Kirthika MURUGENDRAN - Dignon TRAN

TP1 et 2 - Gestion de chaîne de caractères - Norme NMEA

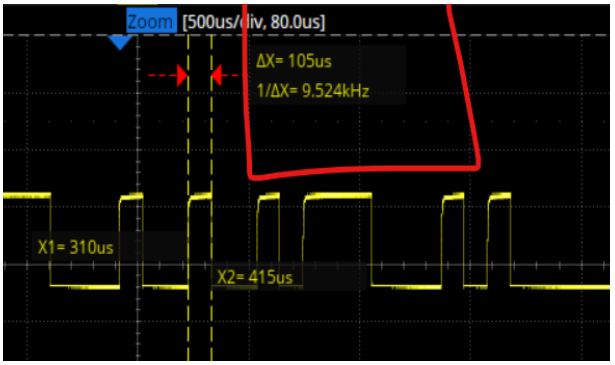
On possède la **carte 20.**

6.1. Objectifs

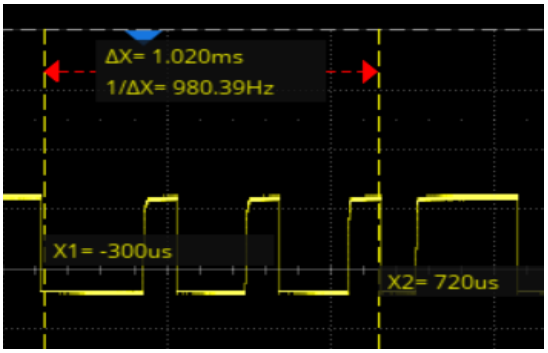
Ce TP permet de nous apprendre à manipuler les chaînes de caractères NMEA grâce à une carte possédant les liaisons série USART1 et USART2.

6.3. Les liaisons séries

L'objectif de cette partie est de comprendre le déroulement de la récupération des messages par la carte.



Un bit dure 105 µs et le $débit = \frac{1}{durée\ d'un\ bit} = \frac{1}{105 \times 10^{-6}} = 9.524\ kbps$ donc on en déduit que le débit binaire réel vaut 9524 bits par seconde, ce qui est proche des 9600 bits par seconde indiqués dans le sujet du TP.
En effet, l'écart relatif entre la valeur mesurée et celle théorique vaut
$\Delta = \frac{9600 - 9524}{9600} = 0.8\%$
L'état de repos est en 1, le premier bit doit donc être un 0, pour indiquer que le message a commencé.
On envoie 10 bits pour émettre un octet en tenant compte des bits de start et de stop.



On compte depuis l'oscilloscope: 0001001001
En enlevant le bit de start et de stop et sachant que l'envoi est "LSB first", on trouve que le premier octet envoyé est : **00100100**. On peut remarquer dans ce cas seulement qu'en "LSB first" ou "MSB first" le résultat serait le même (par symétrie).

Cet octet correspond au caractère “**$**” d’après la table ASCII.

6.6.2. Boucle de recopie entrée / sortie

On avait perdu 30 min sur cette partie car la console n’affichait pas ce qu’on voulait. Le problème a été résolu quand le professeur a enlevé la sonde de la carte, car la sonde créait un court-circuit.
Finalement, on a bien obtenu ce qu’on voulait (voir image ci-dessous).

```
$GPGGA,084440.477,4150.109,N,08737.751,W,1,12,1.0,0.0,M,0.0,M,,*74
$GPGSA,A,3,01,02,03,04,05,06,07,08,09,10,11,12,1.0,1.0,1.0*30
$GPRMC,084440.477,A,4150.109,N,08737.751,W,,,170723,000.0,W*60
```

La console affiche les 3 trames

6.6.3. Fonction fillBuffer

```
int fillBuffer(char * buffer, int size){
    int i;
    int total = 0;
    char received_char = 0;
    for (i=0; i < size; i++){
        buffer[i]=0;
    }

    HAL_UART_Abort(&huart1);
    HAL_StatusTypeDef status = HAL_UART_Receive(&huart1, (uint8_t *)&received_char, 1, HAL_MAX_DELAY);
    if (status != HAL_OK){
        return -1;
    }

    do{
        while(HAL_UART_Receive(&huart1, (uint8_t *)&received_char, 1, HAL_MAX_DELAY)!= HAL_OK){}
    }while (received_char != '$');

    do{
        *buffer = received_char;
        while(HAL_UART_Receive(&huart1, (uint8_t *)&received_char, 1, HAL_MAX_DELAY)!= HAL_OK){}
        total++;
        buffer++;
    }while (received_char != '\n');

    *buffer = received_char;

    return total;
}
```

Le programme ci-dessus permet d’afficher dans la console ce que la carte a reçu :

```
Received 67 char : $GPGGA,084440.477,4150.109,N,08737.751,W,1,12,1.0,0.0,M,0.0,M,,*74

Received 62 char : $GPGSA,A,3,01,02,03,04,05,06,07,08,09,10,11,12,1.0,1.0,1.0*30

Received 63 char : $GPRMC,084440.477,A,4150.109,N,08737.751,W,,,170723,000.0,W*60
```

La variable “buffer” n’est jamais remise à sa valeur initiale car c’est un pointeur, et quand on modifie “buffer”, on modifie la valeur de l’adresse du pointeur donc le pointeur lui-même.

6.7. Vérification de l'intégrité des trames

#Notre version

```
int isGPGGA(char * frame){
    char compared[6];
    for (int i = 0; i<6; i++){
        compared[i]= frame[i];
    }
    if (strcmp(compared, "$GPGGA") == 0){
        return 1;
    }
    else{
        return -1;
    }
}
```

#Version améliorée par le professeur

```
int isGPGGA(char * frame){
    char * compared = "$GPGGA";
    for (int i = 0; i<6; i++){
        if(compared[i] != frame[i]){
            return -1;
        }
    }
    return 1;
}


int extractChecksum(char * buffer){
    while (*buffer != '*'){
        buffer++;
    }
    return (buffer[1]&0x0f)*16 + (buffer[2]&0x0f);
}


int calculateChecksum(char * buffer){
    int checksum = 0;
    buffer++;
    while (*buffer != '*'){
        checksum = checksum ^ *buffer;
        buffer++;
    }
    return checksum;
}
```

D’après le pseudo-code suivant (issu de l’énoncé du TP):

```
1   FONCTION extractChecksum :
2       TANT QUE VARIABLE POINTE PAR buffer DIFFERENT de '*' FAIRE
3           buffer <- buffer+1
4       FIN TANT QUE.
5
6       RENVOIE (buffer[1]&0x0f)*16 +(buffer[2]&0x0f);
7   FIN extractChecksum
8
9   FONCTION calculateChecksum :
10      variable checksum
11      checksum <- 0
12      buffer <- buffer+1
13      TANT QUE VARIABLE POINTE PAR buffer DIFFERENT de '*' FAIRE
14          checksum <- checksum ^ (VARIABLE POINTE PAR buffer)
15          buffer <- buffer+1
16      FIN TANT QUE.
17
18      RENVOI checksum;
19  FIN calculateChecksum
```

La ligne 6 permet de mettre à 0 les bits de poids fort et de conserver les bits de poids faible. Puis on multiplie la valeur obtenue de buffer[1]&0x0f par 16, ce qui revient à le décaler à gauche de 4 bits, on combine ensuite avec buffer[2]&0x0f pour former un nouvel octet. Le programme renvoie à la fin la valeur décimale du nouvel octet formé.

La console renvoie cette réponse:

```
Just received 63 bytes
Just received 62 bytes
Just received 67 bytes
Trame GPGGA
Checksum -> calculated 74        extract 74
```

On observe que la valeur calculée et la valeur extraite sont les mêmes, ce qui est cohérent. On a passé une heure à essayer de vérifier que les programmes s’exécutent bien, à afficher les trames sur la console, où le fait de reboot suffisait largement pour faire afficher ce qu’il faut à la console. On a également aidé un groupe à confirmer que leur carte ne fonctionnait pas en testant notre programme sur leur carte.

```
int checkFrame(char * buffer){
    int calculate = calculateChecksum(buffer);
    int extract = extractChecksum(buffer);

    if ( (calculate == extract) && (isGPGGA(buffer) == 1) ){
        return 1;
    }
    else{
        return 0;
    }
}

void loop3(){
    int size = fillBuffer(buffer, BUFFER_MAX_SIZE);
        printf("Just received %d bytes \r\n",size);
    if (checkFrame(buffer) == 1){
        printf("Trame GPGGA détectée \r\n");
    }
    else{
        printf("N'est pas une trame GPGGA \r\n");
    }

}
```

La console renvoie ce qui est demandé. Il fallait définir buffer en tant que variable globale pour que le programme s'exécute correctement, ce qui nous a bloqués pendant 1h car on ne trouvait pas d'erreur dans les fonctions. On a même testé à côté chacune des fonctions extractChecksum et calculateChecksum pour voir si les programmes étaient corrects.

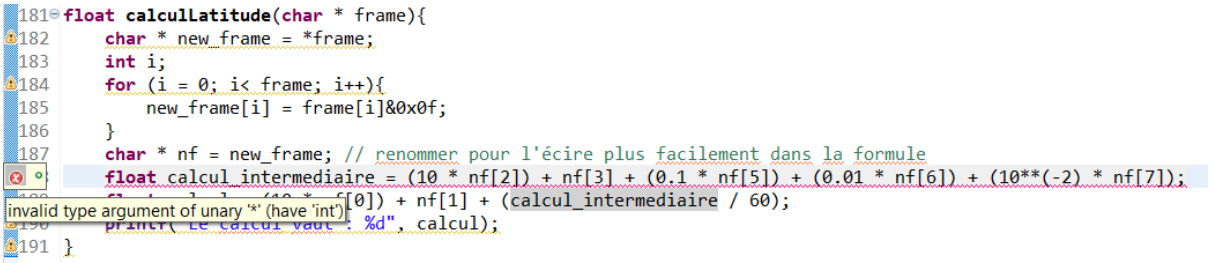
```
Just received 63 bytes
N'est pas une trame GPGGA
Just received 62 bytes
N'est pas une trame GPGGA
Just received 67 bytes
Trame GPGGA détectée
```

6.8. Récupération de l'information au format flottant

Cette fonction correspond à une fonction intermédiaire ; elle est utile ensuite pour trouver la latitude.

```
char trouverVirgule(char * frame){
    while( *frame != ','){
        frame++;
    }
    frame++; // renvoie la nouvelle frame sans la virgule au début
    printf("Nouvel frame : %s \r\n",frame);
}
```

On a voulu faire une fonction intermédiaire qui calcule la latitude en valeur absolue (qui correspond à la formule indiquée dans le TP) pour éviter une fonction trop longue et incompréhensible.
On s'est arrêté ici à la fin du TP avec des erreurs.



Les programmes ci-dessous sont des propositions de réponse pour la suite du TP.

Le problème majeur venait de 10^{-2} car on devait utiliser la fonction pow depuis la librairie math.h qu'on devait importer pour mettre en notation scientifique. D'ailleurs en lisant l'énoncé, il semblerait que le dernier terme devrait être 10^{-3} et pas 10^{-2}. Nous n'avons pas pu tester si la fonction calculLatitude est correcte ou non en l'absence de la carte du TP mais au moins il n'y a pas d'erreur du côté du debugger.

```
float calculLatitude(char * frame){
    char * new_frame = *frame;
    int i;
    for (i = 0; i< *frame; i++){
        new_frame[i] = frame[i] & 0x0f;
    }
    char * nf = new_frame; // renommer pour l'écire plus facilement dans la formule
    float calcul_intermediaire = (10 * nf[2]) + nf[3] + (0.1 * nf[5]) + (0.01 * nf[6]) + (pow(10, -2) * nf[7]);
    float calcul = (10 * nf[0]) + nf[1] + (calcul_intermediaire / 60);
    printf("Le calcul vaut : %d", calcul);
}
```

On calcule la distance entre deux virgules par cette fonction.

```
int positionVirgule(char * frame){
    int position = 0;
    while( *frame != ','){
        position++;
        frame++;
    }
    return position;
}
```

La fonction ci-dessous permet d'avoir le signe de la latitude.

```
int signeLatitude(char lettre){
    int signe = (lettre == 'N') ? 1 : -1;
    printf("La latitude est du signe de %d" , signe);
    return 0;
}
```

Il ne manquerait plus qu'à combiner ces fonctions dans la fonction getLatitude(char * frame) (en espérant que les fonctions intermédiaires soient correctes avant) pour obtenir la fonction souhaitée.

Conclusion : Le TP était relativement long à traiter, on était restés assez longtemps bloqués à certaines questions, ce qui est très frustrant car il ne manquait pas grand-chose à chaque fois pour aboutir (problèmes de variables globales/locales, etc.). Les pseudo-codes ont été assez clairs pour nous, et nous avions pu avancer sur ces fonctions-là. La carte n'exécutait pas correctement nos programmes, même si le débug n'indiquait pas d'erreurs. Il fallait alors rallumer par moment la carte pour qu'elle réexécute le nouveau programme. On a appris des choses dans ce TP même si certains points ne sont toujours pas clairs au sujet de la trame