ETIN80 — Algorithms in Signal Processors
Information Sheet

Tekn.Dr. Mikael Swartling

Lund Institute of Technology

Department of Electrical and Information Technology

1 Learning the Environment

Read the document *Visual DSP++ 5.0 Getting Started Guide* provided on the course web page, primarily section 2 covering the basic tutorials. The guide shows to how to create and setup a project, and how to configure a simulator or emulator session to run your programs. Follow the exercises to get familiar with Visual DSP++. Copy the tutorial projects from the specified location to your own folder if you intend to change it or build it. Important changes:

- Replace *Blackfin ADSP-BF533* with *SHARC ADSP-21262* throughout the guide.
- Use the *ADSP-21262 via HPUSB-ICE* emulator platform.

Simulator Session

A *simulator* session is used without a DSP connected. This session type can be used if you want to run a software emulation of the DSP in order to try some of the tutorial examples without having to attach a physical DSP to the computer. The simulator platform *does not* provide audio or keyboard interrupts and it is therefore not possible to run your final audio processing algorithms in the simulator.

Emulator Session

An *emulator* session requires the physical DSP to be connected to your computer. The DSP is attached to the in-circuit emulator via USB and allows you to run your programs on the DSP.

Common Errors

Two common errors and corresponding solutions:

```
cc1458:  Error:  could not obtain license
```

The project from the Getting Started Guide is made for a Blackfin DSP that we don't have a license for. Change the target DSP to the ADSP-21262.

```
Attempt to read from non-existent memory with PC at:  xxx
```

The active session is a Simulator session that is unable to run the framework provided with the course materials. In order to run the framework on the actual DSP, select an emulator session.

2 Software Framework

Download the framework for the DSP provided on the course web page. The framework provides code to configure the DSP and supplies a very primitive API towards the audio codecs, the keypad and the LEDs. Add the supplied source and header files to your project. The file `main.c` is not a part of the framework itself, but can be used as a base for your project.

Functions to control or query the DSP:

`dsp_init`	Call once at the beginning to initialize the framework.
`dsp_start`	Call to start the serial ports connected to the codecs.
`dsp_stop`	Call to stop the serial ports connected to the codecs.
`dsp_get_keys`	Returns a bitmask of keys currently pressed. Bit `DSP_SWn` (for n in 1 to 4) is set in the return value if key n is pressed, and reset if key n is not pressed.
`dsp_set_leds`	Sets the LEDs according to a bitmask. Bit `DSP_Dn` (for n in 1 to 2) is set in the parameter if LED n is to be turned on, and reset if LED n is to be turned off.

Four stereo-channels are available and are accessed via the following functions. The functions return a pointer to the current audio block to be processed. The number of samples to process is set by the `DSP_BLOCK_SIZE` parameter (see below about configuring the framework).

`dsp_get_audio_u30`	Input from `Line in 1` with microphone driver disabled, no output.
`dsp_get_audio_u31`	Input from `Line in 2` with microphone driver enabled, no output.
`dsp_get_audio_u32`	Input from `Mic 1` and `Mic 2`, output to `Headset`.
`dsp_get_audio_u33`	Input from `Mic 3` and `Mic 4`, no output.

Interrupts raised by the framework:

`SIG_SP1`	Interrupt to process audio data. Call `dsp_get_audio` to get the current audio block.
`SIG_USR0`	Interrupt when a button is pressed. Call `dsp_get_keys` to get the state of all keys.
`SIG_TMZ`	Interrupt when the timer counter expires.

3 Project Settings and Flash Loader

The DSP does not have internal flash memory to store your program and must be flash-programmed in order to run outside the emulator. Observe that you do *not* have to flash-program when the DSP is connected to the emulator and you are running your program from within Visual DSP++.

Running in Emulator Mode

When running with the DSP attached to the computer and Visual DSP++, your project shall be set to create an executable file. An executable file with the extension `.dxe` is created that you load into the DSP.

- In the *Project* menu, select *Project Options*.
- In the *Project* settings, set *Type* to *Executable file*.

Running in Stand-alone Mode

When running the DSP stand-alone without being attached to the computer or Visual DSP++, the DSP has to be flash-programmed. A loader file with the file extension `.ldr` is created that you load using the flash programmer tool.

- In the *Project* menu, select *Project Options*.
- In the *Project* settings, set *Type* to *Loader file*.
- In the *Project/Load/General* settings, set *Boot Type* to *SPI flash*.

The flash programmer tool can now program the DSP.

- Ensure that the DSP is connected to the computer and Visual DSP++, and that an emulator session is active.
- In the *Tools* menu, select *Flash Programmer*.
- Select the *Flash* tab.
- In the *Driver* tab:
  - Select the supplied driver file in *Driver file*. The file is `21262EzFlashDriver_Serial.dxe` available from the course web page.
  - Press *Load Driver*.
- In the *Programming* tab:
  - Select your compiled loader file in *Data file*. The loader file is the `.ldr` file in your project's `Release` or `Debug` folder.
  - Press *Program* to write your application to flash memory.
- Disconnect the session, disconnect the DSP from the computer and power-cycle the DSP. The DSP will now run your program in stand-alone mode as soon as power is connected to the DSP.

4 Configuring the Framework

The header file `framework.h` has four values you must configure to your needs.

`DSP_SAMPLE_RATE`	The audio sample rate in Hz.
`DSP_BLOCK_SIZE`	The block size in samples.
`DSP_INPUT_GAIN`	The microphone input gain in dB.
`DSP_OUTPUT_ATTENUATION`	The speaker output attenuation in dB.

Configure the sample rate and block size according to your needs. Input gain and output attenuation can initially be set to 10 dB and 0 dB, respectively, until you need different gains to accommodate your application.

5 Block Buffers and the Framework

The function `dsp_get_audio` returns a pointer to an array of `DSP_BLOCK_SIZE` fixed point-valued stereo samples. Blocks are processed in place, which means that you both read the input from and write the output to the same physical block buffer. This function can be called once within the process interrupt function triggered by the audio codecs to access the current frame of the audio stream.

The pointer returned by `dsp_get_audio` has the type `sample_t *` where `sample_t` is defined as follows:

```
typedef struct {
    fract left;
    fract right;
} sample_t;
```

The `sample_t` structure thus contains a left-right sample pair of a stereo signal and all the samples in the current audio block can be accessed by indexing the pointer. See the example attached with the framework for how you can access and process the audio stream.