# Embedded and Real-time Systems (EENG34030)
# Coursework Assignment (Part 1) – Max. 50 Marks

*Note: You may work in pairs for this assignment.*

**Task Description:**

You need to create an embedded system capable of verifying 16-digit identification numbers found on payment cards, social security IDs, survey codes, and similar documents. This validation process relies on the Luhn Algorithm, which was developed by IBM scientist Hans Peter Luhn and involves a straightforward summation of check digits. The design specifications are illustrated in Figure 1. Please familiarize yourself with the procedure outlined from steps 1 to 6.

To gain a better grasp of UART communication between the MSP430FR2433 board and a PC, it is advisable to go through and simulate the code included in the appendix of this document. This exercise will aid in your comprehension of the UART communication protocol and verify its proper operation.

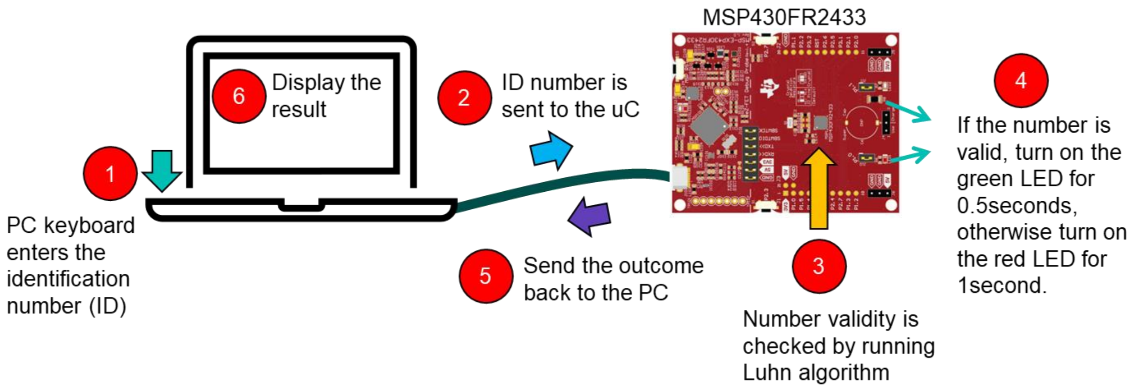

MSP430FR2433
1 PC keyboard enters the identification number (ID)
2 ID number is sent to the uC
3 Number validity is checked by running Luhn algorithm
4 If the number is valid, turn on the green LED for 0.5seconds, otherwise turn on the red LED for 1second.
5 Send the outcome back to the PC
6 Display the result

Figure 1 Embedded System for ID verification

You are required to submit a report of no more than 10 pages on this design project. The report should include the following sections, and your evaluation will be based on the specified marking criteria:

| Section | What to include | Marks% |
|---|---|---|
| Description | Explore the Luhn Algorithm. Provide a brief introduction of the design. | 10 |
| Design Approach | Explain the approach. Include flowcharts, algorithms, and diagrams as necessary to illustrate your design methodology. | 20 |
| Implementation | Present your C-code that implements the designed system. Show how different code fragments correspond to the design approach you described in the previous section. | 40 |
| Design Correctness | Demonstrate the functionality and correctness of your design. Show that it produces the desired outputs as expected. Describe the testing procedures you followed, including the number of test cases and their results. | 10 |
| Clarity/Conciseness | Ensure that the description and results are presented clearly and concisely. Avoid unnecessary clutter or redundancy in your report. | 10 |
| Quality of Write-up | Evaluate the overall quality of your report's presentation. Ensure that it follows a clear and organized format. Make sure the report is easy to read and understand. Use professional figures, plots, and diagrams with appropriate labels and explanations. | 10 |

Remember to adhere to these guidelines while preparing your report to maximize your marks. It's essential to strike a balance between providing comprehensive information and maintaining clarity and conciseness in your report.

**Appendix:**

The code [1] given at the end of this document starts with the fundamentals of configuring a UART at the register level and demonstrates how to send and receive a byte from the MSP430 via UART. The example code given at the end of this document shows proper initialization of registers and interrupts to receive and transmit data.

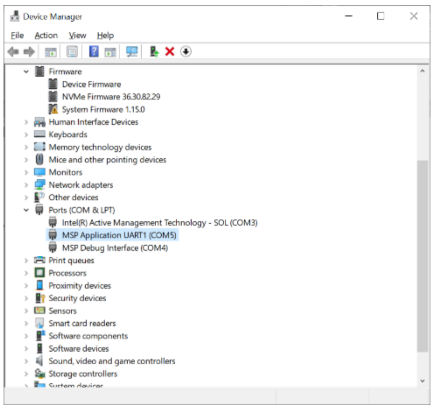
Let's take a look at the UART initialization code. This example uses 115200 baud. Baud is the UART's bit rate and is sourced from the MSP system clock (SMCLK) running at 1MHz. This 1MHz clock must be divided down to create the 115200 Baud clock. Table 22-4 in [2] provides the fractional constant values to be setup for different baud-rates. After setting up the baud rate, the UART module is enabled and its receive interrupt is enabled.

```
    // Configure UART for Power UP default clock rate
    UCA0CTLW0 |= UCSWRST;                      // Put eUSCI in reset
    UCA0CTLW0 |= UCSSEL__SMCLK;                // set equal to SMCLK

    // Baud Rate calculation 115200 baud
    UCA0BR0 = 8;                             // 1000000/115200 = 8.68
    UCA0MCTLW = 0xD600;                      // Remainder - 0.68

    // 9600 baud
    UCA0BR0 = 104;                             // 1000000/9600 = 104.167  104
```

The interrupt service routine (ISR) handler (shown below) uses a switch statement that selects the appropriate code section of the ISR to execute based on the interrupt vector value. Once a character is received, a '*' character is sent back to the PC through the serial terminal and the red LED is toggles at the same time.

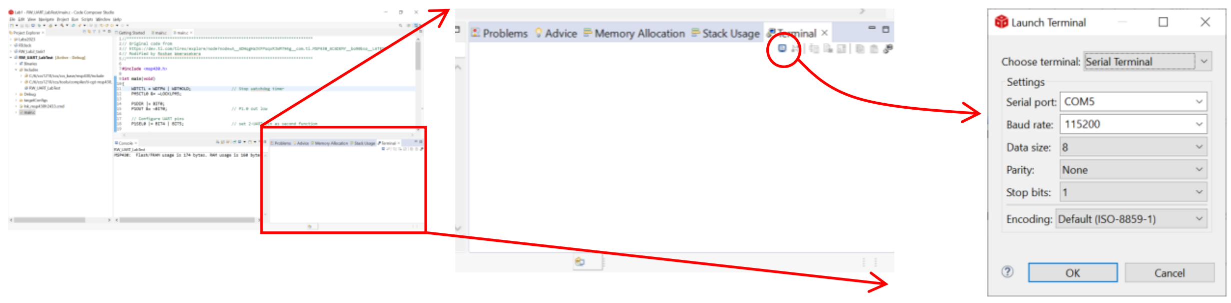
```
switch(UCA0IV)
        {
            case USCI_NONE: break;
            case USCI_UART_UCRXIFG:
                while(!(UCA0IFG & UCTXIFG)); // Wait for TX buffer to be ready for
new data
                P1OUT ^= BIT0; // turn off the RED LED
                UCA0TXBUF = '*';
                break;
            case USCI_UART_UCTXIFG: break;
            case USCI_UART_UCSTTIFG: break;
            case USCI_UART_UCTXCPTIFG: break;
        }
```

First of all find the COM port that the board is connected using the Device Manager. In this example in the right, the serial port COM5 is used for UART communication.



```
Firmware
    Device Firmware
    NVMe Firmware 36.30.82.29
    System Firmware 1.15.0
Human Interface Devices
Keyboards
Memory technology devices
Mice and other pointing devices
Monitors
Network adapters
Other devices
Ports (COM & LPT)
    Intel(R) Active Management Technology - SOL (COM3)
    MSP Application UART1 (COM5)
    MSP Debug Interface (COM4)
Print queues
Processors
Proximity devices
Security devices
Sensors
Smart card readers
Software components
Software devices
Sound, video and game controllers
Storage controllers
```

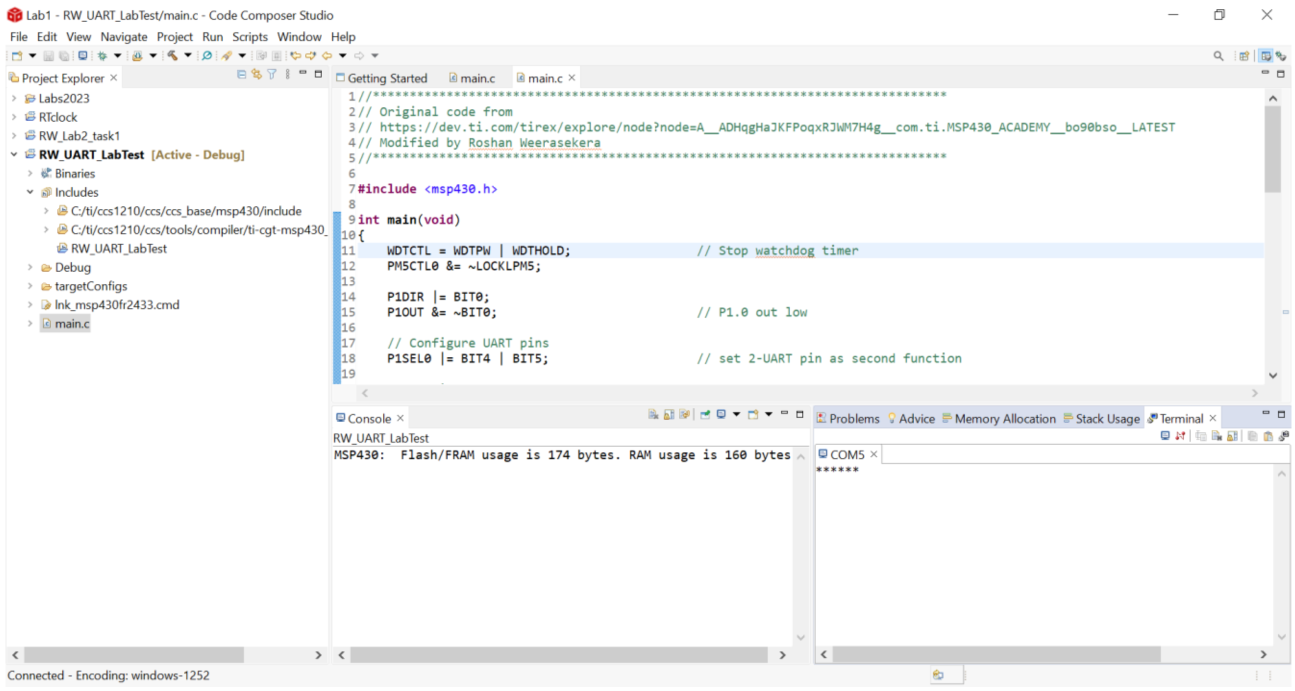
Now open the terminal in CCS going to the View menu. A Terminal setup window will appear below right corner of CCS as shown in Figure 1.



Choose terminal: Serial Terminal
Settings
Serial port: COM5
Baud rate: 115200
Data size: 8
Parity: None
Stop bits: 1
Encoding: Default (ISO-8859-1)

*Figure 2 Terminal and serial port setting*

You will have to set-up the correct baud-rate 115200 in this example and COM port for correct functionality. Figure 3 demonstrates the function of the code.


Lab1 - RW_UART_LabTest/main.c - Code Composer Studio
File Edit View Navigate Project Run Scripts Window Help
Project Explorer
Labs2023
RTclock
RW_Lab2_task1
RW_UART_LabTest [Active - Debug]
Binaries
Includes
C:/ti/ccs1210/ccs/ccs_base/msp430/include
C:/ti/ccs1210/ccs/tools/compiler/ti-cgt-msp430_
RW_UART_LabTest
Debug
targetConfigs
lnk_msp430fr2433.cmd
main.c
Getting Started
main.c
main.c
Console
Problems
Advice
Memory Allocation
Stack Usage
Terminal
COM5
Connected - Encoding: windows-1252

```
//******************************************************************************
// Original code from
// https://dev.ti.com/tirex/explore/node?node=A__ADHqgHaJKFPoqxRJWM7H4g__com.ti.MSP430_ACADEMY__bo90bso__LATEST
// Modified by Roshan Weerasekera
//******************************************************************************

#include <msp430.h>

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;               // Stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;

    P1DIR |= BIT0;
    P1OUT &= ~BIT0;                         // P1.0 out low

    // Configure UART pins
    P1SEL0 |= BIT4 | BIT5;                  // set 2-UART pin as second function
```

```
RW_UART_LabTest
MSP430:  Flash/FRAM usage is 174 bytes. RAM usage is 160 bytes
```

```
******
```

*Figure 3 Terminal in CCS demonstrating the operation.*

```
//******************************************************************************
// Original code from
//
https://dev.ti.com/tirex/explore/node?node=A__ADHqgHaJKFPoqxRJWM7H4g__com.ti.MSP430_ACA
DEMY__bo90bso__LATEST
// Modified by Roshan Weerasekera
//******************************************************************************

#include <msp430.h>

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;                       // Stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;

    P1DIR |= BIT0;
    P1OUT &= ~BIT0;                                 // P1.0 out low
    // Configure UART pins
    P1SEL0 |= BIT4 | BIT5;                    // set 2-UART pin as second function

    // Configure UART for Power UP default clock rate
    UCA0CTLW0 |= UCSWRST;                     // Put eUSCI in reset
    UCA0CTLW0 |= UCSSEL__SMCLK;               // set equal to SMCLK

    // Baud Rate calculation 115200 baud
    UCA0BR0 = 8;                           // 1000000/115200 = 8.68
    UCA0MCTLW = 0xD600;                    // Remainder - 0.68  Table 21.4 & 21.5

    UCA0CTLW0 &= ~UCSWRST;                    // Initialize eUSCI
    UCA0IE |= UCRXIE;                         // Enable USCI_A0 RX interrupt

    __bis_SR_register(LPM0_bits|GIE);     // Enter LPM0 CPU off, SMCLK running
    __no_operation();                     // For debugger

    return 0;

}


#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
{
    switch(UCA0IV)
        {
            case USCI_NONE: break;
            case USCI_UART_UCRXIFG:
                while(!(UCA0IFG & UCTXIFG)); // Wait for TX buffer to be ready for new
data
                P1OUT ^= BIT0; // turn off the RED LED
                UCA0TXBUF = '*';
                break;
            case USCI_UART_UCTXIFG: break;
            case USCI_UART_UCSTTIFG: break;
            case USCI_UART_UCTXCPTIFG: break;
        }

}
```

References:

[1] https://dev.ti.com/tirex/explore/node?node=A__ADHqgHaJKFPoqxRJWM7H4g__com.ti.MSP430_ACADEMY__bo90bso__LATEST

[2] https://www.ti.com/lit/ug/slau445i/slau445i.pdf