ARD3	ESP32 : Interruption et Timer	Durée 3h.

Le développement se fait sous l'IDE Arduino. Choisir la carte NodeMCU-32S

Type de carte: "NodeMCU-32S"
Upload Speed: "921600"
Flash Frequency: "80MHz"
Port: "COM13"

Les projets sont à sauvegarder dans l'espace de travail sous KWARTZ, sous répertoire **Arduino_ESP32**.

1 – Présentation interruption

Une interruption du programme peut être provoquée par un événement sur une patte du microcontrôleur ou par un périphérique (UART, TIMER, SPI …) interne au microcontrôleur.

Lors d'une interruption, un sous-programme d'interruption est exécuté (concrètement le programme se branche à une adresse particulière).

En règle générale une interruption peut être validée ou non (à l'exception des interruptions non masquables NMI), et un bit drapeau (Flag) passe à 1 pour mémoriser la demande d'interruption.

Sous Arduino, des fonctions destinées à simplifier la gestion des interruptions ont été développées.

La fonction '*attachInterrupt*' va permettre d'associer une interruption (numéro ou patte) à un sous-programme à exécuter (ISR), suivant un mode défini.

`attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)` (recommended)
`attachInterrupt(interrupt, ISR, mode)` (not recommended)
`attachInterrupt(pin, ISR, mode)` (Not recommended. Additionally, this syntax only works on Arduino SAMD Boards, Uno WiFi Rev2, Due, and 101.)

Parameters

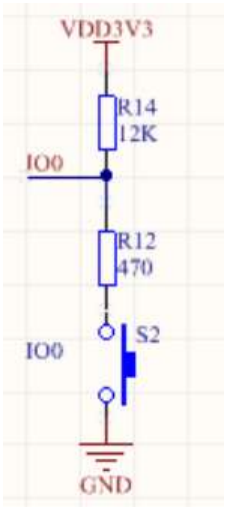
`interrupt`: the number of the interrupt. Allowed data types: `int`.
`pin`: the Arduino pin number.
`ISR`: the ISR to call when the interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an interrupt service routine.
`mode`: defines when the interrupt should be triggered. Four constants are predefined as valid values:

- **LOW** to trigger the interrupt whenever the pin is low,
- **CHANGE** to trigger the interrupt whenever the pin changes value
- **RISING** to trigger when the pin goes from low to high,
- **FALLING** for when the pin goes from high to low.

2 – Interruption externe

Remarques :

- Le bouton BOOT peut être utiliser comme bouton poussoir général (lorsque le µP tourne normalement). Le bouton BOOT est câblé sur GPIO0, de la manière suivante :



- Pour la carte ESP32, il est préconisé d'ajouter l'attribut IRAM_ATTR au sous-programme d'interruption pour forcer le compilateur à placer la gestion des interruptions en IRAM (instruction RAM).

⇨ Pour le programme ci-dessous, indiquer :

- Quelle action va provoquer l'interruption
- Quel sous-programme va être exécuter au moment de l'interruption.

⇨ Tester le programme (demander une led au prof.)

Remarque : lors de l'appui sur un bouton poussoir, il peut exister des rebonds mécaniques provoquant plusieurs fronts montants et descendants sur le signal électrique.

```
#include <Arduino.h>

#define BP 0
#define LED0 2
#define LED1 4

void IRAM_ATTR inter(void);

void setup() {
  pinMode(BP,INPUT);
  pinMode(LED0,OUTPUT);
  pinMode(LED1,OUTPUT);
  attachInterrupt(digitalPinToInterrupt(BP), inter, FALLING);

}

void loop() {
  digitalWrite(LED1,1);
  delay(1000);
  digitalWrite(LED1,0);
  delay(1000);
}

void IRAM_ATTR inter(){
  if (digitalRead(LED0)==0) digitalWrite(LED0,1); else digitalWrite(LED0,0);
}
```

⇨ Modifier le programme pour avoir une interruption sur front montant. Tester.

3 – Interruption par Timer interne

L'ESP32 dispose de plusieurs Timers (compteurs). La documentation du microcontrôleur est donnée en annexe. La partie concernée par les questions suivantes se trouve à partir de la page 487.

⇨ Répondre aux questions suivantes :

Nombre de Timers généraux de 64 bits dans l'ESP32	
Nombre de bits du pré-diviseur	
Fréquence normale sur le pré diviseur	
Possibilité de comptage et décomptage	OUI / NON
Possibilité d'un auto rechargement du compteur après une alarme	OUI / NON
Possibilité d'interruption de programme sur alarme	OUI / NON

Le programme de test ci-dessous est fourni sur le site suivant :
https://techtutorialsx.com/2017/10/07/esp32-arduino-timer-interrupts/

```
volatile int interruptCounter;
int totalInterruptCounter;

hw_timer_t * timer = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

void IRAM_ATTR onTimer() {
  portENTER_CRITICAL_ISR(&timerMux);
  interruptCounter++;
  portEXIT_CRITICAL_ISR(&timerMux);
}

void setup() {
  Serial.begin(115200);
  timer = timerBegin(0, 80, true);
  timerAttachInterrupt(timer, &onTimer, true);
  timerAlarmWrite(timer, 1000000, true);
  timerAlarmEnable(timer);
}

void loop() {
  if (interruptCounter > 0) {
    portENTER_CRITICAL(&timerMux);
    interruptCounter--;
    portEXIT_CRITICAL(&timerMux);
    totalInterruptCounter++;
    Serial.print("An interrupt as occurred. Total number: ");
    Serial.println(totalInterruptCounter);
  }
}
```

⇨ Tester le programme sur la carte (ouvrir le moniteur).

La fonction ***timerBegin*** accepte 3 arguments : numéro du timer, la valeur du prédiviseur, comptage/décomptage

hw_timer_t * timerBegin(uint8_t num, uint16_t divider, bool countUp)

La fonction ***timerAttachInterrupt*** associe la source de l'interruption avec le sous-programme à exécuter.

void timerAttachInterrupt(hw_timer_t *timer, void (*fn)(void), bool edge)

La fonction ***timerAlarmWrite*** définit le nombre de périodes d'horloge avant l'alarme (avec auto rechargement possible).

void timerAlarmWrite(hw_timer_t *timer, uint64_t alarm_value, bool autoreload)

La fonction ***timerAlarmEnable*** valide l'alarme.

void timerAlarmEnable(hw_timer_t *timer)

Remarque : Les fonctions portENTER_CRITICAL(_ISR) et portEXIT_CRITICAL(_ISR) sont préconisées lorsque des variables sont à la fois modifiées par un sous-programme d'interruption et un autre sous-programme (ou programme principal). Il est probable (à confirmer) que les interruptions soient bloquées entre ces 2 fonctions pour éviter tout conflit.

⇨ Déterminer la fréquence d'horloge du compteur (après division) dans le cas du programme fourni.

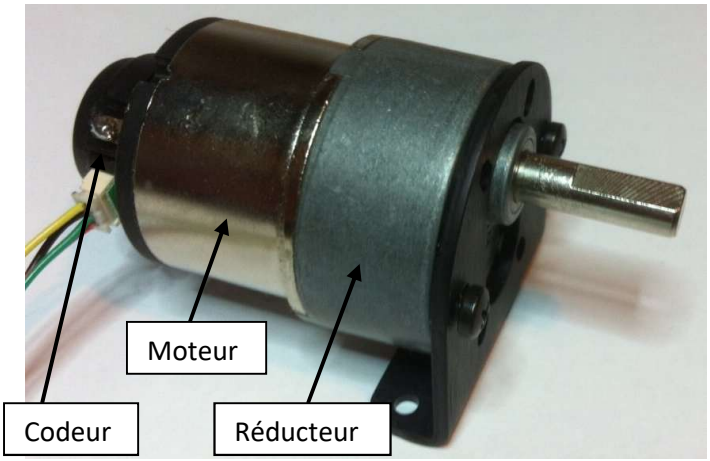
⇨ Déterminer le nom du sous-programme d'interruption dans le cas du programme fourni.

⇨ Justifier l'interruption toutes les 1s.

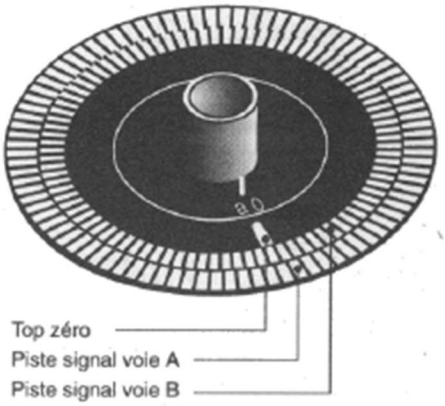
⇨ Proposer une modification du programme pour avoir une interruption toutes les 10s, en modifiant le pré diviseur uniquement dans un premier temps, puis la valeur écrite dans le timer uniquement dans un deuxième temps. Tester.

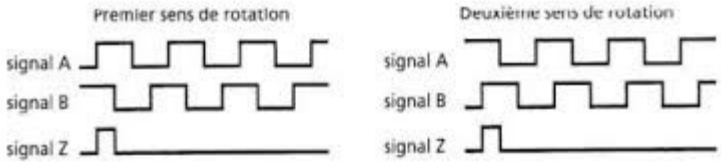
4 – Mesure de vitesse de rotation

Dans cette partie, il faut utiliser la maquette de commande d'un moteur à courant continu.



Etude du codeur





Un codeur incrémental dispose de 2 pistes et délivre 2 signaux : A et B.
Les signaux A et B sont en quadrature, c'est-à-dire qu'ils sont déphasés de ¼ de période.

Les principales caractéristiques du motoréducteur sont les suivantes

Hardware:
- 1 x GH Motor 7.2vdc 50:1 175rpm (6mm shaft) (GHM-04)
- 1 x Quadrature Motor Encoder (QME-01)

Goal:
- Install a wheel encoder on a motor.

Specs:
- GHM-04 motor RPM under load = 7500 rpm
- Encoder = 120 cycles per revolution
- Encoder = 480 quadrature counts per revolution
- Frequency = 15khz
- 3428 quadrature counts per inch with a 2.5" tire

Le réducteur a un rapport de 50 :1 ⇨ lorsque l'axe en sortie du réducteur fait 1 tour, l'axe moteur en fait 50
Le codeur délivre 120 impulsions, sur une voie, lorsque le moteur fait 1 tour.

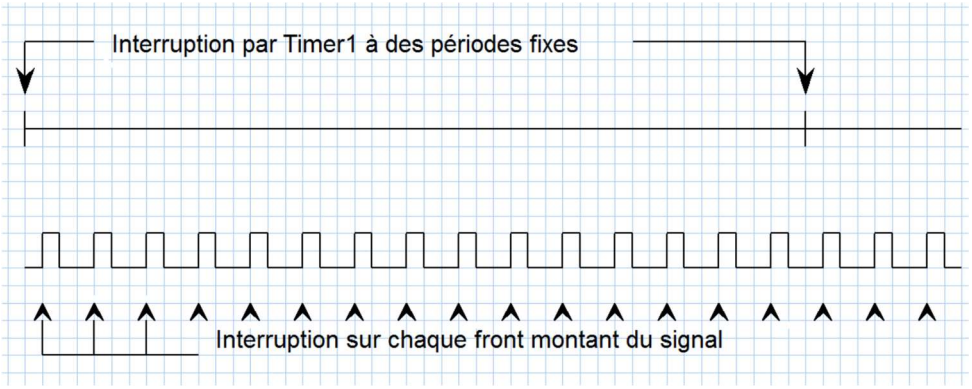
⇨ Déterminer la vitesse de rotation du moteur lorsqu'on relève une fréquence de 10kHz sur une voie du codeur.

⇨ Déterminer la vitesse de rotation en sortie du réducteur, dans ces mêmes conditions.

Mesure de la vitesse avec une carte ARDUINO

Pour mesurer la fréquence, on compte le nombre d'impulsions fournies par le capteur pendant un temps fixe. Pour cela, on utilise 2 types d'interruption au niveau du microcontrôleur :

- Une interruption d'un timer, intégré au µP, à des intervalles de temps régulier
- Une interruption sur chaque front montant ou descendant du signal fourni par le capteur



Programme

Le programme incomplet est donné ci-après. La commande du moteur reste identique au TP ARD2.
Les fonctions portENTER_CRITICAL(_ISR) et portEXIT_CRITICAL(_ISR) ne sont pas utilisées ici.

A partir de l'analyse du programme :
⇨ A chaque front d'une voie du codeur, indiquer la variable qui est incrémentée dans le sous-programme d'interruption inter.
⇨ Indiquer à quel instant cette variable est remise à 0.
⇨ Indiquer quelle variable permet d'informer le programme principal qu'une nouvelle mesure est disponible.

La voie A du codeur est reliée sur GPIO26.

⇨ En s'inspirant des parties 2 et 3 de ce TP, compléter les fonctions **setup_inter_codeur** et **setup_timer** pour avoir une mesure de la vitesse toutes les secondes.

⇨ Tester le programme une fois complété.

⇨ Pour un rapport cyclique de 50% environ, relever à l'oscilloscope la fréquence sur une voie du codeur (voir implantation en annexe) et comparer par rapport à l'affichage sur le moniteur. Conclure

⇨ Chronométrer le temps pour que l'axe en sortie du réducteur fasse 10 tours. En déduire la vitesse de rotation de l'axe en sortie du réducteur, puis la vitesse de rotation du moteur et enfin la fréquence sur une voie du codeur. Vérifier la conformité avec les résultats affichés par le moniteur.

```
#define CODEUR_A 26

#define In1 13
#define In2 14
#define EnA 12
#define FREQ  20000
#define CHANEL0 0
```

```
#define RESOLUTION 10

void IRAM_ATTR inter(void);
void IRAM_ATTR onTimer(void);

void setup_cmd_moteur(void);
void MoteurA_Sens1(unsigned int vitesse);
void MoteurA_Sens2(unsigned int vitesse);

void setup_inter_codeur(void);
void setup_timer(void);

int INTER_CPT=0;
int FREQ_MOTEUR=0;
bool NEW_VAL=false;

int RAPPORT=0;
String ST;

hw_timer_t * timer = NULL;

void setup() {
  Serial.begin(115200);
  Serial.setTimeout(1);
  setup_cmd_moteur();
  setup_inter_codeur();
  setup_timer();
}

void loop() {
  ST="";
  if (Serial.available() > 0){
    ST=Serial.readString();
    RAPPORT=ST.toInt();
    MoteurA_Sens1(RAPPORT);
  }
  if (NEW_VAL==true){
    Serial.print("Fréquence codeur: ");
    Serial.println(FREQ_MOTEUR);
    NEW_VAL=false;
  }
}

void IRAM_ATTR inter(){
  INTER_CPT++;
}

void IRAM_ATTR onTimer() {
  FREQ_MOTEUR=INTER_CPT;
  INTER_CPT=0;
  NEW_VAL=true;
}
```

```
void setup_inter_codeur(void){
        à compléter…
}

void setup_timer(void){
        à compléter…
}

void setup_cmd_moteur() {
  pinMode(In1,OUTPUT);
  pinMode(In2,OUTPUT);
  pinMode(EnA,OUTPUT);
  ledcSetup(CHANEL0, FREQ, RESOLUTION);
  MoteurA_Sens1(0);
}

void MoteurA_Sens1(unsigned int vitesse){
  digitalWrite(In2,LOW);
  digitalWrite(EnA,HIGH);
  ledcAttachPin(In1, CHANEL0);
  ledcWrite(CHANEL0, vitesse);
}

void MoteurA_Sens2(unsigned int vitesse){

}
```