**ECSE 426 - MICROPROCESSOR SYSTEMS**

**LAB 3 NOTES**

**Wireless Transceiver Interfacing**

**Group 6**

Hossam Al-Saati 260169285

Vinod Sridharan 260256868

1. **INTRODUCTION**

This lab involved designing a program to communicate with a wireless CC2500 transceiver using the SPI module, and using this transceiver to communicate with another CC2500 wirelessly. Communication with the user was achieved by means of a menu, displayed on screen using the UART module.

**2. FUNCTIONAL SPECIFICATIONS**

**Function:** void clk\_set(void)   
**Purpose:** Set up the Sub-Master Clock of the MCU to the high frequency oscillator  
**Inputs:** None

**Output:** None  
**Special Cases/Error Conditions:**

* None

**Function:** void spi\_init(void)  
**Purpose:** Started up the SPI with the required and predetermined conditions (such as clock speed and the pin configurations)  
**Inputs:** None

**Output:** None  
**Special Cases/Error Conditions:**

* The function required the Sub-Master Clock of the MCU to be configured to the high frequency oscillator beforehand

**Function:** char spiTransmit(char c)

**Purpose:** Sent a strobe command to the CC2500. Strobe commands consisted of single bytes with the addresses in memory of the CC2500 that represented the respective commands (for instance SRES was 0x30)

**Input:** c: a character which represented the command to be sent

**Output:** The chip status byte that was returned on the SOMI line (Slave Out Master In line)

**Special Cases/Error Conditions:**

* None

**Function:** int spi\_read (short header)

**Purpose:** Read data from a predetermined address in the CC2500, with the address specified in the header byte input

**Input:** header: a character which represented the address of the data to be read

**Output:** The data that was read from the address specified by the header

**Special Cases/Error Conditions:**

* Header had to have an MSB bit set to 1. Since no burst reading was done in our program, bit 6 had to be 0

**Function:** int spi\_write(short header, short data)

**Purpose:** Wrote data to a predetermined address in the CC2500, with the address specified in the header byte input, and the data specified by the data byte input

**Input:** header: a character which represented the address of the data to be written; data: the data byte to be written in this address

**Output:** The chip status byte

**Special Cases/Error Conditions:**

* Header had to have an MSB bit set to 0. Since no burst reading was done in our program, bit 6 had to be 0

**Function:** void uart\_init(void)  
**Purpose:** Started up the UART with the required and predetermined conditions  
**Inputs:** None

**Output:** None  
**Special Cases/Error Conditions:**

* The function required the Sub-Master Clock of the MCU to be configured to the high frequency oscillator beforehand

**Function:** void sendStr(char \*string)

**Purpose:** sendStr used TXBUF0 to send a string through the UART to the Hyper Terminal

**Inputs:** \*string – a pointer to the string that needed to be sent

**Output:** While the function did not explicitly return a value, it sent the string, character by character, to the terminal

**Special Cases/Error Conditions:** None

**Function:** int getKey(void)

**Purpose:** Got a valid address from the user

**Inputs:** The function did not have an input, but expected inputs from the UART. Note that this method used the keyPress() method to obtain each key from the UART

**Output:** The function returned an address between 0x00 and 0x2F

**Special Cases/Error Conditions:**

* If the user pressed the Escape key, the operation aborted and returned -1
* Only the last two characters before the ‘return’ key were returned
* The function only returned if 2 valid characters were entered before pressing the ‘return’ key.

**Function:** void keyPress(void)

**Purpose:** Got a valid character from the UART

**Inputs:** The function did not have an explicit input, but expected an input character from the UART

**Output:** A number between 0 and 17, representing hex numbers between 0 and F, the ‘enter’ and the escape key

**Special Cases/Error Conditions:**

* If the user entered a character other than those accepted, it was ignored by the keyPress method

**Function:** void menu(void)

**Purpose:** This function had an infinite while loop which contained the Finite State Machine that controlled the main menu that the user could see

**Inputs:** None

**Output:** None

**Special Cases/Error Conditions:** None

**3. IMPLEMENTATION**

The main menu was implemented using a Moore finite state machine. There were a total of six states, representing the various options available to the user. The different states and the actions inside each state are highlighted below. Furthermore, the state transitions are shown in the state transition diagram (Figure1). Note that the system’s UART and SPI were configured before entering the FSM.

|  |  |
| --- | --- |
| **FSM STATE** | **Purpose and tasks** |
| MAIN\_MEN | * Print the Main Menu * Move to the GETKEY state |
| GETKEY | * Wait for the user to enter a valid option specified by the menu * If the user pressed either 1, 2, 3, or 4, move to the appropriate state in the FSM |
| DUMP\_REG | * Read all the registers in the CC2500 and print them on the UART terminal in order as follows:   REG\_ADDR: REG\_NAME = REG\_VAL   * Returns to MAIN\_MEN |
| MOD\_REG | * Modifies the value of a register in the CC2500 with the value and address provided by the user * Gets an address and a value from the user, and writes to the CC2500 using the SPI module * If the user presses ‘Escape’, it cancels and returns to MAIN\_MEN. Otherwise, it completes the write and then returns to MAIN\_MEN |
| SEND\_DAT | * Sends a data packet wirelessly to another device (with a definite address) through the CC2500 * Gets an address from the user and writes this value (if valid) in the ADDR register of the CC2500 * Gets a string of predetermined length from the user and writes this string to the Transmit Queue (TX\_FIFO) * Enables Transmitting and transmits this string * Returns to MAIN\_MEN |
| REC\_DAT | * Receives a data packet wirelessly from another device through the CC2500 * Sets up the CC2500, and enables reception using the strobe command SRX * Listens for any packet. If a valid packet is received, quits reception and reads the value in the Receive Queue (RX\_FIFO) * Prints the string * Returns to MAIN\_MEN |

*Table 1: Finite State Machine: State actions*

The main menu displayed to the user had four options in total and had the following format:

MAIN MENU:

1. DUMP ALL REGISTERS
2. MODIFY A REGISTER
3. TRANSMIT DATA
4. RECEIVE DATA

Configuring the SPI required knowledge of the limits of the peripheral CC2500 that we had connected to. Considering the maximum frequency of transmission, we set up the UCLK at 2.67 MHz, well below the 6.5 MHz specified by the data sheet. We also had to configure the MCU to receive and transmit on the rising edge of the UCLK, and configure the STE (which went low right before a transmission and high right after).



*Figure 1: Finite State Machine – Flowchart*



*Figure 2: Program flow of MOD\_REG*



*Figure 3: Program flow of SEND\_DAT*

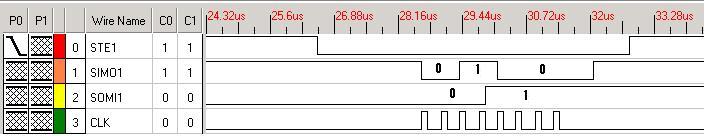


*Figure 4: Program flow of REC\_DAT*

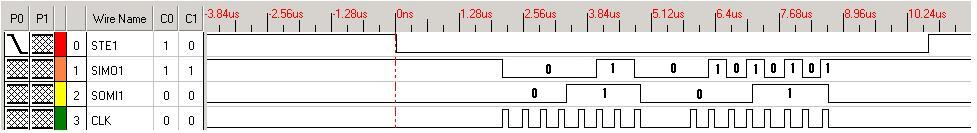
**4. PERFORMANCE ANALYSIS**

The analysis performed in this lab included testing the FSM/Menu and testing inputs from the USART. We had already tested the UART module extensively in previous lab[[1]](#footnote-1) and consequently did not test it again. Validating the menu functionality was done easily by verifying the output that we received. Since the construct of the FSM was simple, the testing of the menu was trivial.

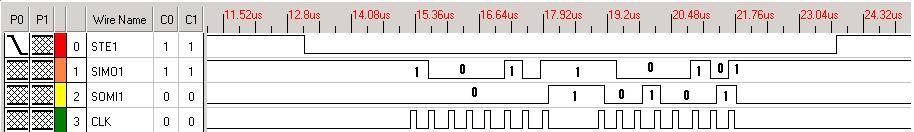
In addition, we had to test the SPI interface, and verify the integrity of the data to and from the CC2500 (SIMO and MOSI). The following extracts from the logic analyzer confirm that the SPI was indeed configured correctly and functioned normally. An infinite while loop was configured, in which we issued a Software Reset (SRES = 0x30), then wrote a value of 0x55 to the register 0x06, and then read from an address 0x05 (the command 0x85)



*Figure 4: Issue Software Reset*



*Figure 5: Write 0x55 to 0x06*



*Figure 6: Read from 0x05 (0x91 seen)*

However, testing the address input was non-trivial as it included several possible input combinations. Several tests were performed to check if the inputs entered return the desired outputs. Below is a table listing a few test cases that were used for the purpose specified above. Note that the ‘Return’ character is represented by ‘R’ and the ‘Escape’ character is represented by ‘Q’

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Output** | **Test Case Purpose** | **Pass or Fail** |
| B3R | ABORTED | The system should accept valid addresses in the CC2500 | Pass |
| 2314B3R | ABORTED | Only the last 2 characters are accepted, and if they produce an invalid address, operation should be aborted | Pass |
| 246122ER | 2E | Entering a valid address should be accepted by the system | Pass |
| 325323Q | ABORTED | Pressing escape while getting inputs should abort the operation | Pass |

*Table 2: Inputs test cases for the menu*

**4. REFERENCES**

[1] 304-426A Microprocessor Systems Fall 2009 Experiment 3: Wireless Transceiver Interfacing, McGill University, Fall 2009

[2] Texas Instruments, *MSP430x1xx Family User’s Guide*, Mixed Signal Products, 2006

[3] Texas Instruments, *Implementing an Ultralow-Power Keypad Interface With the MSP430*, Mike Mitchell, 2002

[4] Texas Instruments, *CC2500**Low-Cost Low-Power 2.4 GHz RF Transceiver,* 2009

[5] Texas Instruments, *Design Note DN503,* Siri Namtvedt, 2007

1. See Lab 2 notes [↑](#footnote-ref-1)