Épreuve d’admissibilité de conception préliminaire d'un système, d'un procédé ou d'une organisation

A. Présentation de l’épreuve

Arrêté du 28 décembre 2009 modifié

- Durée totale de l'épreuve : 6 heures
- Coefficient 1

L'épreuve est spécifique à l'option choisie.

À partir d'un dossier technique comportant les éléments nécessaires à l'étude, l'épreuve a pour objectif de vérifier les compétences d'un candidat à synthétiser ses connaissances pour proposer ou justifier des solutions de conception et d'industrialisation d'un système technique dans le domaine de la spécialité du concours dans l'option choisie.

B. Sujet

Le sujet est disponible en téléchargement sur le site du ministère à l’adresse :
https://media.devenirenseignant.gouv.fr/file/agregation_externe/00/0/s2021_agreg_externe_sii_informatique_3_1390000.pdf

C. Éléments de correction

Partie 1. Conception du système d'information

Q1. 8 +32 + 32 + 64 +8 = 144 bits de données.
Soit un total de 144 bits à écrire. Avec une correction d'erreur de niveau Q, la version 2 (de 25x25) permet de saisir toutes les données.

Q2.

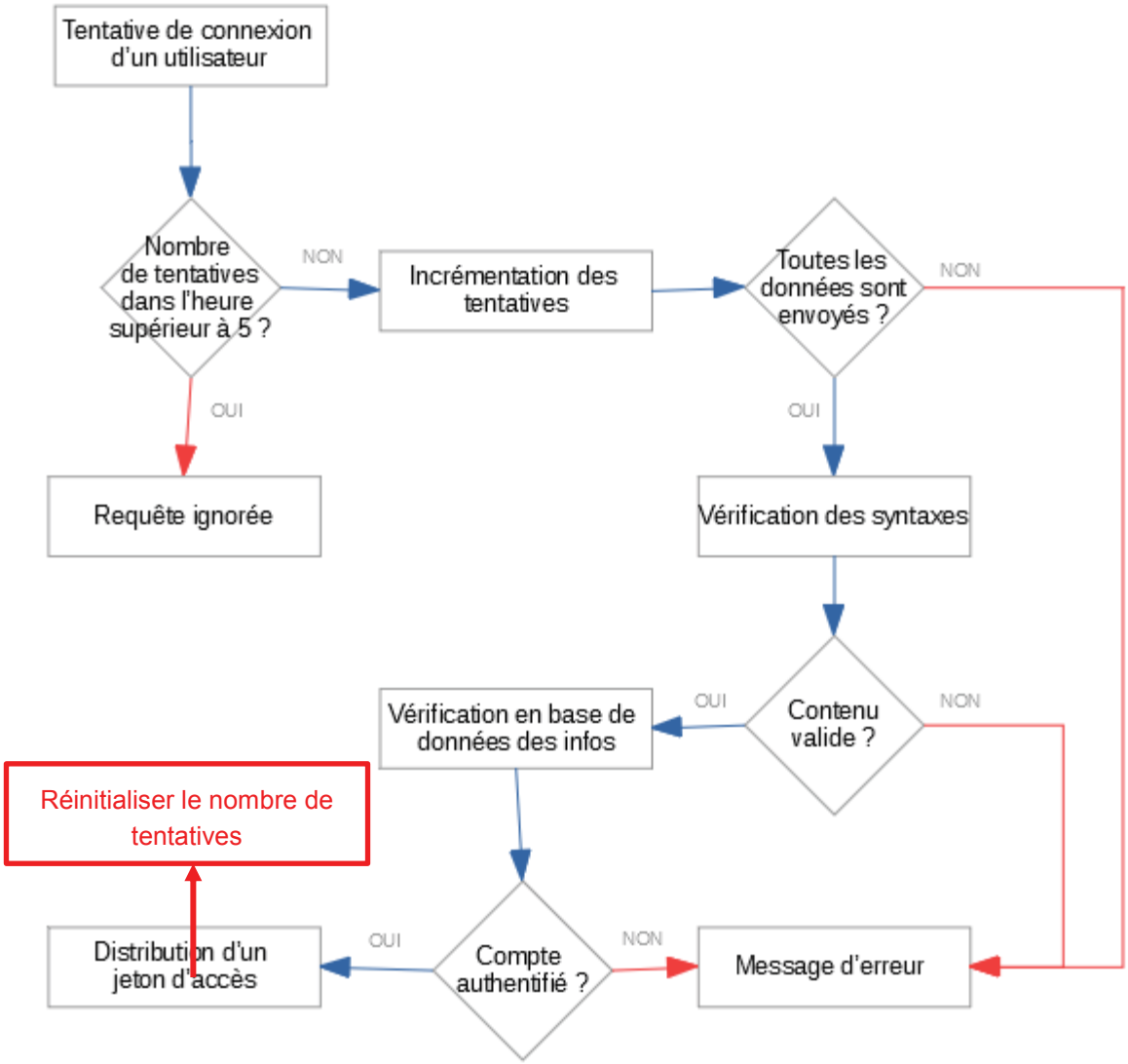
```
QRGEncoder qrgEncoder = new QRGEncoder(data, null, QRGContents.Type.TEXT,
imageSize);
        return qrgEncoder.getBitmap();
```

Q3. ^[A-Za-z0-9._%+-]{1,}@(?:[A-Za-z0-9-]{1,}\\.){1,}[A-Za-z]{2,}$

Q4.

```
public class MailUtilsTest {
    @Test
    public void isValid() {
        assertEquals(true,
MailUtils.isValid("dupont.robert@urbanloop.fr"));
        assertEquals(true, MailUtils.isValid("urbanloop@univ-
lorraine.fr"));
        assertEquals(false, MailUtils.isValid("ne-pas-
répondre@ubanloop.fr"));
         assertEquals(true,
MailUtils.isValid("Alain.Genieur@prod.urbanloop.com"));
         assertEquals(true,
MailUtils.isValid("capsule45217854@54000.urbanloop"));
         assertEquals(false,
MailUtils.isValid("jdoe@urbanloop..fr"));
    }
}
```

Q5.




```
SELECT (SELECT station.name FROM station WHERE station.id = run.ending_station_id) as nom
FROM run
INNER JOIN customer ON run.customer_id = customer.id
WHERE customer.mail = 'dupont.robert@fournisseur.fr'
GROUP BY run.ending_station_id
ORDER BY count(run.ending_station_id) DESC
LIMIT 3
```

Q7. La réponse du serveur est donnée au format JSON. Il s'agit d'un format de données répandu, simple et léger, facilement interprétable.

Q8. Le code d'erreur 401 est le code attendu. Le code 403 est plus approprié dans le cas où une fonctionnalité est non accessible à l'utilisateur.

Q9. En ajoutant un ou plusieurs serveurs applicatifs, il est possible de répartir la charge sur les différentes machines. Le serveur HTTPS frontal doit alors répartir les requêtes vers l'un ou l'autre des serveurs. Cette évolution est répétée autant de fois que nécessaire à l'absorption de la charge (scaling horizontal).

Partie 2. Transmission de la feuille de route

Q10. D'après la documentation du DT6 la topologie Peer-to-peer est la plus adaptée car elle permet d'utiliser des routeurs intermédiaires pour étendre la portée du réseau et fiabiliser les échanges en cas de défaillance d'une route. Un réseau de type Mesh comportant des routeurs intermédiaires, dans les stations par exemple, peut être créé pour joindre toutes les capsules. Le coordinateur est la centrale de gestion et les capsules des « end devices ». La proposition de la topologie cluster tree convient également.

Q11. Les deux modes de fonctionnement des modules Xbee sont le mode transparent et le mode API. En mode transparent, les modules se contentent de retransmettre les octets reçus par la liaison UART à un module distant appairé. Il n'y a pas de gestion réseau disponible dans ce mode ni de communication sécurisée. La configuration en mode API est indispensable pour pourvoir former un réseau de communication sécurisé et fiable entre les différents modules Xbee.

Q12.

Paramètre	Noeud 1	Noeud 2
Adresse 64 bits (hexadecimal)	00 13 A2 00 41 B1 AB 01	00 13 A2 00 41 B6 3D 21
Adresse réseau 16 bits (hexadecimal)	93 47	C1 A6
Identifiant du noeud (Node identifier) (caractères)	C005	C014

Q13. Un code CRC est ajouté à la trame après application d'un algorithme sur les bits de la trame. Le destinataire peut appliquer le même algorithme pour vérifier le code CRC.
Un checksum est une simple somme de contrôle calculée en faisant la somme d'un certain nombre de bits de la trame avant sa transmission.
Le principal avantage du checksum est sa compacité, en général un octet, et la facilité de programmation et de vérification.
L'avantage du code CRC est de permettre de réparer certaines erreurs de transmission, ce que ne permet pas le checksum.

Q14. D'après le DT6 pour vérifier la validité d'une trame, on additionne tous les octets d'une trame y compris le checksum mais sans le start delimiter et la taille (les 3 premiers octets). La trame est valide si l'octet de poids faible obtenu est à FF.
Pour la trame du nœud 1 on obtient en hexadécimal :
95+00+13+A2+00+41+B1+AB+01+93+47+02+93+47+00+13+A2+00+41+B1+AB+01+43+30+30+35+00+00+00+02+02+C1+05+10+1E+3E = 8FF
L'octet de poids faible est bien à FF, la trame du nœud 1 est valide.

Q15. Adressage 16bits : l'adresse 64 bits est mise à 0xFFFFFFFFFFFFFFFF (DT7).
Transmission de l'heure~~s~~ préfixe 0x48 et traduction hexadécimale de 1 599 208 323 = 0x5F51FB83

7E **00 12 10** 01 **FF FF FF FF FF FF FF FF 93 47** 00 00 **48 5F 51 FB 83 A6**

Q16. Un trajet dure environ 5 minutes soit 300 secondes. Il faut 2 octets pour coder le temps.

Q17. L'échantillonnage de 1 Hz donne une position par seconde soit 4 octets par seconde dans le fichier, ce qui donne une taille de 4*300 = 1200 octets
Chaque trame commence par l'identifiant 0x46 de transmission de la feuille de route, il reste 254 octets pour les données. Pour ne pas couper un couple temps-position il faut envoyer un multiple de 4 octets soit 254/4=63,5 -> 63*4 octets = 252 octets.
On envoie au maximum 252 octets par trame. Il faut donc 1200/252 = 4,7 soit 5 trames pour transmettre le fichier.

Q18.

```
static void receive_cb(const RemoteXBeeZB& remote, bool broadcast, const
uint8_t *const data, uint16_t len)
{
    uint32_t heure = 0;
```

```
    static uint8_t indice=0;

    switch(data[0])
    {
        case 0x46 :
          heure = data[1]<<24 |data[2]<<16 | data[3]<<8 | data[4];
          set_time(heure);
          break;
        case 0x48 :
          int k=1;
          while(k<len){
             uint16_t buffer=0;
             buffer = data[k]<<8 | data[k+1];
             feuille_route[indice][0] = buffer;
             buffer =data[k+2]<<8 | data[k+3];
             feuille_route[indice][1] = buffer;
             k=k+4;
             indice++;
           }
           break;
        default :
         printf("invalid data ID\r\n");
         break;
         }

}
```

Q19. D'après le DT6, l'adressage du coordinateur consiste à placer son adresse 64 bits entièrement à 0 soit : #define ADRR64_COORDINATOR ((uint64_t)0x0000000000000000)
D'après le code de la fonction, le booléen syncr à false force le Frame ID à 0 pour désactiver l'envoi d'une trame de réponse (cf. DT6 description de la trame Transmit Request), si le booléen syncr est à True, une trame de réponse est émise avec le même frame ID que la trame reçue.

Q20. L'initialisation du booléen syncr est faite dans le fichier Xbee.ZB.h qui définit la classe XbeeZB.
Si le booléen syncr est placé à TRUE, le module émetteur d'une trame de données reçoit une réponse de type Extended Transmit Status lui permettant de s'assurer de la bonne réception des données. Si le booléen syncr est placé à FALSE il n'y a pas de trame de réponse.
La réception d'une réponse lors de la transmission de l'heure et de la feuille de route par le coordinateur permet de s'assurer de la bonne réception des trames et une retransmission en cas de problème. Il est donc important de maintenir le paramètre syncr à TRUE.

Q21. D'après les questions précédentes, une trame de Transmit Request compte 18 octets d'entête plus les octets de données. Il faut 5 trames pour transmettre l'ensemble d'une feuille route. Soit 18*5+1200 = 1290 octets.
Temps de transmission RF : 1290*8/250000 = 41 ms.
Temps de transmission UART : 1290*8/115200 = 89,5 ms.

Q22. La latence est définie comme le délai entre le moment où une information est envoyée et le moment où elle est reçue.
Pour 1 octet, 8/250000 = 32 µs, pour 32 octets, 1 ms

Q23. On attend ici des candidats qu'ils évoquent l'allongement de la durée des transmissions en cas de réponse à chaque envoi de données. Les durées de transmission étant de l'ordre de la milliseconde la communication temps réel est compromise.

Q24. Le réseau Zigbee est facile d'installation et de mise en œuvre car les modules sont configurés pour communiquer de manière automatique lorsqu'ils sont sur le même réseau. L'utilisateur n'a pas besoin de gérer la partie RF.
Cependant le débit de ce genre de réseau reste limité et la latence relativement élevée, le protocole Zigbee est développé dans l'optique de communications basse consommation et non pour des communications en temps réel avec plus d'une centaine de modules.

Un protocole de communication plus complexe et haut débit est nécessaire dans le cadre de l'exploitation d'une centaine de capsules par réseau. La 4G ou la 5G semble plus adaptée.

Partie 3. Génération des feuilles de routes

Q25. S 7 3 5 22.2

Q26. Pour dessiner un trajet de 5 km il faut 1000 nœuds et 1000 segments.
Un nœud est composé 16 caractères environ avec des identifiants à 3 chiffres soit 18 octets.
Un segment est composé 16 caractères environ en considérant que les identifiants seront sur 3 chiffres soit 16 octets environ.
La taille du fichier peut être estimée à 18000 octets + 16000 octets = 34000 octets. La taille d'un fichier texte est de l'ordre de 30 koctets

Q27.

```
for element in document:
    if element[0] == 'N':
        numero = int(element[1]) # Compléter ici
        coordx = int(element[2]) # Compléter ici
        coordy = int(element[3]) # Compléter ici
        type_point = int(element[4]) # Compléter ici
        point = Point((coordx, coordy),type_point=type_point,
numero=numero)
        self.ajoute_point(point)

        s = Segment(noeud_origine, noeud_destination,
limitation_vitesse=limitation_vitesse) # Compléter ici
```

Q28.

```
def suivant(etat,delta_t,acceleration,vitesseMax):
    distanceParcourue=etat[0]
    vitesse=etat[1]
    etat=[ distanceParcourue + vitesse *
delta_t,min(vitesse+acceleration*delta_t,vitesseMax)]
    return  etat
```

Q29. L'accélération est en théorie infinie pour un moment extrêmement court.

Q30. L'accélération maximale dépend du pas de temps.

Q31. Dans un mouvement circulaire uniforme, l'accélération normale vaut $\frac{v^2}{R}$ donc $V_{max} = \sqrt{\gamma_{max}\, R}$

Q32. Avec trois points non alignés : $P1(x_1, y_1)$, $P2(x_2, y_2)$, $P3(x_3, y_3)$ le rayon du cercle devient :

$$R_c = \sqrt{(x_1 - x_c)^2 + (y_1 - y_c)^2}$$

avec

$$x_c = \frac{\frac{x_3^2 - x_2^2 + y_3^2 - y_2^2}{2(y_3 - y_2)} - \frac{x_2^2 - x_1^2 + y_2^2 - y_1^2}{2(y_2 - y_1)}}{\frac{x_2 - x_1}{y_2 - y_1} - \frac{x_3 - x_2}{y_3 - y_2}} \quad \text{et} \quad y_c = -\frac{x_2 - x_1}{y_2 - y_1} x_c + \frac{x_2^2 - x_1^2 + y_2^2 - y_1^2}{2(y_2 - y_1)}$$

Q33. $d_{cible} = \frac{V_{cible} - V}{\gamma}$

Q34. $d = \int v(t)dt$ et avec $v(t) = V + \gamma t$ il vient $d = \frac{(V_{cible}^2 - V^2)}{2\gamma}$

AN : $t_{cible} = 7{,}4$ s et d $= 111{,}1$ m

Q35.

```
def DistanceProchainFreinage(vitesseEntree,listeSegments,gamma):
```

```
    """retourne la distance du prochain freinage par rapport au point
    d'entrée dans le segment 0"""
distance_cumulee=0
    resultat=float('inf')
    for i in range(1, len(listeSegments)):
        vitesse_cible=listeSegments[i][1]
         d=(vitesse**2-vitesse_cible**2)/(2*gamma)
        distance_cumulee =distance_cumulee+listeSegments[i-1][0]
        if distance_cumulee+d<resultat:
            resultat=distance_cumulee+d
            segment=i
        return resultat
```

Q36.
Si Distance à parcourir avant prochain freinage <= 0 :
alors décélérer ;
Sinon si vitesse courante==vitesse_max :
alors maintenir la vitesse ;
sinon
accélérer ;

Q37. La fonction permet de calculer le plus rapide chemin entre 2 stations à l'aide de l'algorithme de Dijkstra. Le critère d'optimisation est la somme des temps de trajet entre chaque station.
Le dictionnaire temps_minimal_detecté tient à jour, pour chaque point, la durée minimale trouvée. Le bloc try except permet de gérer le cas où le point n'a pas encore été visité et n'a pas de temps de trajet de référence.

Q38. Toutes les capsules ont les mêmes consignes de vitesse aux mêmes endroits, c'est-à-dire que même si la vitesse d'une capsule varie au cours du temps, toutes les capsules subissent les mêmes variations de vitesse aux mêmes endroits. Le flux de capsule se compresse et se dilate mais les capsules n'entrent pas en collision. Pour insérer une capsule dans le réseau, une solution est de calculer la trajectoire de la capsule en amont et de vérifier qu'à chaque nœud d'intersection, le nœud est disponible au temps où la capsule passera dedans. Tant que tous les nœuds ne sont pas disponibles, le départ de la capsule peut être retardé d'un petit délai.

Partie 4. Système de positionnement

Q39. Périmètre de la roue P = π*0,4 = 1,25 m

Vitesse min 25 km/h soit 7 m/s, vitesse de rotation 7/1,25 = 5,5 tr/s
Soit un signal de sortie du capteur de 5,5 Hz.
Vitesse moyenne de la capsule = 60 km/h soit 16,67 m/s, vitesse de rotation 16,67/1,25 = 13,3 tr/s
Soit un signal de sortie du capteur de 13,3 Hz
Vitesse max = 80 km/h, soit 22,2 m/s, vitesse de rotation 22,2/1,25 = 17,76 tr/s
Soit un signal de sortie du capteur de 17,8 Hz

Q40. Un Timer est un compteur synchronisé sur une horloge. La connaissance de la fréquence de l'horloge permet de transposer la valeur du registre de comptage en temps écoulé depuis le début du comptage. En mode capture, la valeur du compteur du Timer est recopiée dans un registre à chaque changement d'état de la broche du microcontrôleur sur laquelle est configurée la capture, soit à chaque passage de l'aimant devant le capteur à effet Hall.
La soustraction d'une valeur du registre avec la précédente permet de connaître le temps écoulé entre 2 captures, connaissant le diamètre de la roue, cela permet de calculer la vitesse.

Q41. La fréquence d'acquisition est au maximum de l'ordre de 18 Hz, pour éviter des débordements trop fréquents du Timer l'horloge externe ACLK de fréquence 32768 Hz est préconisée.

Q42.
```
// Intialisation du Timer en mode continu
Timer_A_initContinuousModeParam Modeparam = {0};

Modeparam.clockSource = TIMER_A_CLOCKSOURCE_ACLK;
```

```
Modeparam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_1;
Modeparam.timerInterruptEnable_TAIE = TIMER_A_TAIE_INTERRUPT_DISABLE;
Modeparam.timerClear = TIMER_A_DO_CLEAR;
Modeparam.startTimer = false;
Timer_A_initContinuousMode(TIMER_A0_BASE, &Modeparam);

Timer_A_clearCaptureCompareInterrupt(TIMER_A0_BASE,
    TIMER_A_CAPTURECOMPARE_REGISTER_0);
Timer_A_initCaptureModeParam CompParam = {0};
CompParam.captureRegister = TIMER_A_CAPTURECOMPARE_REGISTER_0;
CompParam.captureMode = TIMER_A_CAPTUREMODE_FALLING_EDGE;
CompParam.captureInputSelect = TIMER_A_CAPTURE_INPUTSELECT_CCIxA;
CompParam.synchronizeCaptureSource = TIMER_A_CAPTURE_SYNCHRONOUS;
CompParam.captureInterruptEnable = TIMER_A_CAPTURECOMPARE_INTERRUPT_ENABLE;
CompParam.captureOutputMode = TIMER_A_OUTPUTMODE_OUTBITVALUE;
Timer_A_initCaptureMode(TIMER_A0_BASE, &initCompParam);

Timer_A_startCounter(TIMER_A0_BASE, TIMER_A_CONTINUOUS_MODE);
```

Q43. Lignes 45, 46, 47 et 48 du DT12

Q44. À chaque interruption déclenchée par une capture sur le capteur à effet Hall, le temps de comptage est obtenu en soustrayant la valeur actuelle du registre du Timer notée Reg et sa valeur lors de l'interruption précédente stockée dans l'entier timervalue.
La soustraction Reg-timervalue donne un résultat positif sauf si le Timer est arrivé au maximum et repassé à 0 entre 2 captures, dans ce cas le résultat est négatif. Le registre du Timer étant 16 bits, le résultat correct devrait être 2^{16} – Reg + timervalue, ou encore 2^{16} – (Reg – timervalue) ce qui correspond au codage **en complément à 2** du résultat négatif de la soustraction effectivement faite dans le code.
Le résultat de la soustraction est donc correct même en cas de débordement du registre du Timer.

Q45. En suivant la procédure décrite DT10 :
N = fBRCLK/baud rate = 8000000/9600 = 833,333 N > 16 on utilise l'oversampling UCOS16=1
clockPrescalar = INT(N/16) = 52, correspond à la ligne 78 du programme.
firstModReg = INT([(N/16) – INT(N/16)] × 16) = 0,0625*16 = 1, correspond à la ligne 79
Partie fractionnaire de N = 0,333, soit UCBRSx = secondModReg = 0x49, correspond à la ligne 80.

Q46. Pour une vitesse de 9600 bauds, 1 bit = 0,104 ms.
Un caractère = 10 bits (8 données+start+stop) = 1,04 ms
Le code-barres compte 3 lettres et 4 chiffres = 7 caractères + 0x0D = 8 caractères
Une trame = 8 caractères + 7 IDLE = 8*1,04+7*4,5 = 40 ms

Q47. Une interruption est une suspension temporaire de l'exécution d'un programme informatique par le microprocesseur afin d'exécuter un programme prioritaire (appelé service d'interruption).
Le vecteur TIMER0_A0_VECTOR et la routine Timer_A0_ISR correspondent à la capture d'un évènement sur le timer et sont déclenchés par le passage de l'aimant devant le capteur à effet Hall
Le vecteur PORT1_VECTOR et la routine Port_1_ISR correspondent à la détection d'un changement d'état sur une broche du microcontrôleur et sont déclenchés par le capteur inductif.
Le vecteur USCI_A0_VECTOR et la routine USCI_A0_ISR correspondent à la liaison UART et sont déclenchés par l'arrivée d'un caractère lors de la lecture du code-barres

Q48. La détection des traverses et la lecture de leur code-barres sont simultanées. Il y a un risque de déclenchement simultané des deux interruptions. Le risque est moindre avec l'interruption de mesure de vitesse mais pas totalement absent.

D'après le DT13 l'interruption UART de lecture du code-barres est prioritaire devant l'interruption de capture du timer et devant celle du capteur inductif.
Dans le cas où les 3 interruptions seraient simultanées, la lecture d'un caractère du code-barres est prioritaire. Puis vient l'interruption Timer sur le capteur à effet Hall et en dernier l'interruption sur le capteur inductif.
L'interruption UART se déclenche à chaque caractère reçu (les caractères sont espacés de 4,5 ms).
L'acquisition d'un caractère prend 1,04 ms
Une instruction s'exécute en maximum 6 cycles d'horloge soit 750 ns.
Pour que le temps d'exécution d'une des deux autres interruptions soit significatif il faudrait qu'elle comporte plusieurs centaines d'instructions, ce qui n'est pas réaliste.
Après chaque caractère les deux autres interruptions sont prises en compte et n'allongent pas significativement le temps d'acquisition des données. Ce temps est bien de l'ordre de la milliseconde.

Q49. Un parcours optimisé du tableau de code-barres permettant de synchroniser le numéro de traverse était attendu.

```
unsigned int barcode_synchro(const char *barcode){
    unsigned int taille_barcode=0,trouve=0,num_traverse=0;
    taille_barcode = strlen(barcode);
    while((trouve==0)&&(num_traverse<taille_barcode)){
        if(strcmp(barcode,traverse_bc[num_traverse])==0){
            trouve=1;
        }
        Num_traverse++;
    }
    return num_traverse;
}
```

Q50. Le traitement le plus long correspond à la lecture du code-barres des traverses et au parcours du tableau traverse_bc pour associer le code-barres lu au numéro de traverse. Cette opération est effectuée dans la fonction main à chaque fin de lecture d'un code-barres par les lignes 17 à 23.
Pour minimiser le temps d'exécution, le parcours du tableau de code-barres n'est effectué que si le code-barres lu ne correspond pas à celui du numéro de traverse actuel. S'il n'y a pas d'erreur, l'exécution du test sans parcours du tableau est très courte. Les autres calculs de mise à jour de la position sont très rapides. Le numéro de traverse et la position sont mis à jour dans l'interruption GPIO sans délai notable.

Q51. D'après la question 39, la fréquence maximum de détection du capteur à effet Hall est de 18 Hz soit une période d'environ 55 ms, Les traverses sont positionnées tous les 2 m, à la vitesse maximum, leur période de détection est d'environ 90 ms.
La question 45 donne une durée de 40 ms pour la réception d'un code-barres. La question 48 permet de montrer que la mise à jour de la position est de l'ordre de la milliseconde.
Le traitement le plus long est le parcours du tableau des code-barres en cas d'erreur de détection. Ce traitement est supposé ne pas dépasser la milliseconde.
L'acquisition et la mise à jour de la position pourra donc être au maximum de l'ordre de 42 ms ce qui est en dessous de la période de détection du capteur à effet Hall et des traverses. Le temps d'acquisition d'une position est donc compatible avec la fréquence de mise à jour des données des capteurs.

Q52. La question précédente permet de montrer que la mise à jour de la position à chaque traverse est de l'ordre de 42 ms ce qui est compatible avec un suivi temps réel de la position de la capsule.
A la vitesse maximum, la transmission des certificats ne pose pas de problème puisque la correction de la position s'effectue toutes les 90 ms soit suffisamment régulièrement pour un certificat toutes les 500 ms.
À la vitesse minimum, le rafraichissement de la position est de l'ordre de 5 Hz par mesure de vitesse et de 3,5 Hz par comptage des traverses soit 286 ms environ. Il y a peu de rafraichissements de la position entre 2 certificats, le suivi en temps réel sera plus délicat mais les glissements moins nombreux donc le suivi de la position par mesure de vitesse sera plus précis et la transmission d'un certificat a une fréquence de 2 Hz est réaliste.