ECSE 426 – Lab 4

*Osama Abdulhadi (260361758)*

*Kevin Cadieux (260277248)*

*Lab Group Number 4*

# Abstract

# Problem Statement

# Theory and Hypotheses

# Implementation

## CC2500 Driver

### SPI Configuration

Communicating with the CC2500 wireless chip is done using the SPI protocol. For this reason, the main low level initialization task of the CC2500 driver consists of setting up one of the SPI modules available on either the STM32F407 or the STM32F429 board. The SPI settings differ depending on which STM board is being used. Table 1 describes the configuration for the F407 board, and Table 2 describes the configuration for the F429 board. Finally, Table 3 shows the configuration that is common to both boards.

Table 1 - SPI configuration for the F407 board.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Value** | **Justification** |
| SPI Module | SPI2 | SPI2 is not tied to any other function, such as the accelerometer. |
| SCLK Pin | B13 | SCLK for SPI2 is the alternate function of B13 |
| CSN Pin | B12 | CSN for SPI2 is the alternate function of B12 |
| MISO Pin | B14 | MISO for SPI2 is the alternate function of B14 |
| MOSI Pin | B15 | MOSI for SPI2 is the alternate function of B15 |
| Baud Rate Prescaler | 8 | Minimum required prescaler to obtain a safe frequency for burst mode SPI transfers. |
| Peripheral Bus | APB1 | SPI2 is connected to APB1. |

Table 2 - SPI configuration for the F429 board.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Value** | **Justification** |
| SPI Module | SPI4 | SPI4 is not tied to any other function, and is using free pins that are not tied to other modules such as the SDRAM or the LCD. |
| SCLK Pin | E2 | SCLK for SPI4 is the alternate function of E2 |
| CSN Pin | E4 | CSN for SPI4 is the alternate function of E4 |
| MISO Pin | E5 | MISO for SPI4 is the alternate function of E5 |
| MOSI Pin | E6 | MOSI for SPI4 is the alternate function of E6 |
| Baud Rate Prescaler | 16 | Minimum required prescaler to obtain a safe frequency for burst mode SPI transfers. |
| Peripheral Bus | APB2 | SPI4 is connected to APB2. |

Table 3 - Common SPI configuration for all boards.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Value** | **Justification** |
| Direction | Full Duplex | Required by the CC2500. |
| Data Size | 8 bits | Required by the CC2500. |
| Clock Polarity | Active Low | Required to satisfy the SPI timing constraints of the CC2500. |
| Clock Phase | 1 Edge | Required to satisfy the SPI timing constraints of the CC2500. |
| Slave Select Mode | Software | The slave select pin is managed by software, independently of the SPI module. |
| First Bit | MSB | Required by the CC2500. |
| CRC Polynomial | 7 | Default value. |
| SPI Mode | Master | The CC2500 is a slave. |

The selection of the SPI pins and busses was guided by the datasheet of each ST board (STMicroelectronics, 2014) (STMicroelectronics, 2012). The other SPI configuration parameters were chosen to satisfy the constraints established by the CC2500 datasheet (Texas Instruments, 2014).

Special care needed to be taken when selecting the baud rate prescaler, as the selected SPI modules are not always connected to the same peripheral bus. Also, the clock management unit of the ST boards can be configured to provide different clock speeds. For all these reasons, the peripheral bus on which the SPI module is connected may vary in clock speed. To avoid having the user specify the baud prescaler every time such changes are made, the CC2500 driver we developed dynamically chooses the baud rate prescaler based on the current clock speed settings written in the reset and clock control (RCC) registers of the board. The baud rate prescaler values shown in Table 1 and Table 2 are the values that correspond to our specific RCC settings.

In addition to configuring the SPI module, the individual pins used in SPI communication had to be configured as well. The SCLK, MOSI, and MISO pin configuration can be seen in Table 4. Since the slave select pin is not managed by the SPI module hardware but with software, it needed to be configured differently. Its configuration is shown in Table 5.

Table 4 - GPIO configuration for SPI pins.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Value** | **Justification** |
| Mode | Alternate Function | SPI |
| Speed | 50 MHz | Largely sufficient because the SPI clock will run at around 6.5 MHz. |
| Pull-Up/Pull-Down | Pull-Down | Input voltage is pulled down when no input is present. |

Table 5 - GPIO configuration for the slave select SPI pin.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Value** | **Justification** |
| Mode | Output | Slave select is managed by software so a general output GPIO port is needed. |
| Speed | 50 MHz | Largely sufficient because the SPI clock will run at around 6.5 MHz. |
| Output Type | Pull up / Pull down | Output needs to be pulled up to Vdd when the pin is set. |

### Programming Interface

Our CC2500 driver comes with a programming interface that allows the user to take advantage of every feature of the CC2500 chip.

#### Register and Command Strobe Macros

Every configuration register, status register, and command strobe can be named using the appropriate macro. All macros take the following form: CC2500\_<name of register or command strobe>\_<R or W (command strobes only)>. In the case of registers (configuration or status), the user only needs to provide its name (e.g. CC2500\_PKTLEN). For command strobes, the user needs to append the name with either \_R or \_W, depending on whether the read or the write version of the strobe is required (e.g. CC2500\_STX\_R).

#### Configuration Register Setting Macros

All possible settings or values for the configuration and status registers are accessible via macros. For example, to configure the PKTCTRL0 register to use CRC with variable length packets, one would write: CC2500\_PKTCTRL0\_CRC\_EN | CC2500\_PKTCTRL0\_Length\_VARIABLE. We refer the reader to the cc2500.h header file to see a list of these macros, and to the CC2500 datasheet to understand what they do (Texas Instruments, 2014).

#### Initialization Structure

For convenience, an initialization structure called CC2500\_InitTypeDef is made available. It contains a field for every configuration register of the CC2500 chip. This structure can be written to and passed to the CC2500\_WriteConfig function to set up all of the CC2500 registers at once. The fields of this structure can be initialized to their default values using the CC2500\_StructInit function.

#### Chip Status Structure

Most driver functions return an instance of the CC2500\_StatusTypeDef structure, which contains all the status information of the chip that was returned from the previous interaction with it. The information available is:

* Chip Ready bit.
* Current state.
* FIFO bytes available.

#### Driver Functions

The driver provides many functions to interact with the CC2500 chip. These functions are described in Table 6.

Table 6 - CC2500 driver functions.

|  |  |  |
| --- | --- | --- |
| **Name** | **Parameters** | **Description** |
| CC2500\_Init | None | Initializes SPI communication with the CC2500 chip. Must be called before anything else. |
| CC2500\_WriteConfig | CC2500\_InitTypeDef\* init\_struct | Writes all the configuration registers of the CC2500 chip based on the values found in <init\_struct>. |
| CC2500\_StructInit | CC2500\_InitTypeDef\* init\_struct | Initializes an initialization structure to the default values. |
| CC2500\_ReadConfigRegister | uint8\_t addr  uint8\_t\* destination | Reads the configuration register at the address specified in <addr>, and writes the content of this register in <destination>. |
| CC2500\_ReadStatusRegister | uint8\_t addr  uint8\_t\* destination | Reads the status register at the address specified in <addr>, and writes ton content of this register in <destination>. |
| CC2500\_ReadRxFIFO | uint8\_t\* destination,  uint8\_t nb\_bytes | Reads the number of bytes specified in nb\_bytes and writes them in <destination>. |
| CC2500\_ReadPATABLE | CC2500\_PATABLETypeDef\* patable | Reads the PA TABLE from the CC2500 chip and writes it in <patable>. |
| CC2500\_ReadConfiguration | CC2500\_InitTypeDef\* init\_struct | Reads all the configuration registers from the CC2500 chip and fills <init\_struct> with their values. |
| CC2500\_WriteConfigRegister | uint8\_t addr  uint8\_t byte | Writes <byte> into the configuration register specified by <addr>. |
| CC2500\_WriteTxFIFO | uint8\_t\* data,  uint8\_t nb\_bytes | Writes a number of bytes specified by <nb\_bytes> into the TX FIFO using the bytes pointed to by <data>. |
| CC2500\_WritePATABLE | CC2500\_PATABLETypeDef\* patable | Writes <patable> into the PA TABLE of the CC2500. |

All but the first 3 functions in this table return a CC2500\_StatusTypeDef structure, which contains the status of the chip. Also, the driver automatically opts for burst transfer SPI operations when it is advantageous to do so, such as during FIFO operations.

## Wireless Module

The wireless module sits on top of the CC2500 driver and provides higher level functions that encapsulate all interactions with the CC2500 chip. It provides support for sending and receiving packets without forcing the user to worry about the intricacies of the CC2500 chip, such as initializing the chip with the correct parameters, or configuring interrupts.

### CC2500 Configuration

Table 7 below shows how the CC2500 has been configured to obtain the desired behavior and packet format.

Table 7 - CC2500 configuration

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Value** | **Justification** |
| PATABLE[0] | 0xFF | Used for maximum output power. |
| FSCTRL1 | 0x0C |  |
| FSCTRL0 | 0x00 | FSCTRL values were provided by SmartRF Studio. |
| FREQ2 | 0x5D |  |
| FREQ1 | 0x93 |  |
| FREQ0 | 0xC5 | FREQ values were selected to obtain the desired frequency of 2,433,008 KHz. |
| MDMCFG4 | 0x0E |  |
| MDMCFG3 | 0x3B |  |
| MDMCFG2 | 0x73 |  |
| MDMCFG1 | 0xC2 |  |
| MDMCFG0 | 0xF8 | MDMCFG values were mostly provided by SmartRF Studio. A slight modification has been made to MDMCFG1 to activate forward error correction on the packets by setting bit 7. We found through experimentation that this greatly reduced packet loss due to error. |
| DEVIATN | 0x00 | Provided by SmartRF Studio. |
| FREND1 | 0xB6 |  |
| FREND0 | 0x10 | FREND values were provided by SmartRF Studio. |
| MCSM1 | 0x30 / 0x32 | 0x30 makes the chip go back to IDLE state after a packet has been sent or received.  0x32 has the same behavior except that the chip will stay TX after a packet has been sent. This is used when we wish to send a packet burst without going to IDLE mode between each packet. We have found through experimentation that this significantly speeds up burst transmissions. |
| MCSM0 | 0x18 | Instructs the chip to use automatic calibration when going from IDLE state to RX or TX states. |
| FOCCFG | 0x1D | Provided by SmartRF Studio. |
| BSCFG | 0x1C | Provided by SmartRF Studio. |
| AGCTRL2 | 0xC7 |  |
| AGCTRL1 | 0x40 |  |
| AGCTRL0 | 0xB0 | AGCTRL values were provided by SmartRF Studio. |
| FSCAL3 | 0xEA |  |
| FSCAL2 | 0x0A |  |
| FSCAL1 | 0x00 |  |
| FSCAL0 | 0x19 | FSCAL values were provided by SmartRF Studio. |
| FIFOTHR | 0x0F | Instructs the chip to use a threshold of 1 for the TX FIFO and a threshold of 64 for the RX FIFO. This value has been chosen to facilitate interrupt generation when a packet or packet burst has been sent (See IOCFG0 below). |
| IOCFG0 | 0x01 / 0x02 | 0x01 is used to instruct the chip to generate an interrupt either when the RX FIFO is above threshold or a packet has been sent. Since our packet length is below the RX FIFO threshold (see FIFOTHR above and PKTLEN below), the interrupt is always generated when a packet has been received. This setting is written to the CC2500 before receiving a packet.  0x02 is used to instruct the chip to generate an interrupt when the TX FIFO goes below the TX FIFO threshold. Since the TX FIFO threshold is 1 (see FIFOTHR above), this effectively means that an interrupt occurs when the TX FIFO is emptied, and thus when a packet or packet burst is finished being sent. This setting is written to the CC2500 before transmitting a single packet or a packet burst. |
| PKTCTRL1 | 0x07 | Used to append the reception status at the end of every packet received and to perform an address check for every packet received. |
| PKTCTRL0 | 0x44 | Used to perform data whitening on the packets, to enable CRC verification for each packet, and to use fixed packet length. We chose to use data whitening to obtain a smooth power distribution over the occupied bandwidth. |
| PKTLEN | 0x2 | Sets the fixed packet length to 2: 1 address byte + 1 payload byte. |
| ADDR | 0x00 | Default address. Will change every time the user wishes to receive a packet. The address is specified every time one of the ReceivePacket functions are called. |
| CHANNR | 0x00 | Channel 0 is always used. |

### Packet Format

For our purposes, the packet format only needs to support sending a very small number of commands to different addresses. The command can be encoded as a single byte. Therefore, only 1 address byte and 1 command byte are required, for a total of 2 bytes per packet.

### Programming Interface

The programming interface is comprised of a WLESS\_StatusCodeTypeDef structure and a small number of functions. All packet related functions (WLESS\_SendPacket, WLESS\_ReceivePacket, WLESS\_SendPacketBurst, and WLESS\_ReceivePacketVerified) return an instance of the WLESS\_StatusCodeTypeDef structure to indicate if the operation was a success or a failure and why. Table 8 summarizes all the functions available to the user of this module. In this table, WLESS\_PACKET\_LENGTH has a value of 1 because only the payload byte is visible to the user. The wireless module takes charge of adding and removing the addresses from the packets.

Table 8 - Wireless module functions.

|  |  |  |
| --- | --- | --- |
| **Name** | **Parameters** | **Description** |
| WLESS\_Init | None | Initializes the wireless module according to the CC2500 configuration shown in Table 7. |
| WLESS\_SendPacket | uint8\_t\* packet\_bytes,  uint8\_t address | Sends a number of bytes equal to WLESS\_PACKET\_LENGTH pointed to by <packet\_bytes> to the address specified in <address>. The address is automatically added to the packet, so <packet\_bytes> only contains the actual payload. This is a blocking function that waits until packet transmission is over. |
| WLESS\_SendPacketBurst | uint8\_t\* packet\_bytes,  uint8\_t address,  uint8\_t burst\_size | Does the same operation as WLESS\_SendPacket, except that it is repeated for the number of times specified in <burst\_size>. This is typically used to send multiple copies of the same packet in order to increase the chance that at least one of them is received without error by the receiver. To perform this burst operation, the value of MCSM1 is changed to 0x32 to avoid having the CC2500 go back to IDLE between every packet. This is a blocking function that waits until the packet burst transmission is over. |
| WLESS\_ReceivePacket | uint8\_t address,  uint8\_t\* packet\_bytes | Receives a packet at the address specified in <address> and writes it at the location pointed to by <packet\_bytes>. <packet\_bytes> is expected to be of size WLESS\_PACKET\_LENGTH. The address is removed from the packet before writing into <packet\_bytes>. This is a blocking function that waits until a packet is received. |
| WLESS\_ReceivePacketVerified | uint8\_t address,  uint8\_t\* packet\_bytes | Does the same operation as WLESS\_ReceivePacket, except that it discards the packet and waits for another one if CRC verification has failed. This is a blocking function that waits until a packet that has passed CRC verification is received. |
| WLESS\_GetLatestRSSI | None | Returns a byte containing the most recent RSSI value reported by the CC2500 chip when a packet was last received. |
| WLESS\_GetLatestDecibelRSSI | None | Returns the same as WLESS\_GetLatestRSSI, except that the value is converted into Db format. |

# Tests and Observations

# Conclusion

# References

STMicroelectronics. (2012). UM1472 User Manual - STM32F4DISCOVERY - STM32F4 high-performance discovery board.

STMicroelectronics. (2014). STM32F427xx STM32F429xx Datasheet - production data.

Texas Instruments. (2014). CC2500 - Low-Cost Low-Power 2.4 GHz RF Transceiver. Dallas, Texas, United States.