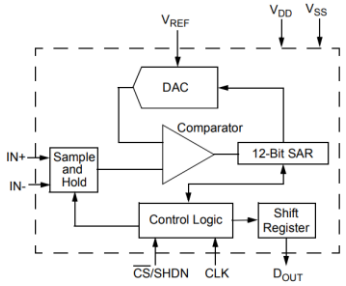
TD SPI

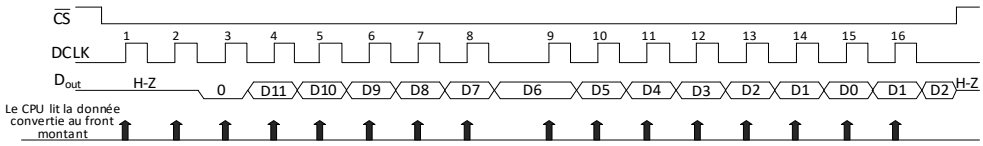
Exercice N°1

Le MCP3201 est une convertisseur analogique numérique série 12 bits. Sa sortie série est compatible avec le protocole SPI. Le circuit peut atteindre une vitesse de transmission 1,6MHz, sous une alimentation de 5V. Le microcontrôleur maitre du bus, fournit les signaux $\overline{CS}$ et SCK et l'ADC répond sur la broche Dout ; c'est un transfert dans un seul sens, la sortie SDO du microcontrôleur est non connectée.



Tant que la broche $\overline{CS}$ est à l'état haut, l'ADC est en mode veille et la sortie D_{out} est à l'état haute impédance (H-Z). Le passage de $\overline{CS}$ à l'état bas réveille l'ADC et lance la procédure de conversion à l'arrivée du premier front montant d'horloge. L'opération de conversion A/N prend 1,5 cycles ; au front descendant du 2ème cycle d'horloge, la sortie SDO passe à l'état bas. Après la transmission du bit nul, l'ADC procède au décalage de la donnée de conversion à chaque front descendant ; il appartient donc au maitre de lire les données sur le front montant.

Le format de données série pris en charge par le MCP3201 est illustré dans le chronogramme de la figure suivante :



Ecrire un programme qui permet de faire à chaque seconde une opération de conversion analogique numérique. La donnée convertie sera envoyée sur le port série.

Exercice N°2

Le TC72 est un capteur de température numérique, qui peut être connecté au bus SPI via les lignes SDI, SDO et SCK. La ligne CE, active au niveau haut, est utilisée particulièrement, lorsqu'on veut connecter plusieurs circuits sur le bus.

Le TC72 peut opérer en mode "One Shot" ou en mode continu. En mode One Shot, la mesure de la température s'effectue après une demande de lecture. Alors, qu'en mode continu, la mesure s'effectue approximativement toutes les 150ms.

Le circuit TC72 peut mesurer des températures de −55°C à +125°C. La valeur numérique est codée sur 10 bits en complément à 2 avec une précision de 0,25°C/bit. Cette valeur est représentée sur deux registres de 8 bits. Le registre MSB contient la partie entière et le registre LSB contient la partie fractionnaire. Uniquement les bits 7 et 6 sont utilisés dans ce dernier registre.

D7	D6	D5	D4	D3	D2	D1	D0	Register
Sign	2^6	2^5	2^4	2^3	2^3	2^1	2^0	Temp. MSB
2^{-1}	2^{-2}	0	0	0	0	0	0	Temp. LSB

Opération de lecture/écriture

Le TC72 peut opérer sur le front montant ou descendant du signal d'horloge SCK. La polarité de l'horloge est déterminée lors du passage de la ligne CE à l'état haut. Nous nous plaçons dans cet exercice dans le cas où CPOL = 0 et CPHA = 1. La fréquence maximale de transfert est de 7,5MHz.

Le transfert de donnée consiste à envoyer en premier lieu un octet d'adresse suivie d'un ou de plusieurs octets de données. Le bit MSB d'adresse (A7) détermine si une opération de lecture ou écriture aura lieu. Si A7 = 1, un ou plusieurs cycles d'écriture sont requis ; si A7 = 0, une opération de lecture est requise. Le TC72 dispose de quatre registres, un registre de contrôle deux registres de données et un registre ID (un code pour l'identification du circuit).

Register	Read Address	Write Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit3	Bit2	Bit1	Bit0	Value on POR/BOR
Control	00hex	80hex	0	0	0	One-Shot	0	1	0	Shutdown	05hex
LSB Temperature	01hex	-	T1	T0	0	0	0	0	0	0	00hex
MSB Temperature	02hex	-	T9	T8	T7	T6	T5	T4	T3	T2	00hex
Manufacturer ID	03hex	-	0	1	0	1	0	1	0	0	54hex

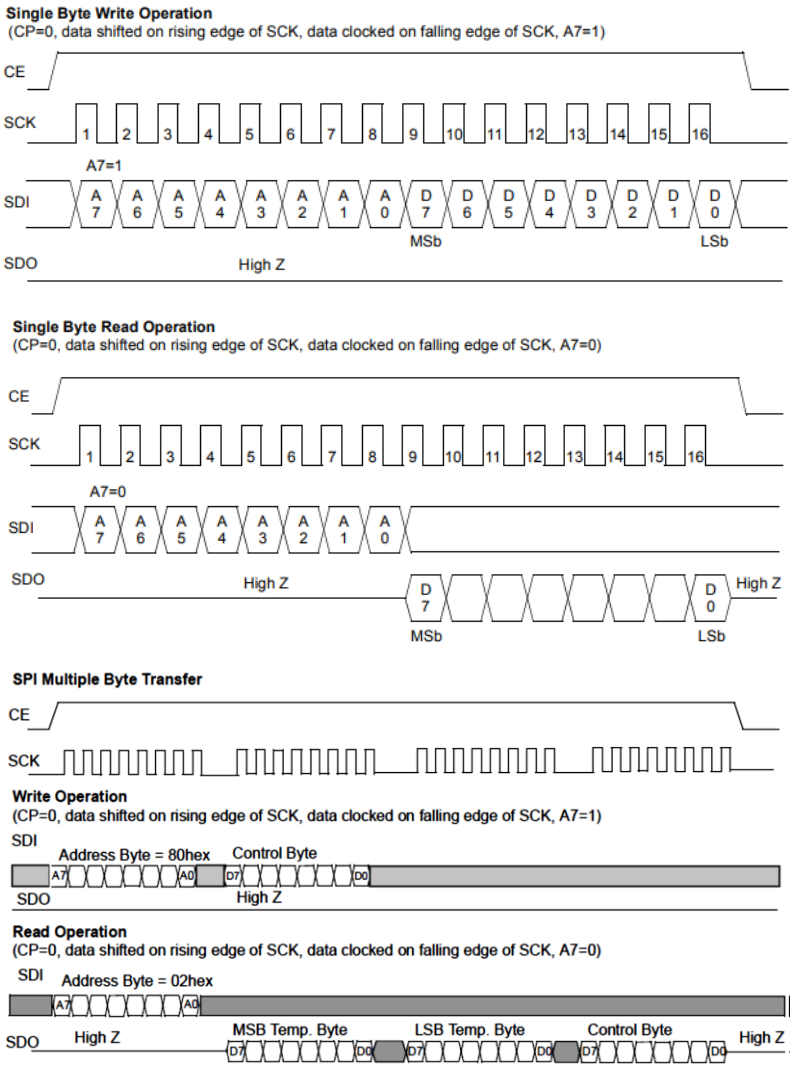
Le registre de contrôle permet à travers les bits **One Shot** et **Shutdown** de sélectionner le mode Shutdown (mode économique), conversion continue ou à la demande (One Shot).

Operational Mode	One-Shot Bit 4	Shutdown Bit 0
Continuous Temperature Conversion	0	0
Shutdown	0	1
Continuous Temperature Conversion (One-Shot Command is ignored if Shutdown = '0')	1	0
One-Shot	1	1

A chaque fois que la combinaison One-Shot =1 et Shutdown = 1 est sélectionné (mode One Shot), une seule mesure est effectuée. Après la fin de conversion le bit **One-Shot** revient à zéro (le circuit revient au mode Shutdown).

Que ce soit en mode One Shot ou continu, la donnée de mesure devient disponible après 150ms environ.

La figure suivante illustre les chronogrammes des opérations d'écriture et de lecture.



La lecture de la température nécessite les étapes suivantes :

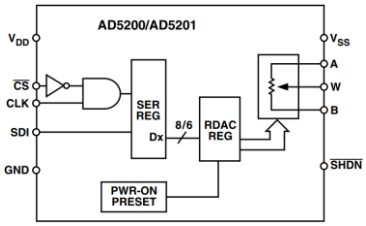
- Activez TC72 (CE = 1),

- Envoyez l'adresse 0x80 (A7 = 1), suivi de la commande One-Shot (contrôle = 0001 0101),
- Désactiver TC72 (CE = 0),
- Attendez au moins 150 ms pour que la température soit disponible,
- Activez TC72 (CE = 1, pour le transfert de données multiples),
- Envoyer la commande de lecture (adresse de lecture = 0x02),
- Lire la température deux octets consécutifs MSB puis LSB,
- Désactivez le transfert de données TC72 (CE = 0),

Ecrire un programme qui permet de faire l'acquisition de la température toutes les secondes.

Exercice N° 3

L'AD5200 est une résistance programmable, avec 256 positions, qui peut être contrôlée numériquement via une interface série SPI à 3 fils. L'AD5200 offre une valeur de résistance entièrement programmable, entre la borne A et le curseur, ou la borne B et le curseur. La résistance terminale entre A à B est de 10 kΩ ou 50 kΩ.

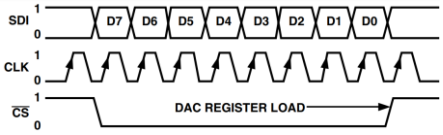


La formule suivante donne le lien entre la valeur de la résistance entre les broches A et W, et la valeur numérique (D).

$$R_{AW}(D) = \frac{255 - D}{255} R_{AB} + 50\Omega$$

Le potentiomètre est préréglé en interne lors de la mise sous tension ; ce préréglage interne force le curseur en position moyenne en chargeant la valeur 80H dans le verrou de l'AD5200. De plus, l'AD5200 contient une broche $\overline{SHDN}$ de coupure d'alimentation qui placent le RDAC dans un état de consommation d'énergie nulle où les commutateurs immédiats à côté des bornes A et B sont en circuit ouvert. Pendant ce temps, le curseur W est connecté à la borne B, ce qui entraîne uniquement une consommation de courant de fuite. L'interface numérique est toujours active pendant l'arrêt afin que des modifications de code puissent être apportées.

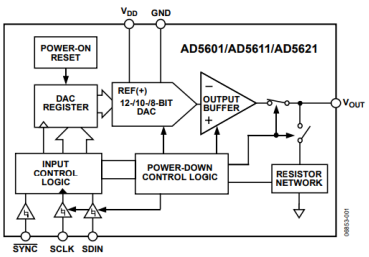
Huit bits de données constituent le mot de données qui est cadencé dans le registre d'entrée série.



Pour un potentiomètre de 10kΩ ; écrire un programme qui permet de fixe la résistance entre les broches A et W à 7,5kΩ.

Exercice N° 4

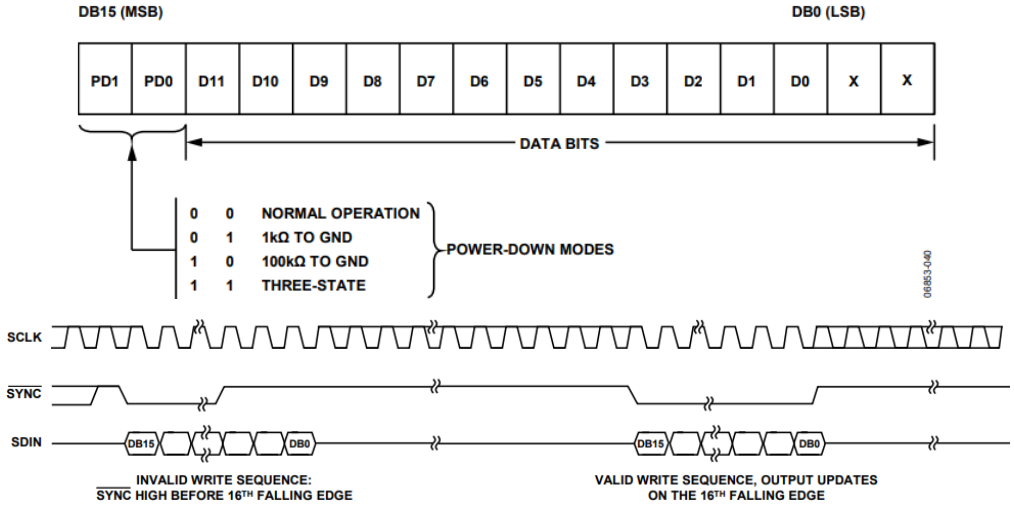
L'AD5621 est un convertisseur numérique-analogique de 12 bits à sortie tension. Ce circuit utilise une interface série à trois fils compatible SPI.



La tension de sorte est régie par l'équation suivante :

$$V_{out} = V_{DD} \times \frac{(D)}{2^{12}}$$

L'AD5621 contient une fonction de mise hors tension pour réduire sa consommation. Il fournit également des charges de sortie sélectionnables pour la mise hors tension.



Ecrire un programme qui permet de générer à la sortie du DAC une tension de 2,5V. la tension d'alimentation V_{DD} = 3,3V

Exercice N° 5

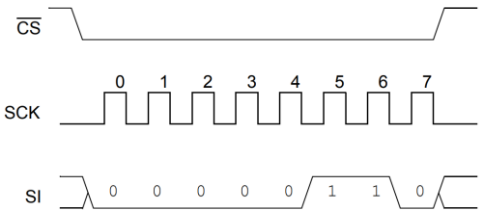
Les 25AA256/25LC256 sont des PROMs série de 256 Kbits effaçables électriquement. La mémoire est accessible via un simple bus série compatible SPI. L'accès à l'appareil est contrôlé par une entrée Chip Select ($\overline{CS}$). La communication avec le circuit peut être interrompue via la broche de maintien ($\overline{HOLD}$).

Le 25XX256 contient un registre d'instructions de 8 bits. L'accès au circuit s'effectue via la broche SI, les données étant synchronisées sur le front montant du SCK. La broche $\overline{CS}$ doit être à l'état bas et la broche $\overline{HOLD}$ doit être à l'état haut pour toute l'opération. Le Tableau suivant contient la liste des octets d'instructions possibles et le format pour le fonctionnement du circuit.

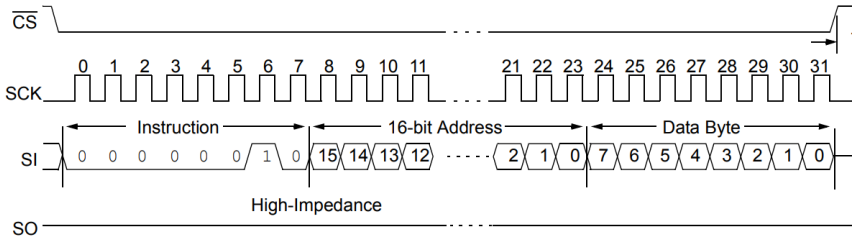
Instruction Name	Instruction Format	Description
READ	0000 0011	Read data from memory array beginning at selected address
WRITE	0000 0010	Write data to memory array beginning at selected address
WRDI	0000 0100	Reset the write enable latch (disable write operations)
WREN	0000 0110	Set the write enable latch (enable write operations)
RDSR	0000 0101	Read STATUS register
WRSR	0000 0001	Write STATUS register

Opération d'écriture

Avant toute tentative d'écriture de données sur le 25XX256, le verrou de validation d'écriture doit être défini en émettant l'instruction WREN. Cela se fait en mettant $\overline{CS}$ à un niveau bas, puis en synchronisant les instructions appropriées dans le 25XX256. Une fois que les huit bits de l'instruction ont été transmis, le $\overline{CS}$ doit être mis à l'état haut pour la validation d'écriture. Une fois que le verrou de validation d'écriture est défini, l'utilisateur peut continuer en définissant le $\overline{CS}$ au niveau bas, en émettant une instruction WRITE, suivie de l'adresse 16 bits, puis des données à écrire. Jusqu'à 64 octets de données peuvent être envoyés en un cycle d'écriture. La seule restriction est que tous les octets doivent résider dans la même page. Le cycle d'écriture prend environ 5ms.



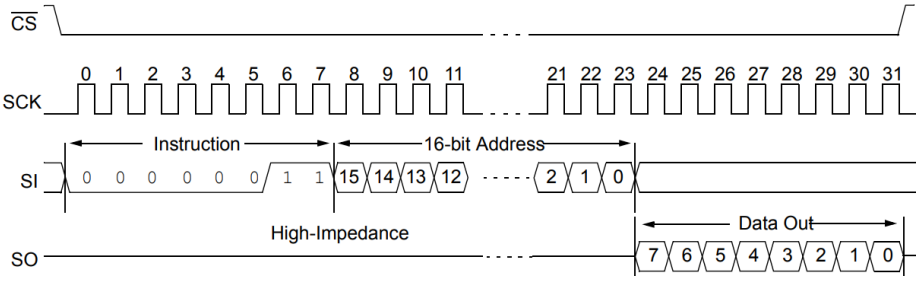
Séquence WREN



Séquence d'écriture

Opération de lecture

Le circuit est sélectionné en tirant $\overline{CS}$ vers le bas. L'instruction READ est transmise au 25XX256 suivi de l'adresse 16 bits. Une fois que l'instruction et l'adresse READ correctes ont été envoyées, les données stockées dans la mémoire à l'adresse sélectionnée sont décalées sur la broche SO. Les données stockées dans la mémoire à l'adresse suivante peuvent être lues séquentiellement en continuant à fournir des impulsions d'horloge. Lorsque l'adresse la plus élevée est atteinte (7FFFh), le compteur d'adresses retourne à l'adresse 0000h. L'opération de lecture se termine lorsque la broche $\overline{CS}$ passe à l'état haut.



Séquence de lecture

1. Ecrire une fonction qui permet d'écrire un octet dans l'EEPROM.
2. Ecrire une fonction qui renvoie la valeur stocker dans l'EEPROM à partir d'une adresse donnée.
3. Ecrire un programme qui permet de stocker à partir de l'adresse 0x0000, 20 valeurs consécutives, puis de les restituer.

Corrigé

On suppose que fréquence système = 100MHz et la fréquence du bus APB2 = 50MHz. PE3 pour la sélection du circuit.

Exercice N° 1

Voir corrigé TD 2

Exercice N° 2

Voir corrigé Examen 2020/21

Exercice N° 3

$R_{AW} = 7{,}5k\Omega \;\Rightarrow\; D = 65$

```
SPI_HandleTypeDef hspi1;
GPIO_InitTypeDef GPIO_InitStruct ;
Uint8_t data = 65 ;
:
:
void SPI_Init(void){

/**SPI1 GPIO Configuration
    PA5     ------> SPI1_SCK
    PA6     ------> SPI1_MISO
    PA7     ------> SPI1_MOSI
    PE3     ------> CS
    */
    GPIO_InitStruct.Pin = GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF5_SPI1;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pin : PE3 */
   GPIO_InitStruct.Pin = GPIO_PIN_3;
   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
   GPIO_InitStruct.Pull = GPIO_NOPULL;
   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
   HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

/* SPI1 parameter configuration*/
   hspi1.Instance = SPI1;
   hspi1.Init.Mode = SPI_MODE_MASTER;
   hspi1.Init.Direction = SPI_DIRECTION_2LINES;
   hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
   hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;   // Mode 0
   hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
   hspi1.Init.NSS = SPI_NSS_SOFT;
   hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_64;
   hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
   HAL_SPI_Init(&hspi1) ;
}
void main(void) {
  SPI.Init();

  HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);
  HAL_SPI_Transmit(&hspi1, &data, 1, 2000);
  HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET)}
  while(1){}
}
```

Exercice N° 4

$V_{out} = 2{,}5V\;;\quad V_{DD} = 3{,}3V \qquad \Rightarrow\; D = 3103$

```
SPI_HandleTypeDef hspi1;
GPIO_InitTypeDef GPIO_InitStruct ;
Uint16_t ADC_Value = 3103 ;
```

```
char data[3] ;
:
:
void SPI_Init(void){

/**SPI1 GPIO Configuration
    PA5     ------> SPI1_SCK
    PA6     ------> SPI1_MISO
    PA7     ------> SPI1_MOSI
    PE3     ------> SYNC
    */
    GPIO_InitStruct.Pin = GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF5_SPI1;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pin : PE3 */
   GPIO_InitStruct.Pin = GPIO_PIN_3;
   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
   GPIO_InitStruct.Pull = GPIO_NOPULL;
   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
   HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

/* SPI1 parameter configuration*/
   hspi1.Instance = SPI1;
   hspi1.Init.Mode = SPI_MODE_MASTER;
   hspi1.Init.Direction = SPI_DIRECTION_2LINES;
   hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
   hspi1.Init.CLKPolarity = SPI_POLARITY_HIGH;  // Mode 2
   hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
   hspi1.Init.NSS = SPI_NSS_SOFT;
   hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_64;
   hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
   HAL_SPI_Init(&hspi1) ;
}
void main(void) {
  SPI.Init();
  ADC_Value = ADC_Value << 2;   // ajuster la valeur avec Power Down Modes en Normal operation
  Data[0] = data >> 8;          // décomposer la valeur en deux octets
  Data[1] = data & 0xFF;
  HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);
  HAL_SPI_Transmit(&hspi1, (uint8_t *)data, 2, 2000);
  HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET)}
  while(1){}
}
```

Exercice N°5

```
SPI_HandleTypeDef hspi1;
GPIO_InitTypeDef GPIO_InitStruct ;
uint16_t Adr = 0 ;   // Adresse de la mémoire
uint8_t Ctrl ;       // Octet de contrôle
char Tab[20] ;
uint8_t i ;
     :
     :
void SPI_Init(void){

/**SPI1 GPIO Configuration
    PA5     ------> SPI1_SCK
    PA6     ------> SPI1_MISO
    PA7     ------> SPI1_MOSI
    PE3     ------> CS
    */
    GPIO_InitStruct.Pin = GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF5_SPI1;
```

```
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pin : PE3 */
   GPIO_InitStruct.Pin = GPIO_PIN_3;
   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
   GPIO_InitStruct.Pull = GPIO_NOPULL;
   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
   HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

/* SPI1 parameter configuration*/
   hspi1.Instance = SPI1;
   hspi1.Init.Mode = SPI_MODE_MASTER;
   hspi1.Init.Direction = SPI_DIRECTION_2LINES;
   hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
   hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;    // Mode 0
   hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
   hspi1.Init.NSS = SPI_NSS_SOFT;
   hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_64;
   hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
   HAL_SPI_Init(&hspi1) ;
}

void Write25AA256(uint16_t Adr, char Value){
   char MemAdr[2];
   MemAdr[0] = Adr >> 8;           // Décomposer l'adresse sur deux octets
   MemAdr[1]= Adr & 0xFF;
   Ctrl = 0x06 ;                   // WREN
   HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);
   HAL_SPI_Transmit(&hspi1, &Ctrl, 1, 2000);      // validation d'écriture dans la mémoire (WREN)
   HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);
   Ctrl = 0x02                     // WRITE
   HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);
   HAL_SPI_Transmit(&hspi1, &Ctrl, 1, 2000);                    // commande d'écriture
   HAL_SPI_Transmit(&hspi1, (uint8_t *)MenAdr, 2, 2000);        // envoi d'adresse
   HAL_SPI_Transmit(&hspi1, (uint8_t *)&value, 1, 2000);         // donnée à écrire
   HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);
   HAL_Delay(5);                                                // attendre la fin d'écriture
}

uint8_t Read25AA256(uint16_t Adr)
{  char MemAdr[2];
   uint8_t data ;
   MemAdr[0] = Adr >> 8;           // Décomposer l'adresse sur deux octets
   MemAdr[1]= Adr & 0xFF;
   Ctrl = 0x03                     // READ
   HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);
   HAL_SPI_Transmit(&hspi1, &Ctrl, 1, 2000);                    // commande de lecture
   HAL_SPI_Transmit(&hspi1, (uint8_t *)MenAdr, 2, 2000);        // envoi d'adresse
   HAL_SPI_Receive(&hspi1,  &data, 1, 2000);                    // lecture de la donnée
   HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);
   return(data);
}

void main(void) {
  SPI.Init();

  for(i = 0; i < 20; i++)
  {
    Write25AA256(Adr++, i++); // écriture dans la mémoire de 20 valeurs
  }
  Adr = 0;
 for(i = 0; i < 20; i++)
 {
    Tab[i] = Read25AA256(Adr++);  // lecture de 20 à partir de la mémoire
 }

While(1){}
}
```