C For Embedded Systems report draft

## Introduction and the Luhn algorithm:

The Luhn algorithm is a “check digit” algorithm developed by Hans Peter Luhn working at IBM in 1960. It is designed to check for accidental errors in a number of different identification numbers but does not work well for detecting malicious attacks as it can be easily bypassed. It is one of several algorithms emerging at the time that were designed to be easily to implement and efficient to run in software, giving them an advantage over earlier mathematical formulae that were not designed with software in mind.

In this project we are designing an interface to validate numbers typed onto a computer keyboard using the Luhn algorithm. This is done by connecting a PC to a TI MSP430 development board, on which the code is run. The device accepts an input from the keyboard of the computer using the UART communications channel on the board, validates the number and flashes a green or red light and sends a signal back to the PC telling it the status depending on the outcome.

## Description of the Luhn algorithm:

The Luhn algorithm follows a simple set of rules, which allows it to be easily implemented in software even if it would be extremely difficult to carry out in classical mathematics.

Using the example number 12345678:

1. Double every second digit, working backwards:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

7 is the second last digit, so starting at 7 every other digit is doubled ^

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 2 | 6 | 4 | 10 | 6 | 14 | 8 |

1. In the event where a digit of the number is greater than 9 (where one digit becomes two), the digits of this number are summed to replace it:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 2 | 6 | 4 | 10 | 6 | 14 | 8 |

10 and 14 are greater than 9 so their digits are added (1 + 0 is 1, and 1 + 4 is 5)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 2 | 6 | 4 | 1 | 6 | 5 | 8 |

1. The resultant digits are summed:
2. Using the modulo function, it is checked if the sum is divisible by 10:

A valid result is 0, any other number gives an invalid result

In this case the number is not valid, an example of a valid number would be 12345658:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 5 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 2 | 6 | 4 | 10 | 6 | 10 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 2 | 6 | 4 | 1 | 6 | 1 | 8 |

As the result of the algorithm in this case is 0, the number is considered valid.

## Implementation:

In order to implement the algorithm in c while maintaining it’s legibility and ease of understanding we created a function to run the algorithm, making it as concise as possible:

The function starts by accepting an input, which is a character string, and converts it to an integer string in the code:

/\*Lines 98-105 \*/

The integer string is fed through the Luhn algorithm, digit by digit, in a for loop and the appropriate maths is carried out on each digit in the code:

/\* Lines 17 to 35 \*/

Then each digit is added to the sum and the function returns a 1 or 0 dependent on the value of the modulus:

/\* Lines 37 to 42 \*/

If the input is valid the green LED flashes on for 0.5 seconds, then turns off. If invalid the red LED flashes for one second, then turns off.

## Testing

We tested the program by providing a number of valid and invalid inputs and checking the results by hand. The testing was not automated.

Note: The algorithm will not detect the difference between 22 and 55, 33 and 66 or 44 and 77.

Valid input:

4304025946658368

4272759033136742

Invalid input:

4304027746658368

4276959033136742

Invalid input that appears valid:

4272759066136742