ELEN0019-1 Audio Signal Processing
Principles and Experiments

Julien Osmalskyj

University of Liège

2014

Contents

1 Course organization

2 Introduction

3 DSP Architecture

4 Input / Output

5 Software : Code Composer Studio

6 Project example

Contents

1 Course organization

2 Introduction

3 DSP Architecture

4 Input / Output

5 Software : Code Composer Studio

6 Project example

General information

- Julien Osmalskyj
- Office R28
- `www.montefiore.ulg.ac.be/~josmalskyj/dsp.php`
- Email : `josmalsky@ulg.ac.be`

Course organization

Practical course organized in 12 lab sessions of 4 hours.
Groups of 2 - 3 students.

- 6 sessions for the course
- 6 sessions for the final project

No theoritical session except this first one.

Evaluation

Evaluation is based only on the students final project. No evaluation of the lab sessions.

Everybody has to attend each lab session in order to succeed the course.

Students projects examples

- Guitar Tuner
- Equalizer
- Tap-delay
- Chorus - Flanger
- Compressor
- Loudspeaker frequency correction
- Signal analyzer
- etc.

Contents

1 Course organization

2 Introduction

3 DSP Architecture

4 Input / Output

5 Software : Code Composer Studio

6 Project example

Introduction

Digital signal processors are used in many areas such as

- sound
- video
- computer vision
- music analysis
- etc.

They are found in

- Cellular phones
- Disk drives
- MP3 players
- etc.

Introduction

Principle

A DSP digitizes an analog signal, manipulates it using mathematical and logical operations and converts the result back to an analog wave form.

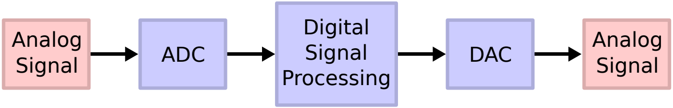

FIGURE : General DSP system

Introduction

Real-time constraint

DSP are processors specialized for real-time processing. Audio samples arrive at a constant rate (*e.g.* every $1/48000$ seconds) and must be processed before the next samples arrive.

DSP have a specific architecture and specialized instructions optimized to minimize the number of CPU clock cycles.

- MAC instruction (Multiply ACcumulate) for fast convolutions
- Fast memory access
- Harvard or modified Harvard architecture
- Programmable in Assembly or C language

Contents

1 Course organization

2 Introduction

3 DSP Architecture

4 Input / Output

5 Software : Code Composer Studio

6 Project example

Texas Instrument C6748 Processor

In this course, we use a Texas Instrument OMAP-L138 processor which is a single chip containing a C6748 DSP core and and ARM9 processor.

- Based on Texas Instrument very long instruction word (VLIW)
- Clock rate of 375 MHz
- Fetches eight 32-bit instruction every clock cycle
- Both floating-point and fixed-point architecture

Interrupts

4 types of interrupt on the CPU :

- Reset
- Maskable
- Non-maskable (NMI)
- Exception

Reset and Non-maskable interrupts have the highest priority.
12 maskable interrupts (INT4 - INT15) can be associated with external devices, on-chip peripherals or software control.

Receive samples

In this course, events corresponding to a new input sample (events #61 or #8) are associated with INT4 interrupt.

Interrupt selector (IS)

128 systems events are available in the DSP. The IS allows to select one event and route it to the appropriate CPU interrupt.

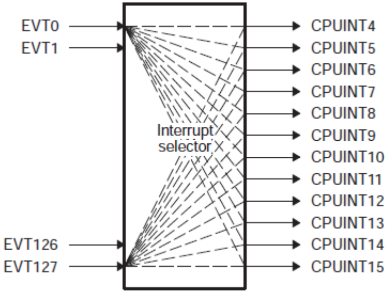

FIGURE : Interrupt selector

Interrupt Service Table (IST)

The code executed when an interrupt occurs is determined by the content of the IST. The IST contains one Interrupt Service Fetch Packet (ISFP) associated with each maskable interrupt. The ISFP associated with INT4 contains a branch instruction to a function `interrupt4()` which must be defined in the C program.

xxxx 000h	RESET ISFP
xxxx 020h	NMI ISFP
xxxx 040h	Reserved
xxxx 060h	Reserved
xxxx 080h	INT4 ISFP
xxxx 0A0h	INT5 ISFP
xxxx 0C0h	INT6 ISFP
xxxx 0E0h	INT7 ISFP
xxxx 100h	INT8 ISFP
xxxx 120h	INT9 ISFP
xxxx 140h	INT10 ISFP
xxxx 160h	INT11 ISFP
xxxx 180h	INT12 ISFP
xxxx 1A0h	INT13 ISFP
xxxx 1C0h	INT14 ISFP
xxxx 1E0h	INT15 ISFP

Program memory

FIGURE : Interrupt Service Table

OMAP-L138 eXperimenter

The board used in this course is the Zoom OMAP-L138 eXperimenter. It includes the OMAP-L138 processor and many on-board peripherals.

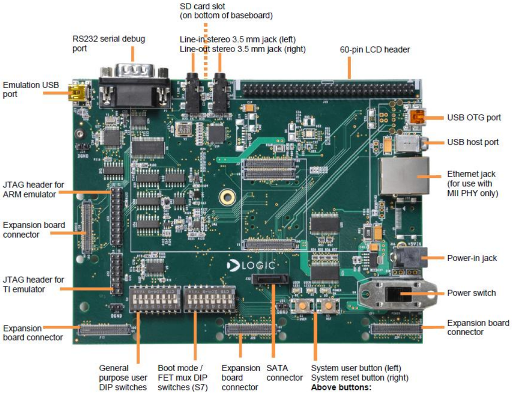

FIGURE : OMAP-L138 eXperimenter

OMAP-L138 eXperimenter

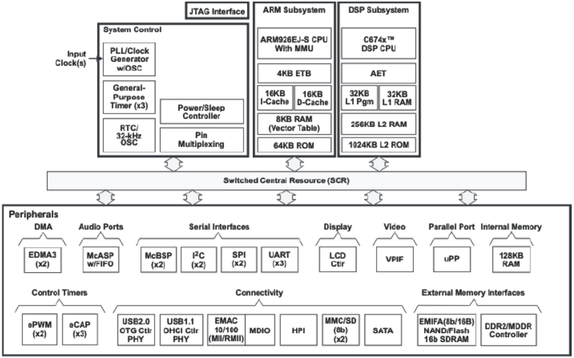

FIGURE : OMAP-L138 eXperimenter

Contents

1 Course organization

2 Introduction

3 DSP Architecture

4 Input / Output

5 Software : Code Composer Studio

6 Project example

Input / Output

Library

A basic library for accessing DSP inputs and outputs is used for this course.

3 ways of reading input samples and writing output samples.

- Polling
- Interrupts
- Direct Memory Access (DMA)

Library functions

Functions are available for reading and writing samples in each mode.

- `int32_t input_sample()`
- `int16_t input_left_sample()`
- `int16_t input_right_sample()`
- `void output_sample(int32_t out_data)`
- `void output_left_sample(int16_t out_data)`
- `void output_right_sample(int16_t out_data)`

Data format

AIC3106 codec converts samples of both channels to 16-bit signed integers. Channels are combined to form a 32-bit sample.

```
typedef union {
  uint32_t uint;
  short channel[2];
} AIC31_data_type;
```

FIGURE : Union structure to store samples

```
AIC31_data_type codec_data;
codec_data.uint = input_sample();
short left_sample = codec_data.channel[LEFT];
short right_sample = codec_data.channel[RIGHT];
```

FIGURE : Read both left and right channels

Polling scheme

Principle

Processor queries the codec when the processing is finished.
The input and output functions wait for the codec to be ready.

Initialization function :

```
L138_initialise_poll(FS_48000_HZ, ADC_GAIN_0DB,
DAC_ATTEN_0DB)
```

- `FS_48000_HZ` : Sampling frequency set to 48000 Hz
- `ADC_GAIN_0DB` : Gain at the input set to 0 dB
- `DAC_ATTEN_0DB` : Attenuation at the output set to 0 dB

Polling code example

```
#include "L138_aic3106_init.h"

void main(void) {
  uint32_t sample;

  L138_initialise_poll(FS_48000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB);
  while (1) {
    sample = input_sample();
    output_sample(sample);
  }
}
```

FIGURE : Input / Output using polling

Interrupt-based scheme

Principle

INT4 is triggered when a sample arrives at the input of the codec and the code of function `interrupt4()` is executed.

Initialization function :

```
L138_initialise_intr(FS_48000_HZ, ADC_GAIN_0DB,
DAC_ATTEN_0DB)
```

Interrupt-based code example

```c
#include "L138_aic3106_init.h"

interrupt void interrupt4(void) { // interrupt routine
  uint32_t sample;
  sample = input_sample();
  output_sample(sample);
  return;
}

void main(void) {
  L138_initialise_intr(FS_48000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB);
  while (1) ;
}
```

FIGURE : Input / Output using interrupts

Direct Memory Access (DMA)

Principle

EDMA3 controller transfers **blocks of *N* samples** between the codec and the memory without intervention of the CPU. An interruption is triggered when all *N* samples have been transferred.

Initialization function :

```
L138_initialise_edma(FS_48000_HZ, ADC_GAIN_0DB,
DAC_ATTEN_0DB)
```

DMA code example I

```
interrupt void interrupt4(void) { // interrupt routine
  switch(EDMA_3CC_IPR) {
    case 1:                      // TCC = 0
      procBuffer = PING;         // process ping
      EDMA_3CC_ICR = 0x0001;     // clear EDMA3 IPR bit TCC
      break;

    case 2:                      // TCC = 1
      procBuffer = PONG;         // process pong
      EDMA_3CC_ICR = 0x0002;     // clear EDMA3 IPR bit TCC
      break;

    default:                     // may have missed an interrupt
      EDMA_3CC_ICR = 0x0003;     // clear EDMA3 IPR bits 0 and 1
      break;
  }
  EVTCLR0 = 0x00000100;
  buffer_full = 1;               // flag EDMA3 transfer
  return;
}
```

FIGURE : DMA interrupt routine

DMA code example II

```
int main(void) {
  L138_initialise_edma(FS_48000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB);
  zero_buffers();

  while(1) {
    while (!is_buffer_full());
    process_buffer();
  }
}
```

FIGURE : DMA scheme main function

The `process_buffer()` function implements the processing of the block of samples.

Contents

1 Course organization

2 Introduction

3 DSP Architecture

4 Input / Output

5 Software : Code Composer Studio

6 Project example

Code Composer Studio

Code Composer Studio (CCS) is the integrated development environment provided by Texas Instrument.

It is based on the Eclipse framework and offers a convenient editor and a debugger.

Important note

By default, CCS compiles the code in *debug* mode. To achieve better performances, configure CCS to compile the code in *release* mode.

Code Composer Studio : Editor

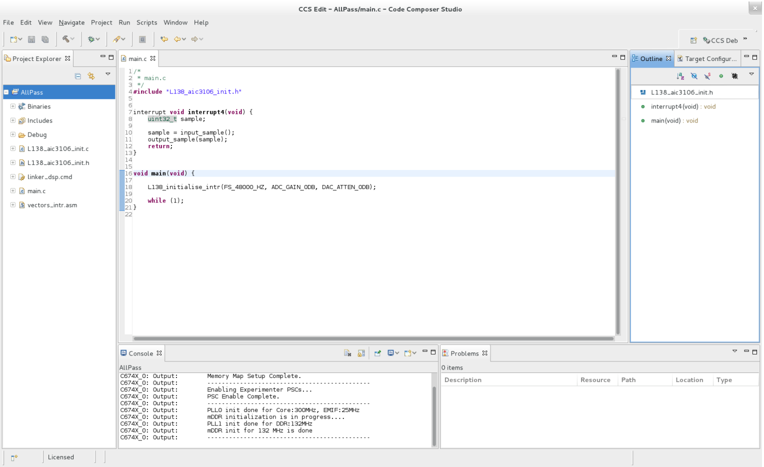

FIGURE : CCS Editor

Code Composer Studio : Debugger

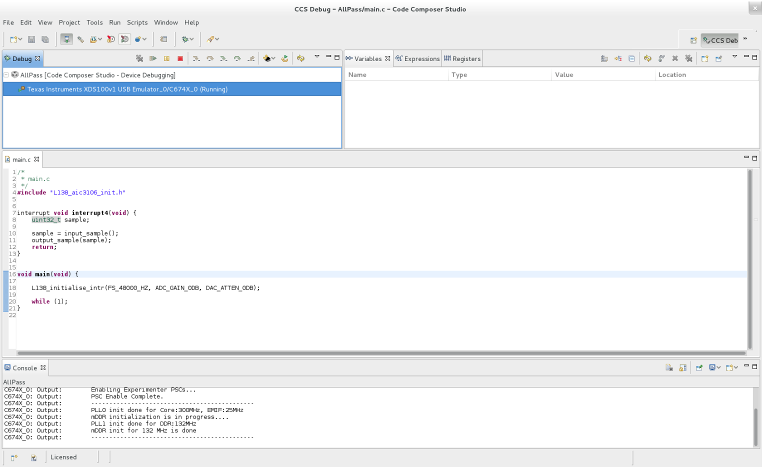

FIGURE : CCS Debugger

Contents

1 Course organization

2 Introduction

3 DSP Architecture

4 Input / Output

5 Software : Code Composer Studio

6 Project example

Project Example

Let us create a simple project which switches the left and right channels of an input signal.

The procedure to program the DSP is the following.

- Create new CCS project
- Configure include paths and linker file search path
- Add necessary files to the projects
- Code the program
- Compile in *debug* or *release* mode
- Run the program by starting the debugger

Detailled procedure for creating a new project in CCS is explained in Chapter 2 of the course notes.

Interrupt service routine

```c
#include "L138_aic3106_init.h"

AIC31_data_type codec_data;
int channel = LEFT;

interrupt void interrupt4(void) {
  codec_data.uint = input_sample();
  short left_sample = codec_data.channel[LEFT];
  short right_sample = codec_data.channel[RIGHT];

  if (channel == LEFT) {
    codec_data.channel[LEFT] = left_sample;
    codec_data.channel[RIGHT] = right_sample;
    channel = RIGHT;
  }
  else {
    codec_data.channel[LEFT] = right_sample;
    codec_data.channel[RIGHT] = left_sample;
    channel = LEFT;
  }
  output_sample(codec_data.uint);
  return;
}
```

Interrupt service routine

The main function only initializes the DSP using interrupt-based scheme and starts the main loop. At each sampling instant

$$T = 1/48000s$$

the program will be interrupted.

```
void main(void) {

  L138_initialise_intr(FS_48000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB);

  while (1);
}
```

FIGURE : Main function