I2C (INTERT-INTEGRATED CIRCUIT)

1. Le protocole I2C

Le protocole SPI soit relativement rapide, cependant, il nécessite au minimum trois lignes de données, plus une ligne de sélection pour chaque périphérique esclave connecté sur le bus. En augmentant l'intelligence de l'esclave, Philips Semiconductors (NXP Semiconductors aujourd'hui) a développé un simple bus série *bidirectionnel* à deux fils seulement, appelé *Inter-Integrated Circuit* (IIC connu plus avec l'abréviation I2C). Tous les circuits compatibles avec le bus I2C intègrent sur puce une interface qui leur permet de communiquer directement entre eux via le bus I2C. Ce concept de conception résout des nombreux problèmes d'interfaçage rencontrés lors de la conception de circuits numériques. Le protocole I2C est devenu une norme standard, implémenté sur plus de 1000 circuits intégrés ; y compris les EEPROM, les capteurs thermiques, les horloges en temps réel, les tuners RF, les décodeurs vidéo, etc.

Seules deux lignes sont nécessaires pour le fonctionnement du bus; une ligne de données série SDA (Serial DAta) et une ligne d'horloge série SCL (Serial CLock). La ligne SCL est aussi bidirectionnelle pour permettre le fonctionnement en mode multi-maîtres. La spécification originale a fixé le débit maximal à 100 kbps en mode standard, mais ce débit a augmenté au fil des années à 400kbps en mode rapide, à 1Mbps en mode rapide plus et à 3,4Mbps en mode haut débit.

L'I2C est un bus série synchrone half-duplex, où plusieurs équipements, maîtres ou esclaves, peuvent être connectés au bus. C'est toujours le maître qui initie le transfert de données et c'est à lui que relève la responsabilité de la génération de l'horloge. Il est possible qu'un équipement maître passe à l'état esclave ou réciproquement, l'essentiel, un seul maître tient le contrôle du bus à un moment donné. L'existence de plusieurs maitres sur le bus (mode multi-maitres) nécessite une procédure de gestion de conflit ; la spécification du protocole I2C, comporte la détection de collision et l'arbitrage pour éviter toute perte de données. Si plus qu'un maître tente à prendre le contrôle du bus en même temps, l'équipement qui perd le contrôle du bus se retire ou se reconvertit en esclave.

Chaque esclave est adressable par une adresse unique. Le protocole I2C spécifie le codage d'adresse sur 7 bits ou 10 bits. Jusqu'à 1024 adresses, ce nombre nous renseigne de la diversité des composants compatibles I2C et ne reflète plus le nombre des esclaves qui peuvent être connectés au bus. Le nombre maximal d'équipements est limité par la capacitance du bus, fixée par la norme standard à 400pf.

Examinons plus en détail les étages d'entrée/sortie adoptées dans les interfaces I2C (Figure 1). Tous les équipements connectés au bus, ayant comme étage de sortie un transistor drain ouvert et que la ligne à laquelle sont connectés (SDA ou SCL) est tirée vers le haut par une seule résistance externe (Pull up). Ce qui permet de réaliser un ET câblés ; l'état haut « 1 » de la ligne, impose le blocage de tous les transistors, pratiquement aucun courant ne circule dans la ligne. Cependant, il suffit qu'un seul transistor se sature pour que toute la ligne passe à l'état bas « 0 ». Lorsqu'un équipement souhaite mettre sa sortie à l'état haut, il n'a qu'à libérer la ligne (bloquer le transistor), mais cet état peut être facilement dominé par un état bas envoyé par un autre. Il est à remarquer aussi, que la sortie reboucle sur l'entrée ; cette structure est de grande utilité pour la détection de collision.

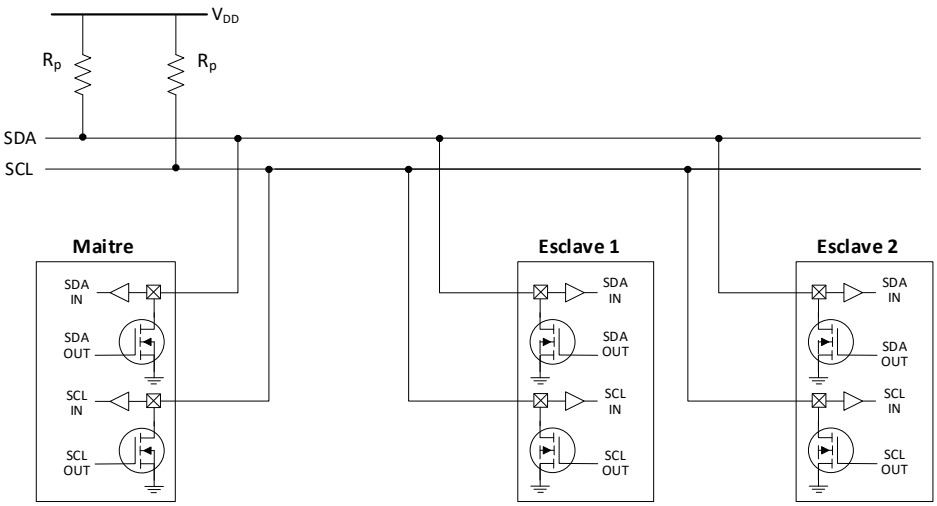

Figure 1

Lorsqu'il n'y a aucune activité sur le bus, les deux lignes doivent être à l'état haut (état de repos). Au cas où, un équipement maitre, souhaite prendre le contrôle du bus doit tirer la ligne SDA à l'état bas pendant que la ligne SCL est encore à l'état haut ; c'est ce qu'on appelle condition de démarrage (START condition) ; voir Figure 2. En plus de la génération de l'évènement de démarrage (bit de START), le maître est responsable de la génération du signal d'horloge et également de l'envoi de l'octet d'adresse ; le bit le plus faible de l'octet d'adresse indique, le sens de transfert du maître à l'esclave (maître-émetteur) ou de l'esclave vers le maître (maître-récepteur). Les données sont envoyées par octets ; les changements sur la ligne SDA ne doivent se produire que lorsque la ligne d'horloge est à l'état bas et le bit de donnée doit rester stable tant que le signal d'horloge est à l'état haut. Les données sont synchronisées dans le récepteur sur le front montant de SCL. Ces octets peuvent représenter des informations d'adresse ou de contrôle ou des données. Le protocole I2C comprend un mécanisme de contrôle de flux. Après le 8ème bit, l'équipement émetteur libère la ligne SDA pour que l'équipement récepteur puisse acquitter les données envoyées par l'émetteur. A la 9ème impulsion d'horloge, SDA est maintenu au niveau bas par le récepteur si la donnée a été acquise avec succès, donnant un état ACK. L'alternative, où le récepteur signale un problème ou qu'il n'est pas en mesure d'accepter plus de données, est appelée Not ACKnowledge ou NACK. Normalement, dans cette situation, le maître de bus met fin à l'opération de transfert. Un autre point fort du protocole I2C, est que l'esclave peut maintenir la ligne d'horloge à un niveau bas, lorsqu'il n'arrive pas à suivre le rythme du maître ; le maître est obligé d'attendre la libération de l'horloge pour continuer le transfert. Dans toutes les situations, il est de la responsabilité du maître de terminer la conversation par une transition positive sur la ligne SDA lorsque la ligne d'horloge SCL est à l'état haut ; signalant une condition d'arrêt (STOP condition).

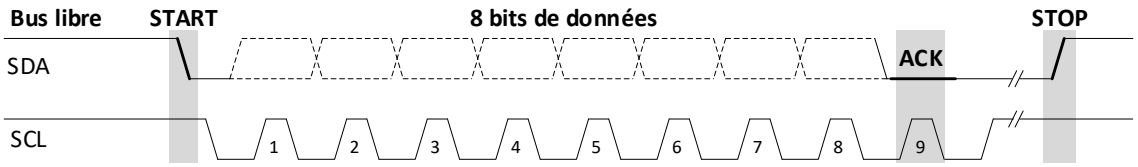

Figure 2

Toutes les transactions commencent par un bit de START (S) et doivent se terminer par un bit de STOP (P). Les conditions START et STOP sont générées par le maître ; le bus est considéré occupé après un bit de START (S) et devient à nouveau libre après un bit de STOP (P). Lorsqu'il est demandé à l'esclave de répondre à une requête, la réponse doit être livrée dans la même trame. Le maître peut générer une condition de START répétée (Sr) à l'intérieur de la trame, s'il veut changer le sens de transfert ; c'est pratiquement le cas de la réponse à une requête. A cet égard, les conditions de START (S) et START répétée

(Sr) sont fonctionnellement identiques et doivent être suivis toujours par un octet d'adresse ; raison pour laquelle, le sympole (S) est utilisé de manière générique pour représenter à la fois les conditions de START et START répétée.

En mode multi-maîtres, deux maîtres par exemple peuvent générer une condition de START valide sur le bus, pratiquement en même moment. Dans ce cas le processus d'arbitrage se déroule bit à bit. Lorsque la ligne SCL est à l'état haut, chaque maître doit vérifier si le niveau de la ligne SDA correspond bien à ce qu'il a envoyé. Une collision se produit lorsqu'un maître essaie d'envoyer un « 1 », mais détecte un « 0 » ; ce dernier abandonne l'échange en libérant le bus. L'autre maître poursuit sa transaction. Aucune information n'est perdue pendant le processus d'arbitrage. Le maître qui perd l'arbitrage doit redémarrer sa transaction ultérieurement.

Après la condition de démarrage START (S), un l'octet d'adresse d'un esclave est envoyé. Cette adresse est codée sur 7 bits, suivie d'un bit de direction des données ($R/\overline{W}$) : un « 0 » indique une transmission (Write) et un « 1 » indique une demande de données (Read). Toutes les adresses ne sont pas autorisées. Les adresses correspondant à 0b0000XXX ou 0b1111XXX sont réservées à des situations spécifiques ; par exemple, l'adresse 0b0000000 indique un appel général diffusé à tous les esclaves sur le bus.

Les formats des trames de lecture/écriture dépendent de l'équipement esclave, toutefois, elles respectent les structures suivantes :

- *Format d'écriture* (voir Figure 3) : Après la condition de START, le maître envoi l'octet d'adresse avec $R/\overline{W} = 0$, suivi des octets des données. Le récepteur esclave acquitte chaque octet reçu. Le dernier acquittement avant le bit de STOP, peut être un ACK ou un NACK ; si c'était un NACK, c'est que l'esclave a demandé l'arrêt du transfert.
- *Format de lecture* (voir figure 4) : Après la condition de START, le maître envoi l'octet d'adresse avec $R/\overline{W} = 1$; l'esclave acquitte l'octet d'adresse. Au moment de ce premier acquittement, le maître-émetteur devient un maître-récepteur et le récepteur esclave devient un esclave-émetteur. Le maître génère des acquittements ultérieurs. La condition STOP est générée par le maître, qui envoie un NACK juste avant la condition de STOP.
- *Format combiné* (voir Figure 5) : Après la condition de START, le maître envoi l'octet d'adresse avec $R/\overline{W} = 0$; le maître envoi ensuite le mot de commande (sur un ou plusieurs octets) ; une fois envoyé, il génère une condition da START répétée suivi de l'octet d'adresse, mais cette fois-ci avec $R/\overline{W} = 1$. A ce moment, le sens de transfert sera inversé, le maître-émetteur devient un maître-récepteur et le récepteur esclave devient un esclave-émetteur. Tous les acquittements avant l'inversion de sens sont générés par l'esclave ; les autres sont générés par le maître. Le dernier acquittement juste avant la condition STOP est un NACK.

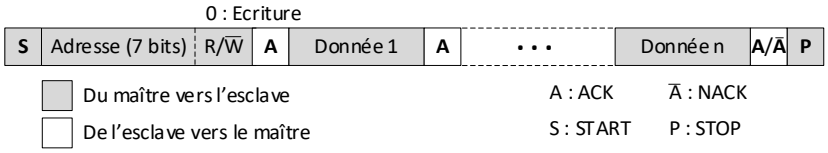

Figure 3: Trame d'écriture

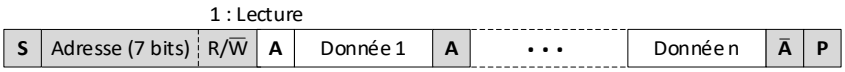

Figure 4 : Trame de lecture

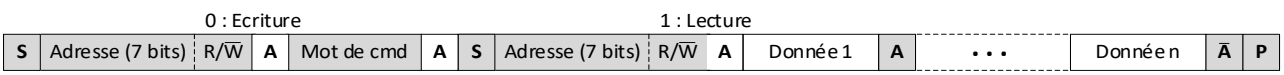

Figure 5 :Trame d'écriture / lecture

2. Gestion de l'I2C avec la librairie HAL

2.1. Initialisation du module I2C

Le librairie CubeHAL définie trois structures pour la gestion du module I2C. La structure *I2C_TypeDef* contient la définition des registres, la structure *I2C_InitTypeDef* définie les paramètres du module SPI et la structure *I2C_HandleTypeDef* contient les données nécessaires pour l'exploitation du module SPI :

```
typedef struct {
uint32_t    ClockSpeed;             /* Specifies the clock frequency */
                                        100000 ou 400000
uint32_t    DutyCycle;              /* Specifies the I²C fast mode duty cycle. */
                                        I2C_DUTYCYCLE_2
                                        I2C_DUTYCYCLE_16_9
uint32_t    OwnAddress1;            /* Specifies the first device own address. */
                                        Adresse sur 7 ou 10 bits
uint32_t    OwnAddress2;            /* Specifies the second device own address */
                                        Adresse sur 7 ou 10 bits
uint32_t    AddressingMode;         /* Specifies if 7-bit or 10-bit addressing mode */
                                        I2C_ADDRESSINGMODE_7BIT
                                        I2C_ADDRESSINGMODE_10BIT
uint32_t    DualAddressMode;        /* Specifies if dual addressing mode is selected. */
                                        I2C_DUALADDRESS_DISABLE
                                        I2C_DUALADDRESS_ENABLE
uint32_t    GeneralCallMode;        /* Specifies if general call mode is selected. */
                                        I2C_GENERALCALL_DISABLE
                                        I2C_GENERALCALL_ENABLE
uint32_t    NoStretchMode;          /* Specifies if nostretch mode is selected. */
                                        I2C_NOSTRETCH_DISABLE
                                        I2C_NOSTRETCH_ENABLE
} I2C_InitTypeDef;
```

La structure *I2C_HandleTypeDef* est définie comme suit :

```
typedef struct {
I2C_TypeDef          *Instance;          /* I²C registers base address */
I2C_InitTypeDef      Init;               /* I²C communication parameters */
uint8_t              *pBuffPtr;          /* Pointer to I²C transfer buffer */
uint16_t             XferSize;           /* I²C transfer size */
__IO uint16_t        XferCount;          /* I²C transfer counter */
  ...
} I2C_HandleTypeDef;
```

La fonction *HAL_StatusTypeDef HAL_I2C_Init(UART_HandleTypeDef *hi2c)* permet d'affecter les paramètres de la structure ***I2C_HandleTypeDef*** aux registres du périphérique SPI.

2.2. Echange de messages avec le module SPI

Une fois le périphérique I2C configuré, nous pouvons commencer à échanger des données avec des équipements esclaves. Le CubeHAL offre trois façons de communiquer sur un bus SPI : polling, interruption et le mode DMA. Nous présentons ici le mode polling (scrutation).

TRANSMISSION D'UN MESSAGE EN MODE MAITRE

HAL_StatusTypeDef HAL_I2C_Master_Transmit(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t Timeout);

RECEPTION D'UN MASSAGE EN MODE MAITRE

HAL_StatusTypeDef HAL_I2C_Master_Receive(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t Timeout);