DSP-kit quick-start

Jonathan Lock, Tomas McKelvey
{lock, tomas.mckelvey}@chalmers.se

September 14, 2017

1 Introduction

This document contains instructions that will help you set up a working development environment for the DSP-kit used in the course projects. These instructions will help you set up an appropriate *toolchain* and *on-line communications channel* with your DSP-kit. These steps are briefly outlined below, with detailed instructions in Sections 2 and 3 respectively.

Toolchain To run code on the DSP-kit a toolchain is required that compiles, links, and generates assembly code that matches the target hardware. The toolchain also manages uploading the assembly code to the DSP-kit, after which all your program is run by the DSP-kit hardware. We will use PlatformIO `http://platformio.org` which in turn downloads and installs all the needed programs and utilities for this.

On-line communication In order to control and read status information from the DSP-kit during operation a serial communication interface is used. This allows for bi-directional character-by-character communication to and from your PC and the DSP-kit. We will suggest an application suited to your PC's operating system in section 3.

You (should have) received the DSP-kit in a cardboard box containing;

- One black/gray cardboard box with a permanently attached black USB cable. This black/gray box contains all electronics and the microcontroller that will be programmed[1]. This should be inside a bubble-wrap bag. **When packed inside the brown cardboard box, keep the DSP hardware inside the bubble-wrap bag!**
- One gray USB-A to mini-USB cable. **Note; if you use a Windows operating system DO NOT not plug the DSP-kit into your computer before installing the necessary drivers!**

[1]The black/gray box contains an STM32F407G-DISC1, which is a development board with a 168MHz ARM cortex-M4 microcontroller and associated electronics.

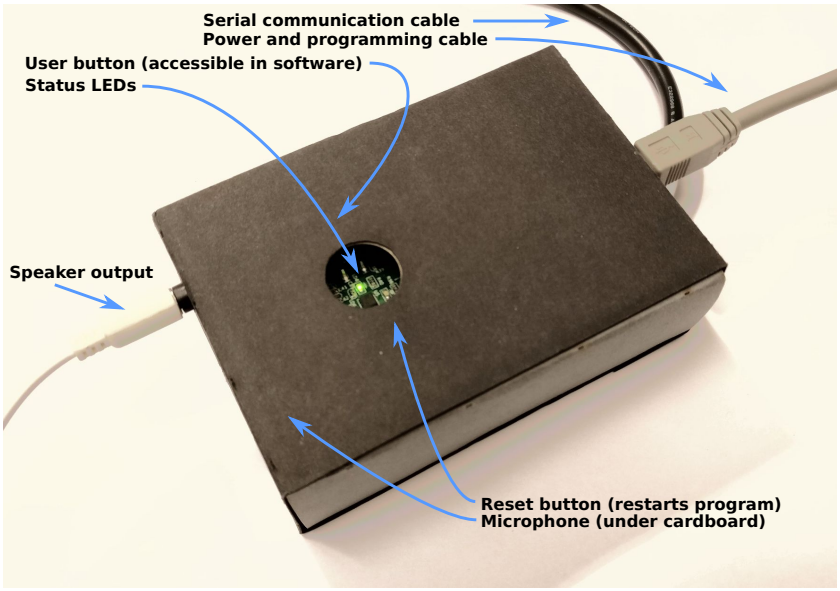

Figure 1: DSP box illustration with input/output, user buttons, and status LED's indicated.

- One pair of small white speakers.

To use the DSP-kit you will need a total of three USB ports. Should you only have access to two you can use a cell phone charger with a USB-A port to power the speakers, they do not need to be connected to your computer.

Figure 1 shows the location of all user-accessible input/output. Note that the user and reset buttons are located just outside the circular cutout and are used by lightly pressing on the cardboard case, which is flexible enough to move with the button.

The current source code skeleton can be downloaded from `https://github.com/lerneaenhydra/ssy130_dspkit`. (In the event of significant updates to the base code a message will be sent.) Download and extract the source code to a convenient location. In its initial state the source code is configured for a demo that shows some of the capabilities of the hardware; to test this demo you "only" need to compile the source code and upload the assembled code.

Note on problems;

Should you experience issues when using the DSP-kit, first check the current state of the online troubleshooting guide `https://docs.google.com/spreadsheets/d/1fsnSaYTMHcnsjN-a5xMeMxK-9kSq6DsIaacUJoeZyMU/edit?usp=sharing`. Should problems remain contact one of the individuals listed at the start of this document.

2 Toolchain Installation and verification

For the most part, installing the toolchain can be done simply by following PlatformIO's instructions; `http://docs.platformio.org/en/latest/core.html`. Any text editor can be used for the programming tasks. One popular editor is Sublime Text; `https://www.sublimetext.com/`, which adds some convenient syntax highlighting and other useful features[2].

Note for Windows users;

To access the development board a USB driver needs to be installed, see `http://www.st.com/en/development-tools/stsw-link009.html`. Download, unpack, and run `stlink_winusb_install.bat` in administrator mode **before plugging the DSP-kit into your computer**.

Note for Linux users;

It is possible that you need superuser privileges to upload the firmware to the DSP-kit, i.e. in the following section prepend all `pio` commands with `sudo`, e.g.

```
sudo pio run -t upload
```

Note for all platforms;

The current version of PlatformIO may have regressed on your computer with the specific hardware we use in the DSP-kit. If you receive an error when attempting to upload the firmware to the DSP-kit, remove the current version of PlatformIO and install version 2.9 with the following terminal/console commands;

```
pip uninstall platformio
pip install -U platformio<2.9
```

Note that the `pip` command will only be available on your system *after* installing Python and correctly updating your path variables, as described in PlatformIO's installation instructions.

The following steps will be shown in a Linux environment, in the event that different commands for Windows platforms the equivalent command will be given.

Open a terminal and change your current path to the root project directory using the `cd` and `ls` (Linux/Max) or `dir` (Windows) commands. This will contain, among others, a file named `platformio.ini`.

```
hydra@hydra-workstation:~/ssy130_dspkit-main$ ls
CHANGES.txt  lib  lib_ext  platformio.ini  README.md  src
```

[2] As the amount of programming you will perform is limited there is no need to install a full IDE (which simplifies some programming actions, e.g. renaming functions, searching for terms in multiple files, and so on). Should you wish to use an IDE this can be installed as well, see `http://docs.platformio.org/en/latest/ide/pioide.html` instead, which will set you up with a relatively light-weight IDE (Atom). This drawback of this is a more complex installation and setup procedure.

Compile the project using the command `pio run`. This will generate a lot of (for us) irrelevant text, but should be something like;

```
hydra@hydra-workstation:~/ssy130_dspkit-main$ pio run
[Fri Sep  8 14:47:04 2017] Processing proj_master ...
...
text       data     bss     dec     hex filename
738460     1960   19348  759768   b97d8 .pioenvs/proj_master/firmware.elf
================== [SUCCESS] Took 5.21 seconds ==========================
```

Plug the DSP-kit into your computer with the detachable cable. This powers the hardware and is also used for programming the device. Upload the compiled application with the command `pio run -t upload`;

```
hydra@hydra-workstation:~/ssy130_dspkit-main$ pio run -t upload
[Fri Sep  8 14:53:33 2017] Processing proj_master ...
...
src/common.c: Loading device parameters....
src/common.c: Device connected is: F4 device, id 0x10076413
src/common.c: SRAM size: 0x30000 bytes (192 KiB), Flash: 0x100000 bytes (1024
    KiB) in pages of 16384 bytes
src/common.c: Attempting to write 740420 (0xb4c44) bytes to stm32 address:
    134217728 (0x8000000)
st-flash 1.3.1
EraseFlash - Sector:0x0 Size:0x4000 EraseFlash - Sector:0x1 Size:0x4000
    EraseFlash - Sector:0x2 Size:0x4000 EraseFlash - Sector:0x3 Size:0x4000
    EraseFlash - Sector:0x4 Size:0x10000 EraseFlash - Sector:0x5 Size:0x20000
    EraseFlash - Sector:0x6 Size:0x20000 EraseFlash - Sector:0x7 Size:0x20000
    EraseFlash - Sector:0x8 Size:0x20000 EraseFlash - Sector:0x9 Size:0x20000
    2017-09-08T14:53:47 INFO src/common.c: Finished erasing 10 pages of 131072
    (0x20000) bytes
src/common.c: Starting Flash write for F2/F4/L4
src/flash_loader.c: Successfully loaded flash loader in sram
src/common.c: Starting verification of write complete
src/common.c: Flash written and verified! jolly good!
...
================== [SUCCESS] Took 30.82 seconds ==================
```

The output from this command can be useful for troubleshooting. In the example above it can be seen that the device is first detected, after which it is erased, programmed, and verified. In the event of an issue the specific point at which an error occurs may help with troubleshooting.

3 On-line communication and verification

The on-line communication channel functions independently of the programming interface and requires a separate application to function. In general, any serial console emulator will work. Should you not have a favorite the following are our suggestions. Regardless of the particular application used, the following settings apply;

Parameter	Value
Baud rate/speed	921600
Data bits	8
Stop bits	1
Parity	none
Flow control	none

3.1 Installing, Windows platforms

PuTTY (`https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html`). Select serial as the connection type. Under the **serial** tab in the left panel, set the speed to **921600** and set **flow control** to **off**. In the **window** tab, set **columns** to **120** and **rows** to **60**. The serial port can be determined by testing COM0, COM1, COM2, and so on until successful. Alternatively, open the device manager (in the control panel) and unplug/plug in the black, permanently connected, USB cable and check which device is removed/added.

3.2 Installing, Linux platforms

minicom (available in all major distributions. In Ubuntu/Debian, install with `sudo apt-get install minicom`). Unplug, then reconnect the black, permanently connected, USB cable. Check where the serial connection is available by calling;

```
hydra@hydra-workstation:~$ dmesg | tail
...
[27299.053268] ftdi_sio 3-2.2:1.0: FTDI USB Serial Device converter detected
[27299.053332] usb 3-2.2: Detected FT232RL
[27299.063354] usb 3-2.2: FTDI USB Serial Device converter now attached to
    ttyUSB0
```

here, the last line indicates that the device has been assigned the name `ttyUSB0`. Now, connect to the device by calling

```
hydra@hydra-workstation:~$ minicom -D /dev/<insert device name> -b 921600
```

which will connect to the device with the correct speed and settings. To exit the application, press <`CTRL`> + `a`, followed by `x`, followed by <`ENTER`>.

3.3 Installing, Mac platforms

screen is already included by default, so there is no need to install any new software. Follow the instructions listed at; `https://pbxbook.com/other/mac-tty.html`, replacing 9600 in their example with 921600.

3.4 Verification, all platforms

To verify the functionality, press the reset button (figure 1). If all is well something like the following text will be displayed in the window of your chosen serial communication application (i.e. PuTTy/minicom/screen).

```
  ____ ______   ___ _____  ___
 / ___/ ___\ \ / / |___ / / _ \
 \___ \___ \\ V /| | |_ \| | | |
  ___) |__) || | | |___) | |_| |
 |____/____/ |_| |_|____/_\___/
 |  _ \ ___ _ __   | |/ (_) |_
 | | | / __| '_ \  | ' /| | __|
 | |_| \__ \ |_) | | . \| | |_
 |____/|___/ .__/  |_|\_\_|\__|
           |_|

Compiled on Sep  8 2017 at 13:13:13

System setup;
        Sample rate           (48000) Hz
        Blocksize;            (256) samples
        Mic. lowpass  cutoff; (0) Hz
        Mic. highpass cutoff; (20) Hz

Use the '+' and '-' keys respectively to change the speaker volume during
    operation.
Use the '0' key to mute/unmute the output.
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Testing functions with reference input parameters
No unit testing needed for selected project configuration.
Testing complete!
--------------------------------------------------------------------------------
Usage guide;
Press the user button to switch between audio sources.
Press any of the following keys to change the current effect
        '1' - No effect applied
        '2' - Flanger
        '3' - Ring modulator
        '4' - Bit crusher
        '5' - Increase bit crusher depth
        '6' - Decrease bit crusher depth
```

Most of the above contents will always be shown on startup, giving some useful system information. If the speakers are plugged in, they should now be playing some music. Pressing the '+', '-', or '0' keys will always increase, decrease, or mute/unmute the speaker output regardless of the configured project, but will not give a response on the screen. Pressing any of the indicated keys ('1' through '6' here) should cause the output audio to change and display a short message on the screen.

If all the above steps have succeeded then your computer is now fully configured for use with the DSP-kit and you should be able to progress with the coding tasks.

4 config.h and changing the system operation mode

`config.h`, located in the `src` directory, contains many useful settings that you will need to change depending on the current assignment you are working on. Most of the configuration statements are fairly well documented, though some in particular are worth additional attention.

4.1 SYSMODE

The `SYSMODE_XXX` define statements control the entire system behavior, setting sample-rate, block-size, and which example/lab functions will be called. In order to compile for e.g. the OFDM-lab, replace the default `SYSMODE` section with

```
//#define SYSMODE_TEST1
//#define SYSMODE_TEST2
//#define SYSMODE_TEST3
//#define SYSMODE_TEST4
//#define SYSMODE_LMS
#define SYSMODE_OFDM
//#define SYSMODE_RADAR
//#define SYSMODE_FFT
```

By default, the application will be compiled for `SYSMODE_TEST1`. Any of the other test modes can be activated simply by uncommenting the relevant line. Details on what each test mode does and how to use them is listed in comments in `config.h`. All source code for the test modes can be found in `example.h/.c`.

4.2 Audio waveform

For `SYSMODE`s where an audio waveform is played back (e.g. `SYSMODE_TEST1`) the audio waveform can be changed by changing the

```
#define AUDIO_WAVEFORM_WAVEFORM1
```

statement to

```
#define AUDIO_WAVEFORM_XYZ
```

where `XYZ` is the name of any file `waveformxyz.h` in `/src/blocks/`, e.g.

```
#define AUDIO_WAVEFORM_WAVEFORM2
```

4.3 Serial baud-rate

If your PC (for some reason) does not seem to support the default baud-rate of 921600, but would instead e.g. only support a rate of 1000000 this can be changed by altering

```
#define DEBUG_USART_BAUDRATE            (921600ULL)
```

to

```
#define DEBUG_USART_BAUDRATE            (1000000ULL)
```

5 The DSP-kit in day-to-day use

At this stage you should have a working toolchain that allows you to program the DSP-kit as well as a working communication channel. A typical workflow for the DSP-kit when implementing/testing the different project tasks is;

1. Connect all three USB cables (using a seperate USB power supply for the loudspeakers if needed) and connect the 3.5 mm speaker plug to the DSP-kit.

2. Open a on-line serial communication window open and read/write to the DSP-kit as needed.

3. Open a terminal window, and set its path to the root project directory. This window will be used for compiling and uploading the firmware. There are typically only three commands you need to use;

   a) Compile and upload the current code

   ```
   pio run -t upload
   ```

   b) Compile the current code but do not upload the result

   ```
   pio run
   ```

   c) Remove all intermediate compiler cache. Useful as a first test should the program not behave as expected after changing fundamental system properties, e.g. changing the SYSMODE parameter. This will only clean your workspace, to test the system behavior you'll need to compile and optionally upload the firmware as a separate command, e.g.

   ```
   pio run -t clean
   pio run -t upload
   ```

4. Keep all relevant source code documents open in the text editor you installed in section 2.

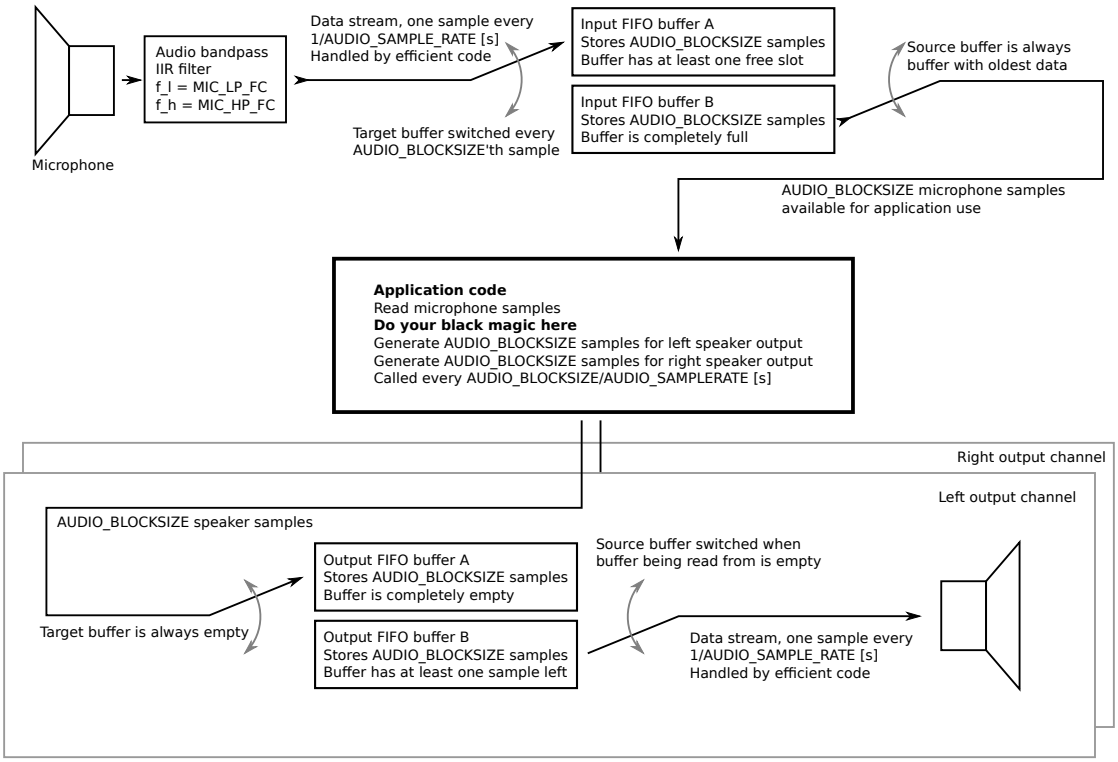

Figure 2: Data stream overview.

During development you'll make changes to various files — when you want to test the behavior of the code you've written; compile and upload the program, then switch to the serial communication window to view/control the system behavior.

In general, there is no need to connect/disconnect any hardware during development, i.e. you may keep the speakers and serial communication USB cable plugged into your computer during all stages of development.

Avoid disconnecting or resetting the DSP-kit using the reset button while uploading new software to the device, as this may require manually resetting the DSP-kit. Though this will not damage the DSP-kit hardware it is somewhat time-consuming to resolve.

6 DSP-kit software back-end

In order to understand the system behavior it is important to have some basic knowledge of the DSP-kit software back-end.

6.1 Microphone and speaker data streams

Figure 2 illustrates the structure of the back-end that handles supplying the application code (i.e. your code) with microphone samples shuffling the requested speaker output to

the physical speakers. Note that there is a single microphone input, but two (identical in structure) speaker outputs.

The source for for `SYSMODE_TEST2` illustrates a minimal example of the application code needed to read from the microphone and write to the speakers, as file `/src/example.c` contains only;

```
#if defined(SYSMODE_TEST2)
void example_test2_init(void){
        //No setup needed
}

void example_test2(void){
        //Allocate space for the microphone and waveform samples.
        float micdata[AUDIO_BLOCKSIZE];
        float wavedata[AUDIO_BLOCKSIZE];

        //Write AUDIO_BLOCKSIZE samples to the microphone and waveform buffer
        blocks_sources_microphone(micdata);
        blocks_sources_waveform(wavedata);

        //Send the waveform data to the left output and the microphone data to
            the right output without any processing
        blocks_sinks_leftout(wavedata);
        blocks_sinks_rightout(micdata);
}
#endif
```

where `example_test2_init` is called once on startup by `main.c` (which does nothing), after which `example_test2` is called by `main.c` each time the next microphone and speaker buffers are full and empty respectively. With the default settings of `AUDIO_SAMPLERATE = 48000` and `AUDIO_BLOCKSIZE = 256` this implies `example_test2` is called once every $\frac{256}{48000} \approx 5.3\,\text{ms}$.

The back-end data stream diagram also gives the following insights,

- There is no requirement to read the microphone data — it can be ignored — which results in the microphone buffer being overwritten.
- The application code is called once an entire block (i.e. a contiguous group of samples) have been buffered. This structure helps reduce function call overhead, leaving more execution time available for useful signal processing in the code you will write.
- The total round-trip latency from the microphone to the loudspeaker will be `2*AUDIO_BLOCKSIZE` (as `AUDIO_BLOCKSIZE` samples are first buffered in the microphone buffer, and then once again in the speaker buffers). This implies that any microphone-to-speaker filtering you perform will be followed by additional latency that can be viewed as a linear system with response;

  $$y[n] = x[n - 2 \cdot \text{AUDIO\_BLOCKSIZE}]$$

  i.e. a simple propagation delay. In practice, the actual delay is slightly longer as the microphone IIR filter also adds some latency.

- It is crucial that the application code finishes execution within `AUDIO_BLOCKSIZE / AUDIO_SAMPLERATE` seconds. If it does not the loudspeaker buffer will run out of queued samples and (by design) the system stop operating. This condition is tested for in the back-end and an error message will be printed should this occur. As a note, if the execution time is too large and does not increase with `AUDIO_BLOCKSIZE` (e.g. you wish a print a long status message, that is not made longer when increasing `AUDIO_BLOCKSIZE`), the permissible time can be increased both by increasing `AUDIO_BLOCKSIZE` and by decreasing `AUDIO_SAMPLERATE`.

6.2 Self-testing

The DSP-kit will perform a self-test of all the functions you need to write code for when the relevant `SYSMODE` configuration is selected.

The system self-test will apply known inputs to your function(s) and compare their output to known-good (i.e. correct) output. Should your functions not return values identical or close enough to the reference output execution will stop and an error message will be displayed. This functionality is useful as it adds some degree confidence that your code works correctly and that the subsequent system behavior you see is correct. However, it is possible for your code to not pass the self-testing despite being correct (false-negative), and conversely, it is possible that your code passes the self-test while in fact generating incorrect results (false-positive). In summary, **though the self-test functionality is useful, do not blindly trust its verdict**.

Should you find a case where you are certain the self-test-functionality returns either a false-positive or false-negative (and your code isn't a horrible abomination), please let one of the individuals listed at the top of this document know so a patch may be issued.