M460 MP3 Recorder

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This example code encodes sound recorded from audio codec to MP3 format and stores it to a microSD card. In turn, this MP3 file can be played by the audio codec.
BSP Version	M460 BSP CMSIS V3.00.001
Hardware	NuMaker-M467HJ V1.0

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1 Overview

This example code uses audio codec to record sound. After encoding the sound data to MP3 format, the encoded data will be written to a microSD card.

To use this example code, user needs to prepare a microSD card and insert it into the T-Flash card slot. Then, user can press button BTN0 to start recording sound and press button BTN1 to stop recording. The MP3 file will be played automatically after the recording is finished or played when the button BTN1 is pressed. User can choose whether to record in real time by defining REC_IN_RT. If REC_IN_RT is defined, while the recording sound data is received, the data will be encoded to MP3 format and stored in the microSD card at the same time. Otherwise, if REC_IN_RT is not defined, the recording sound data will be stored in HyperRAM first until recording is stopped, and then the data will be encoded to MP3 format and stored in the microSD card.

The MP3 recorder is suitable for vocal recording and supports mono mode. The data length is 16 bits, the default sampling rate is 16 kHz and the default bit rate is 64 kbps. Furthermore, the recorder provides different combinations of sampling rates and bit rates, as listed in Table 1-1. User can configure them by defining REC_SAMPLE_RATE and REC_BIT_RATE.

Sampling Rate (Hz)	Bit Rate (kbps)
32,000	<= 128
16,000	<= 64
8,000	<= 32

Table 1-1 Supported Sampling Rate and Bit Rate

1.1 Demo Result

The demo results are shown below.

```
+-------------------------------------------------------------------+
|                 MP3 Recorder Sample with Audio Codec              |
+-------------------------------------------------------------------+
 Please insert a microSD card
 Press BTN0 button to start recording and press BTN1 button to stop recording
 Press BTN1 button can also play MP3 file from microSD card
***** card insert !

Configure NAU8822 ...[OK]
[NAU8822] Configure Sampling Rate to 16000

MPEG-II layer III, mono  Psychoacoustic Model: Shine
Bitrate: 64 kbps  De-emphasis: none   Original

Start recording ... (online)
Encode and write out the MP3 file .......... Done !
```

Figure 1-1 Demo Result of Online Recording

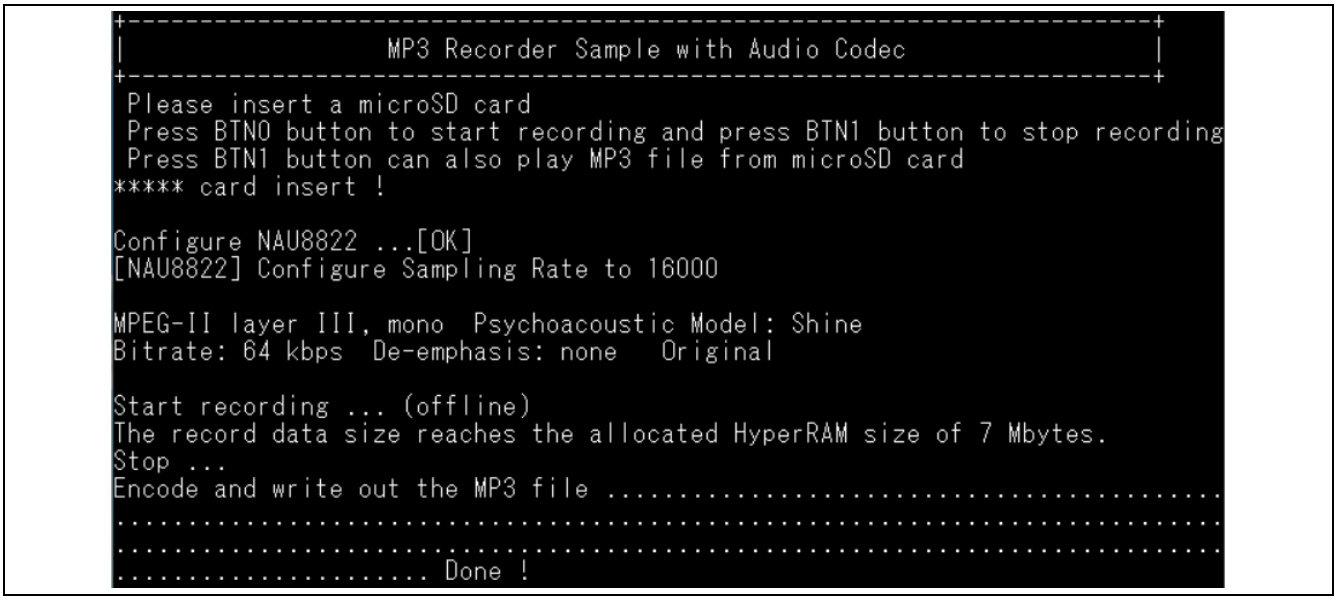

Figure 1-2 Demo Result of Offline Recording

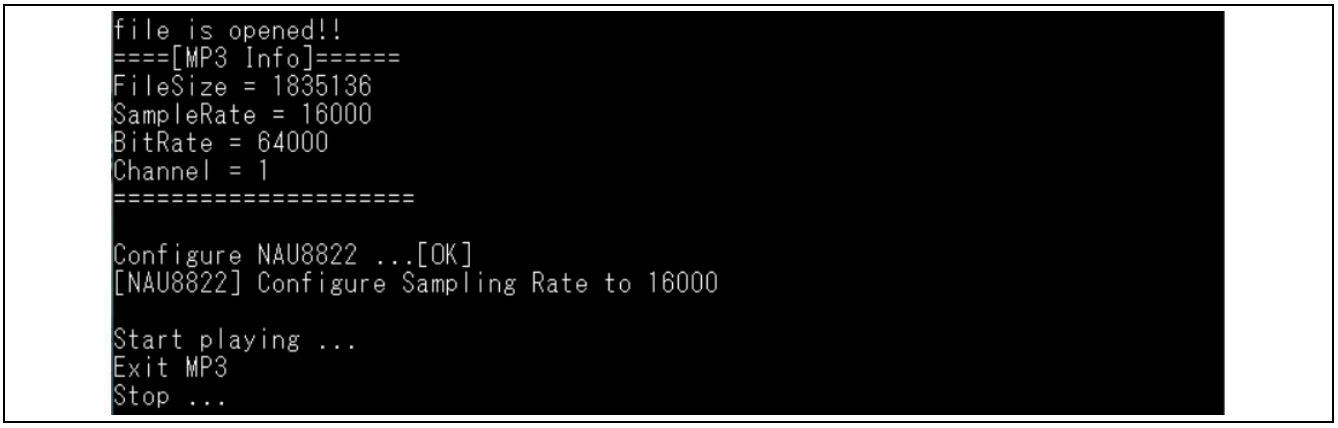

Figure 1-3 Demo Result of Playing Sound

2 Code Description

This example code provides user with online recording and offline recording. User can choose one of them to use according to whether REC_IN_RT is defined in config.h or not.

In main() of main.c, call SYS_Init() and SD_Init() to initialize peripheral clock and related multi-function pin, including UART, I²S, I²C, PDMA, HBI and SD. In the while(1) loop, always check the pin state of PH.0 and PH.1, g_u32RecordStart and g_u32RecordDone value, to start or stop recording.

When PH.0 is 1, g_u32RecordStart and g_u32RecordDone are 0, first, call ClearHyperRAM() only when REC_IN_RT is not defined to clear all data in HyperRAM, then call Recorder_Init() to configure recording conditions, initialize I²S, audio codec and encoder, and then enable I²S receive function to start receiving sound data. When g_u32RecordDone is 1, disable I²S receive function to stop receiving sound data and then call MP3Recorder() to encode sound data to MP3 format and store the encoded data in the microSD card if REC_IN_RT is not defined.

If REC_IN_RT is defined and g_u32RecordStart is 1, the actions of encoding sound data and writing to the microSD card will be executed when I²S receives data of each frame size. It should be noted that if the time to encode sound data and write to the microSD card is greater than the time of I²S receiving data of one frame size, the sound data received by I²S during this period will not be handled. This happens when the microSD card is underperforming. In the meanwhile, error flag, g_u32ErrorFlag, is no longer zero, and the UART message will inform user that some sounds have been lost. To get high recording quality, user needs to replace the microSD card with a higher performance or higher speed microSD card.

After recording is stopped or before recording is started, calling MP3Player() can play the MP3 file from the microSD card when PH.1 is 1.

```
int32_t main(void)
{
    int32_t i32Written;
    uint8_t *pu8Data;
    uint32_t u32PrintFlag = 1;

    TCHAR sd_path[] = { '0', ':', 0 };    /* SD drive started from 0 */

    /* Unlock protected registers */
    SYS_UnlockReg();

    /* Init System, peripheral clock and multi-function I/O */
    SYS_Init();

    /* Init SD */
    SD_Init();

    /* Init UART to 115200-8n1 for print message */
    UART_Open(UART0, 115200);

    printf("+-----------------------------------------------------------------------+\n");
    printf("|                  MP3 Recorder Sample with Audio Codec                 |\n");
    printf("+-----------------------------------------------------------------------+\n");
    printf(" Please insert a microSD card\n");
    printf(" Press BTN0 button to start recording and press BTN1 button to stop recording
           \n");
    printf(" Press BTN1 button can also play MP3 file from microSD card\n");
```

```
    /* Configure PH.0 and PH.1 as Output mode */
    GPIO_SetMode(PH, BIT0, GPIO_MODE_OUTPUT);
    GPIO_SetMode(PH, BIT1, GPIO_MODE_OUTPUT);

    /* Enable PH.1 interrupt by falling edge trigger */
    GPIO_EnableInt(PH, 1, GPIO_INT_FALLING);
    NVIC_EnableIRQ(GPH_IRQn);
    NVIC_SetPriority(GPH_IRQn, (1 << __NVIC_PRIO_BITS) - 2);

    /* Configure FATFS */
    SDH_Open_Disk(SDH0, CardDetect_From_GPIO);
    f_chdrive(sd_path);          /* Set default path */

    /* Init I2C2 to access audio codec */
    I2C2_Init();

    /* Select source from HXT(12MHz) */
    CLK_SetModuleClock(I2S0_MODULE, CLK_CLKSEL3_I2S0SEL_HXT, 0);

    while (1)
    {
        /* Read pin state of PH.0 and PH.1 */
        u32BTN0 = (PH->PIN & (1 << 0)) ? 0 : 1;
        u32BTN1 = (PH->PIN & (1 << 1)) ? 0 : 1;

        if (SD0.IsCardInsert == TRUE)
        {
#ifdef REC_IN_RT

            /* Inform user about microSD card usage */
            if ((u32PrintFlag == 1) && (g_u32ErrorFlag != 0))
            {
                printf("\n\nSome sounds have been lost due to poor microSD card
                        performance.\nPlease replace !!!\n\n");

                u32PrintFlag = 0;
                g_u32ErrorFlag = 0;
            }

            /* Encode sound data to MP3 format and store in microSD card */
            if (g_u32RecordStart == 1)
            {
                if ((g_u32WriteSDToggle == 0) && (g_u32BuffPos1 > 0) &&
                    (g_u32BuffPos1 == ((samples_per_pass * config.wave.channels) >> 1)))
                {
                    pu8Data = shine_encode_buffer_interleaved(s,
                              (int16_t *)(&g_au32PcmBuff1), (int *)&i32Written);

                    if (Write_MP3(i32Written, pu8Data, &config) != i32Written)
                    {
                        printf("shineenc: write error\n");
                    }

                    g_u32BuffPos1 = 0;
                    g_u32WriteSDToggle = 1;
                }
                else if ((g_u32WriteSDToggle == 1) && (g_u32BuffPos2 > 0) &&
                         (g_u32BuffPos2 == ((samples_per_pass * config.wave.channels)
                         >> 1)))
                {
                    pu8Data = shine_encode_buffer_interleaved(s,
                              (int16_t *)(&g_au32PcmBuff2), (int *)&i32Written);

                    if (Write_MP3(i32Written, pu8Data, &config) != i32Written)
                    {
```

```
                        printf("shineenc: write error\n");
                    }

                    g_u32BuffPos2 = 0;
                    g_u32WriteSDToggle = 0;
                }
            }

#endif

            if ((u32BTN0 == 1) && (g_u32RecordStart == 0) && (g_u32RecordDone == 0))
            {
#ifndef REC_IN_RT

                /* Clear HyperRAM */
                if (ClearHyperRAM(HYPER_RAM_MEM_MAP, HYPER_RAM_MEM_MAP + 0x800000) < 0)
                    return -1;

#endif

                /* Configure recording condition, init I2S, audio codec and encoder */
                Recorder_Init();

#ifdef REC_IN_RT

                printf("Start recording ... (online)\n");

#else

                printf("Start recording ... (offline)\n");

#endif

                /* Enable I2S RX function to receive sound data */
                I2S_ENABLE_RX(I2S0);

                g_u32RecordStart = 1;

#ifdef REC_IN_RT

                u32PrintFlag = 1;

                printf("Encode and write out the MP3 file ");

#endif
            }

            /* Play MP3 */
            if ((u32BTN1 == 1) && (g_u32RecordStart == 0))
            {
                MP3Player();
            }

            if (g_u32RecordDone == 1)
            {
                /* Disable I2S RX function */
                I2S_DISABLE_RX(I2S0);

#ifndef REC_IN_RT

                /* Encode sound data to MP3 format and store in microSD card */
                MP3Recorder();

#endif

                /* Close encoder */
```

```
                shine_close(s);

                f_close(&mp3FileObject);

                printf(" Done !\n\n");

                g_u32RecordStart = 0;
                g_u32RecordDone = 0;

                /* Play MP3 */
                MP3Player();
            }
        }
    }
}
```

In Recorder_Init() of mp3recorder.c, initialize I^{2}S interface, audio codec and encoder. Call f_open() to create a writable file in the microSD card with a name defined by MP3_FILE. User can configure recording sampling rate and bit rate by defining REC_SAMPLE_RATE and REC_BIT_RATE in config.h.

```
void Recorder_Init(void)
{
    FRESULT res;

    /* Open I2S0 interface and set to slave mode, audio format, I2S format */
    I2S_Open(I2S0, I2S_MODE_SLAVE, REC_SAMPLE_RATE, I2S_DATABIT_16, REC_FORMAT,
             I2S_FORMAT_I2S);

    NVIC_EnableIRQ(I2S0_IRQn);
    NVIC_SetPriority(I2S0_IRQn, 4);

    /* Set MCLK and enable MCLK */
    I2S_EnableMCLK(I2S0, 12000000);

    /* NAU8822 will store data in left channel */
    I2S_SET_MONO_RX_CHANNEL(I2S0, I2S_MONO_LEFT);

    /* Initialize NAU8822 codec */
    NAU8822_Setup();

    /* Configure NAU8822 to specific sample rate */
    NAU8822_ConfigSampleRate(REC_SAMPLE_RATE);

    /* Enable RX threshold level interrupt */
    I2S_EnableInt(I2S0, I2S_IEN_RXTHIEN_Msk);

    res = f_open(&mp3FileObject, MP3_FILE, FA_CREATE_ALWAYS | FA_WRITE);

    if (res != FR_OK)
    {
        printf("Open file error!\n");
        return;
    }

    set_defaults(&config);

    if (REC_FORMAT == I2S_MONO)
        config.wave.channels = 1;
    else
        config.wave.channels = 2;

    config.wave.samplerate = REC_SAMPLE_RATE;
    config.mpeg.bitr = REC_BIT_RATE;
```

```
    /* See if samplerate and bitrate are valid */
    if (shine_check_config(config.wave.samplerate, config.mpeg.bitr) < 0)
        printf("Unsupported samplerate/bitrate configuration.");

    /* Set to stereo mode if wave data is stereo, mono otherwise. */
    if (config.wave.channels > 1)
        config.mpeg.mode = STEREO;
    else
        config.mpeg.mode = MONO;

    /* Initiate encoder */
    s = shine_initialise(&config);

    check_config(&config);

    samples_per_pass = shine_samples_per_pass(s);
}
```

In I2S0_IRQHandler() of isr.c, if REC_IN_RT is defined, the received sound data for each frame size will be stored in two SRAM buffers in turn. When the received data size reaches one frame, always check g_u32WriteSDToggle to confirm whether the last encoded frame data has already been written to the microSD card. If g_u32WriteSDToggle is not as expected, g_u32ErrorFlag will count up. Conversely, if REC_IN_RT is not defined, the received sound data will be stored in HyperRAM. In the meanwhile, if the recording sound size reaches the allocated HyperRAM size, 7 Mbytes, the I^2S receive function will be stopped automatically.

```
void I2S0_IRQHandler(void)
{
    uint32_t u32Reg;
    uint32_t u32Len, i;
    int32_t *pi32BuffRx1, *pi32BuffRx2;

    u32Reg = I2S_GET_INT_FLAG(I2S0, I2S_STATUS0_RXTHIF_Msk);

    if (u32Reg & I2S_STATUS0_RXTHIF_Msk)
    {
        /* Read RX FIFO Level */
        u32Len = I2S_GET_RX_FIFO_LEVEL(I2S0);

#ifdef REC_IN_RT

        if ((g_u32BuffPos1 > 0) &&
            (g_u32BuffPos1 == ((samples_per_pass * config.wave.channels) >> 1)))
        {
            if (g_u32WriteSDToggle != 0)
            {
                g_u32ErrorFlag++;
                g_u32BuffPos1 = g_u32BuffPos2 = 0;
            }

            g_u32BuffToggle = 1;
        }

        if ((g_u32BuffPos2 > 0) &&
            (g_u32BuffPos2 == ((samples_per_pass * config.wave.channels) >> 1)))
        {
            if (g_u32WriteSDToggle != 1)
            {
                g_u32ErrorFlag++;
                g_u32BuffPos1 = g_u32BuffPos2 = 0;
            }

            g_u32BuffToggle = 0;
        }
```

```
        if (g_u32BuffToggle == 0)
        {
            pi32BuffRx1 = (int32_t *)&g_au32PcmBuff1[g_u32BuffPos1];

            for (i = 0; i < u32Len; i++)
            {
                pi32BuffRx1[i] = I2S_READ_RX_FIFO(I2S0);
            }

            g_u32BuffPos1 += u32Len;
        }
        else if (g_u32BuffToggle == 1)
        {
            pi32BuffRx2 = (int32_t *)&g_au32PcmBuff2[g_u32BuffPos2];

            for (i = 0; i < u32Len; i++)
            {
                pi32BuffRx2[i] = I2S_READ_RX_FIFO(I2S0);
            }

            g_u32BuffPos2 += u32Len;
        }

#else

        for (i = 0; i < u32Len; i++)
        {
            outp32(HYPER_RAM_MEM_MAP + g_u32BuffPos + 4 * i, I2S_READ_RX_FIFO(I2S0));
        }

        g_u32BuffPos += 4 * u32Len;

        if (g_u32BuffPos >= 0x700000)
        {
            /* Disable I2S RX function */
            I2S_DISABLE_RX(I2S0);
            g_u32RecordDone = 1;

            printf("The record data size reaches the allocated HyperRAM size of 7
                    Mbytes.\n");
        }

#endif
    }
}
```

In MP3Recorder() of mp3recorder.c, call shine_encode_buffer_interleaved() to encode data stored in HyperRAM, and call Write_MP3() to write the encoded data in MP3 format in MP3_FILE file.

```
void MP3Recorder(void)
{
    int32_t i32Written;
    uint8_t *pu8Data;
    uint32_t u32BuffPos = 0;
    uint32_t u32Len;

    u32Len = samples_per_pass * config.wave.channels;

    printf("Stop ...\n");
    printf("Encode and write out the MP3 file ");

    for (u32BuffPos = 0; u32BuffPos < g_u32BuffPos; u32BuffPos += 2 * u32Len)
    {
```

```
        pu8Data = shine_encode_buffer_interleaved(s,
                  (int16_t *)(HYPER_RAM_MEM_MAP + u32BuffPos), (int *)&i32Written);

        if (Write_MP3(i32Written, pu8Data, &config) != i32Written)
        {
            printf("shineenc: write error\n");
        }
    }

    g_u32BuffPos = 0;
}

int32_t Write_MP3(long bytes, void *buffer, void *config)
{
    f_write(&mp3FileObject, buffer, bytes, &ReturnSize);

    if (i32Cnt++ >= (bytes / 10))
    {
        i32Cnt = 0;
        printf(".");
    }

    return bytes;
}
```

3 Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M460 Series BSP CMSIS v3.00.001
- IDE version
  - Keil uVersion 5.28

3.2 Hardware Requirements

- Circuit components
  - M467HJHAE
  - T-Flash card (B8502A-13SB-HPA)
  - Audio codec (NAU88C22 with 9.7D2.54mm microphone and 2SJ3080-000111F SMD earphone)
  - HyperRAM (W956A8MBYA5I)
  - Button (SW-2P-SMD)
- Pin Connect

		V_{DD}	↔	V_{DD}		
		GND	↔	GND		
		SD0_DAT0 (PE2)	↔	DAT0		
		SD0_DAT1 (PE3)	↔	DAT1		
	M467	SD0_DAT2 (PE4)	↔	DAT2	T-Flash Card	
		SD0_DAT3 (PE5)	↔	DAT3		
		SD0_CLK (PE6)	↔	CLK		
		SD0_CMD (PE7)	↔	CMD		
		SD0_nCS (PD13)	↔	CD		

			V_{DD}	↔	V_{DD}	
			GND	↔	GND	
			I2C2_SDA (PD0)	↔	SDIO	
			I2C2_SCL (PD1)	↔	SCLK	
	M467		I2S0_BCLK (PI6)	↔	BCLK	Audio Codec
			I2S0_MCLK (PI7)	↔	MCLK	
			I2S0_DI (PI8)	↔	ADCOUT	
			I2S0_DO (PI9)	↔	DACIN	
			I2S0_LRCK (PI10)	↔	FS	
			V_{DD}	↔	V_{DD}	
			GND	↔	GND	
			HBI_nRESET (PJ2)	↔	RESET#	
			HBI_D3 (PJ3)	↔	DQ3	
			HBI_D2 (PJ4)	↔	DQ2	
			HBI_D1 (PJ5)	↔	DQ1	
			HBI_D0 (PJ6)	↔	DQ0	
	M467		HBI_nCS (PJ7)	↔	CS#	HyperRAM
			HBI_nCK (PH12)	↔	CK#	
			HBI_CK (PH13)	↔	CK	
			HBI_RWDS (PH14)	↔	RSDS	
			HBI_D4 (PH15)	↔	DQ4	
			HBI_D5 (PD7)	↔	DQ5	
			HBI_D6 (PD6)	↔	DQ6	
			HBI_D7 (PD5)	↔	DQ7	

		V_{DD}	↔	V_{DD}		
	M467	GND	↔	GND	Button	
		BTN0 (PH0)	↔	BTN_0		
		BTN1 (PH1)	↔	BTN_1		

Figure 3-1 Pin Connect

4 Directory Information

The directory structure is shown below.

EC_M460_MP3_Recorder_V1.00	
Library	Sample code header and source files
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
SampleCode	
ExampleCode	Source file of example code
ThirdParty	
FatFs	A generic FAT file system module for small embedded systems. Its official website is: http://elm-chan.org/fsw/ff/00index_e.html.
LibMAD	A MPEG audio decoder library that currently supports MPEG-1 and the MPEG-2 extension to lower sampling frequencies, as well as the de facto MPEG 2.5 format. All three audio layers — Layer I, Layer II, and Layer III (i.e. MP3) are fully implemented. This library is distributed under GPL license. Please contact Underbit Technologies (http://www.underbit.com/) for the commercial license.
LibShine	A blazing fast MP3 encoding library implemented in fixed-point arithmetic. Its official website is: https://github.com/toots/shine.

Figure 4-1 Directory Structure

5 Example Code Execution

1. Browse the ExampleCode folder as described in the Directory Information section and double-click *M460_MP3_Recorder.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6 Revision History

Date	Revision	Description
2022.05.24	1.00	1. Initially issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.