FM4-176L-S6E2CC Evaluation Board Overview

General information

Datasheets are available under docs/ in this repository.

Before writing your own hardware initialization routines check the code in the system/include/platform folder. Some hardware components are already supported by this code.

1 Hardware overview

This block diagram shows the microcontroller and codec connections which are important for applications which use audio I/O.

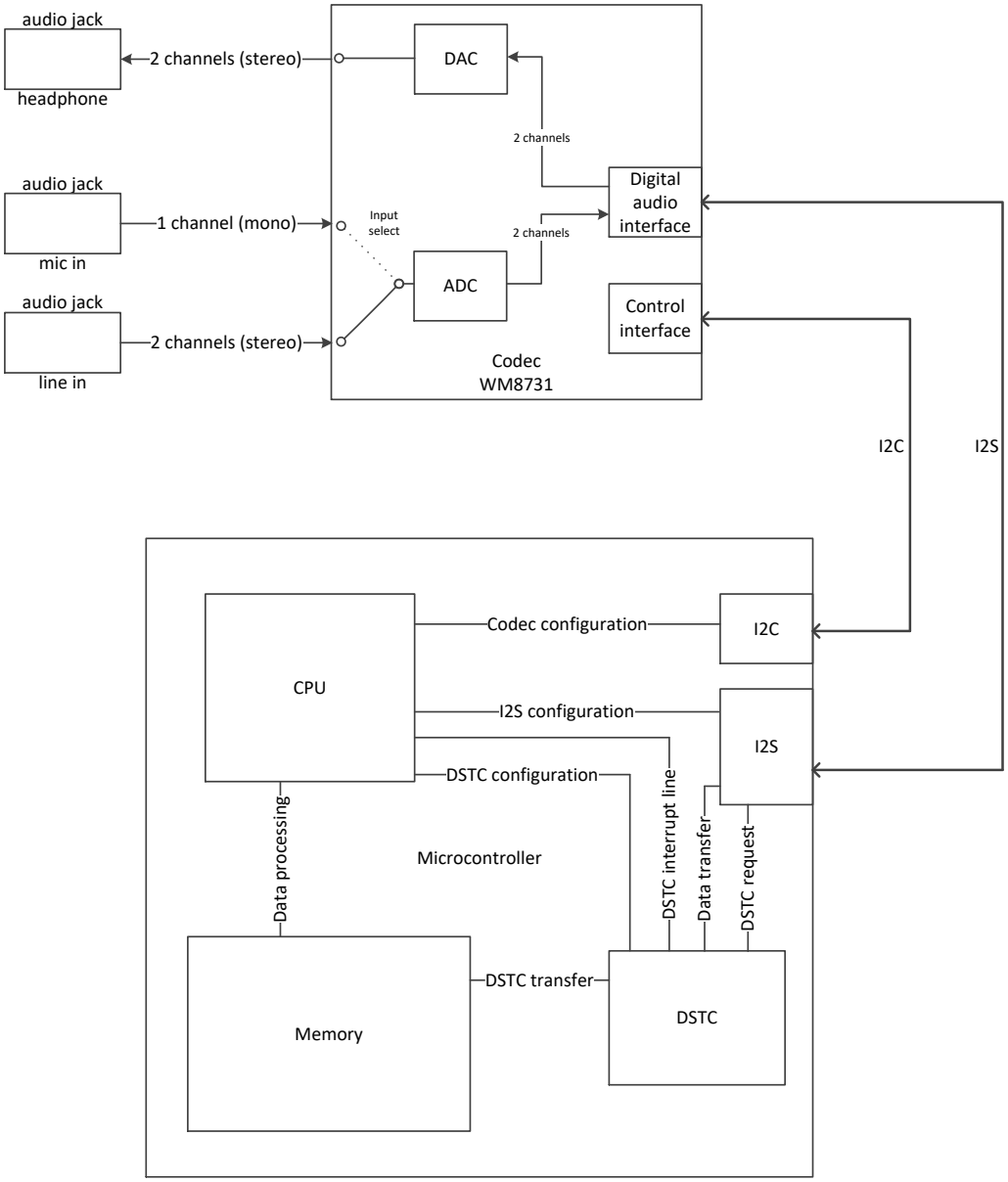

Figure 1-1: A simplified block diagram

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 routed to the ADC.

Also note that the "mic in" audio jack has only one audio channel (mono) connected to the codec, but will be duplicated to left and right ADC channels. A microphone outputs a low-level signal which

is compatible with the “mic in” input but not with the “line in” input. To connect a microphone to the “line in” input, you will need to pre-amplify the signal.

The bit resolution of ADC and DAC in this codec is 16 bit. The samples are transferred via I^2S using a 16 bit data format. Each sample frame on the I^2S bus consists of one left and one right channel sample. The I^2S peripheral stores each sample frame as a 32 bit word in its internal FIFO. The software has to split each word into two 16 bit samples.

The codec is configured using I^2C (address is 0x1A, see schematic). The configuration is often encapsulated in high level functions by the board or microcontroller developer (see PDL). The I^2C bus normally operates at 400 kHz. Tests have shown that codec register access does not work reliably on this evaluation board, it may be necessary to reduce the clock frequency to 100 kHz.

For further reading: https://en.wikipedia.org/wiki/I%C2%B2C

1.2 I^2S

I^2S 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 serial interface is supplied with the master clock (see Y3 on the schematic) of the codec. Data is transferred using a bit clock, left-right-clock and the serial data line. The bit clock pulses once for each bit of data on the data lines and is therefore tied to the sample rate.

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

For further reading: https://en.wikipedia.org/wiki/I%C2%B2S

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 I^2S serial interface of the microcontroller. In a receive/transmit configuration two FIFOs are set up for input and output. Each can store up to 66 words. One word consists of one left and one right channel sample (1 word = 1 sample frame = 32 bit).

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, or Polling can be used to perform internal transfers between the transmit and receive FIFOs and memory whenever data is available.

1.2.2 Transmitting data

DMA/DSTC, Interrupts or Polling can be used to transfer data to the I^2S FIFO as well. The I^2S serial interface will then send the data to the digital audio interface of the codec, where it will then be output using a DAC.

1.3 Polling

The CPU periodically tries to fetch new data 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, an appropriate polling interval must be ensured at all times.

1.4 Interrupts

Important: The complexity of Interrupt service routines (ISR) should be kept at a minimum.

Interrupts suspend the CPUs main task for a very short period of time to process an event-triggered task. An ISR can be interrupted by another ISR with a higher priority. After an ISR is completed, the CPU returns to its previous task.
When data is shared between ISR and main task, it needs to be protected to prevent “simultaneous” access, which can lead to corruption of data.

This can be done by disabling the corresponding interrupt vector in the NVIC (Nested Vectored Interrupt Controller).

In order to use interrupts to transfer data from the I^2S FIFO to memory and back, you will have to set up I^2S to trigger an interrupt and implement an ISR.
An example for such an ISR can be found at downloads/examples.

1.5 DMA/DSTC

1.5.1 DMA

Direct memory access (DMA) can be used to transfer data from a peripheral to memory or from memory to a peripheral. 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 transfer is done.
The DMA is configured using registers.

For further reading: https://en.wikipedia.org/wiki/Direct_memory_access

1.5.2 DSTC

Descriptor System data Transfer Controller (DSTC) is a specific DMA controller which is configured using an in-memory descriptor structure which has to be created by the programmer.

More information 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^2S peripheral to memory and back, you will need to initialize I^2S to start the DSTC (no I^2S ISR is required in this case). Configuring the DSTC is

done by writing a descriptor and initializing the DSTC with that descriptor and an ISR for each transfer direction.

An example for a DSTC ISR can be found at downloads/examples.
A ready-to-use descriptor is implemented within the *DstcInit()* routine from the Platform files.
Note: This descriptor only supports block sizes of the power of two (.., 16, 32, 64, ..).

2 Software overview

2.1 Signal flow

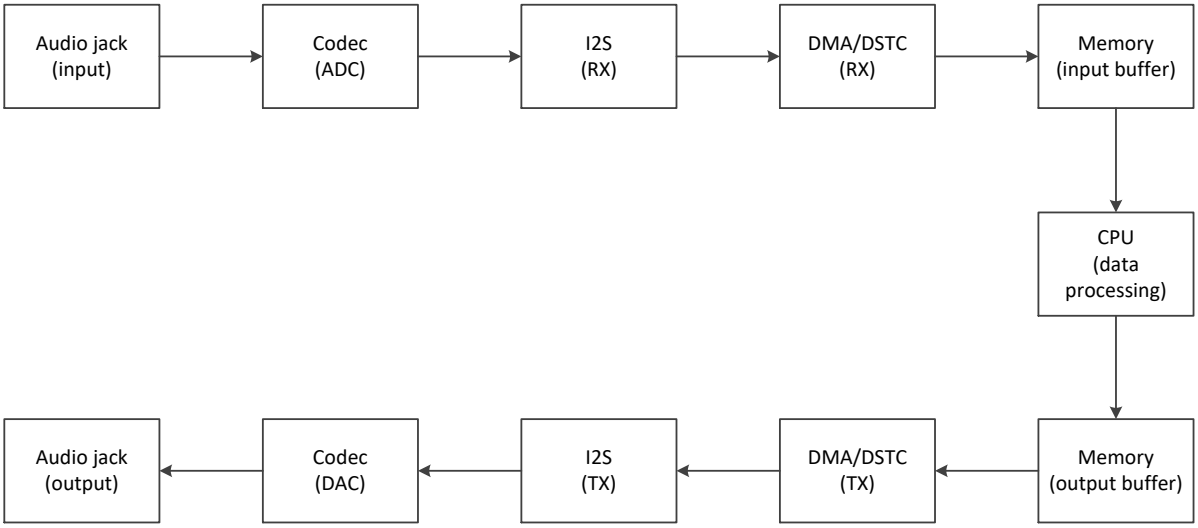

Figure 2-1: Signal flow diagram

2.2 PDL

The PDL is a software library by the chip designers for easier use of the hardware. It includes various routines for initialization and operation of peripherals.

2.3 Platform code

The system/include/platform folder contains header and source files with useful routines which further simplify creating the data processing chain.