arm Education

L.EEC025 - Fundamentals of Signal Processing (FunSP)

2021/2022 – 1st semester

Week07, 29 Nov 2021

Objectives:

-evaluating DMA operation and concluding on its advantages and differences to interrupt-based transfer of individual samples,

-evaluating the graphical representation capabilities of the STM32F746G board and LCD.

DSP Education Kit

LAB 6

DMA operation and LCD graphical capabilities

Issue 1.0

Contents

1 Introduction

1.1 Lab overview

This laboratory experiment motivates DMA-based processing as an alternative to interrupt-based individual audio samples transfer, motivates to the low input-output delay (i.e. low latency) of the A/D and D/A operation on the STM32F746G board, and motivates to the graphical representation capabilities of the STM32F746G board and LCD.

2 Requirements

To carry out this lab, you will need:

- An STM32F746G Discovery board
- A PC running Keil MDK-Arm
- An oscilloscope
- Suitable connecting cables
- An audio frequency signal generator
- Optional: External microphone, although you can also use the microphones on the board
- Stereo headphones

3 DMA-Based Example Program

Direct Memory Access (DMA) is a method in which a hardware component of a computer gains access to the Memory Bus and controls the transfer of data. DMA controllers can be configured to handle data transfers between memories, memory to peripherals, and vice versa, enabling the processor deal with other processes. Essentially, the main benefit of this method is to reduce strain on the CPU. This concept is demonstrated in the block diagram below:

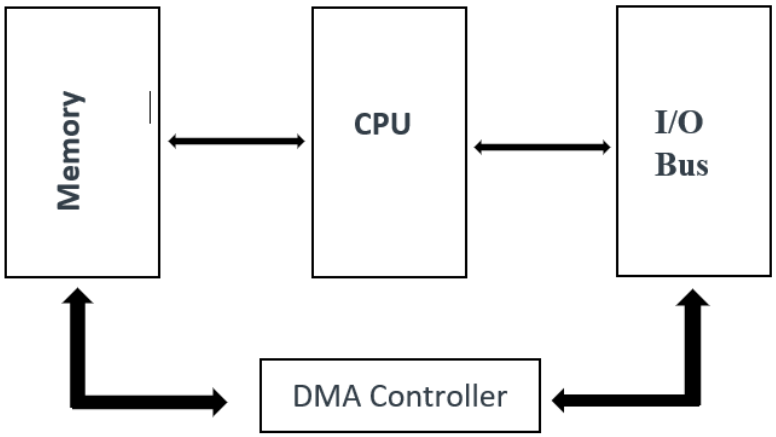

Figure 1: Block diagram representation DMA-Based I/O

3.1 Hearing the effect of the DMA buffer delays

The DMA-based I/O method introduces a delay in the signal path equal to two DMA transfer blocks, or buffers, of samples. The number of sampling periods represented by one DMA transfer block is determined by the value of `PING_PONG_BUFFER_SIZE`, which is 256 samples, and is defined in header file `stm32f7_wm8994_init.h`.

In this part of the lab experiment, we will use `stm32f7_loop_dma.c` C code in order to test how audible the DMA buffering delay is when the STM32F746G board is running the `stm32f7_loop_dma.c` C code in real-time.

Program `stm32f7_loop_dma.c`. has a similar functionality to program `stm32f7_loop_intr.c` except that it uses the DMA I/O, as opposed to interrupt-based.

The `stm32f7_loop_dma.c` C code is listed next.

```
// stm32f7_loop_dma.c

#include "stm32f7_wm8994_init.h"
#include "stm32f7_display.h"

#define SOURCE_FILE_NAME "stm32f7_loop_dma.c"

extern volatile int32_t TX_buffer_empty; // these may not need to be int32_t
extern volatile int32_t RX_buffer_full; // they were extern volatile int16_t
in F4 version
extern int16_t rx_buffer_proc, tx_buffer_proc; // will be assigned token
values PING or PONG


void process_buffer(void) // this function processes one DMA transfer block
worth of data
{
```

```
  int i;
  int16_t *rx_buf, *tx_buf;

  if (rx_buffer_proc == PING) {rx_buf = (int16_t *)PING_IN;}
  else {rx_buf = (int16_t *)PONG_IN;}
  if (tx_buffer_proc == PING) {tx_buf = (int16_t *)PING_OUT;}
  else {tx_buf = (int16_t *)PONG_OUT;}

  for (i=0 ; i<(PING_PONG_BUFFER_SIZE) ; i++)
  {
    *tx_buf++ = *rx_buf++;
    *tx_buf++ = *rx_buf++;
  }

  RX_buffer_full = 0;
  TX_buffer_empty = 0;
}

int main(void)
{
  stm32f7_wm8994_init(AUDIO_FREQUENCY_48K,
                      IO_METHOD_DMA,
                      INPUT_DEVICE_INPUT_LINE_1,
                      OUTPUT_DEVICE_HEADPHONE,
                      WM8994_HP_OUT_ANALOG_GAIN_0DB,
                      WM8994_LINE_IN_GAIN_0DB,
                      WM8994_DMIC_GAIN_9DB,
                      SOURCE_FILE_NAME,
                      NOGRAPH);
  while(1)
  {
    while(!(RX_buffer_full && TX_buffer_empty)){}
    process_buffer();
  }
}
```

Figure 2: Listing of program `stm32f7_loop_dma.c`

Based on this code, take a moment to

- figure out how many buffers are implied in the DMA mechanism,
- make a graphical representation of how the buffers are handled in the data transfer (in this perspective it is important to clarify the meaning of condition that is tested in `while(!(RX_buffer_full && TX_buffer_empty))`),
- understand what sampling frequency and input device are being used.

Now, change this line in the code:

```
                      INPUT_DEVICE_INPUT_LINE_1,
```

to these two:

```
                      INPUT_DEVICE_DIGITAL_MICROPHONE_2,
                      //INPUT_DEVICE_INPUT_LINE_1,
```

Now, proceed as usual to start the Keil MDK-Arm development environment (µVision) and to replace the existing `main()` file in that project by the new `main()` that is `stm32f7_loop_dma.c` .

Remember that the directory where you can find the the **`DSP_Education_Kit.uvprojx`** project file is:

C:\uvision\Keil\STM32F7xx_DFP\2.9.0\Projects\STM32746G-Discovery\Examples\DSP Education Kit\MDK-ARM

You can copy-paste this link directly to File Explorer in Windows for a quick and easy access. For your convenience, this link is also available on a TXT file on Moodle.

Now, proceed as usual to compile the code, downloading it to the STM32F746G board (by starting the debugger), and then to run the code.

Listen to the LINE OUT output signal using headphones. Please make sure that you are using headphones with a two-ring (stereo) mini-jack as it is illustrated in Figure 3. Three-ring mini-jacks are not appropriate as they include an addition microphone signal that the STM32F746G board does not support.

Figure 3: A two ring mini-jack (stereo) should be used.

Question 1: when you hear your own voice through the headphones (which is the signal input to the STM32F746G digital microphones), is the delay between the moment you speak and the moment you hear your voice audible ?

Now, change the sampling frequency first to 32 kHz, then to 16 kHz, and finally to 8 kHz. You can do this by changing in the code the parameter `AUDIO_FREQUENCY_48K` to `AUDIO_FREQUENCY_32K`, `AUDIO_FREQUENCY_16K`, `AUDIO_FREQUENCY_8K`, respectively. Remember that for each one of these cases, you need to stop the STM32F746G board real-time operation, quit the debugger mode and return to the editing mode, modify the C source code, compile, downloading it to the STM32F746G board (by starting the debugger again), and then to run the code.

Question 2: For what sampling frequencies is the DAM buffering delay audible ? What is the corresponding delay (in milliseconds) and do you explain that it becomes audible after a certain limit ?

3.2 Representing time and frequency on the STM32F746G board

In this part of the lab experiment, we use an extended version of the coded tested in 3.1 in the sense that time or frequency representations are displayed on the STM32F746G board LCD.

Now, proceed to replace the existing `main()` file in the STM32F746G project by the new `main()` that is `stm32f7_loop_graph_dma.c` .

The `stm32f7_loop_graph_dma.c` code is listed next.

```
// stm32f7_loop_graph_dma.c

#include "stm32f7_wm8994_init.h"
#include "stm32f7_display.h"

#define PLOTBUFSIZE 128

#define BLOCK_SIZE 1

#define SOURCE_FILE_NAME "stm32f7_loop_graph_dma.c"

extern volatile int32_t TX_buffer_empty; // these may not need to be int32_t
extern volatile int32_t RX_buffer_full; // they were extern volatile int16_t
in F4 version
extern int16_t rx_buffer_proc, tx_buffer_proc; // will be assigned token
values PING or PONG

float32_t x[PING_PONG_BUFFER_SIZE];

float32_t cmplx_buf[2*PING_PONG_BUFFER_SIZE];
float32_t outbuffer[PING_PONG_BUFFER_SIZE] = { 0.0f };

void process_buffer(void) // this function processes one DMA transfer block of
data
{
  int i;
  int16_t *rx_buf, *tx_buf;


  if (rx_buffer_proc == PING) {rx_buf = (int16_t *)PING_IN;}
  else {rx_buf = (int16_t *)PONG_IN;}
  if (tx_buffer_proc == PING) {tx_buf = (int16_t *)PING_OUT;}
  else {tx_buf = (int16_t *)PONG_OUT;}

  for (i=0 ; i<(PING_PONG_BUFFER_SIZE) ; i++)
```

```
  {
    x[i] = (float32_t)(*rx_buf);
    *tx_buf++ = *rx_buf++;
    *tx_buf++ = *rx_buf++;
    cmplx_buf[i*2] = x[i]; // real part
    cmplx_buf[(i*2)+1] = 0.0; // imaginary part
  }

  RX_buffer_full = 0;
  TX_buffer_empty = 0;
}

int main(void)
{
      int i;
      int button = 0;

  stm32f7_wm8994_init(AUDIO_FREQUENCY_8K,
                      IO_METHOD_DMA,
                      INPUT_DEVICE_DIGITAL_MICROPHONE_2,
                      OUTPUT_DEVICE_HEADPHONE,
                      WM8994_HP_OUT_ANALOG_GAIN_0DB,
                      WM8994_LINE_IN_GAIN_0DB,
                      WM8994_DMIC_GAIN_9DB,
                      SOURCE_FILE_NAME,
                      GRAPH);
  while(1)
  {
    while(!(RX_buffer_full && TX_buffer_empty)){}
    BSP_LED_On(LED1);
    process_buffer();
    button = checkButtonFlag();
    if(button == 1)
    {
      for(i=0; i<PING_PONG_BUFFER_SIZE; i++)
      {
        cmplx_buf[2*i] = x[i];
        cmplx_buf[2*i + 1] = 0.0;
      }
      arm_cfft_f32(&arm_cfft_sR_f32_len256, (float32_t *)(cmplx_buf), 0, 1);
      arm_cmplx_mag_f32((float32_t *)(cmplx_buf),(float32_t *)(outbuffer),
PING_PONG_BUFFER_SIZE);
      plotLogFFT(outbuffer, PING_PONG_BUFFER_SIZE, LIVE);
    }
    else
    {
      plotWave(x, PLOTBUFSIZE, LIVE, ARRAY);
    }
    BSP_LED_Off(LED1);
  }
}
```

Figure 4: Listing of program `stm32f7_loop_graph_dma.c`

Compile this new code, and download it to the STM32F746G board. Run program `stm32f7_loop_graph_dma.c` and confirm that the input to the digital microphones (i.e. your voice) is passed to the headphones. A major difference between previous programs and `stm32f7_loop_graph_dma.c` is that the latter program plots the sample values it writes to the WM8994 DAC as a graph on the LCD. Pressing the blue user pushbutton **toggles** between time-domain and frequency-domain representations of those sample values.

Take a moment to observe the STM32F746G LCD graphical representations as you change your voice signal.

Question 3: if you change the code, as suggested in Questions 1 and 2, in order to increase the sampling frequency from 8kHz, to 16 kHz, and then to 32 kHz or 48 kHz, after a certain case you start to hear artifacts on the audio signal coming out of the board LINE OUT. How do you explain these ?
(Hint: think on the additional communication load that is imposed on the STM32F746G board)

Now, change the parameter `INPUT_DEVICE_DIGITAL_MICROPHONE_2` to `INPUT_DEVICE_INPUT_LINE_1` and the sampling frequency back to 8 kHz in program `stm32f7_loop_graph_dma.c` and use a signal generator to input a sinusoid of frequency 180 Hz (and 5 Vpp) to the LINE IN input (Remember: make sure that you use the adapter with the **blue mini-jack** whose interface board has a resistor divider. It is meant to protect the analog input of the ***kit*** against excessive voltage levels).

You should see graphs on the LCD similar to those shown in Figure 5.

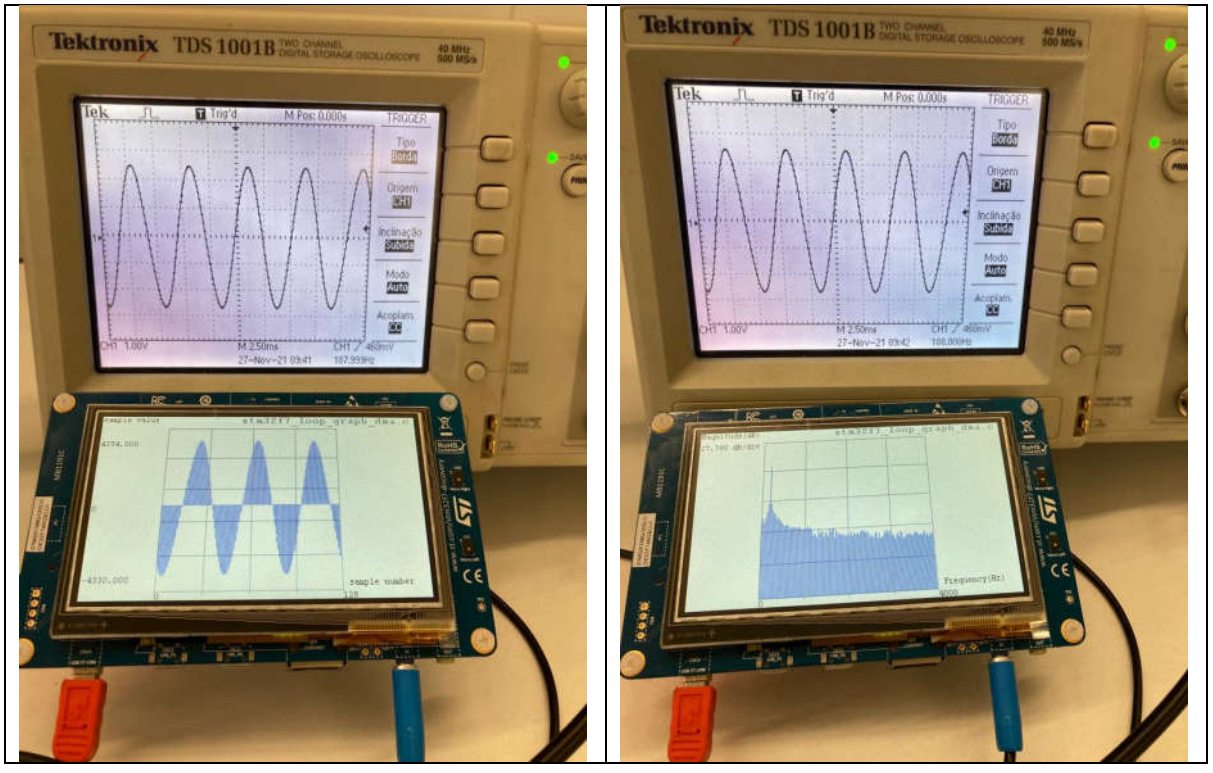

Figure 5: Graphical representation of 180 Hz sinusoidal signal input to program `stm32f7_loop_graph_dma.c` in the time-domain (figure o the left), and in the frequency domain (figure on the right). Sampling frequency is 8 kHz

Question 4: how does the time and frequency representation looks like as you change the input sine wave frequency ?

Question 5: Imagine that you want to implement a trigger effect (as the one that the oscilloscope implements) such that the time representation on the STM32F746G LCD becomes steady. Discuss with your colleagues how that could be achieved.

4 Conclusions

At the end of this exercise, you should have become familiar with the DMA operation, with the differences between interrupt-based transfer of audio samples, and DMA-based transfer of audio samples, and their impact in terms of processing and processor load. The graphical representation on the STM32F746G LCD is a very convenient feature that we will use in subsequent lab experiments.

5 Additional References

Link to Board information and resources:

https://www.st.com/en/evaluation-tools/32f746gdiscovery.html#overview

Using DMA controllers in STM Discovery boards:

https://www.st.com/content/ccc/resource/technical/document/application_note/27/46/7c/ea/2d/91/40/a9/DM00046011.pdf/files/DM00046011.pdf/jcr:content/translations/en.DM00046011.pdf

For more details about DMA:

http://cires1.colorado.edu/jimenez-group/QAMSResources/Docs/DMAFundamentals.pdf