Advanced Operating Systems
Embedded Systems 1

Implementation of a reaction game on a STM32F4 board

Source code:
https://github.com/ArneVlietinck/Reaction_game

Authors:
Simon Mastrodicasa
Arne Vlietinck

Professor:
Fornaciari William

January 4, 2018

Academic Year 2017-2018

Contents

1 Introduction . . . 2
2 Rules of the reaction game . . . 2
3 Game Implementation . . . 2
  3.1 Global Variables . . . 2
  3.2 Main initialisation . . . 3
  3.3 Game loop . . . 4
  3.4 Game play . . . 4
4 Used programs . . . 5
5 Possible improvements . . . 5
  5.1 Input . . . 5
  5.2 Game play . . . 5

Appendices **8**

A Main **8**

B Game play **9**
1 blinkingClockwise() . . . 9
2 blinkLed() . . . 9
3 gamePlay() . . . 10

C Doxygen manual **12**

Bibliography **74**

1 Introduction

The reaction game is developed by a group of Erasmus students, Simon Mastrodicasa (896230) and Arne Vlietinck (895101). It is a reflex based game written in C++ for the courses Advanced Operating Systems (AOS) and Embedded systems 1. Arne takes AOS and Embedded Systems 1, Simon only the first one.
The game implementation is based on Miosix kernel [1] and made for the discovery board STM32F407G.
The source code can be found on GitHub [3] under `https://github.com/ArneVlietinck/Reaction_game`.

2 Rules of the reaction game

The game consists of four LED's (blue, green, orange, red) that are switched on and off. Whenever one blinks two times in a row (e.g. blue, green, green, orange etc.), the player has to push the button before the game goes to the next LED, which is orange in the example above. If the player is too late or didn't push the button at all, the game is lost. Also when the button is pushed on a wrong moment (when the LED blinks once), the game is lost.

If the player pushed the button at the right time, the time between a blinking LED is reduced, consequently the difficulty increases.
A little variant can happen randomly to challenge the player a bit more: instead of doing a constant clockwise sequence, it may change in a counterclockwise sequence.

Two different sounds are implemented in the game over ritual: (I) a buzzer sound and (II) a high score sound. The high score sound is played when the player reaches a new high score. Otherwise the buzzer sound is played.

3 Game Implementation

In order to understand the game, the `main()` in Appendix A will be described.
The sound implementation and guidelines to use the Miosix kernel are based on the examples inside the Miosix kernel.
The implementation of the interrupt routine is based on the code given in the course of Advanced Operating Systems with as subject *os_context_switch* and can be found in the online course environment [2].

3.1 Global Variables

bool game

Boolean which represents the state of the game. If game is equal to 1, the current game is finished. Else if game is equal to 0, the current game is still running.

bool interaction

Boolean which represents the need of a players' interaction. When interaction is equal to true, the player must do something (e.g. press the user button) to avoid a game over.

bool action

Boolean which represents the action of a player. When action is true, the player did an action. When action is false, the player didn't do an action.

int difficulty

Integer which represents the difficulty level of the game. Higher integer means higher degree of difficulty. It affects the time between the blinking LED's.

int level

Integer which represents the level of the game.

int highscore

Integer which represents the current high score of the game.

3.2 Main initialisation

Several outputs need to be initialised in the function `mainInitialisation()`:

(i) User LD3: orange LED is a user LED connected to the I/O PD13,

(ii) User LD4: green LED is a user LED connected to the I/O PD12,

(iii) User LD5: red LED is a user LED connected to the I/O PD14,

(iv) User LD6: blue LED is a user LED connected to the I/O PD15,

(v) B1 USER: User and Wake-Up buttons are connected to the I/O PA0,

(vi) audio DAC (CS43L22) to output sounds through the audio mini-jack connector.

See the user manual [6] for extra info about the hardware initialisation.
Besides the outputs also `highscore` is set to 0.

3.3 Game loop

If the game is over, the software initialisation must be redone in `reInitialisation()`. The global variables (except for `highscore`) are reinitialised, ended by a little visual sign (all LED's blink at the same time). This sign informs the player that a new game is about to start.
Important to note is that `action` is locked by a mutex to protect race conditions. This can occur due the fact that `action` is a shared variable between two threads, the main thread and the interrupt thread.

In any case, `blinkingGame()` will run which sets the first LED and sleepTime (time between a LED blink). Depending on `clockOrCounterClockWise()`, the sequence will rotate clockwise or counterclockwise.

From now on, the real game will start.

3.4 Game play

Let's analyse `blinkingClockwise()`, which is very similar to `blinkingCounterClockwise()`, in Appendix 1.
There are four different LED's (`BLUE`, `GREEN`, `ORANGE` and `RED`). Those are macro and represent an integer. That is the reason why `currentLed` is incremented or decremented in the implementation.
An important note is that the button acts as an interrupt, and the interrupt routine is the only part in the code where `action` is changed in true.

Explanation of the functions

`blinkLed(int currentLed, int sleepTime)` turns the `currentLed` on and off with `sleepTime` between on and off. The code can be consulted in Appendix 2.

`gamePlay(int currentLed, bool clockwise)` deals with all the cases that can exist with the combination of the global variables: `game`, `interaction` and `action`. The if clauses are described in chronological way as written in Appendix 1.

1. When the LED will blink again and the interaction is still false, the interaction is set to true. The LED will blink a second time to alert the player that he/she needs to do an action.
2. Something should have been done but the player didn't do it, starts the game over ritual.
3. Nothing should have been done and the player did something, starts the game over ritual.
4. Something should have been done and the player did it. The global variables go back to normal state with an increased difficulty. Also the `currentLed` is in- or decremented in correct way.

5. Nothing special, only `currentLed` is in- or decremented in correct way.

The in- or decrementing of `currentLed` depends on the fact of the game is in clock- or counterclockwise state.
On the end of the function there is a check if `currentLed` has correct values.

4 Used programs

The source code can be found on GitHub [3] under `https://github.com/ArneVlietinck/Reaction_game`. While developing the code, the feature *issues* inside GitHub is used. This to list some to-dos and attach them to the commits to have a good overview of the solved or open issues. Besides some bugs and mean features, extra features are added inside the issues. All issues are closed, because they are solved or it is not necessary for the moment to solve them (extra features).

The project has an exhaustively documentation to easily understand or adapt the project. For a good overview and interface of the documentation the authors used Doxygen [7]. To use the interface go to the git repository and browse to /html/index.html. The pdf of the documentation generated by Doxygen is added in Appendix C.

To achieve a good group communication, Slack [5] was used. The communication platform helped us to do several discussions in an efficient way.

The LaTeX-document is made in ShareLaTeX [4], again to optimise the group performance.

5 Possible improvements

There are several improvements possible for this reaction game. The following enumeration is not complete and can be extended by new, creative ideas.

5.1 Input

The accelerometer can give extra possibilities for new game play. Some examples:

- When a certain LED blinks the board need to be pushed in a specific direction.
- When a certain sound is played the board need to be pushed in a specific direction.

5.2 Game play

The implementation of `level` can be used to insert several levels with different game play. A possible example:

- Level 1: the blinking game as implemented in the current version of the game.
- Level 2: a blinking game with the accelerometer.
- Level 3: a sound based game whether or not with the accelerometer.

When the player finished the third level, the game can start over again but with an incremented difficulty.

Appendices

Appendix A

Main

```c
int main()
{
    mainInitialisation();
    waitForButton();
    game = GAMEOVER;
    while(1)
    {
        if(game==GAMEOVER) {
            reInitialisation();
        }
        blinkingGame();
    }
}
```

Appendix B

Game play

1 blinkingClockwise()

```
void blinkingClockwise(int currentLed, int sleepTime)
{
  while(currentLed==BLUE && game!=GAMEOVER)
  {
    blinkLed(currentLed, sleepTime);
    currentLed = gamePlay(currentLed, true);
  }

  while(currentLed==GREEN && game!=GAMEOVER)
  {
    blinkLed(currentLed, sleepTime);
    currentLed = gamePlay(currentLed, true);
  }

  while(currentLed==ORANGE && game!=GAMEOVER)
  {
    blinkLed(currentLed, sleepTime);
    currentLed = gamePlay(currentLed, true);
  }

  while(currentLed==RED && game!=GAMEOVER)
  {
    blinkLed(currentLed, sleepTime);
    currentLed = gamePlay(currentLed, true);
  }
}
```

2 blinkLed()

```
void blinkLed(int currentLed, int sleepTime)
{
  if(currentLed==BLUE)
  {
    blueLed::high();
    Thread::sleep(sleepTime);
    blueLed::low();
    Thread::sleep(sleepTime);
  }
  else if(currentLed==GREEN)
  {
    greenLed::high();
    Thread::sleep(sleepTime);
    greenLed::low();
    Thread::sleep(sleepTime);
  }
  else if(currentLed==ORANGE)
  {
    orangeLed::high();
    Thread::sleep(sleepTime);
    orangeLed::low();
    Thread::sleep(sleepTime);
  }
  else if(currentLed==RED)
  {
    redLed::high();
    Thread::sleep(sleepTime);
    redLed::low();
    Thread::sleep(sleepTime);
  }
}
```

3 gamePlay()

```
int gamePlay(int currentLed, bool clockwise)
{
    pthread_mutex_lock(&mutex);
    if(interaction==false && shouldBlinkAgain()==true)
    {
        interaction = true;
    }
    else if(interaction==true && action==false)
    {
        pthread_mutex_unlock(&mutex);
        gameOver();
        pthread_mutex_lock(&mutex);
    }
    else if(interaction==false && action==true)
    {
        pthread_mutex_unlock(&mutex);
        gameOver();
        pthread_mutex_lock(&mutex);
    }
    else if(interaction==true && action==true)
    {
        interaction = false;
        action = false;
        if(clockwise)
        {
          currentLed++;
        }
        else
        {
          currentLed--;
        }
        difficulty++;
        level++;
    }
    else
    {
      if(clockwise)
      {
        currentLed++;
      }
      else
      {
        currentLed--;
      }
    }

    pthread_mutex_unlock(&mutex);

    if(currentLed>RED)
    {
        currentLed = BLUE;
    }

    if(currentLed<BLUE)
    {
        currentLed = RED;
    }

    return currentLed;
}
```

Appendix C

Doxygen manual

Reaction_game

v1.0

Generated by Doxygen 1.8.13

Contents

1 Blinking Reaction Game **1**

1.1 Introduction . . . 1

1.2 Gameplay . . . 1

1.3 Improvements . . . 1

2 Reaction_game **3**

3 Hierarchical Index **5**

3.1 Class Hierarchy . . . 5

4 Class Index **7**

4.1 Class List . . . 7

5 File Index **9**

5.1 File List . . . 9

6 Class Documentation **11**

6.1 ADPCMSound Class Reference . . . 11

6.1.1 Detailed Description . . . 11

6.1.2 Constructor & Destructor Documentation . . . 11

6.1.2.1 ADPCMSound() . . . 11

6.1.3 Member Function Documentation . . . 12

6.1.3.1 fillMonoBuffer() . . . 12

6.1.3.2 fillStereoBuffer() . . . 12

6.1.3.3 rewind() . . . 13

6.2 Player Class Reference . . . 13

6.2.1 Detailed Description 13

6.2.2 Member Function Documentation 13

6.2.2.1 instance() 14

6.2.2.2 isPlaying() 14

6.2.2.3 play() 14

6.3 Sound Class Reference 14

6.3.1 Detailed Description 15

6.3.2 Constructor & Destructor Documentation 15

6.3.2.1 ∼Sound() 15

6.3.3 Member Function Documentation 15

6.3.3.1 fillMonoBuffer() 15

6.3.3.2 fillStereoBuffer() 16

6.3.3.3 rewind() 16

7 File Documentation **17**

7.1 adpcm.c File Reference 17

7.1.1 Function Documentation 17

7.1.1.1 ADPCM_Decode() 17

7.1.1.2 ADPCM_Encode() 18

7.1.2 Variable Documentation 18

7.1.2.1 IndexTable 18

7.1.2.2 StepSizeTable 18

7.2 adpcm.d File Reference 19

7.3 adpcm.h File Reference 19

7.3.1 Function Documentation 19

7.3.1.1 ADPCM_Decode() 19

7.3.1.2 ADPCM_Encode() 19

7.4 button.cpp File Reference 20

7.4.1 Detailed Description 20

7.4.2 Typedef Documentation 20

7.4.2.1 button 21

7.4.3 Function Documentation 21
7.4.3.1 __attribute__() [1/2] 21
7.4.3.2 __attribute__() [2/2] 21
7.4.3.3 configureButtonInterrupt() 21
7.4.3.4 waitForButton() 21
7.4.4 Variable Documentation 21
7.4.4.1 action 21
7.4.4.2 mutex 22
7.5 button.d File Reference 22
7.6 button.h File Reference 22
7.6.1 Detailed Description 22
7.6.2 Function Documentation 22
7.6.2.1 configureButtonInterrupt() 22
7.6.2.2 waitForButton() 22
7.7 Buzzer.h File Reference 23
7.7.1 Detailed Description 23
7.7.2 Variable Documentation 23
7.7.2.1 buzzer_bin 23
7.7.2.2 buzzer_bin_len 23
7.8 convert.cpp File Reference 24
7.8.1 Function Documentation 24
7.8.1.1 main() 24
7.8.1.2 run() 24
7.9 game.cpp File Reference 24
7.9.1 Detailed Description 25
7.9.2 Typedef Documentation 25
7.9.2.1 button 25
7.9.3 Function Documentation 25
7.9.3.1 buzzerSound() 26
7.9.3.2 gameOver() 26

7.9.3.3 gamePlay() 26

7.9.3.4 highscoreSound() 27

7.9.3.5 shouldBlinkAgain() 27

7.9.4 Variable Documentation 27

7.9.4.1 action 27

7.9.4.2 difficulty 28

7.9.4.3 game 28

7.9.4.4 highscore 28

7.9.4.5 interaction 28

7.9.4.6 level 28

7.9.4.7 mutex 28

7.10 game.d File Reference 28

7.11 game.h File Reference 28

7.11.1 Detailed Description 29

7.11.2 Macro Definition Documentation 29

7.11.2.1 GAMEOVER 29

7.11.3 Function Documentation 29

7.11.3.1 buzzerSound() 29

7.11.3.2 gameOver() 30

7.11.3.3 gamePlay() 30

7.11.3.4 highscoreSound() 31

7.11.3.5 shouldBlinkAgain() 31

7.12 highscore.h File Reference 31

7.12.1 Detailed Description 32

7.12.2 Variable Documentation 32

7.12.2.1 highscore_bin 32

7.12.2.2 highscore_bin_len 32

7.13 led.cpp File Reference 32

7.13.1 Detailed Description 33

7.13.2 Typedef Documentation 33

7.13.2.1 blueLed 33
7.13.2.2 greenLed 34
7.13.2.3 orangeLed 34
7.13.2.4 redLed 34
7.13.3 Function Documentation 34
7.13.3.1 blinkingClockwise() 34
7.13.3.2 blinkingCounterClockwise() 35
7.13.3.3 blinkingGame() 35
7.13.3.4 blinkingsequence() 35
7.13.3.5 blinkLed() 36
7.13.3.6 calculateSleepTime() 36
7.13.3.7 clockOrCounterClockWise() 37
7.13.3.8 initLeds() 37
7.13.3.9 onOffBlinking() 37
7.13.3.10 turnAllOff() 37
7.13.3.11 turnAllOn() 37
7.13.4 Variable Documentation 38
7.13.4.1 difficulty 38
7.13.4.2 game 38
7.13.4.3 level 38
7.14 led.d File Reference 38
7.15 led.h File Reference 38
7.15.1 Detailed Description 39
7.15.2 Macro Definition Documentation 39
7.15.2.1 BLUE 39
7.15.2.2 GREEN 39
7.15.2.3 ORANGE 39
7.15.2.4 RED 39
7.15.3 Function Documentation 39
7.15.3.1 blinkingClockwise() 39

7.15.3.2 blinkingCounterClockwise() 40
7.15.3.3 blinkingGame() 40
7.15.3.4 blinkingsequence() 41
7.15.3.5 blinkLed() 41
7.15.3.6 calculateSleepTime() 41
7.15.3.7 clockOrCounterClockWise() 42
7.15.3.8 initLeds() 42
7.15.3.9 onOffBlinking() 42
7.15.3.10 turnAllOff() 43
7.15.3.11 turnAllOn() 43
7.16 main.cpp File Reference 43
7.16.1 Detailed Description 43
7.16.2 Function Documentation 44
7.16.2.1 main() 44
7.16.2.2 mainInitialisation() 44
7.16.2.3 reInitialisation() 44
7.16.3 Variable Documentation 45
7.16.3.1 action 45
7.16.3.2 difficulty 45
7.16.3.3 game 45
7.16.3.4 highscore 45
7.16.3.5 interaction 45
7.16.3.6 level 45
7.16.3.7 mutex 46
7.17 main.d File Reference 46
7.18 player.cpp File Reference 46
7.18.1 Typedef Documentation 46
7.18.1.1 i2c 46
7.18.1.2 lrck 47
7.18.1.3 mclk 47
7.18.1.4 reset 47
7.18.1.5 scl 47
7.18.1.6 sclk 47
7.18.1.7 sda 47
7.18.1.8 sdin 47
7.18.2 Function Documentation 47
7.18.2.1 __attribute__() [1/2] 48
7.18.2.2 __attribute__() [2/2] 48
7.18.2.3 cs43l22volume() 48
7.19 player.d File Reference 48
7.20 player.h File Reference 48
7.21 README.md File Reference 48

Index 49

Chapter 1

Blinking Reaction Game

1.1 Introduction

This game is developed with the Miosix kernel and made for the discoveryboard STM32F407G. It is a project for Embedded Systems 1 and Advanced Operating Systems of prof. William Fornaciari at the Politecnico di Milano.

1.2 Gameplay

The game starts when the player pushes on the button. To emphasize the start of the game, the four LED's will blink once. From then on each button push can cause gameover. The blinking ritual will start with the blue LED and goes random clockwise or counterclockwise. The main issue of the game is pushing the button when the LED (blue/green/orange/red) blinks twice. If the player pushes the button correctly the time between the blinks will be shorter and consequentely the game will become more difficult. If the player pushes the button to late or in a wrong situation then the board will play a buzzer sound or a high score sound. After the buzzer/high score sound, the game will start over again with blinking the LED's 3 times. Have Fun!

1.3 Improvements

Some improvements are listed in the issue area of the git repository. They are signed with the label "extra feature".

Chapter 2

Reaction_game

This game is developed with the Miosix kernel and made for the discoveryboard STM32F407G. It is a project for Embedded Systems 1 and Advanced Operating Systems of prof. William Fornaciari at the Politecnico di Milano.

Miosix_kernel

You can find information on how to configure and use the kernel at the following url: `http://miosix.org`

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Player 13
Sound 14
    ADPCMSound 11

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

ADPCMSound 11
Player 13
Sound 14

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

adpcm.c 17
adpcm.d 19
adpcm.h 19
button.cpp 20
button.d 22
button.h 22
Buzzer.h
The h-file for the buzzer sound. It contains the buzzer_bin[], which representates the char array for the buzzer sound to emphasize the fault of the player 23
convert.cpp 24
game.cpp
The cpp-file of the gameplay library 24
game.d 28
game.h
The h-file of the gameplay library 28
highscore.h
The h-file for the high score sound. It contains the highscore_bin[], which representates the char array for the high score sound to emphasize a new high score of the player 31
led.cpp
The cpp-file of the LED library 32
led.d 38
led.h
The h-file of the LED library 38
main.cpp 43
main.d 46
player.cpp 46
player.d 48
player.h 48

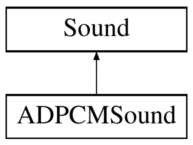
Chapter 6

Class Documentation

6.1 ADPCMSound Class Reference

```
#include <player.h>
```

Inheritance diagram for ADPCMSound:



Public Member Functions

- ADPCMSound (const unsigned char ∗data, int size)
- virtual bool fillMonoBuffer (unsigned short ∗buffer, int size)
- virtual bool fillStereoBuffer (unsigned short ∗buffer, int length)
- virtual void rewind ()

6.1.1 Detailed Description

Class to play a buffer contatinig ADPCM compressed audio

6.1.2 Constructor & Destructor Documentation

6.1.2.1 ADPCMSound()

```
ADPCMSound::ADPCMSound (
            const unsigned char ∗ data,
            int size )  [inline]
```

Constructor

Parameters

data	ADPCM encoded data. Ownership of the buffer remains of the caller, which is responsible to make sure it remains valid for the entire lifetime of this class. This is not a problem in the expected use case of the buffer being const and static
size	size of data

6.1.3 Member Function Documentation

6.1.3.1 fillMonoBuffer()

```
bool ADPCMSound::fillMonoBuffer (
            unsigned short * buffer,
            int size )  [virtual]
```

Fill a buffer with audio samples

Parameters

buffer	a buffer where audio samples (16bit unsigned, 44100Hz) are to be stored. If there is not enough data to fill the entire buffer the remaining part must be filled with 0
length	buffer length, must be divisible by two

Returns

true if this is the last valif buffer (eof encountered)

Implements Sound.

6.1.3.2 fillStereoBuffer()

```
bool ADPCMSound::fillStereoBuffer (
            unsigned short * buffer,
            int length )  [virtual]
```

Fill a stereo buffer with audio samples

Parameters

buffer	a buffer where audio samples (16bit unsigned, 44100Hz) are to be stored. If there is not enough data to fill the entire buffer. The buffer format is alternating left-right samples, so buffer[0] is left buffer[1] is right, buffer[2] is again left... the remaining part must be filled with 0
length	buffer length, must be divisible by four

Returns

true if this is the last valif buffer (eof encountered)

Implements Sound.

6.1.3.3 rewind()

```
void ADPCMSound::rewind ( )  [virtual]
```

Rewind the internal sound pointer so that succesive calls to fillBuffer() start brom the beginning of the sound.

Implements Sound.

The documentation for this class was generated from the following files:

- player.h
- player.cpp

6.2 Player Class Reference

```
#include <player.h>
```

Public Member Functions

- void play (Sound &sound)
- bool isPlaying () const

Static Public Member Functions

- static Player & instance ()

6.2.1 Detailed Description

Class to play an audio file on the STM32's DAC

6.2.2 Member Function Documentation

6.2.2.1 instance()

```
Player & Player::instance ( )  [static]
```

Returns

an instance of the player (singleton)

6.2.2.2 isPlaying()

```
bool Player::isPlaying ( ) const
```

Returns

true if the resource is busy

6.2.2.3 play()

```
void Player::play (
            Sound & sound )
```

Play an audio file, returning after the file has coompleted playing

Parameters

sound	sound file to play

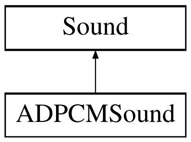
The documentation for this class was generated from the following files:

- player.h
- player.cpp

6.3 Sound Class Reference

```
#include <player.h>
```

Inheritance diagram for Sound:



Public Member Functions

- virtual bool fillMonoBuffer (unsigned short ∗buffer, int length)=0
- virtual bool fillStereoBuffer (unsigned short ∗buffer, int length)=0
- virtual void rewind ()=0
- virtual ∼Sound ()

6.3.1 Detailed Description

Interface class from where all sound classes derive

6.3.2 Constructor & Destructor Documentation

6.3.2.1 ∼Sound()

```
Sound::∼Sound ( )  [virtual]
```

Destructor

6.3.3 Member Function Documentation

6.3.3.1 fillMonoBuffer()

```
virtual bool Sound::fillMonoBuffer (
            unsigned short ∗ buffer,
            int length )  [pure virtual]
```

Fill a buffer with audio samples

Parameters

buffer	a buffer where audio samples (16bit unsigned, 44100Hz) are to be stored. If there is not enough data to fill the entire buffer the remaining part must be filled with 0
length	buffer length, must be divisible by two

Returns

true if this is the last valif buffer (eof encountered)

Implemented in ADPCMSound.

6.3.3.2 fillStereoBuffer()

```
virtual bool Sound::fillStereoBuffer (
             unsigned short * buffer,
             int length )  [pure virtual]
```

Fill a stereo buffer with audio samples

Parameters

buffer	a buffer where audio samples (16bit unsigned, 44100Hz) are to be stored. If there is not enough data to fill the entire buffer. The buffer format is alternating left-right samples, so buffer[0] is left buffer[1] is right, buffer[2] is again left... the remaining part must be filled with 0
length	buffer length, must be divisible by four

Returns

true if this is the last valif buffer (eof encountered)

Implemented in ADPCMSound.

6.3.3.3 rewind()

```
virtual void Sound::rewind ( )  [pure virtual]
```

Rewind the internal sound pointer so that succesive calls to fillBuffer() start brom the beginning of the sound.

Implemented in ADPCMSound.

The documentation for this class was generated from the following files:

- player.h
- player.cpp

Chapter 7

File Documentation

7.1 adpcm.c File Reference

```
#include "adpcm.h"
```

Functions

- uint8_t ADPCM_Encode (int32_t sample)

  ADPCM_Encode.
- int16_t ADPCM_Decode (uint8_t code)

  ADPCM_Decode.

Variables

- const uint16_t StepSizeTable [89]
- const int8_t IndexTable [16] ={0xff,0xff,0xff,0xff,2,4,6,8,0xff,0xff,0xff,0xff,2,4,6,8}

7.1.1 Function Documentation

7.1.1.1 ADPCM_Decode()

```
int16_t ADPCM_Decode (
            uint8_t code )
```

ADPCM_Decode.

Parameters

code	a byte containing a 4-bit ADPCM sample.

Return values

	16-bit ADPCM sample

7.1.1.2 ADPCM_Encode()

```
uint8_t ADPCM_Encode (
            int32_t sample )
```

ADPCM_Encode.

Parameters

sample	a 16-bit PCM sample

Return values

	a 4-bit ADPCM sample

7.1.2 Variable Documentation

7.1.2.1 IndexTable

```
const int8_t IndexTable[16] ={0xff,0xff,0xff,0xff,2,4,6,8,0xff,0xff,0xff,0xff,2,4,6,8}
```

7.1.2.2 StepSizeTable

```
const uint16_t StepSizeTable[89]
```

Initial value:

```
={7,8,9,10,11,12,13,14,16,17,
                              19,21,23,25,28,31,34,37,41,45,
                              50,55,60,66,73,80,88,97,107,118,
                              130,143,157,173,190,209,230,253,279,307,
                              337,371,408,449,494,544,598,658,724,796,
                              876,963,1060,1166,1282,1411,1552,1707,1878,2066,
                              2272,2499,2749,3024,3327,3660,4026,4428,4871,5358,
                              5894,6484,7132,7845,8630,9493,10442,11487,12635,13899,
                              15289,16818,18500,20350,22385,24623,27086,29794,32767}
```

7.2 adpcm.d File Reference

7.3 adpcm.h File Reference

```
#include <stdint.h>
```

Functions

- uint8_t ADPCM_Encode (int32_t sample)

  ADPCM_Encode.
- int16_t ADPCM_Decode (uint8_t code)

  ADPCM_Decode.

7.3.1 Function Documentation

7.3.1.1 ADPCM_Decode()

```
int16_t ADPCM_Decode (
            uint8_t code )
```

ADPCM_Decode.

Parameters

code	a byte containing a 4-bit ADPCM sample.

Return values

	16-bit ADPCM sample

7.3.1.2 ADPCM_Encode()

```
uint8_t ADPCM_Encode (
            int32_t sample )
```

ADPCM_Encode.

Parameters

sample	a 16-bit PCM sample

Return values

	a 4-bit ADPCM sample

7.4 button.cpp File Reference

```
#include "button.h"
#include <miosix.h>
#include <miosix/kernel/scheduler/scheduler.h>
#include <pthread.h>
```

Typedefs

- typedef Gpio< GPIOA_BASE, 0 > button

Functions

- void __attribute__ ((naked)) EXTI0_IRQHandler()
- void __attribute__ ((used)) EXTI0HandlerImpl()
- void configureButtonInterrupt ()
- void waitForButton ()

Variables

- pthread_mutex_t mutex
- bool action

7.4.1 Detailed Description

Author

Federico Terraneo
Simon Mastrodicasa
Arne Vlietinck

Version

1.0

Date

30/12/2017

7.4.2 Typedef Documentation

7.4.2.1 button

```
typedef Gpio<GPIOA_BASE,0> button
```

7.4.3 Function Documentation

7.4.3.1 __attribute__() [1/2]

```
void __attribute__ (
            (naked)  )
```

7.4.3.2 __attribute__() [2/2]

```
void __attribute__ (
            (used)  )
```

7.4.3.3 configureButtonInterrupt()

```
void configureButtonInterrupt ( )
```

7.4.3.4 waitForButton()

```
void waitForButton ( )
```

7.4.4 Variable Documentation

7.4.4.1 action

```
bool action
```

Boolean which represents the action of a player. If (action==true), the player did an action. Elseif (action==false), the player didn't do an action.

7.4.4.2 mutex

```
pthread_mutex_t mutex
```

A pthread_mutex_t variable to prevent a race condition when changing action.

7.5 button.d File Reference

7.6 button.h File Reference

Functions

- void configureButtonInterrupt ()
- void waitForButton ()

7.6.1 Detailed Description

Author

Federico Terraneo
Simon Mastrodicasa
Arne Vlietinck

Version

1.0

Date

30/12/2017

7.6.2 Function Documentation

7.6.2.1 configureButtonInterrupt()

```
void configureButtonInterrupt ( )
```

7.6.2.2 waitForButton()

```
void waitForButton ( )
```

7.7 Buzzer.h File Reference

The h-file for the buzzer sound. It contains the buzzer_bin[], which representates the char array for the buzzer sound to emphasize the fault of the player.

Variables

- const unsigned char buzzer_bin [ ]
- const unsigned int buzzer_bin_len = 67392

7.7.1 Detailed Description

The h-file for the buzzer sound. It contains the buzzer_bin[], which representates the char array for the buzzer sound to emphasize the fault of the player.

Author

Simon Mastrodicasa
Arne Vlietinck

Version

1.0

Date

30/12/2017

7.7.2 Variable Documentation

7.7.2.1 buzzer_bin

```
const unsigned char buzzer_bin[]
```

7.7.2.2 buzzer_bin_len

```
const unsigned int buzzer_bin_len = 67392
```

7.8 convert.cpp File Reference

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cstdlib>
#include "adpcm.h"
```

Functions

- void run (const string &s)
- int main (int argc, char ∗argv[ ])

7.8.1 Function Documentation

7.8.1.1 main()

```
int main (
            int argc,
            char ∗ argv[ ] )
```

7.8.1.2 run()

```
void run (
            const string & s )
```

7.9 game.cpp File Reference

The cpp-file of the gameplay library.

```
#include <miosix.h>
#include <pthread.h>
#include "game.h"
#include "led.h"
#include "player.h"
#include "buzzer.h"
#include "highscore.h"
```

Typedefs

- typedef Gpio< GPIOA_BASE, 0 > button

Functions

- bool shouldBlinkAgain ()
- void buzzerSound ()
- void highscoreSound ()
- void gameOver ()
- int gamePlay (int currentLed, bool clockwise)

Variables

- bool action
- bool game
- bool interaction
- int difficulty
- int level
- int highscore
- pthread_mutex_t mutex

7.9.1 Detailed Description

The cpp-file of the gameplay library.

Author

Simon Mastrodicasa
Arne Vlietinck

Version

1.0

Date

30/12/2017

7.9.2 Typedef Documentation

7.9.2.1 button

```
typedef Gpio<GPIOA_BASE,0> button
```

7.9.3 Function Documentation

7.9.3.1 buzzerSound()

```
void buzzerSound ( )
```

Function which plays the buzzer sound.

7.9.3.2 gameOver()

```
void gameOver ( )
```

Function which does the gameOver ritual.

Postcondition

If and only if (level>highscore), the high score sound is played.
If and only if (level>highscore), highscore is set to level.
If (level<=highscore), the buzzer sound is played.
The gameOver blinking ritual is played.
Game is set to GAMEOVER.

See also

buzzerSound()
highscoreSound()
onOffBlinking()

7.9.3.3 gamePlay()

```
int gamePlay (
            int currentLed,
            bool clockwise )
```

Function which takes care of the gameplay. It checks the several possible conditions and increment or decrement the several game parameters. When the player did a wrong interaction the gameover ritual is started.

Parameters

int	currentled - The number of the current LED.
bool	clockwise - Represents the order of blinking the LED's.

Postcondition

If and only if (interaction==false && shouldBlinkAgain()==true), interaction is set true.
If (interaction==true && action==false), gameOver() is excecuted.
If (interaction==false && action==true), gameOver() is excecuted.
If and only if (interaction==true && action==true), interaction is set false.
If and only if (interaction==true && action==true), action is set false.
If and only if (interaction==true && action==true), difficulty is incremented by one.

If and only if (interaction==true && action==true), level is incremented by one.
If and only if (clockwise==true), currentLed is incremented by one.
If and only if (clockwise!=false), currentLed is decremented by one.
If and only if (currentLed>RED), currentLed is set to the smallest LED (BLUE).
If and only if (currentLed<BLUE), currentLed is set to the biggest LED (RED).

Returns

int currentLed - The number of the current LED.

See also

gameOver()
shouldBlinkAgain();

7.9.3.4 highscoreSound()

```
void highscoreSound ( )
```

Function which plays the high score sound.

7.9.3.5 shouldBlinkAgain()

```
bool shouldBlinkAgain ( )
```

Function which calculate if the LED should blink for a second time.

Returns

Returns a random boolean which tells if the LED should blink again.

Note

The boolean is in 30% of the situations true and in the other 70% false.

7.9.4 Variable Documentation

7.9.4.1 action

```
bool action
```

Boolean which represents the action of a player. If (action==true), the player did an action. Elseif (action==false), the player didn't do an action.

7.9.4.2 difficulty

```
int difficulty
```

Integer which represents the difficulty level of the game. Higher integer means higher degree of difficulty. It affects the time between the blinking LED's.

7.9.4.3 game

```
bool game
```

Boolean which represents the state of the game. If (game==1), the current game is finished. Elseif (game==0), the current game is still running.

7.9.4.4 highscore

```
int highscore
```

Integer which represents the current high score of the game.

7.9.4.5 interaction

```
bool interaction
```

Boolean which represents the need of a players' interaction. When interaction is true, the player must do something (e.g. press the user button) to avoid a game over.

7.9.4.6 level

```
int level
```

Integer which represents the level of the game.

7.9.4.7 mutex

```
pthread_mutex_t mutex
```

A pthread_mutex_t variable to prevent a race condition when changing action.

7.10 game.d File Reference

7.11 game.h File Reference

The h-file of the gameplay library.

Macros

- #define GAMEOVER 1

Functions

- bool shouldBlinkAgain ()
- void buzzerSound ()
- void highscoreSound ()
- void gameOver ()
- int gamePlay (int currentLed, bool clockwise)

7.11.1 Detailed Description

The h-file of the gameplay library.

Author

Simon Mastrodicasa
Arne Vlietinck

Version

1.0

Date

30/12/2017

7.11.2 Macro Definition Documentation

7.11.2.1 GAMEOVER

```
#define GAMEOVER 1
```

7.11.3 Function Documentation

7.11.3.1 buzzerSound()

```
void buzzerSound ( )
```

Function which plays the buzzer sound.

7.11.3.2 gameOver()

```
void gameOver ( )
```

Function which does the gameOver ritual.

Postcondition

If and only if (level>highscore), the high score sound is played.
If and only if (level>highscore), highscore is set to level.
If (level<=highscore), the buzzer sound is played.
The gameOver blinking ritual is played.
Game is set to GAMEOVER.

See also

buzzerSound()
highscoreSound()
onOffBlinking()

7.11.3.3 gamePlay()

```
int gamePlay (
             int currentLed,
             bool clockwise )
```

Function which takes care of the gameplay. It checks the several possible conditions and increment or decrement the several game parameters. When the player did a wrong interaction the gameover ritual is started.

Parameters

int	currentled - The number of the current LED.
bool	clockwise - Represents the order of blinking the LED's.

Postcondition

If and only if (interaction==false && shouldBlinkAgain()==true), interaction is set true.
If (interaction==true && action==false), gameOver() is excecuted.
If (interaction==false && action==true), gameOver() is excecuted.
If and only if (interaction==true && action==true), interaction is set false.
If and only if (interaction==true && action==true), action is set false.
If and only if (interaction==true && action==true), difficulty is incremented by one.
If and only if (interaction==true && action==true), level is incremented by one.
If and only if (clockwise==true), currentLed is incremented by one.
If and only if (clockwise!=false), currentLed is decremented by one.
If and only if (currentLed>RED), currentLed is set to the smallest LED (BLUE).
If and only if (currentLed<BLUE), currentLed is set to the biggest LED (RED).

Returns

int currentLed - The number of the current LED.

See also

gameOver()
shouldBlinkAgain();

7.11.3.4 highscoreSound()

```
void highscoreSound ( )
```

Function which plays the high score sound.

7.11.3.5 shouldBlinkAgain()

```
bool shouldBlinkAgain ( )
```

Function which calculate if the LED should blink for a second time.

Returns

Returns a random boolean which tells if the LED should blink again.

Note

The boolean is in 30% of the situations true and in the other 70% false.

7.12 highscore.h File Reference

The h-file for the high score sound. It contains the highscore_bin[], which representates the char array for the high score sound to emphasize a new high score of the player.

Variables

- const unsigned char highscore_bin [ ]
- const unsigned int highscore_bin_len = 136366

7.12.1 Detailed Description

The h-file for the high score sound. It contains the highscore_bin[], which representates the char array for the high score sound to emphasize a new high score of the player.

Author

Simon Mastrodicasa
Arne Vlietinck

Version

1.0

Date

30/12/2017

7.12.2 Variable Documentation

7.12.2.1 highscore_bin

```
const unsigned char highscore_bin[]
```

7.12.2.2 highscore_bin_len

```
const unsigned int highscore_bin_len = 136366
```

7.13 led.cpp File Reference

The cpp-file of the LED library.

```
#include <miosix.h>
#include "led.h"
#include "game.h"
```

Typedefs

- typedef Gpio< GPIOD_BASE, 12 > greenLed
- typedef Gpio< GPIOD_BASE, 13 > orangeLed
- typedef Gpio< GPIOD_BASE, 14 > redLed
- typedef Gpio< GPIOD_BASE, 15 > blueLed

Functions

- void initLeds ()
- bool clockOrCounterClockWise ()
- int calculateSleepTime (int difficulty)
- void blinkLed (int currentLed, int sleepTime)
- void blinkingClockwise (int currentLed, int sleepTime)
- void blinkingCounterClockwise (int currentLed, int sleepTime)
- void blinkingsequence (int currentLed, int sleepTime, int level)
- void blinkingGame ()
- void turnAllOn ()
- void turnAllOff ()
- void onOffBlinking (int times)

Variables

- bool game
- int difficulty
- int level

7.13.1 Detailed Description

The cpp-file of the LED library.

Author

Simon Mastrodicasa
Arne Vlietinck

Version

1.0

Date

30/12/2017

7.13.2 Typedef Documentation

7.13.2.1 blueLed

```
typedef Gpio<GPIOD_BASE,15> blueLed
```

7.13.2.2 greenLed

```
typedef Gpio<GPIOD_BASE,12> greenLed
```

7.13.2.3 orangeLed

```
typedef Gpio<GPIOD_BASE,13> orangeLed
```

7.13.2.4 redLed

```
typedef Gpio<GPIOD_BASE,14> redLed
```

7.13.3 Function Documentation

7.13.3.1 blinkingClockwise()

```
void blinkingClockwise (
             int currentLed,
             int sleepTime )
```

Function for the clockwise blinking.

Parameters

int	currentLed - The number of the current LED.
int	sleepTime - The sleeptime between the on and off initialistation of the LED.

Postcondition

blinkLed(currentLed, sleepTime)
Sets currentLed to gamePlay(currentLed)

See also

gamePlay()
blinkLed()

7.13.3.2 blinkingCounterClockwise()

```
void blinkingCounterClockwise (
            int currentLed,
            int sleepTime )
```

Function for the counterclockwise blinking.

Parameters

int	currentLed - The number of the current LED.
int	sleepTime - The sleeptime between the on and off initialistation of the LED.

Postcondition

blinkLed(currentLed, sleepTime)
Sets currentLed to gamePlay(currentLed)

See also

gamePlay()
blinkLed()

7.13.3.3 blinkingGame()

```
void blinkingGame ( )
```

Function with the game ritual.

See also

calculateSleepTime(int difficulty)
blinkingsequence(int currentLed, int sleepTime, int level)

7.13.3.4 blinkingsequence()

```
void blinkingsequence (
            int currentLed,
            int sleepTime,
            int level )
```

Function with the gameplay for the blinking sequence.

Parameters

int	currentLed - The number of the current LED.
int	sleepTime - The sleeptime between the on and off initialistation of the LED.
int	level - Specifies the level of the current game.

Postcondition

If and only if (clockOrCounterClockWise() == true), blinkingClockwise() is excecuted.
If and only if (clockOrCounterClockWise() != true), blinkingCounterClockwise() is excecuted.

See also

clockOrCounterClockWise()
blinkingClockwise()
blinkingCounterClockwise()

7.13.3.5 blinkLed()

```
void blinkLed (
             int currentLed,
             int sleepTime )
```

Function which blinks the currentLed (BLUE, GREEN, ORANGE, RED).

Parameters

int	currentLed - The number of the current LED.
int	sleepTime - The sleeptime between the on and off initialistation of the LED.

Postcondition

Repeat the sequence: Turn currentLed high, sleep for sleepTime, turn currentLed low, sleep for sleepTime.

7.13.3.6 calculateSleepTime()

```
int calculateSleepTime (
             int difficulty )
```

Calculate the sleepTime between the on and off initialistation of the LED.

Parameters

int	difficulty - The current difficulty level of the game.

Returns

int sleepTime - The sleepTime between the on and off initialistation of the LED.

7.13.3.7 clockOrCounterClockWise()

```
bool clockOrCounterClockWise ( )
```

Function which calculate if the LED should blink in clock or counterclockwise.

Returns

Returns a random boolean which tells if the LED's should blink in clock or counterclockwise.

7.13.3.8 initLeds()

```
void initLeds ( )
```

Initialisation of the Green, Orange, Red and Blue LED in output mode.

7.13.3.9 onOffBlinking()

```
void onOffBlinking (
            int times )
```

On off blinking ritual by LED's.

Parameters

int	times - Number of times the LED's should blink.

Postcondition

Repeat - times - the sequence: turn all LED's on, sleep for 500ms, turn all LED's off, sleep for 500ms.

7.13.3.10 turnAllOff()

```
void turnAllOff ( )
```

Turn Red, Blue, Green and Orange LED's off.

7.13.3.11 turnAllOn()

```
void turnAllOn ( )
```

Turn Red, Blue, Green and Orange LED's on.

7.13.4 Variable Documentation

7.13.4.1 difficulty

```
int difficulty
```

Integer which represents the difficulty level of the game. Higher integer means higher degree of difficulty. It affects the time between the blinking LED's.

7.13.4.2 game

```
bool game
```

Boolean which represents the state of the game. If (game==1), the current game is finished. Elseif (game==0), the current game is still running.

7.13.4.3 level

```
int level
```

Integer which represents the level of the game.

7.14 led.d File Reference

7.15 led.h File Reference

The h-file of the LED library.

Macros

- #define BLUE 1
- #define GREEN 2
- #define ORANGE 3
- #define RED 4

Functions

- void initLeds ()
- bool clockOrCounterClockWise ()
- int calculateSleepTime (int difficulty)
- void blinkLed (int currentLed, int sleepTime)
- void blinkingClockwise (int currentLed, int sleepTime)
- void blinkingCounterClockwise (int currentLed, int sleepTime)
- void blinkingsequence (int currentLed, int sleepTime, int level)
- void blinkingGame ()
- void turnAllOn ()
- void turnAllOff ()
- void onOffBlinking (int times)

7.15.1 Detailed Description

The h-file of the LED library.

Author

Simon Mastrodicasa
Arne Vlietinck

Version

1.0

Date

30/12/2017

7.15.2 Macro Definition Documentation

7.15.2.1 BLUE

```
#define BLUE 1
```

7.15.2.2 GREEN

```
#define GREEN 2
```

7.15.2.3 ORANGE

```
#define ORANGE 3
```

7.15.2.4 RED

```
#define RED 4
```

7.15.3 Function Documentation

7.15.3.1 blinkingClockwise()

```
void blinkingClockwise (
             int currentLed,
             int sleepTime )
```

Function for the clockwise blinking.

Parameters

int	currentLed - The number of the current LED.
int	sleepTime - The sleeptime between the on and off initialistation of the LED.

Postcondition

blinkLed(currentLed, sleepTime)
Sets currentLed to gamePlay(currentLed)

See also

gamePlay()
blinkLed()

7.15.3.2 blinkingCounterClockwise()

```
void blinkingCounterClockwise (
             int currentLed,
             int sleepTime )
```

Function for the counterclockwise blinking.

Parameters

int	currentLed - The number of the current LED.
int	sleepTime - The sleeptime between the on and off initialistation of the LED.

Postcondition

blinkLed(currentLed, sleepTime)
Sets currentLed to gamePlay(currentLed)

See also

gamePlay()
blinkLed()

7.15.3.3 blinkingGame()

```
void blinkingGame ( )
```

Function with the game ritual.

See also

calculateSleepTime(int difficulty)
blinkingsequence(int currentLed, int sleepTime, int level)

7.15.3.4 blinkingsequence()

```
void blinkingsequence (
             int currentLed,
             int sleepTime,
             int level )
```

Function with the gameplay for the blinking sequence.

Parameters

int	currentLed - The number of the current LED.
int	sleepTime - The sleeptime between the on and off initialistation of the LED.
int	level - Specifies the level of the current game.

Postcondition

If and only if (clockOrCounterClockWise() == true), blinkingClockwise() is excecuted.
If and only if (clockOrCounterClockWise() != true), blinkingCounterClockwise() is excecuted.

See also

clockOrCounterClockWise()
blinkingClockwise()
blinkingCounterClockwise()

7.15.3.5 blinkLed()

```
void blinkLed (
             int currentLed,
             int sleepTime )
```

Function which blinks the currentLed (BLUE, GREEN, ORANGE, RED).

Parameters

int	currentLed - The number of the current LED.
int	sleepTime - The sleeptime between the on and off initialistation of the LED.

Postcondition

Repeat the sequence: Turn currentLed high, sleep for sleepTime, turn currentLed low, sleep for sleepTime.

7.15.3.6 calculateSleepTime()

```
int calculateSleepTime (
             int difficulty )
```

Calculate the sleepTime between the on and off initialistation of the LED.

Parameters

int	difficulty - The current difficulty level of the game.

Returns

int sleepTime - The sleepTime between the on and off initialistation of the LED.

7.15.3.7 clockOrCounterClockWise()

```
bool clockOrCounterClockWise ( )
```

Function which calculate if the LED should blink in clock or counterclockwise.

Returns

Returns a random boolean which tells if the LED's should blink in clock or counterclockwise.

7.15.3.8 initLeds()

```
void initLeds ( )
```

Initialisation of the Green, Orange, Red and Blue LED in output mode.

7.15.3.9 onOffBlinking()

```
void onOffBlinking (
             int times )
```

On off blinking ritual by LED's.

Parameters

int	times - Number of times the LED's should blink.

Postcondition

Repeat - times - the sequence: turn all LED's on, sleep for 500ms, turn all LED's off, sleep for 500ms.

7.15.3.10 turnAllOff()

```
void turnAllOff ( )
```

Turn Red, Blue, Green and Orange LED's off.

7.15.3.11 turnAllOn()

```
void turnAllOn ( )
```

Turn Red, Blue, Green and Orange LED's on.

7.16 main.cpp File Reference

```
#include <miosix.h>
#include <pthread.h>
#include "led.h"
#include "game.h"
#include "player.h"
#include "button.h"
```

Functions

- void mainInitialisation ()
- void reInitialisation ()
- int main ()

Variables

- bool game
- bool interaction
- bool action
- int difficulty
- int level
- int highscore
- pthread_mutex_t mutex =PTHREAD_MUTEX_INITIALIZER

7.16.1 Detailed Description

Author

Simon Mastrodicasa
Arne Vlietinck

Version

1.0

Date

30/12/2017

7.16.2 Function Documentation

7.16.2.1 main()

```
int main ( )
```

7.16.2.2 mainInitialisation()

```
void mainInitialisation ( )
```

Initialisation process of the main.

Postcondition

The LED's are initialised.
The button is initialised.
highscore is set to 0.

See also

initLeds()
configureButtonInterrupt()

7.16.2.3 reInitialisation()

```
void reInitialisation ( )
```

Reinitialisation process of the main. This is used to reset the game without a complete reset by using the computer.

Postcondition

Difficulty is set on 1.
Interaction is set on false.
Action is set on false while protected by mutex.
Game is set on 0.
Level is set on 0.
onOffBlinking(1) is executed.

See also

onOffBlinking()

7.16.3 Variable Documentation

7.16.3.1 action

```
bool action
```

Boolean which represents the action of a player. If (action==true), the player did an action. Elseif (action==false), the player didn't do an action.

7.16.3.2 difficulty

```
int difficulty
```

Integer which represents the difficulty level of the game. Higher integer means higher degree of difficulty. It affects the time between the blinking LED's.

7.16.3.3 game

```
bool game
```

Boolean which represents the state of the game. If (game==1), the current game is finished. Elseif (game==0), the current game is still running.

7.16.3.4 highscore

```
int highscore
```

Integer which represents the current high score of the game.

7.16.3.5 interaction

```
bool interaction
```

Boolean which represents the need of a players' interaction. When interaction is true, the player must do something (e.g. press the user button) to avoid a game over.

7.16.3.6 level

```
int level
```

Integer which represents the level of the game.

7.16.3.7 mutex

```
pthread_mutex_t mutex =PTHREAD_MUTEX_INITIALIZER
```

A pthread_mutex_t variable to prevent a race condition when changing action.

7.17 main.d File Reference

7.18 player.cpp File Reference

```
#include <algorithm>
#include <stdexcept>
#include <cstring>
#include "miosix/kernel/scheduler/scheduler.h"
#include "util/software_i2c.h"
#include "adpcm.h"
#include "player.h"
```

Typedefs

- typedef Gpio< GPIOB_BASE, 6 > scl
- typedef Gpio< GPIOB_BASE, 9 > sda
- typedef Gpio< GPIOA_BASE, 4 > lrck
- typedef Gpio< GPIOC_BASE, 7 > mclk
- typedef Gpio< GPIOC_BASE, 10 > sclk
- typedef Gpio< GPIOC_BASE, 12 > sdin
- typedef Gpio< GPIOD_BASE, 4 > reset
- typedef SoftwareI2C< sda, scl > i2c

Functions

- void __attribute__ ((naked)) DMA1_Stream5_IRQHandler()
- void __attribute__ ((used)) I2SdmaHandlerImpl()
- void cs43l22volume (int db)

7.18.1 Typedef Documentation

7.18.1.1 i2c

```
typedef SoftwareI2C<sda,scl> i2c
```

7.18.1.2 lrck

```
typedef Gpio<GPIOA_BASE,4> lrck
```

7.18.1.3 mclk

```
typedef Gpio<GPIOC_BASE,7> mclk
```

7.18.1.4 reset

```
typedef Gpio<GPIOD_BASE,4> reset
```

7.18.1.5 scl

```
typedef Gpio<GPIOB_BASE,6> scl
```

7.18.1.6 sclk

```
typedef Gpio<GPIOC_BASE,10> sclk
```

7.18.1.7 sda

```
typedef Gpio<GPIOB_BASE,9> sda
```

7.18.1.8 sdin

```
typedef Gpio<GPIOC_BASE,12> sdin
```

7.18.2 Function Documentation

7.18.2.1 __attribute__() [1/2]

```
void __attribute__ (
            (naked)  )
```

DMA end of transfer interrupt

7.18.2.2 __attribute__() [2/2]

```
void __attribute__ (
            (used)  )
```

DMA end of transfer interrupt actual implementation

7.18.2.3 cs43l22volume()

```
void cs43l22volume (
            int db )
```

Parameters

db	volume level in db (0 to -102). Warning: 0db volume is LOUD!

Returns

value to store in register 0x20 and 0x21

7.19 player.d File Reference

7.20 player.h File Reference

```
#include "miosix.h"
```

Classes

- class Sound
- class ADPCMSound
- class Player

7.21 README.md File Reference

Index

__attribute__
    button.cpp, 21
    player.cpp, 47, 48
∼Sound
    Sound, 15

ADPCM_Decode
    adpcm.c, 17
    adpcm.h, 19
ADPCM_Encode
    adpcm.c, 18
    adpcm.h, 19
ADPCMSound, 11
    ADPCMSound, 11
    fillMonoBuffer, 12
    fillStereoBuffer, 12
    rewind, 13
action
    button.cpp, 21
    game.cpp, 27
    main.cpp, 45
adpcm.c, 17
    ADPCM_Decode, 17
    ADPCM_Encode, 18
    IndexTable, 18
    StepSizeTable, 18
adpcm.d, 19
adpcm.h, 19
    ADPCM_Decode, 19
    ADPCM_Encode, 19

BLUE
    led.h, 39
blinkLed
    led.cpp, 36
    led.h, 41
blinkingClockwise
    led.cpp, 34
    led.h, 39
blinkingCounterClockwise
    led.cpp, 34
    led.h, 40
blinkingGame
    led.cpp, 35
    led.h, 40
blinkingsequence
    led.cpp, 35
    led.h, 40
blueLed
    led.cpp, 33
button
    button.cpp, 20
    game.cpp, 25
button.cpp, 20
    __attribute__, 21
    action, 21
    button, 20
    configureButtonInterrupt, 21
    mutex, 21
    waitForButton, 21
button.d, 22
button.h, 22
    configureButtonInterrupt, 22
    waitForButton, 22
Buzzer.h, 23
    buzzer_bin, 23
    buzzer_bin_len, 23
buzzer_bin
    Buzzer.h, 23
buzzer_bin_len
    Buzzer.h, 23
buzzerSound
    game.cpp, 25
    game.h, 29

calculateSleepTime
    led.cpp, 36
    led.h, 41
clockOrCounterClockWise
    led.cpp, 36
    led.h, 42
configureButtonInterrupt
    button.cpp, 21
    button.h, 22
convert.cpp, 24
    main, 24
    run, 24
cs43l22volume
    player.cpp, 48

difficulty
    game.cpp, 27
    led.cpp, 38
    main.cpp, 45

fillMonoBuffer
    ADPCMSound, 12
    Sound, 15
fillStereoBuffer
    ADPCMSound, 12

Sound, 15

GAMEOVER
game.h, 29
GREEN
led.h, 39
game
game.cpp, 28
led.cpp, 38
main.cpp, 45
game.cpp, 24
action, 27
button, 25
buzzerSound, 25
difficulty, 27
game, 28
gameOver, 26
gamePlay, 26
highscore, 28
highscoreSound, 27
interaction, 28
level, 28
mutex, 28
shouldBlinkAgain, 27
game.d, 28
game.h, 28
buzzerSound, 29
GAMEOVER, 29
gameOver, 29
gamePlay, 30
highscoreSound, 31
shouldBlinkAgain, 31
gameOver
game.cpp, 26
game.h, 29
gamePlay
game.cpp, 26
game.h, 30
greenLed
led.cpp, 33

highscore
game.cpp, 28
main.cpp, 45
highscore.h, 31
highscore_bin, 32
highscore_bin_len, 32
highscore_bin
highscore.h, 32
highscore_bin_len
highscore.h, 32
highscoreSound
game.cpp, 27
game.h, 31

i2c
player.cpp, 46
IndexTable
adpcm.c, 18
initLeds
led.cpp, 37
led.h, 42
instance
Player, 13
interaction
game.cpp, 28
main.cpp, 45
isPlaying
Player, 14

led.cpp, 32
blinkLed, 36
blinkingClockwise, 34
blinkingCounterClockwise, 34
blinkingGame, 35
blinkingsequence, 35
blueLed, 33
calculateSleepTime, 36
clockOrCounterClockWise, 36
difficulty, 38
game, 38
greenLed, 33
initLeds, 37
level, 38
onOffBlinking, 37
orangeLed, 34
redLed, 34
turnAllOff, 37
turnAllOn, 37
led.d, 38
led.h, 38
BLUE, 39
blinkLed, 41
blinkingClockwise, 39
blinkingCounterClockwise, 40
blinkingGame, 40
blinkingsequence, 40
calculateSleepTime, 41
clockOrCounterClockWise, 42
GREEN, 39
initLeds, 42
ORANGE, 39
onOffBlinking, 42
RED, 39
turnAllOff, 42
turnAllOn, 43
level
game.cpp, 28
led.cpp, 38
main.cpp, 45
lrck
player.cpp, 46

main
convert.cpp, 24
main.cpp, 44
main.cpp, 43
action, 45

difficulty, 45
game, 45
highscore, 45
interaction, 45
level, 45
main, 44
mainInitialisation, 44
mutex, 45
reInitialisation, 44
main.d, 46
mainInitialisation
main.cpp, 44
mclk
player.cpp, 47
mutex
button.cpp, 21
game.cpp, 28
main.cpp, 45

ORANGE
led.h, 39
onOffBlinking
led.cpp, 37
led.h, 42
orangeLed
led.cpp, 34

play
Player, 14
Player, 13
instance, 13
isPlaying, 14
play, 14
player.cpp, 46
__attribute__, 47, 48
cs43l22volume, 48
i2c, 46
lrck, 46
mclk, 47
reset, 47
scl, 47
sclk, 47
sda, 47
sdin, 47
player.d, 48
player.h, 48

README.md, 48
RED
led.h, 39
reInitialisation
main.cpp, 44
redLed
led.cpp, 34
reset
player.cpp, 47
rewind
ADPCMSound, 13
Sound, 16
run
convert.cpp, 24

scl
player.cpp, 47
sclk
player.cpp, 47
sda
player.cpp, 47
sdin
player.cpp, 47
shouldBlinkAgain
game.cpp, 27
game.h, 31
Sound, 14
∼Sound, 15
fillMonoBuffer, 15
fillStereoBuffer, 15
rewind, 16
StepSizeTable
adpcm.c, 18

turnAllOff
led.cpp, 37
led.h, 42
turnAllOn
led.cpp, 37
led.h, 43

waitForButton
button.cpp, 21
button.h, 22

Bibliography

[1] fedetft. Miosix embedded os. http://miosix.org/.

[2] William Fornaciari. Advanced operating systems (aos). http://home.deib.polimi.it/fornacia/doku.php?id=teaching_rtosMI.

[3] GitHub. Github - built for developers. https://github.com.

[4] ShareLaTeX. Sharelatex - latex, evolved - the easy to use, online, collaborative latex editor. https://www.sharelatex.com.

[5] Slack. Slack - where work happens. https://slack.com/.

[6] STMicroelectronics. Um1472 - user manual - discovery kit with stm32f407vg mcu. http://www.st.com/content/ccc/resource/technical/document/user_manual/70/fe/4a/3f/e7/e1/4f/7d/DM00039084.pdf/files/DM00039084.pdf/jcr:content/translations/en.DM00039084.pdf.

[7] Dimitri van Heesch. Doxygen - generate documentation from source code. http://www.stack.nl/~dimitri/doxygen/.