Hardware Serial Peripheral Interface

|  |  |
| --- | --- |
| Status : working, ~~validated~~, ~~documented~~ | 23/11/2016 |

# Introduction

The ESP8266 has two independent SPI (Serial Peripheral Interface). One is used to connect an external Flash memory for program storage, the other is the main board-level interface between the ESP8266 and hardware. It is through this interface that the most WiFi bandwidth can be exploited.

The second interface is called “HSPI” (possibly for Host SPI).

## State of the library

This library is currently usable but not fully validated.

## Origin of the library

The source files for this library come from an example project in the Unofficial SDK and may be related to a project by someone called “Metal Phreak”. This information will be updated when the origin of the code is ascertained.

This original library counts two files: “hspi.h” and “hspi.c”. It was originally intended for Espressif’s own RTOS.

Nefastor added several modifications to make it compatible with FreeRTOS. This includes the creation of a second header, “spi\_registers\_2.h”, out of Espressif code. This header is used internally by the library and does not need to be included in source files using the HSPI library.

# Using the library

# API

## hspi\_init

|  |  |
| --- | --- |
| HSPI Initialization : pin multiplexing and generic settings | |
| Status | Validated |
| Returns | - |
| Argument | - |

This function should be called first, followed immediately by a call to **hspi\_clock**. It sets the proper pin multiplexing for the HSPI, as well as bit transmission order (low to high).

It enables SSEL setup and hold, to improve reliability at high data rates (though the exact impact of that feature hasn’t been measured)

SPI mode (phase and polarity) is left to default value.

Importantly, this function disables all phases of the HSPI’s SPI transaction sequencer, leaving only the data phase enabled. This is to allow for directly using the optimized 8 / 16 / 32 bits write functions, intended for LCD displays.

## hspi\_gpio

|  |  |
| --- | --- |
| HSPI Initialization : pin multiplexing | |
| Status | Validated |
| Returns | - |
| Argument | - |

This function only takes care of the pin multiplexing. It is called by **hspi\_init**. The HSPI pins are:

* GPIO12 as MISO (data line from slave to master)
* GPIO13 as MOSI (data line from master to slave)
* GPIO14 as SCK (serial clock)
* GPIO15 as SSEL (slave select)

Those are ESP8266 pin names, actual pin names on your board may vary: refer to its schematics.

## hspi\_mode

|  |  |
| --- | --- |
| HSPI Initialization : set the bit rate | |
| Status | Validated |
| Returns | - |
| Argument | Phase : data valid on SCK leading edge (0) or trailing edge (1) |
| Argument | Polarity : SCK can be active high (0) or active low (1) |

The terms “leading” and “trailing” edge are used because the actual edge (rising or falling) depends on the whether SCK is active high or low.

The HSPI defaults in mode zero (SCK active high, data valid on leading edge).

## hspi\_clock

|  |  |
| --- | --- |
| HSPI Initialization : set the bit rate | |
| Status | Validated |
| Returns | - |
| Argument | Pre-scaler ratio (0 to 8192) |

This function’s argument can be either zero or a positive integer.

Zero enables 80 MHz SCK (the system clock is fed directly to SCK).

Otherwise SCK frequency is 40 MHz divided by the argument. Thus:

* Call **hspi\_clock(1)** for 40 MHz operation
* Call **hspi\_clock(4)** for 10 MHz operation
* Call **hspi\_clock(8000)** for 5 KHz operation

This function is designed to produce a duty rate of 50%.

## hspi\_wait\_ready

|  |  |
| --- | --- |
| Wait for transfer to complete | |
| Status | Validated |
| Returns | - |
| Argument | - |

This blocking call will only return when the HSPI is done sending the data in its transmission buffer.

## hspi\_busy

|  |  |
| --- | --- |
| Macro : returns non-zero when the HSPI is busy sending data | |
| Status | Validated |
| Returns | 0 if HSPI is ready, non-0 otherwise |
| Argument | - |

This macro is used by **hspi\_wait\_ready** and should be used directly in multithreaded programs.

## hspi\_send\_uint8

|  |  |
| --- | --- |
| Send a single byte | |
| Status | Validated |
| Returns | - |
| Argument | Byte to transmit over SPI (uint8\_t) |

Speed-optimized transaction limited to the bare minimum code for sending a single byte. This function is mostly useful with display controllers.

## hspi\_send\_uint16

|  |  |
| --- | --- |
| Send a single word | |
| Status | Validated |
| Returns | - |
| Argument | Word to transmit over SPI (uint16\_t) |

Speed-optimized transaction limited to the bare minimum code for sending a single 16-bit word. This function is mostly useful with display controllers.

## hspi\_send\_uint32

|  |  |
| --- | --- |
| Send a single DWord | |
| Status | Validated |
| Returns | - |
| Argument | DWord to transmit over SPI (uint32\_t) |

Speed-optimized transaction limited to the bare minimum code for sending a single 32-bit word. This function is mostly useful with display controllers.

## hspi\_send\_data

|  |  |
| --- | --- |
| Send 1 to 64 bytes from an array | |
| Status | Validated |
| Returns | - |
| Argument | Array of bytes to transmit (uint8\_t\*) |
| Argument | Number of bytes to transmit (uint8\_t) |

Speed-optimized transaction for sending a string of up to 64 bytes.

## TO DO – SAME FUNCTION FOR BURST READ

## hspi\_transaction

|  |  |
| --- | --- |
| Send 1 to 64 bytes from an array | |
| Status | Validated |
| Returns |  |
| Argument |  |
| Argument |  |

## Setup Macros