Drviso7816 driver Documentation

Release 0.9.0

ANSSI

Apr 28, 2020

TABLE OF CONTENTS

1 About the ISO7816 driver **3**
- 1.1 Principles . . . 3

2 About the ISO7816 driver API **5**
- 2.1 Initialization functions . . . 5
- 2.2 Vcc and RST handling . . . 5
- 2.3 I/O line read and write primitives . . . 5
- 2.4 Handling ETU and frequency primitives . . . 6
- 2.5 Time measurement . . . 6
- 2.6 Card insertion detection . . . 6

3 ISO7816 driver FAQ **9**
- 3.1 Is the ISO7816 driver self-contained? . . . 9
- 3.2 Is using USART the only way to handle ISO7816? . . . 9
- 3.3 Why this driver does not make use of USART DMA? . . . 9

This driver is an implementation of the STM32F4 ISO7816 IP (USART in SMARTCARD mode). See the ISO7816-3 standard for insights about this protocol.

CHAPTER

ONE

ABOUT THE ISO7816 DRIVER

1.1 Principles

The ISO7816 driver implements the low level layer of the ISO7816-3 protocol. It configures the SMARTCARD mode of the STM32F4 MCUs USARTs (see the datasheet of the MCU for more information about this mode). The driver also implements the card detection GPIO monitoring, and executes a registered callback when the smart card is inserted or removed.

CHAPTER

TWO

ABOUT THE ISO7816 DRIVER API

The API exposed by the driver is quite simple and mainly exposes:

- Initialization functions
- Primitives to handle Vcc and RST lines (Vcc and RST are mandatory pins of ISO7816)
- Primitives to read and write bytes on the I/O line
- Primitives to set the current USART baudrate depending on an asked ETU and asked frequency
- Primitives to handle time measurement (for polling and dealing with ISO781 timeouts)
- Primitives to handle card insertion detection

Note: The notions of ETU (Elementary Time Unit) and ISO7816 baudrate are defined in the ISO7816-3 ISO standard. Please refer to it for definitions of these concepts

2.1 Initialization functions

The initialization functions are the following:

```
int platform_smartcard_early_init(drv7816_map_mode_t map_mode);
int platform_smartcard_init(void);
void platform_smartcard_reinit(void);
```

The early init function declares the USART to be used in SMARTCARD mode, and the other functions (re)initialize all the necessary variables.

2.2 Vcc and RST handling

Quite straightforward APIs handle the Vcc and RST (Reset) GPIOs:

```
void platform_set_smartcard_vcc(uint8_t val);
void platform_set_smartcard_rst(uint8_t val);
```

2.3 I/O line read and write primitives

The two main exposed functions to read and write bytes on the I/O line are:

```
int platform_SC_getc(uint8_t *c,
                     uint32_t timeout __attribute__((unused)),
                     uint8_t reset __attribute__((unused)));

int platform_SC_putc(uint8_t c,
                     uint32_t timeout __attribute__((unused)),
                     uint8_t reset);
```

The API is quite self-explanatory: `platform_SC_getc` writes the received character in its argument `uint8_t *c`, `platform_SC_putc` pushes the `uint8_t c` byte on the I/O line. These two function are non-blocking: they return -1 in case of failure, and 0 in case of success (i.e. the byte has been properly received or sent).

Note: The `platform_SC_getc` and `platform_SC_putc` have unused arguments. These arguments are here for API compatibility and future use

---

Another function is used for the bytes send/receive primitive: :: int platform_SC_set_direct_conv(void); int platform_SC_set_inverse_conv(void);

This function inverses the convention of the bits received on the line (see the ISO7816 notions of direct and inverse conventions).

Finally, the following function:

```
platform_SC_flush(void);
```

is used to flush the internal buffers of the ISO7816 driver. It must be called whenever a software reset or automaton reinitialization is performed.

2.4 Handling ETU and frequency primitives

The following APU:

```
int platform_SC_adapt_clocks(uint32_t *etu, uint32_t *frequency);
```

adapts the low-level USART baudrate and clocks according to the asked ETU in `uint32_t \*etu` and the asked frequency in `uint32_t \*frequency`. Since all the ETU and frequency are not attainable, these arguments are updated with the chosen values according to a `best fit` algorithm.

2.5 Time measurement

In order to handle ISO7816 timeouts, the driver exposes a time measurement with microseconds precision:

```
uint64_t platform_get_microseconds_ticks(void);
```

This is merely a wrapper to the `sys_get_systick(&tick, PREC_MICRO)` syscall.

2.6 Card insertion detection

A simple API is used to detect if a smart card is present or not:

```
uint8_t platform_is_smartcard_inserted(void);
```

It returns 0 if no smart card is present, and non zero if a smart card is present. This can be used in polling mode. If the user wants to use asynchronous detection, a callback registration API is provided:

```
void platform_smartcard_register_user_handler_action(void (*action)(void));
```

The user provides a `void (*action)(void)` handler that is called whenever the GPIO handling the smart card detection changes its state.

Note: LEDs toggling is also present in the driver (but not exposed in the API) for user interactions in order to show card presence and absence as well as card activity.

Finally, there is an API to be called by upper layers when a smart card is detected as lost:

```
void platform_smartcard_lost(void)
```

this function helps the driver to reinitialize and flush elements, and eventually notify other drivers. It should be called when the upper layer libraries indeed detects a smart card loss.

CHAPTER

THREE

ISO7816 DRIVER FAQ

3.1 Is the ISO7816 driver self-contained?

No, the ISO7816 driver only exposes primitives to send and receive characters on an I/O line as defined by the ISO7816 standard. This driver also generates the associated clock.

3.2 Is using USART the only way to handle ISO7816?

No, the same primitives to send and receive bytes on an ISO7816 half-duplex I/O line can be implemented in a bitbanging flavor with GPIOs. However, for efficiency and performance reasons we have chosen to use the native USART acceleration to do this.

3.3 Why this driver does not make use of USART DMA?

The ISO7816 is a synchronous protocol. The half-duplex nature of the I/O line and the fact that the sender and the receiver are either exclusively sending or receiving makes polling a simple yet still efficient strategy. Moreover, the ISO7816 bus is rather slow (the maximum clock frequency authorized by the standard is 20 MHz, and real life smart cards usually support up to 10 MHz): polling is not really a deal breaker, and using DMA would not drastically improve performance (compared to faster buses).