Hash driver Documentation

Release 0.6.0

ANSSI

Apr 28, 2020

TABLE OF CONTENTS

This library is an implementation of the STM32F4 Hash cryptographic IP driver.

It provide an abstraction of the Hash device interactions through high level API in order to request hashing of any content, using the various hash methods supported by the HASH coprocessor of the STM32F4 SoC.

CHAPTER

ONE

ABOUT THE HASH DRIVER

1.1 Principles

The Hash cryptographic coprocessor provides a hardware implementation of various hash algorithms to support hash computation of input data, as well as HMAC variants of these algorithms. This device is an autonomous device, using a dedicated memory mapping. On STM32F4xx devices, the HASH device support the following hash algorithms:

- MD5
- SHA1
- SHA224 (with and without HMAC mode)
- SHA256 (with and without HMAC mode)

CHAPTER

TWO

ABOUT THE HASH DRIVER API

2.1 Initializing the hash driver

Initializing the hash driver is done with the following API:

```
#include "libhash.h"

typedef enum {
    HASH_TRANS_NODMA,
    HASh_TRANS_DMA,
} hash_transfert_mode_t;

typedef enum {
    HASH_MAP_AUTO,
    HASH_MAP_VOLUNTARY
} hash_map_mode_t;

typedef enum {
    HASH_POLL_MODE,
    HASH_IT_DIGEST_COMPLETE,
} hash_dev_mode_t;

typedef enum {
    HASH_MD5,
    HASH_SHA1,
    HASH_SHA224,
    HASH_SHA256,
    HASH_HMAC_SHA1,
    HASH_HMAC_SHA224,
    HASH_HMAC_SHA256,
} hash_algo_t;

int hash_early_init(hash_transfert_mode_t transfert_mode,
                    hash_map_mode_t       map_mode,
                    hash_dev_mode_t       dev_mode);

int hash_init(cb_endofdigest eodigest_callback, cb_endofdma eodma_callback, hash_algo_
↪t algo);
```

2.1.1 About early init

The hash driver early initialization must be executed before the end of the task initialization phase (see EwoK kernel API). This initialization declares the hash device against the kernel at boot time.

The early init function arguments are the following:

- *transfert_mode*: specifies the data buffer access from the HASH device. It may be a direct copy from the processor (HASH_TRANS_NODMA) or through the HASH DMA channel (HASH_TRANS_DMA), making the HASH device autonomous for the successive data access loops.
- *map_mode*: the HASH device mapping mode. When using HASH_MAP_AUTO, the device is automatically mapped at the end of the initialization phase of the task and for the complete task life-cycle. To support multiple devices in the same time without exhausting the MPU regions, it is possible to request on-demand only memory map, using HASH_MAP_VOLUNTARY option. In this latter case, the task is responsible for mapping and unmapping the device before and after each device usage.
- *dev_mode*: the way the device handles the hash calculation termination. This can be through the status register check (when using HASH_POLL_MODE) or through a dedicated interrupt (when using HASH_IT_DIGEST_COMPLETE).

All these information set the global device behavior and impact the device registration step. There is no impact on the hash algorithm choice at this time.

2.1.2 About init

The init step **does map** the HASH device if it is not. If HASH_MAP_VOLUNTARY is chosen, be sure to handle the unmap step after the HASH device usage.

> **Danger:** This constraint for voluntary mapped devices is specific to the init step. All other functions of the HASH drvier API require the device mapping and unmapping to be handled by the task

At initialization time, the task has to declare the following:

- *eodigest_callback*: the callback that is executed at the end of the current digest calculation
- *eodma_callback*: in DMA mode only, specify the DMA callback that is executed at the end of the DMA transfer. This callback can be NULL, even in DMA mode (in this case no handler is executed when the DMA transfer is over)
- *algo*: the HASH algorithm that has to be configured for all the following steps

---

Hint: It is possible to change the HASH device current algorithm by re-executing the *hash_init()* function

---

2.2 Mapping and unmapping the HASH device

When declaring the HASH device in HASH_MAP_VOLUNTARY mode, the HASH device has to be mapped and unmapped depending on the task memory constraints. The HASH driver provides easy-to-use API for this:

```
#include "libhash.h"

int hash_map(void);
int hash_unmap(void);
```

> **Danger:** Use these API only when declaring the device in HASH_MAP_VOLUNTARY mode. The kernel refuses to (un)map a device that has been declared in HASH_MAP_AUTO mode, making these functions fail

2.3 Calculating a digest

When the HASH device is configured, it is possible to directly request a digest computation from it. This is done with the following API:

```
#include "libhash.h"

typedef enum {
    HASH_REQ_IN_PROGRESS,
    HASH_REQ_LAST
} hash_req_type_t;

int hash_request(hash_req_type_t type, uint32_t addr, uint32_t size);
```

A digest computation can be done using successive requests. Although, the HASH device must be informed that these successive computations are a part of a single digest computation. To support such a behavior, the *hash_request()* function handles a *type* argument, which specifies if the current digest computation is a part of a larger one or the last of a computation sequence. In this latter case, the HASH device finishing the computation with a dedicated pass (including padding management, depending on the HASH algorithm properties) and set its internal registers with the calculated value.

Warning: When the last request is sent to the HASH device, sending a new HASH_REQ_IN_PROGRESS request reset the hash calculation as if the previous one has been finished. Be sure to get back the digest first

Warning: Because of the HASH coprocessor hardware limitations, only the last block of a digest computation is allowed to be word (32 bits) unaligned, and a dedicated padding procedure is performed for this specific last block. This means that all the hash updates performed before the last block must be word (32 bits) aligned.

Requesting a complete digest computation is a sequence between *hash_request()* calls and end of digest callback execution, finishing with a HASH_REQ_LAST request.

2.4 Getting back the digest

When the last data chunk has been sent to the HASH device and the digest computed (i.e. the end of digest callback has been triggered), the digest can be read from the device. This is done using the following API:

```
#include "libhash.h"

int hash_get_digest(uint8_t *digest, uint32_t digest_size, hash_algo_t algo);
```

This function is using the following arguments:

- digest: the output digest buffer, that needs to be previously allocated
- digest_size: the requested digest size. This size is known as the hash algorithm has been previously chosen
- algo: the hash algorithm that was used during the digest computation. The HASH driver has no effective memory and needs this information to be provided again

CHAPTER

THREE

HASH DRIVER FAQ

3.1 Is the Hash driver thread safe?

No, because the device is not. Beware to serialize correctly device requests to avoid any problems.

3.2 Can I request digests from unmapped data when using DMA?

Yes and no. It is possible to use DMA to let the HASH device access the data directly instead of copying them into the input register, but the DMA configuration is controlled by the kernel. You need at least to use DMA SHM EwoK mechanism with another task to request digest calculation from this task memory content.