Lab 4 – Programming USART to parse GPS message - Simulation

In this LAB, you will first get deeper in the implementation of USART interfaces on STM32F10x. In particular, you will be asked to make changes to the USART configuration and measure the impact on the code execution.
Next, you will write a program that parses a **GPGLL**- *GPs Geographic position, Latitude / Longitude and time* - message received via a USART interface and calculate **your virtual distance** from the given location.

Useful sources (QMplus)

- STM32_USART_Tutorial project
- Week 4 Tutorial exercise
- Week 4 Lectures on USART

GPGLL – GPs Geographic position, Latitude / Longitude and time

The GPGLL message is standardised and should include at least the latitude and longitude. Optionally, it may also include the time and the checksum and/or the status.

General Format:

You can find details about the GPGLL message format here. This is what you need for this lab:

$GPGLL,LLLll.mm,N,LLLll.mm,W,hhmmss,A*CS

1. **LLLll.mm**> Latitude LLL deg. ll.mm min
2. **N** => North
3. **LLLll.mm**=> Longitude LLL deg. ll.mm min
4. **W** =>West
5. *Optional*: **hhmmss** =>Fix taken at hh:mm:ss UTC
6. *Optional*: A: status=> A (active) indicates the data is valid; V (void) = invalid
7. *Optional*: ***CS**=> Checksum

Examples:

- eg1. $GPGLL,3751.65,S,14507.36,E*77
- eg2. $GPGLL,4916.45,N,12311.12,W,225444,A*31
- eg3. $GPGLL,5133.81,N,00042.25,W*75

Checksum:

The checksum is the representation of two hexadecimal characters of an XOR of all characters in the sentence between – but not including – the **$** and the ***** character.

Example:

$GPGLL,5300.97914,N,00259.98174,E,125926,A*28

The checksum **CS** is initialised to **0**. Once the **$** sign is received, the checksum calculation starts. For every character, **char,** received between (and not including) **$** and *****, and including the commas, the checksum is updated as **CS=CS^char**. Once a ***** is detected, the checksum is not updated and the process is terminated. For the string shown in this example, the results is **CS**=**0d40**. **CS** is then expressed as two hexadecimal characters

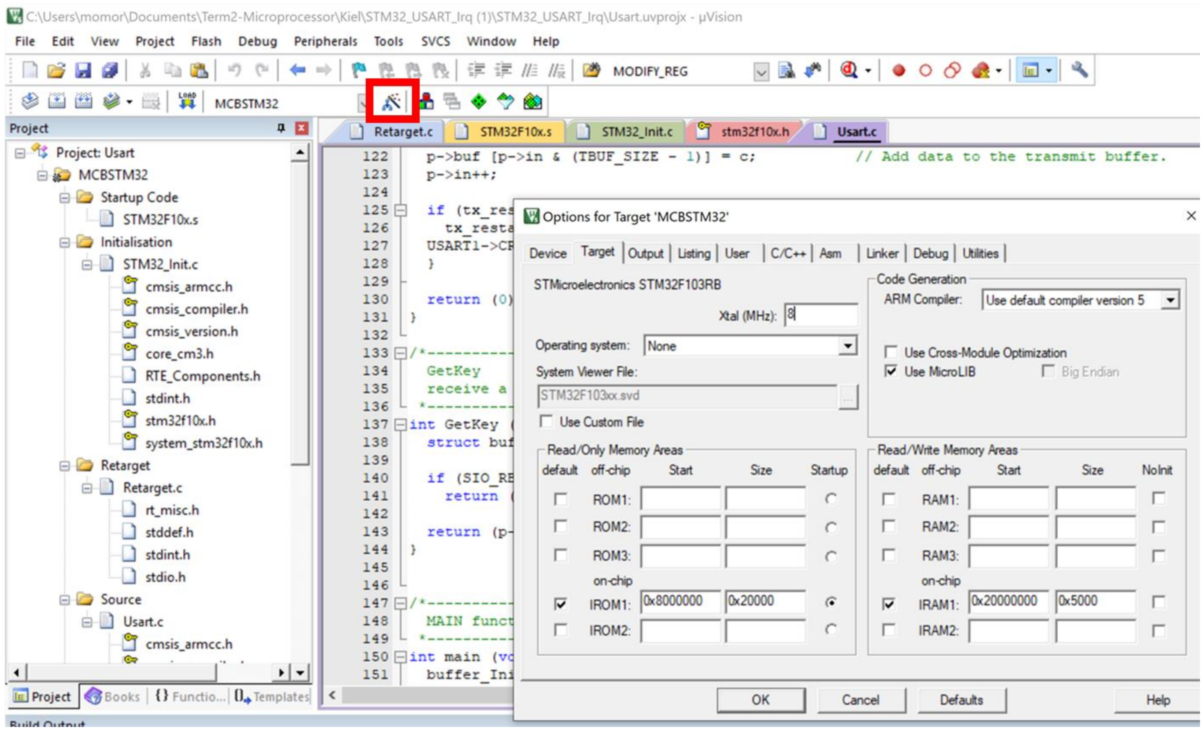
(**0d40=0x28**) and compared to the received checksum. Only if the calculated **CS** matches the received checksum, the message is valid.

Part 1- Getting deeper with USART configuration

Baud Rate

Using the **STM_USART_Tutorial** project, Follow the steps below and note your answers in the lab-sheet:

ATTENTION: Make sure you have the correct crystal frequency setting, **Xtal=8MHz**, before proceeding to the next step. You check this in the 'Options for Target' window as shown below:



```
  p->buf [p->in & (TBUF_SIZE - 1)] = c;           // Add data to the transmit buffer.
  p->in++;

  if (tx_res
    tx_resta
  USART1->CR
  }

  return (0)
}

/*----------
  GetKey
  receive a
 *----------
int GetKey
  struct buf

  if (SIO_RE
    return

  return (p-
}

/*----------
  MAIN funct
 *----------
int main (vo
  buffer_Ini
```

Options for Target 'MCBSTM32'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

STMicroelectronics STM32F103RB

Xtal (MHz): 8

Operating system: None

System Viewer File: STM32F103xx.svd

☐ Use Custom File

Code Generation
ARM Compiler: Use default compiler version 5
☐ Use Cross-Module Optimization
☑ Use MicroLIB ☐ Big Endian

Read/Only Memory Areas

default	off-chip	Start	Size	Startup
☐	ROM1:			○
☐	ROM2:			○
☐	ROM3:			○
	on-chip			
☑	IROM1:	0x8000000	0x20000	●
☐	IROM2:			○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
☐	RAM1:			☐
☐	RAM2:			☐
☐	RAM3:			☐
	on-chip			
☑	IRAM1:	0x20000000	0x5000	☐
☐	IRAM2:			☐

OK | Cancel | Defaults | Help

Step A1: Build the project and start a debugging session. Open USART1 peripheral and UART #1 serial window.

Step A2: Note down the baud rate (**BR**), DIV_M, and DIV_F as shown in USART1 peripheral window.

Step A3: What is f_{PCLK}? Justify your answer.

Step A4: Add a breakpoint at line 80 of usart.c (USART1->SR &= ~USART_SR_TXE; // clear interrupt). Note the time when the letter '**l**' is seen in the UART #1 window and the time when the letter '**n**' is seen. How many $\mathbf{T_{bit} = 1/BR}$ does it correspond to?

Step A5: Change the number of stop bits to two using the USART1 peripheral window and repeat Step 4.

Step A6: Change back to 1 stop bit and enable odd parity using the USART1 peripheral window; repeat Step 4.

Step A7: Disable parity check. What should the value in USART1_BRR be to have a baud rate of 9600 baud? Explain.

Step A8: Enter the value you found in Step 7 and check the resulting baud rate. Note the value you see. Repeat Step 4.
Step A9: Fix the Baud rate to 115200 baud and move to the next phase.

Part 2- Parsing GPS messages

Task1- Build a parser

1. Build a parser function to recognise the following NMEA output sentences by GPS units:
   - $GPGGA: Global Positioning System Fix Data
   - $GPGLL: Geographic position, latitude / longitude
   - $GPRMC: Recommended minimum specific GPS/TRANSIT data
2. The parser is only expected to decode $GPGLL messages. If a $GPGGA or $GPRMC message is received, the parse should generate the following message: “This is a $GP**xyz** message that I am not able to decode.”, where xyz could be GGA or RMC.
3. If a message is received with a different header from those listed above, the parser should be stopped and an error message generate: “This message type: $**abcde**, is not recognised.” Where **abcde** would reflect the characters received after the **$** sign.
4. Any string that does not start with **$** is considered an error and message should be generated: “Error in message, $ expected.”
5. If a $GPGLL header is received, the parse should read strings separated by commas and fill in the variables until a ***** is received. For this message, the following steps should be taken: **$GPGLL,LLLll.mm,N,LLLll.mm,W,hhmmss,S*CS**
   a. Latitude=**LLLll.mm**> LatDeg=LLL; LatMin=ll.mm. NOTE: **ONLY** two digits after the decimal point are expected.
   b. Hem= **N**; if **N** is neither N nor S and error message is generated “Message is corrupt, unrecognised latitude, **N**”.
   c. Longitude=**LLLll.mm**=> LongDeg= LLL ; LongMin=ll.mm min. NOTE: **ONLY** two digits after the decimal point are expected.
   d. EorW=W; if W is neither E nor W and error message is generated “Message is corrupt, unrecognised longitude, W”.
   e. *Optional*: FixedTime=**hhmmss** =>Hour= hh; Min=mm; Sec=ss. If Hour, Min or Sec >24 or negative, generate error message “Message is corrupt, unrecognised time, Hour:Min:Sec”.
   f. *Optional*: status= **S**; if **S**=A=> Valid=true; **S**=V=>Valid=false; else, error message “Message is corrupt, unrecognised status, **S**”.
   g. *Optional*: RxCheck=**CS**
6. If Step 5 is completed without errors and CS is provided, calculate the check sum, CalCheck, by calling the Checksum Function (See Task 2). If CalCheck=RxCheck, the message is deemed correct, otherwise, an error message is generated “Message is corrupt, checksum does not match.”
7. If the variables Latitude, Hem, Longitude, and EorW, are not found in the message, generate an error “Message corrupt, missing latitude and/or longitude information.”.

8. If Step 5 and Step 6 are completed without errors, the following message should be generated:
   This is a $GPGLL message.
   The Latitude is LatDeg degrees and LatMin min Hem
   The Longitude is LongDeg degrees and LongMin min EorW
   The time is Hour:Min:Sec
   Data validity is Valid

Task2- Checksum function

Write a function that takes as an input a string that starts with $ and ends with *, and gives the check sum as an output in decimal. The pseudocode is given below:

```
int checksum (str char){
            define cs and i as integers;
            define c as character;
            initialise cs and i;
            c=first character in string;
            if c is not ${
                  error message}
            else {
                     while (c is not '*'){
                       cs=cs XOR c;
                        increment i;
                        update c;
                     }
                  return(cs);
                  }
}
```

To test your function, calculate the checksum of the following message and make sure it is 57 decimal or 0x39: **$GPGLL,3957.77,N,11621.49,W,103210,A*.**

Testing Parser Functionality

Once you have written the parser as detailed above, you can test the functionality by running the following steps and filling your answers in the lab sheet. Note: you are not asked to stream the message; you can type it in your code, e.g.,   strcpy (str, “GPGLL,5152.41,N,4.04,W,155509*94").

<u>**Step B1**</u>: For the message: “GPGLL,5131.45,N,2.42,W,155509*94”, run the parser and write the output message.

<u>**Step B2**</u>: For the message: “$GPRMC,05131.45,N,2.42,W,155509*94”, run the parser and write the output message.

<u>**Step B3**</u>: For the message: “$GPGLL, 5131.45,N*94”, run the parser and write the output message.

Step B4: For the message: "$GPGLL,3957.77,N,11621.49,W,103210*54", run the parser and write the output message.
Step B5: For the message: "$GPGLL, 3957.77,N,11621.49,W,103210,X*1E", run the parser and write the output message.
Step B6: For the message: $GPGLL, 5131.27,N,2.42,W,155509,V*21", run the parser and write the output message. Can you guess this location?
Step B7: For the message: : $GPGLL,3957.77,N,11621.49,W,103210,A*39", run the parser and write the output message. Can you guess this location?

Part 3- Encoding GPS messages

Task3- Encoding GPGLL messages

Step C1: Enter your virtual location (use the last 6 digits of your BUPT ID for Latitude and the last 6 digits of QMUL ID for Longitude) using UART #1, as follows (no need to embed error detection in the encoder). Example, BUPT ID=12345678 and QMUL ID=87654321, then Latitude=34 deg 56.78 min and Longitude=65 deg 43.21 min. If the last digit of your BUPT ID is even NorS=N for latitude else NorS=S. If the last digit of your QMUL ID is even EorW =W for longitude else EorW =E.

```
Please enter the latitude in deg , min, N/S:
<LLL, ll.mm, NorS >
Please enter the longitude in deg, min, W/E:
<LLL, ll.mm, EorW>
The encoded message is:
$GPGLL,LLLll.mm,NorS,LLLll.mm,EorW
```

Step C2: Write a function FindDistance to calculate the distance DIST between your virtual location and that decoded in Step B7. Output the following message:

```
I am DIST kilometres far from the target location.
```

For this exercise, you will use the Haversine method for calculating the distance between two points define by (Lat_1, $Long_1$) and (Lat_2, $Long_2$), where Lat and Lon are expressed in degrees. This method is used to calculate the great-circle distance between two points – that is, the shortest distance over the earth's surface – giving an 'as-the-crow-flies' distance between the points (ignoring any hills they fly over, of course!). More details can be found here: https://www.movable-type.co.uk/scripts/latlong.html. This can also be used to verify the functionality of the FindDistance function in your code.

Haversine formula:

$$a = \sin^2(\Delta\phi/2) + \cos\phi_1 \cdot \cos\phi_2 \cdot \sin^2(\Delta\lambda/2)$$
$$c = 2 \cdot \text{atan2}(\sqrt{a}, \sqrt{1-a})$$
$$DIST = R \cdot c$$

where:

$\phi_1 = Lat_1 \times \pi/180$ is the latitude in radians,
$\Delta\phi = \phi_1 - \phi_2$ (radians),
$\lambda_1 = Long_1 \times \pi/180$ is the longitude in radians,
$\Delta\lambda = \lambda_1 - \lambda_2$ (radians),

R is earth's radius (mean radius = 6,371 10^3m);
DIST is the distance in meters.

- End of lab 4-