XC800 Family

AP08119
Data Logger Using XC82x/XC83x Emulated EEPROM ROM Library

Application Note
V1.0 2011-02

Microcontrollers
LEGAL DISCLAIMER
THE INFORMATION GIVEN IN THIS APPLICATION NOTE IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

Information
For further information on technology, delivery terms and conditions and prices, please contact the nearest Infineon Technologies Office (www.infineon.com).

Warnings
Due to technical requirements, components may contain dangerous substances. For information on the types in question, please contact the nearest Infineon Technologies Office.
Infineon Technologies components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.
# Table of Contents

1. **Introduction** ..........................................................................................................................5
   1.1 XC82x Block Diagram ...........................................................................................................5
   1.2 XC83x Block Diagram ...........................................................................................................6

2. **Features of the XC82x/XC83x ADC Module** ........................................................................8

3. **Features of the Emulated EEPROM ROM Library** .............................................................9

4. **DAVE™ Configuration for the Data Logger Application** .....................................................10
   4.1 Analog-to-Digital Converter (ADC) Configuration .................................................................10
   4.1.1 Selecting the ADC of the XC822M ....................................................................................10
   4.1.2 Configuration of the ADC Module Clock screen ...............................................................11
   4.1.3 Configuration of the ADC General screen in Basic Mode ...............................................12
   4.1.4 Configuration of the ADC Channels screen in Basic Mode ..............................................13
   4.1.5 Configuration of the ADC Channel 0 General Settings screen, in Basic Mode ...............14
   4.1.6 Configuration of the ADC RequestSources screen in Basic Mode .................................15
   4.1.7 Configuration of the ADC Result Register screen in Basic Mode ....................................16
   4.1.8 Configuration of the ADC Interrupts screen in Basic Mode ............................................17
   4.1.9 Configuration of the ADC Functions screens in Basic Mode ..........................................18
   4.2 Configuration of Timer T2 .......................................................................................................21
   4.2.1 Configuration of the T2 Module Clock screen .................................................................21
   4.2.2 Configuration of the T2 Timer 2 screen ............................................................................22
   4.2.3 Configuration of the T2 T2EX screen .............................................................................23
   4.2.4 Configuration of the T2 Interrupts screen .......................................................................24
   4.2.5 Configuration of the T2 Functions screen .......................................................................25
   4.3 Configuration of emulated EEPROM ROM Library .............................................................26
   4.3.1 Configuration of the EEPROM Functions screen ............................................................26
   4.4 Configuration of UART ..........................................................................................................27
   4.4.1 Configuration of the UART screen ..................................................................................27
   4.4.2 Configuration of the UART BRG screen .........................................................................28
   4.4.3 Configuration of the UART Functions screen .................................................................29
   4.5 Example Code for Data Logger Application ........................................................................30

5. **Data Logger Application Results** .......................................................................................47

6. **Conclusion, Related Documents and Links** .......................................................................53
   6.1 Conclusion .............................................................................................................................53
   6.2 Related Documents and Links ............................................................................................53
1 Introduction

This application note explains the practical application of a simple data logger, using the emulated EEPROM ROM library supported by the XC82x/XC83x 8-bit Microcontrollers with the Infineon Digital Application Virtual Engineer (DAVE™) tool.

The data logger in this example is able to store 124 bytes of data using the emulated EEPROM. The analog voltage from a potentiometer (acting as a sensor) is applied to the on-chip Analog-to-Digital Converter (ADC) for conversion at periodic intervals. The intervals are timed with the on-chip timer ‘T2’. The converted data is then logged into the emulated EEPROM. The EEPROM is emulated using the on-chip flash memory. The logged data is displayed on a HyperTerminal using the on-chip Universal Asynchronous Receiver/Transmitter (UART).

Note: For more detailed information about the EEPROM emulation, please refer to the XC82x User Manual (XC82x_um_v1.1.pdf) and refer to section 24.3 “EEPROM Emulation ROM Library”.

On starting, the application allows the user to either read the existing data in the emulated EEPROM or to store the analog-to-digital converted data in the emulated EEPROM, depending on the input command. The user can press any character key to begin using the application example.

When the user chooses to store the converted data, conversion occurs at a periodic interval of 1 second. The converted data is displayed by the Windows HyperTerminal application via the on-chip UART interface, and is then stored in the emulated EEPROM.

Note: In this application the potentiometer is used to give different analog voltage inputs for analog-to-digital conversion, but this could be used to record environmental parameters such as temperature, relative humidity, wind speed and direction, light intensity, water level and water quality over time, for example.

1.1 XC82x Block Diagram

![XC82x DAVE™ Block Diagram](image-url)
1.2 XC83x Block Diagram
Figure 4  XC83xM-2F Block Diagram
2 Features of the XC82x/XC83x ADC Module

The Analog-to-Digital Converter (ADC) module of the XC82x/XC83x microcontroller’s, uses the successive approximation method to convert analog input values (voltages) to discrete digital values.

Features

- Successive approximation
- 8-bit or 10-bit resolution
- Up to eight (XC83x) or four (XC82x) analog channels and four independent result registers
- Programmable result data protection in case of slow CPU access (wait-for-read mode)
- Single or multiple conversion modes
- Auto scan functionality
- Limit checking for conversion results
- Data reduction filter (accumulation of up to 2 conversion results)
- Two independent conversion request sources with programmable priority
- Selectable conversion request trigger. Software or Hardware (e.g. Timer Events, External Events) Triggers
- Flexible interrupt generation with fixed service nodes
- Cancel/restart feature for interrupted conversions
- Low power modes

The functionality of the XC82x and XC83x ADC module is same as the ADC module of other members of the XC800 family, except for the following additional features:

- Out of range voltage comparator (ORC) detection for each input channel that is able to trigger other modules
- First order digital low pass filter of the conversion results
- Three internal reference voltage sources selectable for each channel
- Configurable limit checker boundary flags that can trigger other modules

For more detailed information on the functionality of the ADC module, please refer to the application note:

XC82x XC83x Analog-to-Digital Converter Basic and Advanced Mode.pdf
3 Features of the Emulated EEPROM ROM Library

EEPROM Emulation increases the maximum flash cycling count by using more flash cells than the emulated dataset bytes.

The EEPROM Emulation ROM Library provides the following features:

- Function to initialize, check, read and write to emulated EEPROM
- Provide emulation mode for multiples of 31 bytes data set using 4 D-Flash sectors
- Support EEPROM emulation size of 31, 62, 93 or 124 bytes
- Support polling (blocking) based flash operation
- Support for KEIL C51 tool chain (small memory model, big endian, register calling convention)
  - limited support for other compilers using small memory model only
- User application to access emulated EEPROM using the Application Program Interface (API) provided in DAVE™

The EEPROM emulation APIs supported in DAVE™ are listed below.

The data structure EEPROMInfo is used to manage EEPROM emulation operation:

```c
typedef struct EEPROMInfo {
    unsigned int ActiveSector;
    unsigned int WriteAddress;
    unsigned char DataSize;
} idata EEPROMInfo;
```

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsigned char InitEEPROM (unsigned char idata mode, EEPROMInfo *config)</td>
<td>Initialize EEPROM emulation and detect EEPROM error</td>
</tr>
<tr>
<td>Void FixEEPROM (ubyte idata sector_status, ubyte idata *buffer)</td>
<td>Fix invalid sectors based on status returned by InitEEPROM ()</td>
</tr>
<tr>
<td>Unsigned char WriteEEPROM (unsigned char idata address, char idata *src, EEPROMInfo *config)</td>
<td>Perform write operation to EEPROM</td>
</tr>
<tr>
<td>Unsigned char ReadEEPROM (unsigned char idata address, char idata *dst, EEPROMInfo *config)</td>
<td>Perform read operation to EEPROM</td>
</tr>
</tbody>
</table>

Note: For more detailed information about the EEPROM emulation, please refer to the XC82x User Manual (XC82x_um_v1.1.pdf) and refer to section 24.3 “EEPROM Emulation ROM Library”.

This application note is intended to show how to use the Data EEPROM (DEE) emulation library, reading from the emulated pages using the DEE APIs supported in DAVE™. An XC822 Starter Kit is used as hardware to demonstrate this usage.
4 DAVE™ Configuration for the Data Logger Application

4.1 Analog-to-Digital Converter (ADC) Configuration

For ease of use, the concept of Basic and Advanced mode has been introduced in the XC82x and XC83x ADC module:

- In Basic mode the basic settings are configured automatically by DAVE™ for typical use similar to a standard ADC.
- In Advanced mode all of the settings need to be configured by the user. This is the normal mode in other 8-bit controller ADC modules.

In this application example the Basic mode has been used for the ADC module.

Screen shots of the configurations for the different DAVE™ screens are shown on the following pages.

4.1.1 Selecting the ADC of the XC822M

![Figure 5 DAVE™ Bit Map Showing the ADC Module](image-url)
4.1.2 Configuration of the ADC Module Clock screen

1. In the **ADC Disabled Flag** section of the screen, the “Enable Module; the peripheral is supplied with the clock signal” option is selected.
2. The **Input Clock** for the ADC module (fADC) is 48.0 MHz.

![ADC Module Clock screen](image)

**Figure 6** ADC Module Clock screen
4.1.3 Configuration of the ADC General screen in Basic Mode

1. Select “Basic mode; the basic settings have been configured”

With Basic mode selected, the following options on the General tab are automatically configured by DAVE™:

- Analog Clock Divider – fADC / 6
- Conversion result – 10 bit resolution
- Enable arbitration slot 0 (Sequential source)
- Arbitration started by pending conversion request

![Figure 7 ADC General screen in Basic Mode](image-url)
4.1.4 Configuration of the ADC Channels screen in Basic Mode

In Basic mode, all of the available channels are automatically selected by DAVE™.

![Figure 8 ADC Channels screen in Basic Mode](image)

**Figure 8** ADC Channels screen in Basic Mode
4.1.5 Configuration of the ADC Channel 0 General Settings screen, in Basic Mode

- Channel 3 is selected
- Result register 0 is selected
- Vddp, Vssp is selected as reference voltage

The same configuration is applied for the other selected channels.

Figure 9 ADC Channel 0 General Settings screen in Basic Mode
4.1.6 Configuration of the ADC RequestSources screen in Basic Mode

- Sequential source 0 is enabled
- Priority is set to low
- Wait-for-start mode is selected

![ADC RequestSources screen in Basic Mode]

Figure 10 ADC RequestSources screen in Basic Mode
4.1.7 Configuration of the ADC Result Register screen in Basic Mode

- No Filter is selected
- Wait-For-Read mode is selected
- Valid-Flag-Reset is selected

![Figure 11 ADC Result Register screen in Basic Mode](image-url)
4.1.8 Configuration of the ADC Interrupts screen in Basic Mode

- No ADC interrupts are selected

![ADC Interrupts screen in Basic Mode](image)

**Figure 12** ADC Interrupts screen in Basic Mode
4.1.9 Configuration of the ADC Functions screens in Basic Mode

The following functions are automatically selected by DAVE™ in Basic mode:

- ADC_vInit - Functions screen
- ADC_Convert_8bit - Functions(3) screen
- ADC_Convert_10bit – Functions(3) screen

Note: In this application example the ADC_Convert_10bit is not used and so can be unchecked; see Functions(3)

Figure 13   ADC Functions screen in Basic Mode
Figure 14 ADC Functions(2) screen in Basic Mode
Figure 15  ADC Functions(3) screen in Basic Mode
4.2 Configuration of Timer T2

4.2.2 Configuration of the T2 Module Clock screen

- Select "Enable Module; the peripheral is supplied with the clock signal"
- The Input Clock for the T2 module is 24.0 MHz

![T2 Module Clock screen](image)

Figure 16 T2 Module Clock screen
4.2.3 Configuration of the T2 Timer 2 screen

- Enable Prescaler (PREN) selected in Prescalar Options
- Select Prescaler (PRE) value as fPCLK/128
- Select 16-bit timer functionality in Timer / Counter Select (C_T2)
- Select Automatic reload when timer 2 overflows in Capture/Reload Select (CP_RL2)
- Set Timer overflow [usec] in the Timer / Counter Register section, as 250 msecs
- Set Enable Timer 2 interrupt (ET2) and Enable Overflow interrupt in the Interrupt Control section

Figure 17 T2 Timer 2 screen
4.2.4 Configuration of the T2 T2EX screen

- No external trigger is selected; i.e. **Timer2 External Trigger Input Select** is set to None

![T2 T2EX screen](image)

Figure 18 T2 T2EX screen
4.2.5 Configuration of the T2 Interrupts screen

- Timer 2 Interrupt is selected

![T2 Interrupts screen](image)

<table>
<thead>
<tr>
<th>Level</th>
<th>Interrupt Source</th>
<th>Priority 0</th>
<th>Priority 1</th>
<th>Priority 2</th>
<th>Priority 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 0</td>
<td>Non Maskable Interrupt (NMI)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 1</td>
<td>EXINT 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 2</td>
<td>Timer 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 3</td>
<td>EXINT 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 4</td>
<td>Timer 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 5</td>
<td>UART</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 6</td>
<td>Timer 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 7</td>
<td>ADC and ORC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 8</td>
<td>SSC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 9</td>
<td>EXINT 2 / IIC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 10</td>
<td>EXINT[5:3] / RTC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 11</td>
<td>CCUB Node 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 12</td>
<td>CCUB Node 1 / LED and TS (Time Frame)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 13</td>
<td>CCUB Node 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 14</td>
<td>CCUB Node 3 / LED and TS (Time Slice)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 19 T2 Interrupts screen
4.2.6 Configuration of the T2 Functions screen

The following functions are selected

- T2_vInit
- T2_vStartTmr
- T2_vStopTmr
- T2_vITmr2

![Figure 20 T2 Functions screen](Image)
4.3 Configuration of emulated EEPROM ROM Library

4.3.1 Configuration of the EEPROM Functions screen

The following functions are selected:

- EEPROM_vInit
- InitEEPROM
- FixEEPROM
- ReadEEPROM
- WriteEEPROM
- The Emulated EEPROM Size is set to 128 (Size in Bytes)

![EEPROM Functions screen](image)

Figure 21  EEPROM Functions screen
4.4 Configuration of UART

4.4.1 Configuration of the UART screen

- P0.5 (TXD_3) is selected as Transmit pin
- P1.0 (RXD_2) is selected as Receive pin
- Baud rate generator is selected as Baudrate Source
- Receiver is enabled
- Mode 1 is selected

![UART (Serial Interface) Configuration](image)

Figure 22 UART screen
4.4.2 Configuration of the UART BRG screen

- In BRG Settings, the Required baud rate [kbaud] is configured as 9.6 Kbaud

![Figure 23 UART BRG screen](image)

- **Prescaler (BRPRE)**
- **Required baud rate [kbaud]**
- **Min. baud rate [baud]**
- **Max. baud rate [baud]**

![BErg Settings](image)
4.4.3 Configuration of the UART Functions screen

The following functions are selected:
- UART_vInit
- UART_bRxReady
- UART_ubGetData8

![UART Functions screen](image)

Figure 24 UART Functions screen
4.5 Example Code for Data Logger Application

The following C-code example shows the Data Logger application using the Emulated EEPROM ROM Library.

The code is for a XC822M device with the Keil v9.0 compiler.

Tools Used:
- DAVE™ v2.1r22
- XC82xM_Series_v2_0.dip
- XC822 Easy Kit Board
- XC800_FLOAD_v50d12
- Keil Compiler v9.0
- DAVE-Bench for XC800

The mode 3 emulation scheme has been used, so the emulated EEPROM size (in bytes) is:
- 124 data + 4 status bytes

The address range used by the ROM Library is:
- 0xAE00 to 0xAFFF for XC83x devices
- 0x0E00 to 0xFFF for XC82x devices

The following code is the main function. The user can enter a command either to read the content of the EEPROM or to write ADC converted data into the EEPROM. Each write of the ADC converted data into the EEPROM occurs at an interval of 1 second, the interval being timed by Timer 2.

The main function is available in the MAIN.C file of DAVE™ generated code.

MAIN.C

```c
void main(void)
{
    // USER CODE BEGIN (MAIN_Main,2)
    char i = 0, j = 0;
    // USER CODE END
    MAIN_vInit();
    // USER CODE BEGIN (MAIN_Main,3)
    while(!RI);
    RI = 0;
    TI = 1;
    // USER CODE END
    while(1)
    {
        // USER CODE BEGIN (MAIN_Main,4)
        printf("\n\nEnter command for EEPROM emulation test");
        printf("\n r - read converted results stored in EEPROM");
        printf("\n w - start adc conversion and store the converted results into EEPROM");
        printf("\n\n");
        command = getkey();
```

Application Note 30 V1.0, 2011-02
switch(command){
    case 'r':
        printf("\nReading converted results stored in EEPROM\n");
        for (i=0; i<4; i++){
            printf("Result   : ");
            putchar((i + '0'));
            printf(" ");
            printf("\n");
            if(STD_OK == ReadEEPROM(i, convresult, &config)){
                for (j = 0; j < 31; j++)
                {
                    printf("  0x%x\n", (int) convresult[j]);
                }
            }
            else{
                printf("Result not found");
            }
        }
        break;
    case 'w':
    case 'W':
        printf("Storing converted results to EEPROM\n");
        j = 0;
        /* Since only 31 bytes can be written last byte is a status byte. So,
         16 used as multiple of 2.
         32 * 4 = 128 bytes of data can be written but 4 bytes are used for
         status byte hence 124 bytes
         writing is possible. But we used 31 * 4 = 124 bytes writing of
         conversion results.
         */
        for (i=0;i<31;i++)
        {
            T2_vStartTmr();
            while (!(FlagT2OvrFlow == 4)); // Wait for 1 sec (250000.000
             usec * 4 = 1.00 sec)
            FlagT2OvrFlow = 0;
            T2_vStopTmr();
            for (uwCount = 0x000; uwCount < 0xFFF; uwCount++)
            uwResult = ADC_Convert_8bit(3);
            convresult[j++] = (int)uwResult; // Get
        }
}

Converted result from channel 3
printf("\r\nconverted result = 0x%x\r\n", (int) convresult[i]);
}
printf("\r\nPlease enter result ID[0-3]\r\n");
command = getkey();
command -= '0';
WriteEEPROM(command, convresult, &config);
for(i=0;i<31;i++)
{
    convresult[i++] = 0x00;
}
printf( "\r\nBuffer contents are cleared after writing to EEPROM.\r\n" );
break;
default:
    printf( "\r\nNot supported command" );
    break;
}
// USER CODE END

} // End of function main

The following code shows the getkey function definition. This code is available in the MAIN.C file.

char getkey(void){
    unsigned char c;
    while(!UART_bRxReady());
    c = UART_ubGetData8();
    return c;
}
ADC.C

The following code shows the ADC Basic mode configuration.
This code is available in the ADC.C file.

```c
void ADC_vInit(void)
{
    // USER CODE BEGIN (ADC_Init,2)
    // USER CODE END

    /// Configuration of Global Control:
    /// - the ADC module clock is enabled
    /// - the ADC module clock = 48.00 MHz
    /// - the ADC Basic mode is selected
    /// - the result is 10 bits wide

    // Configure global control functions
    SFR_PAGE(_ad0, noSST);         // switch to page 0
    ADC_GLOBCTR   =  0x30;         // load global control register

    /// Configuration of Priority and Arbitration:
    /// - the priority of request source 0 is low
    /// - the wait-for-start mode is selected for source 0
    /// - the wait-for-start mode is selected for source 1
    /// - the arbitration started by pending conversion request is selected
    /// - Arbitration Slot 0 is enabled
    /// - Arbitration Slot 1 is disabled
    ADC_PRAR      =  0x50;           // load Priority and Arbitration register

    // Configuration of Channel Control Registers:
    /// Configuration of Channel 0
    /// - the result register0 is selected

    // Application Note 33 V1.0, 2011-02
```
/// - the limit check 0 is selected
ADC_CHCTR0 = 0x00; // load channel control register
/// Configuration of Channel 1
/// - the result register0 is selected
/// - the limit check 0 is selected
ADC_CHCTR1 = 0x00; // load channel control register
/// Configuration of Channel 2
/// - the result register0 is selected
/// - the limit check 0 is selected
ADC_CHCTR2 = 0x00; // load channel control register
/// Configuration of Channel 3
/// - the result register0 is selected
/// - the limit check 0 is selected
ADC_CHCTR3 = 0x00; // load channel control register
SFR_PAGE(_ad0, noSST); // switch to page 0
/// -----------------------------------------------------------------------
/// Configuration of Sample Time Control:
/// -----------------------------------------------------------------------
ADC_INPCR0 = 0x00; // load input class register
/// -----------------------------------------------------------------------
/// Configuration of Out of range comparator:
/// -----------------------------------------------------------------------
ADC_ENORC = 0x00; // load out of range comparator register
SFR_PAGE(_ad4, noSST); // switch to page 4
/// -----------------------------------------------------------------------
/// Configuration of Out of range comparator edge trigger:
/// -----------------------------------------------------------------------
ADC_CNF = 0x00; // load out of range comparator trigger edge
// select register
/// -----------------------------------------------------------------------
/// Configuration of alias register:
/// -----------------------------------------------------------------------
ADC_ALR0 = 0x00; // load alias register 0
/// -----------------------------------------------------------------------
/// Configuration of Result Control Registers:
/// -----------------------------------------------------------------------
/// Configuration of Result Control Register 0
/// - the data reduction filter is disabled
/// - the digital low pass filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is enabled
///  - the VF reset by read access to RESRxH
ADC_RCR0  =  0xC0;         // load result control register 0
///  Configuration of Result Control Register 1
///  - the data reduction filter is disabled
///  - the digital low pass filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is disabled
///  - the VF unchanged by read access to RESRxH
ADC_RCR1  =  0x00;         // load result control register 1
///  Configuration of Result Control Register 2
///  - the data reduction filter is disabled
///  - the digital low pass filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is disabled
///  - the VF unchanged by read access to RESRxH
ADC_RCR2  =  0x00;         // load result control register 2
///  Configuration of Result Control Register 3
///  - the data reduction filter is disabled
///  - the digital low pass filter is disabled
///  - the event interrupt is disabled
///  - the wait-for-read mode is disabled
///  - the VF unchanged by read access to RESRxH
ADC_RCR3  =  0x00;         // load result control register 3
///  ---------------------------------------------------------------
///  Channel Interrupt Node Pointer:
///  ---------------------------------------------------------------
///  - the SR 0 line become activated if any channel interrupt is generated
///  ---------------------------------------------------------------
///  Out of range comparator Interrupt Node Pointer:
///  ---------------------------------------------------------------
///  - the SR 1 line become activated if out of range comparator interrupt
///  for any channel is generated
///  ---------------------------------------------------------------
///  Event Interrupt Node Pointer:
///  ---------------------------------------------------------------
///  - the SR 0 line become activated if the event 0-1 interrupt is
///  generated
/// - the SR 0 line become activated if the event 4-7 interrupt is generated
SFR_PAGE(_ad0, noSST); // switch to page 0

/// Configuration of Limit Check Boundary:
ADC_LCBR0 = 0x70;       // load limit check boundary register 0
ADC_LCBR1 = 0xB0;       // load limit check boundary register 1
SFR_PAGE(_ad6, noSST);  // switch to page 6

/// Configuration of Conversion Queue Mode Register:
/// - the gating line is permanently 1
/// - the external trigger is disabled
ADC_QMR0 = 0x01;       // load queue mode register

/// Configuration of Conversion Request Mode Registers:
/// - the gating line is permanently 0
/// - the external trigger is disabled
/// - the source interrupt is disabled
/// - the autoscan functionality is disabled
ADC_CRMR1 = 0x00;     // load conversion request mode register 1

SFR_PAGE(_ad0, noSST); // switch to page 0
ADC_GLOBCTR |= 0x80;  // turn on Analog part

/// - Out of range comparator -Interrupt (ORCIEN) remains disabled
/// - Channel limit checking -Interrupt (CLCIEN) remains disabled
/// - ADC -Interrupt (EADC) remains disabled

// USER CODE BEGIN (ADC_Init,3)
// USER CODE END

} // End of function ADC_vInit
ADC Basic Mode Functions (in ADC.C)

The following code shows the ADC Basic mode function definition which is used to start ADC conversion for the requested channel. When the conversion is complete, read the 8-bit result from result register 0.

This function, which is available in the ADC.C file, takes the channel number as the input parameter.

```
uword ADC_Convert_8bit(ubyte ubChannelNum)
{
    ubyte ubVal = 0;
    uword uwResult = 0xFFF;
    ubVal = ubVal + (ubChannelNum & 0x07);
    SFR_PAGE(ADC_QINR0_PAGE, SST1); // switch to page 6
    ADC_QINR0 = ubVal;               // requested channel
    SFR_PAGE(_ad0, RST1);            // restore the old ADC page
    SFR_PAGE(_ad0, SST1);            // switch to page 0
    // wait until conversion is complete
    while ((ADC_GLOBSTR & 0x01));
    SFR_PAGE(_ad0, RST1);            // restore the old ADC page
    SFR_PAGE(ADC_RESR0L_PAGE, SST1); // switch to page 2
    if ( ADC_RESR0L & 0x10 )         // if Result Register 0 contains valid data
    {
        // 8-bit conversion (without accumulation)
        uwResult = (((ADC_RESR0L >> 7) & 0x01); // Result Register 0 Low
        uwResult = (((uword)(ADC_RESR0H << 1)) + uwResult); // Result Register 0 High
    }
    SFR_PAGE(_ad0, RST1);            // restore the old ADC page
    return(uwResult);
} // End of function ADC_Convert_8bit
```
EEPROM Emulation Scheme (in EEPROM.C)

The following code shows the EEPROM Emulation scheme configuration.

This code is available in the EEPROM.C file

```c
//****************************************************************************
// @Function      void EEPROM_vInit(void)
//----------------------------------------------------------------------------
// @Description   This is the initialization function of the EEPROM function
//                library. It is assumed that the SFRs used by this library
//                are in their reset state.
//                User can call InitEEPROM() function inside this function
//                for initialization.
//----------------------------------------------------------------------------
// @Returnvalue   None
//----------------------------------------------------------------------------
// @Parameters    None
//----------------------------------------------------------------------------
// @Date          09-08-2010
//****************************************************************************
// USER CODE BEGIN (EEPROM_Init,1)
// USER CODE END

void EEPROM_vInit(void)
{
    /// EEPROM emulation required data structure
    char idata buffer[32];
    unsigned char eeprom_status;
    // USER CODE BEGIN (EEPROM_Init,2)
    // USER CODE END

    eeprom_status = InitEEPROM(MODE_3, &config);
    if(0 != eeprom_status){
        if( 0x0 == config.ActiveSector){
            /// Erase abort detected or first time using EEPROM
            /// Initialise invalid sectors
            FixEEPROM(eeprom_status, buffer);
        }else{
            /// Programming or erasing aborted suddenly during reclaiming operation
            /// Depending on user application they might want to take additional action
        }
    }
}```
// USER CODE BEGIN (EEPROM_Init,3)
// USER CODE END

/// Initialise invalid sectors. Data would be erased
    FixEEPROM(eeprom_status, buffer);

/// Check that EEPROM sectors are in valid condition
    eeprom_status = InitEEPROM(MODE_3, &config);
    if(0 != eeprom_status){
/// Fatal failure in EEPROM
    // USER CODE BEGIN (EEPROM_Init,4)
    // USER CODE END
    while(1);
    }
}

// USER CODE BEGIN (EEPROM_Init,5)
// USER CODE END

} // End of function EEPROM_vInit
Emulated EEPROM ROM Library Routines (EEPROM.H file)
The following code shows the declarations with descriptions of various Emulated EEPROM ROM Library routines. This code is available in the EEPROM.H file

//****************************************************************************
// @Function ubyte InitEEPROM(ubyte idata mode, EEPROMInfo *config)
//*****************************************************************************
// @Description Initialised EEPROM based on selected eeprom emulation scheme.
// Function will initialise EEPROMInfo data structure based on
// the specified EEPROM emulation scheme.
// It will then search for aborted erase situation and return
// the status indicating which sector needs to be erased. The
// algorithm to determine this is given below.
// Only 1 valid bit should be set in any sector. If both
// sector contain valid bits, i.e. programming or erasing was
// aborted during & "reclaiming" operation. The sector with
// the least amount of valid bits is the active sector except
// when only one wordline is written. Function assume that at
// least one sector will have more valid bits than the other.
// If both logical sector are properly initialised, it will
// update the EEPROMInfo->ActiveSector and
// EEPROMInfo->WriteAddress value.
// Weakness, functions unable to detect invalid active sector
// indicator. Thus if active sector indicator is 0xF, it will
// be detected as a valid indicator.
//****************************************************************************
// @Returnvalue status Status of EEPROM emulation
// 0x0 -
// No problem with EEPROM emulation
// 0x1 -
// Sector 8 and 9 needs to be erased
// 0x2 -
// Sector 6 and 7 needs to be erased
// 0x3 -
// Sector 6 to 9 needs to be erased
//****************************************************************************
// @Parameters idata mode:
// mode : size of eeprom emulation scheme i.e. 32, 64, 96 or
// 128
// @Parameters *config:
//              config : pointer to EEPROM configuration
//----------------------------------------------------------------------------
// @Date      09-08-2010
//****************************************************************************
ubyte InitEEPROM(ubyte idata mode, EEPROMInfo *config) small;
//****************************************************************************
// @Function   void FixEEPROM(ubyte idata sector_status, ubyte idata
//                      *buffer)
//----------------------------------------------------------------------------
// @Description Fix EEPROM based on returned status of InitEEPROM.
//                Erase the logical sector based on sector status. It will
//                then write 0x7 to the status byte of the last WL of sector
//                to indicate sector has been erased correctly.
//----------------------------------------------------------------------------
// @Returnvalue None
//----------------------------------------------------------------------------
// @Parameters idata sector_status:
//              sector_status : sector status reported by InitEEPROM()
// @Parameters idata *buffer:
//              buffer : pointer to buffer to be used for programming
//              status byte
//----------------------------------------------------------------------------
// @Date      09-08-2010
//****************************************************************************
void FixEEPROM(ubyte idata sector_status, ubyte idata *buffer) small;
//****************************************************************************
// @Function   ubyte ReadEEPROM(ubyte idata address, char idata *dst,
//                      EEPROMInfo *config)
//----------------------------------------------------------------------------
// @Description Read content given by logical address
//              Search current "active sector" by checking the status bytes
//              for the valid bit and requested logical address. Search
//              will start from the highest address of the active sector.
//              It will return 32 bytes of data(31 data + 1 status).
//              Function will return an error if the logical address can't
//              be found.
//----------------------------------------------------------------------------
// @Returnvalue status Read operation result
// STD_ERR -
// address was not found
// STD_OK - read
// operation successful
//------------------------------------------------------------------
// @Parameters idata address:
// address : logical address to read from
// @Parameters idata *dst:
// dst : pointer to buffer to store requested data
// @Parameters *config:
// config : pointer to EEPROM configuration
//------------------------------------------------------------------
// @Date 09-08-2010
//****************************************************************************
ubyte ReadEEPROM(ubyte idata address, char idata *dst, EEPROMInfo *config) small;
//****************************************************************************
// @Function ubyte WriteEEPROM(ubyte idata address, char idata *src, EEPROMInfo *config)
//****************************************************************************
// @Description Write content to logical address.
// Program data in buffer into location pointed by
// EEPROMInfo->WriteAddress. Each write operation is 32 bytes
// (31 data + 1 status). The buffer must be 32 bytes and
// reside in IRAM.
// Upon completion of write operation, function will check if
// write operation occur in a new active sector.
// If no, update the EEPROMInfo->WriteAddress to the next
// available address.
// If yes, copy other valid data from the previous sector to
// the new active sector.
// After that, erase the previous sector and update
// EEPROMInfo->ActiveSector.
//****************************************************************************
// @Returnvalue status Read operation result
// STD_ERR -
// write operation not successful
// STD_OK -
// write operation successful
//----------------------------------------------------------------------------
// @Parameters  idata address:
//                address : logical address to write to
// @Parameters  idata *src:
//                dst : pointer to buffer containing data to be written
// @Parameters  *config:
//                config : pointer to EEPROM configuration
//----------------------------------------------------------------------
// @Date          09-08-2010
//**************************************************************************
ubyte WriteEEPROM(ubyte idata address, char idata *src, EEPROMInfo *config) small;
Timer 2 Configurations (T2.C file)

The following code shows the Timer 2 configurations used to get a delay of 250000 usec. This also includes the Interrupt Service Routine where the timer overflow flag has been set.

These functions are available in the T2.C file.

---

```c
//****************************************************************************
// @Function      void T2_vInit(void)
// @Description   This is the initialization function of the Timer 2 function
// library. It is assumed that the SFRs used by this library
// are in their reset state.
// The following SFRs and SFR fields will be initialized:
// T2_RC2H/RC2L     - reload/capture timer 2 register
// T2_T2H/T2L      - timer 2 register
// ET2             - timer 2 interrupt enable
// T2_T2MOD        - timer 2 mode register
// CP/RL2          - Capture/Reload select
// EXEN2           - External enable control
// TR2             - Timer2 run control

// USER CODE BEGIN (T2_Init,1)
// USER CODE END

void T2_vInit(void)
{
    // USER CODE BEGIN (T2_Init,2)
    // USER CODE END

    // Configuration of the Module Clock:
    // - the T2 module clock = 24.00 MHz
    // Operating Mode
}```
---

Application Note 44 V1.0, 2011-02
/// 16-bit timer function with automatic reload when timer 2 overflows
/// Prescaler enabled - input clock = fPCLK/128
/// the timer 2 resolution is 5.333 usec
/// the timer 2 overflow is 250000.000 usec
/// timer 2 interrupt: enabled
/// timer 2 remains stopped
///
// Register Initialization
//
T2_T2LH       = 0x48E5;       // load timer 2 register low & high bytes
T2_RC2LH      = 0x48E5;       // load timer 2 reload/capture register low & high bytes
T2_T2MOD      = 0x1E;         // load timer 2 mode register
T2_T2CON1     = 0x02;         // load timer 2 control register 1
/// timer 2 interrupt: enabled
ET2 = 1;                       // Enable interrupt
// USER CODE BEGIN (T2_Init,3)
// USER CODE END
// timer 2 remains stopped
} // End of function T2_vInit

//******************************************************************************************
// @Function    void T2_viTmr2(void)
//******************************************************************************************
// @Description This is the service routine for the Timer 2 interrupt.
// Depending on the selected operating mode it is called when TF2 is set by an overflow or
// underflow of the timer 2 register or when EXF2 is set by a negative transition on
// T2EX.
// Please note that you have to add application specific code to this function.
//******************************************************************************************
// @Returnvalue None
//******************************************************************************************
// @Parameters None
//******************************************************************************************
// @Date 09-08-2010
//******************************************************************************************
void T2_viTmr2(void) interrupt T2INT
{
    if ((TF2))
    {
        TF2 = 0;

        FlagT2OvrFlow++;  
    }

    Note: This is how the DataLogger functionality can be achieved in applications, using the Emulated EEPROM ROM Library.
5 Data Logger Application Results

The following screen shot shows the main application result captured through the serial port (HyperTerminal).

Depending upon the user input command, the Analog-to-Digital converted data has either been read from the EEPROM or written into the EEPROM. Converted data has been captured at an interval of 1 second using Timer 2 as the time delay module.

Figure 25  Storage of Converted Results to EEPROM
Figure 26 Reading Converted Results from the EEPROM
The following table shows the complete output for 124 bytes of data that has been stored to the EEPROM and then read from it.

### Table 2: Storing to / Reading from the EEPROM

<table>
<thead>
<tr>
<th>Converted Results Stored into EEPROM</th>
<th>Converted Results Read from EEPROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storing converted results to EEPROM</td>
<td>Result   : 0</td>
</tr>
<tr>
<td>converted result = 0x7b</td>
<td>0x7b</td>
</tr>
<tr>
<td>converted result = 0x7b</td>
<td>0x7b</td>
</tr>
<tr>
<td>converted result = 0x7c</td>
<td>0x7c</td>
</tr>
<tr>
<td>converted result = 0x7b</td>
<td>0x7b</td>
</tr>
<tr>
<td>converted result = 0x78</td>
<td>0x78</td>
</tr>
<tr>
<td>converted result = 0xff</td>
<td>0xff</td>
</tr>
<tr>
<td>converted result = 0xd9</td>
<td>0xd9</td>
</tr>
<tr>
<td>converted result = 0xbc</td>
<td>0xbc</td>
</tr>
<tr>
<td>converted result = 0xff</td>
<td>0xff</td>
</tr>
<tr>
<td>converted result = 0xff</td>
<td>0xff</td>
</tr>
<tr>
<td>converted result = 0xff</td>
<td>0xff</td>
</tr>
<tr>
<td>converted result = 0xff</td>
<td>0xff</td>
</tr>
<tr>
<td>converted result = 0xc3</td>
<td>0xc3</td>
</tr>
<tr>
<td>converted result = 0xb0</td>
<td>0xb0</td>
</tr>
<tr>
<td>converted result = 0xa6</td>
<td>0xa6</td>
</tr>
<tr>
<td>converted result = 0xff</td>
<td>0xff</td>
</tr>
<tr>
<td>converted result = 0xce</td>
<td>0xce</td>
</tr>
<tr>
<td>converted result = 0xbd</td>
<td>0xbd</td>
</tr>
<tr>
<td>converted result = 0xb2</td>
<td>0xb2</td>
</tr>
<tr>
<td>converted result = 0xa8</td>
<td>0xa8</td>
</tr>
<tr>
<td>converted result = 0xa0</td>
<td>0xa0</td>
</tr>
<tr>
<td>converted result = 0x96</td>
<td>0x96</td>
</tr>
<tr>
<td>converted result = 0x8d</td>
<td>0x8d</td>
</tr>
<tr>
<td>converted result = 0x83</td>
<td>0x83</td>
</tr>
<tr>
<td>converted result = 0x7b</td>
<td>0x7b</td>
</tr>
<tr>
<td>converted result = 0x76</td>
<td>0x76</td>
</tr>
<tr>
<td>converted result = 0x72</td>
<td>0x72</td>
</tr>
<tr>
<td>converted result = 0x73</td>
<td>0x73</td>
</tr>
<tr>
<td>converted result = 0x74</td>
<td>0x74</td>
</tr>
<tr>
<td>converted result = 0x78</td>
<td>0x78</td>
</tr>
<tr>
<td>converted result = 0x74</td>
<td>0x74</td>
</tr>
</tbody>
</table>

Application Note 49 V1.0, 2011-02
<table>
<thead>
<tr>
<th>Storing converted results to EEPROM</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>converted result = 0x67</td>
<td>0x67</td>
</tr>
<tr>
<td>converted result = 0x63</td>
<td>0x63</td>
</tr>
<tr>
<td>converted result = 0x72</td>
<td>0x72</td>
</tr>
<tr>
<td>converted result = 0x91</td>
<td>0x91</td>
</tr>
<tr>
<td>converted result = 0x81</td>
<td>0x81</td>
</tr>
<tr>
<td>converted result = 0x7a</td>
<td>0x7a</td>
</tr>
<tr>
<td>converted result = 0x79</td>
<td>0x79</td>
</tr>
<tr>
<td>converted result = 0x7a</td>
<td>0x7a</td>
</tr>
<tr>
<td>converted result = 0x79</td>
<td>0x79</td>
</tr>
<tr>
<td>converted result = 0x75</td>
<td>0x75</td>
</tr>
<tr>
<td>converted result = 0x6d</td>
<td>0x6d</td>
</tr>
<tr>
<td>converted result = 0x64</td>
<td>0x64</td>
</tr>
<tr>
<td>converted result = 0x60</td>
<td>0x60</td>
</tr>
<tr>
<td>converted result = 0x5f</td>
<td>0x5f</td>
</tr>
<tr>
<td>converted result = 0x61</td>
<td>0x61</td>
</tr>
<tr>
<td>converted result = 0x65</td>
<td>0x65</td>
</tr>
<tr>
<td>converted result = 0x6b</td>
<td>0x6b</td>
</tr>
<tr>
<td>converted result = 0x6c</td>
<td>0x6c</td>
</tr>
<tr>
<td>converted result = 0x6a</td>
<td>0x6a</td>
</tr>
<tr>
<td>converted result = 0xff</td>
<td>0xff</td>
</tr>
<tr>
<td>converted result = 0xca</td>
<td>0xca</td>
</tr>
<tr>
<td>converted result = 0xb0</td>
<td>0xb0</td>
</tr>
<tr>
<td>converted result = 0xa1</td>
<td>0xa1</td>
</tr>
<tr>
<td>converted result = 0x9b</td>
<td>0x9b</td>
</tr>
<tr>
<td>converted result = 0x97</td>
<td>0x97</td>
</tr>
<tr>
<td>converted result = 0x92</td>
<td>0x92</td>
</tr>
<tr>
<td>converted result = 0x89</td>
<td>0x89</td>
</tr>
<tr>
<td>converted result = 0x7d</td>
<td>0x7d</td>
</tr>
<tr>
<td>converted result = 0x73</td>
<td>0x73</td>
</tr>
<tr>
<td>converted result = 0x6e</td>
<td>0x6e</td>
</tr>
<tr>
<td>converted result = 0x6e</td>
<td>0x6e</td>
</tr>
<tr>
<td>Storing converted results to EEPROM</td>
<td>Result</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>converted result = 0x6e</td>
<td>0x6e</td>
</tr>
<tr>
<td>converted result = 0x68</td>
<td>0x68</td>
</tr>
<tr>
<td>converted result = 0x6a</td>
<td>0x6a</td>
</tr>
<tr>
<td>converted result = 0x6e</td>
<td>0x6e</td>
</tr>
<tr>
<td>converted result = 0x72</td>
<td>0x72</td>
</tr>
<tr>
<td>converted result = 0x71</td>
<td>0x71</td>
</tr>
<tr>
<td>converted result = 0x6a</td>
<td>0x6a</td>
</tr>
<tr>
<td>converted result = 0x64</td>
<td>0x64</td>
</tr>
<tr>
<td>converted result = 0x5e</td>
<td>0x5e</td>
</tr>
<tr>
<td>converted result = 0x5c</td>
<td>0x5c</td>
</tr>
<tr>
<td>converted result = 0x5c</td>
<td>0x5c</td>
</tr>
<tr>
<td>converted result = 0x60</td>
<td>0x60</td>
</tr>
<tr>
<td>converted result = 0x64</td>
<td>0x64</td>
</tr>
<tr>
<td>converted result = 0x69</td>
<td>0x69</td>
</tr>
<tr>
<td>converted result = 0x6a</td>
<td>0x6a</td>
</tr>
<tr>
<td>converted result = 0x69</td>
<td>0x69</td>
</tr>
<tr>
<td>converted result = 0x64</td>
<td>0x64</td>
</tr>
<tr>
<td>converted result = 0x5f</td>
<td>0x5f</td>
</tr>
<tr>
<td>converted result = 0x5a</td>
<td>0x5a</td>
</tr>
<tr>
<td>converted result = 0x57</td>
<td>0x57</td>
</tr>
<tr>
<td>converted result = 0x56</td>
<td>0x56</td>
</tr>
<tr>
<td>converted result = 0x59</td>
<td>0x59</td>
</tr>
<tr>
<td>converted result = 0x5c</td>
<td>0x5c</td>
</tr>
<tr>
<td>converted result = 0x61</td>
<td>0x61</td>
</tr>
<tr>
<td>converted result = 0x65</td>
<td>0x65</td>
</tr>
<tr>
<td>converted result = 0x66</td>
<td>0x66</td>
</tr>
<tr>
<td>converted result = 0x65</td>
<td>0x65</td>
</tr>
<tr>
<td>converted result = 0x62</td>
<td>0x62</td>
</tr>
<tr>
<td>converted result = 0x5e</td>
<td>0x5e</td>
</tr>
<tr>
<td>converted result = 0x58</td>
<td>0x58</td>
</tr>
<tr>
<td>converted result = 0x56</td>
<td>0x56</td>
</tr>
</tbody>
</table>
### Storing converted results to EEPROM

<table>
<thead>
<tr>
<th>Converted result</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x60</td>
<td>0x60</td>
</tr>
<tr>
<td>0x5b</td>
<td>0x5b</td>
</tr>
<tr>
<td>0x58</td>
<td>0x58</td>
</tr>
<tr>
<td>0x57</td>
<td>0x57</td>
</tr>
<tr>
<td>0x57</td>
<td>0x57</td>
</tr>
<tr>
<td>0x59</td>
<td>0x59</td>
</tr>
<tr>
<td>0x5b</td>
<td>0x5b</td>
</tr>
<tr>
<td>0x5e</td>
<td>0x5e</td>
</tr>
<tr>
<td>0x62</td>
<td>0x62</td>
</tr>
<tr>
<td>0x64</td>
<td>0x64</td>
</tr>
<tr>
<td>0x64</td>
<td>0x64</td>
</tr>
<tr>
<td>0x64</td>
<td>0x64</td>
</tr>
<tr>
<td>0x61</td>
<td>0x61</td>
</tr>
<tr>
<td>0x60</td>
<td>0x60</td>
</tr>
<tr>
<td>0x5d</td>
<td>0x5d</td>
</tr>
<tr>
<td>0x5b</td>
<td>0x5b</td>
</tr>
<tr>
<td>0x58</td>
<td>0x58</td>
</tr>
<tr>
<td>0x56</td>
<td>0x56</td>
</tr>
<tr>
<td>0x55</td>
<td>0x55</td>
</tr>
<tr>
<td>0x54</td>
<td>0x54</td>
</tr>
<tr>
<td>0x53</td>
<td>0x53</td>
</tr>
<tr>
<td>0x53</td>
<td>0x53</td>
</tr>
<tr>
<td>0x52</td>
<td>0x52</td>
</tr>
<tr>
<td>0x55</td>
<td>0x55</td>
</tr>
<tr>
<td>0x57</td>
<td>0x57</td>
</tr>
<tr>
<td>0x5a</td>
<td>0x5a</td>
</tr>
<tr>
<td>0x5d</td>
<td>0x5d</td>
</tr>
<tr>
<td>0x5f</td>
<td>0x5f</td>
</tr>
<tr>
<td>0x61</td>
<td>0x61</td>
</tr>
<tr>
<td>0x61</td>
<td>0x61</td>
</tr>
<tr>
<td>0x60</td>
<td>0x60</td>
</tr>
</tbody>
</table>
6 Conclusion, Related Documents and Links

6.1 Conclusion

This application note describes the implementation of Data Logger functionality using the XC82x/XC83x Emulated EEPROM ROM Library in DAVE™, with the ADC converted data as storage data. In this example, 124 bytes of converted data have been stored and can be read by the user as required, using the available commands.

6.2 Related Documents and Links

- **XC82x User’s Manual PDF v1.1** (released 2nd July 2010) 5.3 MB
- **XC82x Product Information**
- **XC83x Product Information**
- **Starter Kit (Board Manual), XC82x/XC83x Development Tools and Software**
- **DAVE™ for the Infineon XC82x/XC83x microcontroller Family**
- Application Note: **XC82x/XC83x Analog-to-Digital Converter Basic and Advanced Mode using DAVE™**
- XC866 EEPROM Emulation **PDF** and **EXE**