Spi driver Documentation

Release 0.8.0

ANSSI

Oct 10, 2019

TABLE OF CONTENTS

This library is an implementation of the STM32F4 Serial Peripheral Interface (SPI).

It provide an abstraction of the SPI device interactions through high level API and allows multiple drivers to share a given SPI line.

CHAPTER

ONE

SPI BUS AND DRIVER PRINCIPLE

1.1 Principles

A SPI interface is a full and half duplex synchronous communication interface widely used in embedded systems. This communication bus is used for various onboard devices such as screens, touchpad, memories, and so on.

The STM32F4xx support up to 4 SPI interfaces (depending on the way the SoC GPIOs are configured). SPI devices support direct and DMA-accelerated transfers.

CHAPTER

TWO

THE SPI DRIVER API

2.1 Initializing the SPI driver

Initializing the spi driver is done with the following API

```
#include "libspi.h"

#define    SPI_BAUDRATE_375KHZ  7
#define    SPI_BAUDRATE_750KHZ  6
#define    SPI_BAUDRATE_1500KHZ 5
#define    SPI_BAUDRATE_3MHZ    4
#define    SPI_BAUDRATE_6MHZ    3
#define    SPI_BAUDRATE_12MHZ   2
#define    SPI_BAUDRATE_24MHZ   1
#define    SPI_BAUDRATE_48MHZ   0

uint8_t spiX_early_init();
uint8_t spiX_init(uint8_t baudrate);
```

Caution: Each SPI function is named after the current SPI device identifier. For e.g., when using SPI2, each function is prefixed with `spi2_`. This mechanism is based on macros and permit to unify the SPI functions implementations

2.1.1 About early init

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

Danger: The existance of the spiX_early_init() function depends on the SPI driver configuration. In the menuconfig, select which SPI blocks should be supported by the SPI driver

2.1.2 About init

The init step initialize the SPI X bus with basic necessary configuration to be up and running. The init function handle the following argument:

- *baudrate*: The SPI bus baudrate, through its canonical named defined above.

Caution: Depending on the SoC, some SPI bus may not support the SPI_BAUDRATE_48MHZ, depending on the SPI bus AHB/APB master bus connectivity

Hint: If the SPI device does not support this baudrate, spiX_set_baudrate() returns MBED_ERROR_TOOBIG

2.2 Enabling and disabling the SPI

The SPI bus is not enable until the task explicitly enable it. When modifying the SPI configuration, the task needs to disable the SPI. To do these two kind of actions, the SPI driver provides the following API

```
#include "libspi.h"

void spiX_enable();
void spiX_disable();
```

spiX_disable() disable the SPI bus. *spiX_enable()* enable the SPI bus.

2.3 Getting SPI bus state

The SPI bus may be shared between multiple board devices. Though, devices may require different SPI configuration, like for example, the SPI baudrate.

Caution: EwoK and the SPI driver allow to use multiple devices on the same SPI bus, but does not permit to share the SPI bus between two tasks

To switch between multiple devices, the SPI driver provides the following API

```
#include "libspi.h"

int          spiX_is_busy();
mbed_error_t spiX_set_baudrate(uint8_t baudrate);
uint8_t      spiX_get_baudrate(void);
```

spiX_is_busy() returns non-zero value if the SPI bus is currently exchanging data. It returns 0 if the SPI bus is idle and its FIFOs are empty.

spiX_get_baudrate() returns the currently set baudrate, conformed to the canonical baudrate definitions set above.

spiX_set_baudrate() change the current baudrate of the SPI. It takes as parameter the canonical baudrate defined above.

Danger: Changing the baudrate must be done with the SPI bus disabled

When handling two independent peripherals on the same bus, a typical SPI handling would look like

```
#define PERIPH_ONE_BAUDRATE SPI_BAUDRATE_1500KHZ
#define PERIPH_TWO_BAUDRATE SPI_BAUDRATE_12MHZ
```

(continues on next page)

(continued from previous page)

```
/* sporadic events on peripheral one, taking the
 * SPI bus for short times (for e.g. on EXTI events)
 */
void handling_peripheral_one(void)
{
   uint8_t br = spi1_get_baudrate();
   /* wait for potential current data flow to finish */
   while (spi1_is_busy());

   spi1_disable();
   spi1_set_baudrate(PERIPH_ONE_BAUDRATE);
   spi1_enable();
   /* handling periph one actions */

   /* reconfigure the SPI bus for the other peripheral */
   spi1_disable();
   spi1_set_baudrate(PERIPH_ONE_BAUDRATE);
   spi1_enable();
}

/* more usual events on peripheral two, keeping the
 * SPI bus in the background (e.g. DMA based peripheral)
 */
void handling_peripheral_two(void)
{
   /* handling periph two actions */
}
```

2.4 Communicating with SPI peripherals

The basic way to communicate with SPI peripheral is through direct transfers.

This is done using the following API

```
#include "libspi.h"

uint8_t spiX_master_send_byte_sync(uint8_t data);
uint8_t spiX_master_recv_byte_sync(void);
```

The SPI bus is a serial char-based synchronous communication bus.

When sending data on the bus, another data is potentially received in the same time. This data is then returned by the *spiX_master_send_byte_sync()* function.

When receiving data, the task can use the *spiX_master_recv_byte_sync()* function. As the SPI bus always communicate in a full duplex mode in this driver configuration, this function calls *spiX_master_send_byte_sync(0x42)*. This function is only a sugar function for the user code.

SPI buses can also support DMA-based transfers. The SPI driver supports circular DMA handling only by now. This is typically used for screens. Circular DMA based data transfer is done using the following API

```
#include "libspi.h"

void spi_master_send_bytes_async_circular(uint8_t *data, uint32_t datalen, uint32_t␣
↪totallen);
```

Circular DMA based transfer uses the following arguments:

- **data**: the input data buffer
- **datalen**: the input data buffer length (in bytes)
- **totallen**: the total length to transfer (which may include multiple circular transfer of the same buffer).

CHAPTER

THREE

SPI DRIVER FAQ

3.1 Can the flash driver handle multiple SPI devices in the same time

It should. Though, we have not hardly tested it as Wookey is using a single SPI bus per board.