Freescale Semiconductor
Application Note

Document Number: AN2681
Rev. 1, 10/2005

1-Wire Interface on the i.MX21 Applications Processor

MC9328MX21

by: Anish Trivedi

Contents

1 Abstract

The 1-Wire® Interface on the i.MX21 Applications Processor is an on-chip peripheral device that establishes bi-directional communication with one or more external 1-Wire devices. This application note illustrates the steps required to establish communication between the on-chip 1-Wire Interface and an external MAXIM/Dallas Semiconductor Multichemistry Battery Fuel Gauge device, DS2751.

The i.MX21 General Purpose I/O (GPIO) port E can be configured for use as a 1-Wire port. A 1-Wire system consists of an I/O data pin that can be driven to logic high, driven to logic low, or can act as an input. There is also an associated ground pin. The system requires one bus master, and can support multiple slave devices. There is only one slave device (the DS2751) for the scenario considered in this application note, with the i.MX21 acting as the bus master.

1.1 1-Wire Battery Fuel Gauge Circuit

The fuel gauge circuit is shown in Figure 1. It contains sensors for battery voltage, current flowing through the 25 mΩ sense resistor, and die temperature. These values are stored in internal registers that are updated every few ms. Registers are read through the data pin on the DS2751 that is connected to the 1-Wire port on the i.MX21.

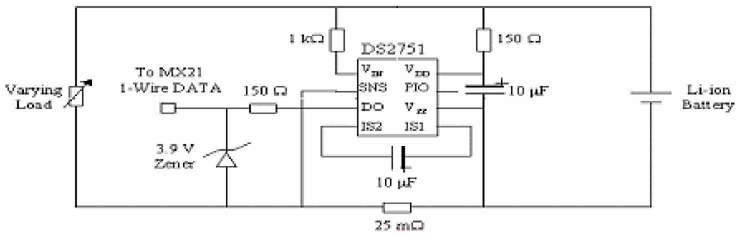

Figure 1. DS2751 Fuel Gauge Circuit

1.2 Communication Protocol

To communicate with the DS2751, the 1-Wire Interface must follow a protocol. The flowchart in Figure 2, is reproduced from the DS2751 specification, illustrates the transaction flow required between the i.MX21 processor and the DS2751.

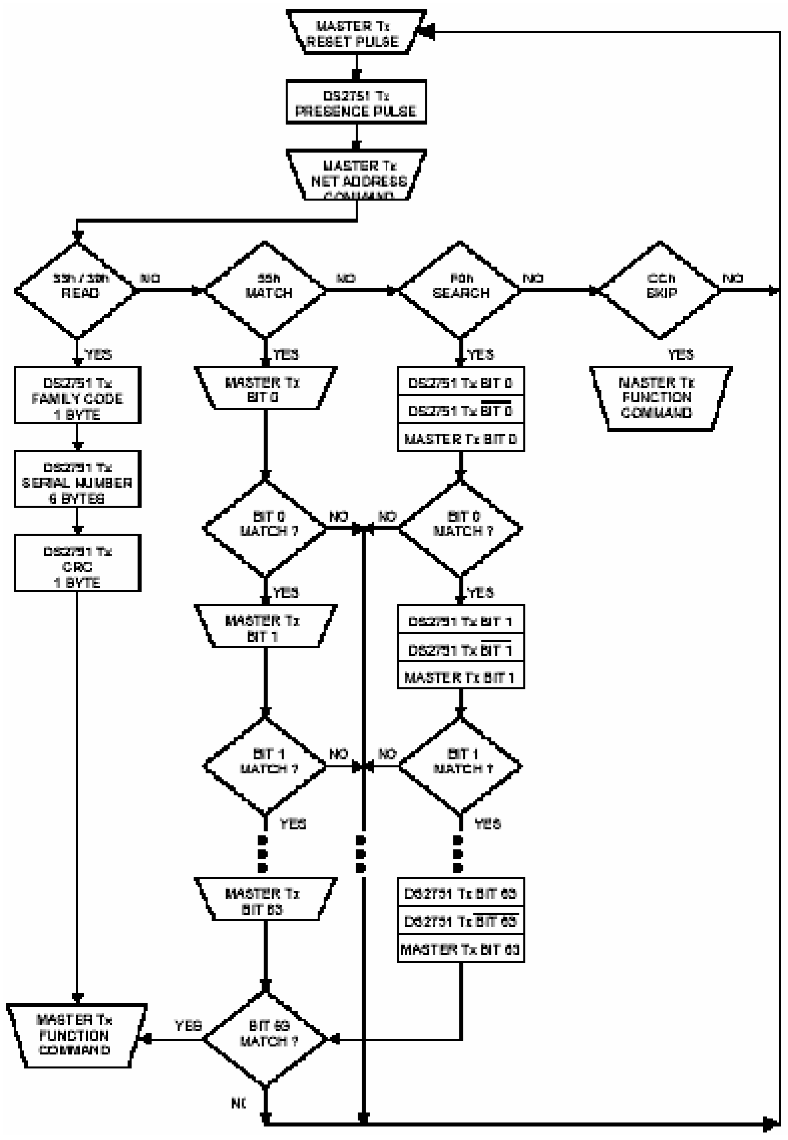

Figure 2. DS2751 1-Wire Communication

When idle, the 1-Wire bus is high until the i.MX21 drives it low during a reset pulse. The bus is held low for a specified time interval, subsequently the slave devices respond with a presence pulse. When the presence of slave devices has been detected, the i.MX21 proceeds to the addressing portion of the sequence.

As shown in Figure 2, there are four valid Net Address Commands:

1. *Read:* the DS2751 address, if the DS2751 is the only device connected. Otherwise, all devices try to transmit their address, which results in a data collision.
2. *Match:* an address that is transmitted bit-by-bit, while all the slaves listen. As soon as a mismatch occurs, the slave ignores the rest of the transmission until a reset pulse is seen.

3. *Search:* learn the addresses of all devices connected to the bus via the process of elimination. This scenario does not use this address command, so it will not be discussed in detail. See Chapter 5 of the *Book of DS19xx_iButton Standards* found at www.maxim-ic.com for a complete explanation of net address *search*.
4. *Skip:* addressing all together, as long as only the DS2751 is connected. This is the scheme employed for the testing documented in this paper. The DS2751 is ready to accept a function command after receiving the *skip* net address command.

The DS2751 is equipped with internal registers, a 32-byte EEPROM, and a 16-byte SRAM for battery statistics storage and battery capacity calculations. The function commands recognized by the DS2751 include *Read Data* from DS2751 registers and memory, *Write Data* to certain registers and memory, *Copy Data* from the SRAM to the EEPROM, *Recall Data* from the EEPROM to the SRAM, and *Lock* an EEPROM block containing the specified address.

For this scenario, a Lithium Ion battery is connected as shown in Figure 1. The rate at which the battery loses its charge depends on the load current, which was controlled using the potentiometer. The DS2751 records the voltage readings to its 16-bit voltage register. The i.MX21 is programmed to read the voltage readings from the DS2751 through the 1-Wire interface every few minutes while the battery is discharging. The following sections explain how to interface with the 1-Wire module on the i.MX21, and also provide the software for communication with the DS2751.

2 i.MX21 1-Wire Hardware Interface

GPIO port E on the i.MX21 processor can be configured as a 1-Wire bus. Timing requirements are met in hardware with the help of the clock (1 MHz) and the 1-Wire state machine. The registers identified in Table 1are available to control the 1-Wire communication.

Table 1. 1-Wire Module Register Memory Locations

Description	Name	Address
1-Wire Control Register	CONTROL	0x10009000
1-Wire Time Divide Register	TIME_DIVIDER	0x10009002
1-Wire Reset Register	RESET	0x10009004

2.1 Control Register

The 16-bit Control register is used to drive the communication with the 1-Wire external device.

Table 2. Control Register Description

Name	Description	Settings
Bits 15–8	Reserved bits	N/A
RPP Bit 7	**RESET PRESENCE PULSE—**This bit is self-clearing, and is cleared after the detection of the presence pulse from the 1-Wire interface.	0 = Do nothing / pulse complete. 1 = Generate Reset Pulse and sample for DS2502 presence pulse. This bit is self-clearing and will be cleared after the presence is detected.

Table 2. Control Register Description (continued)

Name	Description	Settings
PST Bit 6	**PRESENCE STATUS—**This bit is valid after the RPP bit is self-cleared.	0 = Device not present. 1 = Device present. This bit is valid after the RPP bit is self-cleared.
WR0 Bit 5	**WRITE 0—**This bit is self-clearing and will be cleared when the write of the bit is complete.	0 = Do nothing / Write sequence complete. 1 = Write a 0 bit to the interface. This bit is self-clearing and will be cleared when the write of the bit is complete.
WR1 Bit 4	**WRITE 1 / READ—**This bit is self-clearing and will be cleared when the write of the bit is complete. This also reads the bit because Write 1 and Read timings are identical. The value of the read bit is stored in RDST, and is valid after WR1/RD is self-cleared.	0 = Do nothing / Write sequence complete. 1 = Write a 1 bit to the interface. This bit is self-clearing and will be cleared when the write of the bit is complete. When used for a Read operation, the read bit is stored in RDST, and is valid after WR1/RD is self-cleared.
RDST Bit 3	**READ STATUS—**This bit is valid after the WR1/RD bit is self-cleared.	0 = A 0 was sampled during a read. 1 = A 1 was sampled during a read. This bit is valid after the WR1/RD bit is self-cleared.
Reserved Bits 2–0	Reserved Bits	N/A

The programmer only needs to set the bits as specified in the Control register when communicating with the 1-Wire device and then signal the self-clearing bits when the transaction is complete.

2.2 TIME_DIVIDER Register

The TIME_DIVIDER register divides the peripheral clock, ipg_clock, to generate the internal clock to the 1-Wire module. The value in the register must be such that IPG_CLOCK / (TIME_DIVIDER+1) ˜ 1 MHz

Table 3. TIME_DIVIDER Register Description

Name	Description	Settings
Bits 15–8	Reserved bits	N/A
dvdr Bits 7–0	**Predivider Factor—**This field is used to set the clock divider setting to control the frequency of the generated clock.	0 = Divider value is 1 (default). 1 = Divider value is 2. … FF = Divider value is 256.

2.3 RESET Register

The RESET register resets the 1-Wire state machine. Resetting the state machine aborts any transaction that is currently taking place, and reverts the bus to logic high.

Table 4. RESET Register Description

Name	Description	Settings
Reserved Bits 15–1	Reserved bits	N/A
RST Bit 0	**Software Reset—**The reset register is used to reset the module through software.	0 = 1-Wire is not reset. 1 = 1-Wire is reset.

3 Configuration

The Table 5 settings must be configured to ensure proper operation for the 1-Wire interface.

Table 5. Configuration Settings

Register Name	Description	Settings
MPCTL0 MPCTL1	Set MCU & System PLL (MPLL) value. Set BRMO, which affects jitter performance of the MPLL.	See Chapter 7 of the *MC9328MX21 Applications Processor Reference Manual* for information on these registers.

CSCR	Scale the MPLL by a factor between 1 and 4 to set FCLK. Configure HCLK by setting BCLKDIV, which divides FCLK to generate HCLK. Configure IPDIV, the HCLK divider that generates IPG_CLOCK, the clock to the 1-Wire interface.
	HCLK = FCLK / (BCLKDIV+1)
	IPG_CLOCK = HCLK / (IPDIV+1)
		Bits [15-14]: **PRESC** is the 2-bit scaling factor to generate FCLK. 00 = divider is 1, … 11 = divider is 4.
		Bits [13-10]: **BCLKDIV** is the divider that generates HCLK. Must equal ‘0010’ (divider is 3). (See note below)
		Bit 9: **IPDIV** is the divider that generates the clock to the 1-Wire™. Must equal ‘1’ (divider is 2). (See note below)

CSCR MPCTL1	Restart the MPLL for the settings to take effect. The read-only Lock Flag (LF) in the MPCTL1 register indicates whether the MPLL output is valid. This bit must be high before using the 1-Wire interface.	Bit 21 of CSCR: Set **MPLL_RESTART** to restart the MPLL at the new frequency. Bit 15 of MPCTL1: **LF** will be set when the restart has completed.
PCCR0	Enable GPIO.	Bit 11: **GPIO_EN** must be set.
PTE_GIUS GPR_E	Configure GPIO port E for 1-Wire use.	Clear Bit 16 of PTE_GIUS, and set Bit 16 of GRP_E.
AIPI_PSR0 AIPI_PSR1	Configure the AIPI for 16-bit communication, since the 1-Wire registers are 16-bit wide.	Set AIPI_PSR0 Bit 9, and clear AIPI_PSR1 bit 9.
PCCR1	Enable the ipg_clock to the 1-Wire module.	Set Bit 31.
TIME_DIVIDER	Set the value such that **IPG_CLOCK / (TIME_DIVIDER+1) = 1 MHz**. A 1-Wire clock as close to 1 MHz as possible is required for proper operation.	Bits[7-0]: 0 = Divider is 1 1 = Divider is 2 … FF = Divider is 256.

NOTE

The errata document for the i.MX21 reference manual, *MC9328MX21 Chip Errata* (order number MC9328MX21CE), notes that the BCLKDIV parameter must equal 0010 so that HCLK = FCLK / (BCLKDIV+1) = FCLK / 3, and that IPDIV parameter must equal 1, so that IPG_CLOCK = HCLK / (IPDIV+1) = HCLK / 2.

4 Program Source Code

Example 1 code configures the i.MX21 applications processor 1-Wire interface, and reads the value from the DS2751 voltage register every five minutes. The *Battery_Gauge_1-Wire_Test.mcp* file contains all the supporting software necessary to run this code on an i.MX21 ADS or EVB. The file was created in the Metrowerks CodeWarrior IDE.

4.1 Configuration Code

The *SysInit()* function, found in the file *SysInit.c*, performs the necessary configuration of the PLLs and GPIO port E as noted in section 3. It initializes the MPLL to 266 MHz, with a PRESC divide factor of 1, and an IPDIV divide factor of 2, resulting in a 44.3 MHz IPG_CLOCK.

Example 1. SysInit()

```
void SysInit(void)
{
         // initialize PLL and clocks here
         // Set up the MPLL for 266.0000537MHz
         // PD = 0; MFI = 7; MFN = 115; MFD = 123
         *(p_uint32_t)CRM_MPCTL0 = 0x007B1C73;
         *(p_uint32_t)CRM_MPCTL1 = 0x00000040;
    // set BRMO since 1/10 < MFN/MFD+1 < 9/10

         // Set up the SPLL for 287.9999978Mhz operation
         *(p_uint32_t)CRM_SPCTL0 = 0x03B02227;

         // now configure the CSCR register
         // clear all bits except for PRESC
         *(p_uint32_t)CRM_CSCR &= 0x0000C000;

         // Now set USBDIV=5; SD_CNT=3; BCLKDIV=2; IPDIV=1; HCLK will be 88.6MHz
         // IPG_CLOCK = HCLK / (IPDIV+1) => IPG_CLOCK will be 44.3 MHz
         *(p_uint32_t)CRM_CSCR |= 0x17000A07;

         // Last step, clear PRESC to 0
         *(p_uint32_t)CRM_CSCR &= 0xFFFF3FFF;

         // Now, restart the PLLs
         *(p_uint32_t)CRM_CSCR |= 0x00600000;

         // Wait for lock flag to set
         while((*(p_uint32_t)CRM_MPCTL1 & 0x00008000) != 0x00008000);
         while((*(p_uint32_t)CRM_SPCTL1 & 0x00008000) != 0x00008000);

         // enable the following in the PCCR0
         // HCLK_DMA, LCDC, LCDC_PIXCLK, DMA_EN, GPIO_EN
         *(p_uint32_t)CRM_PCCR0 |= 0x44042800;

         /* Enable 1-Wire Bus */

         // Configure GPIO pin for 1-Wire use
         *(p_uint32_t)GPIOE_GIUS &= 0xFFFEFFFF;
         *(p_uint32_t)GPIOE_GPR |= 0x00010000;
         // enable the ipg_clock to 1-Wire interface
```

```
        *(p_uint32_t)CRM_PCCR1 |= 0x80000000;
        // Configure AIPI for 16 bit 1-Wire communication
        *(p_uint32_t)AIPI1_PSR0 |= 0x00000200;
        *(p_uint32_t)AIPI1_PSR1 &= 0xFFFFFDFF;

}
```

The PRESC bits are not touched during the initial clear of the CSCR register, but are later cleared after the remaining PLL settings have been set. This is the convention followed for setting the PRESC bits for the i.MX21.

4.2 1-Wire Code

The Main.c file is listed Example 2. The main function sets up the 1-Wire clock to 1 MHz, then sets the stopwatch timer to 5 minutes. Subsequently an infinite *while* loop is entered. After the 5-minute timer is up, the i.MX21 initializes communication (using a reset pulse) to detect any 1-Wire devices. If a presence pulse is not detected, then the program exits. The program will also exit when the battery voltage goes below the threshold voltage for the DS2751, approximately 2.5 V, which causes the DS2751 to shut off. After the initialization, a *read* command is sent with the address for the voltage register. This is followed by a *receive* function, which listens for data from the DS2751 and then assimilates it properly into a register. The Multi-ICE® tool was used to debug over the JTAG interface on the i.MX21 with the ARM® eXtended Debugger (AXD) tool.

Example 2. Main.c

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

 C   M O D U L E   F I L E

 (c) Copyright Freescale Semiconductor Inc. 2000-2003
 ALL RIGHTS RESERVED

*******************************************************************************

 Project Name                       : Maxim DS2751 Battery Fuel Gauge 1-Wire Verification
 Project No.                        :
 Title                              :
 File Name                          : main.c
 Last Modified                      : 10/20/2003
 (MM/DD/YYYY)

Description                :                                 The main function for the 1-Wire
device, DS2751, functional    verification. This test will communicate with the DS2751 over the
1-Wire data bus using the direct addressing mode, assuming that only one slave is present on
the bus. The program will read back voltage values recorded by the
                                     DS2751 to ensure proper functionality.

 Author:                             Anish Trivedi

 History (MM/DD/YYYY):
 10/20/2003 - Initial Proposal

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

```
#include <stdio.h>
#include "common.h"
#include "tht_memory_map_defines.h"
#include "testcase.h"

/* Modify SysInit() for different system initialization settings */
extern int SysInit(void);
extern int MemInit(void);

/*****************************************************************
# D E F I N E S
*****************************************************************/


/******************************************************************
 Public Functions
******************************************************************/

int init_1wire(void);
void write1(void);
void write0(void);
void read(uint8_t addr);
uint16_t receive(void);
float volts_decode(uint16_t volts);
float temp_decode(uint16_t temp);
float acc_curr_decode(int16_t acc_curr);

int32_t main(void)
{

        uint16_t voltage, temperature, acc_current;

        SysInit();
        MemInit();

        // enable the stopwatch interrupt
        *(p_uint32_t)(RTC_RTCIENR)  = 0x00000001;

        // enable the RTC in the PCCR1
        *(p_uint32_t)(CRM_PCCR1)  |= 0x20000000;

        // set the time dividor=44-1 to produce a ~1 MHz clock
        *(p_uint16_t)OWIRE_TIME_DIV = 0x002B;

        // set the stopwatch to 5 mins
        *(p_uint32_t) RTC_STPWCH = 4;

        while (1) {

                // wait for stopwatch interrupt
                while ((*(p_uint32_t)(RTC_RTCISR) & 0x00000001) != 0x00000001);

                // initialize communication with 1-wire device
                if (init_1wire() == 0 )
                        return -1;

                // read from voltage register
```

```
                    read(0x0C);

                    // Receive voltage register MSB & LSB from DS2751
                    voltage = receive();

                    printf("%f\n", volts_decode(voltage));

                    // clear the stopwatch interrupt
                    *(p_uint32_t)(RTC_RTCISR) |= 0x00000001;

                    // set the stopwatch to 5 mins again
                    *(p_uint32_t) RTC_STPWCH = 4;
          }
}


int init_1wire(void) {

          // Send a reset pulse
          *(p_uint16_t)OWIRE_CTRL = 0x0080;

          // wait for RPP bit to clear
          while ( (*(p_uint16_t)OWIRE_CTRL & 0x0080) != 0);

          if ( (*(p_uint16_t)OWIRE_CTRL & 0x0040) == 0 ) {
                    printf("One-Wire Device not present.\n");
                    return 0;
          }

          // Send "Skip Net Address" Command (0xCC = 11001100)
          write0(); write0(); write1(); write1();
          write0(); write0(); write1(); write1();

          return 1;

}

void write1 (void) {

          // write a 1 to the 1-wire data pin
          *(p_uint16_t)OWIRE_CTRL |= 0x0010;
          // wait until the transaction is complete
          while ( (*(p_uint16_t)OWIRE_CTRL & 0x0010) != 0);
}

void write0 (void) {

          // write a 0 to the 1-wire data pin
          *(p_uint16_t)OWIRE_CTRL |= 0x0020;
          // wait until the transaction is complete
          while ( (*(p_uint16_t)OWIRE_CTRL & 0x0020) != 0);
}

void read (uint8_t addr) {

          // Send the "Read" command (0x69 = 01101001)
          write1(); write0(); write0(); write1();
```

```
        write0(); write1(); write1(); write0();

        // Send the address to read from

        (addr & 0x01) ? write1() : write0();
        (addr & 0x02) ? write1() : write0();
        (addr & 0x04) ? write1() : write0();
        (addr & 0x08) ? write1() : write0();
        (addr & 0x10) ? write1() : write0();
        (addr & 0x20) ? write1() : write0();
        (addr & 0x40) ? write1() : write0();
        (addr & 0x80) ? write1() : write0();

}

uint16_t receive(void) {

        int i;
        uint16_t in_bit;
        uint16_t x = 0;


        // read the 16 bit value, 1 bit at a time
        i = 0;
        while (i < 16) {

                // Read timing is same as Write 1
                write1();
                in_bit = *(p_uint16_t)OWIRE_CTRL & 0x0008;

                // MSB
                if (i < 8)
                        in_bit = in_bit << (5+i);
                // LSB
                else if (i > 7 && i < 12)
                        in_bit = in_bit >> (i-12);
                else
                        in_bit = in_bit << (i-11);

                x |= in_bit;

                i++;
        }

        return x;
}


float volts_decode(uint16_t volts) {

        int coded_volts;
        float decoded_volts;

        coded_volts = volts >> 5;

        // check sign bit, front fill with 1's if negative
        if (volts & 0x8000)
```

```
                coded_volts |= 0xFFFFF800;

        // units are 4.88 mV -> convert to volts
        decoded_volts = coded_volts * 4.88 / 1000;

        return decoded_volts;

}

float temp_decode(uint16_t temperature) {

        int coded_temp;
        float decoded_temp;

        coded_temp = temperature >> 5;

        // check sign bit, front fill with 1's if negative
        if (temperature & 0x8000)
                coded_temp |= 0xFFFFF800;

        // units are .125 deg C -> convert to deg C
        decoded_temp = coded_temp * .125;

        return decoded_temp;

}

float acc_curr_decode(int16_t acc_curr) {

        int coded_acc_curr;
        float decoded_curr;

        coded_acc_curr = acc_curr >> 3;

        if (acc_curr & 0x8000)
                coded_acc_curr |= 0xFFFFE000;

        // units are 6.25 uV, convert to mA given 25 mohm resistor
        decoded_curr = coded_acc_curr * 6.25 / 1000 / 25;

        return decoded_curr;

}
/************************************************************/
/**************** END OF FILE *****************************/
```

4.2.1 Reusability of 1-Wire Code

Communication with any 1-Wire device requires sending an initial reset pulse, subsequently, all devices on the 1-Wire bus respond with a presence pulse. After the detection of a presence pulse, the net address command is sent by the i.MX21. The net address command may be one of the four enumerated commands in Section 1.2, on page 3. All 1-Wire devices must follow this handshaking procedure to ensure proper communication with the bus master. Therefore, the device detection and addressing portions of the code, along with the i.MX21 setup and control code, can be reused for other 1-Wire devices.

After the handshaking procedure has successfully completed, and communication between the i.MX21 processor and the slave device has been established, the i.MX21 issues a function command (such as *read* the voltage register on the DS2751). This command can be unique to each 1-Wire device. Code as shown in Example 2 (*main.c*) must be modified accordingly.

5 Experimental Battery Discharge Data

Using the code presented and the scenario described here, battery voltage readings were taken every few minutes from the battery gauge over the 1-Wire interface with the load current held at a constant value. Data for two values of the current, 500 mA and 250 mA, are presented in Table 6.

5.1 500 mA Current

The battery voltage was read every 15 minutes, and the potentiometer was set to such a value as to draw a load current of 500 mA. Table 6 lists the amount of time elapsed since the battery was connected to the circuit, and the battery voltage was read from the DS2751.

Table 6. Battery Voltage During Discharge with 500 mA Current

Elapsed Time (Minutes)	Battery Voltage (V)
0	4.19
15	4.01
30	3.95
45	3.90
60	3.84
75	3.80
90	3.75
105	3.66
120	3.54
135	3.38
150	3.23
165	3.00
180	2.80
195	2.58
200	0

No voltage is seen across the battery terminals after the voltage drops below 2.6 V, because of the protection circuit built-in to the battery. At the same point, the DS2751 also shuts down, since its threshold voltage for operation is 2.5 V.

5.2 250 mA Current

To obtain a different discharging curve, the load current was held at 250 mA. Using the stopwatch timer in the i.MX21, the battery voltage was read from the DS2751 every 5 minutes. Table 7 lists the voltage values observed over time.

Table 7. Battery Voltage During Discharge with 250 mA Current

Elapsed Time (Minutes)	Battery Voltage (V)
0	4.19
5	4.09
10	4.08
15	4.07
20	4.06
25	4.04
30	4.04
35	4.02
40	4.01
45	4.00
50	3.99
55	3.99
60	3.98
65	3.97
70	3.96
75	3.95
80	3.95
85	3.94
90	3.93
95	3.92
100	3.91
105	3.90
110	3.90
115	3.89
120	3.88
125	3.87
130	3.87
135	3.86
140	3.86
145	3.85
150	3.84
155	3.83

Table 7. Battery Voltage During Discharge with 250 mA Current (continued)

Elapsed Time (Minutes)	Battery Voltage (V)
160	3.82
165	3.81
170	3.80
175	3.79
180	3.77
185	3.76
190	3.74
195	3.72
200	3.70
205	3.68
210	3.66
215	3.65
220	3.63
225	3.61
230	3.58
235	3.56
240	3.54
245	3.51
250	3.49
255	3.46
260	3.44
265	3.42
270	3.39
275	3.36
280	3.34
285	3.31
290	3.28
295	3.25
300	3.23
305	3.20
310	3.17
315	3.14
320	3.11
325	3.08
330	3.05
335	3.02
340	2.99

Table 7. Battery Voltage During Discharge with 250 mA Current (continued)

Elapsed Time (Minutes)	Battery Voltage (V)
345	2.97
350	2.94
355	2.91
360	2.88
365	2.85
370	2.83
375	2.80
380	2.77
385	2.74
390	2.71
395	2.67
400	2.63
405	2.55
410	2.42
415	0

The battery lasts approximately twice as long when the load current is divided by half.

5.3 Estimating Battery Capacity

After the characterization of the battery is complete, the remaining capacity can be calculated using a few equations. Knowing the $FULL_I$ and $EMPTY_I$ values of the battery voltage for a given load current I, and given the most recent voltage reading V, the remaining battery capacity can be estimated as:

$$\text{Capacity} = [(V - EMPTY_I) / (FULL_I - EMPTY_I)] \times 100\%$$

The estimated battery capacity for both the 500 mA and 250 mA load currents over the duration of the discharge are presented in the Figure 3 plot. The FULL value for both cases is 4.19V, while the $EMPTY_{500}$ = 2.42V and $EMPTY_{250}$ = 2.58V.

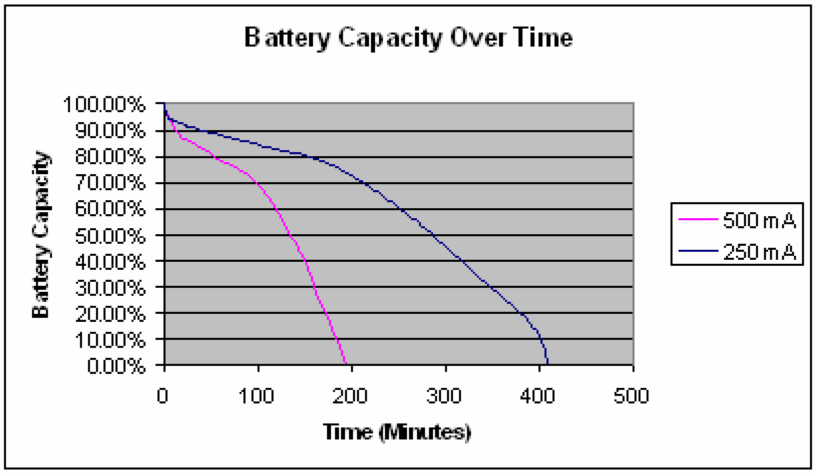

Figure 3. Battery Capacity Calculation

6 Summary

The 1-Wire Interface on the i.MX21 applications processor requires configuration of the MPLL, GPIO, and AIPI registers before any 1-Wire hardware registers can be accessed. Timing requirements are crucial for proper operation, and the 1-Wire state machine and the internal clock provide the necessary signals. The clock must be configured to approximately 1 MHz. The user can then set the 1-Wire Control register to send and receive bits over the 1-Wire bus.[1]

7 Referenced Documents

1. *DS2751 Multichemistry Battery Fuel Gauge Data Sheet*, Maxim/Dallas Semiconductor
2. *Lithium-Ion Cell Fuel Gauging with Dallas Semiconductor Battery Monitor ICs,* Maxim/Dallas Semiconductor (order number App Note 131)
3. *MC9328MX21 Chip Errata,* (order number MC9328MX21CE)
4. *MC9328MX21 Applications Processor Reference Manual,* (order number MC9328MX21RM)
5. *Software for i.MX21 Applications Processor 1-Wire Interface* (order number AN2681SW)

1.

1-Wire® is a registered trademark of Maxim /Dallas Semiconductor.

Document Number: AN2681
Rev. 1
10/2005