

L.EEC025 - Fundamentals of Signal Processing (FunSP)

2025/2026 – 1st semester

Week02, 22 Sept 2025

Objectives:

-getting started with the DSP Education kit

- experiment 7 (LUT-based Real-Time Sine Wave Generation)
- experiment 4 (Basic Analogue Input & Output Using the STM32F746G Disco Board)
- experiment 5 (Delaying the Signal)

DSP Education Kit

LAB 1

Analog Input and Output

Issue 1.0

Contents

1	Introduction.....	1
1.1	Lab overview	1
2	Requirements	1
2.1.1	Overview of STM32F746G Discovery board	1
3	Basic Digital Signal Processing System.....	3
4	Basic Analogue Input and Output Using the STM32F746G Discovery Board (this experiment is to be executed after the one described in Section 7)	4
4.1	Program operation of stm32f7_loop_intr.c.....	5
4.2	Running the program	5
5	Delaying the Signal (NOTE: only if time permitting)	10
6	Creating a Fading Echo Effect (NOTE: we skip this section).....	11
6.1.1	Exercise	12
7	Real-Time Sine Wave Generation.....	13
7.1	Loading a project and running the project	13
7.2	Program operation.....	16
8	Conclusions.....	20
9	Additional References.....	20

1 Introduction

1.1 Lab overview

In the labs for this course, we will use the STM32F7 Discovery Kit to practice digital signal processing concepts and explore some of their applications using advanced hardware.

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 Overview of STM32F746G Discovery board

The STM32F746G Discovery board features a Cirrus Logic WM8994 stereo audio codec, which is accessed via I2C for control and I2S, using the STM32F746G microcontroller's serial audio interface (SAI) peripheral, for data. Analogue input and output signals are accessible via two three-pole (TRS) 3.5 mm jack sockets (LINE IN (CN11) and HEADPHONE OUT(CN10)).

As configured for these exercises, the WM8994 converts an analogue input signal into 16-bit signed integer sample values, and the DAC converts 16-bit signed integer sample values into an analogue output signal.

Additionally, the WM8994 has a digital microphone interface, and the STM32F746G Discovery board provides two MEMS microphones as input devices.

Some of the hardware features of the STM32F746G Discovery board are highlighted in Figure 0.1.

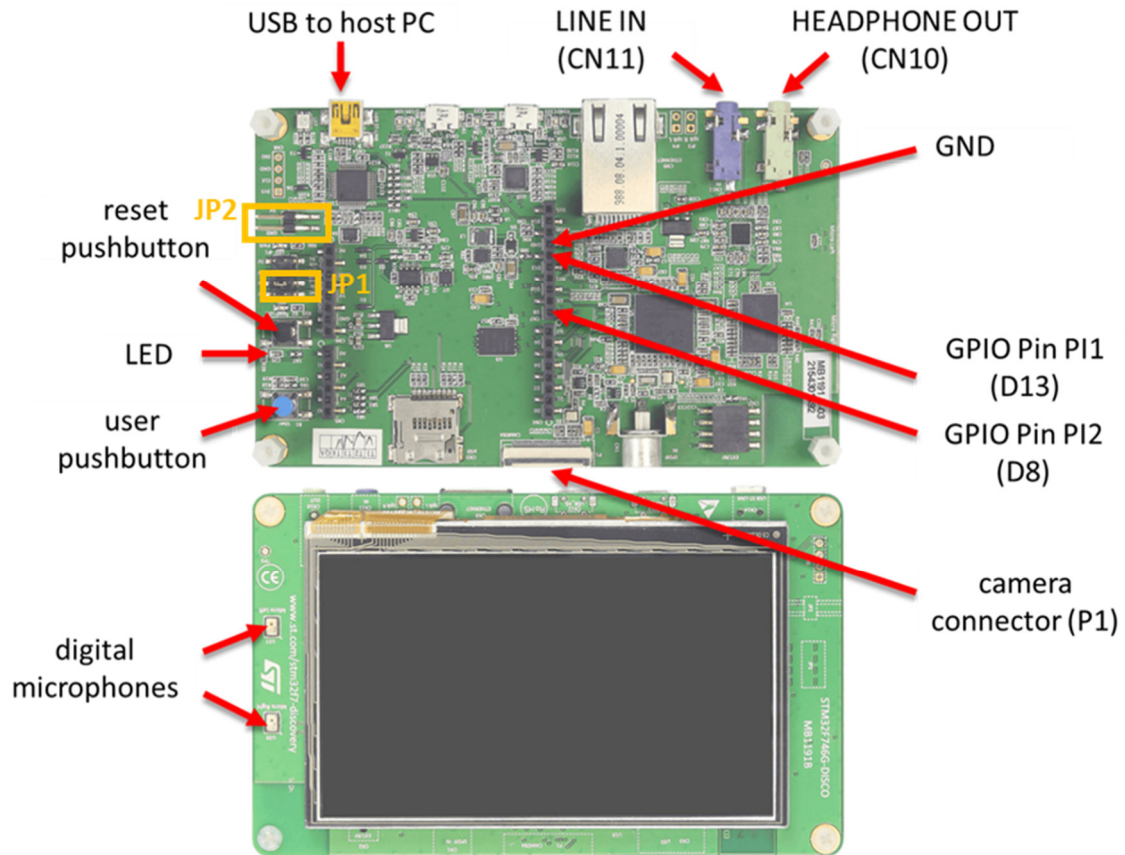


Figure 0.1: STM32F746G Discovery board

Note: For this course lab exercises, JP1 should be set to “5V link USB + 5V.” JP2 should be left open (all STM32F746G Discovery Kits in the FSP labs have been configured this way already)

In order to prevent hazardous electrical contacts, in the lab, a casing protects the STM32F746G Discovery board as it is illustrated in Figure 0.2.



Figure 0.2: Actual lab casing of the STM32F746G Discovery board

Very important notice: when plugging the source generator to the input of the *kit*, please 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.

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 2. 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 2.

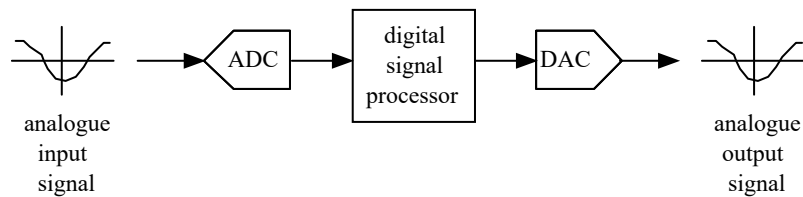


Figure 2: Basic digital signal processing system

Program code may be developed, downloaded, and run on the STM32F746G Discovery board using the **Keil MDK-Arm integrated development environment** (IDE). 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 Basic Analogue Input and Output Using the STM32F746G Discovery Board (this experiment is to be executed after the one described in Section 7)

The code snippet below shows a source file for a program that simply copies input samples read from two digital microphones mounted on the board and connected to the WM8994 codec and the WM8994 DAC. In effect, the program connects the digital microphones to the headphone output socket on the board. This simple program is important because many of the other example programs that will be used in subsequent laboratory exercises use the same *interrupt-based, real-time* structure. It is worth taking the time to ensure that you understand how program `stm32f7_loop_intr.c` works, which will also be explained in this document.

In addition, this example describes the *MDK-Arm* development environment and the editing, compiling, linking, and downloading procedures that you will use again for subsequent examples.

```
// stm32f7_loop_intr.c

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

#define SOURCE_FILE_NAME "stm32f7_loop_intr.c"

extern int16_t rx_sample_L;
extern int16_t rx_sample_R;
extern int16_t tx_sample_L;
extern int16_t tx_sample_R;

void BSP_AUDIO_SAI_Interrupt_CallBack()
{
    // when we arrive at this interrupt service routine
    // 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 = rx_sample_L;
    tx_sample_R = rx_sample_R;
    BSP_LED_Toggle(LED1);
    return;
}

int main(void)
{
    stm32f7_wm8994_init(AUDIO_FREQUENCY_48K,
                        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_17DB,
                        SOURCE_FILE_NAME,
                        NOGRAPH);

    while(1) {}
}
```

4.1 Program operation of stm32f7_loop_intr.c

The operation of program `stm32f7_loop_intr.c` is as follows.

In function `main()`, an initialization function `stm32f7_wm8994_init()` is called. This configures the STM32F746G processor and WM8994 codec such that the codec will read (left and right channel) sample values from the digital microphones and interrupt the processor at a sampling frequency determined by the parameter `AUDIO_FREQUENCY_48K` passed to the function.

Parameter `INPUT_DEVICE_DIGITAL_MICROPHONE_2` specifies that input to the WM8994 will come from the digital microphones on the STM32F746G Discovery board.

Parameter `IO_METHOD_INTR` passed to function `stm32f7_wm8994_init()` determines that interrupt, as opposed to DMA-based I/O, will be used by the program.

Parameter `OUTPUT_DEVICE_HEADPHONE` is redundant insofar as the headphone socket (CN10) is the only audio output currently supported by the DSP Education Kit.

Parameters `WM8994_HP_OUT_ANALOG_GAIN_6DB`, `WM8994_LINE_IN_GAIN_0DB`, and `WM8994_DMIC_GAIN_17DB` concern the configuration of programmable gain blocks in the signal path through the codec.

Parameters `SOURCE_FILE_NAME` and `NOGRAPH` influence what will be shown on the Discovery board's LCD.

There is no need to understand every detail of the initialization carried out by function `stm32f7_wm8994_init()`. After it has been called, interrupts generated by the Serial Audio Interface (SAI) peripheral in the STM32F746G microcontroller (to which the WM8994 codec is connected) will be enabled, and each time an interrupt occurs, the interrupt service routine function `BSP_AUDIO_SAI_Interrupt_Callback()` will be called. One interrupt will occur per sampling period, and both left and right channel samples are processed in one call to function `BSP_AUDIO_SAI_Interrupt_Callback()`.

Following initialization, function `main()` enters an endless `while()` loop, doing nothing but waiting for interrupts.

When function `BSP_AUDIO_SAI_Interrupt_Callback()` is called, new input sample values (from the WM8994 codec¹) may be read as variables `rx_sample_L` and `rx_sample_R`, and sample values written to variables `tx_sample_L` and `tx_sample_R` will be written to the WM8994 DAC at the next sampling instant.

¹Input sample values may have come either from the analogue LINE IN socket (CN11) on the Discovery board, via the WM8994 ADC, or from the two digital microphones on the Discovery board, via a digital interface on the WM8994.

4.2 Running the program

The following steps assume that you have followed all the steps described in **Section 7**.

To run the `stm32f7_loop_intr.c` program, follow these steps:

1. If the μ Vision 5 project **DSP_Education_Kit** is not yet open, open it by double-clicking on its icon, as indicated in **Section 7.1 b)**.
2. **Right-click** on the **STM32F746_DISCOVERY** folder in the **Project** pane and select **Manage Project Items**, after which you should get a window like the one shown below:

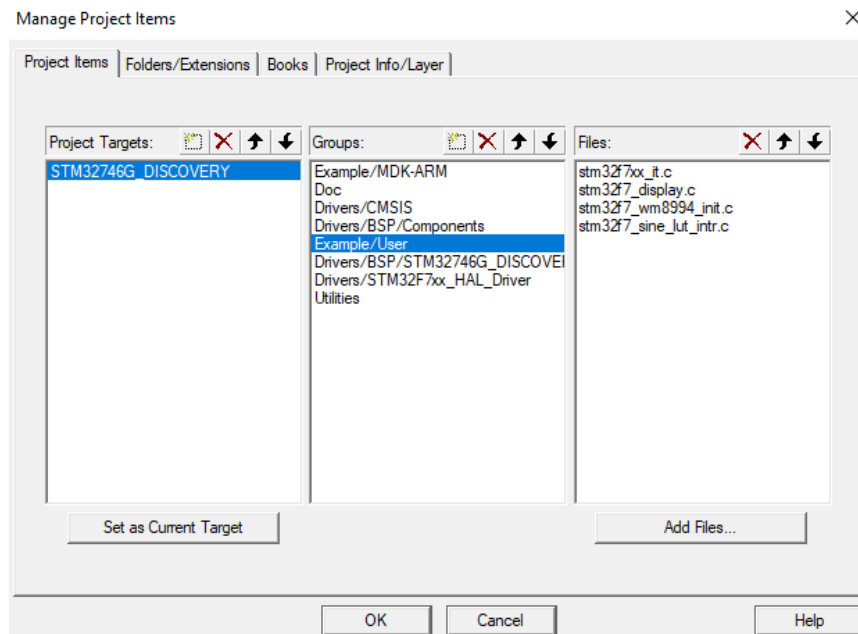
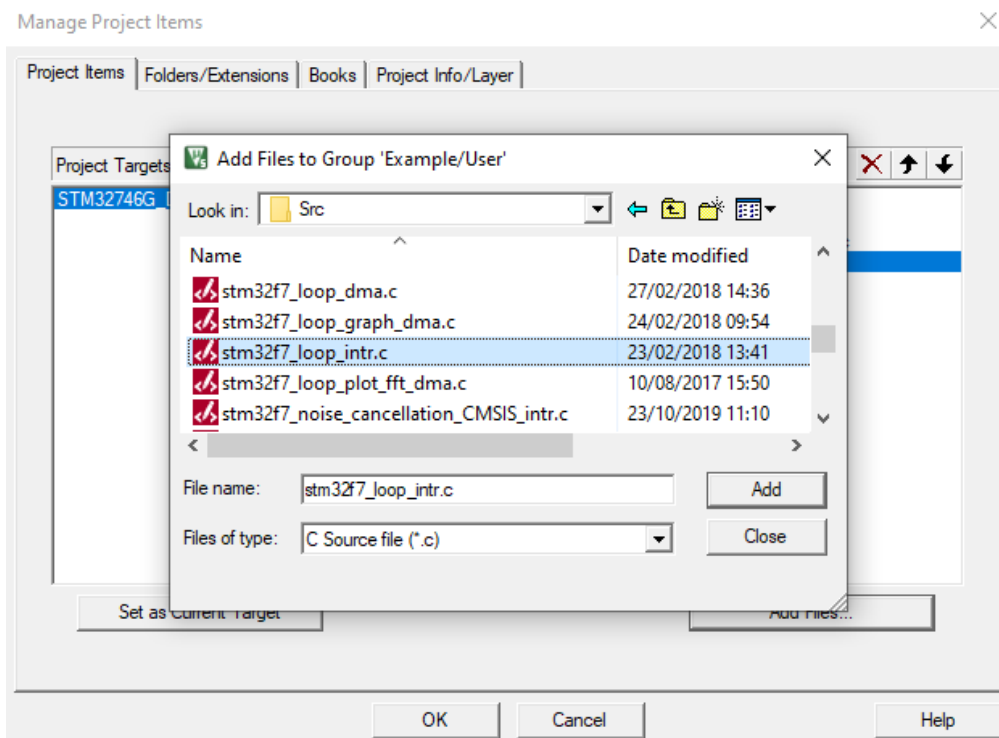
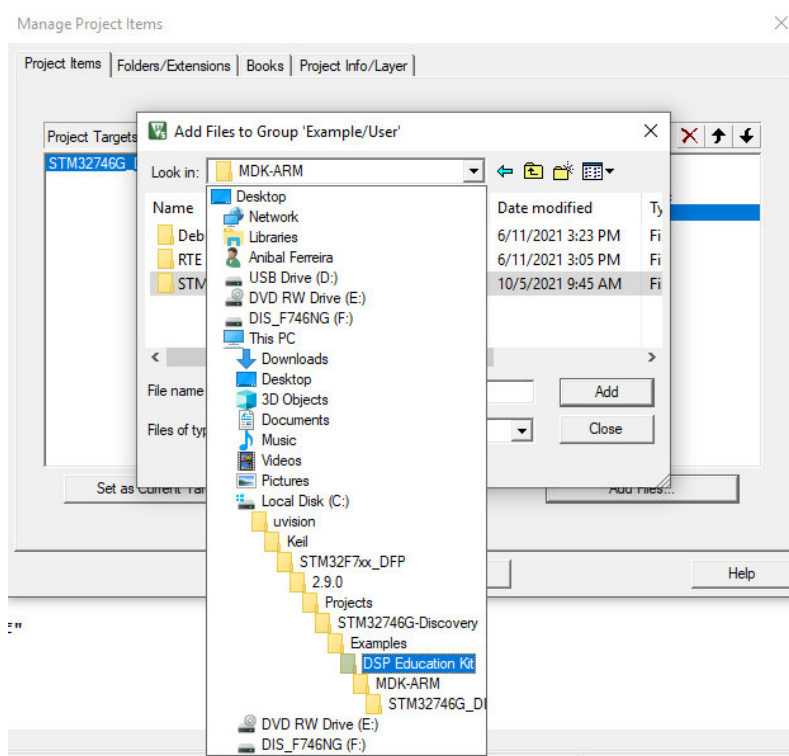
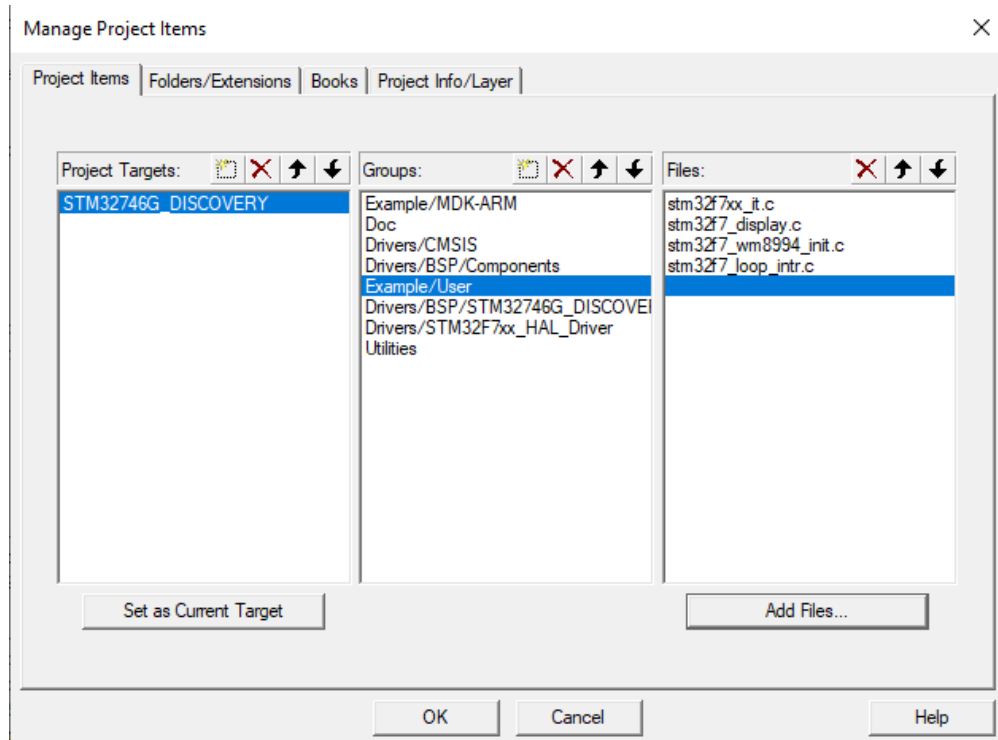


Figure 3: Screenshot of Manage Project Items

3. Delete `stm32f7_sine_lut_intr.c` using the delete icon  on the top right of the **Files** pane and then click on **Add Files**.
4. Find `stm32f7_loop_intr.c` in the **DSP Education Kit\Src** folder and add it to the project. Click **OK**. This is illustrated in the following two images.





5. You should now see a project structure like that shown in the following figure.

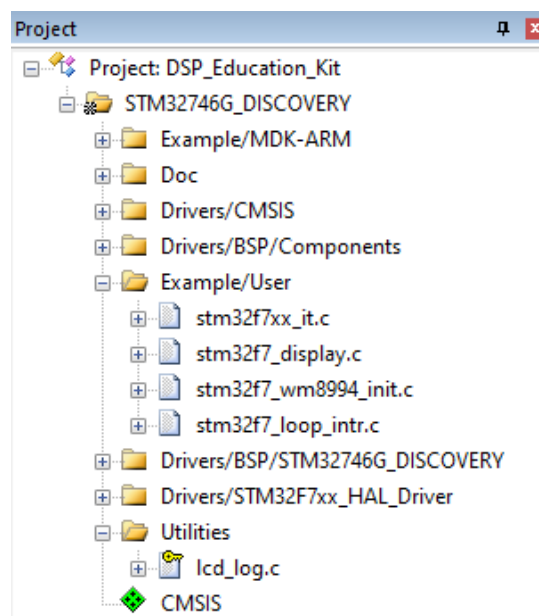


Figure 4: Screenshot of MDK-Arm showing the DSP_Education_Kit project

Note: Files `stm32f7_loop_intr.c`, `stm32f7_wm8994_init.c`, `stm32f7xx_it.c`, and `stm32f7_display.c` are supplied as part of the DSP Education Kit. Other files making up the project shown in Figure 4 are part of the STM32F746 Discovery board DFP Software Pack.

6. Connect the STM32F746 Discovery board to the host PC using a USB A to mini-b cable.
7. Plug the headphones into the headset jack socket (CN10) on the board.
8. Build the project by selecting the **Project > Build target** or by clicking on the **Build** toolbar button .
9. After successfully building the project with no errors, switch to the debugger mode (and download the executable code into flash memory) by clicking on the **Start/Stop Debug Session** toolbar button .
10. Once the **Debugger View** has appeared, click on the **Run** toolbar button .
11. Once the program is running, you should see a start screen on the LCD on the board as shown in Figure 5. You should be able to hear sounds picked up by the digital microphones on the STM32F746 Discovery board (micro right and micro left on the right side of the LCD screen as shown in Figure 5). Depending on the characteristics of the headphones you are using, the sound may be loud or quiet. If you cannot hear anything, try blowing gently onto the microphones.

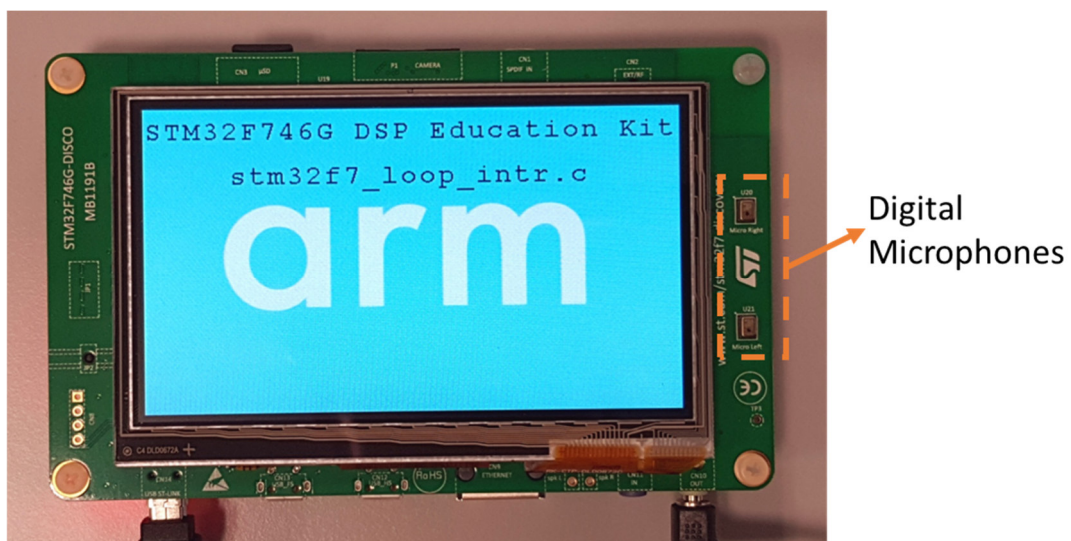


Figure 5: Start screen for program `stm32f7_loop_intr.c`

Optional: If you would like to use an external microphone instead of the microphones on the board, you can pass the parameter `INPUT_DEVICE_INPUT_LINE_1` (instead of `INPUT_DEVICE_DIGITAL_MICROPHONE_2`) to function `stm32f7_wm8994_init()`, you can listen to a signal input either via the LINE IN (CN11) socket on the board or via the digital microphones on the Discovery board. You can do this by editing the source file `stm32f7_loop_intr.c`, re-building the project, downloading, and running the program.

5 Delaying the Signal (NOTE: only if time permitting)

Some simple, yet striking, effects can be achieved simply by delaying the samples as they pass from input to output. Program `stm32f7_delay_intr.c` demonstrates this. In order to run this program simply repeat steps 1-10 of Section 4.2. In step 3, delete file `stm32f7_loop_intr.c` and, in step 4, look for file `stm32f7_delay_intr.c` and add it to the project.

A delay line is implemented using the array `buffer` to store samples as they are read from the digital microphones. Once the array is full, the pointer `bufptr` is reset and program overwrites the oldest stored input sample with the newest input sample. Just prior to overwriting the oldest stored input sample in `buffer`, that sample is retrieved, added to the current input, and written to the WM8994 DAC. The length of the delay is determined by the value of the constant `DELAY_BUF_SIZE`. As supplied, this is equal to 24000 samples, corresponding to a delay of 500 ms at a sampling rate of 48 kHz.

The following code snippet shows the source code of `stm32f7_delay_intr.c`.

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

#define SOURCE_FILE_NAME "stm32f7_delay_intr.c"
#define DELAY_BUF_SIZE 24000

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 buffer[DELAY_BUF_SIZE];
int16_t bufptr = 0;

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
    int16_t delayed_sample;

    delayed_sample = buffer[bufptr];
    tx_sample_L = delayed_sample + rx_sample_L;
    buffer[bufptr] = rx_sample_L;
    bufptr = (bufptr+1) % DELAY_BUF_SIZE;
    tx_sample_R = tx_sample_L;

    BSP_LED_Toggle(LED1);
}
```

```

    return;
}

int main(void)
{
    stm32f7_wm8994_init(AUDIO_FREQUENCY_48K,
                        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_17DB,
                        SOURCE_FILE_NAME,
                        NOGRAPH);

    while(1){}
}

```

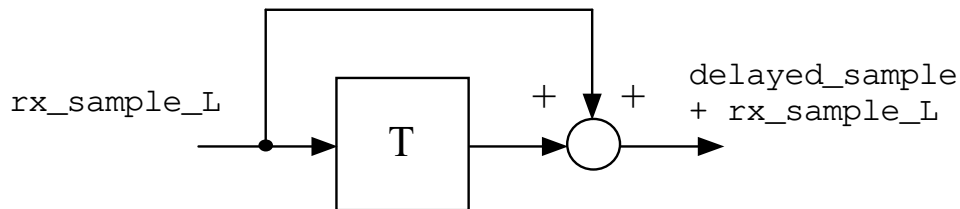


Figure 6: Block diagram representation of program *stm32f7_delay_intr.c*

6 Creating a Fading Echo Effect (**NOTE: we skip this section**)

By feeding back a fraction of the output of the delay line to its input, a fading echo effect can be realized. Program *stm32f7_echo_intr.c*, shown in the following code snippet, does this.

```

// stm32f7_echo_intr.c

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

#define SOURCE_FILE_NAME "stm32f7_echo_intr.c"
#define DELAY_BUF_SIZE 6000
#define GAIN 0.6f

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 buffer[DELAY_BUF_SIZE];
int16_t bufptr = 0;

```

```

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
    int16_t delayed_sample;

    delayed_sample = buffer[bufptr];
    tx_sample_L = delayed_sample + rx_sample_L;
    buffer[bufptr] = rx_sample_L + delayed_sample*GAIN;
    bufptr = (bufptr+1) % DELAY_BUF_SIZE;
    tx_sample_R = 0;

    BSP_LED_Toggle(LED1);

    return;
}

int main(void)
{
    stm32f7_wm8994_init(AUDIO_FREQUENCY_48K,
                        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_17DB,
                        SOURCE_FILE_NAME,
                        NOGRAPH);

    while(1){}
}

```

6.1.1 Exercise

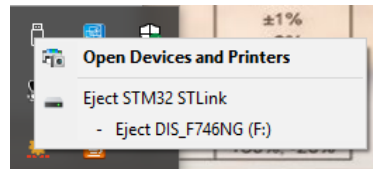
Experiment with different values of the constants `DELAY_BUF_SIZE` and `GAIN` (the delay in seconds is equal to `DELAY_BUF_SIZE` divided by the sampling frequency in Hz, and the fraction of the delayed signal fed back is equal to `GAIN`.)

1. What would happen if the value of `GAIN` were made greater than or equal to 1?
2. Study the program listing in `stm32f7_echo_intr.c` and, with reference to Figure 6, draw a block diagram of the system it implements in the space provided below. In the space below that, sketch what you think its response to a unit impulse at time $t = 0$ would be (with a `GAIN` of 0.6 and a `DELAY_BUF_SIZE` size of 2000 samples).

Block diagram representation of program `stm32f7_echo_intr.c`:

6.1.2 Disconnecting the STM32F7 Discovery Kit

When you are done with all of your experiments and before leaving the class, it is always a good idea to eject the device prior to unplugging it, you can do this by selecting “Safely Remove Hardware and Eject Media” from the toolbar and then click on Eject DIS_F746NG, as the following figure illustrates.



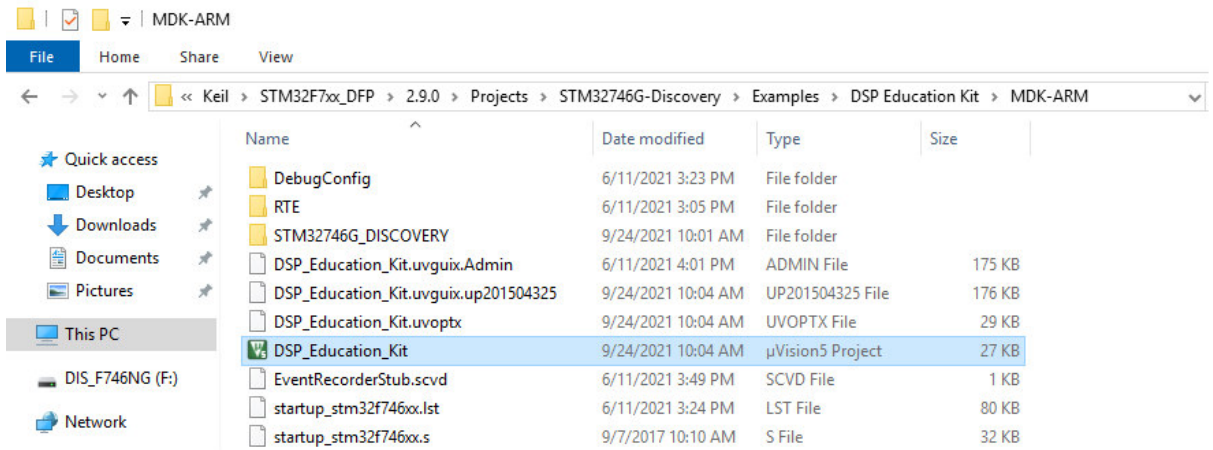
7 Real-Time Sine Wave Generation

Several ways exist to generate deterministic ([and periodic](#)) discrete-time signals such as a sine wave. We mention briefly three and focus on one of them for the purpose of this lab class. One way is by calling a mathematical function that generates samples of that discrete-time signal. In this case, an argument, or parameter, is provided when that function is called, for example, $\sin(\theta)$, or $\cos(\theta)$. Another way is by using a difference equation as we shall see later on in this course. This difference equation generates a new sample based on other samples that have been generated previously, or that are available by some other means. A third way is by means of a lookup table. In this case, all samples of interest of that discrete-time signal are precomputed and stored in memory, in a table, also called lookup table ([LUT](#)). The signal samples are simply read from that LUT, in a sequential fashion, which saves arithmetic computations.

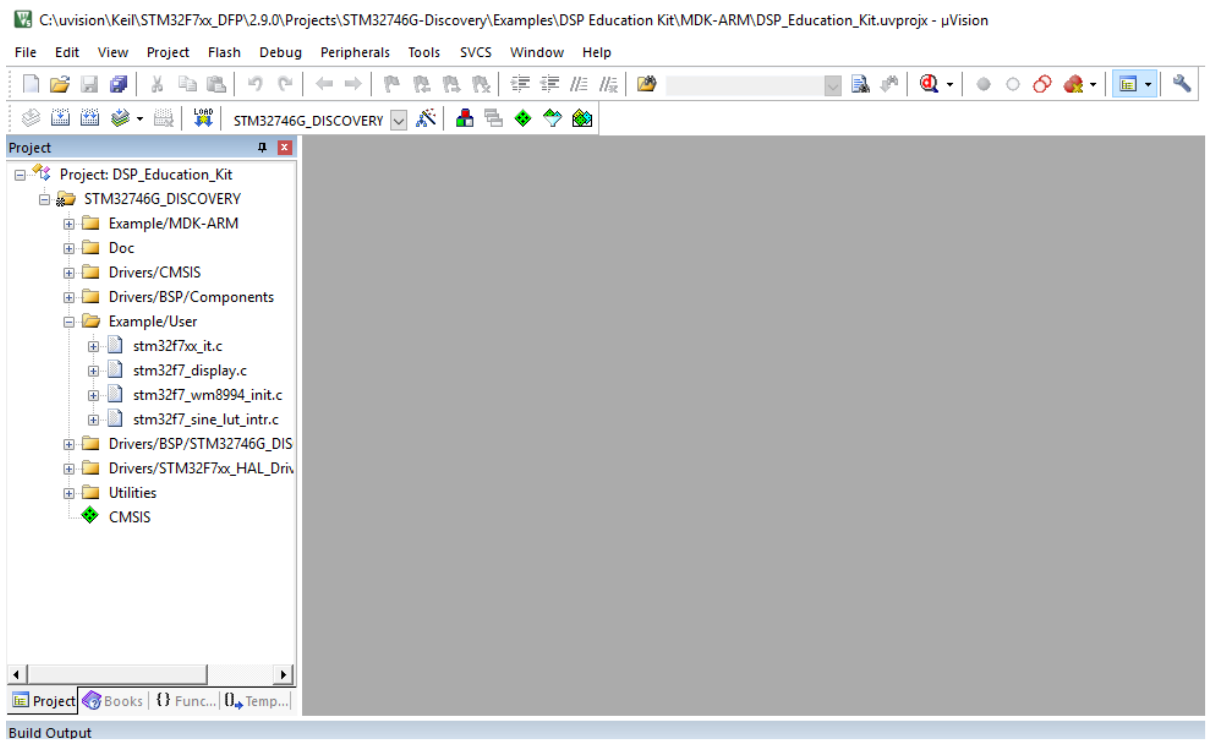
7.1 Loading a project and running the project

A project consists of a set of files, including configuration files and C code files, that are combined in the Keil MDK-Arm development environment (IDE), allowing them to be compiled, linked, and downloaded to run on the STM32F746G Discovery board. We illustrate how to follow through these steps by using the main C source file called `stm32f7_sine_lut_intr.c` which allows to generate a sinusoidal signal using interrupts ([see Section 4](#)) and a table lookup method.

- a) Using the Windows File Explorer, navigate to folder:
`C:\uvision\Keil\STM32F7xx_DFP\2.9.0\Projects\STM32746G-Discovery\Examples\DSP Education Kit\MDK-ARM`



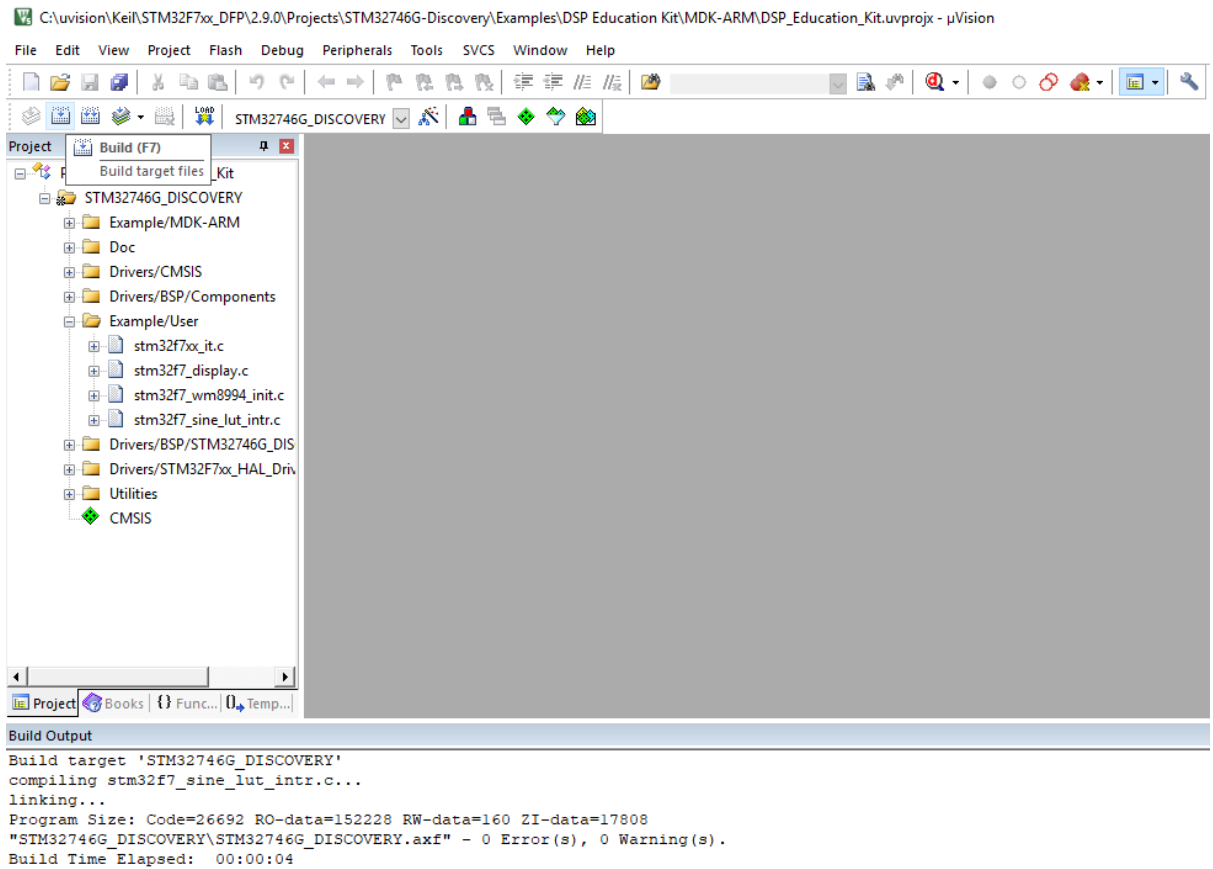
- b) In this folder you will find the **DSP_Education_Kit.uvprojx** project file which we will use as our baseline project in all of our lab classes. This project file is represented by the icon  **DSP_Education_Kit.uvprojx**, or just  **DSP_Education_Kit**. Double-click on this file/icon.
- c) Doing so opens up the Keil MDK-Arm development environment (Keil IDE, for short) which is called microVision (**μVision**). You will see then a structure of files belonging to the project as illustrated next.



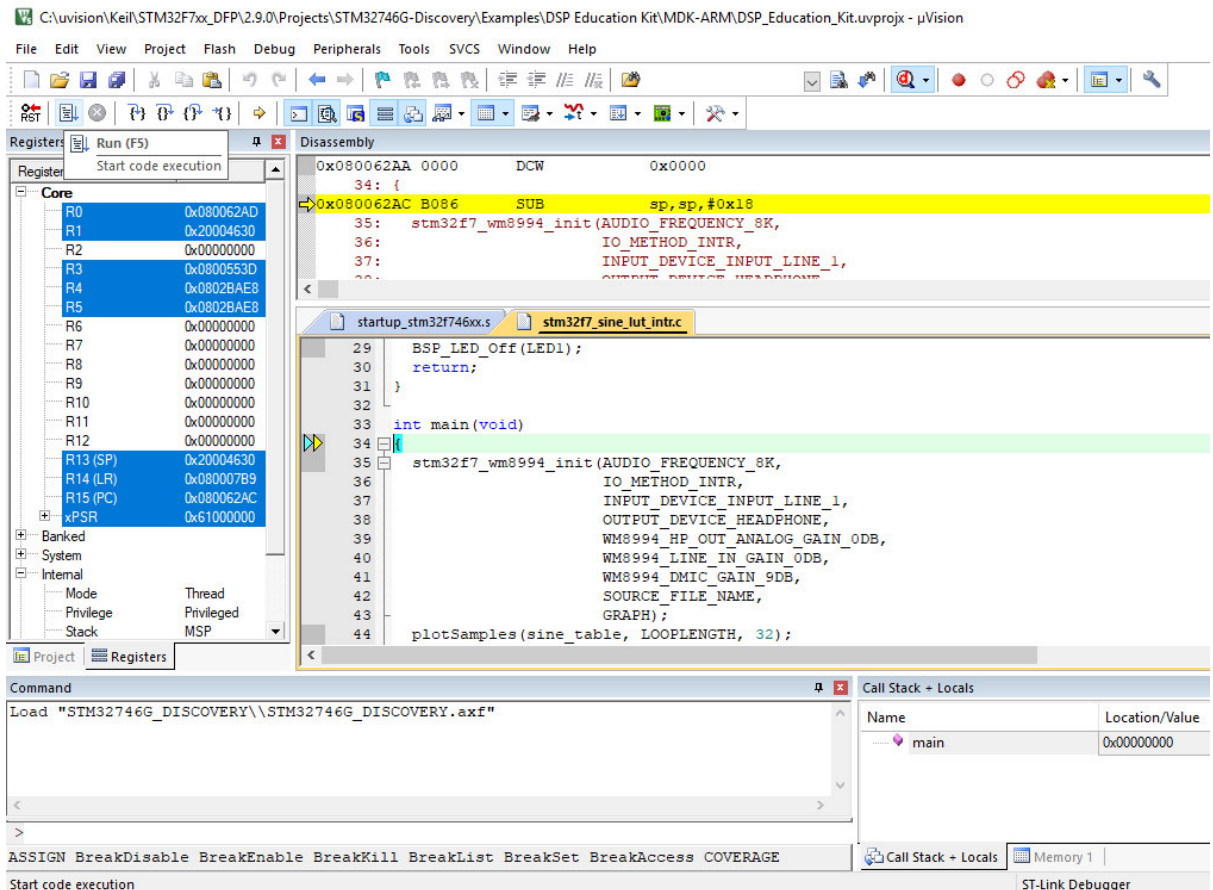
The project contains the example file **stm32f7_sine_lut_intr.c**, which, in turn, contains the `main()` function.

- d) Connect the STM32F746 Discovery board to the host PC using a USB A to mini-b cable that is available on the laboratory bench.
- e) Connect the output audio jack socket (CN10) on the board to the oscilloscope using the cable that is available on the laboratory bench (in this and future labs you may also plug the headphones into the output audio jack socket (CN10) on the board).

- f) We are now ready to start the compilation process. Build the project by selecting **Project > Build target** or by clicking on the *Build* toolbar button  (or press F7), as illustrated next.



- g) After successfully building the project with no errors, switch to the debugger mode (and download the executable code into the STM32F746 flash memory) by clicking on the **Start/Stop Debug Session** toolbar button .
- h) Once the **Debugger View** has appeared, click on the **Run** toolbar button  (or press F5), as shown in the following figure:



The program is now running on the STM32F746 Kit. Let's have a look at its structure and operation.

IMPORTANT NOTE: If you want to run different example programs, simply replace file `stm32f7_sine_lut_intr.c` in the *MDK-Arm* project with another source file. This procedure is explained in detail in Section 4 above. All of the source files are in the **DSP Education Kit\Src** folder. Detailed instructions for the different program examples will be provided throughout the semester.

7.2 Program operation

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, $\varphi = 0$. The `LOOPLength` values in array `sine_table` are samples of exactly one cycle of a sinusoid.

Just as in the previous examples (namely in Section 4), 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 7. This will be explained further in future FunSP classes.

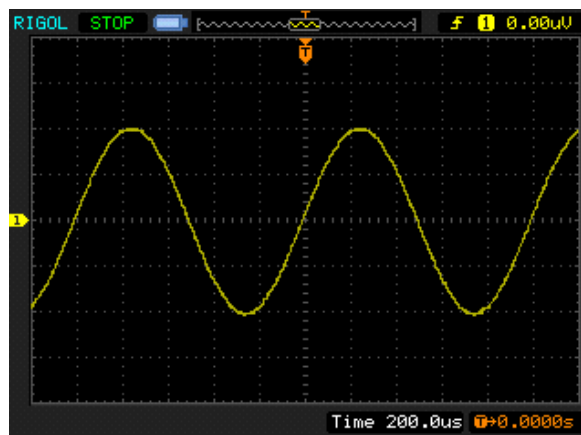


Figure 7: Analog output generated by program `stm32f7_sine_lut_intr.c`

When you run the program, you should see a start screen on the LCD as shown in Figure 8. Press the blue user pushbutton to continue (see Figure 0.1), and you should see on the LCD a graphical representation of the sequence of discrete sample values being written to the DAC (Figure 9). 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 (**NOTE: frequency-domain view is not necessary for now**).

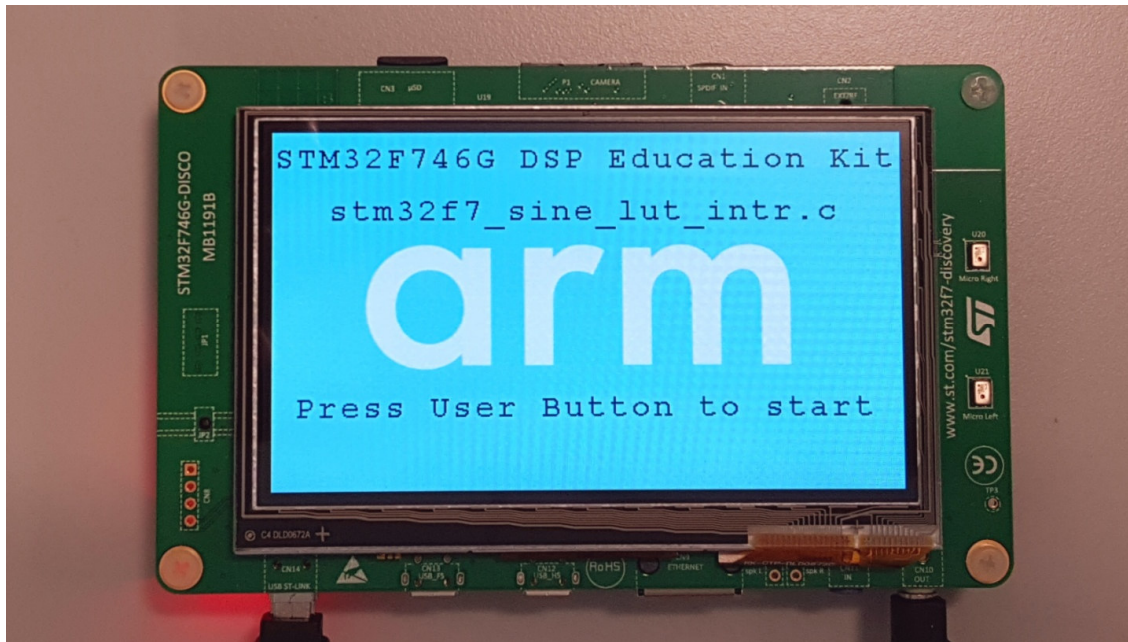


Figure 8: Start screen for program `stm32f7_sine_lut_intr.c`

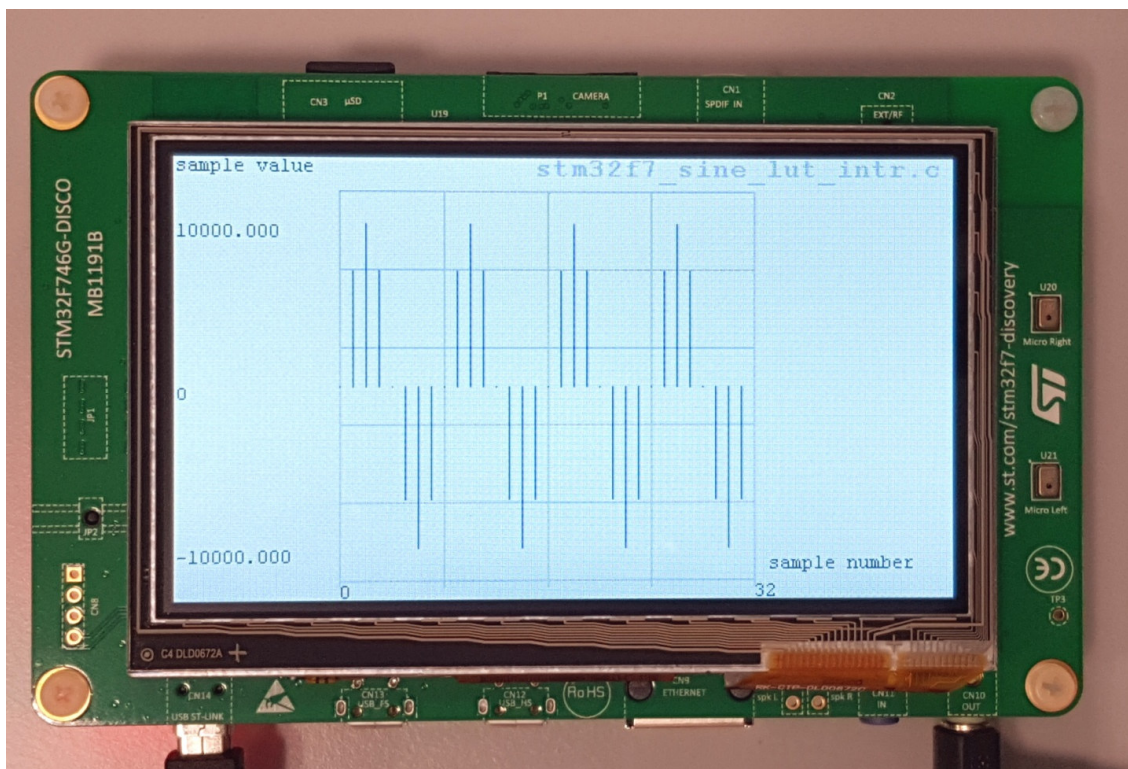


Figure 9: Graphical representation of first 32 sample values output by program `stm32f7_sine_lut_intr.c`

Now that you successfully observed, on both output channels (Left and Right), the same 1 kHz output sinusoidal signal, proceed to modify the code (i.e., edit the source file `stm32f7_sine_lut_intr.c`) such as to:

- a) observe on the Left and Right channels two out-of-phase 1 kHz sinusoids,
- b) observe on the Left and Right channels a 2 kHz output sinusoid (either in-phase or out-of-phase on both channels).
- c) How should you modify `stm32f7_sine_lut_intr.c` in order to generate a 500 Hz sinusoid ? [NOTE: you do not need to implement this, just answer the question]

After this experiment, time permitting (i.e., it is not mandatory) you can proceed to Section 4, and then to Section 5.

8 Conclusions

At the end of this exercise, you should have become familiar with several of the tools and techniques that you will use in subsequent lab exercises.

9 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>