HANDS-ON REAL-TIME DSP TEACHING USING INEXPENSIVE ARM CORTEX M4 DEVELOPMENT SYSTEMS

Donald S. Reay

School of Engineering and Physical Sciences,
Heriot-Watt University, Edinburgh EH14 4AS, United Kingdom
d.s.reay@hw.ac.uk

ABSTRACT

Hands-on, real-time digital signal processing (DSP) program examples from the textbook *Digital Signal Processing and Applications with the OMAP-L138 eXperimenter* [1] have been ported to a number of different, inexpensive ARM Cortex M4 microcontroller-based evaluation modules (EVMs). The ARM Cortex M4 is a DSP-enhanced microcontroller with floating point unit (FPU) that is capable of running these real-time examples. EVMs using this microcontroller are an order of magnitude less expensive than more conventional DSP EVMs (including the OMAP-L138 eXperimenter.) ARM Cortex M4 EVMs including the STMicro STM32F4 Discovery, Texas Instruments Tiva Launchpad, and Freescale FRDM-K20D50M represent a quantum change in the cost of hardware suitable for hands-on, real-time DSP teaching. However, these EVMs do not include audio codecs, necessitating the development of additional hardware. Prototype audio codec daughter cards for three different EVMs have been demonstrated successfully.

Index Terms— digital signal processing, engineering education

1. INTRODUCTION

The use of digital signal processor (DSP) evaluation modules (EVMs) in university teaching laboratories worldwide is well established. Numerous textbooks [2, 3, 4], on-line material, and the efforts of different manufacturers' university programmes provide support for this. Conventionally, EVMs featuring specialised DSP hardware and typically costing over $300 each are used. In many cases this hardware finds little use outside DSP teaching and, to some extent, perpetuates a mystique surrounding dedicated DSP hardware.

The advent of DSP-enhanced microcontrollers in the form of the ARM Cortex M4 and a number of different EVMs costing approximately $10 each affords an exciting opportunity to spread hands-on DSP teaching both to laboratories equipped for more general microcontroller education and to institutions that have found the cost of providing dedicated DSP hardware too high. Already, ARM Cortex M4 EVMs are being adopted for microcontroller education [6].

Examples of inexpensive EVMs include the STMicro STM32F4 Discovery, the Texas Instruments Tiva Launchpad, and the Freescale Freedom FRDM-K20D50M. However, these EVMs do not feature hardware audio codecs (although the STM32F4 Discovery features an audio DAC) to allow for real-time analogue input and output. For the purposes of porting the hands-on laboratory examples developed previously for the Texas Instruments C6713 DSK and OMAP-L138 EVMs [2, 1], inexpensive audio codec daughter cards have been developed.

2. HANDS-ON DSP TEACHING USING ARM CORTEX M4 EVMS

Hands-on examples of digital signal processing algorithms using actual real-time audio signals in a laboratory are provided. Their aim is not to teach the architecture or programming model of the ARM Cortex M4 microcontroller, or the use of the MDK-ARM or CCS development environments, beyond what is necessary to carry out the experiments, but to re-inforce digital signal processing theory taught in lectures.

The teaching materials [1] comprise a large number of simple program examples that may act as the starting point for further teaching exercises. Specifically, they are intended to demonstrate fundamental DSP concepts including FIR and IIR filters, the FFT, and LMS adaptive filters.

A basic audio frequency digital signal processing system comprises analogue to digital and digital to analogue converters as well as a processor. In the case of the OMAP-L138 eXperimenter, such a system was provided on a single EVM board, with 3.5mm jack sockets enabling connection to oscilloscopes, signal generators, PC soundcards, loudspeakers, headphones, mp3 players, etc.

Although they have less computational power, each of the aforementioned ARM Cortex M4 EVMs can run the examples developed originally for the OMAP-L138. Code may be developed for each of the EVMs using the same development environment (MDK-ARM, available free of charge in a code-size limited version) or, in the case of the Tiva Launchpad, using Code Composer Studio. The different manufacturers' processors share the same ARM Cortex M4 core, including DSP instructions and FPU but differ slightly in terms of the on-chip peripheral functions provided. The DSP instructions are important in enabling the microcontrollers to run the real-time example programs. The FPU is important also because, in the interests of clarity, the example programs are written using floating point variables.(The Kinetis K20 processor on the FRDM-K20D50M does not have a FPU but is able to run several of the example programs.)

Each manufacturer provides slightly different set of library functions to support their device. Hence, the low-level parts of the source code for each different EVM is slightly different. However this level of detail is effectively hidden from a student concentrating on the overall structure of the program examples and on the DSP algorithms implemented. There are also slight differences in the processor clock speeds used by each EVM and in the physical connections provided. Audio codecs are interfaced using I2C and I2S peripherals.

The STMicro STM32F4 Discovery board has a processor clock speed of 168 MHz, spare I2S and I2C peripherals, and also two on-board 12 bit DACs and one external stereo audio DAC. The Freescale FRDM-K20D50M EVM has a processor clock speed of 50 MHz and

978-1-4799-2893-4/14/$31.00 ©2014 IEEE

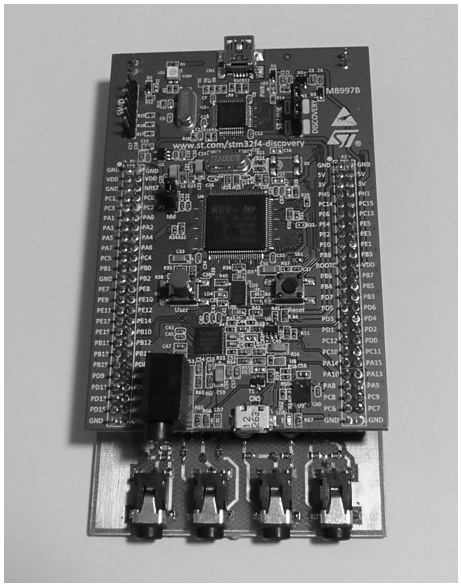

Fig. 1. STMicro STM32F4 Discovery EVM and AIC23-based audio codec daughter card.

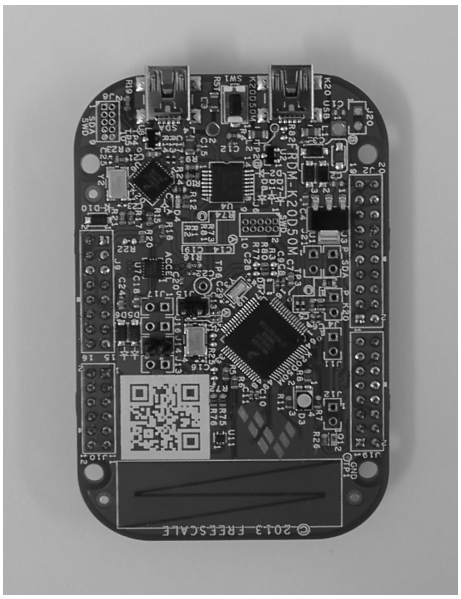

Fig. 2. Freescale FRDM-K20D50M EVM.

spare I2S and I2C peripherals. The Texas Instruments Tiva Launchpad has a processor clock speed of 80 MHz, a spare I2C peripheral, and I2S emulation is possible using two SSI peripheral interfaces and a single external inverter. Additionally, the lack of a conventional I2S interface means that the Tiva Launchpad cannot easily provide the master clock signal required by an audio codec and therefore its daughter card must include a XTAL oscillator module. All three EVMs derive their power from a USB connection to a host PC running the IDE. Each of the EVMs stores program code in flash memory and will run that program on power up and reset.

Prototype audio codec daughter cards using a Texas Instruments AIC23 codec have been built and tested for the three different EVMs. Figure 1 shows an ST32F4 Discovery EVM fitted with an AIC23-based daughter card. The four jack sockets are for mono MIC IN, and stereo LINE IN, LINE OUT and HP OUT. This is directly equivalent to the audio interface on the Texas Instruments C6713 DSK which also used the AIC23 codec. Figure 2 shows the Freescale FRDM-K20D50M EVM. Figure 3 shows an AIC23-based audio daughter card for the Tiva Launchpad. The electrical connections to each EVM are I2C for codec configuration, I2S for duplex data communication and a 3V3 power supply. Figure 4 shows a Tiva Launchpad and daughter card running an example program.

In addition to constructing AIC23-based daughter cards, the EVMs have been tested successfully connected to an EVM for the Texas Instruments AIC3106 audio codec used on the OMAP-L138 eXperimenter.

3. REAL-TIME CONSIDERATIONS

The hands-on DSP teaching examples serve as an introduction to the structure of C programs operating with hard real-time constraints. In view of the simple nature of each example, it is not necessary to make use of a real-time operating system. Three different methods of i/o; polling-, interrupt-, and DMA-based are used. Several of the simpler examples illustrating sampling and reconstruction of signals, and many of the FIR and IIR filter examples, use interrupt-driven (one interrupt per sampling instant) i/o. FFT examples use DMA-based i/o, and several examples are provided in both interrupt-driven and DMA-based i/o versions.

One way in which the examples differ from those developed for the OMAP-L138 is that due to the lesser computational power of the ARM Cortex M4, greater use has been made of optimised functions provided by the CMSIS DSP library as opposed to clearer, explicit, but computationally inefficient, programming of algorithms. A number of different examples are provided that trade off sampling rate, filter or FFT order, with clarity of source coding. Many of the CMSIS DSP functions are designed to process blocks of data and consequently a greater proportion of the example programs provided for the ARM Cortex M4 use DMA-based i/o.

The following examples illustrate the use of the different i/o methods on Tiva and STM32F4 EVMs simply to copy samples from ADC to DAC. In each case, filtering or other algorithms can be inserted at the points indicated. These examples are presented in order to give an idea of the structure of the programs provided.

3.1. Simple Program Example - `tiva_loop_poll.c`

Figure 5 shows a listing of example program `tiva_loop_poll.c` which runs on the Texas Instruments Tiva Launchpad. This program uses polling-based i/o. Function `main()` comproses an endless loop within which sample values are read from the ADC and then written to the DAC. Functions `input_sample()`, `output_sample()` and `tiva_init_poll()` are defined in the source file `aic23_init.c` in order to hide unnecessary low-level detail from the student. Function `tiva_init_poll()` initialises the AIC23 codec for polling-based i/o. 32-bit variable `sample` comprises two 16-bit (left and right channel) samples.

While demonstrating real-time audio input and output with a minimum of code, polling-based i/o is not computationally efficient.

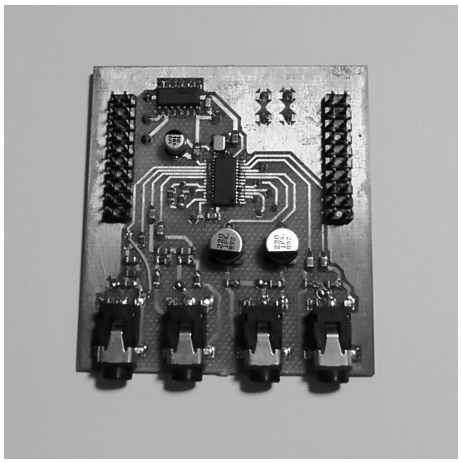

Fig. 3. AIC23-based boosterpack daughter card for Texas Instruments Tiva Launchpad.

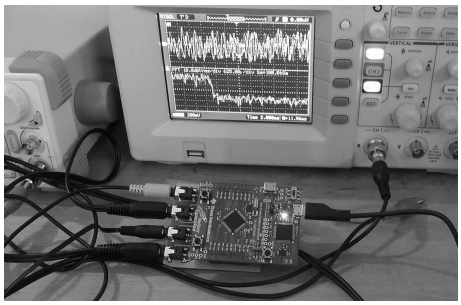

Fig. 4. Texas Instruments Tiva Launchpad and AIC23 boosterpack running an example program.

3.2. Simple Program Example - `stm32f4_loop_intr.c`

Figure 6 shows a listing of program `stm32f4_loop_intr.c` which runs on the STMicro STM32F4 Discovery. This program uses interrupt-based i/o. Function `SPI2_IRQHandler()` is called at everysample instant and in this case simply reads a sample value from the ADC and writes it to the DAC. As in the previous example, functions `input_sample()`, `output_sample()` and `stm32f4_init_intr()` are defined in file `aic23_init.c` in order to hide unnecessary low-level detail from the student. Function `stm32f4_init_intr()` initialises the AIC23 codec for interrupt-based i/o.

Following initialisation, function `main()` simply enters an infinite loop and all subsequent processing takes place within the interrupt service routine `SPI2_IRQHandler()`. This simple program example is instructive because its structure is used in many more example programs. Processing algorithms may be inserted between `input_sample()` and `output_sample()` function calls.

3.3. Simple Program Example - `stm32f4_loop_dma.c`

Program `stm32f4_loop_dma.c`, partially listed in figure 7, uses DMA-based i/o and functions `DMA1_Stream3_IRQHandler()` and `DMA1_Stream4_IRQHandler()` are executed each time an input or output DMA transfer between ADC and memory or memory and DAC is completed. Implementation of ping-pong buffering

```
// tiva_loop_poll.c

#include "aic23_init.h"

main()
{
 uint32_t sample;

 tiva_init_poll(i2S_AudioFreq_8k,
                AIC23_LINE_IN);
 while(1)
 {
  sample = input_sample();
//
// insert processing algorithm here
//
  output_sample(sample);
 }
}
```

Fig. 5. Listing of program `tiva_loop_poll.c`.

```
// stm32f4_loop_intr.c

#include "aic31_init.h"

void SPI2_IRQHandler()
{
 int16_t sample;

 sample = input_left_sample();
//
// insert processing algorithm here
//
 output_left_sample();
}

main()
{
 stm32f4_init_intr(i2S_AudioFreq_8k,
                   AIC23_LINE_IN);
 while(1);
}
```

Fig. 6. Listing of program `stm32f4_loop_intr.c`.

is made relatively easy by the STM32F4 DMA mechanism. Following initialisation, function `main()` waits for both input and output DMA transfers to complete before processing a new block of input samples by calling function `process_buffer()`.

4. EXPERIMENTAL RESULTS

Figure 8 shows an oscilloscope trace of the output from an example program that passed an internally-generated PRBS through a band-pass FIR filter before outputting it via the audio codec in order to illustrate the frequency characteristics of the filter. This program was run on a Tiva Launchpad using an AIC23-based daughter card.

5. RELATION TO PRIOR WORK

The work presented here extends directly that described, most recently, in [1]. It represents the first comprehensive and practical implementation of that work using very significantly less expensive

```
// stm32f4_loop_dma.c

#include "aic23_init.h"

extern uint16_t pingIN[BUFSIZE],pingOUT[BUFSIZE],
                pongIN[BUFSIZE],pongOUT[BUFSIZE];
int rx_proc_buffer, tx_proc_buffer;
volatile int RX_buffer_full = 0;
volatile int TX_buffer_empty = 0;

void DMA1_Stream3_IRQHandler()
{
 if(DMA_GetITStatus(DMA1_Stream3,DMA_IT_TCIF3))
 {
  DMA_ClearITPendingBit(DMA1_Stream3,DMA_IT_TCIF3);
  if(DMA_GetCurrentMemoryTarget(DMA1_Stream3))
   rx_proc_buffer = PING;
  else
   rx_proc_buffer = PONG;
  RX_buffer_full = 1;
 }
}

void DMA1_Stream4_IRQHandler()
{
 if(DMA_GetITStatus(DMA1_Stream4,DMA_IT_TCIF4))
 {
  DMA_ClearITPendingBit(DMA1_Stream4,DMA_IT_TCIF4);
  if(DMA_GetCurrentMemoryTarget(DMA1_Stream4))
   tx_proc_buffer = PING;
  else
   tx_proc_buffer = PONG;
  TX_buffer_empty = 1;
 }
}

void process_buffer()
{
 int i;
 uint16_t *rxbuf, *txbuf;

 if (rx_proc_buffer == PING) rxbuf = pingIN;
 else rxbuf = pongIN;
 if (tx_proc_buffer == PING) txbuf = pingOUT;
 else txbuf = pongOUT;

 for (i=0 ; i<BUFSIZE ; i++)
//
// insert block processing algorithm here
//
  *txbuf++ = *rxbuf++;
 TX_buffer_empty = 0; RX_buffer_full = 0;
}

int main(void)
{
 aic23_udma_init(I2S_AudioFreq_8k, AIC23_LINE_IN);
 while(1)
 {
  while (!(RX_buffer_full && TX_buffer_empty));
   process_buffer();
 }
}
```

Fig. 7. Partial listing of program `stm32f4_loop_dma.c`.

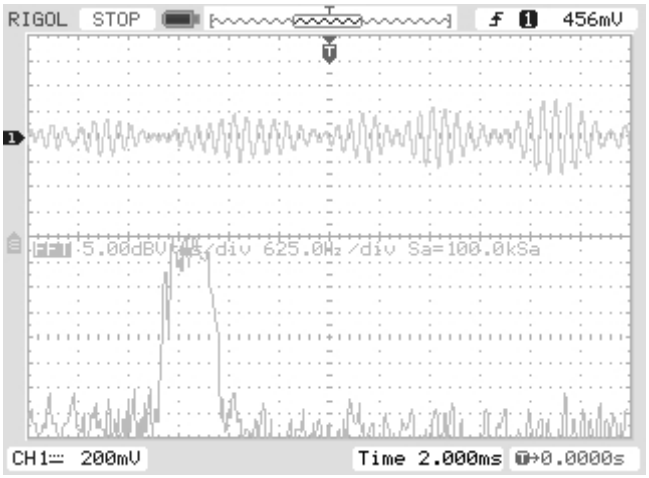

Fig. 8. Output from program `tiva_firprn_dma.c` showing band-pass filtered PRBS.

hardware and, in so doing, moves laboratory-based DSP education closer to more general microcontroller education.

6. CONCLUSIONS

Low-cost ARM Cortex M4 microcontroller-based EVMs available from different manufacturers have the computational power needed to implement hands-on real-time DSP teaching examples, coded in C and using floating point variables, previously run on significantly more expensive platforms. It is hoped that this may lead to a greater uptake of hands-on DSP teaching in universities. Prototype inexpensive audio codec daughter cards providing audio frequency analogue i/o have been developed and demonstrated successfully. Commercial availability of at least one of the daughter cards described is anticipated later this year.

7. REFERENCES

[1] D. S. Reay, *Digital Signal Processing and Applications with the OMAP-L138 eXperimenter*, Wiley, 2012.

[2] R. Chassaing and D. S. Reay, *Digital Signal Processing and Applications with the TMS320C6713 and TMSC6416 DSK*, Wiley, 2008.

[3] R. Chassaing, *Digital Signal Processing and Applications with the C6713 and C6416 DSK*, Wiley, 2005.

[4] S. M. Kuo, B. H. Lee, and W. Tian, *Real-Time Digital Signal Processing: Fundamentals, Implementations and Applications*, Wiley, 2013.

[5] J. W. Valvano, *Embedded Systems: Introduction to ARM Cortex-M Microcontrollers: 1*, Create Space Independent Publishing Platform, 2012.