FM4-176L-S6E2CC Overview

General information

Datasheets are available under docs/ in this repository.

Before writing your own initialization routines check all **utils** files.

1 Hardware overview

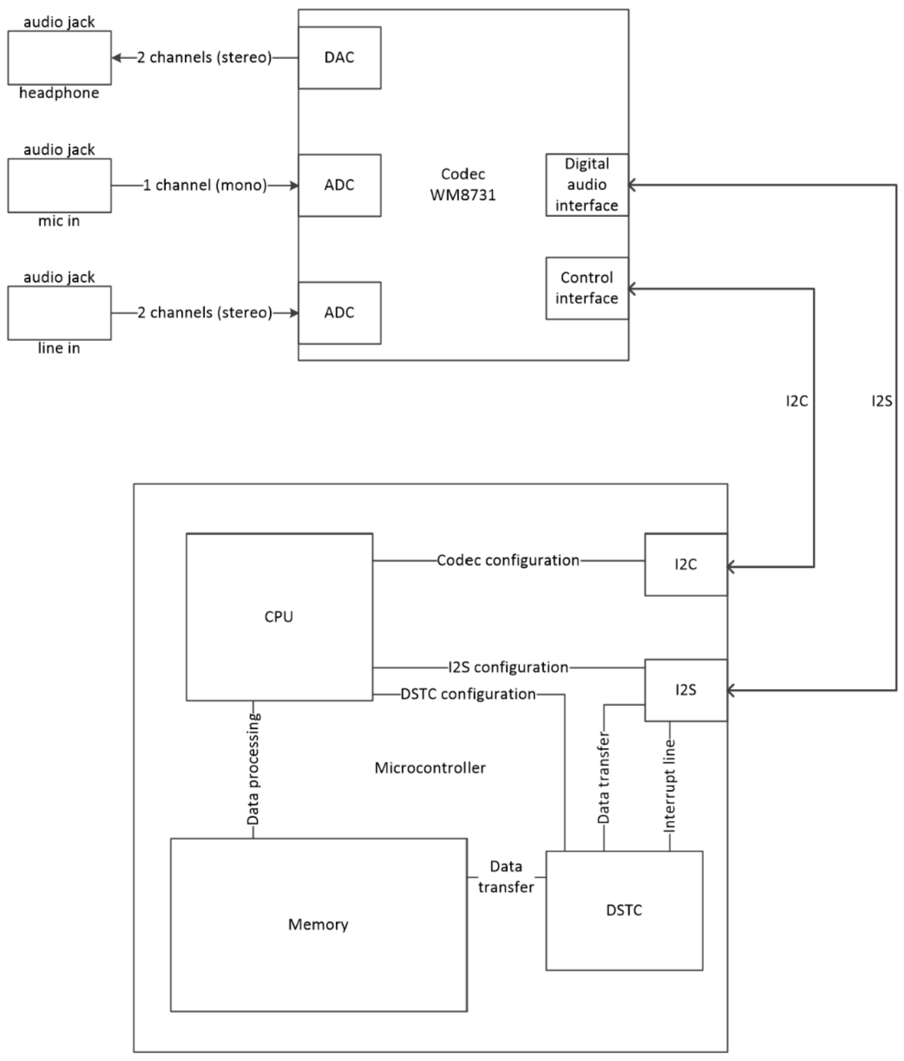

Figure 1-1: This image shows a very simple hardware overview.

1.1 Codec

A codec consists of ADCs and DACs. It is connected to two audio jacks for input, "mic in" and "line in", and one audio jack "headphone" for output. Important to note is that only one of the two inputs can be used during operation.

Also note that the "mic in" audio jack has only one audio channel (mono) connected to the codec and is also far more sensitive to low-level signals. When using the "line in" for a microphone, you will need to preamplify the signal. Otherwise, it will, if at all, be barely noticeable.

Each sample has a bit depth of 32-bit and has to be split into two 16-bit wide samples for each audio channel (stereo, left and right channel).

The codec is configured using I^{2}C (address is 0x1A, see schematic). The configuration is often encapsulated in high level functions by the board or microcontroller developer (see PDL). Transfers over I^{2}C are not always reliable at high frequencies. If a routine gets stuck waiting for a response, it may be sensible to reduce the clock frequency.

1.2 I^{2}S

I^{2}S is a serial interface protocol for transmitting digital audio. This protocol is used to transfer the sampled audio data from the codec to the microcontroller. For this the codec is supplied with a master clock (see Y3 on the schematic). The codec drives the bit clock, left-right-clock and the serial data line to transfer the data. The bit clock pulses once for each bit of data on the data lines and is therefore tied to the set sample rate.

When configuring the system for receive/transmit, each direction requires data to be transmitted, else an error may occur.

1.2.1 Receiving data

The digital audio interface of the codec transmits the samples according to the clock signals.

The transferred samples are stored inside of FIFO buffers integrated within the microcontroller. In a receive/transmit configuration two FIFOs for input and output can store 66 samples (1 sample = 32 bit) each.

More information about the specific hardware implementation can be found here:
docs/FM4-S6E2C_User_Manual/
Infineon-32_Bit_Microcontroller_FM4_Family_Peripheral_Manual_Communication_Macro_Part-UserManual-v04_00-EN

DMA/DSTC, Interrupts, and Polling can be used to perform internal transfers between the transmit and receive FIFOs and memory when the set number of samples have been transferred.

1.2.2 Transmitting data

To transmit data use DMA/DSTC, Interrupts or Polling to transfer data to the I^{2}S FIFO which is then send to the digital audio interface of the codec.

1.3 DMA/DSTC

Direct memory access (DMA) can be used to transfer data from a peripheral buffer to memory and back. The DMA controller can access memory independently of the CPU, which allows the CPU to perform other tasks. The CPU has to set up the DMA and receives an interrupt from the DMA controller when the task is done.
The DMA is configured using registers.

Descriptor System data Transfer Controller (DSTC) is similar to the DMA but is configured using a descriptor which has to be created first.

More information about both systems can be found here:
docs/FM4-S6E2C_User_Manual/
Infineon-FM4_Family_Peripheral_Manual_Main_Part_(TRM)-UserManual-v06_00-EN.pdf

If you want to use the DSTC to transfer data from the I^{2}S buffer to memory and back, you will need to initialize I^{2}S to start the DMA (no isr required at this point). Configuring the DSTC is done by writing a descriptor and initializing the DSTC with that descriptor and an isr for each transfer direction.

When calling *init_platform()* from ‘utils/platform.cpp‘ the DSTC will be fully set up. Calling *platform_start()* will start the clock for I^{2}S and the DSTC and therefore start transferring data.

1.4 Interrupts

Important: Interrupt service routines (isr) should only contain code that is absolutely necessary.

Interrupts halt the CPU within a very short period of time to process an event triggered task. An interrupt can be interrupted by an interrupt with a higher priority. After an interrupt task is completed, the CPU returns to its previous task.
When using interrupts to access memory that is used by your main loop as well, be sure to lock your resources with a mutex or semaphore.

Another method can be disabling the NVIC (nested vector interrupt controller), but this should be used with caution. Disabling the NVIC disables all interrupts until the NVIC is enabled again.

1.5 Polling

The CPU requests new data in a regular interval from a buffer. This requires the get-routine to support this method by returning a value that can be interpreted as "not ready". In order to not miss data, the request interval must be ensured at all times.

2 Software overview

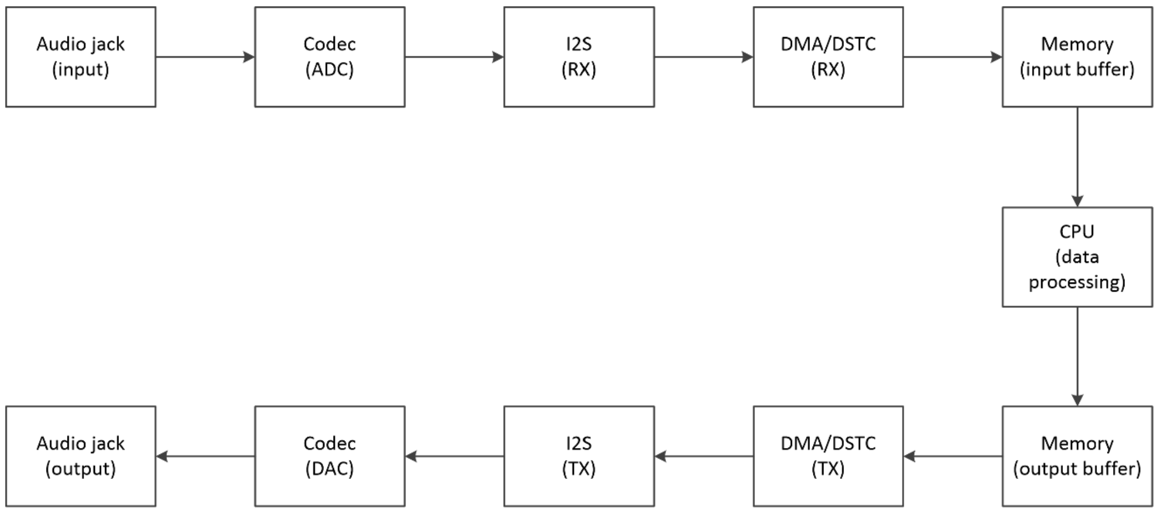

Figure 2-1: This image shows a very simple signal flow diagram.

2.1 PDL

The PDL is a software library for easier use of the hardware. This includes, for example, routines to initialize different peripherals of a microcontroller. This software library also provides routines for using these peripherals at runtime.

2.2 utils

The utils folder contains header and source files with useful routines which further simplify creating the data processing chain.