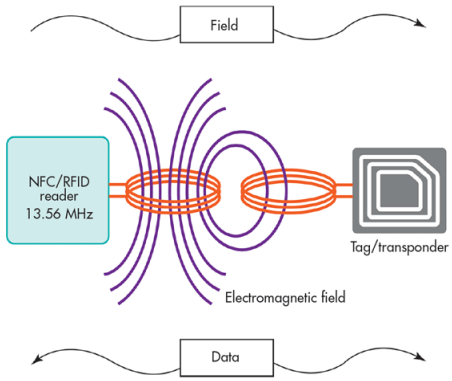
Assignment 3 - (5pts)

EGN 3211 Engineering Analysis and Computation

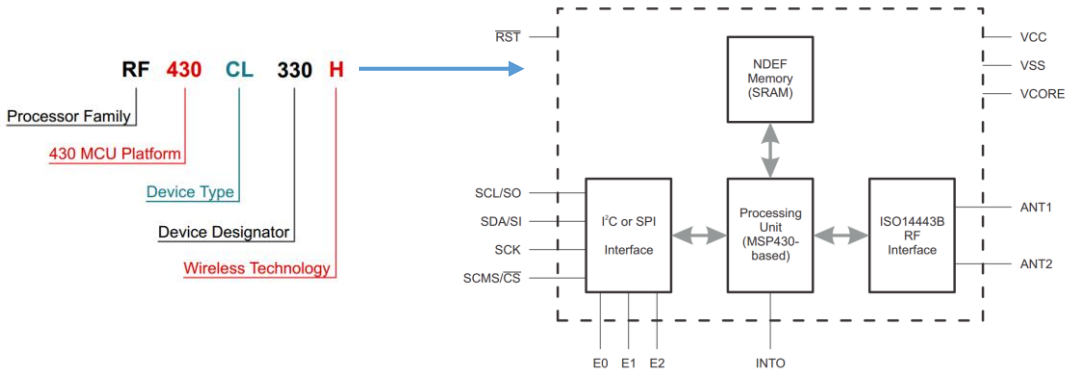
Due by 11:59PM on July 13th

Recent advances in near field communication (NFC) has allowed for the development and widespread use of new technology, such as RFID (Radio Frequency Identification). RFID is commonly used by gated residential communities as a key to enter the development, the gym, and other secured common areas. Additional RFID applications include hotel room keys, toll road transponders, shipment tracking, pet identification, etc. The global RFID market has had substantial growth over the past few years going from a $7 billion value industry in 2012 to almost a $9 billion one in 2014. A typical RFID system consists of an active NFC transponder that retrieves data from a passive RFID tag.



A RFID reader uses electromagnetic induction to power a RFID tag.

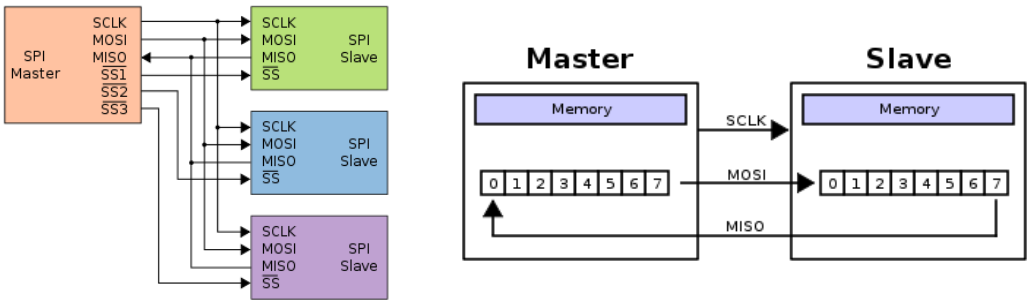
NFC capable cell phones can transfer contact information, photos, and other data with another device by simply placing them next to each other. Bluetooth enabled devices can use NFC to pair with one another instead of having to enter a passphrase. Texas Instruments sells a dynamic NFC Type 4 Tag called the *RF430CL330H* which makes use of the 430 MCU platform.



Block diagram of the RF430CL330H

A dynamic tag allows for another device to update the identification buffer NDEF that gets transmitted over the NFC channel. A MSP430G2553 could be used to set the value of NDEF using the SPI communication protocol. The SPI protocol is synchronized using SCLK (Serial Clock) and provides a full-

duplex channel allowing for simultaneous communication in both directions by using the MOSI (Master Out Slave In) and MISO (Master In Slave Out) bus. The SS (Slave Select) line is used by the SPI Master to select which peripheral it is actively communicating with over the MOSI & MISO bus. The SPI Master will set the SS line low to rotate the 8 bits in the SPI Master's buffer into the SPI Slave's buffer. This can be implemented in hardware using two 8-bit shift registers to form a circular buffer. One form of SPI captures data when the clock is 1 and updates the output buffer when the clock is 0.



Example SPI configuration between four devices (left) using the circular buffer (right).

1) Circular Buffer Simulation (filename: circularBuffer.c)

Write a C program to simulate the circular buffer used by the SPI protocol. The global unsigned char variables *SCLK, masterShiftRegister, slaveShiftRegister, MISO* and *MOSI* must be used to represent the serial clock, master device's 8-bit shift register, slave device's 8-bit shift register, the value of the last bit in the slave's register, and the value of the last bit in the master's register respectively. Your program must prompt the user to enter the initial values of the master and slave shift registers and the number of clock transitions to simulate. The main function calls *simulateCommunication* starts with SCLK at 1 (HIGH) and simulates the SPI communication by executing the *onClockChangeEvent* followed by flipping the SCLK for *numClockTransitions* times. The *onClockChangeEvent* should update the MISO and MOSI variables when the SCLK is 1 and should update the masterShiftRegister and slaveShiftRegisterare variables when SLCK is 0. Finally, the program should print a summary of the shift registers, MISO, and MOSI after the simulation ends as unsigned 8 bit binary values.

Mandatory global variables and simulateCommunication function:

```
unsigned char SCLK, masterShiftRegister, slaveShiftRegister, MISO, MOSI;

void simulateCommunication(unsigned int numClockTransitions)
{
        SCLK = 1;
        while(numClockTransitions --)
        {
                onClockChangeEvent();
                SCLK ^= 0x01;
        }
}
```

Example simulation using 65, 37, and 3 for the SPI master buffer, SPI slave buffer, and clock transitions:

0 Cycles: masterBuffer → 01000001	slaveBuffer → 00100101	MOSI → 1	MISO → 1
1 Cycles: masterBuffer → 01000001	slaveBuffer → 00100101	**MOSI → 1**	**MISO → 1**
2 Cycles: **masterBuffer → 10100000**	**slaveBuffer → 10010010**	MOSI → 1	MISO → 1
3 Cycles: masterBuffer → 10100000	slaveBuffer → 10010010	**MOSI → 0**	**MISO → 0**

Sample output screenshot:

```
Please enter the 8-bit value for the SPI master buffer:
65
Please enter the 8-bit value for the SPI slave buffer:
37
Please enter the number of clock transitions:
3

After the simulation:
 The value of the master buffer: 10100000
 The value of the slave buffer: 10010010
 The value of MOSI: 0
 The value of MISO: 0
end
```

2) RFID Passphrase Encryption (filename: encryption.c)

Most RFID tags store and transmit encrypted data to prevent easy interception of passphrases. This means that our *MSP430G2553* must encrypt all data prior to updating the *RF430CL330H*.

Write a C program that implements a simple encryption algorithm built using an 8-bit shared key. The key will be known by the sender and receiver prior to transmission and must have six 1-bits and two 0-bits. Suppose that every 8-bit frame contains **6** data bits and **2** parity bits. The encryption algorithm will build frames according to the 8-bit shared key where the **6** 1-bits correspond to the **6** data bits from left to right and the **2** 0-bits are determined by the data bits.

- The first 0-bit location in the shared key is 1 if there are an odd number of 1-bits in the first 3 of 6 data bits and 0 if there are an even number of 1-bits. (from left to right)
- The second 0-bit location in the shared key is 1 if there are an even number of 1-bits in the remaining 3 data bits and 0 if there are an odd number of 1-bits. (from left to right)

Your program must prompt the user to enter a string with at most 10 characters and the hexadecimal value of the 8-bit shared key. The characters are then parsed into the 8-bit frames to be transmitted and stored by the RFID tag. If there are fewer than 6 data bits in the last frame then the appropriate number of 0's are appended onto to the right of the last digit (see example and sample output). The program should print the binary values of every 8-bit frame before exiting.

Example encryption of the string "go" using the shared key EE_{16} (11101110_2):

'g' → 01100111 'o' → 01101111 (ASCII values of the characters 'g' and 'o')

"go" → 0110011101101111 → 0110011101101111 00 (remaining bits in the last frame are set to 0)

$Frame_1$ → 01100010 $Frame_2$ → 11001101 $Frame_3$ → 11111000 (after adding the parity bits)

Sample output screenshot:

```
Please enter a string of up to 10 characters:go
Please enter the 8-bit shared encryption key as a hexadecimal value:EE

The following frames were generated after encryption:
Frame 1: 01100010
Frame 2: 11001101
Frame 3: 11111000
```

RFID Reference: http://www.ti.com/product/RF430CL330H

Guidelines for receiving full credit:

Write readable code with descriptive comments, appropriate line spacing and white space, and meaningful identifiers for variables and functions.

Make your output look similar to the sample output and allow your program to exit. Do not prevent termination because of statements like *system("pause")*, *getchar()*, or anything else that will require further interaction from the user.

Submit C programs with the name stated in the problem description and that can be compiled following the C99 standard. Do not use features introduced in the C11 standard.

Do your own work, a grade of 0 will be given if cheating is suspected.