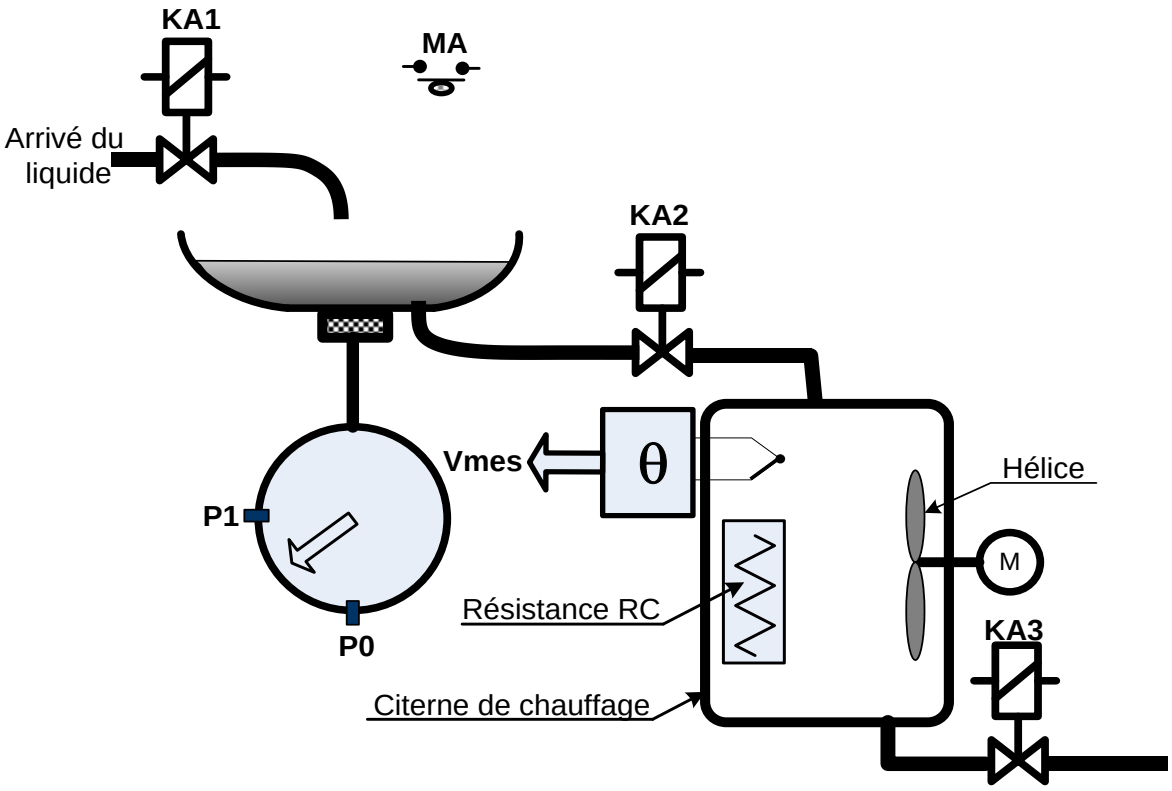
TD DSC- STM32F207

Dans une usine de produits agro-alimentaires une partie de processus de traitement d'un produit consiste à doser une quantité d'un liquide visqueux pour la porter à une température de 100°C. Ce système comporte :

- un bac de dosage
- une citerne de chauffage



Fonctionne du système

L'action sur le bouton poussoir de mise en marche MA déclenche le cycle suivant :

- Dosage du liquide : l'électrovanne KA1 autorise le remplissage du bac doseur jusqu'à la détection de la quantité souhaitée par le capteur P1.

- Lorsque P1 est actionné, KA1 interrompe le remplissage et autorise l'écoulement du liquide du bac vers la citerne de chauffage par l'intermédiaire de l'électrovanne KA2.

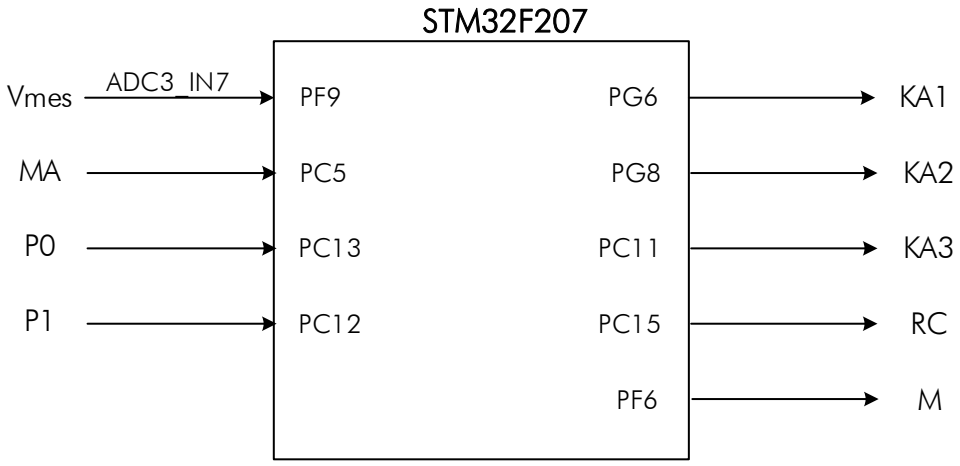
La fin de l'écoulement du liquide détectée par le capteur P0 entraîne à la fois :

- Le chauffage du liquide par la résistance RC.

- Le brassage du liquide par la rotation des hélices entraînées par le moteur M ; le moteur tourne à vitesse constante de 75% de sa valeur maximale.

Lorsque la température de chauffage atteint 100°C, le chauffage et le brassage seront arrêtés. La circulation du liquide vers la suite du processus est autorisée par l'électrovanne KA3 pendant une durée de 20s.

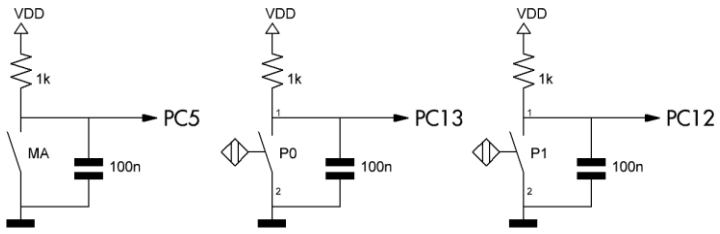
Le système est commandé par un microcontrôleur STM32F207. Le capteur de température LM35 délivre une tension Vmes comprise entre 0 et 1,5V pour des températures allant de 0 à 150ºC.



TRAVAIL DEMANDE

PARTIE 1 – GPIOs

Le schéma de connexion de l'interrupteur MA et les capteurs fin de course est donné par la figure suivant :



1. Quel est le rôle de la capacité de 100nf ?
2. Quel est le niveau logique en PC5 lorsque MA est fermé ?
3. Quel est le niveau logique en PC5 lorsque MA est ouvert ?
4. Faites les configurations nécessaires des lignes PC5, PC13 et PC12.
5. Les électrovannes KA1, KA2 et KA3 sont alimentées par une tension continue de 24V; la résistance chauffante RC est alimentée par une tension alternative de 220V.
    a. Proposer le schéma du circuit d'interface à intercaler entre le microcontrôleur et les actionneurs.
    b. Faites les configurations nécessaires des lignes PG6, PG8, PC11 et PC15.
6. Ecrire le pseudocode qui commande les électrovannes KA1 et KA2 lorsque P1 est actionné.

PARTIE 2 - ADC

Le capteur de température est connecté à la ligne PF9, donc au canal 7 de l'ADC3.

PF9	I/O	FT	(4)	TIM14_CH1, FSMC_CD, EVENTOUT	ADC3_IN7

1. On veut acquérir la température du liquide à 0,5°C près et en prenant $V_{REF+} = V_{DDA} = 3,3V$.
   a. Sur combien de bit doit-on coder la température du liquide pour garantir cette résolution ?
   b. Quelle est la résolution réelle du convertisseur pour une conversion sur 10 bits ?
   c. Que serai cette résolution pour une conversion sur 12 bits.
2. L'entrée V_{REF+} est alimentée par 1,5V. Déterminer la résolution du convertisseur pour une conversion sur 10 bits. Calculer l'erreur de quantification dans ce cas.

On prend dans la suite de cette partie : $V_{REF+} = V_{DDA} = 3,3V$ et $N = 10$ bits.

3. Calculer le temps d'acquisition T_S (en nombre de cycles) pour les paramètres suivants :
   $R_{AIN} = négligéable \;\; ; \;\; R_{ADC} = 2k\Omega \;\; ; \;\; C_{ADC} = 4pf \; ; \; F_{ADC} = 15Mhz \;\; ;$
4. Donner le temps de conversion total.
5. Ecrire le pseudocode de configuration de la broche PF9 en analogique.
6. Ecrire le code de la procédure init_ADC3() qui initialise le convertisseur ADC3 en mode simple conversion sur un seul canal ($F_{ADC} = 15Mhz$ et $T_S = 3cycles$).
7. Ecrire le pseudocode qui permet de faire l'acquisition de la température et de sauvegarder la valeur convertie dans la variable globale TempVal.

PARTIE 3 - TIMER

Le moteur de brasage M est un moteur à courant continu 24V – 2A, Commandé par un hacheur série (commande PWM).

Le TIMER10 du microcontrôleur STM32 génère sur la broche PF6 un signal MLI de fréquence 10Khz.

PF6				TIM10_CH1

1. Donner le schéma du montage du hacheur.
2. La tension moyenne aux bornes du moteur : $U_{moy} = \alpha \cdot E$, avec E : alimentation du moteur et α : la rapport cyclique.
   On veut une résolution sur le rapport cyclique de 0,1%. Faites les configurations nécessaires du TIMER10 pour générer le signal désiré.
   Indication : SYSCLK = 120Mhz.

PARTIE 4 – Programme principal

CORRIGE

PARTE 1

1. La capacité sert à filtrer les impulsions engendrées par le rebondissement du bouton.
2. MA : fermé → PC5 = 0V (0L).
3. MA : ouvert → PC5 = VDD = 3,3V (1L).
4. Configurations des lignes PC5, PC13, PC12

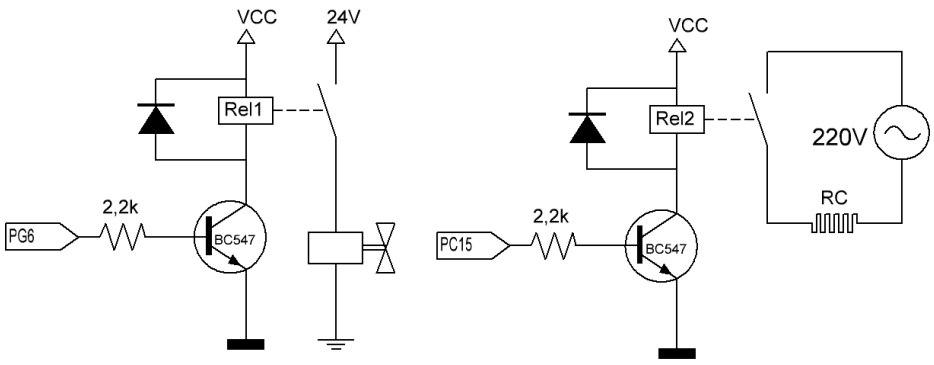
En utilisant les registres

```
GPIOC→MODER & = 0xF0FFF3FF ;
GPIOC→PUPDR & = 0xF0FFF3FF ;
```

En utilisant la librairie

```
GPIO_InitTypeDef  GPIO_InitStructure;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_13 |GPIO_Pin_12;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOC, &GPIO_InitStructure);
```

5. Sorties TOR
   a. Interface de puissance.



   b. Configuration des lignes PG6, PG8, PC11 et PC15.

En utilisant les registres

```
GPIOG→MODER &     = 0xFFFDDFFF ; GPIOG→MODER | = 0x00011000 ;
GPIOG→OTYPER &    = 0xFEBF ;
GPIOG→PUPDR &     = 0xFFFCCFFF ;
GPIOG→OSPEEDER &  = 0xFFFCCFFF ;
GPIOC→MODER &     = 0x7F7FFFFF ; GPIOC→MODER | = 0x40400000 ;
```

```
GPIOC→OTYPER &    = 0x77FF ;
GPIOC→PUPDR &     = 0x3F3FFFFF ;
GPIOC→OSPEEDER & = 0x3F3FFFFF;
```

En utilisant la librairie

```
GPIO_InitTypeDef  GPIO_InitStructure;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_8 ;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;;
GPIO_Init(GPIOG, &GPIO_InitStructure);
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11 | GPIO_Pin_15 ;
GPIO_Init(GPIOC, &GPIO_InitStructure);
```

6. Pseudocode :

En utilisant les registres

```
if((GPIOC→IDR & 0x1000) == 0)
{
    GPIOG→ODR & = 0xFFbF ;
    GPIOG→ODR | = 0x0100 ;
    // Ou bien en utilisant le registre BSRR : GPIOG→BSRR = 0x00400100 ;
}
```

En utilisant la librairie

```
if(GPIO_ReadInputDataBit (GPIOC, GPIO_Pin_12) == 0)
{
    GPIO_WriteBit(GPIOG, GPIO_Pin_6, Bit_RESET) ;
    GPIO_WriteBit(GPIOG, GPIO_Pin_8, Bit_SET) ;
}
```

PARTE 2

1. $V_{REF+} = V_{DDA} = 3,3V$
   a. Le capteur LM35 délivre une tension de 1,5V pour une température de 150°C, soit de 10mV/°C.

      D'où 0,5°C correspond à 5mV qui correspond à son tour à 1 LSB (q = 5mV).

      $2^N = \frac{V_{REF+}}{q} = \frac{3,3V}{5mV} = 660$; 256 < 660 < 1024 ; donc *N* = 10 bits.

   b. $q = \frac{3,3V}{1024} = 3,22mV$ cette valeur correspond à 0,322°C < 0,5°C.

   c. $q = \frac{3,3V}{4096} = 0,8mV$ cette valeur correspond à 0,08°C < 0,1°C.

2. La résolution du convertisseur :

   $q = \frac{1,5V}{1024} = 1,46mV$ cette valeur correspond à 0,146°C < 0,5°C.

   L'erreur de quantification = $\frac{1}{2}LSB = \frac{1,46mV}{2} = 0,73mV$, soit une erreur de 0,073°C.

3. Temps d'acquisition :

   $T_s = 2 \cdot 10^3 \times 15 \cdot 10^6 \times 4 \cdot 10^{-12} \times 7,62 = 0,914 cycles$ soit $T_s = 3 cycles$

4. Temps de conversion total : $t_{convtotal} = (T_s + T_{conv}) \cdot T_{ADC} = (3+10)\frac{10^{-6}}{15} = 0,86\mu s$.

5. Configuration de la broche PF9 en analogique :

   En utilisant les registres

```
GPIOF→MODER | = 0x000C0000 ;
GPIOC→PUPDR & = 0xFFF3FFFF ;
```

   En utilisant la librairie

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 ;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOF, &GPIO_InitStructure);
```

6. Procédure init_ADC3()

   En utilisant les registres

```
void init_ADC3(void)
{
    ADC3→CCR &    =     0xFFFDFFE0 ;
    ADC3→CCR |    =     0x00010000 ;
    ADC3→SMPR2 &  =     0xFF1FFFFF ;
    ADC3→SQR1 &   =     0xFF0FFFFF ;
    ADC3→SQR3 &   =     0xFFFFFFE7 ;
    ADC3→SQR3 |   =     0x00000007 ;
    ADC3→CR1 &    =     0xFDFFFEFF ;
    ADC3→CR1 |    =     0x01000000 ;
    ADC3→CR2 &    =     0x8FFFF7FD ;
    ADC3→CR2 |    =     0x00000401;
}
```

   En utilisant la librairie

```
ADC_InitTypeDef       ADC_InitStructure;
ADC_CommonInitTypeDef ADC_CommonInitStructure;
void int_ADC3(void)
{
  ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
  ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div4;
  ADC_CommonInit(&ADC_CommonInitStructure);
  ADC_InitStructure.ADC_Resolution = ADC_Resolution_10b;
  ADC_InitStructure.ADC_ScanConvMode = DISABLE;
  ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
  ADC_InitStructure.ADC_ExternalTrigConvEdge =
                                          ADC_ExternalTrigConvEdge_None;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfConversion = 1;
  ADC_Init(ADC3, &ADC_InitStructure);
```

```
    ADC_RegularChannelConfig(ADC3, ADC_Channel_7,1,ADC_SampleTime_15Cycles);
    ADC_EOCOnEachRegularChannelCmd ( ADC3, ENABLE);
    ADC_Cmd(ADC3, ENABLE);
}
```

7. Pseudocode d'acquisition de la température.

En utilisant les registres

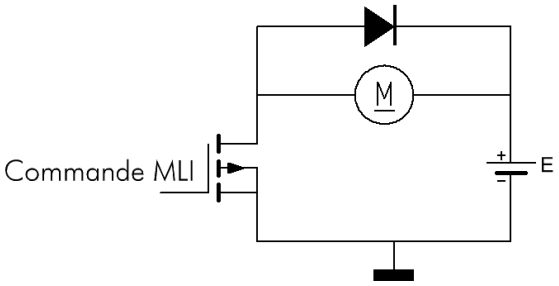
```
ADC3→CR2 | = 0x40000000 ;     // start
while((ADC3→ SR & 0x00000002)==0) ;
    TempVal = ADC3→DR ;
```

En utilisant la librairie

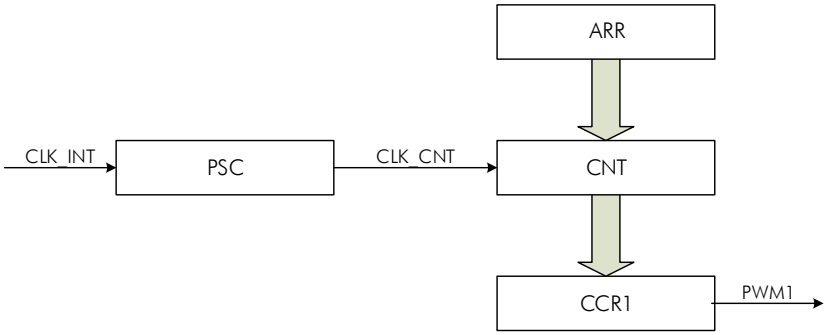
```
ADC_SoftwareStartConv(ADC3);
while(!ADC_GetFlagStatus ( ADC3, ADC_FLAG_EOC));

TempVal = ADC_GetConversionValue(ADC3);
```

PARTE 3

1. schéma du hacheur :



2. Si on désire une résolution de 0,1% de α ; il faut alors disposer de $\frac{1}{0,1\%}=1000$ valeurs. Le compteur CNT doit compter 1000 impulsions (modulo 1000).



La période du signal PWM est la période de débordement du compteur CNT. D'où $CLK\_CNT = 1000 \times 10^4 = 10Mhz$.

CLK_INT = CLKSYS = 120Mhz, alors $PSC = \frac{CLK\_INT}{CLK\_CNT} - 1 = \frac{120}{10} - 1 = 11$

On doit charger le registre PSC par 11 et le registre ARR par 1000 – 1 =999.

Etapes à suivre :

- Configurer la proche PF6 en « Alternate function » , PushPull, sans PU,PD, « out speed » maximal puis la faire associer PF6 au timer10.
- Initialiser les registres PSC et ARR et CCR1 (rapport cycle).

- Configurer CC1 en mode sortie (CC1S)
- Choix de mode PWM1 (CC1M)
- Auto-chargement des registres PSC, ARR, CCR1 sur des événements. (ARPE et OC1PE)
- Validation du timer (CEN)

En utilisant les registres

```
void init_TIM10(void)
{
  GPIOF→MODER &    = 0xFFFFEFFF ;
  GPIOF→MODER |    = 0x00002000 ;
  GPIOF→OTYPER &   = 0xFFBF ;
  GPIOF→OSPEEDER|  = 0x00003000 ;
  GPIOF→PUPDR &    = 0xFFFFCFFF ;
  GPIOF→ AFR[0] &  = 0xF3FFFFFF ;
  GPIOF→ AFR[0] |  = 0x03000000 ;
  TIM10→PSC        = 11 ;
  TIM10→ARR        = 999 ;
  TIM10→CCR        = 0 ;
  TIM10→CCMR1      = 0x006C ;
  TIM10→CR1        = 0x0081 ;
}
```

En utilisant la librairie

```
GPIO_InitTypeDef  GPIO_InitStructure;
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
TIM_OCInitTypeDef TIM_OCInitStructure;
void init_TIM10(void)
{
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 ;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_Init(GPIOF, &GPIO_InitStructure);
  GPIO_PinAFConfig(GPIOF, GPIO_PinSource6, GPIO_AF_TIM10);

  TIM_TimeBaseStructure.TIM_Period = 999;
  TIM_TimeBaseStructure.TIM_Prescaler = 11;
  TIM_TimeBaseInit(TIM10, &TIM_TimeBaseStructure);

  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
  TIM_OCInitStructure.TIM_Pulse = 0;
  TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
  TIM_OC1Init(TIM10, &TIM_OCInitStructure);
  TIM_OC1PreloadConfig(TIM10, TIM_OCPreload_Enable);
  TIM_ARRPreloadConfig(TIM10, ENABLE);
  TIM_Cmd(TIM10, ENABLE);
}
```