NGX LPC4330-Xplorer: Audio Recorder

MDK Version 5 Tutorial

AN272, March 2015, V 1.0.3

ARM KEIL Microcontroller Tools

The latest version of this document is here: www.keil.com/appnotes/docs/apnt_272.asp

Abstract

This tutorial runs you through the development of an audio recorder middleware application on the **NGX LPC4330-Xplorer** development board. The application samples audio data from an external codec IC and stores it onto a microSD card. To control the recorder, a modern web application is implemented using the HTTP-Server of the MDK Middleware.

Contents

Prerequisites

Development Tools

For this workshop you should install MDK-ARM and the following Software Packs (or latest):

- MDK-ARM Version 5.14 or later (www.keil.com/demo) with a MDK-Professional license.
- ARM::CMSIS 4.3.0
- Keil::MDK-Middleware 6.3.0
- Keil::Compiler Pack 1.0.0
- Keil::LPC4300_DFP 2.3.0
- Keil::Jansson 1.0.0

Setup the Development Tools for this workshop as described below. For more information refer to www.keil.com/mdk5/install

Install MDK:

- Install MDK-ARM Version 5.14 or later; use the default folder C:\Keil_v5 for the purposes of this tutorial.

Run the Pack Installer to download and install the following Software Packs:

- Select **Keil::XMC4300_DFP**. Click Install.
- Select **Keil::MDK-Middleware**. Click Update.
- Select **ARM::CMSIS**. Click Update.
- Select **Keil::ARM_Compiler**. Click Install.
- Double-click **Keil.Jansson.1.0.0.pack** that is part of the app note's ZIP file.

Activate the MDK-Professional license.

- In uVision use from the menu **File - License Management**. As a user of the evaluation version, you may use the button 

which gives you on-time access for 7 days to all features of MDK Professional. This option is only available once for evaluation users. If you need a longer period for evaluation, please contact your local distributor via www.keil.com/distis.
- Refer to www.keil.com/license for more information about the license activation.

Application Hardware

The following hardware is required to run the workshop application

- NGX 4330-Xplorer board
- Mini- and Micro-USB cable
- Ethernet cable to connect to a TCP/IP network
- Audio input (for example media player/smartphone) and output device (speaker/headphone)
- Stereo jack for connecting the audio input device
- microSD card for storing audio data

Debug Adapter

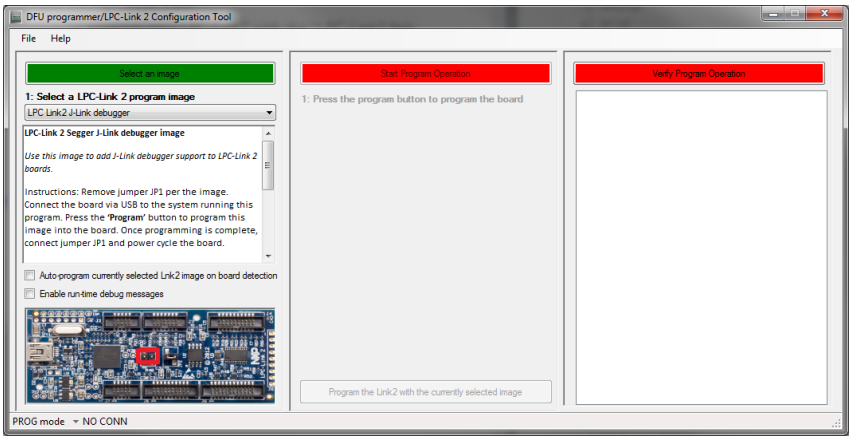
In this workshop we are using the **LPC-Link2 debug adapter** with J-Link firmware. You will need a Mini-USB cable to connect the LPC-Link2 with the PC that runs the development tools. The LPC-Link2 Debug Adapter should be configured as described below.

Download and Install J-Link Software & Documentation Pack for Windows

Visit www.segger.com/jlink-software.html and download the latest version of the *J-Link software and documentation pack for Windows*. The ZIP file contains an EXE file that needs to be installed on your computer before the configuration of the LPC-Link2 that is described in the next step.

Configure the LPC-Link2 as J-LINK debugger

Visit www.lpcware.com/lpclink2 to obtain the latest LPC-Link Configuration Tool. After installation, run the tool and follow the on-screen instructions to program your LPC-Link2 with the "LPC-Link2 J-Link debugger" firmware.



Introduction

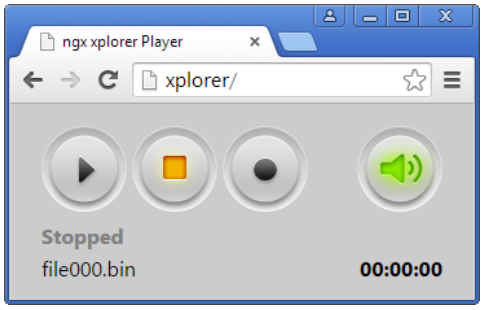
This workshop explains how to create a software framework for a microcontroller application using CMSIS software components and middleware. The application created in this workshop implements the following functions:

- Read the audio data stream from a CODEC
- Record: Store this audio data stream to a SD card using the FAT file system
- Play: Read the audio data file output the data stream to a CODEC
- Record and Play is controlled via a web interface using CGI and JavaScript/JSON

The application is implemented on a NXP LPC4330 dual-core microcontroller (contains a Cortex-M0 and a Cortex-M4 core) and consists therefore of two projects (one for each processor core).

- Project CM4 reads and outputs the audio data stream and interfaces to a CODEC.
- Project CM0 implements the file system and provides the user interface using a web server.

Sample Hardware Setup: The NGX board records audio data stream provided by an FM radio. A speaker is connected to the audio output for playback. An Ethernet cable connects the board to a LAN for web server access.



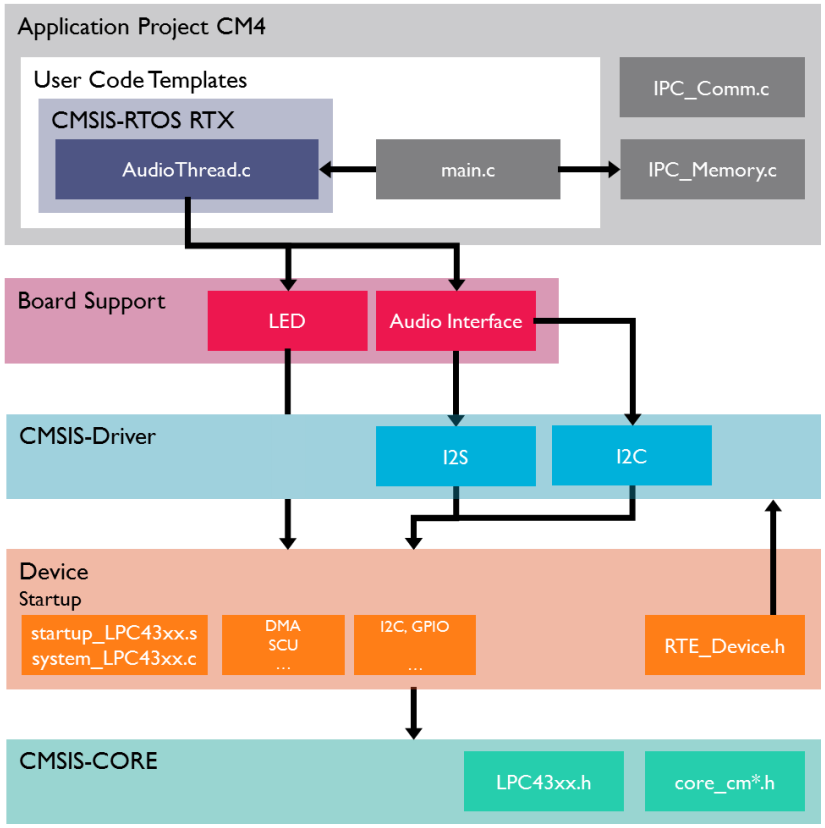
The NGX board is accessed using the URL "xplorer". A web interface enables you to record and playback audio.

Software Structure

Most parts of the application are created with software components and user code templates. The following diagrams describe the software structure of both projects and show the usage of the software components.

Project CM4

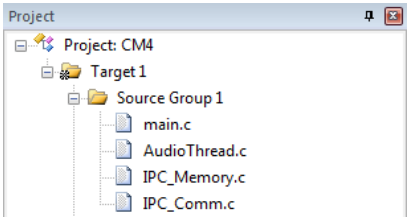
This part of the application reads and outputs the audio data stream. It uses CMSIS-Driver to access the external audio codec. The interface to the CODEC itself is provided by a software component from the Board Support group.



Source Files

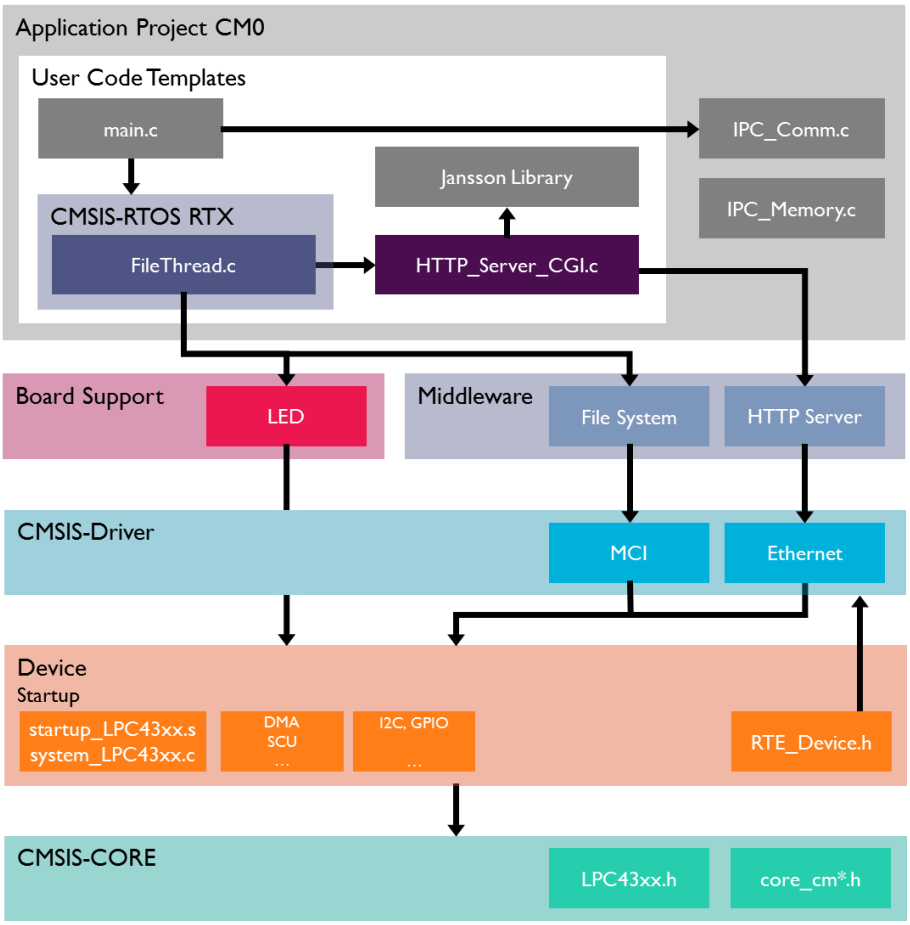
The Source Group 1 contains of four source code files:

- **main.c** is created from a main function user code template and contains mainly initialization functions
- **AudioThread.c** contains the actual thread used for the audio processing and the user callback functions
- The files **IPC_Memory.c** and **IPC_Comm.c** are used for an inter-process communication layer that implements the data communication between the two processor cores on the device.



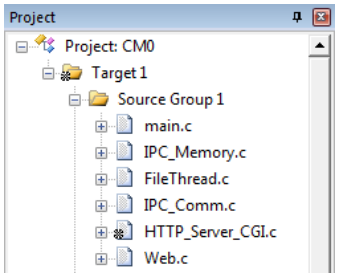
Project CM0

The application in the CM0 project uses several software components from the MDK-Professional Middleware and an open source software component to process JSON data.



Source Files

The Source Group 1 contains of four source code files:



- **main.c** is created from a main function user code template and contains mainly initialization functions
- **FileThread.c** contains the actual thread used for file operations
- **HTTP_Server_CGI.c** contains the functions that will react on CGI commands via the web interface. All JSON data will also be processed in functions from this file.
- The files **IPC_Memory.c** and **IPC_Comm.c** are used for an inter-process communication layer that implements the data communication between the two processor cores on the device.
- **Web.c** is the generated C source code containing all web resources required for the application.

Run the Application contained in the ZIP Archive

This application note is accompanied by a ZIP file that contains the µVision projects for each processor core. If you do not have the ZIP file available, check the latest version at www.keil.com/appnotes/docs/apnt_272.asp. Download the ZIP file and unpack it to any convenient folder on your computer as this contains the complete application with all relevant configuration settings.

Build the Project CM0

1. Start µVision and open the project file **Audio_CM0\CM0.uvprojx**
2. Click on **Build** or press **F7** to build the complete project

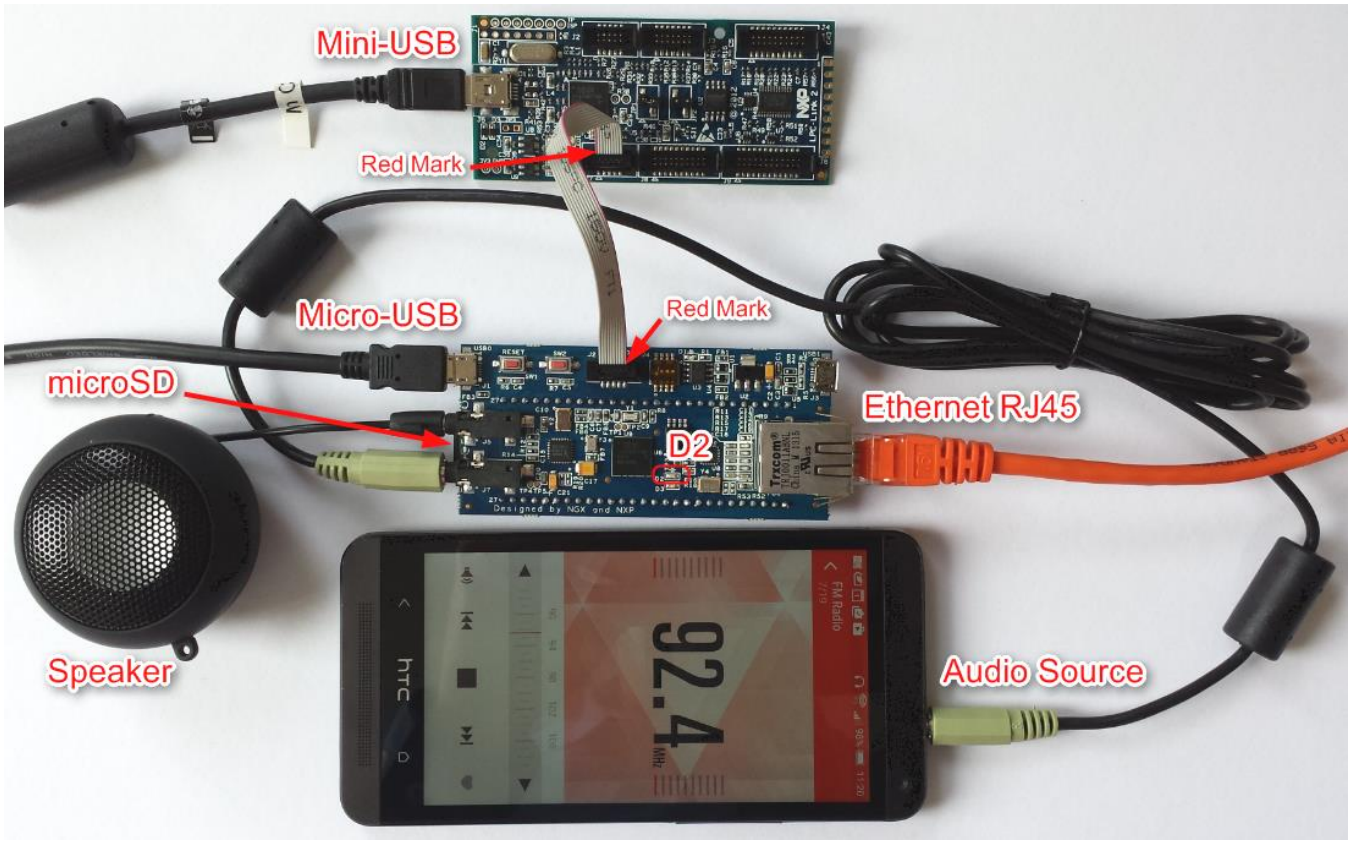
Build the Project CM4

3. Start a second instance of µVision and open the project file **Audio_CM4\CM4.uvprojx**
4. Click on **Build** or press **F7** to build the complete project

Hardware Setup

Before downloading the project to the target, make sure that the hardware is set up correctly:

- Connect the Mini-USB cable to the LPC-Link 2 and the Micro-USB cable to a USB connector on the LPC4330-Xplorer board
- Connect the two boards with the flat cable (make sure the red mark on the cable is on the right side of the connectors of each board)
- Attach a speaker/headphone to J5 to be able to listen to the audio
- Attach an audio source (mobile phone for example) to J7
- Connect the board with an Ethernet cable to your LAN. Make sure that you have a DHCP server in the LAN so that the Xplorer board will get an IP address assigned (LED D2 indicates this)
- Insert a FAT formatted microSD card into the card holder J9 at the bottom of the Xplorer board

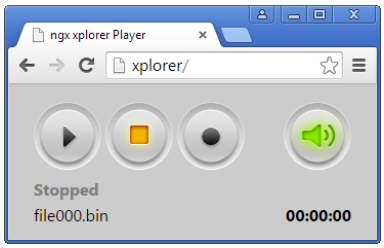


Download and run the Application

Use the µVision instance with project file **Audio_CM4\CM4.uvprojx** and use **Load**  to download the code to the SPI Flash-ROM on the Xplorer board. You might need to **Accept** a pop-up license dialog that describes the *"Terms of use of the LPCXpresso V2 J-Link firmware"*. If you discover problems with the download verify that the Debug Adapter is correctly configured with the J-Link firmware as described on page 3.

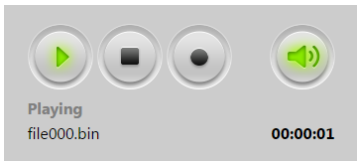
1. You might now start the application using **Debugger** or by pressing the reset button on the on the Xplorer board.

2. Ensure that the Xplorer board is connected to a LAN. Once the LED D2 is on, the board has obtained a valid IP address with a DHCP service. Then start a web browser on a computer that is connected to this LAN and enter http://xplorer to open the application's web interface. This opens control interface of our application that should be now in the stated **Stopped**.

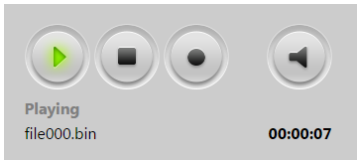


3. Start the audio source and press the **Record** button to record the data stream. The control interface will change and a flashing LED D3 on the Xplorer board will indicate that audio is being recorded. **Note:** If D3 is not flashing, this means that no audio data is recorded. The audio driver needs a while to initialize fully. Wait a few seconds and try again.

4. **Stop** the recording when finished, then press **Play** to output the recorded audio data.



5. While playing, you can **Mute** the music:



Solving Problems

If the application is not working, check the following:

- Are all cables connected correctly? Especially, check the connections of the audio source and the speaker.
- Is a microSD card inserted? Without the card, the application will not work. You will not be able to use the control interface (buttons will not react on clicks)
- Has the board obtained a valid IP address on the LAN network? Check LED D2 for a proper connection. If the LED is out although the Ethernet cable is connected, check your DHCP server for IP address assignment.

Analysis of the Operation using the Debugger

The LPC Link debug connection is preconfigured in both projects to address the correct core on the JTAG chain for debugging. The following steps guide you through interesting aspects of the application in the debug view.

1. Launch the M0 project in the debugger
2. Set a breakpoint to the `Init_FileThread`.

```
29
30  osKernelInitialize ();                    // initialize CMSIS-RTOS
31  Init_FileThread();
```

3. Open the System and Thread Viewer from Debug → OS Support. This will open the list of currently active threads and their current status:

System and Thread Viewer

Property	Value
System	

Item	Value
Tick Timer:	1.000 mSec
Round Robin Timeout:	
Default Thread Stack Size:	512
Thread Stack Overflow Check:	Yes
Thread Usage:	Available: 7, Used: 3 + o...

Threads

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	osTimerThread	High	Wait_MBX				cur: 7%, max: 7% [72/1024]
2	main	255	Running				cur: 3%, max: 14% [64/2048]
255	os_idle_demon	None	Ready				cur: 4%, max: 13% [280/2048]

4. Step over the call and see how the `FileThread` appears in the Thread Viewer.
5. Continue single stepping (Step-Over) until you stepped over `net_initialize();`

This will spawn another task with the default priority of **High** called `eth_thread`. It is an internal thread of the TCP/IP stack. There is a deterministic period of time that is available to finish reading or writing a block of samples to and from the SD-Card. To make sure these requests get fulfilled in time `FileThread` is on the RTX's highest priority **Realtime**.

Threads

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	osTimerThread	High	Wait_MBX				cur: 7%, max: 7% [72/1024]
2	main	255	Running				cur: 3%, max: 14% [140/2048]
3	FileThread	Realtime	Ready				cur: 3%, max: 13% [64/2048]
4	eth_thread	High	Ready				cur: 12%, max: 12% [64/512]
255	os_idle_demon	None	Ready				

6. Open the HTTP_Server_CGI.c and set a breakpoint in the `cgi_script` function to stop at the JSON-RPC calls.
7. Run the web application in your browser (http://xplorer) and press the record icon on the page.

```
101  case 'r' :
102  {
103      char* var;
104      json_error_t jerror;
105      json_t* jmethod, *jparams;
106      if (json_rpc_cmd == NULL) break;
107
```

8. Step until the variable **`var`** is assigned. You will see in the Call Stack + Locals windows which command was received from the decoding of the JSON data received.

main : 2	0x1401B850	Task
cgi_script	0x2000013C	unsigned int f(char *,c...
env	0x20006456 "r"	param - char *
buf	0x20006588 ""	param - char *
buflen	0x0000053E	param - unsigned int
pcgi	0x200085F8	param - unsigned int *
len	0x00000000	auto - unsigned int
jdata	0x20008A78	auto - struct json_t *
error_flag	0x00000000	auto - int
var	0x20008B40 "record"	auto - char *

9. Remove all breakpoints and set a new one to the `M0_M4Core_IRQHandler` in the module IPC_Comm.c.

```
 9  void M0_M4CORE_IRQHandler (void) {
10    LPC_CREG->M4TXEVENT = 0;
11    if (tid_FileThread != NULL)
12    {
13      //__SEV();
14      osSignalSet(tid_FileThread, 0x0001);
15      //M0_EventResp = 1;
16    }
17  }
```

The breakpoint will be hit when the M4 core sends a message to the M0 core. This happens when a new block of audio samples needs to be written or read from the SD-Card.

10. Start the debugger of the M4 project.
11. Run to initialization of the AudioThread.

```
46  void AudioThread (void const *argument) {
47    int i;
48    for (i=0; i<sizeof(Data); i++ ){
49      Data[0][i]=0xff;
50    }
51
52    Audio_Initialize(&Audio_Cbk);
53    Audio_SetDataFormat(AUDIO_STREAM_OUT, AUDIO_DATA_16_MONO);
54    Audio_SetFrequency (AUDIO_STREAM_OUT,48000);
55    Audio_SetMute (AUDIO_STREAM_OUT, 0, 0);
```

12. Single step into the control loop.

```
64      Audio_SetVolume(AUDIO_STREAM_OUT, 0, IPC_Memory.volume);
65      switch (IPC_Memory.M4_Command) {
66        case START_REC:
67          IPC_Memory.lastdata = (uint32_t*) Data[0];
68          IPC_Memory.nextdata = (uint32_t*) Data[1];
69          Audio_ReceiveData(IPC_Memory.nextdata, SAMP_NUM);
```

13. Open the Watch Window to read for the status of the system:

Watch 1

Name	Value	Type
IPC_Memory	0x2000FF00 &IPC_Memory	struct <untagged>
volume	0x37 '7'	unsigned char
state	0x00 STOP	enum (uchar)
M0_Command	0x00 CMD_CLR	enum (uchar)
M4_Command	0x00 CMD_CLR	enum (uchar)
M0_ready	0x00000001	unsigned int
M4_ready	0x00000000	unsigned int
nextdata	0x00000000	unsigned int *
lastdata	0x00000000	unsigned int *
time_in_sec	0x00000000	unsigned int

Call Stack + Locals | Watch 1

14. Remove all breakpoints and run the application

15. Start using the web application interface and watch how the status changes on the fly in the Watch 1 window. You can also set breakpoints into the individual state cases and get details about the execution by single stepping there.

For in detail information about dual core debugging on the LPC4300 series and the extended debug capabilities of ULINK*pro* refer to application note 241.

Setup of a Multi-Core Project

The LPC4330 is a multi-core microcontroller implementing an ARM Cortex-M4 and one ARM Cortex-M0 core. All cores have access to the complete memory map. The ARM Cortex-M4 is used as the main processor performing the audio data processing. The ARM Cortex-M0 core is used as a co-processor to off-load the ARM Cortex-M4 and runs the web server with the file system.

The M4 processor is used after reset as the top-level system controller. After power-up, the M0 core remains in reset until the reset is released by software running on the M4 core. Then, the cores can communicate with each other through shared memory space and interrupts.

Both cores will be set up as individual projects in MDK-ARM. There are several considerations to make that go beyond the standard project setup as described in the Getting Started manual. One important step is partitioning the memory in a way that every core gets its individual RAM and ROM areas:

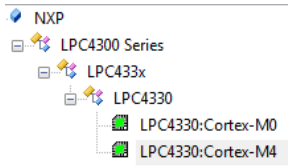
Address	Size		Memory Type
0x2000 FF00	0x0000 0100	IPC Memory	On-chip SRAM
0x2000 0000	0x0000 FF00	64 kB M0 RAM	
0x1401 0000	0x001F 0000	2 MB M0 ROM	SPIFI Flash
0x1400 0000	0x0001 0000	64 kB M4 ROM	
0x1000 0000	0x0002 0000	128 kB M4 RAM	On-chip SRAM

Note: This memory map is valid for this application. For a general memory map of the LPC4330 please consult the reference manual.

Flash programming should only be set up on one of the projects. The following shows the setup for the Audio application:

Create a Cortex-M4 Project

1. Start µVision.
2. Create a new µVision Project: Select Project/New µVision Project…
3. Create a folder for your project and give it a name.
4. Create a subfolder called CM4 for the Cortex-M4 core project. Enter this folder.
5. Enter a project name in the Name: box. Click OK. The Select Device… window opens.
6. Select NXP LPC4300:Cortex-M4 as shown here:



7. Click on OK. The Manage Run-Time Environment window opens.

Configure the Run-Time Environment

1. Make these selections:

   Board Support::LED API:LED

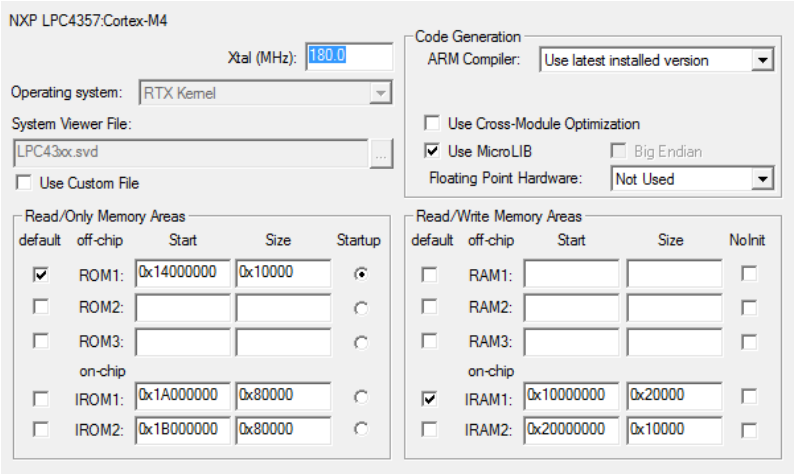
   CMSIS::RTOS(API)::Keil RTX

   Note there are some orange blocks. Click on Resolve and µVision will automatically select the required files.
2. All blocks will now be green. Select OK and they are added to your project and listed in the Project window.
3. In the Project window, expand the heading Target 1.
4. Right click on Source Group 1 and select Add New item to Group Source Group 1…
5. The Add New Item… window opens.
6. Select User Code Template and then expand the CMSIS heading.

7. Select Add "CMSIS-RTOS main function". Click on Add and it is added to your project under Source Group 1.

Configure Target Options

1. Click on the Options for Target icon  or press ALT-F7 and configure the Target Options as follows:



Include a Header File

1. In main.c, right-click on line 7 and select Insert "# include file".
2. Select `#include 'Board_LED.h'`. This line will be added to main.c.

Configure CMSIS-RTOS RTX

1. In the Project window, expand the CMSIS heading and double-click on **RTX_Conf_CM.C** to open it.
2. Click on the **Configuration Wizard** tab at the bottom of this window.
3. Click on **Expand All** button and make the following changes:
4. Set RTOS Kernel Timer input Clock frequency to **180 000 000** Hz. (180 MHz)
5. Click on File/Save All or  .

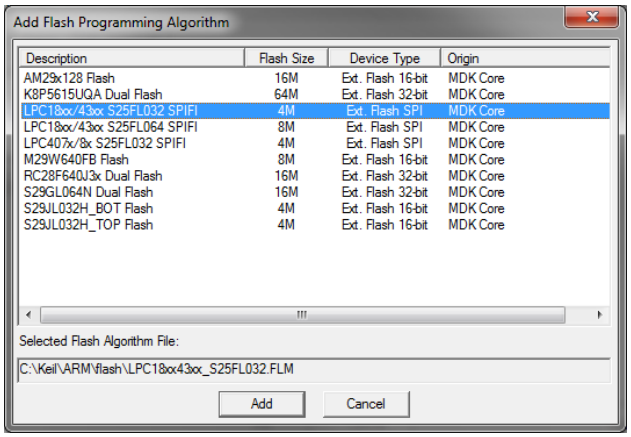
Add Code to main.c

1. Near line 9 in main.c add this line: `extern void hardware_init (void);`
2. Add these lines to main.c:

```
1. int main (void) {
2. osKernelInitialize(); // initialize CMSIS-RTOS
3.     hardware_init();
4.     LED_Initialize();
5.     osKernelStart(); // start thread execution
6.
7. while(1) {
8.         osDelay(500);
9.         LED_On(1);     //red LED
10.                osDelay(500);
11.                LED_Off(1);
12.       }
```

3. Click on File/Save All or  .

Configure the Flash Download



1. In the CM4 project open **Options for Target → Debug** and from the **Settings…** select **Flash Download**. There you can add the SPIFI flash algorithm as shown here:

2. Move over to the **Utilities** tab and create a FLASH.INI file. Use the LOAD command there to load the image file of the CM0 project together (see next section) with the CM4 image and flash both into the RAM using the CM4 project:

```
LOAD ..\Audio_CM0\Objects\CM0.axf INCREMENTAL
```

Audio Thread

The audio I/O is mainly event driven and most of its logic is controlled in a callback function. In addition to this, a thread is created to maintain status changes, for reconfiguration and to start or stop audio streams. The `AudioThread` is called periodically every 10ms and checks relevant status updates in the SharedMemory area.

File	Details	runs on Cortex-M4
AudioThread.c	Defines the CMSIS-RTOS thread `AudioThread` and the audio interface callback function.	

The `AudioThread` initializes the buffers and configures the audio in and out streams to 48 kHz/16-bit precision. Also note the initial input volume is reduced to 87 (of 100) to avoid distortion of higher level input signals. This value might be adapted also later in the application depending on the input level of your hardware.

```
1.  void AudioThread (void const *argument) {
2.  int i;
3.  for (i=0; i<sizeof(Data); i++ ){
4.    Data[0][i]=0xff;
5.  }
6.
7.  Audio_Initialize(&Audio_Cbk);
8.  Audio_SetDataFormat(AUDIO_STREAM_OUT, AUDIO_DATA_16_MONO);
9.  Audio_SetFrequency (AUDIO_STREAM_OUT,48000);
10. Audio_SetMute (AUDIO_STREAM_OUT, 0, 0);
11. Audio_SetDataFormat(AUDIO_STREAM_IN, AUDIO_DATA_16_MONO);
12. Audio_SetFrequency (AUDIO_STREAM_IN, 48000);
13. Audio_SetVolume(AUDIO_STREAM_IN, 0, 87);
14. Audio_SetMute (AUDIO_STREAM_IN, 0, 0);
```

The main loop of the thread is periodically called (every 1ms) to check for status updates. Commands for the M0 core are set in the `IPC_Memory.M0_Command` field, which gets notified using the `protected_sev()` call. This basically checks if the M0 core is ready to receive an interrupt and fires the SEV command.

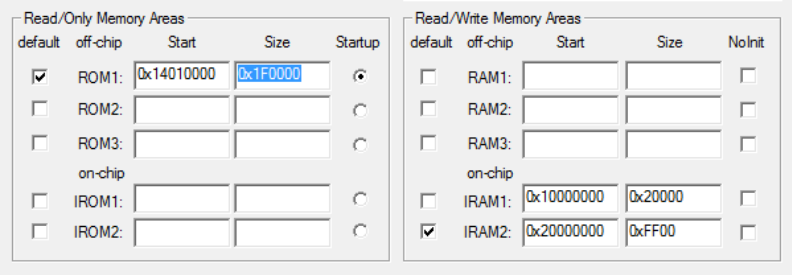
```
while (1) {
        Audio_SetVolume(AUDIO_STREAM_OUT, 0, IPC_Memory.volume);
switch (IPC_Memory.M4_Command) {
            case START_REC:
                IPC_Memory.lastdata = (uint32_t*) Data[0];
             IPC_Memory.nextdata = (uint32_t*) Data[1];
             Audio_ReceiveData(IPC_Memory.nextdata, SAMP_NUM);
                Audio_Start (AUDIO_STREAM_IN);
             IPC_Memory.M4_Command = CMD_CLR;
                      break;
              case STOP_CMD:
                      Audio_Stop(AUDIO_STREAM_IN);
                   Audio_Stop(AUDIO_STREAM_OUT);
                   IPC_Memory.M4_Command = CMD_CLR;
                      break;
                 case START_PLY:
      IPC_Memory.state = PLAY;
                  while(!IPC_Memory.M0_Command == CMD_CLR);
                      IPC_Memory.M0_Command = PLY_NEXT;
                   IPC_Memory.nextdata = (uint32_t*) Data[0];
                   protected_sev();
                   osDelay(20);
                   IPC_Memory.M4_Command = CMD_CLR;
                   IPC_Memory.lastdata = IPC_Memory.nextdata;
                   while(!IPC_Memory.M0_Command == CMD_CLR);
                   IPC_Memory.M0_Command = PLY_NEXT;
                   IPC_Memory.nextdata = (uint32_t*) Data[1];
                   protected_sev();
                   Audio_SendData(IPC_Memory.lastdata , SAMP_NUM);
                   Audio_Start(AUDIO_STREAM_OUT);
                      IPC_Memory.M4_Command = CMD_CLR;
                 break;
             }
      osDelay(1);
          }
```

The `Audio_Cbk()` is the callback function that is called by the audio driver whenever a reception or transmission is completed. In that case, new buffers are assigned to be played or recorded next and an according command to the M0 core is sent.

```
1. void Audio_Cbk (uint32_t event) {
2. uint32_t val;
3. if (event & ARM_SAI_EVENT_SEND_COMPLETE) {
4.              Audio_SendData(IPC_Memory.nextdata , SAMP_NUM);
5.          ptrSwap(&IPC_Memory.nextdata ,&IPC_Memory.lastdata);
6.              while(!IPC_Memory.M0_Command == CMD_CLR);
7.              IPC_Memory.M0_Command = PLY_NEXT;
8.              protected_sev ();
9.     }
10.
11.       if (event & ARM_SAI_EVENT_RECEIVE_COMPLETE) {
12.                ptrSwap(&IPC_Memory.nextdata ,&IPC_Memory.lastdata);
13.                Audio_ReceiveData(IPC_Memory.nextdata, SAMP_NUM);
14.                       while(!IPC_Memory.M0_Command == CMD_CLR);
15.            IPC_Memory.M0_Command = REC_NEXT;
16.       protected_sev ();
17.           }
18.       }
```

Create Cortex-M0 Project

1. Create a folder called CM0 at the same level than the CM4.
2. Rerun all steps from the CM4 project above but choose the LPC4330:Cortex-M0 as a device for the new project.
3. Configure the memory to these settings:



Release the Cortex-M0 Core from Reset

By default, the Cortex-M0 core of the LPC4330 is held in reset until the application in the Cortex-M4 is setting the reset vector and releasing the internal reset. The following two instructions are required at minimum to start the Cortex-M0 core. Use them at the beginning of your main loop in the CM4 project:

```
1.   LPC_CREG->M0APPMEMMAP = 0x14004000;
2.   LPC_RGU->RESET_CTRL1 = 0;
```

Using the RITIMER for CMSIS-RTOS RTX

The LPC4000 series does not feature a SysTick timer on the Cortex-M0 core. Therefore the CMSIS-RTOS RTX must be retargeted to use the RITIMER by adapting the functions in file RTX_Conf_CM.c of the CM0 project:

```
/*---------------------------- os_tick_init -----------------------------------*/
// Initialize alternative hardware timer as RTX kernel timer
// Return: IRQ number of the alternative hardware timer
int os_tick_init (void) {

LPC_CCU1->CLK_M4_RITIMER_CFG = (1UL << 0);

LPC_RITIMER->COMPVAL = OS_TRV; // Set match value
LPC_RITIMER->COUNTER = 0; // Set count value to 0
LPC_RITIMER->CTRL = (1UL << 3) | // Timer enable
(1UL << 2) | // Timer enable for debug
(1UL << 1) | // Timer enable clear on match
(1UL << 0); // Clear interrupt flag

return (M0_RITIMER_OR_WWDT_IRQn); /* Return IRQ number of timer (0..239) */
}

/*---------------------------- os_tick_val ------------------------------------*/

uint32_t os_tick_val (void) {
return (LPC_RITIMER->COUNTER);
}

/*---------------------------- os_tick_ovf ------------------------------------*/
// Get alternative hardware timer overflow flag
// Return: 1 - overflow, 0 - no overflow
uint32_t os_tick_ovf (void) {
if (LPC_RITIMER->CTRL & 1) {
return (1);
}
return (0);
}

/*---------------------------- os_tick_irqack ---------------------------------*/
// Acknowledge alternative hardware timer interrupt
void os_tick_irqack (void) {

LPC_RITIMER->CTRL |= (1UL << 0); // Clear interrupt flag
}
```

FAT File System on SD-Card

The application on the Cortex-M0 core takes ownership of the memory card interface (MCI) of the board and reads or writes blocks of audio samples on request of the Cortex-M4 core's `AudioThread`. The main state machine of the application is integrated into a thread called `FileThread`. It is triggered by the M4 event handler on buffer events (read new buffer to play; store buffer from recording). It can also be triggered from the HTTP Server CGI interface to reflect user commands from the web interface.

File	Details	runs on Cortex-M0
FileThread.c	Defines the CMSIS-RTOS thread FileThread.	

The `FileThread` checks the state of the system and if a state transmission command is pending. In that case, it will take care of opening, reading, writing or closing the file. The buffers that are written or read are set by the `AudioThread` that owns the buffer handling.

```
1. void FileThread (void const *argument) {
2. static FILE* file_handle;
3. unsigned int count = 0;
4.
5.     while (1) {
6. osSignalWait(0x0001, osWaitForever);
7.         switch(IPC_Memory.state) {
8.             case STOP:
9.             if (IPC_Memory.M0_Command == START_REC) {
10.                         file_handle = fopen(RECORDERFILENAME, "wb");
11.                         IPC_Memory.state = RECORD;
12.                         IPC_Memory.M0_Command = CMD_CLR;
13.                         IPC_Memory.M4_Command = START_REC;
14.                     }
15.                     if (IPC_Memory.M0_Command == START_PLY) {
16.                         file_handle = fopen(RECORDERFILENAME, "rb");
17.                         fseek(file_handle, 0, 0);
18.                         IPC_Memory.state = PLAY;
19.                         IPC_Memory.M0_Command = CMD_CLR;
20.                         IPC_Memory.M4_Command = START_PLY;
21.                     }
22.                     break;
23.                 case PLAY:
24.                     if (file_handle == NULL) __breakpoint(0);
25.                 if (IPC_Memory.M0_Command == STOP_CMD) {
26.                         stop_mark:
27.                      fclose(file_handle);
28.                         IPC_Memory.M0_Command = CMD_CLR;
29.                         IPC_Memory.M4_Command = STOP_CMD;
30.                         IPC_Memory.state = STOP;
31.                     }
32.                     if (IPC_Memory.M0_Command == PLY_NEXT) {
```

```
33.                             LED_On(0);
34.                             count = fread(IPC_Memory.nextdata, 2, 2048,
file_handle);
35.                             LED_Off(0);
36.                           IPC_Memory.M0_Command = CMD_CLR;
37.                           IPC_Memory.M4_Command = PLY_NEXT;
38.                             if (count < 2048) {
39.                                 IPC_Memory.M0_Command = STOP_CMD;
40.                             }
41.                       }
42.                         break;
43.                     case RECORD:
44.                         if (file_handle == NULL) __breakpoint(0);
45.             if (IPC_Memory.M0_Command == STOP_CMD) {
46.                          if (file_handle == NULL) __breakpoint(0);
47.       //                      fflush(file_handle);
48.                             fclose(file_handle);
49.                             IPC_Memory.M4_Command = STOP_CMD;
50.                             IPC_Memory.M0_Command = CMD_CLR;
51.                             IPC_Memory.state = STOP;
52.                         }
53.                         if (IPC_Memory.M0_Command == REC_NEXT) {
54.                          LED_On(0);
55.                             count=fwrite(IPC_Memory.lastdata,2,2048,file_handle);
56.                             IPC_Memory.M0_Command = CMD_CLR;
57.                             LED_Off(0);
58.                             if (count < 2048) {
59.                                 IPC_Memory.M0_Command = STOP_CMD;
60.                             }
61.                         }
62.
63.                         break;
64.                     default:
65.                         __breakpoint(0);
66.                         break;
67.                 }
68.       }
69.       }
```

CGI Interface and Web Application

The HTTP Server is implemented on the M0 core. This allows it to be tightly coupled to the File I/O for shorter response times to commands and also frees up the M4 for additional DSP tasks.

File	Details	runs on Cortex-M0
HTTP_Server_CGI.c	Is derived from the HTTP Server CGI Template. Includes JSON library to communicate with the web application.	

File	Details	runs on web client
index.htm	The initial page. Defines the layout of the web player and all UI elements for the JavaScript.	
Player.js	JavaScript that communicates with the webserver.	
rpc.cgx	JSON RPC (remote procedure call) processing.	
status.cgx	JSON data provider that returns the current status of the application to the web application using JSON format.	
***.png**	Image assets. Buttons for the player are images that represent the two possible states for every button. Depending on the values from status.cgx the correct buttons are displayed.	

The **HTTP_Server_CGI.c** is derived from a user code template and contains the full logic of all dynamically generated content that the webserver provides.

The `cgi_script()` function is called whenever a .cgi or .cgx file is requested from the webserver. It processes commands and fills up the reply buffer of the script file, line by line.

```
1.  uint32_t cgi_script (const char *env, char *buf, uint32_t buflen, uint32_t *pcgi) {
2.  ...
3.
4.  switch (env[0]) {
5.  ...
```

In case of the command byte `'s'`, a simple `sprintf` is used to generate a JSON formatted reply package when the status.cgx is requested by the web client:

```
1.  case 's' :
2.      sprintf(buf, "{\"state\":\"%d\", \"vol\":%d, \"err\":\"%d\"}" \
3.  ,IPC_Memory.state, IPC_Memory.volume, 0);
4.  len = strlen(buf);
5.  return (len| (1U<<30));
6.  break;
```

Jansson JSON library

More complex JSON formatted data should be processed by a library. Jansson is a very flexible and fast library to do so. It is published as open source and has been packaged into a component, which is available from the Pack Installer.

It is used in the CGI processing to handle the JSON RPC calls that allow control of the play and record tasks from the web interface. Detailed documentation for the full library is accessible from the component.

```
case 'r' :
{
      char* var;
      uint32_t i;
      json_error_t jerror;
      json_t* jmethod, *jparams;
      if (json_rpc_cmd == NULL) break;

      /* Load the JSON string from POST data */
      jdata = json_loads(json_rpc_cmd, 0, &jerror);
      if(jdata) {
            /* Parse Parameters from the JSON string */
            jmethod = json_object_get(jdata, "method");
            jparams = json_object_get(jdata, "params");
            var = (char*)json_string_value(jmethod);
            switch (var[0]) {
                case 'r':
                     ...
                     osSignalSet(tid_FileThread, 0x0001);
                break;

            }

            /* Acknowledge the JSON RPC call*/
            strcpy(buf, "{\"jsonrpc\": \"2.0\", \"result\": 1, \"id\":\"jrpc\"}");
            len = 44;
        }
json_decref( jdata );
json_decref( jmethod );
json_decref( jparams );
```

Please note that Jansson has some increased heap usage as most objects are allocated dynamically. A heap size of at least 0x800 bytes is recommended, but might be increased or decreased depending on the amount of JSON data that is processed.

Simplified Inter-Processor Communication Layer

Although the two cores of the LPC4330 run totally individual applications both can equally access all peripheral and memory resources. The audio application is only exchanging information about audio buffers and the current status of the player. In this limited scope, the simplest solution is a shared memory block at a fixed location. The SEV (Send Event) instruction is used to trigger an interrupt on the other core.

For a scalable IPC communication solution study the NXP application note AN1117. NXP provides two additional implementation templates for Inter-Processor Communication (IPC) between the two cores of the LPC4300 series.

- **Message Queue:** Two areas of shared memory are defined. The Command buffer is used exclusively by the master (M4) to send commands to the slave (M0). The Message buffer is used exclusively by the slave to send data to the master. An interrupt mechanism is used to signal to the core a message or command is available.
- **Mailbox:** An area in RAM is used by the sending processor to place a message for the receiving processor. The master uses an interrupt to signal to the slave that data has been placed in the mailbox(s).

File	Details runs on Cortex-M0 and Cortex-M4
IPC_Memory.c	Shared variables on an absolute memory location. Part of both projects (CM0 and CM4)
IPC_Memory.h	External declarations and type definitions for shared variables.
IPC_Comm.c	Contains functions to signal the other core and an interrupt handler to receive the signal.
IPC_Comm.h	External declarations for IPC communication functions.

```
1. typedef struct
2. {
3. uint8_t volume;
4.     state_t state;
5.     cmd_t M0_Command;
6.     cmd_t M4_Command;
7.     uint32_t M0_ready;
8.     uint32_t M4_ready;
9.     uint32_t* nextdata;
10.          uint32_t* lastdata;
11.          uint32_t time_in_sec;
12.      } IPC_Memory_t;
13.
14.      extern volatile IPC_Memory_t IPC_Memory;
```

The `IPC_Memory` structure is accessible from both cores, because it is compiled into both projects.
Using the `__attribute__ at` it is located at a known fixed location at the end of the SRAM:

```
1.   volatile IPC_Memory_t IPC_Memory __attribute__((at(0x2000FF00)));
```

For simplified debugging and also to avoid assigning invalid values to the system state or command variables enums have been declared. Since two differently compiled applications access the same enums it is strongly advised to not let the compiler assign the values, but declare them manually.

```
1. typedef enum {
2.     STOP = 0,
3.     PLAY = 1,
4.     RECORD = 2,
```

```
    NODISK = 4
} state_t;

typedef enum {
    CMD_CLR = 0,
          START_REC = 1,
          START_PLY = 2,
          STOP_CMD = 4,
          PLY_NEXT = 16,
          REC_NEXT = 32
      } cmd_t;
```

The initialization routine on the `IPC_Comm` of the M4 core also releases the M0 from reset:

```
#define M0_CODE_START 0x14010000

void M0APP_IRQHandler (void) {
LPC_CREG->M0APPTXEVENT = 0;
    LED_On(0);
}

void Init_IPC_Comm (void) {
     /* Stop CM0 core */
LPC_RGU->RESET_CTRL1 = (1 << 24);
    LPC_CREG->M0APPMEMMAP = M0_CODE_START;
    LPC_RGU->RESET_CTRL1 = 0;
    NVIC_EnableIRQ (M0APP_IRQn); /* Enable IRQ from the M0APP Core */
}

void protected_sev () {
    if (IPC_Memory.M0_ready == 1) {
        __sev();
    }
}
```

Conclusion

The application note introduces many different technologies that can be applied to real-life requirements today. It runs on three processing streams (Cortex-M0 core/Cortex-M4 core and web client) and shows the fundamentals to synchronize application control between them. Technologies like JSON-RPC will be used in embedded networking more often in the near future. The MDK Middleware HTTP server is already prepared to operate as a node in such a network.

There are many ways to extend the application or re-use parts of the code. The AudioThread for example can easily be extended to run additional DSP tasks using the CMSIS-DSP library. The utilization of the Cortex-M4 core is less than 10% as all the HMI (Human-Machine-Interface) related tasks are fulfilled by the Cortex-M0 core.

Appendix

Web Fundamentals for this Application

JavaScript: A script language that is interpretable by every modern web browser. JavaScript can be embedded in any html page and/or be loaded from a separate JavaScript module (*.js).

JavaScript mainly provides functions to dynamically alter HTML objects on a website that was loaded in the context.

jQuery: A very common JavaScript framework. The comprehensive API simplifies the work with JSON data, AJAX and many other common JavaScript tasks.

JSON: **J**ava**S**cript **O**bject **N**otation is a format for data representation in a string format. It is natively supported by JavaScript and has many advantages over data exchange using XML. Foremost the resource and bandwidth usage is lower as there is less overhead from the markup. On the embedded webserver a CGI script is used to format the data strings.

JSON RPC: **R**emote **P**rocedure **C**all protocol using JSON format to invoke procedures on a peer node (processed by the HTTP_Server_CGI.c in this application). Typically the transmission is using the HTTP protocol.

```
Example:

Request { "method": "echo", "params": ["Hello JSON-RPC"], "id": 1}

Reply   { "result": "Hello JSON-RPC", "error": null, "id": 1}
```

AJAX: **A**synchronous **J**avaScript **A**nd **X**ML describes a concept which uses JavaScript to render dynamic website content that is provided by a JSON or XML data source. Before AJAX was available updates to webpage content required a complete reload of the page.

Optimize the JavaScript ROM Usage

The JavaScript files have been [provided to you compressed using gzip. This is done to save program RO space. JavaScript files can be provided uncompressed so you can easily read them during development work and also compressed for final production.

All modern browsers support automatic decompression of zipped resources. Depending on the used libraries this can drastically reduce the required ROM space and loading time of pages. Using the project described in this application note, these are the size of the executable file: You can clearly see the RO-data in these cases is less than half of uncompressed.

Example without gzip-compression:

Program Size: Code=43636 RO-data=**117624** RW-data=244 ZI-data=28684

Example with gzip-compression:

Program Size: Code=43636 RO-data=**45320** RW-data=244 ZI-data=28684

How to use gzip Compression in your project:

Here are the steps needed to compress your JavaScript files using gzip. You do not have to do this in the application note since we already did this for you. You can un-gzip them to look inside if you like.

1. Download gzip Binaries from http://gnuwin32.sourceforge.net/packages/gzip.htm
2. Unzip the \bin\ folder to your projects web folder.
3. Zip jquery and smoothie libraries with the commands:
4. .\bin\gzip.exe -c jquery.min.js > jquery.min.js.gz
5. Patch the headers of html files that refer to the JavaScript libraries, like:

```
1. <html>
2.   <head>
3.     <script language="javascript" type="text/javascript" src="jquery.min.js.gz"></script>
```

Document Resources

Books:

1. ***NEW!*** **Getting Started with MDK 5:** Obtain this free book here: www.keil.com/mdk5/.
2. There is a good selection of books available on ARM processors. A good list of books on ARM processors is found at: www.arm.com/support/resources/arm-books/index.php
3. µVision contains a window titled Books. Many documents including data sheets are located there.
4. Or search for the Cortex-M processor you want on www.arm.com.
5. The Definitive Guide to the ARM Cortex-M0/M0+ by Joseph Yiu. Search the web for retailers.
6. The Definitive Guide to the ARM Cortex-M3/M4 by Joseph Yiu. Search the web for retailers.
7. Embedded Systems: Introduction to Arm Cortex-M Microcontrollers (3 volumes) by Jonathan Valvano.

Application Notes (www.keil.com/appnotes):

1. Using Infineon DAVE with µVision: www.keil.com/appnotes/files/apnt_258.pdf
2. Using Cortex-M3 and Cortex-M4 Fault Exceptions www.keil.com/appnotes/files/apnt209.pdf
3. CAN Primer using NXP LPC1700: www.keil.com/appnotes/files/apnt_247.pdf
4. CAN Primer using the STM32F Discovery Kit www.keil.com/appnotes/docs/apnt_236.asp
5. Segger emWin GUIBuilder with µVision™ www.keil.com/appnotes/files/apnt_234.pdf
6. Porting an mbed project to Keil MDK™ www.keil.com/appnotes/docs/apnt_207.asp
7. MDK-ARM™ Compiler Optimizations www.keil.com/appnotes/docs/apnt_202.asp
8. Using µVision with CodeSourcery GNU www.keil.com/appnotes/docs/apnt_199.asp
9. CMSIS-RTOS RTX in MDK 5 Eval Version: www.keil.com/cmsis/rtx
10. Barrier Instructions http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html
11. Lazy Stacking on the Cortex-M4 www.arm.com and search for DAI0298A
12. ***NEW!*** Cortex-M Processors for Beginners: http://community.arm.com/docs/DOC-8587
13. Cortex Debug Connectors: www.keil.com/coresight/coresight-connectors
14. Sending ITM printf to external Windows applications: www.keil.com/appnotes/docs/apnt_240.asp
15. Migrating fom Cortex-M4 to Cortex-M4 Processors: www.keil.com/appnotes/docs/apnt_270.asp

Useful ARM Websites

1. Cortex-M Learning Platform www.keil.com/learn
2. ARM Compiler Qualification Kit: www.keil.com/safety
3. CMSIS Standards: www.keil.com/cmsis
4. ARM and Keil Community Forums: www.keil.com/forum and http://community.arm.com/groups/tools/content
5. ARM University Program**:** www.arm.com/university
6. ARM Accredited Engineer Program: www.arm.com/aae
7. mbed: http://mbed.org

Keil Direct Sales In USA: sales.us@keil.com or 800-348-8051. **Outside the US:** sales.intl@keil.com
Keil Distributors: See www.keil.com/distis/ **DS-5 Direct Sales Worldwide:** orders@arm.com
Keil Technical Support in USA: support.us@keil.com or 800-348-8051. Outside the US: support.intl@keil.com.