arm Education

L.EEC025 - Fundamentals of Signal Processing (FunSP)

2024/2025 – 1st semester

Week06, 21 Oct 2024

Objectives:

-understanding the DMA mechanism and frame-based signal processing,

-evaluating the 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 5
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 (in an autonomous way). DMA controllers can be configured to handle data transfers between memories, memory to peripherals, and vice versa, enabling the processor to deal with other processes. Essentially, the main benefit of this method is to reduce strain on the CPU. This concept is suggested by the block diagram in Figure 1.

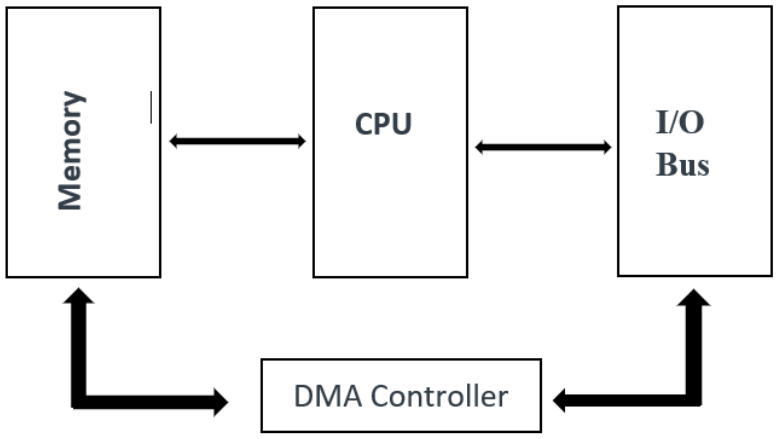

Figure 1: Block diagram representation DMA-Based I/O

Program `stm32f7_loop_dma.c` makes use of a "ping-pong" mode of multi-buffering possible on the STM32F746G to implement frame-based processing, instead of sample-based interrupt processing, as it was the case in previous laboratory experiments. The schematic representation of the DMA operation is represented in Figure 2.

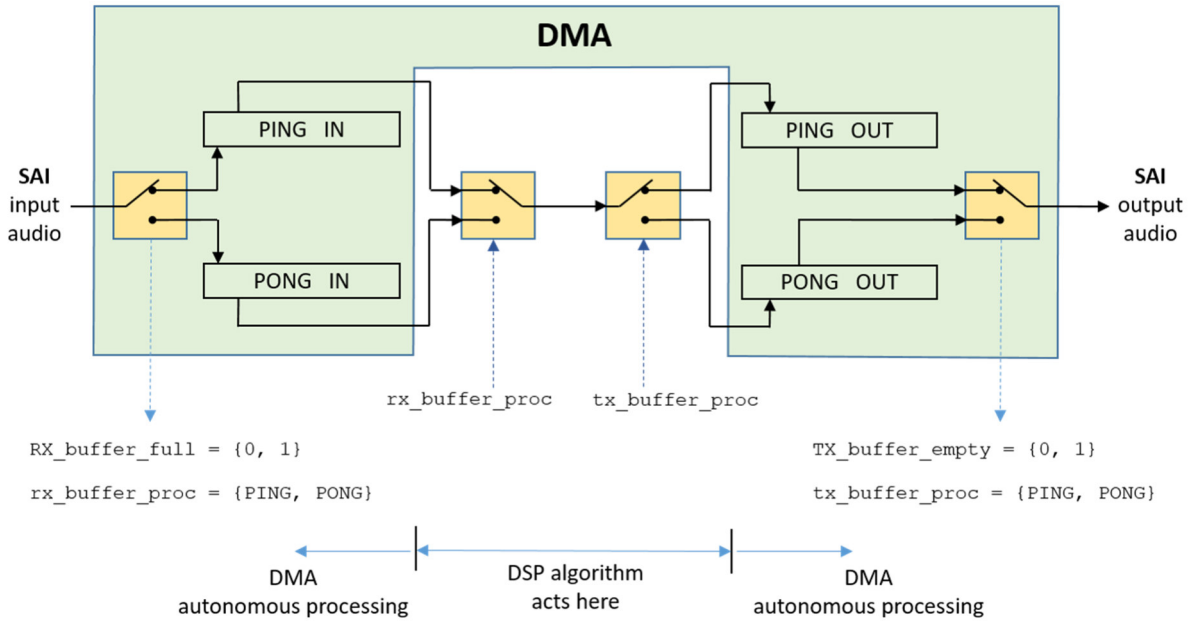

Figure 2: Schematic representation of the DMA operation in connection with `stm32f7_loop_dma.c`

3.1.1 Frame based processing

Rather than processing one sample at a time, as in previous laboratory experiments, signal processing algorithms may also be applied to blocks, or frames, of samples. This is called frame-based processing, or block-based processing. This is especially important in real-time signal processing and, therefore, requires a different approach to that used for input and output in most of the previous lab experiments. This approach requires the implementation of buffering, as it is illustrated in Figure 2, and to process blocks of samples, using buffer-based I/O interrupt, rather single sample-based I/O interrupt (as in previous laboratory experiments). Thus, DMA-based I/O is more suitable for frame-based processing and will be used in this lab experiment (and also in future lab experiments).

3.1.2 stm32f7_wm8994_init() DMA operation [this is imported from original ARM lab 5 -FFT]

The DMA in the STM32F746G is organized into unidirectional streams, two of which are used for this application. In function `stm32f7_wm8994_init()`, DMA stream 7 is configured to make DMA transfers between the Synchronous Audio Interface (SAI) peripheral and input buffers (arrays) in memory (alternately `PING_IN` and `PONG_IN`). It generates an interrupt when a transfer of `PING_PONG_BUFFER_SIZE` 32-bit words has completed. Each 32-bit word comprises two 16-bit sample values (LEFT and RIGHT channels). The value `PING_PONG_BUFFER_SIZE` is therefore equivalent to the number of sampling instants represented by one DMA transfer. The value of `PING_PONG_BUFFER_SIZE` is defined in file `stm32f7_wm8994_init.h`.

DMA stream 3 is configured to make DMA transfers between output buffers in memory (alternately `PING_OUT` and `PONG_OUT`) and the SAI peripheral. It too generates an interrupt when a transfer of `PING_PONG_BUFFER_SIZE` 32-bit words has completed.

Just for information purposes (you do not need to understand this for this laboratory experiment), four different interrupt service routines (functions) are involved in the DMA ping-pong buffering process. `BSP_AUDIO_OUT_TransferComplete_CallBack()`, `BSP_AUDIO_IN_TransferComplete_CallBack()`, `BSP_AUDIO_OUT_TransferCompleteM1_CallBack()`, and `BSP_AUDIO_IN_TransferCompleteM1_CallBack()`, defined in file `stm32f7_wm8994_init.c`, are associated with completion of DMA transfers from array `PING_OUT` to the SAI peripheral, from the SAI peripheral to array `PING_IN`, from array `PONG_OUT` to the SAI peripheral, and from the SAI peripheral to array `PONG_IN`, respectively.

3.1.3 stm32f7_loop_dma.c operation [this is imported from original ARM lab 5 -FFT]

The actions carried out in the routines mentioned in Section 3.1.2 are simply to toggle the values of variables `rx_buffer_proc` and `tx_buffer_proc` between `PING` and `PONG`, and to set flags `RX_buffer_full` and `TX_buffer_empty`, as it is suggested in Figure 2. Switching between buffers `PING_IN`, `PONG_OUT` and `PING_OUT` and `PONG_OUT` in the DMA streams is handled automatically by the DMA multi-buffering mechanism.

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++; // one sample regarding the LEFT channel
    *tx_buf++ = *rx_buf++; // one sample regarding the RIGHT channel
  }

  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 3: Listing of program `stm32f7_loop_dma.c`

Function `main()` waits until both `RX_buffer_full` and `TX_buffer_empty` flags are set, that is, until both DMA transfers have completed, before calling function `process_buffer()`. In program `stm32f7_loop_dma.c`, function `process_buffer()` simply copies the contents of the most recently filled input buffer (`PING_IN` or `PONG_IN`) to the most recently emptied output buffer (`PING_OUT` or `PONG_OUT`), according to the values of variables `rx_buffer_proc` and `tx_buffer_proc`. In general, frame-based processing will be carried out in function `process_buffer()` using the contents of the most recently filled input buffer as input, and writing output sample values to the most recently emptied output buffer.

DMA transfers will complete, and function `process_buffer()` will be called every `PING_PONG_BUFFER_SIZE` sampling instants and, therefore, any processing *must be completed within* `PING_PONG_BUFFER_SIZE` /fs seconds (fs represents the sampling frequency), that is, before the next DMA transfer completion.

The expected delay between input and output signals is `PING_PONG_BUFFER_SIZE*2/fs` seconds.

3.2 Experiments with `stm32f7_loop_dma_FunSP.c`

In this lab experiment, we will use the modified `main()` project file that is named `stm32f7_loop_dma_FunSP.c` and that is available on the Moodle platform. Its C code is listed next.

```
// stm32f7_loop_dma_FunSP.c

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

#define SOURCE_FILE_NAME "stm32f7_loop_dma_FunSP.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++)
  {
```

```
    if ( (i%1) == 0)  // either use (i%1) or (i%2)
    {
       *tx_buf++ = *rx_buf++;
       *tx_buf++ = *rx_buf++;
     }
     else
     {
        *tx_buf++ = (-1) * *rx_buf++;
        *tx_buf++ = (-1) * *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 4: Listing of program `stm32f7_loop_dma_FunSP.c`

The C code `stm32f7_loop_dma_FunSP.c` just adds a simple modification to the original code `stm32f7_loop_dma.c`. Based on this code, take a moment to identify what the differences are.

Question: what is the sampling frequency that this code specifies ? What is the Nyquist frequency ?

After unzipping it, take the `stm32f7_loop_dma_FunSP.c` file to the “src” directory that is located under folder:

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

Now, proceed as usual to start the Keil MDK-Arm development environment (µVision) and to replace the existing `main()` file in the existing project by the new `main()` file that is `stm32f7_loop_dma_FunSP.c` .

Remember that the directory where you can find 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.

Set the function generator to generate a sine wave having 5 Vpp and 1000 Hz. Using a "T" and a BNC-BNC cable, take the output of the function generator to CHAN1 of the oscilloscope.

As usual, connect the output of the sinusoidal signal generator (i.e. the function generator) to the (LEFT channel of the) LINE IN socket on the Discovery board (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 input voltage levels).

Then, using another BNC-BNC cable, take the LEFT channel of the STM32F746G LINE OUT output to the CHAN2 input of the oscilloscope.

Using the oscilloscope SETTINGS button and menu, make sure that the Vpp and frequency of both input and output signals are being measured in real-time.

Question 1 [1 pt / 10 **]**: As you vary the input frequency between low frequencies and the Nyquist frequency, what analog system is equivalent to the complete signal processing chain implemented on the STM32F7 kit ?

Now, stop the kit operation, exit the debugger environment and, on the editor environment, change the following line in the `stm32f7_loop_dma_FunSP.c` code:

```
if ( (i%1) == 0)
```

so that it becomes:

```
if ( (i%2) == 0)
```

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

As you increase the frequency of the input sinusoid, note the real-time measurements of the oscilloscope and find the sum of two frequencies: the frequency measured from the sinusoid at the input of the STM32F7 kit, and the frequency measured from the sinusoid at its output.

Question 2 [3 pt / 10 **]**: When the input frequency is 4 kHz, what is the output frequency ? When the input frequency is 12 kHz, what is the output frequency ? When the input frequency is 20 kHz, what is the output frequency ? How do you explain this behavior considering the implication of the above code modification (i.e., by replacing `(i%1)` by `(i%2)`) ?

Hint: the signal modification discussed here, as well as its frequency-domain impact, is discussed in the context of exercise 2 of the set of exercises proposed for week5 (09-13 Oct), and whose explanation on video is available on Moodle.

3.3 Representing time and frequency on the STM32F746G board

In this part of the lab experiment, we use an extended version of the code listed in Section 3.1.3 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` and that is available on the usual "src" directory.

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 5: 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 oscilloscope, or to the headphones if you happen to have a pair of these. 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 (try to say a few sustained vowels).

Now, change the parameter `INPUT_DEVICE_DIGITAL_MICROPHONE_2` to `INPUT_DEVICE_INPUT_LINE_1` and make sure that the sampling frequency is 8 kHz. Now, 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 input voltage levels).

You should see graphs on the LCD similar to those shown in Figure 6.

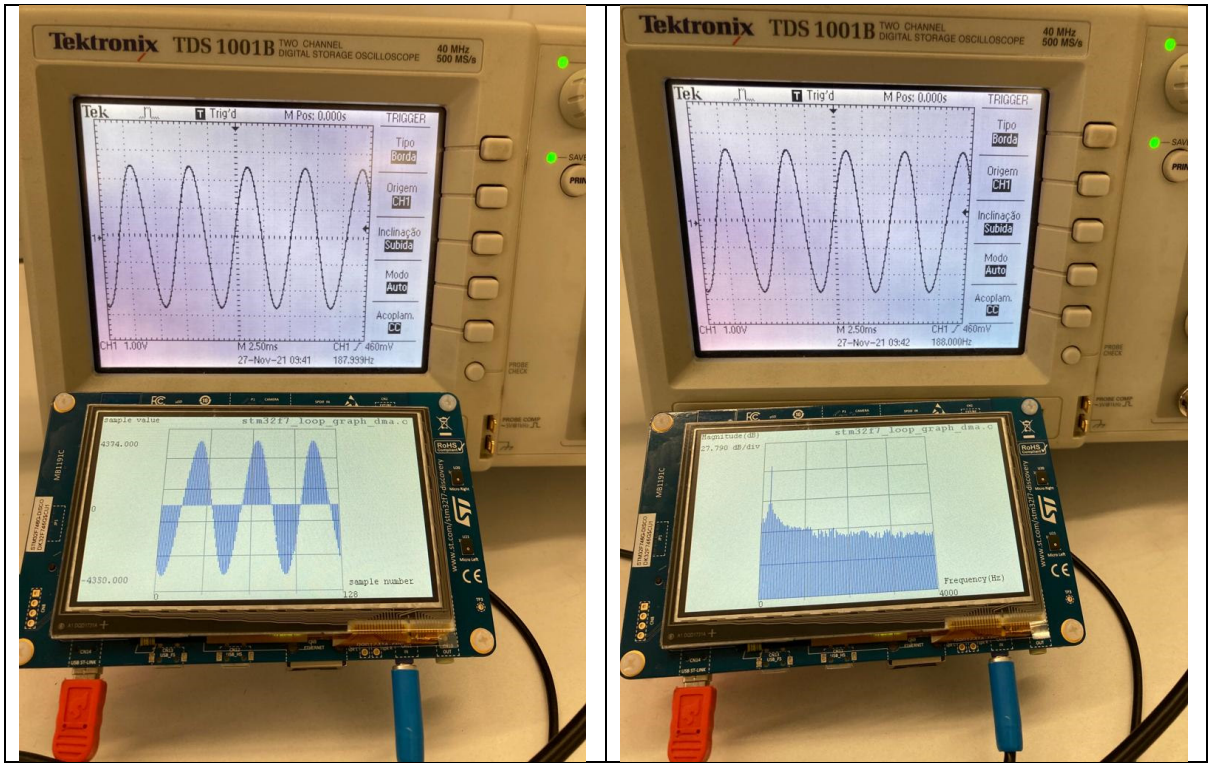

Figure 6: Graphical representation of 180 Hz sinusoidal signal input to program `stm32f7_loop_graph_dma.c` in the time-domain (figure o the left), and in the frequency domain (figure on the right). Sampling frequency is 8 kHz

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

Question 3 [2 pt / 10 **]**: Considering that the length of the DMA buffers is 256 samples, what is the lowest sinusoidal input wave whose graphical representation on the LCD screen of the STM32F7kit stands still, i.e. the represented wave of the LCD screen of the STM32F7kit does not glide to the left or right of the screen ? Show the sinusoidal representation for that particular frequency (**Note**: you may need to adjust the decimal part of the frequency).

Question 4 [2 pt / 10 **]**: Now, consider higher frequencies of the sinusoidal input signal whose graphical representation on the screen of the STM32F7kit becomes again still. How is that those frequencies relate to the lowest possible frequency identified in the previous question ?

Question 5 [2 pt / 10 **]**: When the input sinusoid signal is such that it generates a still representation on the STM32F7kit LCD screen, toggle the graphical representation to the frequency-domain so that you can see the spectral magnitude of the signal. Is this representation as expected ?

Question: for other “non-special” frequencies of the input sinusoid (as presumed in the previous question), the magnitude spectrum becomes different from that observed under the conditions of the previous question. How do you explain that ?

3.4 Hearing the effect of the DMA buffer delays [optional]

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 the `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.

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.

NOTE: Try to use headphones with a two-ring (stereo) mini-jack as it is illustrated in Figure 7. Three-ring mini-jacks may not be appropriate as they include an addition microphone signal that the STM32F746G board does not support.

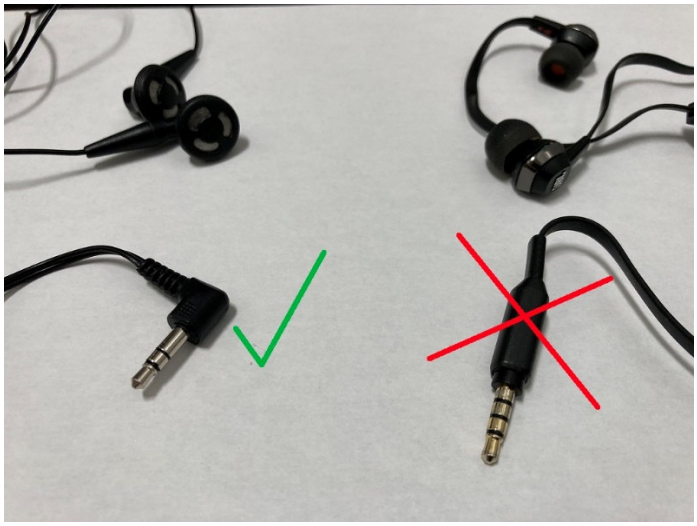

Figure 7: A two ring mini-jack (stereo) should be used.

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

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 time-domain and frequency-domain graphical representation on the STM32F746G LCD are very convenient features that we will use in future 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