPROM was empty, this will initialize
                                                  //first bit
               while (1)
          {
               if ((IOPIN0 & 0x00004000) == 0x00000000)    //check p0.14 is pushed
                                        // Looks for the interrupt button to be pushed
                    {
                    IOCLR0 = 0x00000001;            //Turning off test pin to show in shutdown
```

```
75                        write(mask, 0x0);          //Store data to external EEPROM
76                        IODIR0 = 0x00800000;            //set port 0.23 as an out.
77                        IOCLR0 = 0x00800000;       //Sets extreme power-down mode
78                        }
79                  IOSET1 = mask;                   //Continue blinking where it left off
80                  for (cnt = 0; cnt < n; cnt++);        //Delay
81                  IOCLR1 = 0x00FF0000;             //Turns off LED
82                  for (cnt = 0; cnt < n; cnt++);        //Delay
83                  mask = mask << 1;                //Shifts to next LED bit
84                        if (mask != 0x01000000)         //checks if last location
85                        {
86                        IOSET1 = mask;             //blinks next LED
87                        for (cnt = 0; cnt < n; cnt++);   //Delay
88                        }
89                        else
90                        mask=0x00010000;      //If LED was last bit, this will set to first
91             }
92        }
93
94   }
95
96   //Checking the memory//////////////////////////////////////////////////////////////////////////
97   void memcheck(void)                              //Memory Check
98   {
99        inc=0;                                      //Initialize
100       initial = 0x00010000;                       //Initialize
101       for (p=0; p<8; p++)                         //Loops through all 8 bits essential to LED
102       {
103            m = Var;                         //Stores EEPROM data into new variable
104            m = m & initial;                 //Bitwise add
105            initial = initial << 1;                  //Shifts reference bit to next
106            m = m >> 16+p;                      //Checks if bit in new variable is 1
107            if (m == 0x00000001)
108            {                                //Increments
109            inc++;
110            }
111       }
112  }
113
114  //Initializing SPI/////////////////////////////////////////////////////////////////////////////
115  void Initialize()
116  {
117       PINSEL0=0x1500;                             //Configure Pin Connect Block
118       VPBDIV=0x1;                                 //Set pclk to same as cclk
119       S0SPCCR=0x6;                                //Set to highest speed for SPI at 10 MHz
120       S0SPCR=0x20;                                //Device selected as master
121  }
122
123  //Writing to EEPROM////////////////////////////////////////////////////////////////////////////
124  void write(int var, int writeloc)
125  {
```

```
126
127                 Initialize();                   //Initialize EEPROM
128                 IODIR0  = 0x00000080;           //Initialize chip select
129
130                 init = S0SPSR;                  //Initialize status
131                 init = S0SPDR;                  //Initialize data reg
132
133                 IOCLR0 = 0x00000080;            //Initialize write enable
134
135                 S0SPDR = 0x06;                  //Write latch enable command
136                 while((S0SPSR & S0SPIF)==0);          //Check if command is sent
137
138                 IOSET0 = 0x00000080;                  //Complete write enable
139                 IOCLR0 = 0x00000080;            //Enable chip select
140
141                 S0SPDR = 0x02;                  //Write command
142                 while((S0SPSR & S0SPIF)==0);          //Check if command is sent
143
144                       S0SPCR = 0x24;            //Enable 16-bit
145
146                 S0SPDR = writeloc;              //Write location
147                 while((S0SPSR & S0SPIF)==0);          //Check if command is sent
148
149                       S0SPCR = 0x20;            //Enable 8-bit
150
151                 for (z=0; z<25 ; z=z+8)               //Write loop for 4 cycles of 32-bit data
152                 {
153                 shiftvar = shiftvar >> z;
154                 byte = var & shiftvar;
155                 byte = byte >> 24-z;
156
157                 S0SPDR = byte;
158                 while((S0SPSR & S0SPIF)==0);
159                 }
160
161                 IOSET0 = 0x00000080;            //Disable chip select
162   }
163
164   //Initializing EEPROM for first read//////////////////////////////////////////////////////////////////////
165   void initread(int readloc)
166   {
167                 Initialize();                   //Initialize EEPROM
168                 IODIR0  = 0x00000081;           //Initialize chip select and test pin (p0.0)
169
170                 init = S0SPSR;                  //Initialize status
171                 init = S0SPDR;                  //and data register
172
173                 IOCLR0 = 0x00000080;            //Enable chip select
174
175                 S0SPDR = 0x03;                  //Read command
176                 while((S0SPSR & S0SPIF)==0);          //Check if command is sent
```

```
177
178                       S0SPCR = 0x24;             //Enable 16-bit
179
180                  S0SPDR = readloc;                    //Read location
181                  while((S0SPSR & S0SPIF)==0);         //Check if command is sent
182
183                       S0SPCR = 0x20;             //Enable 8-bit
184
185                  S0SPDR = 0x0;                   //Read first 8-bits
186                  while((S0SPSR & S0SPIF)==0);         //Check if command is sent
187                  READ = S0SPDR;                  //Store into read variable
188                  Var=READ<<24;                   //Shift to next 8-bits
189
190                  S0SPDR = 0x0;                   //Read Second 8-bits
191                  while((S0SPSR & S0SPIF)==0);         //Check if command is sent
192                  READ = S0SPDR;                  //Store into read variable
193                  READ= READ<<16;
194                  Var =  Var & 0xFF000000;
195                  READ = READ& 0x00FF0000;
196                  Var = Var | READ;
197
198                  S0SPDR = 0x0;                   //Read Second 8-bits
199                  while((S0SPSR & S0SPIF)==0);         //Check if command is sent
200                  READ = S0SPDR;                  //Store into read variable
201                  READ= READ<<8;
202                  Var =  Var & 0xFFFF0000;
203                  READ = READ& 0x0000FF00;
204                  Var = Var | READ;
205
206                  S0SPDR = 0x0;                   //Read Second 8-bits
207                  while((S0SPSR & S0SPIF)==0);         //Check if command is sent
208                  READ = S0SPDR;                  //Store into read variable
209                  READ= READ<<0;
210                  Var =  Var & 0xFFFFFF00;
211                  READ = READ& 0x000000FF;
212                  Var = Var | READ;
213
214                  IOSET0 = 0x00000081;            //Disable chipselect
215  }
216  //Reading from EEPROM////////////////////////////////////////////////////////////////////////////
217  void read(int readloc)
218  {
219                  Initialize();                   //Initialize EEPROM
220                  IODIR0  = 0x00000081;           //Initialize chip select and test pin
221
222                  init = S0SPSR;                  //Initialize status
223                  init = S0SPDR;                  //and data register
224
225                  IOCLR0 = 0x00000080;            //Enable chip select
226
227                  S0SPDR = 0x03;                  //Read command
```

```
228                 while((S0SPSR & S0SPIF)==0);          //Check if command is sent
229
230                       S0SPCR = 0x24;             //Enable 16-bit
231
232                 S0SPDR = readloc;                     //Read location
233                 while((S0SPSR & S0SPIF)==0);          //Check if command is sent
234
235                       S0SPCR = 0x20;             //Enable 8-bit
236
237                 S0SPDR = 0x0;                   //Read first 8-bits
238                 while((S0SPSR & S0SPIF)==0);          //Check if command is sent
239                 READ = S0SPDR;                  //Store into read variable
240                 Var=READ<<24;                   //Shift to next 8-bits
241
242                 S0SPDR = 0x0;                   //Read Second 8-bits
243                 while((S0SPSR & S0SPIF)==0);          //Check if command is sent
244                 READ = S0SPDR;                  //Store into read variable
245                 READ= READ<<16;
246                 Var =  Var & 0xFF000000;
247                 READ = READ& 0x00FF0000;
248                 Var = Var | READ;
249
250                 S0SPDR = 0x0;                   //Read Second 8-bits
251                 while((S0SPSR & S0SPIF)==0);          //Check if command is sent
252                 READ = S0SPDR;                  //Store into read variable
253                 READ= READ<<8;
254                 Var =  Var & 0xFFFF0000;
255                 READ = READ& 0x0000FF00;
256                 Var = Var | READ;
257
258                 S0SPDR = 0x0;                   //Read Second 8-bits
259                 while((S0SPSR & S0SPIF)==0);          //Check if command is sent
260                 READ = S0SPDR;                  //Store into read variable
261                 READ= READ<<0;
262                 Var =  Var & 0xFFFFFF00;
263                 READ = READ& 0x000000FF;
264                 Var = Var | READ;
265
266                 IOSET0 = 0x00000081;            //Disable chipselect
267   }
268
269   ////////////////////////////////////////////////////////////////////////////////////////////////////
```

10. Appendix B

```
1   /*****************************************************************************************/
2   /*                                                                                       */
3   /*  EE_demo.C:  Use of LPC213x on-chip Flash as an EEPROM.                               */
4   /*  For details on an EEPROM specifications, see LPC2k_ee.h file.                        */
5   /*                                                                                       */
6   /*  Following functions are specified in LPC2k_ee.c file:                                */
7   /*                                                                                       */
8   /* ee_erase(command_ee, result_ee[]): erases all EEPROM                                  */
9   /* ee_write(command_ee, result_ee[]): writes record of ee_data (defined in LPC2k_ee.h)  */
10  /* ee_read(command_ee, result_ee[]) : reads the last record added into EEPROM           */
11  /* ee_readn(command_ee, result_ee[]): reads the n-th record in EEPROM                   */
12  /* ee_count(command_ee, result_ee[]): counts records of ee_data type in EEPROM          */
13  /*                                                                                       */
14  /*****************************************************************************************/
15
16  #include <LPC213x.h>                    /* LPC213x definitions */
17  #include <LPC2K_EE.H>           /* LPC2000 EEPROM definitions */
18  #include <stdio.h>
19
20  void _display_time(short int);
21
22  void main (void){
23
24  volatile unsigned int status, records0, loop_cnt;
25
26  struct ee0_data ee0_temp, ee0_read, *ee0_pnt;
27  const struct ee0_data ee0_ini = {EE_REC_ID,0,0x0000,0x00000000,0x00000000,0x00000000};
28
29  unsigned int command_ee, response_ee[2];
30
31  //pin configuration section
32       PINSEL0 = (PINSEL0 & 0x0FFFFFF0) | 0x00000005;    //P0.01=RxD0,P0.00=TxD0
33
34  //UART0 setup: PC communication
35       U0LCR = 0x80;                    //enable latch register access
36       U0DLL = 0xC3;                    //UART0 operates at the...
37       U0DLM = 0x00;                    //...19200 @ 60 MHz (60000000/16/19200=0x00C3)
38       U0LCR = 0x03;                    //no parity, 8 data + 1 stop
39       U0FCR = 0x07;                    //1 char trigger, enable and reset Rx & Tx FIFO
40
41  //count records in EEPROM0
42       ee_count(0,command_ee,response_ee);
43       status = response_ee[0];
44       records0 = response_ee[1];
45
46       //if the Flash is blank, initialize it
47       if (records0 == 0){
48       //copy initial data into EEPROM0
49            command_ee=(unsigned int) (&ee0_ini);
50            ee_write(0,command_ee,response_ee);
```

```
51              status = response_ee[0];
52         }
53
54         //read the last Flash entry
55         ee_read(0,command_ee,response_ee);
56         status = response_ee[0];
57         ee0_pnt = (struct ee0_data *) response_ee[1];
58         ee0_read._id         = (*ee0_pnt)._id;
59         ee0_read._ee_id      = (*ee0_pnt)._ee_id;
60         ee0_read._count      = (*ee0_pnt)._count;
61         ee0_read._SEC        = (*ee0_pnt)._SEC;
62         ee0_read._MIN        = (*ee0_pnt)._MIN;
63         ee0_read._HOUR       = (*ee0_pnt)._HOUR;
64
65         //initialize the RTC
66        CCR   = 0x00;
67        CCR   = 0x02;
68        CCR   = 0x00;
69        CCR   = 0x10;
70        ILR   = 0x03;          // Clear the Interrupt Location Register
71        CIIR  = 0x01;          // Increment of seconds generates an interrupt
72        AMR   = 0xFF;          // Alarm interrupts are not allowed
73          SEC   = ee0_read._SEC;
74          MIN   = ee0_read._MIN;
75          HOUR  = ee0_read._HOUR;
76        CCR   = 0x11;
77
78         //set the time and counter
79         loop_cnt = ee0_read._count;
80
81
82         printf("\n\n\n");
83
84         while(1){
85              if((ILR&0x01)==0x01){
86                   ILR = 0x01;
87                   loop_cnt=(loop_cnt+1)&0x3FF;
88                   _display_time(loop_cnt);
89                   putchar(0x0D);
90              }
91              if((IOPIN0&0x00004000)==0x00000000){
92                   ee0_temp._id          = EE_REC_ID; //user's code MUST provide valid record ID!
93                   ee0_temp._ee_id   = 0;
94                   ee0_temp._count       = loop_cnt;
95                   ee0_temp._SEC         = SEC;
96                   ee0_temp._MIN         = MIN;
97                   ee0_temp._HOUR        = HOUR;
98                   status = response_ee[0];
99                   command_ee=(unsigned int) (&ee0_temp);
100                  ee_write(0,command_ee,response_ee);                  //write data in EEPROM0
101                  status = response_ee[0];
```

```
102                 if (status==NO_SPACE_IN_EEPROM){
103                       ee_erase(0,command_ee,response_ee);              //erase EEPROM0
104                       status = response_ee[0];                    //reading of status
105                       command_ee=(unsigned int) (&ee0_temp);
106                       ee_write(0,command_ee,response_ee);              //write data in EEPROM0
107                       status = response_ee[0];
108                 }
109                 printf("\n\nData saved. Power-down mode entered...\n");
110                 while((U0LSR&0x60)!=0x60);
111                 PCON = 0x02;
112                 while(1);
113             }
114       };
115  }
116
117  void _display_time(short int count){
118       unsigned long int time_capture,local_hour,local_min,local_sec;
119
120       time_capture = CTIME0;
121       local_hour = (time_capture>>16) & 0x0000001F;
122       local_min  = (time_capture>> 8) & 0x0000003F;
123       local_sec  = time_capture & 0x0000003F;
124
125       printf("count=%4u  time=",count);
126
127       //display hour(s)
128       if(local_hour<10)
129             printf("%1u%1u:",0,local_hour);
130       else
131             printf("%2u:",local_hour);
132       //display minute(s)
133       if(local_min<10)
134             printf("%1u%1u:",0,local_min);
135       else
136             printf("%2u:",local_min);
137       //display second(s)
138       if(local_sec<10)
139             printf("%1u%1u",0,local_sec);
140       else
141             printf("%2u",local_sec);
142  }
```

11. Appendix C

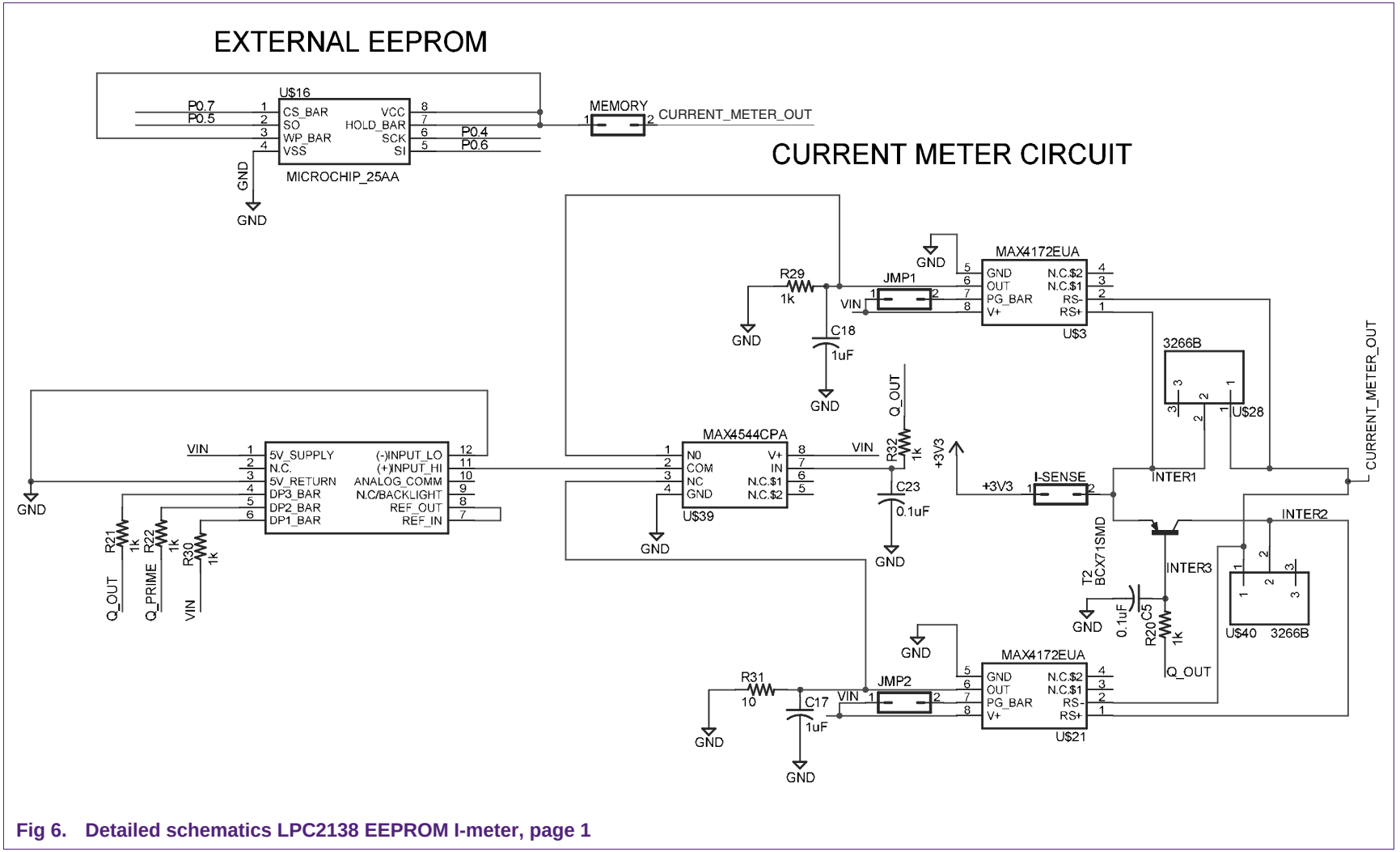

Fig 6. Detailed schematics LPC2138 EEPROM I-meter, page 1

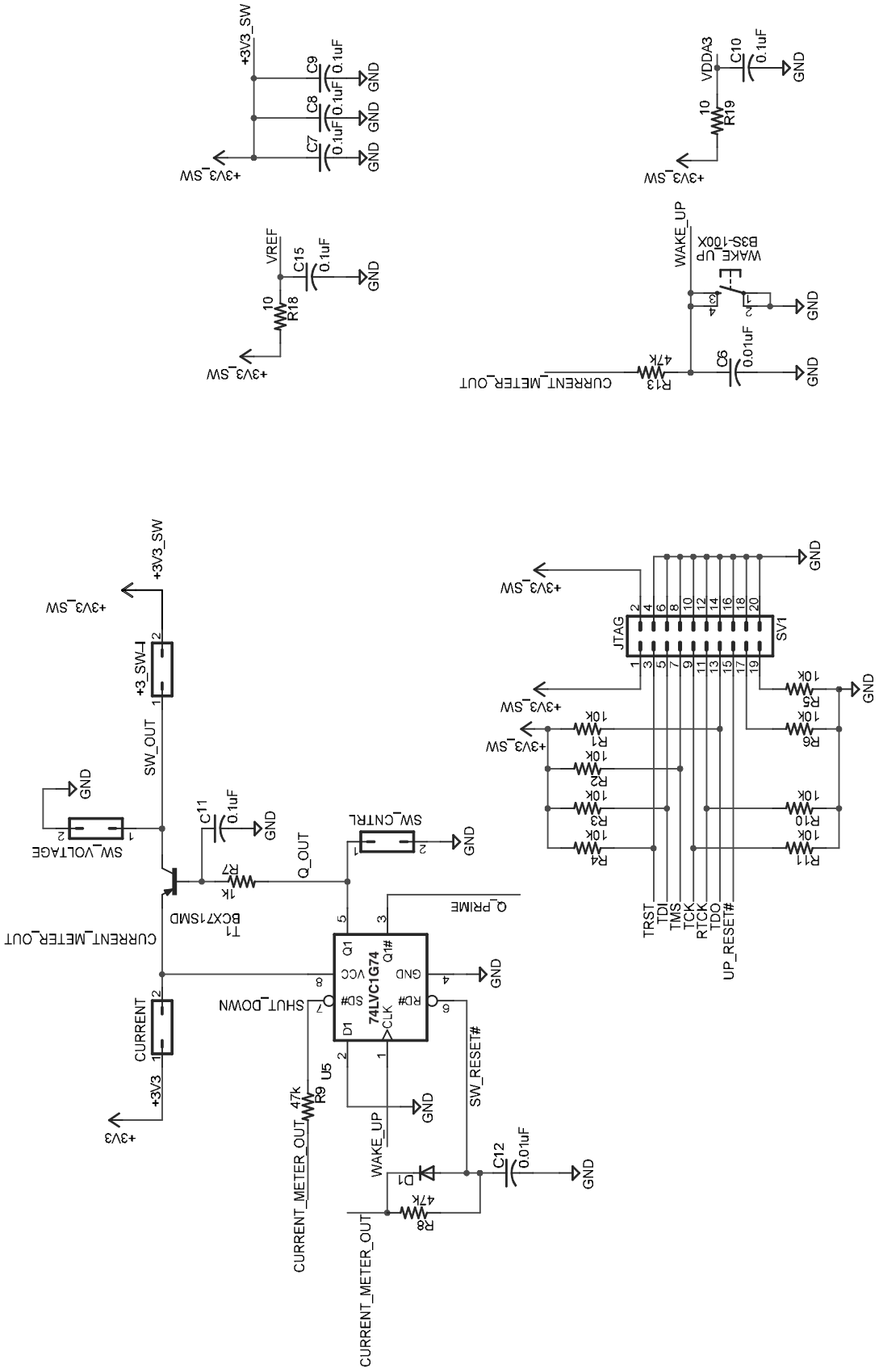

Fig 7. Detailed schematics LPC2138 EEPROM I-meter, page 2

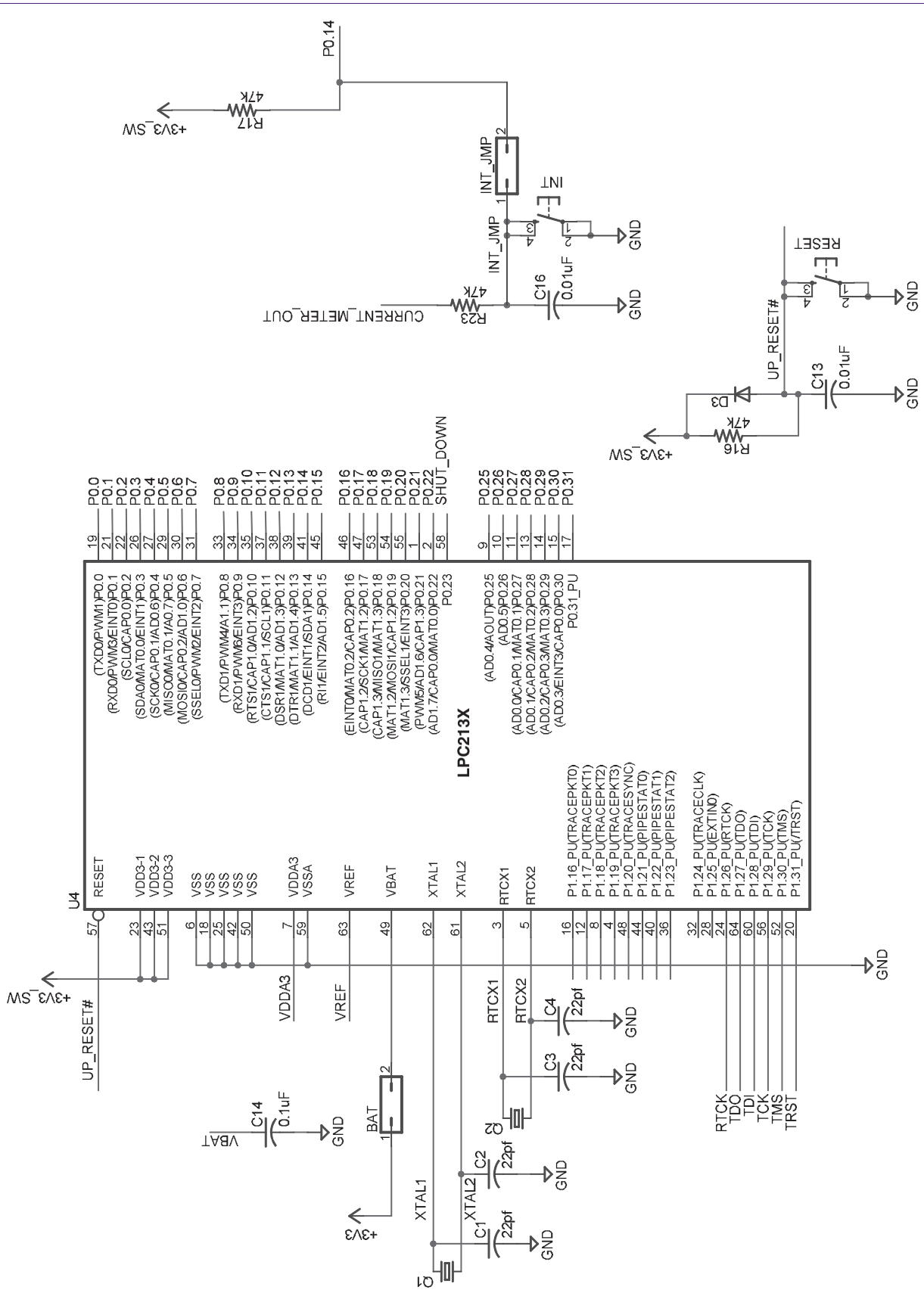

Fig 8. Detailed schematics LPC2138 EEPROM I-meter, page 3

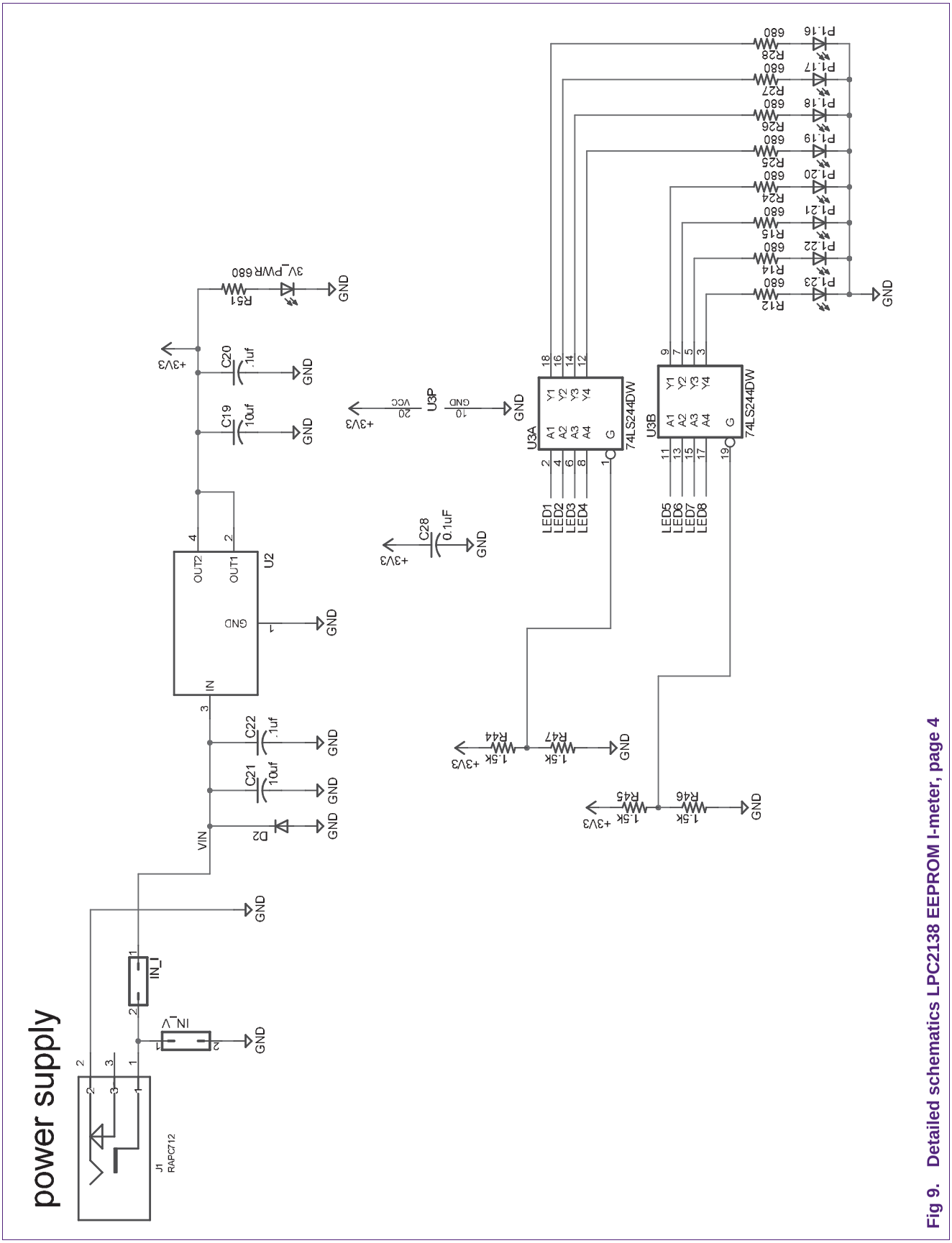

Fig 9. Detailed schematics LPC2138 EEPROM I-meter, page 4

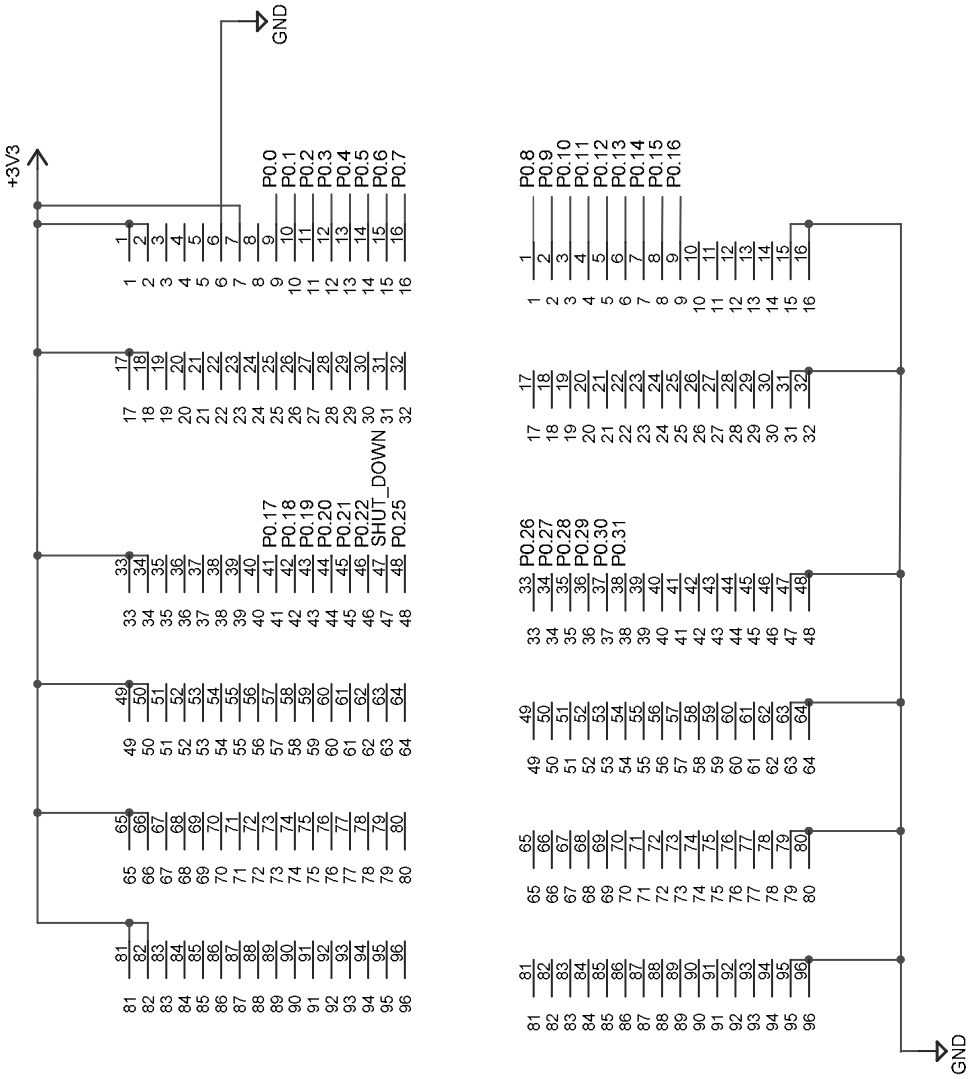

Fig 10. Detailed schematics LPC2138 EEPROM I-meter, page 5

12. References

[1] UM10120 (LPC2138 User manual)

[2] EEPROM emulation application note

[3] BCX71 datasheet

[4] 25AA160A datasheet

[5] Kemet Ceramic capacitors datasheet

[6] ERJ Precision Thick Film Chip Resistors datasheet

[7] 74LVC1G74 datasheet

13. Legal information

13.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

13.2 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.

13.3 Trademarks

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

14. Contents

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

Date of release: 6 December 2006
Document identifier: AN10535_1