NUC505 MP3 Player from internal Flash

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	A MP3 player demo using internal codec to play MP3 file stored in internal SPI flash.
BSP Version	NUC505 Series BSP CMSIS V3.02.000
Hardware	NuTiny-EVB-NUC505 V1.6

1 Function Description

1.1 Introduction

This sample code uses NUC505 internal codec to play MP3 file stored in internal SPI flash. User needs to store MP3 file first and then modify MP3_address to set MP3 file storage address and modify MP3File_Size to set MP3 file size in config.h. Please note that the MP3 file cannot exceed the NUC505 memory location. This sample code will check MP3 file header to verify correctness of MP3 file.

MP3 file storage address range	MP3 file location parameter	MP3 file size parameter
0x11000~0x1F_FFFF	**MP3_address**	**MP3File_Size**

This sample code supports mono/stereo mode, data length is 16 bits and sampling rates is 8k, 11025, 12k, 16k, 22050, 24k, 32k, 44100 and 48k. It will automatically set the APLL value according to different sampling rates.

1.2 Demo Result

```
Relocate vector table in SRAM (0x20000000) for fast interrupt handling, size = 0xbc.
+------------------------------------------------------------+
|         Play MP3 from internal Flash Sample Code           |
+------------------------------------------------------------+
  Please store MP3 first in the internal Flash.
  MP3 storage location cannot exceed flash size
  NOTE: Need head-phone.

MP3 pre-stored in internal SPI Flash address from 0x00011000
168 byte(s) to the 1st frame.
====[MP3 Info]======
FileSize = 416 KB
SampleRate = 44100
BitRate = 320000
Channel = 2
====================

Configure Internal CODEC ...[OK]
Start playing ...
Exit MP3
Stop ...
```

2 Code Description

First, initialize the NUC505 in main.c, and put the code into the SRAM to increase its execution speed. Read the MP3 file according to the address set by MP3_address.

```
int32_t main(void)
{
    /* Init System, IP clock and multi-function I/O */
    SYS_Init();

    /* Init UART0 to 115200-8n1 for print message */
    UART0_Init();

    /* Relocate vector table in SRAM for fast interrupt handling. */
    {
        extern uint32_t __Vectors[];
        extern uint32_t __Vectors_Size[];
        extern uint32_t Image$$ER_VECTOR2$$ZI$$Base[];

        printf("Relocate vector table in SRAM (0x%08X) for fast interrupt handling, size =
0x%x.\n", Image$$ER_VECTOR2$$ZI$$Base, (unsigned int) __Vectors_Size);
        memcpy((void *) Image$$ER_VECTOR2$$ZI$$Base, (void *) __Vectors, (unsigned int)
__Vectors_Size);
        SCB->VTOR = (uint32_t) Image$$ER_VECTOR2$$ZI$$Base;
    }
    /* Init I2S */
    I2S_Init();

    printf("+------------------------------------------------------------+\n");
    printf("|         Play MP3 from internal Flash Sample Code           |\n");
    printf("+------------------------------------------------------------+\n");
    printf("  Please store MP3 first in the internal Flash.\n");
    printf("  MP3 storage location cannot exceed flash size\n");
    printf("  NOTE: Need head-phone. \n\n");

    /* Play MP3 from internal flash at MP3_address*/
    MP3Player(MP3_address);

    while (1);
}
```

In the MP3Player(MP3_address) function, the program will check MP3 file header to verify correctness of MP3 file, and to parse information about the MP3 file. And set the APLL according to the sampling rate.

```
    /* Parse MP3 header */
    i32Offset = MP3_ParseHeaderInfo(MP3StartAddr);

    if (i32Offset < 0)
        return;
    else {
        /* Check if the MP3 SamplRate frequency is 11025 multiples */
        if (audioInfo.mp3SampleRate % 11025) {
            /* APLL is 49152031Hz for 12k, 16k, 24k, 32k, 48k, and 96k sampling rate */
            CLK_SET_APLL(CLK_APLL_49152031);
            if (audioInfo.mp3SampleRate == 8000)
                CLK_SetModuleClock(I2S_MODULE, CLK_I2S_SRC_APLL, 1);
        }
```

After MP3 decode, use ping-pong buffer to play MP3 file. The process is as follows:

```
    /* Decode finished, try to copy pcm data to audio buffer */
        if (audioInfo.mp3Playing) {
            /* If next buffer is still full (playing), wait until it's empty */
            if (aPCMBuffer_Full[u8PCMBufferTargetIdx] == 1)
                while (aPCMBuffer_Full[u8PCMBufferTargetIdx]);
        } else {
            /* All buffers are full, wait */
            if ((aPCMBuffer_Full[0] == 1) && (aPCMBuffer_Full[1] == 1)) {
                /* Start to play */
                StartPlay();
                while (aPCMBuffer_Full[0]);
            }
        }
        for (i = 0; i < (int)Synth.pcm.length; i++) {
            /* Get the left/right samples */
            sampleL = Synth.pcm.samples[0][i];
            sampleR = Synth.pcm.samples[1][i];
            //FIXME mono application
            //if ( audioInfo.mp3Channel )
            //    sampleR = sampleL;

            /* Fill PCM data to I2S(PDMA) buffer */
```

```
            aPCMBuffer[u8PCMBufferTargetIdx][pcmbuf_idx++] = sampleR | (sampleL << 16);

            /* Need change buffer ? */
            if (pcmbuf_idx == PCM_BUFFER_SIZE) {
                NVIC_DisableIRQ(I2S_IRQn);
                /* Set buffer full flag */
                aPCMBuffer_Full[u8PCMBufferTargetIdx] = 1;
                NVIC_EnableIRQ(I2S_IRQn);
                u8PCMBufferTargetIdx ^= 1;
                pcmbuf_idx = 0;
                //printf("change to ==>%d ..\n", u8PCMBufferTargetIdx);
            }
        }
```

Clear buffer full flag when buffer transmit completed in nuc505_isr.c

```
void I2S_IRQHandler(void)
{
    uint32_t u32I2SIntFlag;

    /* Check interruput flag */
    u32I2SIntFlag = I2S_GET_INT_FLAG(I2S, I2S_STATUS_TDMATIF_Msk |
I2S_STATUS_TDMAEIF_Msk);

    if (u32I2SIntFlag & I2S_STATUS_TDMATIF_Msk) {
        I2S_CLR_INT_FLAG(I2S, I2S_STATUS_TDMATIF_Msk);

        /* Buffer 0 transfer completed, clear buffer full flag */
        aPCMBuffer_Full[0] = 0;
    } else if (u32I2SIntFlag & I2S_STATUS_TDMAEIF_Msk) {
        I2S_CLR_INT_FLAG(I2S, I2S_STATUS_TDMAEIF_Msk);
        /* Buffer 1 transfer completed, clear buffer full flag */
        aPCMBuffer_Full[1] = 0;
    }
}
```

3 Software and Hardware Environment

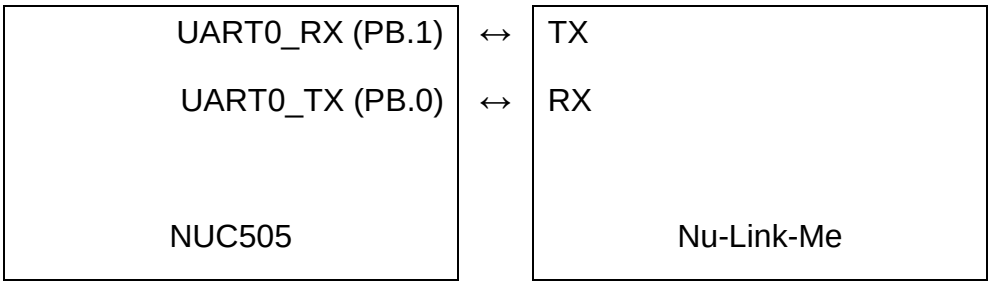
Software environment

- BSP version
  - NUC505 Series BSP CMSIS V3.02.000
- IDE version
  - Keil uVersion 5.26

Hardware environment

- Circuit components
  - NuTiny-EVB-NUC505 V1.6
  - 3.5 mm headphones
- Diagram

Use 3.5 mm headphones to connect to onboard socket(HP OUT). NUC505's UART0_RX (PB.1) and UART0_TX (PB.0) are connected to Nu-Link Me to print the message. Set COM port and Baud. The number of the COM Port can be found in the device manager "NuBridge Virtual Com Port (COMX)" and Baud is set to 115200.



4 Directory Information

- EC_NUC505_MP3PlayerSPIFlash_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 which 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

5 How to Execute Example Code

1. Browsing into sample code folder by Directory Information (section 4) and double click MP3PlayerSPIFlash.uvproj
2. Enter Keil compile mode
   a. Build
   b. Download
   c. Start/Stop debug session
3. Enter debug mode
   a. Run

6 Revision History

Date	Revision	Description
Jul. 03, 2019	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.