L.EEC025 - Fundamentals of Signal Processing (FunSP)

2021/2022 – 1st semester

Week03, 02 Nov 2021

Objectives:

-getting started with the DSP Education kit

- **viewing program output**

DSP Education Kit

LAB 2
Viewing Program Output

Issue 1.0

Contents

1 Introduction

1.1 Lab overview

The STM32F746G Discovery board is a low-cost development platform featuring a 212 MHz Arm Cortex-M7 floating-point processor. It connects to a host PC via a USB A to mini-b cable and uses the ST-LINK/V2 in-circuit programming and debugging tool. The Keil MDK-Arm development environment, running on the host PC, enables software written in C to be compiled, linked, and downloaded to run on the STM32F746G Discovery board. Real-time audio I/O is provided by a Wolfson WM8994 codec included on the board.

This laboratory exercise introduces the use of the STM32F746G Discovery board and several of the procedures and techniques that will be used in subsequent laboratory exercises.

2 Requirements

To carry out this lab, you will need:

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

2.1.1 STM32F746G Discovery board

An overview of the STM32F746G Discovery board can be found in the Getting Started Guide.

3 Basic Digital Signal Processing System

A basic DSP system that is suitable for processing audio frequency signals comprises a digital signal processor and analogue interfaces as shown in Figure 1. The STM32F746G Discovery board provides such a system, using a Cortex-M7 floating point processor and a WM8994 codec.

The term codec refers to the *coding* of analogue waveforms as digital signals and the *decoding* of digital signals as analogue waveforms. The WM8994 codec performs both the Analogue to Digital Conversion (ADC) and Digital to Analogue Conversion (DAC) functions shown in Figure 1.

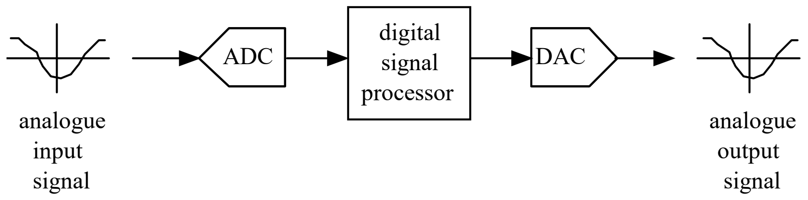


Figure 1: Basic digital signal processing system

Program code may be developed, downloaded, and run on the STM32F746G Discovery board using the *Keil MDK-Arm* integrated development environment. You will not be required to write C programs from scratch, but you will learn how to compile, link, download, and run the example programs provided, and in some cases, make minor modifications to their source files.

You will learn how to use a subset of the features provided by MDK-Arm in order to do this (using the full capabilities of MDK-Arm is beyond the scope of this set of laboratory exercises). The emphasis of this set of laboratory exercises is on the digital signal processing concepts implemented by the programs.

Most of the example programs are quite short, and this is typical of real-time DSP applications. Compared with applications written for general purpose microprocessor systems, DSP applications are more concerned with the efficient implementation of relatively simple algorithms. In this context, efficiency refers to speed of execution and the use of resources such as memory.

The examples in this document introduce some of the features of *MDK-Arm* and the STM32F746G Discovery board. In addition, you will learn how to use *MATLAB in* order to analyze audio signals.

4 Real-Time Sine Wave Generation

4.1 Program operation (done aready in week01, proceed to 4.2)

The C source file `stm32f7_sine_lut_intr.c`, shown in the code snippet below, generates a sinusoidal signal using interrupts and a table lookup method.

```
// stm32f7_sine_lut_intr.c


#include "stm32f7_wm8994_init.h"

#include "stm32f7_display.h"


#define SOURCE_FILE_NAME "stm32f7_sine_lut_intr.c"

#define LOOPLENGTH 8


extern int16_t rx_sample_L;

extern int16_t rx_sample_R;

extern int16_t tx_sample_L;

extern int16_t tx_sample_R;


int16_t sine_table[LOOPLENGTH] = {0, 7071, 10000, 7071, 0, -7071, -10000, -7071};

int16_t sine_ptr = 0;  // pointer into lookup table


void BSP_AUDIO_SAI_Interrupt_CallBack()

{

// when we arrive at this interrupt service routine (callback)

// the most recent input sample values are (already) in global variables

// rx_sample_L and rx_sample_R

// this routine should write new output sample values in

// global variables tx_sample_L and tx_sample_R


  BSP_LED_On(LED1);

  tx_sample_L = sine_table[sine_ptr];

  sine_ptr = (sine_ptr+1)%LOOPLENGTH;
  tx_sample_R = tx_sample_L;
  BSP_LED_Off(LED1);

  return;
}

int main(void)
{
  stm32f7_wm8994_init(AUDIO_FREQUENCY_8K,
                      IO_METHOD_INTR,
                      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,
                      GRAPH);
  plotSamples(sine_table, LOOPLENGTH, 32);
  while(1){}
}
```

An eight-point lookup table is initialized using the array `sine_table` such that the value of `sine_table[i]` is equal to

$$\text{sine\_table}[i] = 10000\, sin(\,(2\pi i/8) + \varphi)$$

where in this case, $\phi = 0$. The `LOOPLENGTH` values in array `sine_table` are samples of exactly one cycle of a sinusoid.

Just as in the previous examples, in function `main()`, initialization function `stm32f7_wm8994_init()` is called. This configures processor and codec such that the WM8994 will sample, and interrupt the processor, at a frequency determined by the parameter value `AUDIO_FREQUENCY_8K`, i.e., in this case at 8 kHz. Interrupts will occur every 0.125 ms.

Following the call to function `stm32f7_wm8994_init()`, function `main()` enters an endless loop, doing nothing but waiting for interrupts (which will occur once per sampling period).

On interrupt, the interrupt service routine function `BSP_AUDIO_SAI_Interrupt_CallBack()` is called, and in that routine, the most important program statements are executed: the sample values read from array `sine_table` are written to both channels to the DAC and the index variable `sine_ptr` is incremented to point to the next value in the array.

The 1 kHz frequency of the sinusoidal output signal corresponds to the eight samples per cycle output at a rate of 8 kHz.

The WM8994 DAC is effectively a low pass reconstruction filter that interpolates between output sample values to give a continuous sinusoidal analogue output signal as shown in Figure 2. This will be explained further in the next lab exercise.

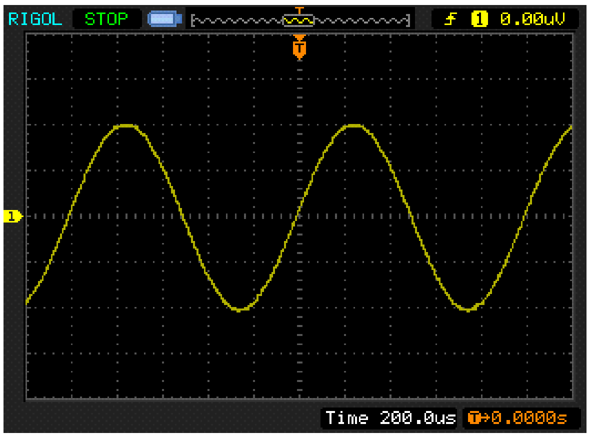


Figure 2: Analog output generated by program `stm32f7_sine_lut_intr.c`

When you run the program, you should see a start screen on the LCD as shown in Figure 3. Press the blue user pushbutton to continue, and you should see on the LCD a graphical representation of the sequence of discrete sample values being written to the DAC (Figure 4). The sample values are represented as bars in the graph on the LCD to emphasize that it is the discrete sample values written to the DAC that are being shown and not the continuous-time signal output by the DAC. Connect one channel of the audio card HEADPHONE OUT output to an oscilloscope and verify that the output signal is a 1 kHz sinusoid using both time-domain and frequency-domain oscilloscope displays.

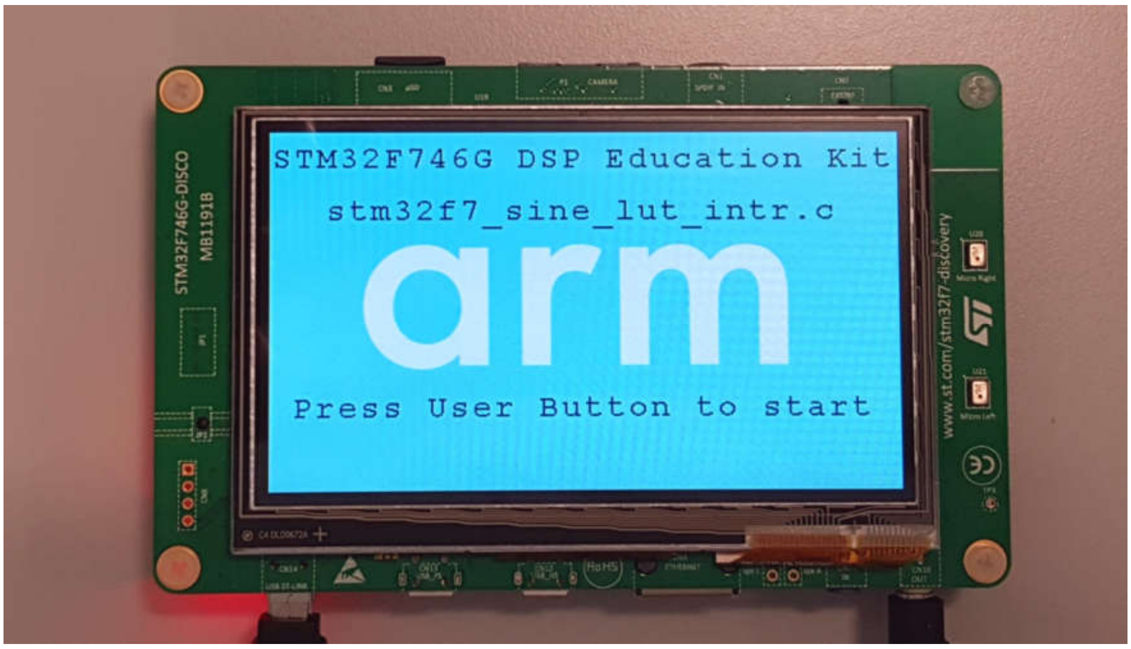

STM32F746G DSP Education Kit
stm32f7_sine_lut_intr.c
arm
Press User Button to start
STM32F746G-DISCO
MB1191B
www.st.com/stm32f7-discovery

Figure 3: Start screen for program stm32f7_sine_lut_intr.c

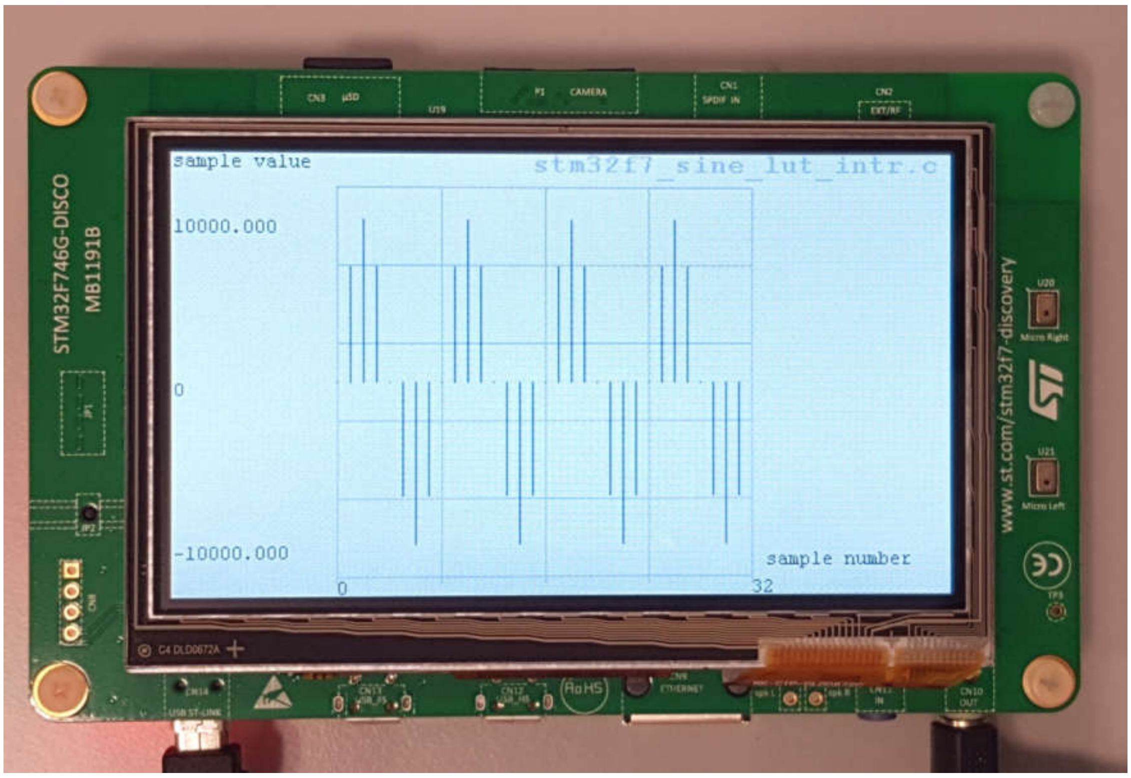


Figure 4: Graphical representation of first 32 sample values output by program stm32f7_sine_lut_intr.c

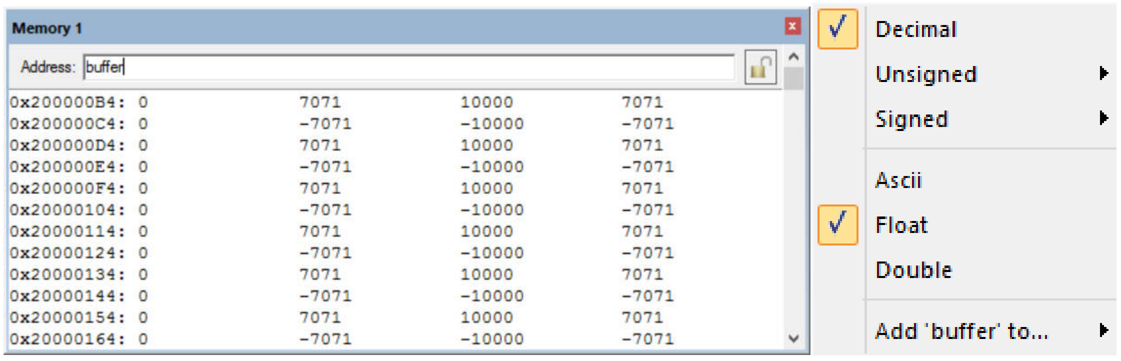
4.2 Viewing program output using MATLAB (sinusoid)

To view your program output in Matlab, you can first store the output values into a file and then use Matlab to load the values from the saved file.

`stm32f7_sine_lut_buf_intr.c` shows how to store the output values, it is very similar to program `stm32f7_sine_lut_intr.c`, but it also stores the most recent `BUFFER_LENGTH` number of output values in the array `buffer`. Array `buffer` is of type `float32_t` for compatibility with the *MATLAB* function that will be used to view its contents.

To save the program output into a file and view them in Matlab, follow these steps:

1. Run the program and press the user button to start the program.
2. Halt it by clicking on the ***Stop*** toolbar button in the MDK_Arm debugger.
3. Type the variable name **`buffer`** as the ***Address*** in the debugger's ***Memory 1*** window. Right-click on the ***Memory 1*** window and set the displayed data type to ***Decimal*** and ***Float*** as shown in Figure 5.



```
Memory 1
Address: buffer
0x200000B4: 0      7071     10000    7071
0x200000C4: 0     -7071    -10000   -7071
0x200000D4: 0      7071     10000    7071
0x200000E4: 0     -7071    -10000   -7071
0x200000F4: 0      7071     10000    7071
0x20000104: 0     -7071    -10000   -7071
0x20000114: 0      7071     10000    7071
0x20000124: 0     -7071    -10000   -7071
0x20000134: 0      7071     10000    7071
0x20000144: 0     -7071    -10000   -7071
0x20000154: 0      7071     10000    7071
0x20000164: 0     -7071    -10000   -7071
```

Figure 5: Memory 1 window showing the contents of array `buffer`

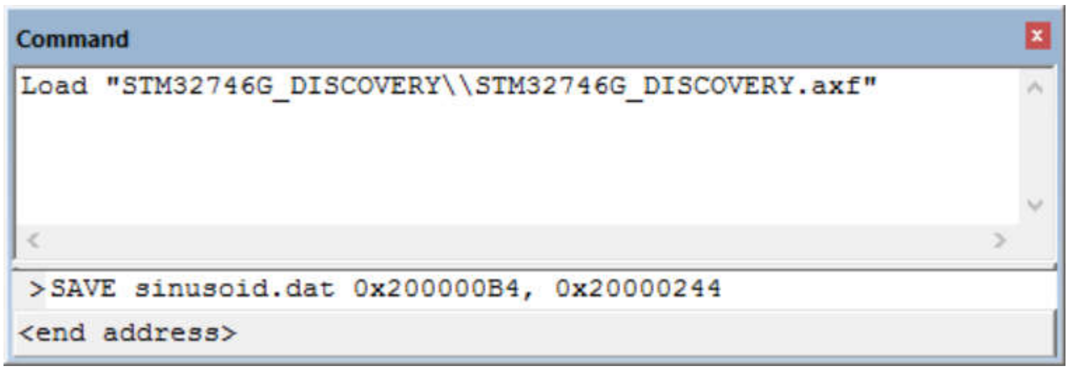
   The start address of array `buffer` will be displayed in the top left-hand corner of the window.

4. Use the following command at the prompt in the debugger's ***Command*** window to save the contents of array **`buffer`** to a file in your project folder.

   `SAVE <filename> <start address>, <end address>`

   The end address should be the start address plus 0×190 (bytes) representing 100 32-bit sample values. For example,

   ```
   SAVE sinusoid.dat 0x200000B4, 0x20000244
   ```


Command

```
Load "STM32746G_DISCOVERY\\STM32746G_DISCOVERY.axf"
```

```
>SAVE sinusoid.dat 0x200000B4, 0x20000244
<end address>
```

Figure 6: Saving data to file in MDK-Arm

5. Launch *MATLAB* and run the *MATLAB* function `stm32f7_logfft.m` (provided with the DSP Education Kit in **General_Matlab_Files**) to obtain a graphical representation of the contents of the buffer. The *MATLAB* function will require you to input some information, such as the saved `.dat` filename (full path) and sampling frequency.

NOTE: Matlab function `stm32f7_logfft.m` is available on Moodle. You also need function `hexsingle2num.m` which should be co-located with `stm32f7_logfft.m` .

4.3 Viewing program output using MATLAB (noise)

Repeat 4.2 but this time using the provided C file named `stm32f7_sine_lut_buf_intr_modOCT2021.c`, which you should copy to the `Examples\DSP Education Kit\Src` directory (in case it is not yet there). This code is a modified version of the previous code in the sense that it calls function `prand()` that generates pseudorandom sample values using the Park-Miller algorithm (a random number generator). Thus, the left channel outputs a sinusoid whereas the right channel outputs noise.

```c
// stm32f7_sine_lut_buf_intr_modOCT2021.c


#include "stm32f7_wm8994_init.h"

#include "stm32f7_display.h"


#define SOURCE_FILE_NAME "stm32f7_sine_lut_buf_intr_modOCT2021.c"

#define LOOPLENGTH 8

#define BUFFER_LENGTH 1000 // was 100


extern int16_t rx_sample_L;
extern int16_t rx_sample_R;

extern int16_t tx_sample_L;

extern int16_t tx_sample_R;


int16_t sine_table[LOOPLENGTH] = {0, 7071, 10000, 7071, 0, -7071, -10000, -7071};

int16_t sine_ptr = 0;  // pointer into lookup table

float32_t buffer[BUFFER_LENGTH];

int16_t buf_ptr = 0;   // pointer into buffer


void BSP_AUDIO_SAI_Interrupt_CallBack()

{

// when we arrive at this interrupt service routine (callback)

// the most recent input sample values are (already) in global variables

// rx_sample_L and rx_sample_R

// this routine should write new output sample values in

// global variables tx_sample_L and tx_sample_R


  tx_sample_L = sine_table[sine_ptr];

  tx_sample_R = prand();

      //tx_sample_R = tx_sample_L;

  buffer[buf_ptr] = tx_sample_R;

      // buffer[buf_ptr] = tx_sample_L;

      sine_ptr = (sine_ptr+1)%LOOPLENGTH;

      buf_ptr = (buf_ptr+1)%BUFFER_LENGTH;


  BSP_LED_Toggle(LED1);


  return;

}


int main(void)

{

  stm32f7_wm8994_init(AUDIO_FREQUENCY_8K,
                        IO_METHOD_INTR,
                        INPUT_DEVICE_DIGITAL_MICROPHONE_2,
                        OUTPUT_DEVICE_HEADPHONE,
                        WM8994_HP_OUT_ANALOG_GAIN_6DB,
                        WM8994_LINE_IN_GAIN_0DB,
                        WM8994_DMIC_GAIN_9DB,
                        SOURCE_FILE_NAME,
                        GRAPH);
        plotSamples(sine_table, LOOPLENGTH, 32);
  while(1){}
}
```

Replace the code file in 4.2 by this new file `stm32f7_sine_lut_intr_modOCT2021.c` and build and run the program again.

Repeat the above procedure to save the contents of array **`buffer`** to a file (`noise.dat)` in your project folder.

> **`SAVE <filename> <start address>, <end address>`**
>
> The end address should be the start address plus 0xFA0 (bytes) representing 1000 32-bit sample values. For example,
>
> `SAVE noise.dat 0x200000B4, 0x20001054`

Proceed to repeat the above file analysis by running the *MATLAB* function `stm32f7_logfft.m` .

Does it look like expected ?

The following is to be carried out of class: how would you characterize the noise that is generated by the DSP Education kit pseudorandom generator ? Adapt the Matlab code to check its auto-correlation function and its PDF (Probability Density Function).

5 Conclusions

At the end of this exercise, you should have become familiar with a simple procedure to import to Matlab data that is generated in the DSP Education Kit. This will be used subsequent lab exercises.

6 Additional References

Link to Board information and resources:

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