TD4

Exercice N° 1

L'horloge temps réel (RTC) DS1307 comporte une horloge / calendrier à codage (BCD) et une zone mémoire Non-Volatile SRAM de 56 octets. L'adresse et les données sont transmises en série via une interface I2C.

Les informations de l'heure et de calendrier sont obtenues en lisant les octets des registres appropriés. L'heure et le calendrier sont définis ou initialisés au format BCD. Le bit 7 du registre 0 est le bit d'arrêt d'horloge (CH). Lorsque ce bit est mis à 1, l'oscillateur est désactivé. Lorsqu'il est remis à 0, l'oscillateur est activé. Veuillez activer l'oscillateur (bit CH = 0) lors de la configuration initiale.

Adresses	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Fonction	Plage
0x00	CH	10 Secondes			Secondes				Secondes	00 - 59
0x01	0	10 Minutes			Minutes				Minutes	00 – 59
0x02	0	12 / 24	10 Hrs / PM/AM	10 Hrs	Heures				Heures	1 - 12 +AM/PM 00 – 23
0x03	0	0	0	0	0	Jour			Jour	1 – 7
0x04	0	0	10 Date		Date				Date	1 - 31
0x05		0	0	10 mois	Mois				Mois	1 – 12
0x06	10 Année				Année				Année	00 - 99
0x07	OUT	0	0	SQWE	0	0	RS1	RS0	Contrôle	-
0x08 – 0x3F									SRAM 56 cases	0x00 – 0xFF

Le DS1307 peut fonctionner en mode 12 heures ou 24 heures. Le bit 6 du registre des heures est défini comme le bit de sélection du mode 12 ou 24 heures. Lorsqu'il est à l'état haut, le mode 12 heures est sélectionné. En mode 12 heures, le bit 5 est le bit AM / PM avec le niveau logique haut pour PM. En mode 24 heures, le bit 5 est le deuxième bit pour les heures.

Le DS1307 fonctionne comme un circuit esclave avec une adresse sur 7 bits (1101000). La figure suivante illustre les différents formats des trames de lecture et écriture :

Trame d'écriture – Esclave en mode réception

S	1101000	0	A	Adresse	A	Donnée1	A	•••	Donnéen	A	P

Trame de lecture – Esclave en mode transmission

S	110 1000	1	A	Donnée1	A	•••	Donnéen	$\bar{A}$	P

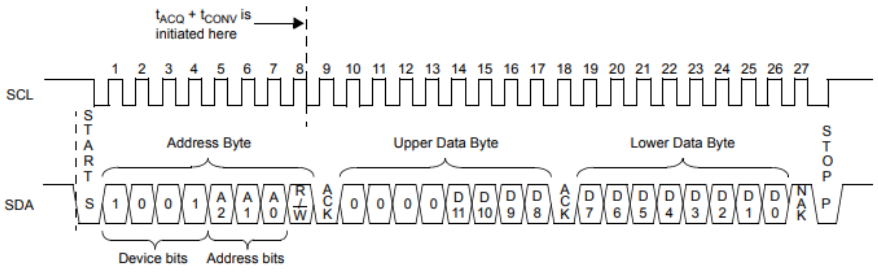
Trame de lecture (écriture du pointeur , puis lecture) - Esclave en réception et transmission

S	1101000	0	A	Adresse	A	S	1101000	1	A	Donnée1	A	•••	Donnéen	$\bar{A}$	P

Ecrire un programme qui permet de lire l'heure et la date toutes les secondes.

Exercice N° 2

Le circuit MCP3221 est un convertisseur analogique numérique de 12 bits à sorte série compatible I2C. La trame de lecture est la suivante :



Ecrire un programme qui lit t la valeur convertie toutes les séconde.

Exercice N° 3

Le TC74 est un capteur de température à sortie série compatible I2C. La plage de mesure est située entre −55°C et +125°C . La trame de lecture de la température est donnée par la figure suivante :

Read Byte Format

S	Address	WR	ACK	Command	ACK	S	Address	RD	ACK	Data	NACK	P
	7 Bits			8 Bits			7 Bits			8 Bits		

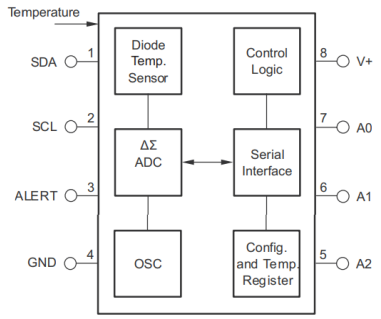
Slave Address — Command Byte: selects which register you are reading from. — Slave Address: repeated due to change in data-flow direction. — Data Byte: reads from the register set by the command byte.

- Adresse de l'esclave égale à 0x90
- Le mot de commande de la lecture de température égale à 0x00.

Ecrire un programme qui permet de lire dans la variable TempVal la température toutes les secondes.

Exercice N° 4

Le circuit TMP75 est un thermomètre numérique qui peut mesurer des températures de -40 à 125°C. Ce circuit est équipé d'un ADC 12 bits, offrant ainsi une résolution de 0,0625°C. Le TMP75 comporte une interface de sortie à deux fils compatible I2C.



SDA : donnée série
SCL : horloge
A0, A1, A2 : Adresse de sélection
ALERT : signal d'alerte lorsque la température atteint une certaine limite.
GND, V+ : alimentation de 2,7 à 5,5 V

Adresse du TMP75

1 0 0 1 A2 A1 A0

Le TMP75 comporte 5 registres : registre pointeur, registre de configuration, registres de température et deux registres pour fixer les seuils de température T_{LOW} et T_{HIGH}.

Registre pointeur

7.5.1.1 Pointer Register Byte (pointer = N/A) [reset = 00h]

Table 7-4. Pointer Register Byte

P7	P6	P5	P4	P3	P2	P1	P0
0	0	0	0	0	0	Register Bits	

7.5.1.2 Pointer Addresses of the TMP175

Table 7-5. Pointer Addresses of the TMP175 and TMP75

P1	P0	TYPE	REGISTER
0	0	R only, default	Temperature register
0	1	R/W	Configuration register
1	0	R/W	T_{LOW} register
1	1	R/W	T_{HIGH} register

Registre de configuration

D7	D6	D5	D4	D3	D2	D1	D0
OS	R1	R0	F1	F0	POL	TM	SD

OS : Le TMP75 dispose d'un mode de mesure de température en un seul coup. Lorsque le circuit est en mode d'arrêt, l'écriture d'un 1 sur le bit OS démarre une seule conversion de température. Le circuit revient à l'état d'arrêt à la fin de la conversion.

R1 , R0 : Résolution

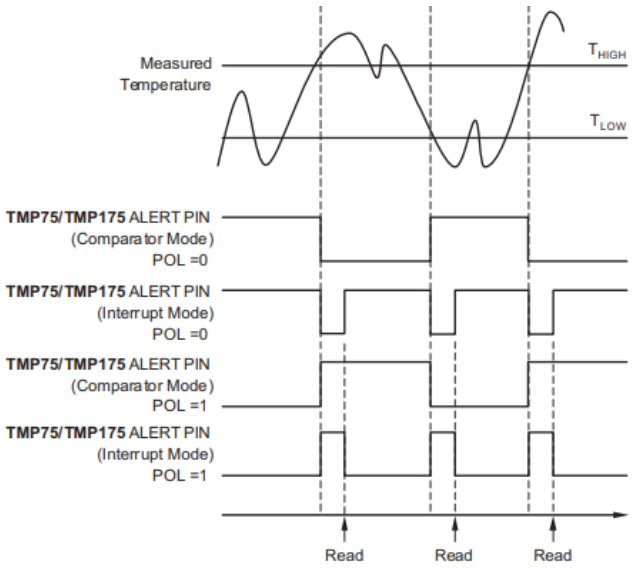
R1	R0	RESOLUTION	CONVERSION TIME (Typical)
0	0	9 bits (0.5 °C)	27.5 ms
0	1	10 bits (0.25 °C)	55 ms
1	0	11 bits (0.125 °C)	110 ms
1	1	12 bits (0.0625 °C)	220 ms

F1, F0 : Fixe le nombre de conditions de défaut nécessaires pour générer une alerte. Une condition de défaut est définie par le dépassement des valeurs limites définies par l'utilisateur dans les registres T_{HIGH} et T_{LOW}.

F1	F0	CONSECUTIVE FAULTS
0	0	1
0	1	2
1	0	4 (TMP175); 3 (TMP75)
1	1	6 (TMP175); 4 (TMP75)

POL : Le bit de polarité du TMP75 permet à l'utilisateur de régler la polarité de la sortie de la broche ALERT.

TM : Ce bit indique au circuit s'il doit fonctionner en mode comparateur (TM = 0) ou en mode interruption (TM = 1).



SD : Le mode d'arrêt (Shutdown Mode), il permet à l'utilisateur d'économiser la puissance en arrêtant tous les circuits autres que l'interface série, ce qui réduit la consommation du courant à moins de 0,1 µA. Le mode d'arrêt est activé lorsque le bit SD est à 1; le circuit s'arrête lorsque la conversion en cours est terminée. Lorsque SD est égal à 0, le circuit maintient un état de conversion continu.

Format de données

Le format de données pour le registre de température est le suivant :

Table 7-6. Byte 1 of the Temperature Register

D7	D6	D5	D4	D3	D2	D1	D0
T11	T10	T9	T8	T7	T6	T5	T4

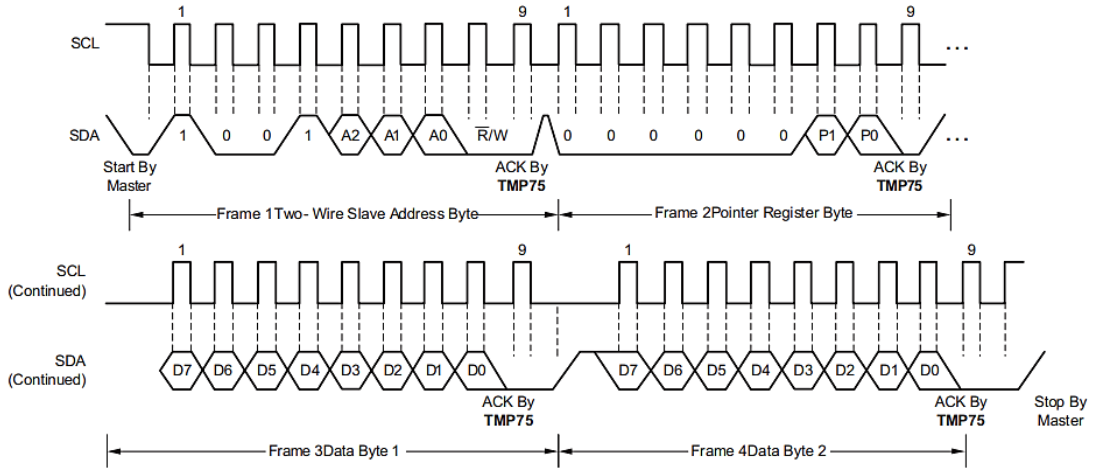
Table 7-7. Byte 2 of the Temperature Register

D7	D6	D5	D4	D3	D2	D1	D0
T3	T2	T1	T0	0	0	0	0

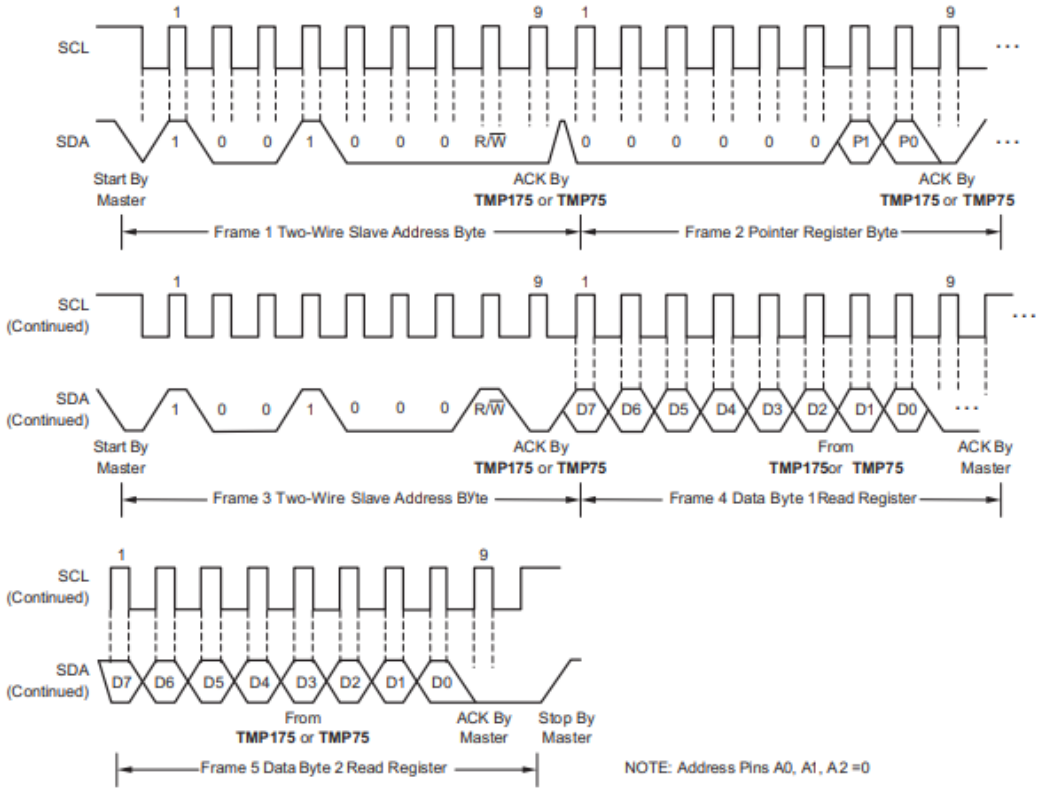
L'octet haut (byte1) représente la partie entière et l'octet bas la partie fractionnaire. Ce format est valable aussi pour les registres T_{LOW} et T_{HIGH}.

Trames de lecture/écriture

Trame de d'écriture



Trame de lecture



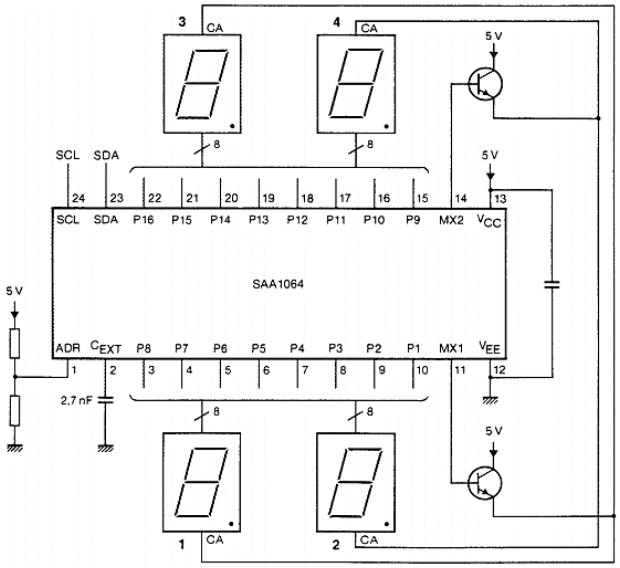
Questions

1. Ecrire le code de la fonction `configTMP75()`, qui permet de configurer le TMP75 avec les paramètres suivants :
   - Conversion continue,
   - Résolution : 12 bits
   - Nombre des conditions de défaut pour signaler une alerte : 1
   - Polarité positive (POL = 1) du signal ALERT en mode comparateur.
2. Ecrire le code de la fonction `limiteTEMP (int Tlow, int Thigh)` permettant d'initialiser les registres T_{LOW} et T_{HIGH}.
3. Ecrire le programme complet du thermomètre numérique. La température doit être lu et chargée dans la variable `tempVal` toutes les secondes.

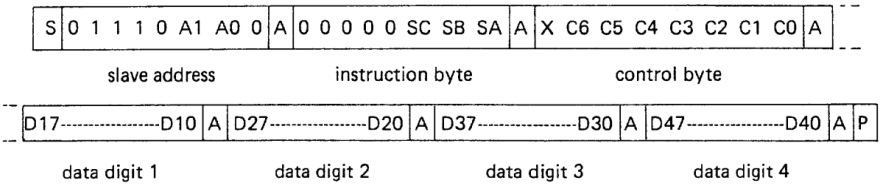
Exercice N° 5

Le SAA1064 est un driver spécialement conçu pour piloter quatre afficheurs LED à 7 segments anodes communes au moyen d'un multiplexage entre deux paires de chiffres. Il dispose d'une interface compatible I2C.

Un exemple d'utilisation du SAA1064 est donnée dans la figure suivante :



Trame d'écriture



Registre d'instruction (adressage des registres)

SC	SB	SA	SUB-ADDRESS	FUNCTION
0	0	0	00	control register
0	0	1	01	digit 1
0	1	0	02	digit 2
0	1	1	03	digit 3
1	0	0	04	digit 4
1	0	1	05	reserved, not used
1	1	0	06	reserved, not used
1	1	1	07	reserved, not used

Registre de contrôle

C0 = 0 : Mode statique, affichage en continu sur digits 1 et 2
C0 = 1 : Mode dynamique, affichage alterné pour digits 1+3 et 2+4
C1 = 0/1 : digits 1+3 sont masqués/non masqués
C2 = 0/1 : digits 2+4 sont masqués/non masqués
C3 = 1 : test des segments (tous les segments sont allumés)
C4 = 1 : ajout d'un coutant de sortie de 3mA par segment
C5 = 1 : ajout d'un coutant de sortie de 6mA par segment
C6 = 1 : ajout d'un coutant de sortie de 12mA par segment

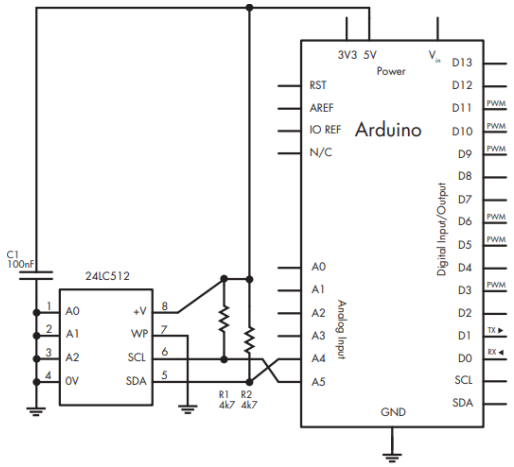
Question

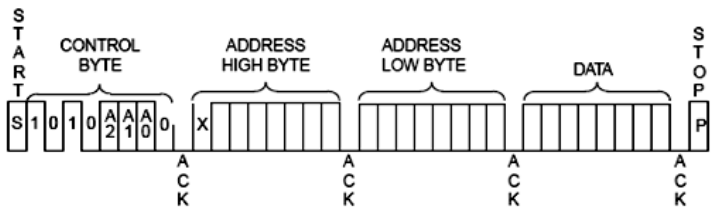
Etant donné le schéma de connexion ci-dessus. Ecrire un programme qui permet de réaliser un compteur modulo 10000. Le compteur s'incrémente à chaque demi-seconde.

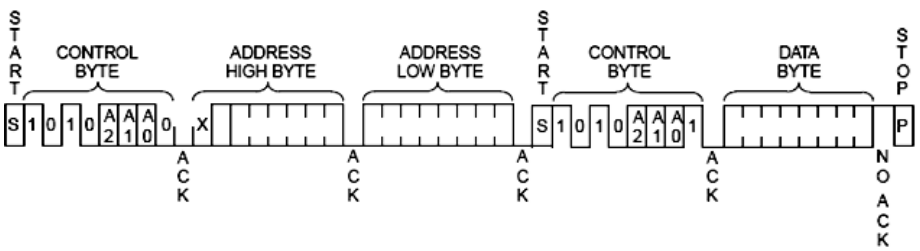
Exercice N° 6

Le EEPROM série sont très utilisées pour stocker des données en permanence. La mémoire 24C256 permet de stocker 32ko. Nous allons à titre d'exemple stocker et lire le 20 octets au début de l'espace mémoire.

1. Ecrire la procédure `void write_24c256(int Ad, byte val)` , permettant d'écrire la valeur `val` à l'adresse `Ad`.
2. Ecrire la procédure `byte read_24c256(int Ad)` permettant de renvoyer le contenu d'une case mémoire d'adresse `Ad`.
3. Ecrire le programme qui permet de stocker les valeurs de 1 à 20 dans les 20 premiers emplacements.







Corrigé

Exercice N° 1

Voir TP I2C

Exercice N° 2

```
I2C_HandleTypeDef hi2c1;
uint8_t RxData[2] ;
uint16_t ADC_val ;
#define Addr 0x90               // @circuit

      :
      :
void I2C_Init(void){
GPIO_InitTypeDef GPIO_InitStruct ;
/**I2C1 GPIO Configuration
    PB6     ------> I2C1_SCL
    PB7     ------> I2C1_SDA
    */
  GPIO_InitStruct.Pin = GPIO_PIN_6|GPIO_PIN_7;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF4_I2C1;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  HAL_I2C_Init(&hi2c1) ;
}

Void main(void){
I2C_Init() ;

 While(1){
   HAL_I2C_Master_Receive(&hi2c1, Addr, RxData, 2, HAL_MAX_DELAY);
   ADC_Val =(RxData[0]<<8) + RxData[1]; // résultat sur 16 bits
   HAL_Delay(1000) ;
 }
}
```

Exercice N° 3

```
I2C_HandleTypeDef hi2c1;
uint8_t Cmd ;
uint8_t TempVal ;
#define Addr 0x90               // @circuit

      :
      :
void I2C_Init(void){
GPIO_InitTypeDef GPIO_InitStruct ;
/**I2C1 GPIO Configuration
    PB6     ------> I2C1_SCL
    PB7     ------> I2C1_SDA
    */
  GPIO_InitStruct.Pin = GPIO_PIN_6|GPIO_PIN_7;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF4_I2C1;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
```

```
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  HAL_I2C_Init(&hi2c1) ;
}

Void main(void){
 I2C_Init() ;

 While(1){
   HAL_I2C_Master_Transmit(&hi2c1, Addr, &cmd, 1, HAL_MAX_DELAY);
   HAL_I2C_Master_Receive(&hi2c1, Addr, &TempVal, 2, HAL_MAX_DELAY);
   HAL_Delay(1000) ;
 }
}
```

Exercice N° 4

```
I2C_HandleTypeDef hi2c1;
uint8_t Tab[3] ;
float TempVal ;
#define Addr 0x90               // @circuit
#define TempMax 120             // limit high 120°C
#define TempMin 40)             // limit low  40°C

      :
      :
void I2C_Init(void){
GPIO_InitTypeDef GPIO_InitStruct ;
/**I2C1 GPIO Configuration
    PB6     ------> I2C1_SCL
    PB7     ------> I2C1_SDA
    */
  GPIO_InitStruct.Pin = GPIO_PIN_6|GPIO_PIN_7;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF4_I2C1;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  HAL_I2C_Init(&hi2c1) ;
}

void configTMP75(){
 Tab[0] = 0x01 ;
 Tab[1] = 0x64 ;
 HAL_I2C_Master_Transmit(&hi2c1, Addr, Tab, 2, HAL_MAX_DELAY);
}

void limiteTEMP (uint16_t Tlow, uint16_t Thigh){
 Tab[0] = 0x02 ;
 Tab[1] = TempMin ;
 Tab[2] = 0 ;
 HAL_I2C_Master_Transmit(&hi2c1, Addr, Tab, 3, HAL_MAX_DELAY);
 Tab[0] = 0x03 ;
 Tab[1] = TempMax ;
 Tab[2] = 0 ;
 HAL_I2C_Master_Transmit(&hi2c1, Addr, Tab, 3, HAL_MAX_DELAY);
}

float liretemp(){
 int16_t t ;
 float temp ;
 Tab[0] = 0x00 ;
```

```
 HAL_I2C_Master_Transmit(&hi2c1, Addr, Tab, 1, HAL_MAX_DELAY);
 HAL_I2C_Master_Receive(&hi2c1, Addr, Tab, 2, HAL_MAX_DELAY);
 t = ((int16_t)Tab[0] << 4)|(Tab[1] >> 4) ;
 temp = t * 0.0625;
 return(temp) ;
}


Void main(void){
 I2C_Init() ;
 configTMP75() ;
 limiteTEMP (TempMin,TempMax) ;
 While(1){

   TempVal = liretemp() ;
   HAL_Delay(1000) ;
 }
}
```

Exercice N° 5

```
I2C_HandleTypeDef hi2c1;
const uint8_t Addr = 0x38 << 1; /* I2C address for 7-Segment */
const int lookup[10] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};
int16_t count;
uint8_t Tab[5] ;
     :
     :
void I2C_Init(void){
GPIO_InitTypeDef GPIO_InitStruct ;
/**I2C1 GPIO Configuration
    PB6     ------> I2C1_SCL
    PB7     ------> I2C1_SDA
    */
  GPIO_InitStruct.Pin = GPIO_PIN_6|GPIO_PIN_7;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF4_I2C1;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  HAL_I2C_Init(&hi2c1) ;
}

void displayNumber( int16_t number)
{uint8_t i , digit;
  Tab[0] = 0x01 ;
  for(int i =1; i <= 4; i++)
  {
    digit= number % 10;
    Tab[i] = lookup[digit];
    number = number / 10;
  }

  HAL_I2C_Master_Transmit(&hi2c1, Addr, Tab, 5, HAL_MAX_DELAY);
}

void main(void)
{
 I2C_Init() ;
 Tab[0] = 0x00 ;
 Tab[1] = 0B01000111 ;
 HAL_I2C_Master_Transmit(&hi2c1, Addr, Tab, 2, HAL_MAX_DELAY);
 while(1){
   for (count = 0; count <= 9999; count++)
   {
```

```
      displayNumber(count);
      HAL_Delay(500);
    }
   }
}
```

Exercice N° 6

```
I2C_HandleTypeDef hi2c1;
const uint8_t Addr = 0x50 << 1; /* I2C address for 7-Segment */
int8_t count;
uint8_t Tab[5] ;
      :
      :
void I2C_Init(void){
GPIO_InitTypeDef GPIO_InitStruct ;
/**I2C1 GPIO Configuration
    PB6     ------> I2C1_SCL
    PB7     ------> I2C1_SDA
    */
  GPIO_InitStruct.Pin = GPIO_PIN_6|GPIO_PIN_7;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF4_I2C1;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  HAL_I2C_Init(&hi2c1) ;
}

void write_24C256(uint16_t address, uint8_t data)
{
  Tab[0] = address >> 8 ;
  Tab[1] = Address &0xFF ;
  Tab[2] = data ;
  HAL_I2C_Master_Transmit(&hi2c1, Addr, Tab, 3, HAL_MAX_DELAY);
  delay(10);
}

uint8_t read_24C256(uint16_t address)
{ uint8_t Data ;
  Tab[0] = address >> 8 ;
  Tab[1] = Address &0xFF ;
  HAL_I2C_Master_Transmit(&hi2c1, Addr, Tab, 2, HAL_MAX_DELAY);
  HAL_I2C_Master_Receive(&hi2c1, Addr, &Data, 1, HAL_MAX_DELAY);
  return (Data);
}
void main(void)
{
 /* Ecriture des données ... */
 for (count=0; count<20; count++)
 {
   write_24C256(count,count+1);
 }
}
```