HapticLib

0.7

Generated by Doxygen 1.8.1.1

Thu Jan 31 2013 02:03:36

Contents

1 Main Page **1**

1.1 Introduction . . . 1

1.2 News . . . 1

1.3 For Users of HapticLib . . . 2

1.4 For Developers of HapticLib . . . 2

1.5 Changelog . . . 2

1.5.1 v0.7 . . . 2

1.5.2 v0.6 . . . 2

1.5.3 v0.5 . . . 3

1.5.4 v0.4 . . . 3

1.5.5 v0.3 . . . 4

1.5.6 v0.2 . . . 4

1.5.7 v0.1 . . . 4

1.6 Additional Info . . . 4

2 Architecture **7**

2.1 HapticLib Structure Overview . . . 7

2.1.1 User API Module: The high-level side . . . 8

2.1.2 Platform Specific Module: the low-level side . . . 9

2.1.2.1 STM32VL-DISCOVERY . . . 9

2.1.3 Pattern Generators Block: the haptic feedbacks . . . 11

2.2 Typical Use Scenarios . . . 12

2.2.1 Example 1: HapticWorld . . . 12

2.2.2 Example 2: HapticCalibrator . . . 14

2.2.3 Example 3: HapticLevel . . . 16

2.3 Application Debugging Feature . . . 19

3 User Guide **21**

3.1 Linux . . . 21

3.1.1 Preliminary Setup . . . 21

3.1.2 Tool-chains supported . . . 21

3.1.2.1 ARM . . . 22

3.1.2.2 MSP430 . . . 22
3.1.2.3 PIC . . . 22
3.1.3 GDB Servers supported . . . 22
3.1.4 Flasher utilities . . . 22
3.1.5 IDE supported . . . 22
3.1.5.1 No IDE . . . 22
3.1.5.2 Eclipse . . . 22
3.2 Windows . . . 23
3.2.1 Preliminary Setup . . . 23
3.2.2 Tool-chains supported . . . 23
3.2.2.1 ARM® . . . 24
3.2.2.2 Texas Instruments™ MSP430® . . . 24
3.2.2.3 Microchip™ PIC32® . . . 24
3.2.2.4 Atmel™ AVR32® . . . 24
3.2.3 Flasher utilities . . . 24
3.2.4 GDB Servers supported . . . 25
3.2.5 IDE supported . . . 25
3.2.5.1 No IDE . . . 25
3.2.5.2 Eclipse . . . 25
3.2.5.3 KEIL MDK uVision4 . . . 25

4 Developer Guide **27**
4.1 SystemDesc system descriptor. . . . 27
4.2 Haptor Descriptor . . . 27
4.3 Pattern Descriptor . . . 28
4.4 Pattern Rendering . . . 28
4.5 Develop a New Pattern Generator . . . 29
4.6 Adding a New Platform . . . 30
4.7 HapticLib vs. bare application comparison . . . 30

5 References **33**

6 Todo List **35**

7 Data Structure Index **37**
7.1 Data Structures . . . 37

8 File Index **39**
8.1 File List . . . 39

9 Data Structure Documentation **41**
9.1 constantStatusParameters Struct Reference . . . 41

9.1.1 Detailed Description . . . 41
9.1.2 Field Documentation . . . 41
9.1.2.1 duty . . . 41
9.2 constantUserParameters Struct Reference . . . 41
9.2.1 Detailed Description . . . 42
9.2.2 Field Documentation . . . 42
9.2.2.1 constant . . . 42
9.3 genericStatusParameters Struct Reference . . . 42
9.3.1 Detailed Description . . . 42
9.3.2 Field Documentation . . . 42
9.3.2.1 duty . . . 42
9.3.2.2 flag . . . 43
9.4 genericUserParameters Struct Reference . . . 43
9.4.1 Detailed Description . . . 43
9.4.2 Field Documentation . . . 43
9.4.2.1 checkParam . . . 43
9.4.2.2 increment . . . 43
9.5 haptor_desc Struct Reference . . . 43
9.5.1 Detailed Description . . . 44
9.5.2 Field Documentation . . . 44
9.5.2.1 activePattern . . . 44
9.5.2.2 id . . . 44
9.5.2.3 max_duty . . . 44
9.5.2.4 min_duty . . . 44
9.5.2.5 nextHaptor . . . 45
9.6 impactStatusParameters Struct Reference . . . 45
9.6.1 Detailed Description . . . 45
9.6.2 Field Documentation . . . 45
9.6.2.1 progress . . . 45
9.7 impactUserParameters Struct Reference . . . 45
9.7.1 Detailed Description . . . 46
9.7.2 Field Documentation . . . 46
9.7.2.1 material . . . 46
9.7.2.2 velocity . . . 46
9.8 pattern_desc Struct Reference . . . 46
9.8.1 Detailed Description . . . 46
9.8.2 Field Documentation . . . 47
9.8.2.1 activeHaptorList . . . 47
9.8.2.2 continuator . . . 47
9.8.2.3 name . . . 47

9.8.2.4 statusParams . . . 47
9.8.2.5 userParams . . . 47
9.9 status_param Union Reference . . . 47
9.9.1 Detailed Description . . . 48
9.9.2 Field Documentation . . . 48
9.9.2.1 constant . . . 48
9.9.2.2 generic . . . 48
9.9.2.3 impact . . . 48
9.9.2.4 test . . . 48
9.10 testStatusParameters Struct Reference . . . 48
9.10.1 Detailed Description . . . 49
9.10.2 Field Documentation . . . 49
9.10.2.1 duty . . . 49
9.10.2.2 flag . . . 49
9.11 user_param Union Reference . . . 49
9.11.1 Detailed Description . . . 49
9.11.2 Field Documentation . . . 50
9.11.2.1 constant . . . 50
9.11.2.2 generic . . . 50
9.11.2.3 impact . . . 50

10 File Documentation **51**

10.1 HapticLib/hapticLib.c File Reference . . . 51
10.1.1 Detailed Description . . . 51
10.1.2 Macro Definition Documentation . . . 52
10.1.2.1 HL_SYSTEM_FILE . . . 52
10.1.3 Function Documentation . . . 52
10.1.3.1 hl_addHaptor . . . 52
10.1.3.2 hl_configure . . . 53
10.1.3.3 hl_initPattern . . . 53
10.1.3.4 hl_startPattern . . . 54
10.1.3.5 hl_stopPattern . . . 55
10.1.4 Variable Documentation . . . 55
10.1.4.1 patternMap . . . 55
10.1.4.2 SystemDesc . . . 55
10.2 HapticLib/hapticLib.h File Reference . . . 56
10.2.1 Detailed Description . . . 57
10.2.2 Macro Definition Documentation . . . 57
10.2.2.1 HL_DEBUG . . . 57
10.2.2.2 STM32VLDISCOVERY . . . 58

10.2.3 Typedef Documentation . . . 58
10.2.3.1 haptor_desc . . . 58
10.2.4 Function Documentation . . . 58
10.2.4.1 hl_addHaptor . . . 58
10.2.4.2 hl_configure . . . 59
10.2.4.3 hl_initPattern . . . 60
10.2.4.4 hl_startPattern . . . 60
10.2.4.5 hl_stopPattern . . . 61
10.3 HapticLib/hl_debug.c File Reference . . . 61
10.3.1 Detailed Description . . . 62
10.3.2 Macro Definition Documentation . . . 62
10.3.2.1 HL_DEBUG . . . 62
10.3.3 Function Documentation . . . 63
10.3.3.1 send_int . . . 63
10.3.3.2 send_string . . . 64
10.4 HapticLib/hl_debug.h File Reference . . . 64
10.4.1 Detailed Description . . . 64
10.4.2 Macro Definition Documentation . . . 65
10.4.2.1 HL_DEBUG . . . 65
10.4.3 Function Documentation . . . 66
10.4.3.1 send_char . . . 66
10.4.3.2 send_int . . . 66
10.4.3.3 send_string . . . 66
10.5 HapticLib/patterns/constant/constant.c File Reference . . . 67
10.5.1 Detailed Description . . . 67
10.5.2 Function Documentation . . . 68
10.5.2.1 constantContinuator . . . 68
10.5.2.2 constantPatternGenerator . . . 69
10.6 HapticLib/patterns/constant/constant.h File Reference . . . 70
10.6.1 Detailed Description . . . 70
10.7 HapticLib/patterns/generic/generic.c File Reference . . . 70
10.7.1 Detailed Description . . . 71
10.7.2 Macro Definition Documentation . . . 72
10.7.2.1 HL_DEBUG . . . 72
10.7.3 Function Documentation . . . 72
10.7.3.1 genericContinuator . . . 72
10.7.3.2 genericPatternGenerator . . . 73
10.8 HapticLib/patterns/generic/generic.h File Reference . . . 73
10.8.1 Detailed Description . . . 74
10.8.2 Typedef Documentation . . . 75

10.8.2.1 genericCheckParam . . . 75
10.8.2.2 genericIncrement . . . 75
10.8.2.3 genericStatusParameters . . . 76
10.8.2.4 genericUserParameters . . . 76
10.8.3 Enumeration Type Documentation . . . 76
10.8.3.1 genericCheckParam . . . 76
10.8.3.2 genericIncrement . . . 76
10.9 HapticLib/patterns/hl_patterns.c File Reference . . . 77
10.9.1 Detailed Description . . . 77
10.9.2 Macro Definition Documentation . . . 78
10.9.2.1 HL_SYSTEM_FILE . . . 78
10.9.3 Function Documentation . . . 78
10.9.3.1 cleanList . . . 78
10.9.3.2 constantPatternGenerator . . . 78
10.9.3.3 dutyConverter . . . 79
10.9.3.4 genericPatternGenerator . . . 79
10.9.3.5 impactPatternGenerator . . . 80
10.9.3.6 patternScheduler . . . 80
10.9.3.7 testPatternGenerator . . . 81
10.9.4 Variable Documentation . . . 81
10.9.4.1 patternMap . . . 81
10.9.4.2 SystemDesc . . . 82
10.10 HapticLib/patterns/hl_patterns.h File Reference . . . 82
10.10.1 Detailed Description . . . 83
10.10.2 Macro Definition Documentation . . . 84
10.10.2.1 MAX_PATTERNS . . . 84
10.10.3 Typedef Documentation . . . 84
10.10.3.1 pattern_continuator . . . 84
10.10.3.2 pattern_desc . . . 84
10.10.3.3 pattern_initiator . . . 84
10.10.3.4 pattern_name . . . 85
10.10.3.5 status_param . . . 85
10.10.3.6 user_param . . . 85
10.10.4 Enumeration Type Documentation . . . 85
10.10.4.1 pattern_name . . . 85
10.10.5 Function Documentation . . . 86
10.10.5.1 cleanList . . . 86
10.10.5.2 dutyConverter . . . 86
10.10.5.3 patternScheduler . . . 86
10.11 HapticLib/patterns/impact/extra/impact.m File Reference . . . 87

10.11.1 Function Documentation . . . 90
10.11.1.1 Copyright . . . 90
10.11.2 Variable Documentation . . . 90
10.11.2.1 ACTION . . . 90
10.11.2.2 CONTRACT . . . 90
10.11.2.3 copy . . . 90
10.11.2.4 DIRECT . . . 91
10.11.2.5 granted . . . 91
10.11.2.6 INDIRECT . . . 91
10.11.2.7 modify . . . 91
10.11.2.8 PROFITS . . . 91
10.11.2.9 SPECIAL . . . 91
10.11.2.10use . . . 91
10.11.2.11USE . . . 92
10.12HapticLib/patterns/impact/impact.c File Reference . . . 92
10.12.1 Detailed Description . . . 92
10.12.2 Optional Parameters . . . 94
10.12.3 Function Documentation . . . 95
10.12.3.1 impactContinuator . . . 95
10.12.3.2 impactPatternGenerator . . . 96
10.12.4 Variable Documentation . . . 96
10.12.4.1 aluminumImpactPattern . . . 96
10.12.4.2 rubberImpactPattern . . . 97
10.12.4.3 woodImpactPattern . . . 97
10.13HapticLib/patterns/impact/impact.h File Reference . . . 98
10.13.1 Detailed Description . . . 99
10.13.2 Typedef Documentation . . . 99
10.13.2.1 ImpactMaterial . . . 99
10.13.2.2 ImpactVelocity . . . 99
10.13.3 Enumeration Type Documentation . . . 99
10.13.3.1 ImpactMaterial . . . 99
10.13.3.2 ImpactVelocity . . . 99
10.14HapticLib/patterns/test/test.c File Reference . . . 100
10.14.1 Detailed Description . . . 100
10.14.2 Function Documentation . . . 101
10.14.2.1 testContinuator . . . 101
10.14.2.2 testPatternGenerator . . . 101
10.15HapticLib/patterns/test/test.h File Reference . . . 102
10.15.1 Detailed Description . . . 102
10.16HapticLib/platforms/STM32VLDISCOVERY/hl_STM32VLDISCOVERY.c File Reference . . . 103

10.16.1 Detailed Description . . . 103
10.16.2 Macro Definition Documentation . . . 104
10.16.2.1 HL_SYSTEM_FILE . . . 104
10.16.3 Function Documentation . . . 104
10.16.3.1 Delay . . . 104
10.16.3.2 GPIO_Configuration . . . 104
10.16.3.3 RCC_Configuration . . . 105
10.16.3.4 SysTick_Handler . . . 106
10.16.3.5 TIM_Channel_DutyChanger . . . 106
10.16.3.6 TIM_Channel_Enable . . . 107
10.16.3.7 TIM_Configuration . . . 107
10.16.4 Variable Documentation . . . 108
10.16.4.1 channelStatus . . . 108
10.16.4.2 SystemDesc . . . 108
10.16.4.3 TimingDelay . . . 109
10.17 HapticLib/platforms/STM32VLDISCOVERY/hl_STM32VLDISCOVERY.h File Reference . . . 109
10.17.1 Detailed Description . . . 110
10.17.2 Macro Definition Documentation . . . 110
10.17.2.1 MAX_HAPTORS . . . 110
10.17.2.2 STM32F10X_MD_VL . . . 110
10.17.3 Function Documentation . . . 111
10.17.3.1 Delay . . . 111
10.17.3.2 GPIO_Configuration . . . 111
10.17.3.3 RCC_Configuration . . . 112
10.17.3.4 SysTick_Handler . . . 113
10.17.3.5 TIM_Channel_DutyChanger . . . 113
10.17.3.6 TIM_Channel_Enable . . . 113
10.17.3.7 TIM_Configuration . . . 114

Chapter 1

Main Page

1.1 Introduction

Welcome to the HapticLib documentation repository.

HapticLib is a haptic library for embedded systems based on microcontrollers like ARM, MSP430, AVR, PIC.

Haptic feedback is obtained driving multiple vibro tactile devices connected to the embedded system.

All the documentation about HapticLib can be found here; some of it is auto generated from source code, so it will always reflect the latest version of the code.

Download the latest version of HapticLib.

Download the PDF version of the documentation.

Download the KEIL uVision4 workspace package updated to the last release version.

1.2 News

The main features added to the last version of HapticLib are:

- **Multi-Haptor** capability! Now HapticLib can manage multiple haptors simultaneously!!!
- Formalized a new Template for the pattern generators.
- Big changes everywhere to reflect new multi-haptor architecture.
- Started **Platform Specific Module** documentation section under Architecture page.

See the Changelog for a detailed list of changes from last version.

Architecture Overview

How HapticLib is structured and how all the pieces work together.

The library is developed with modularity in mind. The functional domains are decoupled and kept separate to allow for easy extensibility.

The main modules are:

- **User API**
- **Platform Specific**

- **Pattern Generators**

To know more:

- Go to Architecture.

1.3 For Users of HapticLib

What to setup to develop an application using HapticLib in different environments.

Specific instructions are provided for **Linux** and **Windows** OS.

Setting up the environment to work with HapticLib can be hard, that's why a detailed guide has been created to help the user get up and running through the components needed and start coding his application as soon as possible.

- Go to User Guide.
- Go directly to Linux Instructions.
- Go directly to Windows Instructions.

1.4 For Developers of HapticLib

The internal details of HapticLib.

This page is useful for developers who want to modify or extend the HapticLib library.

Detailed description on some techniques used inside HapticLib is given.

For example, the **Pattern Generator Template** is a useful tool to use to easily integrate new patterns inside the library.

HapticLib also offer **Debugging Features**; it is a set of tools very handy, but attention must be paid to avoid problems.

For details,

- Go to Developer Guide.

1.5 Changelog

List of changes between versions of HapticLib.

1.5.1 v0.7

- Added support to KEIL uVision4 IDE. Now development can be done using this IDE (and its toolchain)
- Minor fix to library code to remove errors when compiling with armcc.
- Fixed problem on HapticLevel Demo: changed SPI interface to I2C.

1.5.2 v0.6

- Added HapticLevel Demo (need to be tested with actual interface)
- Completed documentation on Architecture page.
- Completed documentation on Developer Guide page.

1.5.3 v0.5

- Implemented new Multi-Haptor architecture.
- STM32VLDISCOVERY now supports MAX_HAPTORS=4, and will raise in the next releases.
- Reorganized almost data structures to reflect new architecture.
- Formalized a new pattern generator Template (initiator/continuator).
- Added data structures to increase code readability and compiler check enforcing (to work around void* argument passing).
- Fixed send_int() to skip leading zeros from printing.
- Fixed send_int() bug preventing 0x0 to be printed at all.
- Moved the Pattern Scheduler in the **User API** module.
- Updated all the demos to make use of the new Multi-Haptor capability.
- Cleaned STM32VLDISCOVERY Platform code. Now The Timers are used in a cleaner way.
- Moved the haptors GPIO to stop interfering with the User LED and Debug LED.
- Fixed a bug in the Makefile that prevented Release version to fully work.
- Fixed a bug in the Makefile that skipped updated .bin target from being built.
- Started **Platform Specific Module** documentation section under Architecture page.
- Restructured directory layout for patterns: now each pattern has its own sub-directory.
- Restructured directory layout for platform: now each platform has its own sub-directory.
- Updated demos Makefile to the new directory structure.
- added utility function to convert duty cycles honoring min_duty and max_duty.
- Almost completed Architecture documentation page.

1.5.4 v0.4

- Created CVS Repository to track code versioning and team development.
- Moved SysTick configuration from hapticlib.c to platform specific code.
- Moved debugging support code from hapticlib.c to hl_Debug.{h,c}.
- Single-Haptor working implementation of the haptic "Impact Pattern".
- Detailed documentation on "Impact Pattern" with theoretical references.
- "Impact Pattern" samples results from Matlab script provided.
- Standardized "Pattern Generators" implementation and documentation.
- Created a Pattern Generator Template to use as starting point to create new patterns.
- Cleaned the pattern header file inclusion system to flawlessly export pattern specific symbols to user program.
- Added preliminary implementation for multi-haptor support. Now the API has changed, but the Platform Specific code still doesn't use multi haptors.
- Cleaned Debugging features inclusion system.
- Updated and expanded code documentation to reflect all code changes.

- New structure for Library documentation.
- Expanded Library documentation.
- Increased granularity for PWM duty-cycle. Now there are 65535 values.
- bug fixes.

1.5.5 v0.3

- Makefile now eclipse friendly.
- Makefile now works fine with Windows (needs basic unix commands [cygwin]).
- Other Makefile bug fixes.
- Elegant modularization of pattern generation functions using function pointers.
- Created directory structure to insert new patterns.
- Code documentation now is in sync with code.

1.5.6 v0.2

- Cleaned the Makefile structure.
- Added Demos/Project_Template skeleton for new applications.
- Moved SysTick configuration inside hapticLib.c.
- Fixed bug on conditional code pre-processing with HL_DEBUG symbol.

1.5.7 v0.1

- First version of the library.

1.6 Additional Info

Copyright Notice and License Terms and Conditions

Version

0.7

Authors

Leonardo Guardati
Silvio Vallorani

Date

2012

Chapter 2

Architecture

2.1 HapticLib Structure Overview

HapticLib structure is composed of different parts working together to expose a simple API to the library user.

The main blocks are:

- **User API** module.

- **Platform Specific** module.

- **Pattern Generator** module.

Concept of Operations:

The user program code uses documented API from the *User API* module (basically `#include "hapticLib.h"` lists the API function offered). The *User API* module then forward the application requests to the other modules. For the initialization phase, and for any hardware interaction, the *Platform Specific* module is always called ultimately. For anything related to the Haptic patterns generation, the *Pattern Generator* module is the one doing the hard work. Here the *User API* module act just as proxy for the application code to call the right pattern generator function, allowing for transparent extension of the Library without breaking existing code.

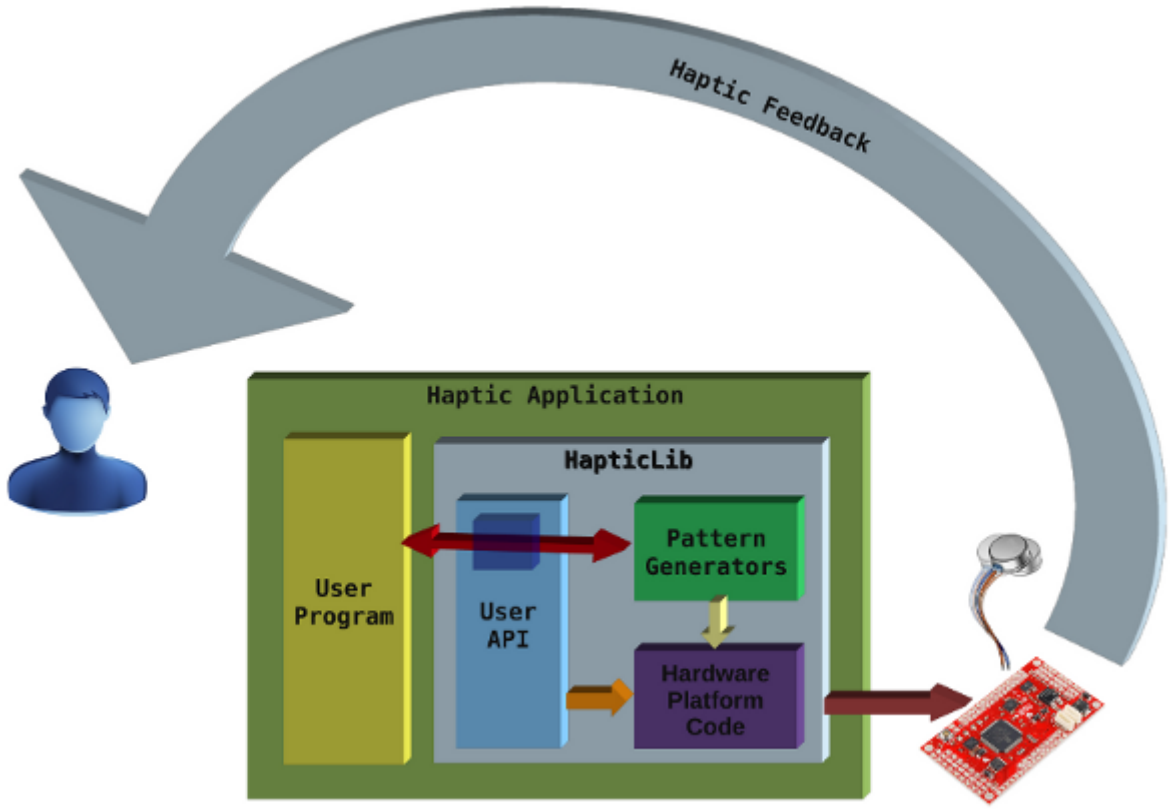

Figure 2.1: HapticLib Architecture

2.1.1 User API Module: The high-level side

The **User API** module is the direct interface to the HapticLib user's application code. The developer should start from this module to learn how to use HapticLib.

Here is a simple use case.

The program starts initializing the system.

```
uint32_t frequency    = 12000000;  // PWM frequency set to 12MHz.
uint8_t  sampleDelay  = 10;            // PWM inter-samples delay 10ms.
uint8_t  numHaptors   = 3;         // Number of haptors used

haptor_desc *myHaptors;                   // haptor descriptors array.

myHaptors = hl_configure(frequency, sampleDelay, numHaptors);
```

Note

The number of haptors available depend on the hardware platform used and on the *Platform Specific* module support for that platform.

Then the program must initialize the patterns used in the application:

```
pattern_desc *myPattern;

myPattern = hl_initPattern(Test, NULL);

hl_addHaptor(&myHaptors[1],myPattern);
```

Now the pattern can be started:

```
hl_startPattern(myPattern);
```

And stopped (if the pattern has not finished yet):

```
        Delay(2000);

hl_stopPattern(myPattern);
```

Note

Make sure to read the documentation of the pattern to know what data type is expected, and what are the correct value ranges for every parameter.

That's it!

From an Interface point of view, there is nothing more to do for the system to work. It is up to the developer now to implement the right behavior of the application.

HapticLib offers a lot more than this simple and plain example. You can simultaneously start different patterns on different haptors. The patterns will run concurrently together.

Even more, multi-haptors patterns can be setup to drive different haptors under the same haptic logic (just call `hl_addHaptor()` multiple times on the same pattern).

HapticLib uses a pattern scheduler to implement this features. If you want to know more, pleas refer to the Developer Guide page.

Please, make sure to also read all the other documentation; in particular check:

- The hardware details of the platform used.
- The details of the pattern generated; the theory behind the haptic feedback and the meaning of the parameters required.

See also

Please refer to the Pattern Generators module to know what patterns are available and learn all the details on them, like the name to use, and any optional parameter required to work.

2.1.2 Platform Specific Module: the low-level side

The interaction with the hardware is done in this module. Actually this module is composed of a pair of .c/.h files for each hardware platform supported by the library.

During the initial configuration phase, the high-level hl_configure() function calls the right implementation of the low level routines based on the platform specific symbols defined. (for example STM32VLDISCOVERY).

If a platform provide a peripheral driver library, this module can make use of it greatly simplifying the code. For example the STM32VLDISCOVERY board is based on a STM32F10x MCU that is supported by STM's StdPeriph Library.

Every platform have its characteristics, so it is important to read about HapticLib implementation for that platform that impact on the high level behavior of the application.

Here is the list of supported platforms:

2.1.2.1 STM32VL-DISCOVERY

This board from STM uses a STM32F100RB ARM based MCU. The library development started on this board, so it is well supported.

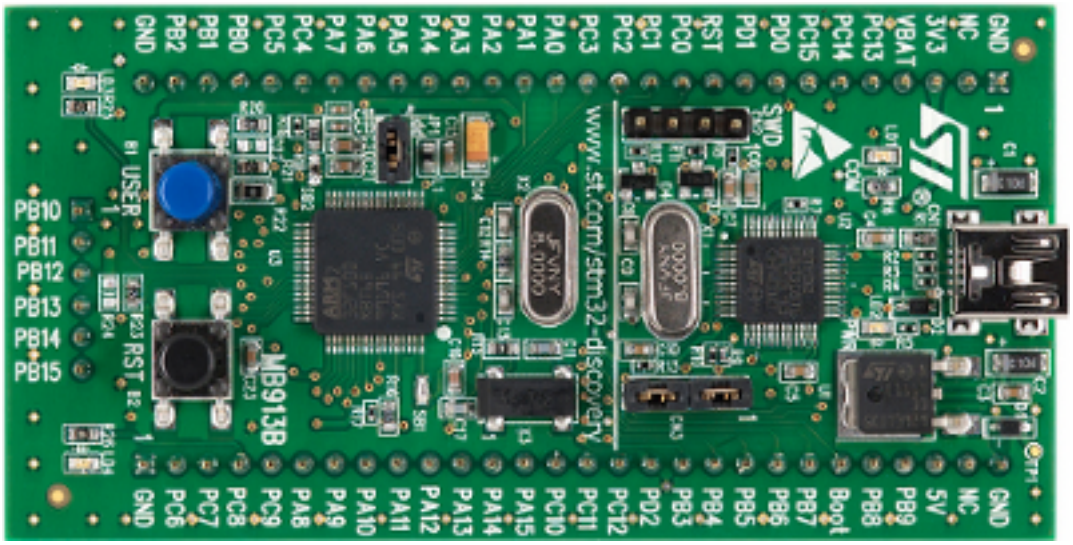

Figure 2.2: STM32VLDISCOVERY

Here the interesting specifications of the MCU from a HapticLib perspective:

- up to 24MHz Core and Peripheral bus speed.
- up to 7 TIMers (resulting in more than 20 PWM channels directly mappable to GPIOs).
- additional System Timer (SysTick) to measure time delays.

For `detailed informations` refer to STM documentation.

The STM32VL-DISCOVERY Developer Board features:

- a lot of IO expansion capabilities, useful for multi-haptor scenarios.
- a built-in user button for basic interaction.
- two built-in LEDs for basic application feedback and debug.

Also from STM, the `StdPeriph` is used to simplify hardware addressing.

Note

HapticLib doesn't distribute the STM's StdPeriph Standard Peripheral. But it is possible to `download` the package directly from ST. Once downloaded just uncompress the content inside the HapticLib root directory and all the features will be available.

Haptic Feedback layout

The STM32VL-DISCOVERY Dev Board, can generate more than 20 PWM control signals directly mappable to GPIO via Alternate Functions. However the HapticLib's STM32VLDISCOVERY Platform Specific module only supports 4 haptic devices to be driven by the application. Future release may add support for additional haptic devices for this platform.

Here is the haptic device hardware layout supported:

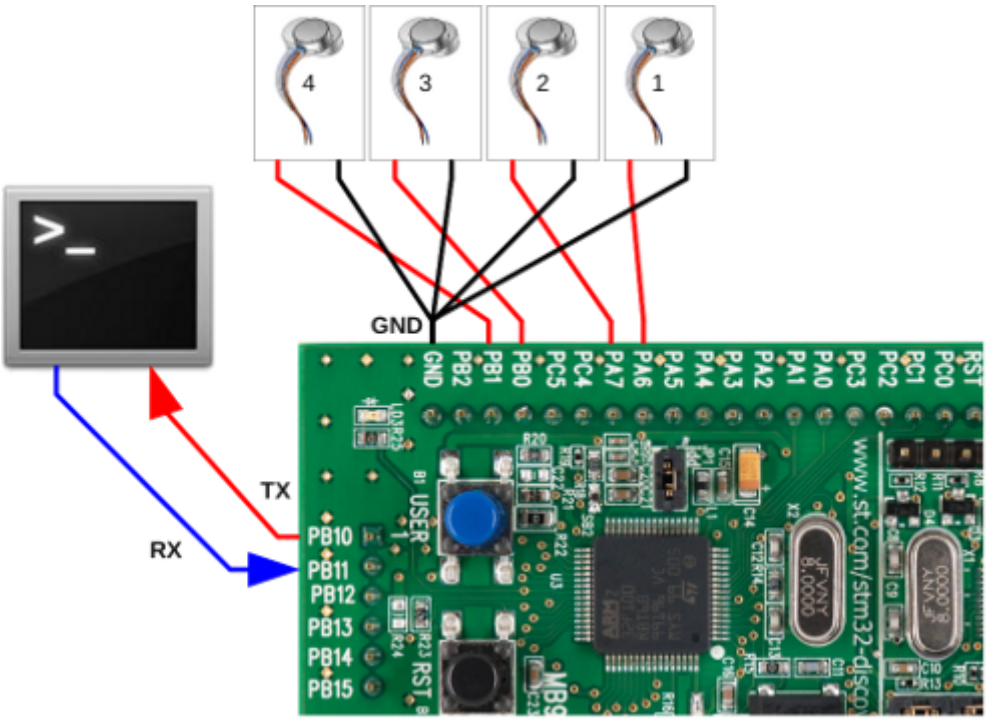

Figure 2.3: STM32VLDISCOVERY Devices

The Serial link available on the PB I/O port, is enabled only when the HL_DEBUG symbol is defined at compile time.

2.1.3 Pattern Generators Block: the haptic feedbacks

To understand how the **Pattern Generators** module works, let's start describing the sequence of events happening when the user starts a pattern.

When the user application code calls hl_startPattern() (after the initialization step), the **User API** module forward the call to the initiator of the pattern passed as parameter.

Once called, the pattern initiator validate the user parameters. If the parameters are valid, the initiator will setup the its initial status, and the pattern instance becomes running.

When the pattern instance becomes running, the **pattern scheduler** will start calling its continuator every time the inter sample delay elapses.

When the pattern ends (if it ever ends) the continuator will remove the pattern instance from the scheduler freeing the resources used (the haptors).

Alternatively, if the pattern is still running and the user needs it to stop (or needs the haptors for other patterns) the hl_stopPattern() API function can be called to force the pattern de-scheduling and resources freeing.

Now that the running flow of events has been described, we can show the Pattern Generator typical structure.

A Pattern Generator is a sort of module plugged in HapticLib. It is composed of two functions:

- the pattern **initiator**
- the pattern **continuator**

and two data structure type definitions:

- the pattern **user parameters**

- the pattern **status parameters**

The **initiator** for a pattern instance gets called only once; when the user application code calls hl_startPattern().

Its job is to setup the initial status of the pattern and schedule the instance for running to the patternScheduler().

The **continuator** for a pattern instance is called by the patternScheduler() every inter-sample delay (after the initiator scheduled it) as long as the pattern instance is *running*. The haptic feedback logic of the pattern decides when to stop the instance. During its last run, the continuator will de-schedule the pattern instance from the scheduler, and free all the resources used (the haptors).

Note

It is important to understand how a pattern works in order to know if the instance will stop by itself or if the user have to call hl_stopPattern() to put an end to it.

A Pattern Generator also defines two data structures to hold the informations needed at runtime.

The **user parameters** is a type definition based on a structure holding all the data the user can fine-tune when creating a new *pattern instance*.

It is important to read the documentation of the pattern to be used, to understand the meaning and the use the pattern will make of every single parameter passed by th user.

To better understand, here is an example of user parameters for an existing pattern, the **impact pattern** generator.

The user parameters accepted to personalize a pattern instance are:

- the impact **material**
- the impact **velocity**

One instance could send the haptic feedback of a **fast** impact on a **wood** surface, then another instance could send the feedback of a **slow** impact on a **rubber** surface.

Refer to the impact pattern generator documentation for detailed informations.

The **status parameters** structure is an internal container of the pattern instance status and progress. It is not exposed to the user, nevertheless it is important to understand that internally the pattern initiator will setup the starting status (probably depending on the provided *user parameters*) and that the continuator will update the status parameters at every patternSchedule() call based on the logic and the progress of the pattern.

Note

In a typical application, the *user parameters* are instantiated in the user application code (for example in the main.c module), while the *status parameters* are hold inside an internal indexing data structure.
Typically the **user parameters** variable is set up before initializing the pattern because the initiator uses it to set the initial values for the *status parameters*.
HapticLib architecture allow also an additional use case for the *user parameters*, that is a way to modulate the pattern *continuator* behavior during the pattern execution. The user has full control over the *user parameter* variable, and the *continuator* can poll the actual value of any user parameter during its executions.

2.2 Typical Use Scenarios

The following examples will describe some uses of the library showing the way to use the *User API* and the different kind of pattern existent.

Under the `Demos/` HapticLib sub-directory are present different demo applications.

2.2.1 Example 1: HapticWorld

HapticWorld is the simplest demo possible, showing how to use HapticLib API.

This demo activate a single haptor test pattern and then exits.

The main.c code is listed below, interleaved with the descriptions.

```
#include "hapticLib.h"
```

This include will enable HapticLib in the application.

```
enum demoHaptors {
     Head          = 0,
     Lhand         = 1,
     Rhand         = 2,
     Back          = 3,

     NumDemoHaptor = 4
};
```

This enumeration will make the code cleaner when referring to the haptor descriptors array.

```
int main(void)
{

     // Pointer to the array containing the setup haptors.
     haptor_desc *myHaptors;

     // Initialize the Haptic System and get the haptors array.
     myHaptors = hl_configure(24000000, 10, NumDemoHaptor);
```

Here the reference to the set of haptors used by the application is declared and initialized. `NumDemoHaptor` can be less than the total available system haptors. From now on every haptor can be referenced in this way:

`myHaptor[Head]` for the first haptor `myHaptor[Lhand]` for the second, etc.

```
     // We are ready to send patterns.

     pattern_desc *myPattern;

     // Use a Test Pattern
     myPattern = hl_initPattern(Test, NULL);
```

`myPattern` is the descriptor of the test pattern we want to use. In this case the second argument of hl_initPattern() is `NULL` because the Test Pattern does not accept any user parameter.

```
     // We need to tell the Test pattern which haptor to use
     hl_addHaptor( &myHaptors[Head], myPattern);
```

The pattern is now setup, but it doesn't know yet on which haptor to work. hl_addHaptor() link the given haptor to the given pattern. In this case `myHaptor[Head]` is used.

```
     // Start Shaking
     hl_startPattern(myPattern);
```

The hl_startPattern() call activates the feedback execution.

In this particular Pattern and with the settings given to hl_configure()the pattern activation will last for some time (about 2 seconds).

```
     // Let it shake for a while...
     Delay(1000);

     // ...or force its stop if we need the haptor for something else.
     hl_stopPattern(myPattern);
```

To show the forced stopping of the pattern, hl_stopPattern() is called after 1 second wait (in the middle of pattern execution).

```
#ifdef HL_DEBUG
            send_string("\r\nHapticWorld: Going to sleep...goodbye!\r\n\0");
#endif

     // go to sleep
     while(1) { __WFE(); }
}
```

In the end the program goes to sleep.

2.2.2 Example 2: HapticCalibrator

HapticCalbirator is a demo application that let the user calibrate the minimum PWM duty cycle for an haptor that delivers a vibration strong enough to be felt.

The HapticCalibrator demo will show how a continuous pattern works, how is possible to modify a user parameter during a pattern execution.

The main.c code is listed below, interleaved with the descriptions.

```
#include "stm32f10x.h"
#include "hapticLib.h"
```

In addition to HapticLib include, this application will use some hardware peripheral directly, that's the reason for the STM include.

```
int main(void)
{

     haptor_desc *myHaptor;
     user_param   myParams;

     myHaptor = hl_configure(24000000,10,1);
```

Here the reference to the haptor used and the pattern's user parameters are defined.

Then the haptor reference is initialized with hl_configure().

```
     // User Button Event Configuration
     GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource0);
     GPIO_EventOutputConfig(GPIO_PortSourceGPIOA, GPIO_PinSource0);

     EXTI_InitTypeDef InitStruct;
     InitStruct.EXTI_Line    = EXTI_Line0;
     InitStruct.EXTI_Mode    = EXTI_Mode_Interrupt;
     InitStruct.EXTI_Trigger = EXTI_Trigger_Rising;
     InitStruct.EXTI_LineCmd = ENABLE;

     EXTI_Init(&InitStruct);

#ifdef HL_DEBUG
     send_string("\r\n\r\n\0");
     send_string("\t\tWelcome to HapticCalibrator!!\r\n\r\n\0");
     send_string("Now we will calibrate the System's Haptors...\r\n\0");
     send_string("As soon as you CLEARLY feel the vibration, hit the User Button!!!\0");
     send_string(" ( Be quick! ;) )\r\n\r\n\r\n\0");
#endif

     Delay(1500);

#ifdef HL_DEBUG
     send_string("Wear the Haptor and when you feel ready, press the User Button.\r\n\0");
#endif
```

```
    while ( EXTI_GetFlagStatus(EXTI_Line0) != SET ) { __WFI(); } ;

    EXTI_ClearFlag(EXTI_Line0);
```

This piece of code is specific to this application. The EXTernal Interrupt is configured with the User Button (PA0), and then the program waits (sleeping) for the user to press the User Button and let the calibration begin.

```
    myParams.constant.constant = 0x0000;
    hl_startPattern( hl_addHaptor(myHaptor, hl_initPattern(Constant, &myParams) ) );
```

The pattern used in this application is the **Constant Pattern**. It takes a user parameter `constant` that represent the duty cycle of this haptor's PWM.

The starting of the pattern here is condensed using call nesting of functions.

To call hl_startPattern(), hl_addHaptor() must be called first, and likewise hl_initPattern() is required by hl_addHaptor().

So the actual sequence is:

- hl_initPattern()
- hl_addHaptor()
- hl_startPattern()

The initial duty cycle is 0x0000.

```
    while ( myParams.constant.constant < 0xff00 )
    {
#ifdef HL_DEBUG
            send_string("Duty Cycle: \0");
            send_int(myParams.constant.constant);
            send_string("\r\n\0");
#endif
            myParams.constant.constant += 10;

            Delay(1);

            if( EXTI_GetFlagStatus(EXTI_Line0) == SET )
            {
                    EXTI_ClearFlag(EXTI_Line0);
#ifdef HL_DEBUG
                    send_string("GOOD! We found the minimum dutyCycle!\r\n\0");
#endif
                    GPIO_WriteBit(GPIOC, GPIO_Pin_8, Bit_SET);
                    break;
            }
    }
```

Now a cycle is started to increase the duty cycle at every iteration. And when the user presses the User Button, the cycle is interrupted.

```
    if( myParams.constant.constant >= 0xff00 )
    {
#ifdef HL_DEBUG
            send_string("BAD!! We reached almost the 100\% DutyCycle and you didn't press.\r\n\0");
            send_string("Make sure your Haptor isn't broken!\r\n\0");
#endif
    }
    else
    {
#ifdef HL_DEBUG
            send_string("Now setting the minimum activating DutyCycle to: \0");
            send_int(myParams.constant.constant);
            send_string("\r\n\0");
```

```
#endif

            myHaptor->min_duty = myParams.constant.constant;
    }
```

Here a check is made to verify the user really pressed the button and the duty didn't reach the top instead.

If the user pressed the button, the haptor's minimum duty cycle is updated with the value set by the user.

```
    hl_stopPattern(myHaptor->activePattern);
```

The **Constant Pattern** is a non stop pattern, this means the user application must call hl_stopPattern() to end the pattern and free the haptor.

```
#ifdef HL_DEBUG
    send_string("\r\nGoodbye....going to sleep...\r\n\0");
#endif

    while(1) { __WFE(); }
}
```

In the end the program goes to sleep.

2.2.3 Example 3: HapticLevel

HapticLevel is a very simple, real world example demo showing how easy is to implement haptic feedback embedded applications. HapticLevel acts as an electronic Level to measure a surface's tilt. The device uses 4 haptors (one for each direction) and one 3-axis accelerometer (STM's LIS302DL). The haptors are connected to the board's GPIO, the sensor is connected to the I2C1 peripheral, and the serial console (enabled only in debug versions) is connected to USART3 as usual.

Here is the complete system layout:

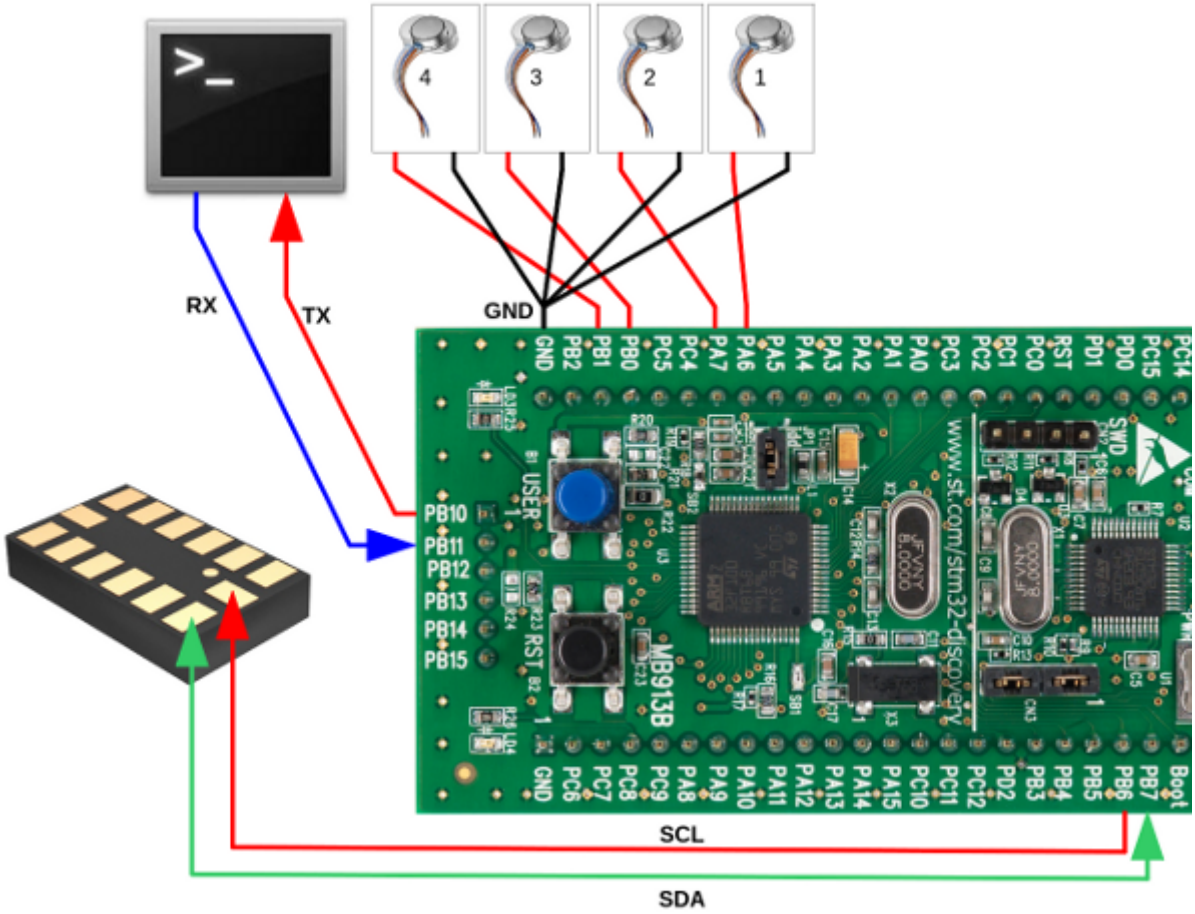

Figure 2.4: HapticLevel Layout

The haptors are placed in pairs along orthogonal axes. When the device is placed on a surface, it will "display" the tilt activating one haptor for each direction.

Now that the device has been described, let's look at HapticLevel's code.

HapticLib knows nothing about the LIS302DL sensor and all the relevant code is jut part of the demo.

To unclutter main.{c,h} source code files, another module has been used to place all sensor specific code.

LIS302DL support SPI and I2C interface. I2C has been used. Routines to manage communications with the sensor are based on existing code by Michele Magno and Bojan Milosevic (MicreLab University of Bologna) located on lis302dl.c, lis302dl.h HAL_LIS3LV02DL.h source code files.

In main.c the fist thing the program does is to init the I2C1 peripheral and then init the sensor as well:

```
// I2C1 peripheral Initialization
LIS3LV02DL_I2C_Init();

// LIS302DL Sensor Initialization
LIS3LV02DL_Acc_Init();
```

The next thing main() does is to initialize HapticLib library, configuring the four haptors, setting up the Constant pattern, and attaching the haptors to the pattern.

```
// Haptors Initialization
// Initialize the Haptic System (setting the PWM global period in Hz)
haptors = hl_configure(24000000,10, NumLevelHaptors);
// Start each haptor
for( i=0 ; i<NumLevelHaptors ; i++)
```

```
    {
      levelParams[i].constant.constant = 0;
        if(  hl_startPattern(
               hl_addHaptor(  &haptors[i],
                      hl_initPattern(Constant,&levelParams[i]) ) )  )
        {
    #ifdef HL_DEBUG
        send_string("HapticLevel: Problem Starting Haptor \0");
        send_int(i);
        send_string(".\r\n\0");
    #endif
        }
     }
```

The starting intensity is set to 0 for each pattern, so they will not vibrate. Now the actual application logic kicks in; the program enters an endless loop where the values of x,y,z acceleration are read from the sensor. The z-axis is used to sense the actual orientation of the device (upside or downside). Then the x and y axis are used to decide which haptor of each pair will vibrate with a new constant value proportional to the entity of directional tilt. The other haptor of the pair is simply kept to a constant value of 0 to prevent vibration.

```
    while(1) {

    // Get X-axis value
    LIS3LV02DL_Acc_Read_RawData(&tmp,LIS_A_OUT_X_L_ADDR);
    x = (int8_t) (tmp >> 8);

    // Get Y-axis value
    LIS3LV02DL_Acc_Read_RawData(&tmp,LIS_A_OUT_Y_L_ADDR);
    y = (int8_t) (tmp >> 8);

    // Get Z-axis value
    LIS3LV02DL_Acc_Read_RawData(&tmp,LIS_A_OUT_Z_L_ADDR);
    z = (int8_t) (tmp >> 8);

    #ifdef HL_DEBUG
        send_string("HapticLevel: X-Value \0");
        send_int(x);
        send_string(".\r\n\0");
    #endif

    // positive orientation
    if( z >= 0 ) {
            if( x >= 0 ) {
                levelParams[Up].constant.constant     = 4096*x;
                levelParams[Down].constant.constant   = 0;
            }
            else {
                levelParams[Down].constant.constant   = -4096*x;
                levelParams[Up].constant.constant     = 0;
            }
            if( y >= 0 ) {
                levelParams[Left].constant.constant   = 4096*y;
                levelParams[Right].constant.constant  = 0;
            }
            else {
                levelParams[Right].constant.constant  = -4096*y;
                levelParams[Left].constant.constant   = 0;
            }
    }
    else { // negative orientation
            if( x <= 0 ) {
                levelParams[Up].constant.constant     = -4096*x;
                levelParams[Down].constant.constant   = 0;
            }
            else {
                levelParams[Down].constant.constant   = 4096*x;
                levelParams[Up].constant.constant     = 0;
            }
            if( y <= 0 ) {
                levelParams[Left].constant.constant   = -4096*y;
                levelParams[Right].constant.constant  = 0;
```

```
            }
            else {
                levelParams[Right].constant.constant  = 4096*y;
                levelParams[Left].constant.constant   = 0;
            }
        }

        // Sleep
        Delay(100);

    }

    while(1) { __WFE(); }
```

The main() code enters an endless loop where it polls for new values of the 3 vectors (x,y,z) from the sensor.

The single direction value is retrieved calling LIS3LV02DL_Acc_Read_RawData() with the first argument being a pointer to int16_t value and the second argument the direction to be retrieved.

The values are then casted to 8 bit signed values.

The first check is on z-axis value to understand if the device is up-face or down-face.

Then the x and y-axis are checked to know the device tilt on each direction.

If the axis value is positive, a scaled value of the sensed amplitude is set as duty-cycle for one haptor. (and a zero duty is set for the opposite one). If the axis value is negative, the same happen but with inverted haptors.

Note the negative scaling coefficient for negative direction. This is needed in order to have coherent proportional intensity variation on tilt.

The last line of code should never be reached.

2.3 Application Debugging Feature

HapticLib offers some debugging features to ease application development and bug hunting.

The debugging features are made of two utility functions:

- send_int()
- send_string()

Implementing those functions, the user won't need any additional code library for basic I/O across a serial link.

All the debugging code inside the library gets conditionally compiled if the preprocessor symbol HL_DEBUG is defined by the user, allowing the complete removal of any debugging code on the release version of the application.

The user application code must adhere to this technique, enclosing any debug related code inside preprocessor rules:

```
#ifdef HL_DEBUG

    // Debugging code

#endif
```

The HL_DEBUG preprocessor symbol must be defined in order to enable the debugging features. Th best place to do this is adding the definition on the compiler invocation command. For example:

```
  gcc ... -DHL_DEBUG ...
```

The Makefile available from the template Demo application defines a variable to easily manage the compilation of DEBUG and/or RELEASE target version of the application.

send_int()

This utility function allow the transmission of a string representation of any unsigned long integer number (uint32_t).

The number is always printed in hexadecimal format (0x...) and additionally, if the value is less than 100000, the decimal representation is print too.

For example:

```
send_int(256);
```

will output the following string:

```
0x100 (256)
```

meanwhile:

```
send_int(456789);
```

will output:

```
0x6f855
```

send_string()

This utility function allow the transmission of a string across the serial link.

The string will be trimmed to 255 characters.

If the string passed has a null terminator ('\0'), only the characters preceding it will be printed.

For example:

```
send_string("Hello World!\n\rGoodbye World!\0Not this");
```

will output:

```
Hello World!
Goodbye World!
```

Platform specific implementations

Both send_int() and send_string() functions rely on the function send_char() to actually output every single character.

send_char() is a platform specific function.

Each platform must implement the send_char() function to enable the debugging features.

For example, the STM32VLDICOVERY platform module uses the USART3 as serial link for debugging messages.

Chapter 3

User Guide

This page is directed to developers of haptic applications on embedded platforms.

Here is explained how to setup the development environment in different configurations.

Operating Systems supported

Specific instruction on how to setup the system follows.

- Linux

  This page documents the tools needed to setup the environment on a Linux system.

- Windows

  This page documents the tools needed to setup the environment on a Windows system.

3.1 Linux

Note

This page is directed to developers of haptic applications on embedded platforms.

Here is explained how to setup the development environment in different configurations on **Linux** Operating System.

3.1.1 Preliminary Setup

Before start, make sure the system provide basic GNU development tools like autotools. Most linux distributions include such tools. Target code compilation is managed using Makefiles that, along with the tool-chain, is the only tool needed to produce code. However, in order to debug application, a gdb-server interfacing with the Hardware target device is needed. To produce code documentation a recent version of doxygen is used. The following section will explain all the single components used in the development work flow.

3.1.2 Tool-chains supported

The tool-chain used in HapticLib development is linaro gcc targeting ARM Cortex-M3 platform, compiled locally using a user-friendly script.

3.1.2.1 ARM

At the moment the ARM platform is the only one supported. In particular the only physical target board used has a Cortex-M3 MCU. Any gcc-derived toolchain should work without problem, but only linaro tool-chain has been tested.

3.1.2.1.1 linaro

The linaro gcc tool-chain has been locally compiled using the building script `summon-arm-toolchain`. It just download all the software components from their respective web-sites and then compiles everything inside a target directory holding the toolchain (including newlib C library and gdb). At the end of the building process, you can just add the resulting bin/ path to the system $PATH and the tool-chain's tools will be available.

3.1.2.1.2 codesourcery

This toolchain has not been tested, but it should work just fine.

3.1.2.2 MSP430

At the moment the library dooesn't support this embedded platform. Probably there should not be any problem porting HapticLib to it.

3.1.2.3 PIC

At the moment the library dooesn't support this embedded platform. Probably there should not be any problem porting HapticLib to it.

3.1.3 GDB Servers supported

The gdb server interfacing with the only dev board (STM32VLDISCOVERY) used during HapticLib development was `texane/stlink`. It is an open source gdb server capable of flashing/erasing flash memory, and debugging running code using stlink protocol with the device and gdb protocol with the gdb client from the toolchain.

3.1.4 Flasher utilities

To flash binary code to the target device (STM32VLDISCOVERY) `texane/stlink` was used. (see GDB server section).

3.1.5 IDE supported

3.1.5.1 No IDE

The library is versatile enough to be used without a complete IDE; in fact you can compile the application and the library (including this documentation) using the provided Makefiles.

The first versions of HapticLib with the demo applications (up to v0.2) were developed on linux using only the `vim` text editor. The code was then compiled using `GNU make` which in turns uses the gcc toolchain to produce the binary code. To flash the code on the device, `texane/st-link` was used. Finally to debug the gdb included in the toolchain was used.

3.1.5.2 Eclipse

Eclipse IDE allow the user to create a Makefile based project. Starting with v0.2 development has been done with Eclipse, offering user-friendly integrated features like CVS and debugging. Still, the use of an IDE is optional.

Makefile building system is still at the core. Some fine-tuning were necessary to make the debugger work inside Eclipse.

3.2 Windows

Note

This page is directed to developers of haptic applications on embedded platforms.

Here is explained how to setup the development environment in different configurations using **Windows** Operating System.

3.2.1 Preliminary Setup

Before start using HapticLib you need to correctly prepare the system. Haptic applications and the library (including this documentation) are compiled using the provided Makefiles (refer to No IDE paragraph for explanations).

Therefore a version of `GNU make.exe` for Windows is necessary.

Furthermore some other Unix Tools are necessary to correctly execute all the operations defined in Makefiles:

- bash.exe
- rm.exe
- mkdir.exe
- cp.exe
- mv.exe

For these reasons a more complete Unix Tools Package for Windows is suggested.

For example two tested packages are:

- `MinGW`
- `Cygwin`

Note

For Cygwin: there is no need to install everything, select only the basic, developer and library parts.

After choosing the preferred package, make sure the above commands are available in PATH.

Note

Developers interested in building documentation must have in PATH Doxygen, LaTeX and ghost script tools. Please refer to Developer Guide for detailed informations.

3.2.2 Tool-chains supported

Now that the operating system is correctly setup, a tool-chain is needed to compile the code. Make sure the compiler collection binaries (arm-none-eabi-$*$) are available in PATH.

A list of tool-chains tested, divided by Platform kind, is provided.

3.2.2.1 ARM®

All the arm tool-chains are similar (all originates from gcc) and probably will work just fine.

The tested ones are:

3.2.2.1.1 Linaro - GCC ARM Embedded

Get the Linaro - GCC ARM Embedded tool-chain.

3.2.2.1.2 Atollic - TrueSTUDIO for ARM Lite

Get the Atollic - TrueSTUDIO for ARM Lite IDE that include tool-chain and gdb-server.

Note

The executable downloaded can be extracted like a zip archive and then only the tool-chain folder (PATH_WHERE_EXTRACT\$_OUTDIR\ARMTools) can be token and used as is.

Warning

Atollic tool-chain binaries are called arm-atollic-eabi-∗ instead of arm-none-eabi-∗.

3.2.2.1.3 Mentor Graphics - Sourcery CodeBench Lite Edition

Get the Mentor Graphics - Sourcery CodeBench Lite Edition tool-chain.

Note

With this tool-chain comes *cs-make.exe* and *cs-rm.exe* that can be used if preferred.

ARM - ARMCC toolchain

This toolchain comes with KEIL's uVision IDE. It works and is supported in HapticLib development. You can download here a ready to use uVision4 workspace with all it's needed.

3.2.2.2 Texas Instruments™ MSP430®

At the moment the library dooesn't support this embedded platform. Probably there should not be any problem porting HapticLib to it.

3.2.2.3 Microchip™ PIC32®

At the moment the library dooesn't support this embedded platform. Probably there should not be any problem porting HapticLib to it.

3.2.2.4 Atmel™ AVR32®

At the moment the library dooesn't support this embedded platform. Probably there should not be any problem porting HapticLib to it.

3.2.3 Flasher utilities

ST Microelectronics™STM32VLDISCOVERY

For correctly attach and use this developing board to Windows based systems, drivers and flasher utility are needed.

Get the official ST Microelectronics - ST-LINK Utility package and install them.

3.2.4 GDB Servers supported

ST Microelectronics™STM32VLDISCOVERY

Projects made for this dev-board can be debugged using gdb that include two elements: gdb-server and gdb-client. While the client is included in tool-chains (arm-none-eabi-gdb.exe), the server must be installed apart.

A gdb-server for Windows tested with this dev-board is provided by Atollic.

Get the `Atollic - TrueSTUDIO for ARM Lite`, extract the executable like a zip archive and take the folder PATH_WHERE_EXTRACT\Servers\ST-LINK_gdbserver .

Warning

ST-LINK_gdbserver.exe doesn't work out of the box: some other tricks are needed:

- Override STLinkUSBDriver.dll with the one that is provided by STM in ST-LINK Utility.
- Ignore the ST-Link_V2_USBdriver.exe included. Those provided by STM in ST-LINK Utility must be used.
- ST-LINK_gdbserver.exe must be invoked with `-d` parameter which enable SWD Debug mode.
- The default listening port of the gdb-server is 61234 but can be changed invoking it with `-p port_num` parameter.

3.2.5 IDE supported

3.2.5.1 No IDE

The library is versatile enough to be used without a complete IDE; in fact you can compile the application and the library (including this documentation) using the provided Makefiles.

The first versions of HapticLib with the demo applications (up to v0.2) were developed in linux using only the `vim` text editor, but every other text editor can be used too. The code was then compiled using `GNU make` which in turns uses the gcc toolchain to produce the binary code. In Windows refer to pre-requirements and tool-chain paragraphs. To flash the code on the device, `texane/st-link` was used. In Windows refer to flasher utility and/or gdb server sections. Finally to debug the gdb included in the toolchain was used; this last step is also valid under Windows.

3.2.5.2 Eclipse

Eclipse IDE allow the user to create a Makefile based project. Starting with v0.2 development has been done with Eclipse, offering user-friendly integrated features like CVS and debugging. Still, the use of an IDE is optional. Makefile building system is still at the core. Some fine-tuning were necessary to make the debugger work inside Eclipse.

3.2.5.3 KEIL MDK uVision4

Even though KEIL's IDE doesn't support Makefile projects, HapticLib developers added support to uVision4 enabling development of haptic application with this tool. A ready-to-use uVision4 workspace is `available here` with all the demo application of HapticLib project.

Warning

The IDE version tested is v4.60. Please make sure to have this version of KEIL's uVision installed along with ST-LINK Utility from STM.

Note

HapticLib main development take place within a different platform than KEIL's. Some features are not available out-of-the-box using uVision4 IDE. It is still possible though to have DBG and REL versions of compiled applications.

Chapter 4

Developer Guide

In this page are explained internal details of HapticLib. The Developer that wants to extend or modify the library will find this page useful.

4.1 SystemDesc system descriptor.

SystemDesc is a global structure holding the references of the application runtime state.

It holds a reference for each initialized haptor and a reference for each initialized pattern.

Using this variable all library code can interact with the rest of the system easily.

Additionally, here are stored the PWM driving signal frequency and the global inter-samples delay.

The scope of this variable would be global, but a preprocessor conditionally inclusion has been made to avoid the exposure of this variable to the user application code.

If the user needs access to this system variable, he must define the HL_SYSTEM_FILE preprocessor symbol on the module using it.

The actual structure definition of system_desc is the following:

```
typedef struct system_desc {
     uint8_t              num_haptors;
     haptor_desc          haptors[MAX_HAPTORS];
     pattern_desc         patterns[MAX_HAPTORS];
     uint32_t             pwm_freq;
     uint8_t              samples_delay;
} system_desc;
```

MAX_HAPTORS is a platform specific definition to limit the actual maximum number of haptors available.

Note

Using static arrays to hold patterns and haptors references is not a smart way to use the memory resource. This way, irrespective of the actual number of haptors/patterns initialized, the maximum number of references is hold in memory. Future versions may include a dynamic memory allocator function, or directly link to a C library offering this kind of features (like the `newlib` C library).

4.2 Haptor Descriptor

SystemDesc holds a reference to each haptor initialized. The reference is a descriptor holding the following information about the haptor:

```
typedef struct haptor_desc {
```

```
    uint8_t             id;
    uint16_t            min_duty;
    uint16_t            max_duty;
    pattern_desc       *activePattern;
    struct haptor_desc *nextHaptor;
   } haptor_desc;
```

min_duty and max_duty are useful to calibrate the range of operation of the physical device.

activePattern refer to the descriptor of the pattern attached to this haptor (if any).

nextHaptor is a reference to another haptor. This pointer creates a linked list of haptors belonging to the same group. It is used to implement multi-haptor patterns. (see below for further informations)

4.3 Pattern Descriptor

SystemDesc hold a reference to each active pattern in the running system.

The reference is a descriptor of a pattern holding the following data:

```
typedef struct pattern_desc {
  pattern_name           name;
  user_param        *userParams;
  status_param           statusParams;
  pattern_continuator continuator;
  struct haptor_desc *activeHaptorList;
} pattern_desc;
```

pattern_name is a numeric id (int) handled as enumeration (pattern_name) to make code cleaner and easy to read.

userParams is a reference to a structure holding the user provided parameters needed by the pattern.

Note

user_param is defined by the pattern developer. The user application must conform to it. The actual variable holding this data is declared by the user in one of its modules (generally main.c) HapticLib just refers to it. (refer below to understand why)

statusParams is a data structure holding the state of the pattern. This structure is defined by the pattern developer and is not exposed to the user.

continuator is a function pointer to the code that will be executed at every scheduled instant by this active pattern. If this pointer is NULL, then the system will consider this pattern not active.

the continuator is a function defined by the pattern developer and is not exposed to the user.

activeHaptorList points to the first haptor of the linked list forming the group of attached haptors. Using this list a pattern can easily developed as multi-haptor.

4.4 Pattern Rendering

When a pattern is initialized, an available slot on SystemDesc is searched for and, if found, the pattern goes there.

The pattern now has no haptor attached to it, so the user must add every haptor to it. At every call of hl_addHaptor() the linked-list is formed for later references by the pattern code.

Now the pattern is ready to be started for the actual rendering.

The rendering of a pattern has two main steps:

- initiator

- continuator

When the pattern is started, a function called pattern initiator is called to initialize the pattern initial status. userParams is used to set the initial status of the pattern that is hold in statusParams.

The initiator must set the pattern continuator on its pattern descriptor.

This operation is the actual start of the pattern, as the patternScheduler check for this pointer to be not-NULL to execute the pattern continuator code.

From now on, when the scheduler decides, the pattern continuator is executed.

The actual logic inside the continuator, is up to the pattern developer.

Usually the task carried by the continuator are:

- read the user parameters for contingent run-time events.
- update the status to reflect the time flow.
- implement an exit strategy

The userParams polling is a key features to let the user drive the pattern logic (as coded by the pattern developer).

The exit strategy is a way for the pattern to self-deallocate from scheduling. This is not mandatory, but the pattern developer must explicitly document the behavior. For example, a pattern could self-stop after some time of execution. Another pattern could run indefinitely and stop only when the user forces its stop calling hl_stopPattern().

4.5 Develop a New Pattern Generator

To create a new pattern generator, the easy way is to copy an existing one and the modify it.

Here are the steps to implement a new test pattern generator:

1. Create a new directory inside HapticLib/patterns: test2
2. Create test2.c and test2.h source code files.
3. add the new pattern in hl_patterns.c / hl_patterns.h

test2.h will contain the definitions of the new pattern:

- test2StatusParameters
- test2UserParameters (optional)
- other optional structures or definitions needed by the pattern.

test2.c will contain at least initiator and continuator code:

- test2PatternGenerator(): the initiator
- test2Continuator(): the continuator
- Doxygen embedded documentation to describe the pattern.

Be sure to implement a correct exit strategy on the continuator triggered by some event (could be a value on some userParam set by the user application code), or explicitly state on pattern documentation that the user must call hl_stopPattern() to stop it.

The documentation template is as follow:

- Introduction
- Theory
- Optional Parameters
- Usage Examples
- Debugging Details
- Additional Notes

Inside hl_patterns.h, references to the new pattern must be added in 4 points:

- add an include to the header: #include "test2.h"
- update pattern_name enumeration with the new pattern name before the last entry: Num_Patterns_Available.
- update status_param union to contain the new statusParams.
- update user_params (if any) to contain the new userParams.

Inside hl_patterns.c, the new pattern must be added in 2 points:

- define a new extern pattern initiator
- add the pointer to the patternMap

4.6 Adding a New Platform

Adding a new platform, like for patterns, consists in adding a new sub-directory inside HapticLib/platforms.

Warning

As for now (HapticLib v.0.7) a cleaner interface between the platform specific module and the rest of the library need to be defined. There are some direct references inside hapticLib.c that are platform specific and need to be moved.

4.7 HapticLib vs. bare application comparison

Comparison of HapticLevel demo application with a modified version of MEMS demo application from STM.

Instructions for HapticLevel compilation:

- Download the workspace_keil.zip environment
- Launch Keil project for HapticLevel demo
- Set "REL" target, and compile
- use arm-none-eabi-objcopy -I ihex HapticLevel.hex -O binary HapticLevel.bin
- HapticLevel.bin is produced

Instructions for modified demo compilation:

- Download original package from STM here
- Download diff patch here

- Unzip and go into the unzipped directory
- Apply patch: patch -p1 < ../bare_app.patch
- Start project using Keil project located in STM32F4-Discovery_FW_V1.1.0/Project/Peripheral_Examples/MEMS/MDK-ARM and compile.
- launch arm-none-eabi-objcopy -I ihex LIS302DL.hex -O binary LIS302DL.bin inside STM32F4-Discovery_FW_V1.1.0/Project/Peripheral_Examples/MEMS/MDK-ARM/LIS302DL dir
- LIS302DL.bin is produced

Chapter 5

References

Here there are links of all the references of HapticLib.

- Example Reference Text
- Hachisu et al - Pseudo-haptic feedback augmented with visual and tactile vibrations
- Okamura et al - Reality-based models for vibration feedback in virtual environments
- GNU make.exe
- MinGW
- Cygwin
- Linaro - GCC ARM Embedded
- Atollic - TrueSTUDIO for ARM Lite
- Mentor Graphics - Sourcery CodeBench Lite Edition
- ST Microelectronics - ST-LINK Utility

Chapter 6

Todo List

Global hl_configure (uint32_t, uint8_t, uint8_t)

Add input validation on pwm_freq and samples_delay passed parameters.

Global hl_configure (uint32_t, uint8_t, uint8_t)

Add input validation on pwm_freq and samples_delay passed parameters.

File hl_patterns.h

Define Return values constants for pattern generators.

Global MAX_PATTERNS

Implement a re-mapping mechanism to easily allow the user the selection of some specific patterns discarding the others.

Global TIM_Channel_DutyChanger (uint16_t, uint8_t)

Clarify the ∗∗User API∗∗ <−> ∗∗Platform Specific∗∗ interface; this logic may need to be changed.

Global TIM_Channel_DutyChanger (uint16_t, uint8_t)

Clarify the ∗∗User API∗∗ <−> ∗∗Platform Specific∗∗ interface; this logic may need to be changed.

Global TIM_Channel_Enable (uint8_t)

Clarify the ∗∗User API∗∗ <−> ∗∗Platform Specific∗∗ interface; this logic may need to be changed.

Global TIM_Channel_Enable (uint8_t)

Clarify the ∗∗User API∗∗ <−> ∗∗Platform Specific∗∗ interface; this logic may need to be changed.

Chapter 7

Data Structure Index

7.1 Data Structures

Here are the data structures with brief descriptions:

constantStatusParameters
: Pattern specific **status** parameters type definition . . . 41

constantUserParameters
: Pattern specific **user** parameter type definition . . . 41

genericStatusParameters
: Pattern specific **status** parameters type definition . . . 42

genericUserParameters
: Pattern specific **user** parameters type definition . . . 43

haptor_desc
: Single haptor descriptor . . . 43

impactStatusParameters
: Impact pattern **status** parameters type definition . . . 45

impactUserParameters
: Struct for user provided parameters to hl_startPattern() . . . 45

pattern_desc
: Pattern descriptor . . . 46

status_param
: Pattern specific status parameters container . . . 47

testStatusParameters
: Test Pattern specific **status** parameters typedef . . . 48

user_param
: Pattern specific optional user parameters container . . . 49

Chapter 8

File Index

8.1 File List

Here is a list of all files with brief descriptions:

HapticLib/hapticLib.c
User API Module definitions . . . 51
HapticLib/hapticLib.h
User API Module header . . . 56
HapticLib/hl_debug.c
Debugging Features definitions . . . 61
HapticLib/hl_debug.h
Debugging Features header . . . 64
HapticLib/patterns/hl_patterns.c
Pattern Generator Module definitions . . . 77
HapticLib/patterns/hl_patterns.h
Pattern Generator Module headers . . . 82
HapticLib/patterns/constant/constant.c
Constant Pattern generator function definition . . . 67
HapticLib/patterns/constant/constant.h
Constant Pattern generator function header . . . 70
HapticLib/patterns/generic/generic.c
Generic Pattern generator function definition [TEMPLATE] . . . 70
HapticLib/patterns/generic/generic.h
Generic Pattern generator function header [TEMPLATE] . . . 73
HapticLib/patterns/impact/impact.c
Impact Pattern generator function definition . . . 92
HapticLib/patterns/impact/impact.h
Impact Pattern generator function header . . . 98
HapticLib/patterns/impact/extra/impact.m . . . 87
HapticLib/patterns/test/test.c
Test Pattern generator function definition . . . 100
HapticLib/patterns/test/test.h
Test Pattern generator function header . . . 102
HapticLib/platforms/STM32VLDISCOVERY/hl_STM32VLDISCOVERY.c
Platform Specific Module STM32VLDISCOVERY definitions . . . 103
HapticLib/platforms/STM32VLDISCOVERY/hl_STM32VLDISCOVERY.h
Platform Specific Module STM32VLDISCOVERY header . . . 109

Chapter 9

Data Structure Documentation

9.1 constantStatusParameters Struct Reference

Pattern specific **status** parameters type definition.

`#include <constant.h>`

Data Fields

- uint16_t duty

  `duty` *value, used to feed the new duty cycle to the PWM and together keep track of it.*

9.1.1 Detailed Description

Pattern specific **status** parameters type definition.

Two status parameters will be used.

Definition at line 53 of file constant.h.

9.1.2 Field Documentation

9.1.2.1 uint16_t duty

`duty` value, used to feed the new duty cycle to the PWM and together keep track of it.

Definition at line 54 of file constant.h.

The documentation for this struct was generated from the following file:

- HapticLib/patterns/constant/constant.h

9.2 constantUserParameters Struct Reference

Pattern specific **user** parameter type definition.

`#include <constant.h>`

Data Fields

- uint16_t constant

value of desired magnitude in PWM-LEVEL (0..65000)

9.2.1 Detailed Description

Pattern specific **user** parameter type definition.

User must provide a constant value from 0 to 65000 as optional parameter.

Definition at line 42 of file constant.h.

9.2.2 Field Documentation

9.2.2.1 uint16_t constant

value of desired magnitude in PWM-LEVEL (0..65000)

Definition at line 43 of file constant.h.

The documentation for this struct was generated from the following file:

- HapticLib/patterns/constant/constant.h

9.3 genericStatusParameters Struct Reference

Pattern specific **status** parameters type definition.

`#include <generic.h>`

Data Fields

- uint8_t flag

  a typical status parameter could be a flag.
- uint16_t duty

  another common status parameter can be a `duty` value, used to feed the new duty cycle to the PWM and together keep track of the progress.

9.3.1 Detailed Description

Pattern specific **status** parameters type definition.

At least one parameter should be defined to help the **continuator** keep track of the pattern generation progress.

See also

Refer to generic.c for a simple working example.

Definition at line 145 of file generic.h.

9.3.2 Field Documentation

9.3.2.1 uint16_t duty

another common status parameter can be a `duty` value, used to feed the new duty cycle to the PWM and together keep track of the progress.

Definition at line 148 of file generic.h.

9.3.2.2 uint8_t flag

a typical status parameter could be a flag.

Definition at line 146 of file generic.h.

The documentation for this struct was generated from the following file:

- HapticLib/patterns/generic/generic.h

9.4 genericUserParameters Struct Reference

Pattern specific **user** parameters type definition.

`#include <generic.h>`

Data Fields

- genericIncrement increment
  An example user parameter.
- genericCheckParam checkParam
  Another example user parameter.

9.4.1 Detailed Description

Pattern specific **user** parameters type definition.

This structure MUST be clearly documented to avoid pitfalls on the generator's use by the application code.

The user provided values, must be validated by the (preferably) pattern **initiator** before using them in the pattern logic.

Definition at line 127 of file generic.h.

9.4.2 Field Documentation

9.4.2.1 genericCheckParam checkParam

Another example user parameter.

Definition at line 132 of file generic.h.

9.4.2.2 genericIncrement increment

An example user parameter.

Definition at line 129 of file generic.h.

The documentation for this struct was generated from the following file:

- HapticLib/patterns/generic/generic.h

9.5 haptor_desc Struct Reference

Single haptor descriptor.

`#include <hapticLib.h>`

Data Fields

- uint8_t id

  Unique ID to designate the haptor.
- uint16_t min_duty

  Minimum duty cycle to activate the tactor.
- uint16_t max_duty

  Maximum duty cycle to feed to the tactor.
- pattern_desc * activePattern

  Reference to the active pattern driving this haptor.
- struct haptor_desc * nextHaptor

  Reference to another haptor_desc.

9.5.1 Detailed Description

Single haptor descriptor.

HapticLib keep track of all the haptors configured in the system using one haptor_desc struct for each of them.

To know what kind of informations holds, please refer to its data structure haptor_desc.

Definition at line 1921 of file hapticLib.h.

9.5.2 Field Documentation

9.5.2.1 pattern_desc* activePattern

Reference to the active pattern driving this haptor.

If it is NULL the haptor is free to be activated by a pattern.

Definition at line 1937 of file hapticLib.h.

9.5.2.2 uint8_t id

Unique ID to designate the haptor.

Note

This is a temporary solution waiting for a cleaner **User API** <–> **Platform Specific** interface to be defined.

Definition at line 1922 of file hapticLib.h.

9.5.2.3 uint16_t max_duty

Maximum duty cycle to feed to the tactor.

Definition at line 1934 of file hapticLib.h.

9.5.2.4 uint16_t min_duty

Minimum duty cycle to activate the tactor.

Definition at line 1931 of file hapticLib.h.

9.5.2.5 struct haptor_desc* nextHaptor

Reference to another haptor_desc.

This links create a list of haptors chained together to form an **haptor group** driven by the same pattern.

Definition at line 1946 of file hapticLib.h.

The documentation for this struct was generated from the following file:

- HapticLib/hapticLib.h

9.6 impactStatusParameters Struct Reference

Impact pattern **status** parameters type definition.

`#include <impact.h>`

Data Fields

- uint8_t progress

9.6.1 Detailed Description

Impact pattern **status** parameters type definition.

Only one parameter is defined to help the **continuator** keep track of the pattern generation progress.

Definition at line 83 of file impact.h.

9.6.2 Field Documentation

9.6.2.1 uint8_t progress

Definition at line 84 of file impact.h.

The documentation for this struct was generated from the following file:

- HapticLib/patterns/impact/impact.h

9.7 impactUserParameters Struct Reference

Struct for user provided parameters to hl_startPattern().

`#include <impact.h>`

Data Fields

- ImpactVelocity velocity

  Impact `velocity` parameter.
- ImpactMaterial material

  Impact `material` parameter.

9.7.1 Detailed Description

Struct for user provided parameters to hl_startPattern().

Please refer to impact pattern generator optional parameters to know the possible values.

Definition at line 69 of file impact.h.

9.7.2 Field Documentation

9.7.2.1 ImpactMaterial material

Impact `material` parameter.

Definition at line 72 of file impact.h.

9.7.2.2 ImpactVelocity velocity

Impact `velocity` parameter.

Definition at line 70 of file impact.h.

The documentation for this struct was generated from the following file:

- HapticLib/patterns/impact/impact.h

9.8 pattern_desc Struct Reference

Pattern descriptor.

`#include <hl_patterns.h>`

Data Fields

- pattern_name name

  Running Pattern type.
- user_param * userParams

  Pointer to the optional user parameters of the pattern.
- status_param statusParams

  Structure holding the pattern specific status informations of the generator.
- pattern_continuator continuator

  pattern continuator callback function pointer.
- struct haptor_desc * activeHaptorList

  Pointer to the first haptor_desc structure forming the list of haptors activated by this pattern.

9.8.1 Detailed Description

Pattern descriptor.

The pattern descriptor holds all the relevant information about a pattern being run on a specific haptor.

Definition at line 211 of file hl_patterns.h.

9.8.2 Field Documentation

9.8.2.1 struct haptor_desc* activeHaptorList

Pointer to the first haptor_desc structure forming the list of haptors activated by this pattern.

Definition at line 227 of file hl_patterns.h.

9.8.2.2 pattern_continuator continuator

pattern continuator callback function pointer.

Definition at line 224 of file hl_patterns.h.

9.8.2.3 pattern_name name

Running Pattern type.

Definition at line 213 of file hl_patterns.h.

9.8.2.4 status_param statusParams

Structure holding the pattern specific status informations of the generator.

Definition at line 219 of file hl_patterns.h.

9.8.2.5 user_param* userParams

Pointer to the optional user parameters of the pattern.

Definition at line 215 of file hl_patterns.h.

The documentation for this struct was generated from the following file:

- HapticLib/patterns/hl_patterns.h

9.9 status_param Union Reference

Pattern specific status parameters container.

`#include <hl_patterns.h>`

Data Fields

- genericStatusParameters generic

  Generic Pattern Status Parameters.
- testStatusParameters test

  Test Pattern Status Parameters.
- impactStatusParameters impact

  Impact Pattern Status Parameters.
- constantStatusParameters constant

  Constant Pattern Status Parameters.

9.9.1 Detailed Description

Pattern specific status parameters container.

This type definition is useful to increase the readability of the code and to enforce compiler type checking when passing arguments to functions.

The status parameters are intended only for the pattern developer. These parameters are not exposed outside the pattern module.

See also

Please refer to the generic Pattern Template Generator (generic.c) for usage example of the status parameters.

Definition at line 126 of file hl_patterns.h.

9.9.2 Field Documentation

9.9.2.1 constantStatusParameters constant

Constant Pattern Status Parameters.

Definition at line 136 of file hl_patterns.h.

9.9.2.2 genericStatusParameters generic

Generic Pattern Status Parameters.

Definition at line 127 of file hl_patterns.h.

9.9.2.3 impactStatusParameters impact

Impact Pattern Status Parameters.

Definition at line 133 of file hl_patterns.h.

9.9.2.4 testStatusParameters test

Test Pattern Status Parameters.

Definition at line 130 of file hl_patterns.h.

The documentation for this union was generated from the following file:

- HapticLib/patterns/hl_patterns.h

9.10 testStatusParameters Struct Reference

Test Pattern specific **status** parameters typedef.

`#include <test.h>`

Data Fields

- uint8_t flag

  status parameter *is used to implement the saw-tooth double ramp.*
- uint16_t duty

  status parameter *is used to hold the new duty-cycle for the PWM and to keep track of the pattern generation progress.*

9.10.1 Detailed Description

Test Pattern specific **status** parameters typedef.

Status parameter used by the continuator.

See also

Refer to generic.c for a simple working example.

Definition at line 50 of file test.h.

9.10.2 Field Documentation

9.10.2.1 uint16_t duty

status parameter is used to hold the new duty-cycle for the PWM and to keep track of the pattern generation progress.

Definition at line 53 of file test.h.

9.10.2.2 uint8_t flag

status parameter is used to implement the saw-tooth double ramp.

Definition at line 51 of file test.h.

The documentation for this struct was generated from the following file:

- HapticLib/patterns/test/test.h

9.11 user_param Union Reference

Pattern specific optional user parameters container.

`#include <hl_patterns.h>`

Data Fields

- impactUserParameters impact

  Impact Pattern User Parameters.
- genericUserParameters generic

  Generic Pattern User Parameters.
- constantUserParameters constant

  Constant Pattern User Parameters.

9.11.1 Detailed Description

Pattern specific optional user parameters container.

This type definition is useful to increase the readability of the code and to enforce compiler type checking when passing parameters to patterns.

Usage example of the Impact Pattern:

```
...
haptor_desc *myHaptor = hl_configure(24000000,10,1);

pattern_desc *myImpactPattern;

user_param myParams;

myParams.impact.velocity = Fast;
myParams.impact.material = Rubber;

myImpactPattern = hl_initPattern(Impact,&myParams);

hl_addHaptor(myHaptor,myImpactPattern);

hl_startPattern(myImpactPattern);
...
```

Definition at line 170 of file hl_patterns.h.

9.11.2 Field Documentation

9.11.2.1 constantUserParameters constant

Constant Pattern User Parameters.

Definition at line 175 of file hl_patterns.h.

9.11.2.2 genericUserParameters generic

Generic Pattern User Parameters.

Definition at line 173 of file hl_patterns.h.

9.11.2.3 impactUserParameters impact

Impact Pattern User Parameters.

Definition at line 171 of file hl_patterns.h.

The documentation for this union was generated from the following file:

- HapticLib/patterns/hl_patterns.h

Chapter 10

File Documentation

10.1 HapticLib/hapticLib.c File Reference

User API Module definitions

```
#include "hapticLib.h"
```

Macros

- #define HL_SYSTEM_FILE

Functions

- haptor_desc * hl_configure (uint32_t pwmFreq, uint8_t samplesDelay, uint8_t numHaptors)
  API exposed to the user *to setup the system.*
- pattern_desc * hl_initPattern (pattern_name patternName, user_param *userParams)
  API exposed to the user *to initialize a new pattern.*
- pattern_desc * hl_addHaptor (haptor_desc *newHaptor, pattern_desc *pattern)
  API exposed to the user *to add an haptor to a pattern.*
- uint8_t hl_startPattern (pattern_desc *pattern)
  API exposed to the user *to send Haptic Feedback.*
- uint8_t hl_stopPattern (pattern_desc *pattern)
  API exposed to the user *to force pattern execution stop.*

Variables

- pattern_initiator patternMap [MAX_HAPTORS]
  Pattern Generators Functions Map.
- system_desc SystemDesc
  Global variable to describe the haptic system.

10.1.1 Detailed Description

User API Module definitions This file is part of the **User API Module** of HapticLib.

The code inside this module is meant to be platform independent.

The functions defined in hapticLib.c are the only ones the library user should need to call for his haptic applications.

Note

The only exception is the Delay() function which now reside in the **Platform Specific Module**.

See also

Please refer to the Architecture page for a complete overview of HapticLib structure.
Please refer to the Developer Guide for the internals of HapticLib's Modules interactions.

Author

Leonardo Guardati

Version

0.7

Date

2012

Definition in file hapticLib.c.

10.1.2 Macro Definition Documentation

10.1.2.1 #define HL_SYSTEM_FILE

Definition at line 47 of file hapticLib.c.

10.1.3 Function Documentation

10.1.3.1 pattern_desc* hl_addHaptor (haptor_desc * *newHaptor,* pattern_desc * *pattern*)

API exposed to the user to add an haptor to a pattern.

The user needs to call hl_addHaptor() to let an already initialized pattern know which haptors to activate with its logic.

If the haptor is good (and free) and the pattern is good, hl_addHaptor() will link the pattern to the haptor and, vice-versa. If the pattern has already haptors linked to it, the `newHaptor` will be added to the list formed by all the haptors already attached to the pattern.

Parameters

`out`	*newHaptor*	This is the pointer to the haptor to be added to the list of attached haptors of the pattern.
`out`	*pattern*	This is the pointer to a `pattern_desc` returned by a precedent hl_initPattern() call.

Returns

`pattern_desc *` hl_addHaptor() returns the same pointer passed as argument by the caller.

Return values

NULL	If the reference is NULL, an error has occurred.

Note

Returning the `pattern_desc *` of the pattern, it is possible to nest the calling sequence that lead to a single haptor pattern activation.

```
hl_startPattern( hl_addHaptor( &myHaptors[2], hl_initPattern(Test,NULL) ) );
```

Definition at line 280 of file hapticLib.c.

10.1.3.2 haptor_desc* hl_configure (uint32_t *pwmFreq*, uint8_t *samplesDelay*, uint8_t *numHaptors*)

API exposed to the user to setup the system.

Here the global SystemDesc variable is initialized. Then all the system peripheral get configured based on the high level informations passed by the user.

Parameters

in	*pwmFreq*	This is a `uint32_t` used to specify the Frequency of the haptors control PWM signals. Expressed in Hz.
in	*samplesDelay*	This is a `uint8_t` used to specify the delay (in ms) between successive haptors control PWM signals updates.

Note

Please refer to the Platform Specific Module implementations of Timers and PWM generations to be sure of the meaning of these parameters.

Parameters

in	*numHaptors*	This is a `uint8_t` to specify the actual number of haptors the application will use.

Note

It is an important feature to be able to specify a minor number of devices to use instead of the hardware defined MAX_HAPTORS. Doing so, not only the code will be faster and the memory footprint smaller (todo), but also the resources that drive the unused haptors will be available to the application for other uses. (todo)

Returns

`haptor_desc *` A reference to the array of the configured haptor is returned to the application.

Return values

NULL	If the reference is NULL, an error has occurred.

Todo Add input validation on pwm_freq and samples_delay passed parameters.

Definition at line 109 of file hapticLib.c.

10.1.3.3 pattern_desc* hl_initPattern (pattern_name *patternName*, user_param * *userParams*)

API exposed to the user to initialize a new pattern.

Calling hl_initPattern(), the system will check for available resources and then create the new pattern descriptor returning its reference to the application who called.

The pattern repository offer a wide selection of haptic feedback patterns.

The user must read the documentation of the patterns she wants to use, in order to know if and what kind of **user parameters** the pattern needs.

Parameters

`in`	*patternName*	This is an integer value (encoded with the pattern_name enumeration) used to select the desired pattern.
`out`	*userParams*	This is a pointer to the user_param structure the application must allocate to store the parameters needed by the pattern.

Note

user_param is a union type definition. With this type the compiler can enforce type checking at compile time to ensure a valid argument is passed. (earlier version of HapticLib used a `void *` to pass the user parameters).

Returns

`pattern_desc *` A pointer to the pattern descriptor is returned to the application. The application will use this descriptor to make successive operations related to this pattern (like, adding haptors to it, starting it, stopping it).

Return values

NULL	If the reference is NULL, an error has occurred.

Definition at line 209 of file hapticLib.c.

10.1.3.4 uint8_t hl_startPattern (pattern_desc * *pattern*)

API exposed to the user to send Haptic Feedback.

hl_startPattern() is used to activate an already initialized pattern with a set of haptors attached to it.

After some validation checks on the pattern passed (it must be ready to be activated), hl_startPattern() will call the pattern's initiator code.

See also

To learn about how a pattern generator works, please read the Developer Page.

hl_startPattern() uses the patternMap array to be able to call the right initiator.

Note

You cannot use the same pattern descriptor to activate the pattern simultaneously on different haptors.
If the pattern is multi haptor, you have to call addHaptor() for each haptor to add to the same pattern.
If the pattern is single-haptor, and you want multiple instances running simultaneously, you need to initialize a new pattern instance for each haptor, call the addHaptor() with the single haptor to attach and then you can startPattern() on all the patterns together.

Parameters

`out`	*pattern*	This is the pointer to a `pattern_desc *` returned by a precedent hl_initPattern() call. This `pattern_desc` should not be already running. If you want many instances of the same pattern, you need to follow all the steps to initialize it.

Returns

uint8_t The result is returned to indicate success or error.

Return values

0	Success
1	Error

Definition at line 364 of file hapticLib.c.

10.1.3.5 uint8_t hl_stopPattern (pattern_desc * *pattern*)

API exposed to the user to force pattern execution stop.

Normally, a well designed pattern will finish its job and then free its instances (the pattern itself and all the haptors used) to be eventually configured later.

There are cases however, when you cannot wait for the pattern to stop.

For example, imagine a continuous pattern that will run until the user feel the feedback, in this case, the pattern must be stopped by the application code (the condition cannot be coded inside the pattern) calling hl_stopPattern() with the right pattern descriptor.

Parameters

out	*pattern*	This is the pattern descriptor reference of a running pattern.

Returns

uint8_t The result is returned to indicate success or error.

Return values

0	Success
1	Error

Definition at line 412 of file hapticLib.c.

10.1.4 Variable Documentation

10.1.4.1 pattern_initiator patternMap[MAX_HAPTORS]

Pattern Generators Functions Map.

This array contains the addresses of all the pattern generator initiator functions.

The index of the array locate the specific pattern using one of the Pattern Generator Functions Index Names defined in the pattern_name enumeration type definition. (e.g. Test, Impact)

Definition at line 79 of file hl_patterns.c.

10.1.4.2 system_desc SystemDesc

Global variable to describe the haptic system.

The Library uses this global variable to describe the system and to keep track of the haptic devices' status and active patterns' progress at any time.

Note

The library user doesn't need to access this variable.

The SystemDesc variable holds a lot of informations, and its members allow the system to access almost all the informations needed.

See also

Please refer to the Developer Guide page for details.

Definition at line 70 of file hapticLib.c.

10.2 HapticLib/hapticLib.h File Reference

User API Module header

```
#include <stdint.h>
#include <stdlib.h>
#include "hl_patterns.h"
#include "hl_STM32VLDISCOVERY.h"
```

Data Structures

- struct haptor_desc

  Single haptor descriptor.

Macros

- #define HL_DEBUG

  *Define this symbol to **enable the Debugging Features**.*
- #define STM32VLDISCOVERY

  This symbol define what platform we are building for.

Typedefs

- typedef struct haptor_desc haptor_desc

  Single haptor descriptor.

Functions

- haptor_desc * hl_configure (uint32_t, uint8_t, uint8_t)

  ***API exposed to the user** to setup the system.*
- pattern_desc * hl_initPattern (pattern_name, user_param *)

  ***API exposed to the user** to initialize a new pattern.*
- pattern_desc * hl_addHaptor (haptor_desc *, pattern_desc *)

  ***API exposed to the user** to add an haptor to a pattern.*
- uint8_t hl_startPattern (pattern_desc *)

  ***API exposed to the user** to send Haptic Feedback.*
- uint8_t hl_stopPattern (pattern_desc *)

  ***API exposed to the user** to force pattern execution stop.*

10.2.1 Detailed Description

User API Module header This file is part of the **User API Module** of HapticLib.

This is the only file to be included by a project to use HapticLib.

This file is the entry point for the HapticLib user.

All the High Level API the user *should* use, are listed here.

Note

Please make sure the right symbols are set when invoking the compiler. (check the Makefile to be sure). For example, using the *STM32VLDISCOVERY* platform these symbols are needed:

- -DSTM32VLDISCOVERY (required by HapticLib **Platform Specific Module**)
- -DSTM32F10X_MD_VL (required by STM StdPeriph)
- -DUSE_STDPERIPH_DRIVER (required by STM StdPeriph)

Author

Leonardo Guardati

Version

0.7

Date

2012

Definition in file hapticLib.h.

10.2.2 Macro Definition Documentation

10.2.2.1 #define HL_DEBUG

Define this symbol to **enable the Debugging Features**.

The Debugging features offered by the library are:

- send_string()
- send_int()

The HapticLib Debugging Features are enabled based on the presence of the symbol HL_DEBUG .

To define the symbol do either:

- explicitly #`define HL_DEBUG` inside a module.
- add `-DHL_DEBUG` to the compiler command line during compilation.

Warning

Always make calls to debugging code inside conditionally included blocks.

```
...
#ifdef HL_DEBUG
        ...
        send_int( 1234567 );
        ...
        send_string("Correct usage.\n\r\0");
        ...
#endif
...
```

Definition at line 1873 of file hapticLib.h.

10.2.2.2 #define STM32VLDISCOVERY

This symbol define what platform we are building for.

This symbol is needed by the **Platform Specific Module** to select the right low level code implementations.

Note

You MUST define this symbol using the compiler command line.

Definition at line 1887 of file hapticLib.h.

10.2.3 Typedef Documentation

10.2.3.1 typedef struct haptor_desc haptor_desc

Single haptor descriptor.

HapticLib keep track of all the haptors configured in the system using one haptor_desc struct for each of them.

To know what kind of informations holds, please refer to its data structure haptor_desc.

10.2.4 Function Documentation

10.2.4.1 pattern_desc* hl_addHaptor (haptor_desc * *newHaptor,* pattern_desc * *pattern*)

API exposed to the user to add an haptor to a pattern.

The user needs to call hl_addHaptor() to let an already initialized pattern know which haptors to activate with its logic.

If the haptor is good (and free) and the pattern is good, hl_addHaptor() will link the pattern to the haptor and, vice-versa. If the pattern has already haptors linked to it, the `newHaptor` will be added to the list formed by all the haptors already attached to the pattern.

Parameters

`out`	*newHaptor*	This is the pointer to the haptor to be added to the list of attached haptors of the pattern.
`out`	*pattern*	This is the pointer to a `pattern_desc` returned by a precedent hl_initPattern() call.

Returns

`pattern_desc` * hl_addHaptor() returns the same pointer passed as argument by the caller.

Return values

NULL	If the reference is NULL, an error has occurred.

Note

Returning the `pattern_desc *` of the pattern, it is possible to nest the calling sequence that lead to a single haptor pattern activation.

```
hl_startPattern( hl_addHaptor( &myHaptors[2], hl_initPattern(Test,NULL) ) );
```

Definition at line 280 of file hapticLib.c.

10.2.4.2 haptor_desc* hl_configure (uint32_t *pwmFreq*, uint8_t *samplesDelay*, uint8_t *numHaptors*)

API exposed to the user to setup the system.

Here the global SystemDesc variable is initialized. Then all the system peripheral get configured based on the high level informations passed by the user.

Parameters

in	*pwmFreq*	This is a `uint32_t` used to specify the Frequency of the haptors control PWM signals. Expressed in Hz.
in	*samplesDelay*	This is a `uint8_t` used to specify the delay (in ms) between successive haptors control PWM signals updates.

Note

Please refer to the Platform Specific Module implementations of Timers and PWM generations to be sure of the meaning of these parameters.

Parameters

in	*numHaptors*	This is a `uint8_t` to specify the actual number of haptors the application will use.

Note

It is an important feature to be able to specify a minor number of devices to use instead of the hardware defined MAX_HAPTORS. Doing so, not only the code will be faster and the memory footprint smaller (todo), but also the resources that drive the unused haptors will be available to the application for other uses. (todo)

Returns

`haptor_desc *` A reference to the array of the configured haptor is returned to the application.

Return values

NULL	If the reference is NULL, an error has occurred.

Todo Add input validation on pwm_freq and samples_delay passed parameters.

Definition at line 109 of file hapticLib.c.

10.2.4.3 pattern_desc* hl_initPattern (pattern_name *patternName,* user_param * *userParams*)

API exposed to the user to initialize a new pattern.

Calling hl_initPattern(), the system will check for available resources and then create the new pattern descriptor returning its reference to the application who called.

The pattern repository offer a wide selection of haptic feedback patterns.

The user must read the documentation of the patterns she wants to use, in order to know if and what kind of **user parameters** the pattern needs.

Parameters

`in`	*patternName*	This is an integer value (encoded with the pattern_name enumeration) used to select the desired pattern.
`out`	*userParams*	This is a pointer to the user_param structure the application must allocate to store the parameters needed by the pattern.

Note

user_param is a union type definition. With this type the compiler can enforce type checking at compile time to ensure a valid argument is passed. (earlier version of HapticLib used a `void *` to pass the user parameters).

Returns

`pattern_desc *` A pointer to the pattern descriptor is returned to the application. The application will use this descriptor to make successive operations related to this pattern (like, adding haptors to it, starting it, stopping it).

Return values

NULL	If the reference is NULL, an error has occurred.

Definition at line 209 of file hapticLib.c.

10.2.4.4 uint8_t hl_startPattern (pattern_desc * *pattern*)

API exposed to the user to send Haptic Feedback.

hl_startPattern() is used to activate an already initialized pattern with a set of haptors attached to it.

After some validation checks on the pattern passed (it must be ready to be activated), hl_startPattern() will call the pattern's initiator code.

See also

To learn about how a pattern generator works, please read the Developer Page.

hl_startPattern() uses the patternMap array to be able to call the right initiator.

Note

You cannot use the same pattern descriptor to activate the pattern simultaneously on different haptors.
If the pattern is multi haptor, you have to call addHaptor() for each haptor to add to the same pattern.
If the pattern is single-haptor, and you want multiple instances running simultaneously, you need to initialize a new pattern instance for each haptor, call the addHaptor() with the single haptor to attach and then you can startPattern() on all the patterns together.

Parameters

out	*pattern*	This is the pointer to a `pattern_desc *` returned by a precedent hl_initPattern() call. This `pattern_desc` should not be already running. If you want many instances of the same pattern, you need to follow all the steps to initialize it.

Returns

uint8_t The result is returned to indicate success or error.

Return values

0	Success
1	Error

Definition at line 364 of file hapticLib.c.

10.2.4.5 uint8_t hl_stopPattern (pattern_desc * *pattern*)

API exposed to the user to force pattern execution stop.

Normally, a well designed pattern will finish its job and then free its instances (the pattern itself and all the haptors used) to be eventually configured later.

There are cases however, when you cannot wait for the pattern to stop.

For example, imagine a continuous pattern that will run until the user feel the feedback, in this case, the pattern must be stopped by the application code (the condition cannot be coded inside the pattern) calling hl_stopPattern() with the right pattern descriptor.

Parameters

out	*pattern*	This is the pattern descriptor reference of a running pattern.

Returns

uint8_t The result is returned to indicate success or error.

Return values

0	Success
1	Error

Definition at line 412 of file hapticLib.c.

10.3 HapticLib/hl_debug.c File Reference

Debugging Features definitions.

```
#include "hl_debug.h"
```

Macros

- #define HL_DEBUG

  *Define this symbol to **enable the Debugging Features**.*

Functions

- uint32_t send_int (uint32_t val)

  DEBUG function *to print over the USART any uint32_t.*
- void send_string (char *string)

  DEBUG function *to print over the USART any string.*

10.3.1 Detailed Description

Debugging Features definitions. This file is part of the **HapticLib Debugging Features** module.

Other HapticLib internal modules use the tools implemented including the file hl_debug.h.

Note

To enable the compilation of this code, the symbol HL_DEBUG must be defined, usually passed to the compiler command line as: `-DHL_DEBUG`

The features implemented are:

- send_string()
- send_int()

Note

User applications can use these functions in their code without directly include hl_debug.h.

Author

Leonardo Guardati

Version

0.7

Date

2012

Definition in file hl_debug.c.

10.3.2 Macro Definition Documentation

10.3.2.1 #define HL_DEBUG

Define this symbol to **enable the Debugging Features**.

The Debugging features offered by the library are:

- send_string()
- send_int()

The HapticLib Debugging Features are enabled based on the presence of the symbol HL_DEBUG .

To define the symbol do either:

- explicitly #`define HL_DEBUG` inside a module.
- add `-DHL_DEBUG` to the compiler command line during compilation.

Warning

Always make calls to debugging code inside conditionally included blocks.

```
...
#ifdef HL_DEBUG
        ...
        send_int( 1234567 );
        ...
        send_string("Correct usage.\n\r\0");
        ...
#endif
...
```

Definition at line 80 of file hl_debug.c.

10.3.3 Function Documentation

10.3.3.1 uint32_t send_int (uint32_t *val*)

DEBUG function to print over the USART any *uint32_t*.

Warning

Only if the HL_DEBUG symbol has been defined, the user can call this function.

send_int() will send the **hexadecimal representation** of any *uint32_t*.

Warning

Always make calls to send_int() inside conditionally included blocks.

```
#ifdef HL_DEBUG
    send_int( 1234567 );
#endif
```

If the value passed is less then 100000, send_int() will also send the **decimal representation** inside brackets.

```
...
#ifdef HL_DEBUG
    ...
    send_int( 12345 );  // outputs &ndash;>   0x3039 (12345)
    ...
#endif
...
```

Note

Implementing this function, dependency on other libraries is avoided.

Parameters

in	*val*	`uint32_t` variable containing the value to print.

Returns

The sent value is returned.

Definition at line 126 of file hl_debug.c.

10.3.3.2 void send_string (char * *string*)

DEBUG function to print over the USART any string.

send_string() will send over the serial link any NULL terminated string (`char *`) smaller than 255 characters.

Warning

Only if the HL_DEBUG symbol has been defined, the user can call this function.
Always make calls to send_string() inside conditionally included blocks.
Also REMEMBER TO APPEND THE NULL TERMINATING CHARACTER TO THE STRINGS PASSED!!!

```
#ifdef HL_DEBUG
     send_string("Correct usage.\n\r\0");
#endif
```

Note

The 255 character limit is arbitrary and only for safety in case you forget to append the '\0' character. If you need to send more, call again send_string().

Parameters

out	*string*	`char*` holding the reference to the string to be printed.

Definition at line 226 of file hl_debug.c.

10.4 HapticLib/hl_debug.h File Reference

Debugging Features header.

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

Macros

- #define HL_DEBUG

  *Define this symbol to **enable the Debugging Features**.*

Functions

- uint8_t send_char (uint8_t ch)
- uint32_t send_int (uint32_t)

  ***DEBUG function** to print over the USART any uint32_t.*
- void send_string (char *)

  ***DEBUG function** to print over the USART any string.*

10.4.1 Detailed Description

Debugging Features header. This file is part of the **HapticLib Debugging Features** module.

It is included by other HapticLib internal modules to use the tools implemented.

Note

To enable the compilation of this code, the symbol HL_DEBUG must be defined, usually passed to the compiler command line as: `-DHL_DEBUG`

The features implemented are:

- send_string()
- send_int()

Note

User applications can use these functions in their code without directly include hl_debug.h.

Author

Leonardo Guardati

Version

0.7

Date

2012

Definition in file hl_debug.h.

10.4.2 Macro Definition Documentation

10.4.2.1 #define HL_DEBUG

Define this symbol to **enable the Debugging Features**.

The Debugging features offered by the library are:

- send_string()
- send_int()

The HapticLib Debugging Features are enabled based on the presence of the symbol HL_DEBUG .

To define the symbol do either:

- explicitly `#define HL_DEBUG` inside a module.
- add `-DHL_DEBUG` to the compiler command line during compilation.

Warning

Always make calls to debugging code inside conditionally included blocks.

```
...
#ifdef HL_DEBUG
        ...
        send_int( 1234567 );
        ...
        send_string("Correct usage.\n\r\0");
        ...
#endif
...
```

Definition at line 90 of file hl_debug.h.

10.4.3 Function Documentation

10.4.3.1 uint8_t send_char (uint8_t *ch*)

10.4.3.2 uint32_t send_int (uint32_t *val*)

DEBUG function to print over the USART any *uint32_t*.

Warning

Only if the HL_DEBUG symbol has been defined, the user can call this function.

send_int() will send the **hexadecimal representation** of any *uint32_t*.

Warning

Always make calls to send_int() inside conditionally included blocks.

```
#ifdef HL_DEBUG
    send_int( 1234567 );
#endif
```

If the value passed is less then 100000, send_int() will also send the **decimal representation** inside brackets.

```
...
#ifdef HL_DEBUG
    ...
    send_int( 12345 );  // outputs &ndash;>   0x3039 (12345)
    ...
#endif
...
```

Note

Implementing this function, dependency on other libraries is avoided.

Parameters

in	*val*	`uint32_t` variable containing the value to print.

Returns

The sent value is returned.

Definition at line 126 of file hl_debug.c.

10.4.3.3 void send_string (char * *string*)

DEBUG function to print over the USART any string.

send_string() will send over the serial link any NULL terminated string (`char *`) smaller than 255 characters.

Warning

Only if the HL_DEBUG symbol has been defined, the user can call this function.
Always make calls to send_string() inside conditionally included blocks.
Also REMEMBER TO APPEND THE NULL TERMINATING CHARACTER TO THE STRINGS PASSED!!!

```
#ifdef HL_DEBUG
     send_string("Correct usage.\n\r\0");
#endif
```

Note

The 255 character limit is arbitrary and only for safety in case you forget to append the '\0' character. If you need to send more, call again send_string().

Parameters

out	*string*	`char*` holding the reference to the string to be printed.

Definition at line 226 of file hl_debug.c.

10.5 HapticLib/patterns/constant/constant.c File Reference

Constant Pattern generator function definition

```
#include "hapticLib.h"
```

Functions

- uint8_t constantContinuator (pattern_desc *pattern)

  *Pattern generator **continuator***
- uint8_t constantPatternGenerator (pattern_desc *pattern)

  *Pattern generator **initiator***

10.5.1 Detailed Description

Constant Pattern generator function definition

Introduction

Constant Pattern can be used to shake haptors providing the shaking magnitude.

Theory

Simply a on/off pattern.

Link to optional tools used on pattern development.

No other tools needed.

Optional Parameters

User must provide a constant value from 0 to 65000:

- 1 to 65000 means set an haptor in busy mode and shake it at provided value.

Usage Examples

```
...
//start one haptor shaking @ 35000
haptor_desc *myHaptor = hl_configure(24000000,15,1);
```

```
user_param myParams; // param must use struct of user_param_t type
myParams.constant.constant = 35000;

pattern_desc *myConstantPattern = hl_initPattern(Constant,&myParams);

hl_addHaptor(myHaptor,myConstantPattern);

hl_startPattern(myConstantPattern);
...
hl_stopPattern(myConstantPattern);
...
```

Debugging Details

Parameters can't be NULL. It NULL is passed return an error. If HL_DEBUG symbol defined a "NULL found" string will sent to UART.

Additional Notes

The `constant` parameter can be used to change the duty-cycle even after the pattern has started! At ever continuator pass, it will poll the actual value of the user parameter `constant` and update the duty-cycle accordingly.

Warning

When use Constant Pattern remember to free an haptor that was previously started using this pattern.

Note

To stop this pattern and free its active haptor, use hl_stopPattern() with the right pattern descriptor passed as argument.

Authors

Leonardo Guardati
Silvio Vallorani

Version

v0.7

Date

2012

Definition in file constant.c.

10.5.2 Function Documentation

10.5.2.1 uint8_t constantContinuator (pattern_desc * *pattern*)

Pattern generator **continuator**

This function is called back from the patternScheduler() every SystemDesc.samples_delay ms.

At every call, this continuator will decide what to do.

For a simple and static pattern, the action could be to simply point to the next element of an array of samples, or to generate it on the fly through a certain formula.

For more complex scenarios, this pattern continuator callback should be able to deliver a good degree of flexibility.

Warning

The **continuator** must implement an exit strategy to decide when the pattern is finished an remove itself from the scheduling patterns. If this is not done right, the pattern will never free the haptor to other patterns.

Parameters

out	*pattern*	A pattern_desc* holding the reference to the running pattern instance to work with. Through this reference, the continuator can extract the user and status parameters in order to execute the haptic pattern.

Returns

uint8_t The result is returned to indicate success or error.

Return values

0	Success
1	Error

Definition at line 127 of file constant.c.

10.5.2.2 uint8_t constantPatternGenerator (pattern_desc * *pattern*)

Pattern generator **initiator**

This function is called by the **User API Module** of HapticLib, when the library user calls hl_startPattern().

The first task for the pattern generator **initiator** is to validate the user provided inputs, this cannot be done anywhere else.

This pattern initiator then sets up all the status parameters.

If the pattern accept user parameters, the status setting can be based on the user provided parameters.

The last duty of this function is to put the pattern continuator up for scheduling (setting `pattern->continuator =&constantContinuator;`), so the next time patternScheduler() will check for active patterns, it will find this pattern.

Warning

If the **initiator** doesn't set the continuator up for scheduling, the pattern will never start, and for HapticLib, the haptor is free to receive other patterns.

Parameters

out	*pattern*	A pattern_desc* holding the running instance of the pattern being started.

Returns

uint8_t The result is returned to indicate success or error.

Return values

0	Success
1	Error

Definition at line 184 of file constant.c.

10.6 HapticLib/patterns/constant/constant.h File Reference

Constant Pattern generator function header

Data Structures

- struct constantUserParameters

  *Pattern specific **user** parameter type definition.*
- struct constantStatusParameters

  *Pattern specific **status** parameters type definition.*

10.6.1 Detailed Description

Constant Pattern generator function header Here the constantPatternGenerator is defined

Author

Silvio Vallorani

Version

v0.7

Date

2012

Definition in file constant.h.

10.7 HapticLib/patterns/generic/generic.c File Reference

Generic Pattern generator function definition [TEMPLATE]

```
#include "hapticLib.h"
```

Macros

- #define HL_DEBUG

  *Generator function **Debugging Capabilities**.*

Functions

- uint8_t genericContinuator (pattern_desc *pattern)

  *Pattern generator **continuator***
- uint8_t genericPatternGenerator (pattern_desc *pattern)

  *Pattern generator **initiator***

10.7.1 Detailed Description

Generic Pattern generator function definition [TEMPLATE] [TEMPLATE]

This is a template file; copy and rename it as starting point for the implementation of a new Pattern Generator.

Follow the structure of this example pattern (code and documentation formats) while implementing a new pattern generator.

The **initiator** and the **continuator** need to be implemented and documented. First read the documentation and the code of both and the implement your own pattern code and document it.

Note

Remember to use the associated header file too. For the template is:

- generic.c: Documentation, initiator and continuator.
- generic.h: Pattern specific definitions, user and status parameters definitions

For your pattern they become:

- `mypattern.c`: Documentation, initiator and continuator.
- `mypattern.h`: Pattern specific definitions, user and status parameters definitions

[TEMPLATE]

Implementation code of the Generic Pattern.

Introduction

Description with main features only mentioned.

Theory

Reference to theoretical documentation on which the pattern is based.

Link to optional tools used on pattern development.

Description of main results.

Optional Parameters

Detailed description of parameters needed and structure defined to contain them.

Parameters values ranges MUST be explained here.

Usage Examples

Put some code snippets to illustrate how to use the pattern.

Debugging Details

Explain (if any) the behavior of optional debugging features present inside the generator.

Additional Notes

Add some optional notes.

Authors

Leonardo Guardati
Silvio Vallorani

Version

v0.7

Date

2012

Definition in file generic.c.

10.7.2 Macro Definition Documentation

10.7.2.1 #define HL_DEBUG

Generator function **Debugging Capabilities**.

The HapticLib Debugging Features are available on the pattern generator function too.

The Debugging Features are enabled based on the presence of the symbol HL_DEBUG.
The code must have conditional inclusion of debugging code.

```
...
#ifdef HL_DEBUG
        ...
        send_string("Debug Message...\n\r\0");
        ...
#endif
...
```

Warning

Unconditionally use of debugging code will result in compiling errors when linking the release version of the object files (compiled without -DHL_DEBUG passed to the compiler).

Definition at line 129 of file generic.c.

10.7.3 Function Documentation

10.7.3.1 uint8_t genericContinuator (pattern_desc * *pattern*)

Pattern generator **continuator**

This function is called back from the patternScheduler() every SystemDesc.samples_delay ms.

At every call, this continuator will decide what to do.

For a simple and static pattern, the action could be to simply point to the next element of an array of samples, or to generate it on the fly through a certain formula.

For more complex scenarios, this pattern continuator callback should be able to deliver a good degree of flexibility.

Warning

The **continuator** MUST implement an exit strategy to decide when the pattern is finished an remove itself from the scheduling patterns. If this is not done right, the pattern will never free the haptor(s) for the other patterns to use it(them) again.

Parameters

out	*pattern*	A pattern_desc* holding the reference to the running pattern instance to work with. Through this reference, the continuator can extract the user and status parameters in order to execute the haptic pattern.

Returns

uint8_t The result is returned to indicate success or error.

Return values

0	Success
1	Error

Definition at line 170 of file generic.c.

10.7.3.2 uint8_t genericPatternGenerator (pattern_desc * *pattern*)

Pattern generator **initiator**

This function is called by the **User API Module** of HapticLib, when the library user calls hl_startPattern().

The first task for the pattern generator **initiator** is to validate the user provided inputs, this cannot be done anywhere else.

The pattern initiator then sets up all the status parameters.

If the pattern accept user parameters, the status setting can be based on the user provided parameters.

The last duty of this function is to put the pattern continuator up for scheduling (setting `pattern->continuator =&genericContinuator;`), so the next time patternScheduler() will check for active patterns, it will find this pattern.

Warning

If the **initiator** doesn't set the continuator up for scheduling, the pattern will never start, and for HapticLib, the haptor is free to receive other patterns.

Parameters

out	*pattern*	A pattern_desc* holding the running instance of the pattern being started.

Returns

uint8_t The result is returned to indicate success or error.

Return values

0	Success
1	Error

Definition at line 272 of file generic.c.

10.8 HapticLib/patterns/generic/generic.h File Reference

Generic Pattern generator function header [TEMPLATE]

Data Structures

- struct genericUserParameters

  *Pattern specific **user** parameters type definition.*
- struct genericStatusParameters

  *Pattern specific **status** parameters type definition.*

Typedefs

- typedef enum genericIncrement genericIncrement

  *User Parameter Values for **increment** user parameter.*
- typedef enum genericCheckParam genericCheckParam

  *User Parameter Values for **checkParam** user parameter.*
- typedef struct genericUserParameters genericUserParameters

  *Pattern specific **user** parameters type definition.*
- typedef struct genericStatusParameters genericStatusParameters

  *Pattern specific **status** parameters type definition.*

Enumerations

- enum genericIncrement { smallIncrement = 0, bigIncrement = 1 }

  *User Parameter Values for **increment** user parameter.*
- enum genericCheckParam { rightValue = 300000 }

  *User Parameter Values for **checkParam** user parameter.*

10.8.1 Detailed Description

Generic Pattern generator function header [TEMPLATE] [TEMPLATE]

This is a template file; copy and rename it as starting point for the implementation of a new Pattern Generator.

Follow the structure of this example pattern (code and documentation formats) while implementing a new pattern generator.

Pattern specific **constant definitions** can be inserted here.

The pattern specific **status parameters** typedef is defined here.

The pattern specific **user parameters** typedef is defined here.

Note

Neither of the two (status or user) typedef are formally necessary, a super simple pattern generator could even be implemented without the status parameters (but then it could be only externally stopped calling hl_stopPattern()).

Please use at least a status Parameter where the pattern progress is stored, to let the **continuator** keep track of itself and decide when to end the pattern and free the haptor(s).

Note

Remember to use the associated module file too. For the template is:

- generic.c: Documentation, initiator and continuator.
- generic.h: Pattern specific definitions, user and status parameters definitions.

For your pattern they become:

- `mypattern.c`: Documentation, initiator and continuator.
- `mypattern.h`: Pattern specific definitions, user and status parameters definitions

[TEMPLATE]

Pattern specific **constant definitions** can be inserted here.

The pattern specific **status parameters** typedef is defined here.

The pattern specific **user parameters** typedef is defined here.

Authors

Leonardo Guardati
Silvio Vallorani

Version

v0.7

Date

2012

Definition in file generic.h.

10.8.2 Typedef Documentation

10.8.2.1 typedef enum genericCheckParam genericCheckParam

User Parameter Values for **checkParam** user parameter.

This example show how to define the valid values for a user parameter. The name convention is as follow:

<pattern_name><parameter_name>

For example, the **checkParam** parameter for the **Generic** pattern accepts only one value:

- rightValue: needed to start the pattern.

This is the enumeration used to hold this value.

10.8.2.2 typedef enum genericIncrement genericIncrement

User Parameter Values for **increment** user parameter.

This example show how to define the valid values for a user parameter. The name convention is as follow:

<pattern_name><parameter_name>

For example, the **increment** parameter for the **Generic** pattern accepts only two values:

- smallIncrement: for slow ramps.
- bigIncrements: for fast ramps.

This is the enumeration used to hold these two values.

10.8.2.3 typedef struct genericStatusParameters genericStatusParameters

Pattern specific **status** parameters type definition.

At least one parameter should be defined to help the **continuator** keep track of the pattern generation progress.

See also

Refer to generic.c for a simple working example.

10.8.2.4 typedef struct genericUserParameters genericUserParameters

Pattern specific **user** parameters type definition.

This structure MUST be clearly documented to avoid pitfalls on the generator's use by the application code.

The user provided values, must be validated by the (preferably) pattern **initiator** before using them in the pattern logic.

10.8.3 Enumeration Type Documentation

10.8.3.1 enum genericCheckParam

User Parameter Values for **checkParam** user parameter.

This example show how to define the valid values for a user parameter. The name convention is as follow:

<pattern_name><parameter_name>

For example, the **checkParam** parameter for the **Generic** pattern accepts only one value:

- rightValue: needed to start the pattern.

This is the enumeration used to hold this value.

Enumerator:

rightValue

Definition at line 113 of file generic.h.

10.8.3.2 enum genericIncrement

User Parameter Values for **increment** user parameter.

This example show how to define the valid values for a user parameter. The name convention is as follow:

<pattern_name><parameter_name>

For example, the **increment** parameter for the **Generic** pattern accepts only two values:

- smallIncrement: for slow ramps.
- bigIncrements: for fast ramps.

This is the enumeration used to hold these two values.

Enumerator:

smallIncrement

bigIncrement

Definition at line 93 of file generic.h.

10.9 HapticLib/patterns/hl_patterns.c File Reference

Pattern Generator Module definitions

```
#include "hapticLib.h"
```

Macros

- #define HL_SYSTEM_FILE

  Used to indicate this module is a System one.

Functions

- uint8_t genericPatternGenerator (pattern_desc ∗)

  *Pattern generator **initiator***
- uint8_t testPatternGenerator (pattern_desc ∗)

  *Test Pattern generator **initiator***
- uint8_t impactPatternGenerator (pattern_desc ∗)

  *Pattern generator **initiator***
- uint8_t constantPatternGenerator (pattern_desc ∗)

  *Pattern generator **initiator***
- void patternScheduler (void)

  Updates all the active patterns' progress.
- void cleanList (haptor_desc ∗haptor)

  Utility function to clean the pattern list of haptors.
- uint16_t dutyConverter (uint16_t duty, haptor_desc ∗haptor)

  Utility function to convert duty-cycles.

Variables

- system_desc SystemDesc

  Global variable to describe the haptic system.
- pattern_initiator patternMap [Num_Patterns_Available]

  Pattern Generators Functions Map.

10.9.1 Detailed Description

Pattern Generator Module definitions hl_patterns.c is part of the **Pattern Generator Module** of HapticLib.

This file lists of all the pattern generators initiators, and uses them to fill the patternMap array.

The technique used by HapticLib make it possible to decouple the **User API Module** and the **Pattern Generator Module**.

See also

Please refer to the HapticLib Developer documentation page.

The **User API Module** only knows the signature of a generic pattern generator function (that is always the same) regardless of what the actual function will be.

Authors

Leonardo Guardati
Silvio Vallorani

Version

v0.7

Date

2012

Definition in file hl_patterns.c.

10.9.2 Macro Definition Documentation

10.9.2.1 #define HL_SYSTEM_FILE

Used to indicate this module is a System one.

Declaring this symbol, some internal data structures will be available to the module.

Definition at line 57 of file hl_patterns.c.

10.9.3 Function Documentation

10.9.3.1 void cleanList (haptor_desc * *haptor*)

Utility function to clean the pattern list of haptors.

This function holds the frequently used code to free the haptor list form the attached pattern.

Parameters

out	*haptor*	The `haptor_desc *` hold by activeHaptorList inside `pattern_desc` structure.

Definition at line 131 of file hl_patterns.c.

10.9.3.2 uint8_t constantPatternGenerator (pattern_desc * *pattern*)

Pattern generator **initiator**

This function is called by the **User API Module** of HapticLib, when the library user calls hl_startPattern().

The first task for the pattern generator **initiator** is to validate the user provided inputs, this cannot be done anywhere else.

This pattern initiator then sets up all the status parameters.

If the pattern accept user parameters, the status setting can be based on the user provided parameters.

The last duty of this function is to put the pattern continuator up for scheduling (setting `pattern->continuator =&constantContinuator;`), so the next time patternScheduler() will check for active patterns, it will find this pattern.

Warning

If the **initiator** doesn't set the continuator up for scheduling, the pattern will never start, and for HapticLib, the haptor is free to receive other patterns.

Parameters

out	*pattern*	A pattern_desc* holding the running instance of the pattern being started.

Returns

uint8_t The result is returned to indicate success or error.

Return values

0	Success
1	Error

Definition at line 184 of file constant.c.

10.9.3.3 uint16_t dutyConverter (uint16_t *duty,* haptor_desc * *haptor*)

Utility function to convert duty-cycles.

Given the absolute duty-cycle (0-65535) and the haptor this function will return the actual duty cycle honoring min_duty and max_duty values of the haptor.

Parameters

in	*duty*	A `uint16_t` value representing the absolute duty cycle.
out	*haptor*	A `haptor_desc *` holding the reference to the haptor for which the duty has to be scaled.

Returns

uint16_t the scaled value of the duty cycle.

Definition at line 163 of file hl_patterns.c.

10.9.3.4 uint8_t genericPatternGenerator (pattern_desc * *pattern*)

Pattern generator **initiator**

This function is called by the **User API Module** of HapticLib, when the library user calls hl_startPattern().

The first task for the pattern generator **initiator** is to validate the user provided inputs, this cannot be done anywhere else.

The pattern initiator then sets up all the status parameters.

If the pattern accept user parameters, the status setting can be based on the user provided parameters.

The last duty of this function is to put the pattern continuator up for scheduling (setting `pattern->continuator =&genericContinuator;`), so the next time patternScheduler() will check for active patterns, it will find this pattern.

Warning

If the **initiator** doesn't set the continuator up for scheduling, the pattern will never start, and for HapticLib, the haptor is free to receive other patterns.

Parameters

out	*pattern*	A pattern_desc* holding the running instance of the pattern being started.

Returns

uint8_t The result is returned to indicate success or error.

Return values

0	Success
1	Error

Definition at line 272 of file generic.c.

10.9.3.5 uint8_t impactPatternGenerator (pattern_desc * *pattern*)

Pattern generator **initiator**

This function is called by the **User API Module** of HapticLib, when the library user calls hl_startPattern().

The first task for the pattern generator **initiator** is to validate the user provided inputs, this cannot be done anywhere else.

This pattern initiator then sets up all the status parameters.

If the pattern accept user parameters, the status setting can be based on the user provided parameters.

The last duty of this function is to put the pattern continuator up for scheduling (setting `pattern->continuator=&impactContinuator;`), so the next time patternScheduler() will check for active patterns, it will find this pattern.

Warning

If the **initiator** doesn't set the continuator up for scheduling, the pattern will never start, and for HapticLib, the haptor is free to receive other patterns.

Parameters

out	*pattern*	A pattern_desc* holding the reference to the running pattern instance to work with. Through this reference, the continuator can extract the user and status parameters in order to execute the haptic pattern.

Returns

uint8_t The result is returned to indicate success or error.

Return values

0	Success
1	Error

Definition at line 431 of file impact.c.

10.9.3.6 void patternScheduler (void)

Updates all the active patterns' progress.

patternScheduler() checks for all the patterns configured in the system. If the pattern is running, its continuator is

called.

patternScheduler() is called by the **Platform Specific** Module (for example in the STM32VLDISCOVERY platform, SysTick_Handler() is the caller).

See also

Please refer to the Developer Guide page to know the internal details of how HapticLib multi-haptor technique works.

Definition at line 100 of file hl_patterns.c.

10.9.3.7 uint8_t testPatternGenerator (pattern_desc * *pattern*)

Test Pattern generator **initiator**

This function is called by the **User API Module** of HapticLib, when the library user calls hl_sendPattern().

The first task for the pattern generator **initiator** is to validate the user provided inputs, this cannot be done anywhere else.

This pattern initiator then sets up all the status parameters.

If the pattern accept user parameters, the status setting can be based on the user provided parameters.

The last duty of this function is to put the pattern continuator up for scheduling (setting `pattern->continuator =&testContinuator;`), so the next time patternScheduler() will check for active patterns, it will find this pattern.

Warning

If the **initiator** doesn't set the continuator up for scheduling, the pattern will never start, and for HapticLib, the haptor is free to receive other patterns.

Note

This pattern doesn't need user parameters, and check for this parameter to be NULL.

Parameters

`out`	*pattern*	A pattern_desc* holding the reference to the running pattern instance to work with. Through this reference, the continuator can extract the user and status parameters in order to execute the haptic pattern.

Returns

uint8_t The result is returned to indicate success or error.

Return values

0	Success
1	Error

Definition at line 189 of file test.c.

10.9.4 Variable Documentation

10.9.4.1 pattern_initiator patternMap[Num_Patterns_Available]

Initial value:

```
                {   NULL,
                        genericPatternGenerator,
                        testPatternGenerator,
                        impactPatternGenerator,
                        constantPatternGenerator
        }
```

Pattern Generators Functions Map.

This array contains the addresses of all the pattern generator initiator functions.

The index of the array locate the specific pattern using one of the Pattern Generator Functions Index Names defined in the pattern_name enumeration type definition. (e.g. Test, Impact)

Definition at line 79 of file hl_patterns.c.

10.9.4.2 system_desc SystemDesc

Global variable to describe the haptic system.

The Library uses this global variable to describe the system and to keep track of the haptic devices' status and active patterns' progress at any time.

Note

The library user doesn't need to access this variable.

The SystemDesc variable holds a lot of informations, and its members allow the system to access almost all the informations needed.

See also

Please refer to the Developer Guide page for details.

Definition at line 70 of file hapticLib.c.

10.10 HapticLib/patterns/hl_patterns.h File Reference

Pattern Generator Module headers

```
#include <stdint.h>
#include <stdlib.h>
#include "test.h"
#include "generic.h"
#include "impact.h"
#include "constant.h"
```

Data Structures

- union status_param

  Pattern specific status parameters container.
- union user_param

  Pattern specific optional user parameters container.
- struct pattern_desc

  Pattern descriptor.

Macros

- #define MAX_PATTERNS 10

  Number of patterns known in the System.

Typedefs

- typedef enum pattern_name pattern_name

  Index of pattern names.
- typedef union status_param status_param

  Pattern specific status parameters container.
- typedef union user_param user_param

  Pattern specific optional user parameters container.
- typedef uint8_t(∗ pattern_continuator)(struct pattern_desc ∗)

  *Pattern Generator **Continuator** callback function pointer.*
- typedef struct pattern_desc pattern_desc

  Pattern descriptor.
- typedef uint8_t(∗ pattern_initiator)(pattern_desc ∗)

  *Pattern Generation **Initiator** callback function pointer.*

Enumerations

- enum pattern_name {
  Null, Generic, Test, Impact,
  Constant, Num_Patterns_Available }

  Index of pattern names.

Functions

- void patternScheduler (void)

  Updates all the active patterns' progress.
- void cleanList (struct haptor_desc ∗)

  Utility function to clean the pattern list of haptors.
- uint16_t dutyConverter (uint16_t, struct haptor_desc ∗)

  Utility function to convert duty-cycles.

10.10.1 Detailed Description

Pattern Generator Module headers hl_patterns.h is part of the **Pattern Generator Module** of HapticLib.

In this file there are all the general definitions needed by HapticLib to call a pattern (the code specific to a particular pattern is in its pattern code module).

Todo Define Return values constants for pattern generators.

Authors

Leonardo Guardati
Silvio Vallorani

Version

0.7

Date

2012

Definition in file hl_patterns.h.

10.10.2 Macro Definition Documentation

10.10.2.1 #define MAX_PATTERNS 10

Number of patterns known in the System.

This is the number of different pattern offered by the library.

There could be use cases where System resources (low memory) impose to lower this number; the result is that only the first MAX_PATTERNS patterns are available to the user.

Note

MAX_PATTERNS does not refer to the maximum number of pattern running simultaneously, instead is the number of them that can be loaded in a running application and then called. (MAX_HAPTORS is the maximum number of running pattern instances).

Todo Implement a re-mapping mechanism to easily allow the user the selection of some specific patterns discarding the others.

Definition at line 84 of file hl_patterns.h.

10.10.3 Typedef Documentation

10.10.3.1 typedef uint8_t(* pattern_continuator)(struct pattern_desc *)

Pattern Generator **Continuator** callback function pointer.

Before reading about this function pointer, please read HapticLib Architecture and Developer documentation pages.

This pointer is used by HapticLib code to transparently call the pattern continuator code of the right pattern.

A running pattern instance is active if its descriptor (pattern_desc) holds a valid `pattern_continuator` reference.

When hl_startPattern() is called by the user application, the right pattern initiator is called. The initiator, after some validation checks, sets the continuator callback function pointer for the passed pattern_desc.

Definition at line 202 of file hl_patterns.h.

10.10.3.2 typedef struct pattern_desc pattern_desc

Pattern descriptor.

The pattern descriptor holds all the relevant information about a pattern being run on a specific haptor.

10.10.3.3 typedef uint8_t(* pattern_initiator)(pattern_desc *)

Pattern Generation **Initiator** callback function pointer.

Before reading about this function pointer, please read HapticLib Architecture and Developer documentation pages.

This pointer is used by HapticLib code to transparently call the pattern initiator code of the right pattern.

HapticLib declares patternMap ; it is an array of `pattern_initiator`s. The index of this array is a pattern_name element. Each element points to the right Pattern Generator's Initiator callback function.

Definition at line 251 of file hl_patterns.h.

10.10.3.4 typedef enum pattern_name pattern_name

Index of pattern names.

This enumeration makes it easy to index arrays with the name of a particular pattern.

Previous versions of HapticLib used #`define` directives for this purpose.

The advantage of using an enumeration type definition is code and debug readability.

10.10.3.5 typedef union status_param status_param

Pattern specific status parameters container.

This type definition is useful to increase the readability of the code and to enforce compiler type checking when passing arguments to functions.

The status parameters are intended only for the pattern developer. These parameters are not exposed outside the pattern module.

See also

Please refer to the generic Pattern Template Generator (generic.c) for usage example of the status parameters.

10.10.3.6 typedef union user_param user_param

Pattern specific optional user parameters container.

This type definition is useful to increase the readability of the code and to enforce compiler type checking when passing parameters to patterns.

Usage example of the Impact Pattern:

```
...
haptor_desc *myHaptor = hl_configure(24000000,10,1);

pattern_desc *myImpactPattern;

user_param myParams;

myParams.impact.velocity = Fast;
myParams.impact.material = Rubber;

myImpactPattern = hl_initPattern(Impact,&myParams);

hl_addHaptor(myHaptor,myImpactPattern);

hl_startPattern(myImpactPattern);
...
```

10.10.4 Enumeration Type Documentation

10.10.4.1 enum pattern_name

Index of pattern names.

This enumeration makes it easy to index arrays with the name of a particular pattern.

Previous versions of HapticLib used #define directives for this purpose.

The advantage of using an enumeration type definition is code and debug readability.

Enumerator:

Null

Generic

Test

Impact

Constant

Num_Patterns_Available

Definition at line 99 of file hl_patterns.h.

10.10.5 Function Documentation

10.10.5.1 void cleanList (haptor_desc * *haptor*)

Utility function to clean the pattern list of haptors.

This function holds the frequently used code to free the haptor list form the attached pattern.

Parameters

out	*haptor*	The haptor_desc * hold by activeHaptorList inside pattern_desc structure.

Definition at line 131 of file hl_patterns.c.

10.10.5.2 uint16_t dutyConverter (uint16_t *duty*, haptor_desc * *haptor*)

Utility function to convert duty-cycles.

Given the absolute duty-cycle (0-65535) and the haptor this function will return the actual duty cycle honoring min_duty and max_duty values of the haptor.

Parameters

in	*duty*	A uint16_t value representing the absolute duty cycle.
out	*haptor*	A haptor_desc * holding the reference to the haptor for which the duty has to be scaled.

Returns

uint16_t the scaled value of the duty cycle.

Definition at line 163 of file hl_patterns.c.

10.10.5.3 void patternScheduler (void)

Updates all the active patterns' progress.

patternScheduler() checks for all the patterns configured in the system. If the pattern is running, its continuator is called.

patternScheduler() is called by the **Platform Specific** Module (for example in the STM32VLDISCOVERY platform, SysTick_Handler() is the caller).

See also

Please refer to the Developer Guide page to know the internal details of how HapticLib multi-haptor technique works.

Definition at line 100 of file hl_patterns.c.

10.11 HapticLib/patterns/impact/extra/impact.m File Reference

Functions

- Copyright (c) 2012

Variables

- Silvio Vallorani
  < silvio.vallorani
  @studio.unibo.it > Permission
  to use
- Silvio Vallorani
  < silvio.vallorani
  @studio.unibo.it > Permission
  to copy
- Silvio Vallorani
  < silvio.vallorani
  @studio.unibo.it > Permission
  to modify
- Silvio Vallorani
  < silvio.vallorani
  @studio.unibo.it > Permission
  to and or distribute this
  software for any purpose with
  or without fee is hereby granted
- Silvio Vallorani
  < silvio.vallorani
  @studio.unibo.it > Permission
  to and or distribute this
  software for any purpose with
  or without fee is hereby
  provided that the above
  copyright notice and this
  permission notice appear in
  all copies THE SOFTWARE IS
  PROVIDED AS IS AND THE AUTHOR
  DISCLAIMS ALL WARRANTIES WITH
  REGARD TO THIS SOFTWARE
  INCLUDING ALL IMPLIED
  WARRANTIES OF MERCHANTABILITY
  AND FITNESS IN NO EVENT SHALL
  THE AUTHOR BE LIABLE FOR ANY SPECIAL

- Silvio Vallorani
  < silvio.vallorani
  @studio.unibo.it > Permission
  to and or distribute this
  software for any purpose with
  or without fee is hereby
  provided that the above
  copyright notice and this
  permission notice appear in
  all copies THE SOFTWARE IS
  PROVIDED AS IS AND THE AUTHOR
  DISCLAIMS ALL WARRANTIES WITH
  REGARD TO THIS SOFTWARE
  INCLUDING ALL IMPLIED
  WARRANTIES OF MERCHANTABILITY
  AND FITNESS IN NO EVENT SHALL
  THE AUTHOR BE LIABLE FOR ANY DIRECT

- Silvio Vallorani
  < silvio.vallorani
  @studio.unibo.it > Permission
  to and or distribute this
  software for any purpose with
  or without fee is hereby
  provided that the above
  copyright notice and this
  permission notice appear in
  all copies THE SOFTWARE IS
  PROVIDED AS IS AND THE AUTHOR
  DISCLAIMS ALL WARRANTIES WITH
  REGARD TO THIS SOFTWARE
  INCLUDING ALL IMPLIED
  WARRANTIES OF MERCHANTABILITY
  AND FITNESS IN NO EVENT SHALL
  THE AUTHOR BE LIABLE FOR ANY INDIRECT

- Silvio Vallorani
  < silvio.vallorani
  @studio.unibo.it > Permission
  to and or distribute this
  software for any purpose with
  or without fee is hereby
  provided that the above
  copyright notice and this
  permission notice appear in
  all copies THE SOFTWARE IS
  PROVIDED AS IS AND THE AUTHOR
  DISCLAIMS ALL WARRANTIES WITH
  REGARD TO THIS SOFTWARE
  INCLUDING ALL IMPLIED
  WARRANTIES OF MERCHANTABILITY
  AND FITNESS IN NO EVENT SHALL
  THE AUTHOR BE LIABLE FOR ANY
  OR CONSEQUENTIAL DAMAGES OR
  ANY DAMAGES WHATSOEVER
  RESULTING FROM LOSS OF USE

- Silvio Vallorani

< silvio.vallorani
@studio.unibo.it > Permission
to and or distribute this
software for any purpose with
or without fee is hereby
provided that the above
copyright notice and this
permission notice appear in
all copies THE SOFTWARE IS
PROVIDED AS IS AND THE AUTHOR
DISCLAIMS ALL WARRANTIES WITH
REGARD TO THIS SOFTWARE
INCLUDING ALL IMPLIED
WARRANTIES OF MERCHANTABILITY
AND FITNESS IN NO EVENT SHALL
THE AUTHOR BE LIABLE FOR ANY
OR CONSEQUENTIAL DAMAGES OR
ANY DAMAGES WHATSOEVER
RESULTING FROM LOSS OF DATA OR PROFITS

- Silvio Vallorani
  < silvio.vallorani
  @studio.unibo.it > Permission
  to and or distribute this
  software for any purpose with
  or without fee is hereby
  provided that the above
  copyright notice and this
  permission notice appear in
  all copies THE SOFTWARE IS
  PROVIDED AS IS AND THE AUTHOR
  DISCLAIMS ALL WARRANTIES WITH
  REGARD TO THIS SOFTWARE
  INCLUDING ALL IMPLIED
  WARRANTIES OF MERCHANTABILITY
  AND FITNESS IN NO EVENT SHALL
  THE AUTHOR BE LIABLE FOR ANY
  OR CONSEQUENTIAL DAMAGES OR
  ANY DAMAGES WHATSOEVER
  RESULTING FROM LOSS OF DATA OR
  WHETHER IN AN ACTION OF CONTRACT

- Silvio Vallorani

< silvio.vallorani
@studio.unibo.it > Permission
to and or distribute this
software for any purpose with
or without fee is hereby
provided that the above
copyright notice and this
permission notice appear in
all copies THE SOFTWARE IS
PROVIDED AS IS AND THE AUTHOR
DISCLAIMS ALL WARRANTIES WITH
REGARD TO THIS SOFTWARE
INCLUDING ALL IMPLIED
WARRANTIES OF MERCHANTABILITY
AND FITNESS IN NO EVENT SHALL
THE AUTHOR BE LIABLE FOR ANY
OR CONSEQUENTIAL DAMAGES OR
ANY DAMAGES WHATSOEVER
RESULTING FROM LOSS OF DATA OR
WHETHER IN AN ACTION OF
NEGLIGENCE OR OTHER TORTIOUS ACTION

10.11.1 Function Documentation

10.11.1.1 Copyright (c)

10.11.2 Variable Documentation

10.11.2.1 Silvio Vallorani<silvio.vallorani@studio.unibo.it> Permission to and or distribute this software for any purpose with or without fee is hereby provided that the above copyright notice and this permission notice appear in all copies THE SOFTWARE IS PROVIDED AS IS AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF DATA OR WHETHER IN AN ACTION OF NEGLIGENCE OR OTHER TORTIOUS ACTION

Definition at line 4 of file impact.m.

10.11.2.2 Silvio Vallorani<silvio.vallorani@studio.unibo.it> Permission to and or distribute this software for any purpose with or without fee is hereby provided that the above copyright notice and this permission notice appear in all copies THE SOFTWARE IS PROVIDED AS IS AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF DATA OR WHETHER IN AN ACTION OF CONTRACT

Definition at line 4 of file impact.m.

10.11.2.3 Silvio Vallorani<silvio.vallorani@studio.unibo.it> Permission to copy

Definition at line 4 of file impact.m.

10.11.2.4 Silvio Vallorani<silvio.vallorani@studio.unibo.it> Permission to and or distribute this software for any purpose with or without fee is hereby provided that the above copyright notice and this permission notice appear in all copies THE SOFTWARE IS PROVIDED AS IS AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT

Definition at line 4 of file impact.m.

10.11.2.5 Silvio Vallorani<silvio.vallorani@studio.unibo.it> Permission to and or distribute this software for any purpose with or without fee is hereby granted

Definition at line 4 of file impact.m.

10.11.2.6 Silvio Vallorani<silvio.vallorani@studio.unibo.it> Permission to and or distribute this software for any purpose with or without fee is hereby provided that the above copyright notice and this permission notice appear in all copies THE SOFTWARE IS PROVIDED AS IS AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY INDIRECT

Definition at line 4 of file impact.m.

10.11.2.7 Silvio Vallorani<silvio.vallorani@studio.unibo.it> Permission to modify

Definition at line 4 of file impact.m.

10.11.2.8 Silvio Vallorani<silvio.vallorani@studio.unibo.it> Permission to and or distribute this software for any purpose with or without fee is hereby provided that the above copyright notice and this permission notice appear in all copies THE SOFTWARE IS PROVIDED AS IS AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF DATA OR PROFITS

Definition at line 4 of file impact.m.

10.11.2.9 Silvio Vallorani<silvio.vallorani@studio.unibo.it> Permission to and or distribute this software for any purpose with or without fee is hereby provided that the above copyright notice and this permission notice appear in all copies THE SOFTWARE IS PROVIDED AS IS AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL

Definition at line 4 of file impact.m.

10.11.2.10 Silvio Vallorani<silvio.vallorani@studio.unibo.it> Permission to use

Definition at line 4 of file impact.m.

10.11.2.11 Silvio Vallorani<silvio.vallorani@studio.unibo.it> Permission to and or distribute this software for any purpose with or without fee is hereby provided that the above copyright notice and this permission notice appear in all copies THE SOFTWARE IS PROVIDED AS IS AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE

Definition at line 4 of file impact.m.

10.12 HapticLib/patterns/impact/impact.c File Reference

Impact Pattern generator function definition

```
#include "hapticLib.h"
```

Functions

- uint8_t impactContinuator (pattern_desc *pattern)
  *Pattern generator **continuator***
- uint8_t impactPatternGenerator (pattern_desc *pattern)
  *Pattern generator **initiator***

Variables

- const uint16_t rubberImpactPattern [256]
  Hardcoded, Haptic Feedback Pattern of impact on Rubber surface.
- const uint16_t woodImpactPattern [256]
  Hardcoded, Haptic Feedback Pattern of impact on Wood surface.
- const uint16_t aluminumImpactPattern [256]
  Hardcoded, Haptic Feedback Pattern of impact on Aluminum surface.

10.12.1 Detailed Description

Impact Pattern generator function definition

Introduction

This is the **Impact Pattern Generator** function that describe an *impact on surface sensation*.

Users can choose from three different surface materials and three hit velocity magnitude.

Note

Refer to hl_startPattern documentation to learn how parameters are passed through modules.
This pattern now supports HapticLib's multi-haptor feature.

Theory

Impact Pattern is based on theoretical research published on paper

- Hachisu et al - Pseudo-haptic feedback augmented with visual and tactile vibrations

and also used in other researches such as the one published on paper

- Okamura et al - Reality-based models for vibration feedback in virtual environments

in which authors propose a mathematics model that describe vibrations in impact on various surface materials.

Original model proposed was sinusoidal one:

$Q(t) = A(v) \cdot e^{(-B \cdot t)} \cdot \sin(2 \cdot \pi \cdot f \cdot t)$

where:

- v is the impact velocity
- $A(v)$ is the amplitude factor
- B is the decay rate of the sinusoid
- f is the frequency of the sinusoid
- t is the time

the amplitude factor $A(v)$ depends on impact velocity in a quasi-linear way, so the model can be simplified in

$Q(t) = A \cdot v \cdot e^{(-B \cdot t)} \cdot \sin(2 \cdot \pi \cdot f \cdot t)$

To use this formula as Haptic Pattern Samples Generator some other manipulations are necessary:

$PS(t) = |A| \cdot v \cdot [2 \cdot e^{(-B \cdot t)} \cdot (1 + \sin(2 \cdot \pi \cdot f \cdot t)]$

This approach ensures non negative samples and avoid discontinuities.

Samples are also scaled to remain in $(1/3) \cdot PWM\_LEVELS$ range for the slow pattern implementation; the normal and fast patterns are then runtime generated taking the slow pattern samples multiplied by the velocity parameter.

$PS(t) = PS(t) \cdot SCALE\_FACTOR \cdot (PWM\_LEVELS/3)$

Note

Refer to SCILAB script impact.sce to learn how samples were generated.

Graphics of vibration patterns obtained using this formula are reported in figure:

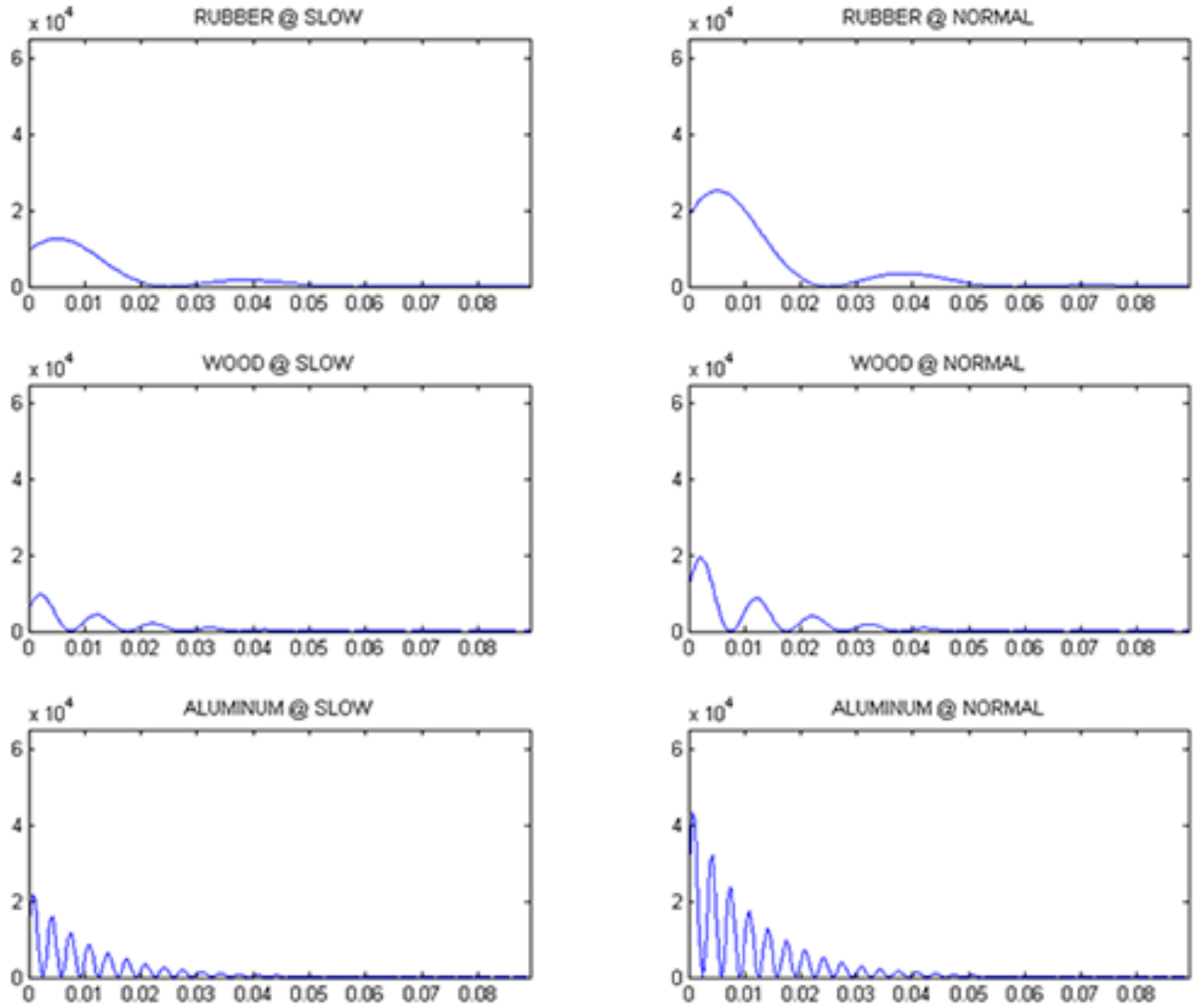

Figure 10.1: Impact Pattern Graphics

10.12.2 Optional Parameters

Impact Pattern Generator needs two optional parameters: `velocity` and `material`, that can be

- `velocity`:
  - `Slow`
  - `Normal`
  - `Fast`
- `material`:
  - `Rubber`
  - `Wood`
  - `Aluminum`

Users must define a structure of impactPatternParameters type to pass these parameters.

Usage Examples

This is a sample code snippets that show how impact pattern must be sent:

```
...
haptor_desc *myHaptor = hl_configure(24000000,15,1);

impactPatternParameters params;

params.velocity = Fast;
params.material = Wood;

pattern_desc *myImpactPattern = hl_initPattern(Impact, &params);

hl_addHaptor(myHaptor,myImpactPattern);

hl_startPattern(myImapctPattern);
...
```

Debugging Details

If HL_DEBUG is defined some useful informations are send to USART.

In particular will be send:

- impact velocity magnitude used
- surface material selected
- first and last 10 samples of the pattern

Additional Notes

No additional notes.

Authors

Leonardo Guardati
Silvio Vallorani

Version

v0.7

Date

2012

Definition in file impact.c.

10.12.3 Function Documentation

10.12.3.1 uint8_t impactContinuator (pattern_desc * *pattern*)

Pattern generator **continuator**

This function is called back from the patternScheduler() every SystemDesc.samples_delay ms.

At every call, this continuator will point to the next element of the right array of samples.

When reached the last sample the pattern is finished an remove itself from the scheduling patterns.

Definition at line 272 of file impact.c.

10.12.3.2 uint8_t impactPatternGenerator (pattern_desc * *pattern*)

Pattern generator **initiator**

This function is called by the **User API Module** of HapticLib, when the library user calls hl_startPattern().

The first task for the pattern generator **initiator** is to validate the user provided inputs, this cannot be done anywhere else.

This pattern initiator then sets up all the status parameters.

If the pattern accept user parameters, the status setting can be based on the user provided parameters.

The last duty of this function is to put the pattern continuator up for scheduling (setting `pattern->continuator =&impactContinuator;`), so the next time patternScheduler() will check for active patterns, it will find this pattern.

Warning

If the **initiator** doesn't set the continuator up for scheduling, the pattern will never start, and for HapticLib, the haptor is free to receive other patterns.

Parameters

`out`	*pattern*	A pattern_desc* holding the reference to the running pattern instance to work with. Through this reference, the continuator can extract the user and status parameters in order to execute the haptic pattern.

Returns

uint8_t The result is returned to indicate success or error.

Return values

0	Success
1	Error

Definition at line 431 of file impact.c.

10.12.4 Variable Documentation

10.12.4.1 const uint16_t aluminumImpactPattern[256]

Initial value:

```
 {
11711,  18304,  21647,  20434,  15299,   8439,   2627,     41,   1417,   5832,
11188,  15131,  16034,  13609,   8947,   3986,    673,    165,   2408,   6234,
 9903,  11848,  11302,   8563,   4809,   1560,     40,    696,   3063,   6008,
 8234,   8819,   7567,   5045,   2304,    423,     66,   1236,   3315,   5355,
 6481,   6247,   4790,   2738,    923,     34,    340,   1606,   3224,   4478,
 4848,   4205,   2842,   1329,    264,     25,    633,   1761,   2894,   3543,
 3452,   2678,   1556,    545,     27,    164,    841,   1728,   2434,   2664,
 2335,   1600,    766,    163,      8,    323,    934,   1562,   1936,   1906,
 1496,    884,    320,     20,     79,    439,    926,   1322,   1463,   1296,
  900,    440,    100,      2,    164,    495,    843,   1057,   1052,    835,
  501,    188,     14,     37,    229,    495,    718,    803,    719,    506,
  253,     61,      0,     83,    262,    454,    577,    580,    466,    284,
  110,     10,     17,    119,    265,    389,    440,    398,    284,    145,
   37,      0,     42,    138,    245,    315,    320,    259,    161,     64,
    6,      8,     62,    141,    211,    241,    221,    159,     83,     22,
    0,     21,     73,    132,    171,    176,    144,     91,     37,      4,
    3,     32,     75,    114,    132,    122,     89,     47,     13,      0,
   10,     38,     71,     93,     97,     80,     51,     21,      3,      1,
   16,     40,     62,     72,     67,     50,     27,      8,      0,      5,
   20,     38,     51,     53,     44,     29,     12,      1,      0,      8,
```

```
  21,     33,     39,     37,     28,     15,      4,      0,      2,     10,
  20,     27,     29,     24,     16,      7,      1,      0,      4,     11,
  18,     21,     20,     15,      8,      2,      0,      1,      5,     11,
  15,     16,     13,      9,      4,      0,      0,      2,      6,      9,
  11,     11,      8,      5,      1,      0,      0,      2,      5,      8,
   8,      7,      5,      2,      0,      0
    }
```

Hardcoded, Haptic Feedback Pattern of impact on Aluminum surface.

The aluminumImpactPattern describe a vibration in response to impact on Aluminum. (256 samples)

Defined as const to store it in flash memory.

Note

Refer to SCILAB script impact.sce to learn how samples was generated.

Definition at line 231 of file impact.c.

10.12.4.2 const uint16_t rubberImpactPattern[256]

Initial value:

```
 {
 9369,   9779,  10165,  10527,  10861,  11167,  11445,  11692,  11909,  12095,
12249,  12372,  12463,  12524,  12553,  12552,  12521,  12462,  12374,  12259,
12118,  11953,  11763,  11552,  11319,  11067,  10798,  10511,  10210,   9896,
 9569,   9233,   8887,   8535,   8177,   7815,   7450,   7084,   6719,   6354,
 5993,   5635,   5283,   4937,   4598,   4267,   3946,   3634,   3333,   3043,
 2765,   2500,   2247,   2008,   1782,   1570,   1372,   1188,   1018,    862,
  720,    592,    477,    375,    287,    211,    148,     97,     57,     28,
    9,      0,      1,     10,     28,     53,     85,    124,    168,    217,
  271,    329,    390,    455,    521,    589,    658,    727,    797,    867,
  935,   1003,   1069,   1133,   1195,   1254,   1310,   1363,   1413,   1459,
 1501,   1540,   1574,   1605,   1631,   1653,   1670,   1684,   1693,   1698,
 1699,   1696,   1688,   1677,   1663,   1644,   1623,   1598,   1570,   1539,
 1506,   1470,   1432,   1391,   1349,   1305,   1260,   1214,   1166,   1118,
 1069,   1020,    970,    921,    871,    822,    774,    726,    679,    633,
  588,    544,    501,    460,    421,    383,    346,    312,    279,    248,
  219,    191,    166,    143,    121,    101,     84,     68,     53,     41,
   30,     22,     14,      8,      4,      1,      0,      0,      1,      3,
    6,     10,     15,     21,     27,     34,     42,     50,     59,     68,
   77,     86,     96,    105,    115,    124,    133,    142,    151,    159,
  167,    175,    182,    189,    196,    201,    207,    212,    216,    219,
  223,    225,    227,    228,    229,    229,    229,    228,    227,    225,
  223,    220,    217,    213,    209,    204,    200,    195,    189,    184,
  178,    172,    165,    159,    152,    146,    139,    132,    126,    119,
  112,    106,     99,     93,     87,     81,     75,     69,     63,     58,
   53,     48,     43,     38,     34,     30,     26,     23,     20,     17,
   14,     11,      9,      7,      6,      0
    }
```

Hardcoded, Haptic Feedback Pattern of impact on Rubber surface.

The rubberImpactPattern describe a vibration in response to impact on Rubber. (256 samples)

Defined as const to store it in flash memory.

Note

Refer to SCILAB script impact.sce to learn how samples was generated.

Definition at line 152 of file impact.c.

10.12.4.3 const uint16_t woodImpactPattern[256]

Initial value:

```
{
5855,   6936,   7894,   8683,   9269,   9626,   9745,   9624,   9278,   8728,
8006,   7149,   6200,   5204,   4205,   3245,   2364,   1593,    958,    479,
 163,     14,     24,    181,    465,    851,   1312,   1818,   2338,   2844,
3309,   3709,   4026,   4246,   4361,   4368,   4269,   4073,   3791,   3438,
3033,   2595,   2145,   1701,   1283,    906,    585,    329,    145,     35,
   0,     33,    129,    278,    468,    685,    917,   1149,   1370,   1568,
1732,   1857,   1936,   1967,   1949,   1885,   1780,   1639,   1469,   1280,
1079,    877,    682,    501,    342,    210,    108,     40,      5,      2,
  30,     84,    159,    251,    352,    457,    560,    655,    738,    804,
 852,    878,    883,    866,    829,    774,    705,    624,    536,    446,
 356,    270,    193,    126,     73,     33,      9,      0,      5,     22,
  51,     88,    131,    178,    225,    270,    311,    345,    371,    389,
 396,    394,    383,    362,    335,    301,    264,    223,    182,    143,
 106,     73,     45,     24,      9,      1,      0,      5,     15,     29,
  47,     68,     89,    110,    129,    146,    160,    170,    176,    178,
 175,    168,    158,    144,    128,    110,     92,     74,     57,     41,
  27,     16,      7,      2,      0,      0,      3,      9,     16,     25,
  34,     44,     53,     61,     68,     74,     78,     80,     79,     77,
  73,     68,     61,     54,     46,     38,     30,     22,     15,      9,
   5,      2,      0,      0,      0,      2,      5,      9,     13,     17,
  21,     25,     29,     32,     34,     35,     36,     35,     34,     32,
  29,     26,     22,     19,     15,     12,      8,      5,      3,      1,
   0,      0,      0,      0,      1,      3,      4,      6,      8,     10,
  12,     13,     14,     15,     16,     16,     15,     15,     14,     12,
  11,      9,      7,      6,      4,      3,      2,      1,      0,      0,
   0,      0,      0,      1,      1,      0
    }
```

Hardcoded, Haptic Feedback Pattern of impact on Wood surface.

The woodImpactPattern describe a vibration in response to impact on Wood. (256 samples)

Defined as const to store it in flash memory.

Note

Refer to SCILAB script impact.sce to learn how samples was generated.

Definition at line 191 of file impact.c.

10.13 HapticLib/patterns/impact/impact.h File Reference

Impact Pattern generator function header.

Data Structures

- struct impactUserParameters

  Struct for user provided parameters to hl_startPattern().
- struct impactStatusParameters

  *Impact pattern **status** parameters type definition.*

Typedefs

- typedef enum ImpactMaterial ImpactMaterial

  *User Parameter Values for **Material** user parameter.*
- typedef enum ImpactVelocity ImpactVelocity

  *User Parameter Values for **Velocity** user parameter.*

Enumerations

- enum ImpactMaterial { Rubber = 1, Wood = 2, Aluminum = 3 }

  *User Parameter Values for **Material** user parameter.*
- enum ImpactVelocity { Slow = 1, Normal = 2, Fast = 3 }

  *User Parameter Values for **Velocity** user parameter.*

10.13.1 Detailed Description

Impact Pattern generator function header. Here impactPatternParameters type is defined.

Author

Silvio Vallorani

Version

v0.7

Date

2012

Definition in file impact.h.

10.13.2 Typedef Documentation

10.13.2.1 typedef enum ImpactMaterial ImpactMaterial

User Parameter Values for **Material** user parameter.

Enumeration of material types that can be passed as parameter to Impact Pattern Generator in a user friendly way.

10.13.2.2 typedef enum ImpactVelocity ImpactVelocity

User Parameter Values for **Velocity** user parameter.

Enumeration of velocities that can be passed as parameter to Impact Pattern Generator in a user friendly way.

10.13.3 Enumeration Type Documentation

10.13.3.1 enum ImpactMaterial

User Parameter Values for **Material** user parameter.

Enumeration of material types that can be passed as parameter to Impact Pattern Generator in a user friendly way.

Enumerator:

Rubber

Wood

Aluminum

Definition at line 41 of file impact.h.

10.13.3.2 enum ImpactVelocity

User Parameter Values for **Velocity** user parameter.

Enumeration of velocities that can be passed as parameter to Impact Pattern Generator in a user friendly way.

Enumerator:

Slow

Normal

Fast

Definition at line 55 of file impact.h.

10.14 HapticLib/patterns/test/test.c File Reference

Test Pattern generator function definition

```
#include "hapticLib.h"
```

Functions

- uint8_t testContinuator (pattern_desc *pattern)

  Test** Pattern generator **continuator
- uint8_t testPatternGenerator (pattern_desc *pattern)

  *Test Pattern generator **initiator***

10.14.1 Detailed Description

Test Pattern generator function definition Implementation code of the Test Pattern.

Introduction

This pattern reproduce two linear ramp from 0% to 100% duty-cycle.

Theory

Simple pattern with sawtooth profile.

Optional Parameters

No optional parameters.

Usage Examples

Here is a typical usage scenario for the Test Pattern:

```
...
haptor_desc *myHaptor = hl_configure(24000000,15,1);

pattern_desc *myTestPattern = hl_initPattern(Test,NULL);

hl_addHaptor(myHaptor, myTestPattern);

hl_startPattern(myTestPattern);
...
```

A condensed version of the same code is the follow:

```
...
haptor_desc *myHaptor = hl_configure(24000000,15,1);

hl_startPattern(hl_addHaptor(myHaptor, hl_initPattern(Test,NULL) ) );
...
```

Note

This shortcut is valid and can be used only for single haptor pattern.

Debugging Details

testPatternGenerator doesn't need optional parameter, just pass a NULL. If any are passed, an error occurs and if HL_DEBUG is defined this error will be sent on UART.

Additional Notes

No additional notes.

Authors

Leonardo Guardati
Silvio Vallorani

Version

v0.7

Date

2012

Definition in file test.c.

10.14.2 Function Documentation

10.14.2.1 uint8_t testContinuator (pattern_desc * *pattern*)

Test Pattern generator **continuator**

This function is called back from the SysTick_Handler every SystemDesc.samples_delay ms.

At every call, this continuator will calculate the next value of PWM duty-cycle.

After two positive ramps the pattern is finished and remove itself from the scheduling patterns.

Definition at line 100 of file test.c.

10.14.2.2 uint8_t testPatternGenerator (pattern_desc * *pattern*)

Test Pattern generator **initiator**

This function is called by the **User API Module** of HapticLib, when the library user calls hl_sendPattern().

The first task for the pattern generator **initiator** is to validate the user provided inputs, this cannot be done anywhere else.

This pattern initiator then sets up all the status parameters.

If the pattern accept user parameters, the status setting can be based on the user provided parameters.

The last duty of this function is to put the pattern continuator up for scheduling (setting `pattern->continuator =&testContinuator;`), so the next time patternScheduler() will check for active patterns, it will find this pattern.

Warning

If the **initiator** doesn't set the continuator up for scheduling, the pattern will never start, and for HapticLib, the haptor is free to receive other patterns.

Note

This pattern doesn't need user parameters, and check for this parameter to be NULL.

Parameters

out	*pattern*	A pattern_desc* holding the reference to the running pattern instance to work with. Through this reference, the continuator can extract the user and status parameters in order to execute the haptic pattern.

Returns

uint8_t The result is returned to indicate success or error.

Return values

0	Success
1	Error

Definition at line 189 of file test.c.

10.15 HapticLib/patterns/test/test.h File Reference

Test Pattern generator function header

Data Structures

- struct testStatusParameters

  *Test Pattern specific **status** parameters typedef.*

10.15.1 Detailed Description

Test Pattern generator function header No Pattern specific **constant definitions** are needed by this pattern.

The Pattern specific **status parameters** typedef is defined here.

No Pattern specific **user parameters** typedef is needed.

Authors

Leonardo Guardati
Silvio Vallorani

Version

v0.7

Date

2012

Definition in file test.h.

10.16 HapticLib/platforms/STM32VLDISCOVERY/hl_STM32VLDISCOVERY.c File Reference

Platform Specific Module STM32VLDISCOVERY definitions.

```
#include "hapticLib.h"
```

Macros

- #define HL_SYSTEM_FILE

Functions

- void RCC_Configuration (void)

  Enable Clock control for peripherals needed by HapticLib.
- void GPIO_Configuration (void)

  Setup the GPIOs used by HapticLib.
- void TIM_Configuration (uint32_t pwm_freq)

  Setup System Timers (TIMx and SysTick).
- uint8_t TIM_Channel_Enable (uint8_t channel)

  Single timer channel activation.
- uint8_t TIM_Channel_DutyChanger (uint16_t new_duty, uint8_t channel)

  Single timer channel duty cycle update.
- void Delay (__IO uint32_t nTime)

  Delay function to wait `nTime` milliseconds.
- void SysTick_Handler (void)

  SysTick ISR override.

Variables

- system_desc SystemDesc

  Global variable to describe the haptic system.
- static __IO uint32_t TimingDelay = 0

  Number of SysTicks to wait for Delay() to return.
- static uint8_t channelStatus = 0xFF

  Timer Channels status informations.

10.16.1 Detailed Description

Platform Specific Module STM32VLDISCOVERY definitions. This file is part of the **Platform Specific Module** of HapticLib.

HapticLib support for the STM32VL-DISCOVERY platform is documented on the Architecture page, please refer to it for additional informations.

Author

Leonardo Guardati

Version

0.7

Date

2012

Definition in file hl_STM32VLDISCOVERY.c.

10.16.2 Macro Definition Documentation

10.16.2.1 #define HL_SYSTEM_FILE

Definition at line 39 of file hl_STM32VLDISCOVERY.c.

10.16.3 Function Documentation

10.16.3.1 void Delay (__IO uint32_t *nTime*)

Delay function to wait `nTime` milliseconds.

Delay() sets the global TimingDelay variable to the number of times SysTick must decrement it until it reaches zero, then Delay() returns.

SysTick generates an interrupt every 1ms.

Note

Delay sleeps while waiting for SysTick.

Parameters

in	*nTime*	The number of tick to wait. (1 tick = 1 ms)

Definition at line 559 of file hl_STM32VLDISCOVERY.c.

10.16.3.2 void GPIO_Configuration (void)

Setup the GPIOs used by HapticLib.

For STM32VL-DISCOVERY, the following IO will ALWAYS be configured:

- GPIOA:
  - PA0: UserButton
  - PA6: TIM3 Ch1 (AFIO)
  - PA7: TIM3 Ch2 (AFIO)
- GPIOB:
  - PB0: TIM3 Ch3 (AFIO)
  - PB1: TIM3 Ch4 (AFIO)
- GPIOC:
  - PC8: User LED

Note

Additionally, only if HL_DEBUG is defined, the following IO will be configured:

- GPIOB:
  - PB10: USART3(TX) (AFIO)
  - PB11: USART3(RX) (AFIO)
- GPIOC:
  - PC9: Debug LED

Definition at line 169 of file hl_STM32VLDISCOVERY.c.

10.16.3.3 void RCC_Configuration (void)

Enable Clock control for peripherals needed by HapticLib.

For STM32VL-DISCOVERY, here is the standard setting:

- APB1:
  - TIM3 (PWM Generator)
- APB2:
  - GPIOA:
    - PA0: UserButton
    - PA6: TIM3 Ch1
    - PA7: TIM3 Ch2
  - GPIOB:
    - PB0: TIM3 Ch3
    - PB1: TIM3 Ch4
  - GPIOC:
    - PC8: User LED
  - AFIO:
    - Enable TIM3 access to GPIOs.

Note

Additionally, only if HL_DEBUG is defined, the following is set too:

- APB2:
  - GPIOB:
    - PB10: USART3(TX)
    - PB11: USART3(RX)
  - GPIOC:
    - PC9: Debug LED
  - AFIO:
    - Enable USART3 access to GPIOs.

Here system debug for core and peripherals are set.

Note

For system debug (that's not HL_DEBUG HapticLib Debugging Features), the following configuration will be setup:

- Debugger connection kept for MCU in:
  - SLEEP MODE
  - STOP MODE
  - STANDBY MODE
- TIM3 counter stopped when Core is halted

Definition at line 127 of file hl_STM32VLDISCOVERY.c.

10.16.3.4 void SysTick_Handler (void)

SysTick ISR override.

At every SysTick Timer event (1ms), SysTick_Handler() is called.

SysTick_Handler() does 2 things:

- Decrements TimingDelay used by Delay()
- Calls the **User API**'s **Pattern Scheduler** to update the active pattern's PWMs.

See also

To know the internal details of how HapticLib multi-haptor technique works, please refer to the documentation on the Developer Guide page.

Definition at line 581 of file hl_STM32VLDISCOVERY.c.

10.16.3.5 uint8_t TIM_Channel_DutyChanger (uint16_t *new_duty*, uint8_t *channel*)

Single timer channel duty cycle update.

TIM_Channel_DutyChanger() uses `new_duty` to update the `channel`-th channel.

Warning

TIM_Channel_DutyChanger() is called internally by the library and should not be used directly by the user.

Todo Clarify the **User API** <–> **Platform Specific** interface; this logic may need to be changed.

Parameters

in	*new_duty*	It is a `uint16_t` value. It is the new duty cycle expressed as fraction of 0xFFFF. The range is from 0x0000 (0%) to 0xFFFF (100%). The resulting duty cycle resolution is 1.5%■.
in	*channel*	It is a `uint8_t` value. It is the ID of the channel whose PWM control signal is going to be updated.

Returns

uint8_t The result is returned to indicate success or error.

Return values

0	Success
1	Error

Definition at line 417 of file hl_STM32VLDISCOVERY.c.

10.16.3.6 uint8_t TIM_Channel_Enable (uint8_t *channel*)

Single timer channel activation.

TIM_Channel_Enable() activates the single channel passed as argument.

Warning

TIM_Channel_Enable() is called internally by the library and should not be used directly by the user.

Before activating the channel, it is checked that the channels are configured (`channelStatus != 0xFF`) and that the channel passed is not already active.

Note

After activation the channel's Duty Cycle is cleared to 0.

Todo Clarify the **User API** <–> **Platform Specific** interface; this logic may need to be changed.

Parameters

`in`	*channel*	this is a `uint8_t` value. Its value represents the channel to be activated.

Returns

uint8_t The result is returned to indicate success or error.

Return values

0	Success
1	Error

Definition at line 332 of file hl_STM32VLDISCOVERY.c.

10.16.3.7 void TIM_Configuration (uint32_t *pwm_freq*)

Setup System Timers (TIMx and SysTick).

This function prepares the timers used by HapticLib. It must be called before any other timer operation.

Warning

TIM_Configuration() is called internally by the library and should not be used directly by the user.

This is what happen:

1. TIM3 timer is setup using `pwm_freq` as parameter.
2. TIM3 Channels are set in PWM Output mode.
3. SysTick is set to **1ms** time resolution.

The Timer PWM frequency is set using the parameter `pwm_freq` expressing the frequency in Hertz.

This platform operates at 24MHz (CPU, AHP, APB1/2) so the TIM timer can reach this speed too.

The SysTick is fixed at 1ms operating frequency, so the time resolution of the application is 1ms.

Note

The Timer ARR is set to 0xFFFF to allow maximum granularity on PWM DutyCylces

Parameters

`in`	*pwm_freq*	this is a `uint32_t` value. Its value represents the PWM frequency in Hz. Valid range is: 367 to 24000000.

Note

Keep in mind that the actual PWM frequency is scaled from SystemCoreClock. So only fractions of 24MHz can be set. However for low values of pwm_freq you can assume the actual frequency set is the same.

Definition at line 261 of file hl_STM32VLDISCOVERY.c.

10.16.4 Variable Documentation

10.16.4.1 uint8_t channelStatus = 0xFF [static]

Timer Channels status informations.

This variable holds the status informations on all the Timers channels present in the system.

The information is encoded as follow:

- `channelStatus = 0xFF`: Initial value; means the channels are not configured yet, so they cannot be enabled.
- `channelStatus = 0x00`: The channels are configured and ready to be enabled.

The status of the n-th channel is then encoded on the n-th bit of channelStatus.

For example, if only channel_1 is enabled, `channelStatus=0x01`

If `channelStatus=0x16` (in binary 0001 0110) there are 3 channels enabled; channel_2,channel_3 and channel_5.

Note

The maximum number of active channels is 0xEF: 7 channels.

Definition at line 82 of file hl_STM32VLDISCOVERY.c.

10.16.4.2 system_desc SystemDesc

Global variable to describe the haptic system.

The Library uses this global variable to describe the system and to keep track of the haptic devices' status and active patterns' progress at any time.

Note

The library user doesn't need to access this variable.

The SystemDesc variable holds a lot of informations, and its members allow the system to access almost all the informations needed.

See also

Please refer to the Developer Guide page for details.

Definition at line 70 of file hapticLib.c.

10.16.4.3 __IO uint32_t TimingDelay = 0 [static]

Number of SysTicks to wait for Delay() to return.

This is used by Delay() and SysTick_Handler() to create time delays on user application.

Delay() set this to a number of tick (by default 1ms) SysTick have to interrupt before it can return.

Definition at line 56 of file hl_STM32VLDISCOVERY.c.

10.17 HapticLib/platforms/STM32VLDISCOVERY/hl_STM32VLDISCOVERY.h File Reference

Platform Specific Module STM32VLDISCOVERY header.

```
#include "stm32f10x.h"
```

Macros

- #define MAX_HAPTORS 4

  Maximum number of Haptor supported in this Platform.
- #define STM32F10X_MD_VL

  Use the STM StdPeriph Library.

Functions

- void RCC_Configuration (void)

  Enable Clock control for peripherals needed by HapticLib.
- void GPIO_Configuration (void)

  Setup the GPIOs used by HapticLib.
- void TIM_Configuration (uint32_t)

  Setup System Timers (TIMx and SysTick).
- uint8_t TIM_Channel_Enable (uint8_t)

  Single timer channel activation.
- uint8_t TIM_Channel_DutyChanger (uint16_t, uint8_t)

  Single timer channel duty cycle update.
- void SysTick_Handler (void)

  SysTick ISR override.
- void Delay (__IO uint32_t)

  Delay function to wait `nTime` milliseconds.

10.17.1 Detailed Description

Platform Specific Module STM32VLDISCOVERY header. This file is part of the **Platform Specific Module** of HapticLib.

HapticLib support for the STM32VL-DISCOVERY platform is documented on the Architecture page, please refer to it for additional informations.

Author

Leonardo Guardati

Version

0.7

Date

2012

Definition in file hl_STM32VLDISCOVERY.h.

10.17.2 Macro Definition Documentation

10.17.2.1 #define MAX_HAPTORS 4

Maximum number of Haptor supported in this Platform.

At the moment **HapticLib STM32VLDISCOVERY Platform Specific Module** supports only 4 Haptic devices.

For hardware connections and details, please refer to HapticLib documentation on the Architecture page.

Note

following versions will increase the number of maximum haptors supported.

Definition at line 53 of file hl_STM32VLDISCOVERY.h.

10.17.2.2 #define STM32F10X_MD_VL

Use the STM StdPeriph Library.

This is the only dependency needed by HapticLib. To successfully compile and use the library code, you need to unpack the StdPeriph on the root of HapticLib.

See also

Please refer to the User Guide for details on user environment setup. MCU device class needed by STM's StdPeriph.

STM32 MCU device class define. (MD: Medium Density, VL: Value Line)

This symbol is required by StdPeriph, and this is the MCU present on the STM32VLDISCOVERY board.

Definition at line 82 of file hl_STM32VLDISCOVERY.h.

10.17.3 Function Documentation

10.17.3.1 void Delay (__IO uint32_t *nTime*)

Delay function to wait `nTime` milliseconds.

Delay() sets the global TimingDelay variable to the number of times SysTick must decrement it until it reaches zero, then Delay() returns.

SysTick generates an interrupt every 1ms.

Note

Delay sleeps while waiting for SysTick.

Parameters

in	*nTime*	The number of tick to wait. (1 tick = 1 ms)

Definition at line 559 of file hl_STM32VLDISCOVERY.c.

10.17.3.2 void GPIO_Configuration (void)

Setup the GPIOs used by HapticLib.

For STM32VL-DISCOVERY, the following IO will ALWAYS be configured:

- GPIOA:
  - PA0: UserButton
  - PA6: TIM3 Ch1 (AFIO)
  - PA7: TIM3 Ch2 (AFIO)
- GPIOB:
  - PB0: TIM3 Ch3 (AFIO)
  - PB1: TIM3 Ch4 (AFIO)
- GPIOC:
  - PC8: User LED

Note

Additionally, only if HL_DEBUG is defined, the following IO will be configured:

- GPIOB:
  - PB10: USART3(TX) (AFIO)
  - PB11: USART3(RX) (AFIO)
- GPIOC:
  - PC9: Debug LED

Definition at line 169 of file hl_STM32VLDISCOVERY.c.

10.17.3.3 void RCC_Configuration (void)

Enable Clock control for peripherals needed by HapticLib.

For STM32VL-DISCOVERY, here is the standard setting:

- APB1:
  - TIM3 (PWM Generator)
- APB2:
  - GPIOA:
    - PA0: UserButton
    - PA6: TIM3 Ch1
    - PA7: TIM3 Ch2
  - GPIOB:
    - PB0: TIM3 Ch3
    - PB1: TIM3 Ch4
  - GPIOC:
    - PC8: User LED
  - AFIO:
    - Enable TIM3 access to GPIOs.

Note

Additionally, only if HL_DEBUG is defined, the following is set too:

- APB2:
  - GPIOB:
    - PB10: USART3(TX)
    - PB11: USART3(RX)
  - GPIOC:
    - PC9: Debug LED
  - AFIO:
    - Enable USART3 access to GPIOs.

Here system debug for core and peripherals are set.

Note

For system debug (that's not HL_DEBUG HapticLib Debugging Features), the following configuration will be setup:

- Debugger connection kept for MCU in:
  - SLEEP MODE
  - STOP MODE
  - STANDBY MODE
- TIM3 counter stopped when Core is halted

Definition at line 127 of file hl_STM32VLDISCOVERY.c.

10.17.3.4 void SysTick_Handler (void)

SysTick ISR override.

At every SysTick Timer event (1ms), SysTick_Handler() is called.

SysTick_Handler() does 2 things:

- Decrements TimingDelay used by Delay()
- Calls the **User API**'s **Pattern Scheduler** to update the active pattern's PWMs.

See also

To know the internal details of how HapticLib multi-haptor technique works, please refer to the documentation on the Developer Guide page.

Definition at line 581 of file hl_STM32VLDISCOVERY.c.

10.17.3.5 uint8_t TIM_Channel_DutyChanger (uint16_t *new_duty,* uint8_t *channel*)

Single timer channel duty cycle update.

TIM_Channel_DutyChanger() uses `new_duty` to update the `channel`-th channel.

Warning

TIM_Channel_DutyChanger() is called internally by the library and should not be used directly by the user.

Todo Clarify the ∗∗User API∗∗ <–> ∗∗Platform Specific∗∗ interface; this logic may need to be changed.

Parameters

`in`	*new_duty*	It is a `uint16_t` value. It is the new duty cycle expressed as fraction of 0xFFFF. The range is from 0x0000 (0%) to 0xFFFF (100%). The resulting duty cycle resolution is 1.5%■.
`in`	*channel*	It is a `uint8_t` value. It is the ID of the channel whose PWM control signal is going to be updated.

Returns

uint8_t The result is returned to indicate success or error.

Return values

0	Success
1	Error

Definition at line 417 of file hl_STM32VLDISCOVERY.c.

10.17.3.6 uint8_t TIM_Channel_Enable (uint8_t *channel*)

Single timer channel activation.

TIM_Channel_Enable() activates the single channel passed as argument.

Warning

TIM_Channel_Enable() is called internally by the library and should not be used directly by the user.

Before activating the channel, it is checked that the channels are configured (`channelStatus != 0xFF`) and that the channel passed is not already active.

Note

After activation the channel's Duty Cycle is cleared to 0.

Todo Clarify the **User API** <–> **Platform Specific** interface; this logic may need to be changed.

Parameters

`in`	*channel*	this is a `uint8_t` value. Its value represents the channel to be activated.

Returns

uint8_t The result is returned to indicate success or error.

Return values

0	Success
1	Error

Definition at line 332 of file hl_STM32VLDISCOVERY.c.

10.17.3.7 void TIM_Configuration (uint32_t *pwm_freq*)

Setup System Timers (TIMx and SysTick).

This function prepares the timers used by HapticLib. It must be called before any other timer operation.

Warning

TIM_Configuration() is called internally by the library and should not be used directly by the user.

This is what happen:

1. TIM3 timer is setup using `pwm_freq` as parameter.
2. TIM3 Channels are set in PWM Output mode.
3. SysTick is set to **1ms** time resolution.

The Timer PWM frequency is set using the parameter `pwm_freq` expressing the frequency in Hertz.

This platform operates at 24MHz (CPU, AHP, APB1/2) so the TIM timer can reach this speed too.

The SysTick is fixed at 1ms operating frequency, so the time resolution of the application is 1ms.

Note

The Timer ARR is set to 0xFFFF to allow maximum granularity on PWM DutyCylces

Parameters

`in`	*pwm_freq*	this is a `uint32_t` value. Its value represents the PWM frequency in Hz. Valid range is: 367 to 24000000.

Note

Keep in mind that the actual PWM frequency is scaled from SystemCoreClock. So only fractions of 24MHz can be set. However for low values of pwm_freq you can assume the actual frequency set is the same.

Definition at line 261 of file hl_STM32VLDISCOVERY.c.

Index

ACTION
impact.m, 90
activeHaptorList
pattern_desc, 47
activePattern
haptor_desc, 44
Aluminum
impact.h, 99
aluminumImpactPattern
impact.c, 96

bigIncrement
generic.h, 76

CONTRACT
impact.m, 90
channelStatus
hl_STM32VLDISCOVERY.c, 108
checkParam
genericUserParameters, 43
cleanList
hl_patterns.c, 78
hl_patterns.h, 86
Constant
hl_patterns.h, 86
constant
constantUserParameters, 42
status_param, 48
user_param, 50
constant.c
constantContinuator, 68
constantPatternGenerator, 69
constantContinuator
constant.c, 68
constantPatternGenerator
constant.c, 69
hl_patterns.c, 78
constantStatusParameters, 41
duty, 41
constantUserParameters, 41
constant, 42
continuator
pattern_desc, 47
copy
impact.m, 90
Copyright
impact.m, 90

DIRECT
impact.m, 90

Delay
hl_STM32VLDISCOVERY.c, 104
hl_STM32VLDISCOVERY.h, 111
duty
constantStatusParameters, 41
genericStatusParameters, 42
testStatusParameters, 49
dutyConverter
hl_patterns.c, 79
hl_patterns.h, 86

Fast
impact.h, 100
flag
genericStatusParameters, 42
testStatusParameters, 49

GPIO_Configuration
hl_STM32VLDISCOVERY.c, 104
hl_STM32VLDISCOVERY.h, 111
Generic
hl_patterns.h, 86
generic
status_param, 48
user_param, 50
generic.h
bigIncrement, 76
rightValue, 76
smallIncrement, 76
generic.c
genericContinuator, 72
genericPatternGenerator, 73
HL_DEBUG, 72
generic.h
genericCheckParam, 75, 76
genericIncrement, 75, 76
genericStatusParameters, 75
genericUserParameters, 76
genericCheckParam
generic.h, 75, 76
genericContinuator
generic.c, 72
genericIncrement
generic.h, 75, 76
genericPatternGenerator
generic.c, 73
hl_patterns.c, 79
genericStatusParameters, 42
duty, 42
flag, 42

generic.h, 75
genericUserParameters, 43
checkParam, 43
generic.h, 76
increment, 43
granted
impact.m, 91

HL_DEBUG
generic.c, 72
hapticLib.h, 57
hl_debug.c, 62
hl_debug.h, 65
HL_SYSTEM_FILE
hapticLib.c, 52
hl_patterns.c, 78
hapticLib.c
HL_SYSTEM_FILE, 52
hl_addHaptor, 52
hl_configure, 53
hl_initPattern, 53
hl_startPattern, 54
hl_stopPattern, 55
patternMap, 55
SystemDesc, 55
hapticLib.h
HL_DEBUG, 57
haptor_desc, 58
hl_addHaptor, 58
hl_configure, 59
hl_initPattern, 59
hl_startPattern, 60
hl_stopPattern, 61
STM32VLDISCOVERY, 58
HapticLib/hapticLib.c, 51
HapticLib/hapticLib.h, 56
HapticLib/hl_debug.c, 61
HapticLib/hl_debug.h, 64
HapticLib/patterns/constant/constant.c, 67
HapticLib/patterns/constant/constant.h, 70
HapticLib/patterns/generic/generic.c, 70
HapticLib/patterns/generic/generic.h, 73
HapticLib/patterns/hl_patterns.c, 77
HapticLib/patterns/hl_patterns.h, 82
HapticLib/patterns/impact/extra/impact.m, 87
HapticLib/patterns/impact/impact.c, 92
HapticLib/patterns/impact/impact.h, 98
HapticLib/patterns/test/test.c, 100
HapticLib/patterns/test/test.h, 102
haptor_desc, 43
activePattern, 44
hapticLib.h, 58
id, 44
max_duty, 44
min_duty, 44
nextHaptor, 44
hl_patterns.h
Constant, 86
Generic, 86
Impact, 86
Null, 86
Num_Patterns_Available, 86
Test, 86
hl_STM32VLDISCOVERY.c
channelStatus, 108
Delay, 104
SysTick_Handler, 106
SystemDesc, 108
TimingDelay, 109
hl_STM32VLDISCOVERY.h
Delay, 111
SysTick_Handler, 112
hl_addHaptor
hapticLib.c, 52
hapticLib.h, 58
hl_configure
hapticLib.c, 53
hapticLib.h, 59
hl_debug.c
HL_DEBUG, 62
send_int, 63
send_string, 63
hl_debug.h
HL_DEBUG, 65
send_char, 66
send_int, 66
send_string, 66
hl_initPattern
hapticLib.c, 53
hapticLib.h, 59
hl_patterns.c
cleanList, 78
constantPatternGenerator, 78
dutyConverter, 79
genericPatternGenerator, 79
HL_SYSTEM_FILE, 78
impactPatternGenerator, 80
patternMap, 81
patternScheduler, 80
SystemDesc, 82
testPatternGenerator, 81
hl_patterns.h
cleanList, 86
dutyConverter, 86
MAX_PATTERNS, 84
pattern_continuator, 84
pattern_desc, 84
pattern_initiator, 84
pattern_name, 85
patternScheduler, 86
status_param, 85
user_param, 85
hl_startPattern
hapticLib.c, 54
hapticLib.h, 60
hl_stopPattern
hapticLib.c, 55

hapticLib.h, 61

INDIRECT
impact.m, 91
id
haptor_desc, 44
Impact
hl_patterns.h, 86
impact
status_param, 48
user_param, 50
impact.h
Aluminum, 99
Fast, 100
Normal, 99
Rubber, 99
Slow, 99
Wood, 99
impact.c
aluminumImpactPattern, 96
impactContinuator, 95
impactPatternGenerator, 95
rubberImpactPattern, 97
woodImpactPattern, 97
impact.h
ImpactMaterial, 99
ImpactVelocity, 99
impact.m
ACTION, 90
CONTRACT, 90
copy, 90
Copyright, 90
DIRECT, 90
granted, 91
INDIRECT, 91
modify, 91
PROFITS, 91
SPECIAL, 91
USE, 91
use, 91
impactContinuator
impact.c, 95
ImpactMaterial
impact.h, 99
impactPatternGenerator
hl_patterns.c, 80
impact.c, 95
impactStatusParameters, 45
progress, 45
impactUserParameters, 45
material, 46
velocity, 46
ImpactVelocity
impact.h, 99
increment
genericUserParameters, 43

MAX_HAPTORS
hl_STM32VLDISCOVERY.h, 110
MAX_PATTERNS
hl_patterns.h, 84
material
impactUserParameters, 46
max_duty
haptor_desc, 44
min_duty
haptor_desc, 44
modify
impact.m, 91

name
pattern_desc, 47
nextHaptor
haptor_desc, 44
Normal
impact.h, 99
Null
hl_patterns.h, 86
Num_Patterns_Available
hl_patterns.h, 86

PROFITS
impact.m, 91
pattern_continuator
hl_patterns.h, 84
pattern_desc, 46
activeHaptorList, 47
continuator, 47
hl_patterns.h, 84
name, 47
statusParams, 47
userParams, 47
pattern_initiator
hl_patterns.h, 84
pattern_name
hl_patterns.h, 85
patternMap
hapticLib.c, 55
hl_patterns.c, 81
patternScheduler
hl_patterns.c, 80
hl_patterns.h, 86
progress
impactStatusParameters, 45

RCC_Configuration
hl_STM32VLDISCOVERY.c, 105
hl_STM32VLDISCOVERY.h, 111
rightValue
generic.h, 76
Rubber
impact.h, 99
rubberImpactPattern
impact.c, 97

SPECIAL
impact.m, 91
STM32F10X_MD_VL

hl_STM32VLDISCOVERY.h, 110
STM32VLDISCOVERY
hapticLib.h, 58
send_char
hl_debug.h, 66
send_int
hl_debug.c, 63
hl_debug.h, 66
send_string
hl_debug.c, 63
hl_debug.h, 66
Slow
impact.h, 99
smallIncrement
generic.h, 76
status_param, 47
constant, 48
generic, 48
hl_patterns.h, 85
impact, 48
test, 48
statusParams
pattern_desc, 47
SysTick_Handler
hl_STM32VLDISCOVERY.c, 106
hl_STM32VLDISCOVERY.h, 112
SystemDesc
hapticLib.c, 55
hl_patterns.c, 82
hl_STM32VLDISCOVERY.c, 108

TIM_Channel_DutyChanger
hl_STM32VLDISCOVERY.c, 106
hl_STM32VLDISCOVERY.h, 113
TIM_Channel_Enable
hl_STM32VLDISCOVERY.c, 107
hl_STM32VLDISCOVERY.h, 113
TIM_Configuration
hl_STM32VLDISCOVERY.c, 107
hl_STM32VLDISCOVERY.h, 114
Test
hl_patterns.h, 86
test
status_param, 48
test.c
testContinuator, 101
testPatternGenerator, 101
testContinuator
test.c, 101
testPatternGenerator
hl_patterns.c, 81
test.c, 101
testStatusParameters, 48
duty, 49
flag, 49
TimingDelay
hl_STM32VLDISCOVERY.c, 109

USE
impact.m, 91
use
impact.m, 91
user_param, 49
constant, 50
generic, 50
hl_patterns.h, 85
impact, 50
userParams
pattern_desc, 47

velocity
impactUserParameters, 46

Wood
impact.h, 99
woodImpactPattern
impact.c, 97