

Informatique industrielle

TD, TP et mini-projet – ATmega328P

A. RICCA, H. HADJ-ABDELKADER & S. Otmane

ATMega328P and Arduino Uno Pin Mapping

Arduino function				Arduino function
reset	(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13) analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12) analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11) analog input 3
digital pin 2	(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10) analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9) analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8) analog input 0
VCC	VCC	7	22	GND GND
GND	GND	8	21	AREF analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5) digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4) digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3) digital pin 11(PWM)
digital pin 7	(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2) digital pin 10 (PWM)
digital pin 8	(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1) digital pin 9 (PWM)

Digital Pins 11, 12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17, 18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

BUT GEII – programmation par registres

Table des matières

1	Organisation et règles de travail	2
2	Travaux Dirigés	3
3	Travaux Pratiques	5
4	Mini-projet : mesure de vitesse et du sens de rotation	8
4.1	Principe du banc simulé	8
4.2	Étape 1 – Générer un encodeur A/B simulé	9
4.3	Étape 2 – Déterminer le sens de rotation	9
4.4	Étape 3 – Méthode A : comptage sur une fenêtre temporelle	9
4.5	Étape 4 – Méthode B : mesure de la période entre deux fronts	10
4.6	Étape 5 – Établir une vitesse de référence	10
4.7	Étape 6 – Comparaison expérimentale des deux méthodes	11
4.8	Étape 7 – Intégration finale	11
4.9	Livrables du mini-projet	11

Organisation et règles de travail

Objectifs

Mettre en pratique GPIO, interruptions, timers, PWM, ADC et signaux d'encodeur, puis réinvestir ces notions dans un mini-projet de mesure de vitesse et de sens de rotation.

Pour chaque exercice, les registres, les bits et les calculs doivent être justifiés. Les fonctions Arduino de haut niveau ne doivent pas remplacer la configuration des périphériques par registres. La liaison série peut être utilisée pour le débogage et l'affichage des résultats.

Travaux Dirigés

TD1 – GPIO et masques

1. Convertir `DDRB=0x28` en binaire et identifier les sorties.
2. Mettre PB4 à 1 puis à 0 sans modifier les autres bits.
3. Configurer PB0/PB1 en entrées avec pull-up et PB5 en sortie.
4. LED PB5 commandée par bouton PB0 vers GND : écrire la condition.
5. Établir la table logique de la sirène PD5 active si PB0 et PB1 sont appuyés.
6. Reprendre la bascule RS du TP1 : R=PB0, S=PB3, Q=PB4, /Q=PB5 et tenir compte des entrées actives bas.
7. Expliquer la différence entre `PORTB=...` et une modification par masque.

TD2 – Interruptions externes et PCINT

1. Configurer INT0 sur front montant, puis descendant.
2. Expliquer le rôle de EICRA, EIMSK, EIFR et du bit I de SREG.
3. Expliquer *volatile*.
4. Pourquoi éviter `Serial.print` ou un traitement long dans une ISR ?
5. Proposer un anti-rebond non bloquant.
6. Comparer INT0/INT1 et PCINT.
7. Un capteur produit 20000 fronts/s : estimer le temps disponible entre deux événements.

TD3 – Timers et CTC

On suppose $f_{CPU} = 16$ MHz.

1. Calculer T_{tick} pour $N = 1, 8, 64, 256, 1024$.
2. Trouver OCR1A pour 1 kHz, 100 Hz et 10 Hz en CTC.
3. Timer0 toggle CTC : déterminer $(N, OCR0A)$ pour 40 kHz, 10 kHz et 1 kHz.
4. Comparer Timer0 8 bits et Timer1 16 bits.
5. Calculer la durée maximale avant overflow du Timer1 pour plusieurs prescalers.

TD4 – PWM

1. Fast PWM Timer0, $N = 64$: fréquence.
2. OCR approché pour 20, 50 et 70%.

3. Timer1 mode 14 : choisir ICR1 pour 20 kHz, $N = 1$.
4. Calculer OCR1A pour 30% et 75%.
5. Expliquer le rôle d'un driver de puissance et pourquoi le moteur ne doit pas être branché directement au microcontrôleur.

TD5 – ADC

1. Résolution pour $V_{REF} = 5$ V et 3,3 V.
2. Codes pour 1 V, 2 V, 4 V avec $V_{REF} = 5$ V.
3. Tension correspondant aux codes 100, 400, 800.
4. Capteur $V = 0,01T + 0,5$: code à 25 degrés C et formule inverse $T(ADC)$.
5. Discuter choix du prescaler ADC et cadence d'échantillonnage.
6. Pourquoi une acquisition périodique est-elle préférable pour une commande ?

TD6 – Encodeur et mesure de vitesse

1. Encodeur 500 périodes/tour : comptes/tour en x1, x2, x4.
2. Dessiner A/B pour les deux sens.
3. 240 comptes x4 en 100 ms : calculer RPM.
4. À 60 RPM et 2000 comptes/tour : fréquence des fronts.
5. Période entre comptes 500 μ s, 500 comptes/tour : RPM.
6. Quantifier la résolution en RPM d'une fenêtre de 100 ms.
7. Comparer mesure par fenêtre et mesure de période à faible et forte vitesse.

Travaux Pratiques

TP1 – Révision GPIO

Travail demandé

Valider chaque étape à la LED ou à l'oscilloscope et joindre le code commenté.

1. PB3 à 1 puis 0.
2. Refaire sur PORTD.
3. Clignotement PB5 puis PB1.
4. PB4 commandée par PB0.
5. Sirène PD5 si PB0 et PB1 actifs.
6. Bascule RS.
7. Réécrire sans affectations globales aux ports.

TP2 – Interruptions externes et événements

Objectifs

Mettre en œuvre INT0, observer le caractère événementiel d'une interruption, mettre en évidence le rebond d'un bouton et construire un traitement anti-rebond sans utiliser de générateur basse fréquence.

Travail demandé

Le bouton est utilisé comme source d'événements. Il est connecté à INT0 avec une configuration adaptée (par exemple entrée avec pull-up et bouton vers GND). Les traitements longs doivent rester dans la boucle principale.

- Étape 1. Première interruption.** Configurer INT0 sur front descendant. À chaque appui sur le bouton, l'ISR doit uniquement signaler l'événement. Dans la boucle principale, inverser l'état de la LED PB5.
- Étape 2. Observation des rebonds.** Remplacer l'inversion de LED par un compteur d'interruptions. Réaliser un seul appui physique et observer le nombre d'interruptions détectées. Répéter plusieurs fois. Expliquer pourquoi un bouton mécanique peut produire plusieurs fronts.
- Étape 3. Anti-rebond temporel.** Mettre en place une stratégie de validation temporelle : après un événement accepté, les nouveaux fronts trop proches temporellement sont ignorés. Justifier la durée choisie et vérifier qu'un appui physique ne produit plus qu'un événement validé.

- Étape 4. Validation quantitative sans GBF.** Initialiser un compteur d'appuis validés. Effectuer exactement 10 appuis distincts sur le bouton. Le programme doit afficher ou signaler exactement 10 événements. Comparer le compteur brut d'interruptions et le compteur d'appuis validés.
- Étape 5. Séparation ISR / programme principal.** L'ISR ne doit ni afficher le compteur, ni réaliser de temporisation longue. Elle met à jour une information minimale (compteur ou flag). L'affichage série ou le traitement est réalisé dans la boucle principale. Expliquer l'intérêt de cette organisation.
- Étape 6. Extension PCINT.** Connecter un second bouton sur une broche disposant de PCINT. Configurer l'interruption sur changement de broche et distinguer appui et relâchement en lisant l'état de l'entrée. Comparer le comportement avec INT0.

TP3 – Timers et base de temps sans delay()

Objectifs

Construire une base de temps périodique à partir du Timer1 en mode CTC et utiliser cette base de temps pour ordonnancer plusieurs tâches sans blocage.

À retenir

Un **flag logiciel** est une variable servant de message entre une ISR et le programme principal. Exemple : toutes les 100 ms, l'ISR positionne `flag_100ms=1`. La boucle principale détecte cette valeur, remet le flag à 0 puis exécute la tâche périodique. L'ISR du Timer est bien l'interruption ; le flag n'est qu'une information logicielle produite par cette ISR.

- Étape 1.** Configurer Timer1 en mode CTC pour générer une interruption toutes les 1 ms. Justifier le prescaler et la valeur de OCR1A.
- Étape 2.** Dans l'ISR du Timer1, construire un compteur logiciel de millisecondes.
- Étape 3.** Faire clignoter PB5 avec une période de 1 s (changement d'état toutes les 500 ms), sans utiliser `delay()`.
- Étape 4.** Ajouter une seconde tâche périodique, par exemple l'inversion d'une deuxième LED toutes les 2 s. Vérifier que les deux tâches coexistent sans blocage.
- Étape 5.** Créer un flag logiciel `flag_100ms`. Toutes les 100 interruptions de 1 ms, l'ISR positionne ce flag à 1. Dans la boucle principale, lorsque le flag vaut 1 : le remettre à 0 puis inverser une sortie de test. Cette structure sera réutilisée dans le TP6 pour la mesure de vitesse par fenêtre.
- Étape 6.** Vérifier à l'oscilloscope la période de la sortie de test et comparer la valeur mesurée à la valeur théorique.

TP4 – CTC et PWM

Objectifs

Valider la génération temporelle en CTC, puis utiliser uniquement les deux configurations PWM retenues dans le module : Fast PWM 8 bits et Fast PWM Timer1 mode 14 avec TOP programmable.

- Étape 1. Timer0 en CTC.** Générer successivement 40 kHz, 10 kHz et 1 kHz sur OC0A. Pour chaque fréquence, justifier le prescaler et OCR0A, puis vérifier la fréquence à l'oscilloscope.

- Étape 2. Timer1 en CTC.** Générer 40 kHz sur OC1A. Comparer le calcul à celui réalisé avec Timer0.
- Étape 3. Fast PWM 8 bits.** Configurer une sortie en Fast PWM et réaliser un rapport cyclique $D = 0,7$. Mesurer la fréquence et le rapport cyclique à l'oscilloscope. Modifier ensuite OCR pour observer l'effet sur D .
- Étape 4. Timer1 Fast PWM – mode 14.** Utiliser ICR1 comme TOP. Choisir une fréquence PWM imposée par l'enseignant, calculer ICR1, puis réaliser $D = 0,3$. Vérifier expérimentalement la fréquence et le rapport cyclique.

TP5 – ADC

1. Potentiomètre ADC0, polling.
2. Conversion code-tension.
3. Passage à interruption ADC.
4. Échantillonnage 100 Hz.
5. ADC vers consigne PWM.
6. Validation oscilloscope.

Mini-projet : mesure de vitesse et du sens de rotation

Objectifs

Réaliser un mini-projet de tachymétrie numérique. L'encodeur incrémental A/B est d'abord **simulé par des signaux numériques générés à l'aide d'un Timer**. Le système doit déterminer le **sens de rotation** et mesurer la **vitesse** avec les deux méthodes étudiées dans le cours : comptage sur fenêtre temporelle et mesure de période entre deux fronts. Les résultats des deux méthodes sont ensuite comparés.

Travail demandé

La mesure de position angulaire n'est pas demandée. Le compteur d'événements n'est qu'une variable intermédiaire nécessaire à la mesure de vitesse. Le résultat final attendu est : **sens de rotation + vitesse en tr/min**.

4.1 Principe du banc simulé

Le mini-projet comporte deux fonctions :

1. un **simulateur d'encodeur** produisant deux signaux A et B en quadrature ;
2. un **tachymètre** recevant A et B et estimant sens et vitesse.

Le simulateur peut être réalisé avec un Timer configuré en CTC. À chaque interruption périodique, une machine d'états fait évoluer les sorties A/B selon la séquence de quadrature. Par exemple, pour un sens :

$$00 \rightarrow 01 \rightarrow 11 \rightarrow 10 \rightarrow 00,$$

et pour l'autre sens :

$$00 \rightarrow 10 \rightarrow 11 \rightarrow 01 \rightarrow 00.$$

La fréquence du Timer détermine la vitesse simulée. Une entrée (bouton ou commutateur) peut sélectionner le sens.

Attention

Le Timer utilisé pour générer les signaux A/B doit être choisi de façon à ne pas entrer en conflit avec le Timer nécessaire à la mesure de période. Avant de programmer, établir un tableau : *fonction – ressource – broche – registre*. Une solution pédagogique possible consiste à utiliser Timer0 ou Timer2 pour le simulateur et Timer1 pour la mesure temporelle. Le choix exact est à valider en fonction du câblage de la séance.

4.2 Étape 1 – Générer un encodeur A/B simulé

1. Choisir deux GPIO de sortie pour A et B.
2. Configurer un Timer en CTC pour obtenir une interruption périodique de simulation.
3. Dans l'ISR du Timer, faire évoluer une variable d'état parmi 0, 1, 2, 3.
4. À partir de cet état, générer les niveaux logiques A et B correspondant à la quadrature.
5. Ajouter une variable ou une entrée permettant d'inverser l'ordre des états et donc le sens simulé.
6. Observer A et B à l'oscilloscope. Vérifier le déphasage et les deux séquences de sens.

Méthode de travail

Commencer à faible fréquence afin de faciliter l'observation. Une fois la quadrature validée, augmenter progressivement la fréquence de simulation. Noter pour chaque réglage la fréquence des événements générés.

4.3 Étape 2 – Déterminer le sens de rotation

On utilisera un décodage simple x1 : un front de A déclenche l'acquisition et l'état de B permet de déterminer le sens.

1. Relier A à une entrée d'interruption adaptée, par exemple INT0.
2. Relier B à une entrée GPIO.
3. Configurer l'interruption sur le front retenu de A.
4. Dans l'ISR, lire B et mettre à jour une variable **sens** valant par exemple +1 ou -1.
5. Afficher périodiquement **Sens +** ou **Sens -** dans la boucle principale.
6. Inverser le sens du simulateur et vérifier que le tachymètre détecte correctement l'inversion.

À retenir

La position n'est pas calculée. L'information de quadrature est utilisée uniquement pour déterminer le signe de la vitesse. On pourra donc afficher une vitesse signée :

$$RPM_{\text{signe}} = \text{sens} \times |RPM|.$$

4.4 Étape 3 – Méthode A : comptage sur une fenêtre temporelle

On suppose que le signal exploité fournit N_c événements comptés par tour dans le mode de décodage choisi. Si n événements sont comptés pendant une fenêtre T_w :

$$RPM = \frac{60n}{N_c T_w}.$$

1. Dans l'ISR associée à A, incrémenter un compteur **pulse_count** à chaque événement retenu.
2. Construire une fenêtre temporelle, par exemple $T_w = 100$ ms, à partir de la base de temps disponible.
3. Toutes les T_w , copier de manière cohérente le nombre d'impulsions comptées, remettre le compteur de fenêtre à zéro puis positionner un flag de calcul.
4. Dans la boucle principale, calculer la vitesse en tr/min à partir de n , N_c et T_w .
5. Appliquer le signe déterminé avec la voie B.
6. Répéter les mesures pour plusieurs fréquences du simulateur.

7. Tester au moins deux valeurs de fenêtre, par exemple 100 ms et 500 ms. Comparer stabilité, résolution et rapidité de mise à jour.

La résolution théorique liée à un événement de comptage vaut :

$$\Delta RPM = \frac{60}{N_c T_w}.$$

L'étudiant doit calculer cette résolution pour les fenêtres testées.

4.5 Étape 4 – Méthode B : mesure de la période entre deux fronts

La seconde méthode consiste à mesurer le temps ΔT séparant deux événements consécutifs :

$$RPM = \frac{60}{N_c \Delta T}.$$

Si la durée est mesurée avec un Timer et si M ticks de durée T_{tick} séparent deux fronts :

$$RPM = \frac{60}{N_c M T_{tick}}.$$

1. Configurer Timer1 comme base de temps de mesure avec un prescaler justifié.
2. Mesurer le nombre de ticks séparant deux fronts successifs de A. La mise en œuvre peut utiliser une lecture temporelle associée à l'interruption externe ou, si le câblage le permet, l'unité Input Capture ICP1.
3. Stocker la période mesurée dans une variable partagée correctement déclarée.
4. Dans la boucle principale, convertir la période en tr/min.
5. Appliquer le signe fourni par la voie B.
6. Réaliser les mesures pour les mêmes vitesses simulées que dans la méthode A.
7. Prévoir une détection d'arrêt : si aucun nouveau front n'arrive pendant un temps maximal fixé, imposer $RPM = 0$.

Attention

Avec la méthode par période, lorsque le signal s'arrête, aucune nouvelle période n'est mesurée. Sans timeout, le programme risque donc de conserver indéfiniment la dernière vitesse calculée.

4.6 Étape 5 – Établir une vitesse de référence

La fréquence du signal simulé est connue puisqu'elle est construite à partir d'un Timer. Elle permet de calculer une vitesse théorique de référence.

Si la fréquence des événements comptés vaut f_e :

$$RPM_{ref} = \frac{60 f_e}{N_c}.$$

Pour chaque réglage du simulateur :

1. calculer f_e à partir des paramètres du Timer générateur ;
2. en déduire RPM_{ref} ;
3. comparer cette valeur aux deux mesures réalisées par le tachymètre.

4.7 Étape 6 – Comparaison expérimentale des deux méthodes

Construire un tableau de résultats de la forme suivante :

Essai	Sens	RPM_{ref}	$RPM_{fenetre}$	$RPM_{periode}$	Observations
1	+				
2	+				
3	+				
4	-				
5	-				

Pour chaque méthode, calculer si possible l'erreur relative :

$$\varepsilon = \frac{|RPM_{mes} - RPM_{ref}|}{RPM_{ref}} \times 100\%.$$

La discussion finale doit répondre aux questions suivantes :

1. Quelle méthode donne la mesure la plus stable aux vitesses élevées ?
2. Quelle méthode est la plus intéressante lorsque les fronts deviennent espacés ?
3. Quel est l'effet de la durée T_w sur la méthode par fenêtre ?
4. Quel est l'effet du prescaler et de la résolution du Timer sur la méthode par période ?
5. Comment chaque méthode détecte-t-elle l'arrêt ?
6. Que se passe-t-il lors d'un changement brutal de vitesse ?
7. Les deux méthodes donnent-elles le même signe lors de l'inversion du simulateur ?

4.8 Étape 7 – Intégration finale

Le programme final doit fonctionner sans temporisation bloquante et fournir périodiquement au minimum :

- le sens détecté ;
- la vitesse de référence simulée ;
- la vitesse obtenue par comptage sur fenêtre ;
- la vitesse obtenue par mesure de période ;
- éventuellement les erreurs relatives des deux méthodes.

Travail demandé

Le mini-projet est considéré comme validé lorsque les deux sens sont correctement détectés, que les deux méthodes de mesure fonctionnent pour plusieurs vitesses simulées, que l'arrêt est traité correctement et qu'une comparaison argumentée des performances est fournie.

4.9 Livrables du mini-projet

Le compte rendu doit contenir :

1. le schéma de câblage et le tableau d'allocation des ressources ;
2. les calculs de configuration du Timer générateur ;
3. les chronogrammes A/B observés ;
4. le calcul de RPM_{ref} ;
5. le principe et les calculs de la méthode par fenêtre ;

6. le principe et les calculs de la méthode par période ;
7. le tableau comparatif des mesures ;
8. une discussion des erreurs, de la résolution et du temps de réponse ;
9. le code commenté et structuré en ISR courtes + traitements dans la boucle principale.

Projet intégrateur – Pour aller plus loin

Le TP6 constitue désormais le mini-projet centré sur la tachymétrie. Une extension ultérieure peut intégrer un véritable moteur et son encodeur, une consigne analogique par ADC, une PWM de commande et éventuellement une régulation. Cette extension n'est pas nécessaire pour valider le mini-projet de mesure de vitesse.