DIGITAL SIGNAL PROCESSING

Using the ARM® Cortex®-M4

Donald S. Reay

WILEY

DIGITAL SIGNAL PROCESSING USING THE ARM® CORTEX®-M4

DIGITAL SIGNAL PROCESSING USING THE ARM® CORTEX®-M4

DONALD S. REAY
Heriot-Watt University

WILEY

Library of Congress Cataloging-in-Publication Data:

Reay, Donald (Donald S.), author.
Digital signal processing using the ARM Cortex-M4 / Donald Reay.
pages cm
Includes bibliographical references and index.
ISBN 978-1-118-85904-9 (pbk.)
1. Signal processing–Digital techniques. 2. ARM microprocessors. I. Title.
TK5102.9.R4326 2015
621.382′2–dc23

2015024771

Typeset in 10/12pt TimesLTStd by SPi Global, Chennai, India

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

1 2016

To Reiko

CONTENTS

PREFACE

This book continues the series started in 1990 by Rulph Chassaing and Darrell Horning's *Digital Signal Processing with the TMS320C25*, which tracked the development of successive generations of digital signal processors by Texas Instruments. More specifically, each book in the series up until now has complemented a different inexpensive DSP development kit promoted for teaching purposes by the Texas Instruments University Program. A consistent theme in the books has been the provision of a large number of simple example programs illustrating DSP concepts in real time, in an electrical engineering laboratory setting.

It was Rulph Chassaing's belief, and this author continues to believe, that hands-on teaching of DSP, using hardware development kits and laboratory test equipment to process analog audio frequency signals, is a valuable and effective way of reinforcing the theory taught in lectures.

The contents of the books, insofar as they concern fundamental concepts of digital signal processing such as analog-to-digital and digital-to-analog conversion, finite impulse response (FIR) and infinite impulse response (IIR) filtering, the Fourier transform, and adaptive filtering, have changed little. Every academic year brings another cohort of students wanting to study this material. However, each book has featured a different DSP development kit.

In 2013, Robert Owen suggested to me that hands-on DSP teaching could be implemented using an inexpensive ARM® Cortex-M4® microcontroller. I pointed out that a Texas Instruments C674x processor was very significantly more computationally powerful than an ARM Cortex-M4. But I also went ahead and purchased a Texas Instruments Stellaris LaunchPad. I constructed an audio interface using a Wolfson WM8731 codec and successfully ported the program examples from my previous book to that hardware platform.

This book is aimed at senior undergraduate and postgraduate electrical engineering students who have some knowledge of C programming and linear systems theory, but it is intended, and hoped, that it may serve as a useful resource for anyone involved in teaching or learning DSP and as a starting point for teaching or learning more.

I am grateful to Robert Owen for first making me aware of the ARM Cortex-M4; to Khaled Benkrid at the ARM University Program and to the Royal Academy of Engineering for making possible a six-month Industrial Secondment to ARM during which teaching materials for the STM32f01 platform were developed; to Gordon McLeod and Scott Hendry at Wolfson Microelectronics for their help in getting the Wolfson Pi audio card to work with the STM32f01 Discovery; to Sean Hong, Karthik Shivashankar, and Robert Iannello at ARM for all their help; to Joan Teixidor Buixeda for helping to debug the program examples; to Cathy Wicks at the TI University Program and Hieu Duong at CircuitCo for developing the audio booster pack; and to Kari Capone and Brett Kurzman at Wiley for their patience. But above all, I thank Rulph Chassaing for inspiring me to get involved in teaching hands-on DSP.

Donald S. Reay

Edinburgh
2015

1

ARM® CORTEX®-M4 DEVELOPMENT SYSTEMS

1.1 INTRODUCTION

Traditionally, real-time digital signal processing (DSP) has been implemented using specialized and relatively expensive hardware, for example, digital signal processors or field-programmable gate arrays (FPGAs). The ARM® Cortex®-M4 processor makes it possible to process audio in real time (for teaching purposes, at least) using significantly less expensive, and simpler, microcontrollers.

The ARM Cortex-M4 is a 32-bit microcontroller. Essentially, it is an ARM Cortex-M3 microcontroller that has been enhanced by the addition of DSP and single instruction multiple data (SIMD) instructions and (optionally) a hardware floating-point unit (FPU). Although its computational power is a fraction of that of a floating-point digital signal processor, for example, the Texas Instruments C674x, it is quite capable of implementing DSP algorithms, for example, FIR and IIR filters and fast Fourier transforms for audio signals in real-time.

A number of semiconductor manufacturers have developed microcontrollers that are based on the ARM Cortex-M4 processor and that incorporate proprietary peripheral interfaces and other IP blocks. Many of these semiconductor manufacturers make available very-low-cost evaluation boards for their ARM Cortex-M4 microcontrollers. Implementing real-time audio frequency example programs on these platforms, rather than on more conventional DSP development kits, constitutes a reduction of an order of magnitude in the hardware cost of implementing hands-on

DSP teaching. For the first time, students might realistically be expected to own a hardware platform that is useful not only for general microcontroller/microprocessor programming and interfacing activities but also for implementation of real-time DSP.

1.1.1 Audio Interfaces

At the time that the program examples presented in this book were being developed, there were no commercially available low-cost ARM Cortex-M4 development boards that incorporated high-quality audio input and output. The STMicroelectronics STM32F407 Discovery board features a high-quality audio digital-to-analog converter (DAC) but not a corresponding analog-to-digital converter (ADC). Many ARM Cortex-M4 devices, including both the STMicroelectronics STM32F407 and the Texas Instruments TM4C123, feature multichannel instrumentation-quality ADCs. But without additional external circuitry, these are not suitable for the applications discussed in this book.

The examples in this book require the addition (to an inexpensive ARM Cortex-M4 development board) of an (inexpensive) audio interface.

In the case of the STMicroelectronics STM32F407 Discovery board and of the Texas Instruments TM4C123 LaunchPad, compatible and inexpensive audio interfaces are provided by the Wolfson Pi audio card and the CircuitCo audio booster pack, respectively. The low-level interfacing details and the precise performance characteristics and extra features of the two audio interfaces are subtly different. However, each facilitates the input and output of high-quality audio signals to and from an ARM Cortex-M4 processor on which DSP algorithms may be implemented.

Almost all of the program examples presented in the subsequent chapters of this book are provided, in only very slightly different form, for both the STM32F407 Discovery and the TM4C123 LaunchPad, on the partner website `http://www.wiley.com/go/Reay/ARMcortexM4`.

However, in most cases, program examples are described in detail, and program listings are presented, only for one or other hardware platform. Notable exceptions are that, in Chapter 2, low-level i/o mechanisms (implemented slightly differently in the two devices) are described in detail for both hardware platforms and that a handful of example programs use features unique to one or other processor/audio interface.

This book does not describe the internal architecture or features of the ARM Cortex-M4 processor in detail. An excellent text on that subject, including details of its DSP-related capabilities, is *The Definitive Guide to ARM® Cortex®-M3 and Cortex®-M4 Processors* by Yiu [1].

1.1.2 Texas Instruments TM4C123 LaunchPad and STM32F407 Discovery Development Kits

The Texas Instruments and STMicroelectronics ARM Cortex-M4 processor boards used in this book are shown in Figures 1.1 and 1.2. The program examples presented in this book assume the use of the *Keil MDK-ARM* development environment, which is compatible with both development kits. An alternative development environment,

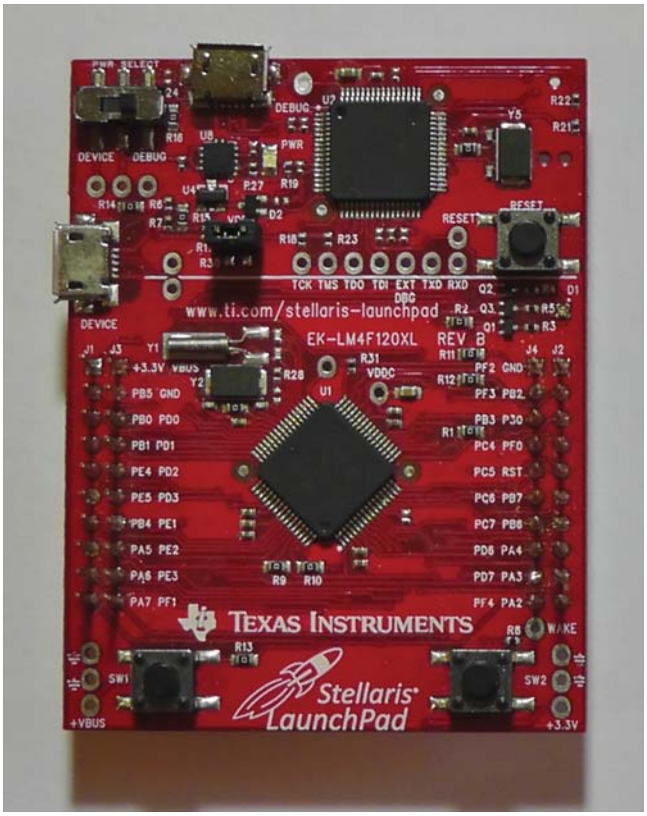

Figure 1.1 Texas Instruments TM4C123 LaunchPad.

Texas Instruments' *Code Composer Studio*, is available for the TM4C123 Launch-Pad and the program examples have been tested using this. Versions of the program examples compatible with *Code Composer Studio version 6* are provided on the partner website `http://www.wiley.com/go/Reay/ARMcortexM4`.

The CircuitCo audio booster pack (for the TM4C123 LaunchPad) and the Wolfson Pi audio card (for the STM32F407 Discovery) are shown in Figures 1.3 and 1.4. The audio booster pack and the launchpad plug together, whereas the Wolfson audio card, which was designed for use with a Raspberry Pi computer, must be connected to the Discovery using a custom ribbon cable (available from distributor Farnell).

Rather than presenting detailed instructions here that may be obsolete as soon as the next version of *MDK-ARM* is released, the reader is directed to the "getting started" guide at the partner website `http://www.wiley.com/go/Reay/ARMcortexM4` and before progressing to the next chapter of this book will need to install *MDK-ARM*, including the "packs" appropriate to the hardware platform being used and including the CMSIS DSP library, download the program examples from the website, and become familiar with how to open a project in *MDK-ARM*, add and

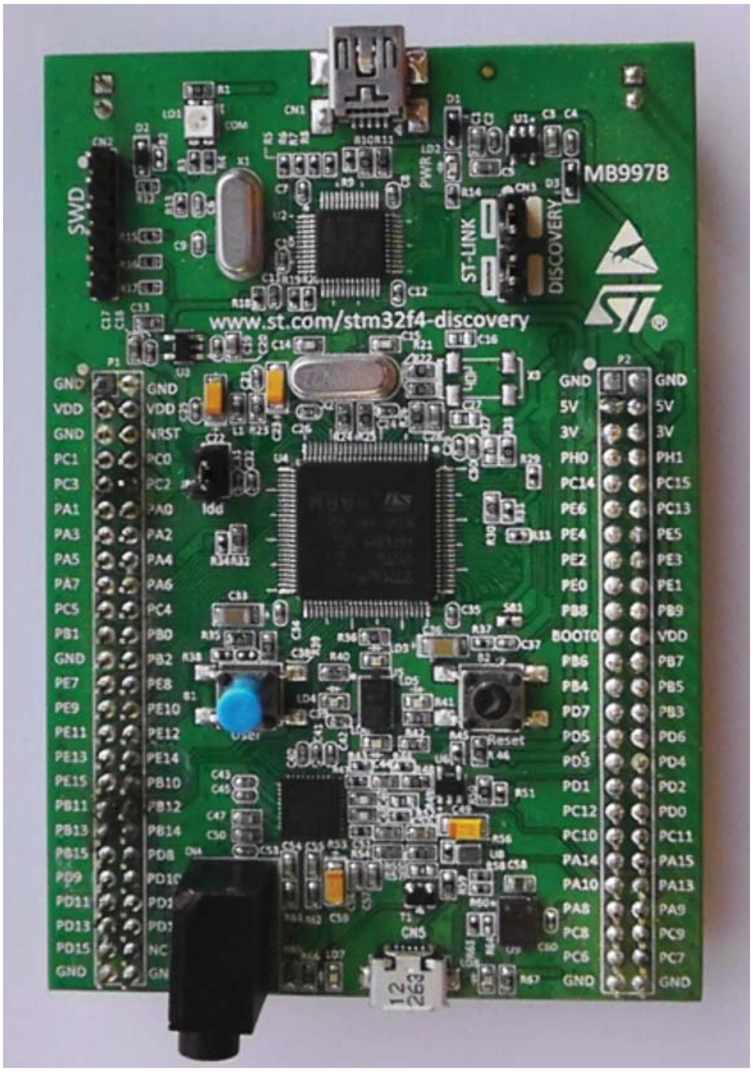

Figure 1.2 STMicroelectronics STM32F407 Discovery.

remove files from a project, build a project, start and stop a debug session, and run and halt a program running on the ARM Cortex-M4 processor.

Some of the example programs implement DSP algorithms straightforwardly, and with a view to transparency and understandability rather than computational efficiency or elegance. In several cases, ARM's CMSIS DSP library functions are used. These are available for both the STMicroelectronics and Texas Instruments processors as part of the *MDK-ARM* development environment. In appropriate circumstances, these library functions are particularly computationally efficient.

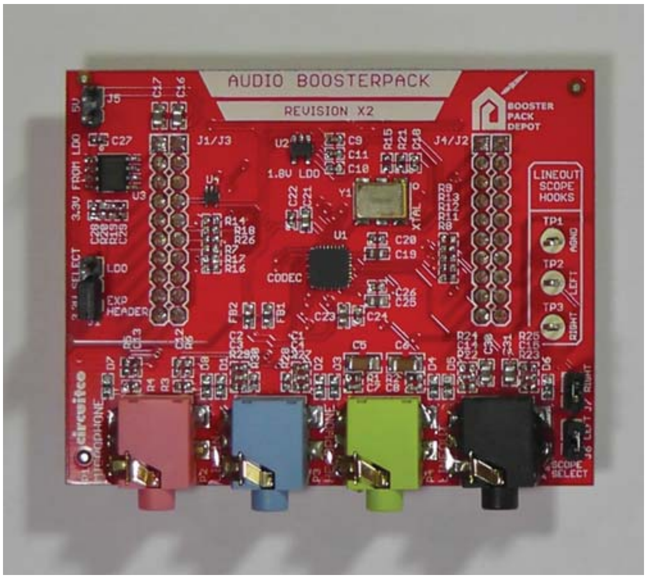

Figure 1.3 AIC3104 audio booster pack.

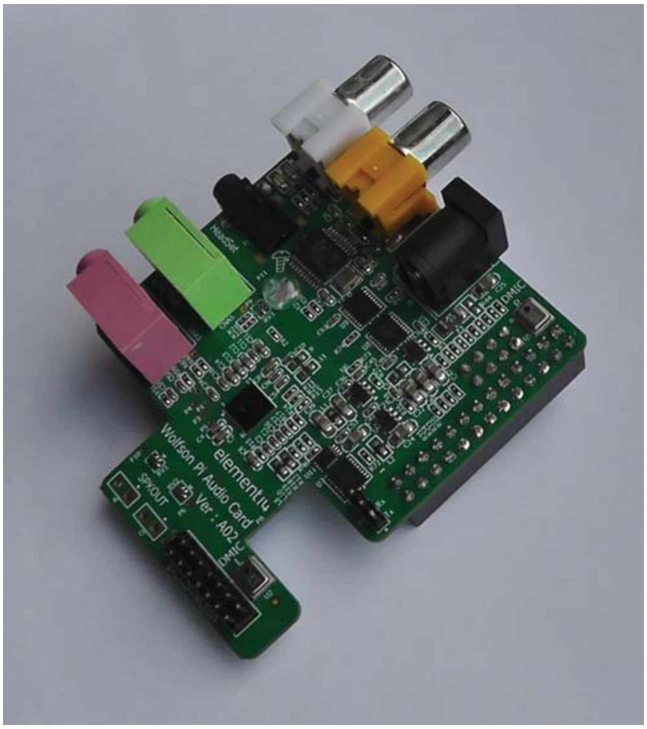

Figure 1.4 Wolfson Pi audio card.

This is useful in some of the program examples where the demands of running in real-time approach the limits of what is achievable with the ARM Cortex-M4. One difference between the two devices used in this book is that STM32F407 uses a processor clock speed of 168 MHz, whereas the TM4C123 clock speed is 84 MHz. As presented in the book, all of the program examples will run in real time on either device. However, if the parameter values used are changed, for example, if the number of coefficients in an FIR filter is increased, it is likely that the limits of the slower device will be reached more readily than those of the faster one.

All of the program examples have been tested using the free, code size-limited, version of *MDK-ARM*. The aim of hands-on DSP teaching, and the intention of this book, is not to teach about the architecture of the ARM Cortex-M4. The device is used because it provides a capable and inexpensive platform. Nor is it the aim of hands-on DSP teaching, or the intention of this book, to teach about the use of *MDK-ARM*. The aim of hands-on DSP teaching is to reinforce DSP theory taught in lectures through the use of illustrative examples involving the real-time processing of audio signals in an electrical engineering laboratory environment. That is to say where test equipment such as oscilloscopes, signal generators, and connecting cables are available.

1.1.3 Hardware and Software Tools

To perform the experiments described in this book, a number of software and hardware resources are required.

1. An ARM Cortex-M4 development board and audio interface. Either a Texas Instruments TM4C123 LaunchPad and a CircuitCo audio booster pack or an STMicroelectronics STM32F407 Discovery board and a Wolfson Microelectronics Pi audio card are suitable hardware platforms.
2. A host PC running an integrated development environment (IDE) and with a spare USB connection. The program examples described in this book were developed and tested using the *Keil MDK-ARM* development environment. However, versions of the program examples for the TM4C123 LaunchPad and project files compatible with Texas Instruments *Code Composer Studio* IDE are provided on the partner website `http://www.wiley.com/go/Reay/ARMcortexM4`.
3. The TM4C123 LaunchPad and the STM32F407 Discovery board use slightly different USB cables to connect to the host PC. The launchpad is supplied with a USB cable, while the STM32F407 Discovery is not.
4. Whereas the audio booster pack and the launchpad plug together, the Wolfson Pi audio card does not plug onto the STM32F407 Discovery board. Connections between the two can be made using a custom ribbon cable, available from distributor Farnell.
5. An oscilloscope, a signal generator, a microphone, headphones, and various connecting cables. Several of these items will be found in almost any electrical engineering laboratory. If you are using the STM32F407 Discovery and Wolfson Pi audio card, then a microphone is unnecessary. The audio card has built-in

digital MEMS microphones. The Wolfson Pi audio card is also compatible with combined microphone and headphone headsets (including those supplied with Apple and Samsung smartphones). Stereo 3.5 mm jack plug to 3.5 mm jack plug cables and stereo 3.5 mm jack plug to (two) RCA (phono) plugs and RCA to BNC adapters are the specific cables required.

6. Project and example program files from the partner website `http://www.wiley.com/go/Reay/ARMcortexM4`.

REFERENCE

1. Yiu, J., "The Definitive Guide to ARM® Cortex®-M3 and Cortex®-M4 Processors", Third Edition, Elsevier Inc., 2014.

2

ANALOG INPUT AND OUTPUT

2.1 INTRODUCTION

A basic DSP system, suitable for processing audio frequency signals, comprises a digital signal processor (DSP) and analog interfaces as shown in Figure 2.1. The Texas Instruments TM4C123 LaunchPad and audio booster pack provide such a system, using a TM4C123 ARM® Cortex®-M4 processor and a TLV320AIC3104 (AIC3104) codec [1]. The STMicro STM32F407 Discovery and the Wolfson audio card provide such a system, using an STM32407 ARM® Cortex®-M4 processor and a WM5102 codec [2]. The term codec refers to the coding of analog waveforms as digital signals and the decoding of digital signals as analog waveforms. The AIC3104 and WM5102 codecs perform both the analog-to-digital conversion (ADC) and digital-to-analog conversion (DAC) functions shown in Figure 2.1.

Both the AIC3104 and WM5102 codecs communicate with their associated processors (TM4C123 and STM32F407) using I2C bus for control (writing to the codec's control registers) and I2S for (audio) data transfer.

2.1.1 Sampling, Reconstruction, and Aliasing

Within DSPs, signals are represented as sequences of discrete sample values, and whenever signals are sampled, the possibility of aliasing arises. Later in this chapter, the phenomenon of aliasing is explored in more detail. Suffice to say at this stage that aliasing is undesirable and that it may be avoided by the use of an antialiasing filter placed at the input to the system shown in Figure 2.1 and by suitable design

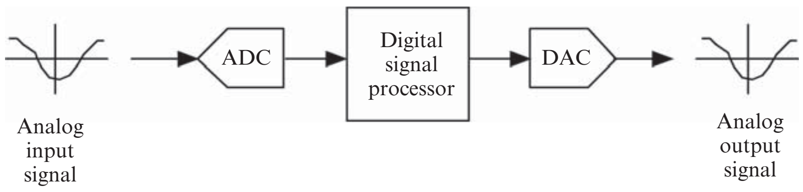

Figure 2.1 Basic digital signal processing system.

of the DAC. In a low-pass system, an effective antialiasing filter is one that allows frequency components at frequencies below half the sampling frequency to pass but that attenuates greatly, or stops, frequency components at frequencies greater than or equal to half the sampling frequency. A suitable DAC for a low-pass system is itself a low-pass filter having characteristics similar to the aforementioned antialiasing filter. The term DAC commonly refers to an electronic device that converts discrete sample values represented in digital hardware into a continuous analogue electrical signal. When viewed purely from a signal processing perspective, a DAC acts as a reconstruction filter. Although they differ in a number of respects, both the AIC3104 and WM5102 codecs contain both digital and analog antialiasing and reconstruction filters and therefore do not require additional external filters.

2.2 TLV320AIC3104 (AIC3104) STEREO CODEC FOR AUDIO INPUT AND OUTPUT

The audio booster pack makes use of a TLV320AIC3104 (AIC3104) codec for analog input and output (see Figures 2.2 and 2.3). The AIC3104 is a low-power stereo

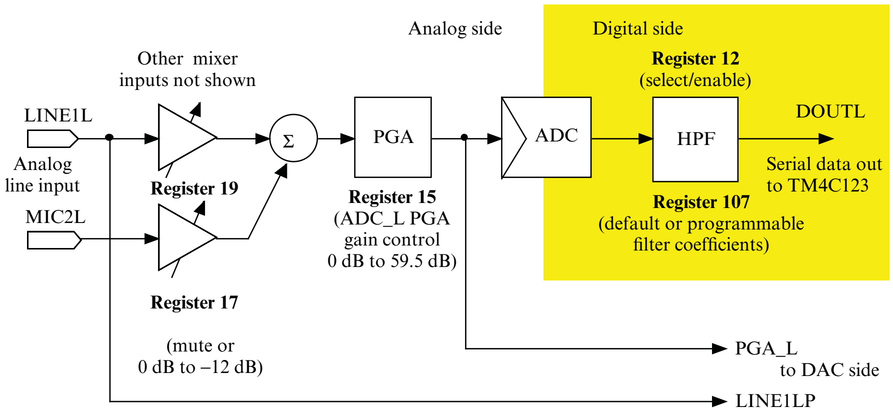
Analog side
Digital side
Other mixer inputs not shown
Register 12 (select/enable)
LINE1L
Analog line input
MIC2L
Σ
PGA
ADC
HPF
DOUTL
Serial data out to TM4C123
Register 19
Register 15 (ADC_L PGA gain control 0 dB to 59.5 dB)
Register 107 (default or programmable filter coefficients)
Register 17
(mute or 0 dB to −12 dB)
PGA_L to DAC side
LINE1LP

Figure 2.2 Simplified block diagram representation of input side of AIC3104 codec showing selected blocks and signal paths used by the example programs in this book (left channel only).

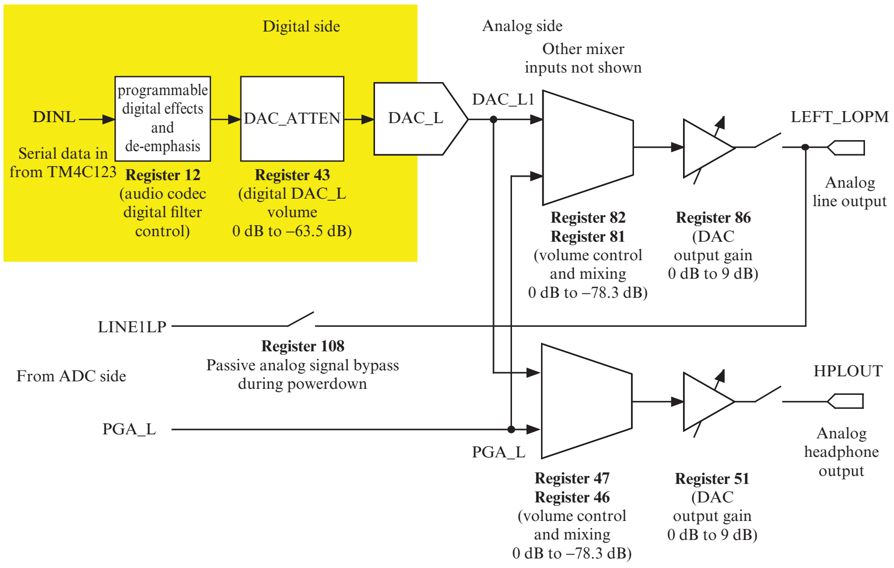
Digital side
Analog side
Other mixer inputs not shown
DINL
Serial data in from TM4C123
programmable digital effects and de-emphasis
Register 12 (audio codec digital filter control)
DAC_ATTEN
Register 43 (digital DAC_L volume 0 dB to −63.5 dB)
DAC_L
DAC_L1
Register 82
Register 81 (volume control and mixing 0 dB to −78.3 dB)
Register 86 (DAC output gain 0 dB to 9 dB)
LEFT_LOPM
Analog line output
LINE1LP
Register 108 Passive analog signal bypass during powerdown
From ADC side
PGA_L
HPLOUT
Analog headphone output
Register 47
Register 46 (volume control and mixing 0 dB to −78.3 dB)
Register 51 (DAC output gain 0 dB to 9 dB)

Figure 2.3 Simplified block diagram representation of output side of AIC3104 codec showing selected blocks and signal paths used by the example programs in this book (left channel only).

audio codec, based on sigma-delta technology, and designed for use in portable battery-powered applications. It features a number of microphone and line-level inputs, configurable for single-ended or differential connection. On its output side, a number of differential and high-power outputs are provided. The high-power outputs are capable of driving headphones. A number of different sampling rates ranging from 8 to 96 kHz are supported by the device. The analog-to-digital converter (ADC), or coder, part of the codec converts an analog input signal into a sequence of (16-bit, 24-bit, or 32-bit signed integer) sample values to be processed by the DSP. The digital-to-analog converter (DAC), or decoder, part of the codec reconstructs an analog output signal from a sequence of (16-bit, 24-bit, or 32-bit signed integer) sample values that have been processed by the DSP and written to the DAC.

Also contained in the device are several programmable digital filters and gain blocks. The codec is configured using a number of control registers, offering so many options that it is beyond the scope of this text to describe them fully. However, choices of sampling frequency, input connection, and ADC PGA gain are made available in the example programs through the parameters passed to function `tm4c123_aic3104_init()`. In addition, it is possible to write to any of the codec control registers using function `I2CRegWrite()`.

Later in this chapter, examples of enabling some of the internal digital filter blocks by writing to the control registers of the AIC3104 are described. In Chapter 4, the characteristics of the programmable digital filters within the AIC3104 are examined in greater detail.

Data is passed to and from the AIC3104 via its I2S serial interface. MIC IN (pink), LINE IN (blue), LINE OUT (green), and HP OUT (black) connections are made available via four 3.5 mm jack sockets on the audio booster pack, and these are connected to the AIC3104 as shown in Figure 2.4. In addition, for reasons explained later in this chapter, jumpers J6 and J7 on the audio booster pack allow connection of first-order low-pass filters and scope hook test points TP2 and TP3 to LINE OUT on the AIC3104.

2.3 WM5102 AUDIO HUB CODEC FOR AUDIO INPUT AND OUTPUT

The Wolfson audio card makes use of a WM5102 audio hub for analog input and output. The WM5102 features a low-power, high-performance audio codec.

Data is passed to and from the WM5102 via its I2S serial interface, and the device is configured by writing to its control registers via an I2C interface. In addition to a number of configurable filter and gain blocks, the WM5102 codec contains a programmable DSP. However, use of this proprietary DSP is beyond the scope of this book.

LINE IN (pink), LINE OUT (green), and combined MIC IN and HP OUT (black) connections are made available via three 3.5 mm jack sockets on the Wolfson audio card.

2.4 PROGRAMMING EXAMPLES

The following examples illustrate analog input and output using either the TM4C123 LaunchPad and audio booster pack or the STM32F407 Discovery and Wolfson audio card. The program examples are available for either platform, although in most cases, only one platform is mentioned per example. A small number of example programs in this chapter concern programming the internal digital filters in the AIC3104 codec and are therefore applicable only to the Texas Instruments hardware platform. A small number of example programs concern use of the 12-bit DAC built in to the STM32F407 processor and are therefore applicable only to the STMicroelectronics hardware platform.

The example programs demonstrate some important concepts associated with analog-to-digital and digital-to-analog conversion, including sampling, reconstruction, and aliasing. In addition, they illustrate the use of polling-, interrupt-, and DMA-based i/o in order to implement real-time applications. Many of the concepts and techniques described in this chapter are revisited in subsequent chapters.

2.5 REAL-TIME INPUT AND OUTPUT USING POLLING, INTERRUPTS, AND DIRECT MEMORY ACCESS (DMA)

Three basic forms of real-time i/o are demonstrated in the following examples. Polling- and interrupt-based i/o methods work on a sample-by-sample basis, and

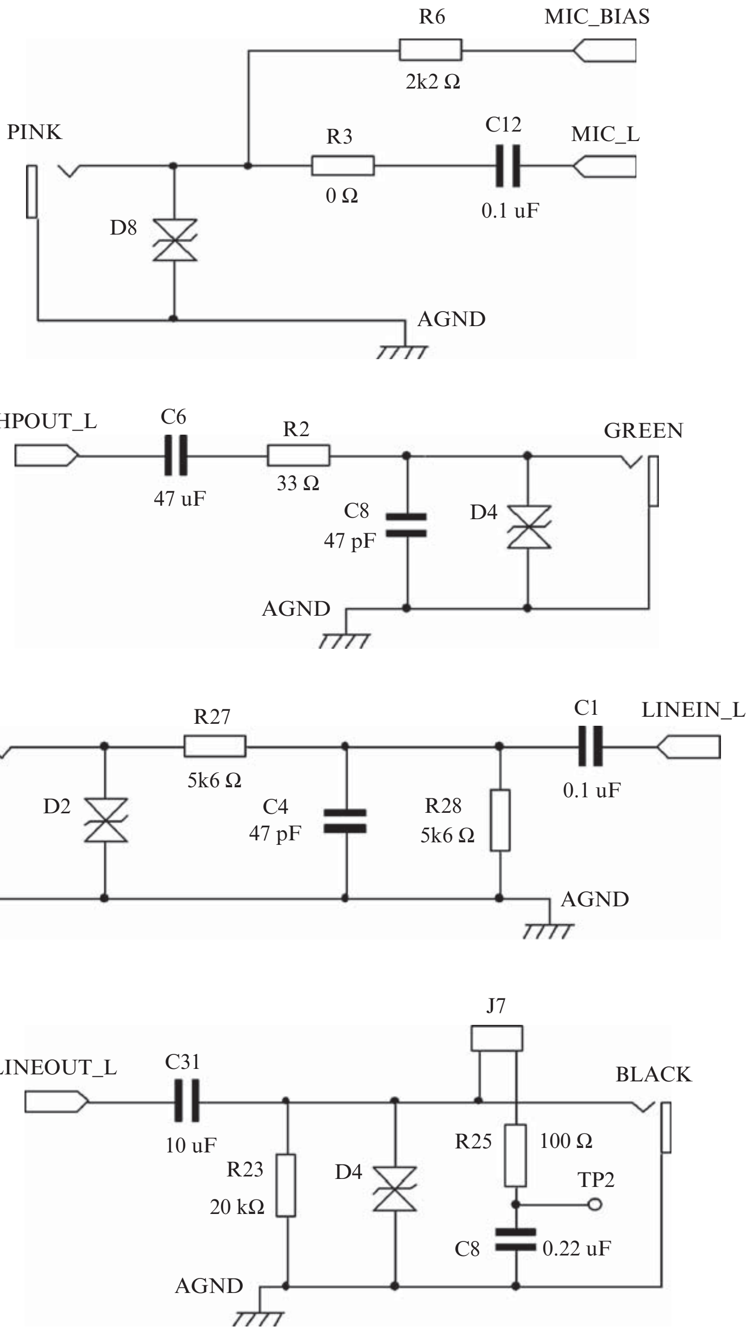

Figure 2.4 Analog input and output connections on the AIC3104 audio booster pack.

processing consists of executing a similar set of program statements at each sampling instant. DMA-based i/o deals with blocks, or frames, of input and output samples and is inherently more efficient in terms of computer processing requirements. Processing consists of executing a similar set of program statements after each DMA transfer. Block- or frame-based processing is closely linked to, but not restricted to, use with frequency-domain processing (using the FFT) as described in Chapter 5.

The following examples illustrate the use of the three different i/o mechanisms in order to implement a simple talk-through function. Throughout the rest of this book, use is made primarily of interrupt- and DMA-based methods. Compared to polling- and interrupt-based methods, there is a greater time delay between a signal entering the digital signal processing system and leaving it introduced by the DMA-based method. It is possible to make use of the DMA mechanism with a frame size of just one sample, but this rather defeats the purpose of using DMA-based i/o.

Example 2.1 **Basic Input and Output Using Polling (`tm4c123_loop_poll.c`).**

Listing 2.1 Program `tm4c123_loop_poll.c`.

```
// tm4c123_loop_poll.c

#include "tm4c123_aic3104_init.h"

void SSI_interrupt_routine(void){while(1){}}

int main(void)
{
  AIC3104_data_type sample_data;
  float32_t input_left, input_right;

  tm4c123_aic3104_init(FS_48000_HZ,
                       AIC3104_MIC_IN,
                       IO_METHOD_POLL,
                       PGA_GAIN_6_DB);
  while(1)
  {
    SSIDataGet(SSI1_BASE,&sample_data.bit32);
    input_left = (float32_t)(sample_data.bit16[0]);
    SSIDataGet(SSI0_BASE,&sample_data.bit32);
    input_right = (float32_t)(sample_data.bit16[0]);

    sample_data.bit32 = ((int16_t)(input_left));
    SSIDataPut(SSI1_BASE,sample_data.bit32);
    sample_data.bit32 = ((int16_t)(input_right));
    SSIDataPut(SSI0_BASE,sample_data.bit32);
  }
}
```

The C language source file for program, `tm4c123_loop_poll.c`, which simply copies input samples read from the AIC3104 codec ADC to the AIC3104 codec DAC as output samples, is shown in Listing 2.1. Effectively, the MIC IN input socket is connected straight through to the LINE OUT and HP OUT output sockets on the audio booster pack via the AIC3104 codec and the TM4C123 processor. Function `tm4c123_aic3104_init()`, called by program `tm4c123_loop_poll.c`, is defined in support file `tm4c123_aic3104_init.c`. In this way, the C source file `tm4c123_loop_poll.c` is kept as short as possible and potentially distracting low-level detail is hidden. The implementation details of function `tm4c123_aic3104_init()` and other functions defined in `tm4c123_aic3104_init.c` need not be studied in detail in order to use the examples presented in this book.

2.5.1 I2S Emulation on the TM4C123

The TM4C123 processor does not feature an I2S interface. Instead, two synchronous serial interface (SSI) interfaces, SSI0 and SSI1, are used to emulate a bidirectional stereo I2S interface and to pass audio data to and from the AIC3104 codec. One SSI interface handles the left channel and the other handles the right channel. Details of the I2S emulation are described in application note SPMA042 [3].

2.5.2 Program Operation

Following the call to function `tm4c123_aic3104_init()`, program `tm4c123_loop_poll.c` enters an endless while loop and repeatedly copies left and right channel input sample values into variables `input_left` and `input_right`, using function `SSIDataGet()`, before writing these sample values to the AIC3104 DAC, using function `SSIDataPut()`. Function `SSI-DataGet()` waits until there is data in the receive FIFO of the specified SSI peripheral, `SSI0_BASE` or `SSI1_BASE`, and function `SSIDataPut()` waits until there is space available in the transmit FIFO of the specified SSI peripheral. In this way, the real-time operation of the program is controlled by the timing of the I2S interface, which, in turn, is determined by the AIC3104 codec (acting as I2S master). Functions `SSIDataGet()` and `SSIDataPut()` are defined in the TM4C123 device family pack (DFP) installed as part of the *MDK-ARM* development environment. Although the AIC3104 is configured to use 16-bit sample values, function `SSIDataGet()` returns a 32-bit value and function `SSIDataPut()` is passed a 32-bit value.

Function `SSI_interrupt_routine()` is not used by program `tm4c123_loop_poll.c` but has been defined here as a trap for unexpected SSI peripheral interrupts.

In this simple example, it is not strictly necessary to convert the 16-bit sample values read from the AIC3104 ADC by function `SSIDataGet()` into 32-bit floating-point values. However, in subsequent program examples, DSP algorithms are implemented using floating-point arithmetic and input sample values are converted into type `float32_t`. Processing of the floating-point sample values could be implemented by adding program statements between

```
    input_right = (float32_t)(sample_data.bit16[0]);
```

and

```
    sample_data.bit32 = ((int16_t)(input_left));
```

2.5.3 Running the Program

Connect a microphone to the (pink) MIC IN socket on the audio booster card and headphones to the (green) HP OUT socket. Run the program and verify that the input to the microphone can be heard in the headphones.

2.5.4 Changing the Input Connection to LINE IN

Change the program statement

```
tm4c123_aic3104_init(FS_48000_HZ,
                     AIC3104_MIC_IN,
                     IO_METHOD_POLL,
                     PGA_GAIN_6_DB);
```

to read

```
tm4c123_aic3104_init(FS_48000_HZ,
                     AIC3104_LINE_IN,
                     IO_METHOD_POLL,
                     PGA_GAIN_6_DB);
```

Rebuild the project and run the program again using a signal from a sound card, a signal generator, or an MP3 player connected to the (blue) LINE IN socket as input.

2.5.5 Changing the Sampling Frequency

Change the sampling frequency used by passing parameter value `FS_8000_HZ` rather than `FS_48000_HZ` to the codec initialization function, that is, by changing the program statement

```
tm4c123_aic3104_init(FS_48000_HZ,
                     AIC3104_LINE_IN,
                     IO_METHOD_POLL,
                     PGA_GAIN_6_DB);
```

to read

```
tm4c123_aic3104_init(FS_8000_HZ,
                     AIC3104_LINE_IN,
                     IO_METHOD_POLL,
                     PGA_GAIN_6_DB);
```