BLE 1.4 SPI Driver Design

Version 1.x (Draft)

Document Number: TBD

TABLE OF CONTENTS

TABLE OF FIGURES

1. Functional Overview

The SPI protocol consists of 4 lines – SCLK, DATA, MOSI and MISO. In this system consisting of SPI Master and SPI Slave, there is a requirement for handshake between the master and the slave. Hence, an SRDY pin

is used. This is a GPIO input to the master. So, effectively the SPI uses 5 pins to communicate between SPI Master and Slave.

The current implementation is based on combined MRDY and CSn. The code is located at –

http://sancsvn01.sanc.design.ti.com/svn/r1/branches/BLE_SPI_test

2. Definitions, Abbreviations, Acronyms

Term	Definition
API	Application Programming Interface
MRDY	Master-Ready: an indication and/or wakeup event to a SPI slave.
OSAL	Operating System Abstraction Layer
PM	Power-Mode – a low power or sleep mode to reduce power consumption.
SOC	System on a Chip
SPI	Synchronous Peripheral Interface – a synchronous mode of a USART
SRDY	Slave-Ready: an indication and/or wakeup event to a SPI master.
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
RBA	UART to SPI bridge

3. Revision History

Date	Document Revision	Sections Effected	Summary of Changes
07/25/2012	0.1.0	All	New document.
08/09/2012	0.9.0	All of 8 & 10.	Amend for review comments.
08/10/2012	0.9.1		Amend for review comments.
09/12/2012	0.9.2	9	Fixed SRDY pin assignments.
03/25/2013	1.0	All	Renamed as BLE 1.3 SPI driver design, added brief overview of the SPI driver and low level details
07/24/2013	1.x	All	Renamed as BLE 1.4 SPI driver design. Added summary of modifications to the SPI driver

4. SPI Interface

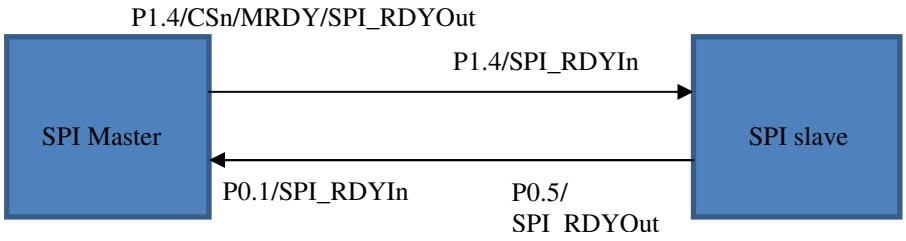

Figure 1 :MRDY and SRDY signals

4.1 SPI lines

4.1.1 MISO

Master-In/Slave-Out is a unidirectional line on which the SPI slave makes data bits available and which are clocked out by the master driving the SCLK. The slave shall tri-state this line when its CSn is not asserted.

4.1.2 MOSI

Master-Out/Slave-In is a unidirectional line on which the SPI master makes data bits available which are clocked out by the master driving the SCLK.

4.1.3 CSn/ MRDY

Chip-Select-Not is a unidirectional line driven by the master. The SPI Master currently uses CS line as MRDY. CS needs to be low during the entire duration of any SPI transaction during which the SPI Master is sending/receiving data and the SCLK is ON.

SPI Master must wait for SRDY to go low before starting the DMA TX.

4.1.4 SCLK

SPI Clock is the clock which simultaneously drives data on both the MISO and MOSI lines.

4.1.5 SRDY

The SRDY is an output GPIO pin from the SPI slave to the SPI Master. The SRDY is set to low whenever it receives a falling edge ISR due to CS/MRDY from the Master. If the SPI slave has nothing to send, SRDY will be set to high again.

If the slave has any data to send, the SRDY will stay low for the duration of the SPI transaction till the DMA TX End ISR is received on the SPI Slave.

4.2 SPI Packet

4.2.1 Packet Format

The SPI Packet format consists of 3 bytes in addition to the data payload of 1 to SPI_MAX_DAT_LEN bytes:

<SOF><LEN><1…SPI_MAX_DAT_LEN data bytes><FCS>

4.2.1.1 SOF

The Start-of-Frame marks the beginning of a SPI Packet and shall be 0xFE.

4.2.1.2 LEN

The Length byte indicates how many data bytes follow, and shall be a valid value between 1…SPI_MAX_DAT_LEN.

SPI_MAX_DAT_LEN is the maximum length of data bytes that can be transmitted by one SPI Packet. For the sake of allowing efficient uint8 math on the 8051 when indexing circular receive and transmit buffers, this value is arbitrarily set to 256 – 3 = 253.

4.2.1.3 FCS

The Frame-Check-Sequence is a simple byte-wise exclusive-OR of all of the data bytes and the LEN byte.

4.3 SPI Driver API

- HalUARTWriteSPI( ) is the invoked with parameters – buffer (pointer), sizeof(buffer) to write to the SPI
- HalUARTReadSPI( ) is used to read the data bytes from a received SPI packet. The parameters passed are pointer to the destination buffer and number of bytes to be read.
- HalUARTInitSPI() is the initialization function for SPI.
- HalUARTRxAvailSPI() returns the number of bytes in the RX Buffer for SPI.

4.4 SPI Protocol Illustration

An illustration of SPI slave transmit to SPI master –

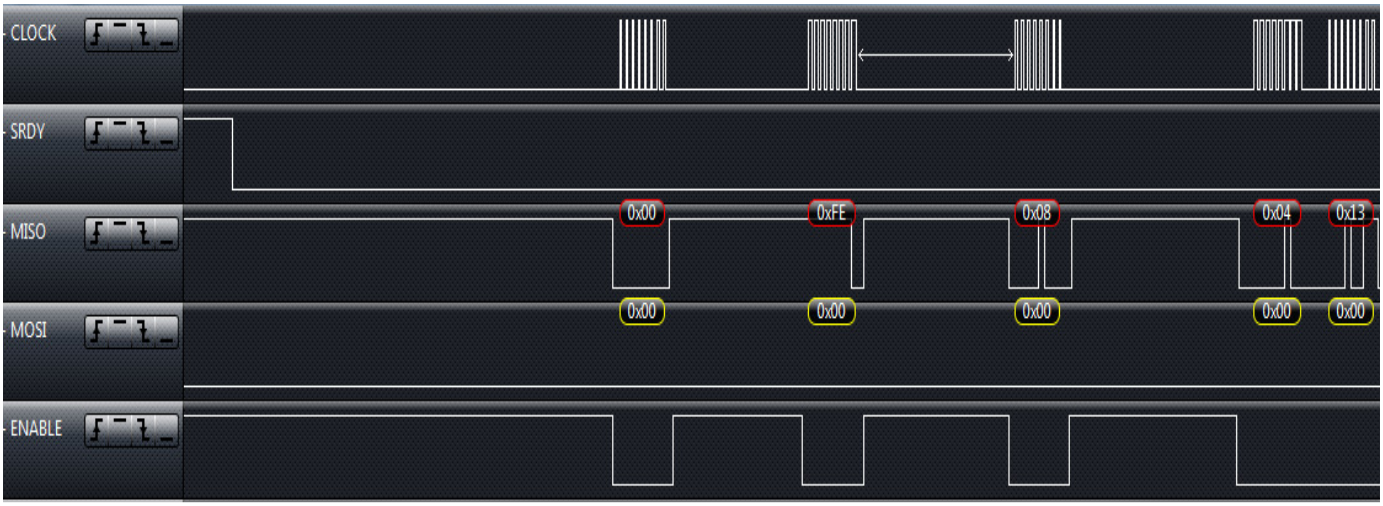

Figure 2 : SPI slave transmits data to SPI Master

The above figure illustrates that to send this SPI packet from SPI slave to SPI master –

< SOF, LEN, DATA, FCS > which is < 0xFE, 0x08, <BYTE1, BYTE2…, BYTEN>, <FCS>>

- SRDY goes low to indicate it has data to transmit. SPI slave is waiting for SCLK to send its data to the SPI master.
- The SPI Master polls for a new byte. If no new byte is found in its DMA Rx buffer, it sends a 0x00 out to the SPI slave for every iteration to get new bytes.
- SPI Slave has the frame ready to be transmitted. It sends out SOF (0xFE)
- MRDY/CS pulls high and again goes low, this time SPI slave sends the next byte – 0x08 LEN and then it sends 0x04 in the next SPI burst which is the OPCODE for HCI_EVENT
- Then a third SPI transaction takes place with SPI master clocking out 0x00s till it receives all the bytes indicated by the LEN field along with the last byte which has the FCS (XOR of all the bytes)
- Finally MRDY/CS goes high and SRDY also goes high.

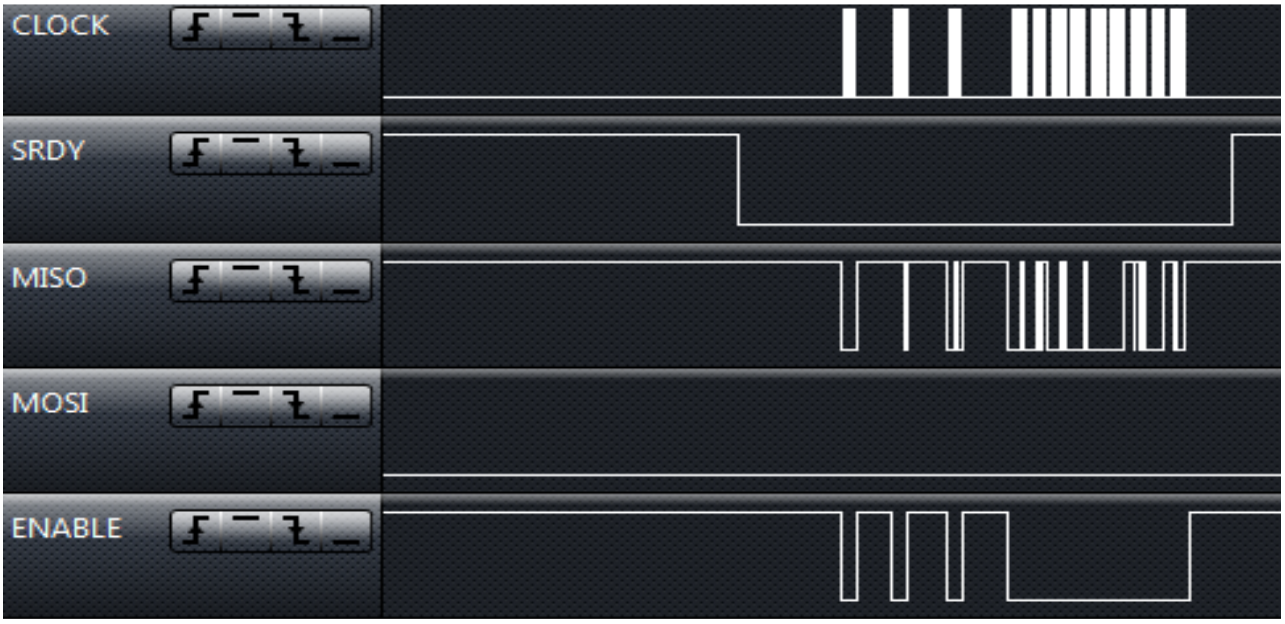

Figure 3: Complete SPI Slave to SPI Master transmit operation

An illustration of SPI Master sending to SPI slave-

Figure 4: SPI Master transmits data to SPI slave

Figure 5: Initial part of SPI Master TX

The above figure illustrates that to send this SPI packet from SPI Master to SPI slave –

< SOF, LEN, ….., FCS_value> which is < 0xFE, 0x1F, …….,.>

- MRDY/CS is held low by the SPI Master.
- SRDY goes low when it receives MRDY interrupt. SRDY is pulled high again.
- SRDY's interrupt on SPI Master triggers DMA TX from SPI Master to SPI slave and TX begins.
- Finally MRDY/CS goes high after SPI Master is done sending all the bytes to the SPI slave.

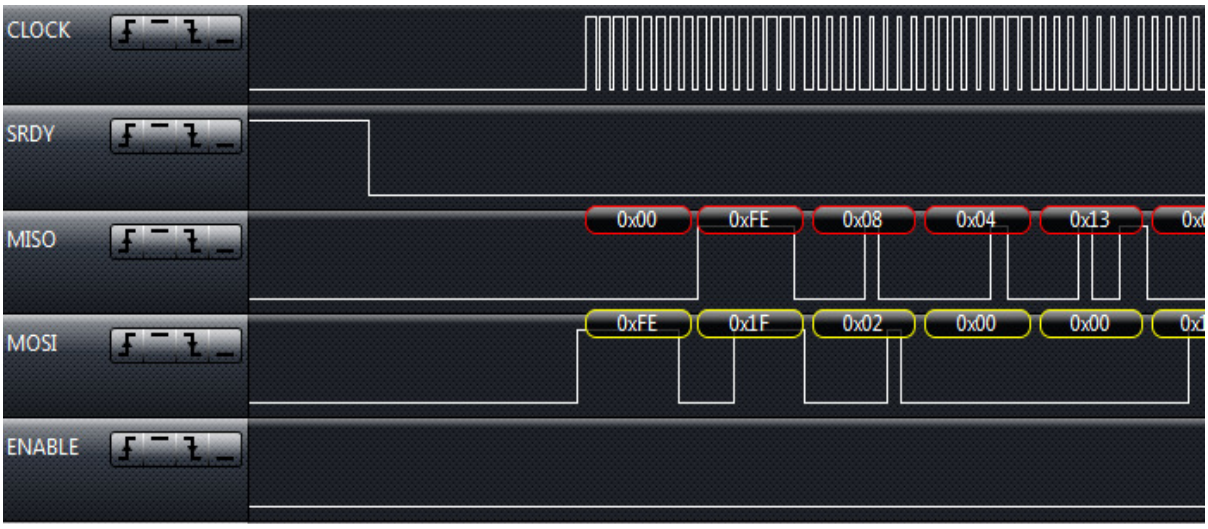

Figure 6: Full-duplex SPI TX/RX

The above figure illustrates simultaneous transmission from both SPI Master and SPI Slave –

- MRDY/CS is held low by the SPI Master since it has to transmit.
- SRDY goes low since slave has to transmit

- SRDY's interrupt on SPI Master triggers DMA TX from SPI Master to SPI slave and TX begins for both SPI Master as well as SPI slave.
- Finally MRDY/CS goes high after SPI Master is done sending all the bytes to the SPI slave.

5. Default SPI Configurations.

The 8051 SOC has 2 USARTS, each of which can be configured for SPI on two alternate pin-outs. The shaded regions are the configurations currently supported for SPI = 1 and SPI = 2

	USART0		**USART1**	
	`HAL_UART_SPI = 1`		`HAL_UART_SPI = 2`	
	Alt. 1	Alt. 2	Alt. 1	Alt. 2
MISO	P0.2	P1.4	P0.5	P1.7
MOSI	P0.3	P1.5	P0.4	P1.6
CSn	P0.4	P1.2	P0.2	P1.4
SCLK	P0.5	P1.3	P0.3	P1.5
SRDYOut from SPI Slave	P1.1			P0.5
SRDYIn for SPI Master	P0.1	P0.1	P0.1	P0.1

Figure 7: 8051 SOC USART Permutations.

6. Implementing a SPI Slave Application

The source code module which abstracts the SPI Protocol can usually be found in the HAL–Target or HAL-Target-Common sub-directory as _hal_uart_spi.c.

For HCITestApp – SPI Controller workspace, the following options were used –

```
CC2541
FAST_TX                         - Used only by cc2541
POWER_SAVING                    - Used only for POWER SAVING
xBLE_BADDR_FROM_FLASH
HAL_FLASH=FALSE
HAL_AES_DMA=TRUE
HAL_DMA=TRUE
HAL_UART=TRUE
HAL_UART_DMA=0
HAL_UART_ISR=0
HAL_UART_SPI=2                  - Can be set to either 1 or 2
HAL_KEY=FALSE
HAL_LCD=FALSE
HAL_LED=FALSE
xLL_COLLECT_METRICS
xHALNODEBUG
xDEBUG
xDEBUG_GPIO
xDEBUG_ENC
```

Apart from these settings, the NPI layer should have the correct UART port. The test branch has the modified NPI.h file which should be used.

Note: `HAL_SPI_QUEUED_TX` has to be set to `FALSE`

```
#define HAL_SPI_QUEUED_TX          FALSE - Defined inside SPI Driver file
```

7. Implementing a SPI Master Application

For implementing SPI Master use the following defines –

```
HAL_SPI_CH_RX=1
HAL_SPI_CH_TX=2
HAL_SPI_MASTER
HAL_UART_SPI=2                - Can be set to either 1 or 2
HAL_KEY=FALSE
HAL_LCD=FALSE
HAL_LED=FALSE
```

For the UART-to-SPI bridge, the following defines are needed in addition to the ones defined above –

```
HAL_SBL_BOOT_CODE_RBA
RBA_UART_TO_SPI
HAL_UART=TRUE
HAL_UART_DMA=1
HAL_UART_TX_BY_ISR=1
```

8. SPI Driver Design and Considerations

The SPI Driver is designed to make sure that the SPI Master is able to send data to the sleeping/non-sleeping SPI slave. The MRDY/CS input to the slave is sort of a wakeup signal to make sure that the SPI slave is awake and ready to receive data on SPI lines.

The two handshake signals used in the driver are –

- MRDY /CS
  The CS/MRDY on CC2541 is a GPIO used to indicate that MRDY is asserted/de-asserted to the SPI Slave. It is an output signal to the SPI slave. It stays low for the entire duration of the SPI Master transmission.

- SRDYOut
  The SRDYOut is an output from SPI slave to SPI Master to indicate that the SPI Slave is ready to send/receive depending on the type of operation. It stays low for the entire duration of the SPI slave transmission.

For implementing SPI Master and the SPI slave, there are a couple of considerations such as the ones listed below.

8.1 Clock frequency (SCLK)

For implementing SPI Master, the SCLK frequency can be set to a maximum of 4 MHz on CC2541. The clock polarity and phase are both set to 0 for this SPI driver. Transmission is MSB first and is set for byte sized transfers.

8.2 Full-duplex TX/RX

The SPI interface is typically full-duplex in its design because whenever a byte is clocked out, a byte gets clocked in. The SPI full-duplex in this document refers to the simultaneous transmission of data on MISO and MOSI lines. This feature makes the SPI TX/RX efficient.

8.3 End of transmit Interrupt

The DMA ISR TX done interrupt is used to pull the CS line high after the SPI Master is done sending data. In case of another MCU, this functionality needs to be made available in another ISR which correctly pulls CS/MRDY high at the end of transmit.

8.4 Timings

Figure 8: Time between CS low and SRDY pulse

The time gap between chip select/MRDY going low till the point the SRDY toggles from high to low and back to high ranges from 0.181ms to 1.2 ms. This value depends on what the SPI slave is processing, so it could be that the slave is already awake or it could be otherwise. The SPI slave can also be in the process of writing to the SPI when the CS/MRDY goes low. This would use the SPI more efficiently because the TX from both sides would happen almost simultaneously.

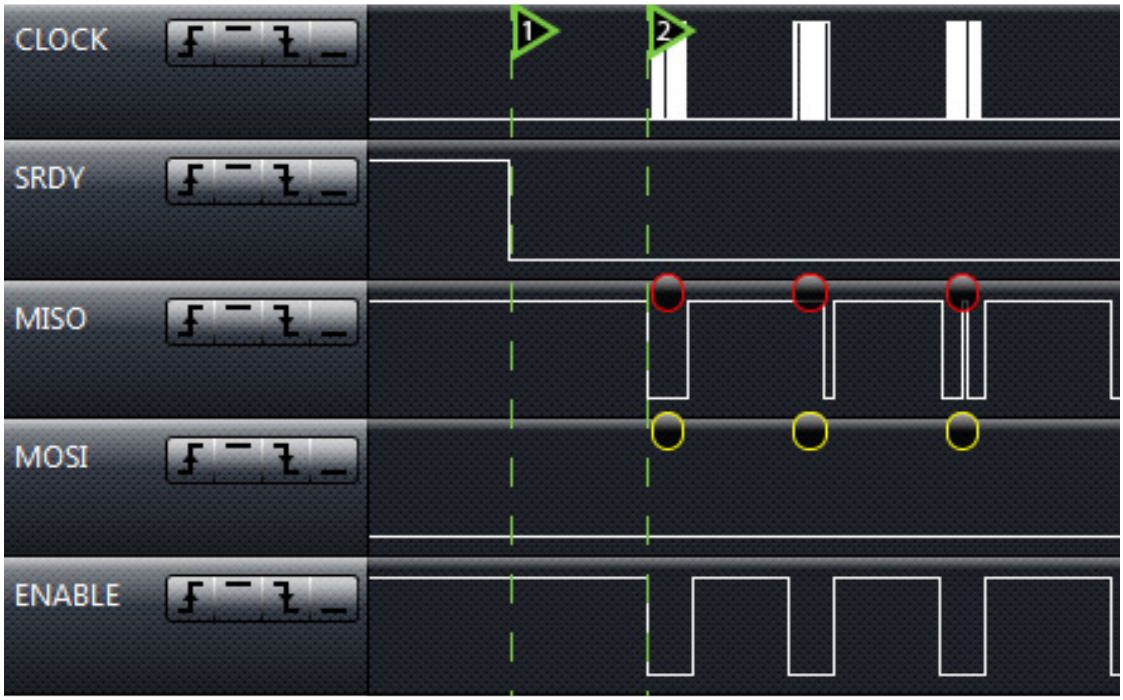

Figure 9: Time between SRDY low and CS pulse

The timing diagram shown above is for SPI Slave transmitting data to the SPI Master. The SPI Master polls the SPI slave one byte at a time to get the SOF, LEN bytes. It then clocks out the remaining data+FCS bytes as per the length byte it receives from the SPI slave.

Time gap between SRDY low to the first CS/MRDY pulse is typically between 4us to 7.83us. Effectively, the SRDY low would cause the SPI Master to actually start the SCLK when it tries to poll the SPI slave. Ideally this would happen without latency because the SPI poll would be executed every time the SPI Master runs the OSAL loop.

Figure 10: Full duplex SPI TX/RX example

The above figure shows a simultaneous SPI TX/RX. Both SPI Master and SPI Slave are sending/receiving data over SPI. In this case, MRDY/CS will be held low by the SPI Master for the duration of the operation. That is, the SPI Master would try to send its entire frame and also receive the bytes from the SPI slave before pulling MRDY/CS high again. In case the SPI slave has a frame which exceeds the size of the frame from the SPI Master, the SPI Master would keep clocking out bytes till SRDY goes high again to ensure that the entire frame from the SPI Slave is received.

In figure 10 above, for sending an ACL Data packet of size 26 bytes over SPI, the SPI Master takes approximately 67.8 us. From the time the CS/MRDY goes low to the time it goes high, the entire operation takes 0.18ms.