



TP 3

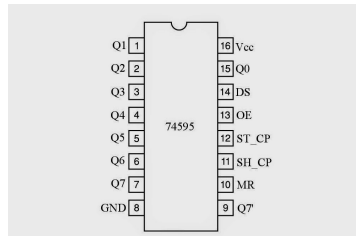
Architecture et plateformes IoT

Année universitaire : 2025-2026

Chiheb Ameer ABID

Pour économiser le nombre de broches GPIO utilisées sur un microcontrôleur ou une carte Raspberry Pi, on peut utiliser à un registre à décalage **série-parallèle**. Ce type de composant reçoit les données **bit par bit** en série et les restitue simultanément sur **plusieurs sorties** (parallèle).

Dans cet exercice, nous utiliserons le registre **74HC595**.



Principe de fonctionnement

Le registre lit les données *bit par bit* via son entrée **DS**, puis les transfère dans huit mémoires internes associées aux sorties **Q0 à Q7**.

Chaque mémoire peut contenir une valeur logique *0* ou *1*.

- À **chaque front montant** du signal **SH_CP**, tout le contenu du registre est décalé d'une position :
 - la mémoire de **Q7** reçoit la valeur précédemment dans **Q6**,
 - celle de **Q6** reçoit celle de **Q5**,
 - ...
 - celle de **Q0** reçoit le bit présent sur **DS**.

À ce stade, les sorties **Q0–Q7** ne changent pas encore : seules les mémoires internes sont modifiées.

- Les valeurs réellement visibles sur les broches **Q0 à Q7** ne sont mises à jour qu'au **front montant** de **ST_CP** : ce signal copie toutes les mémoires internes vers les sorties.

Entrées et sorties du 74HC595

- GND** : masse
- Vcc** : alimentation
- SH_CP** : horloge du registre à décalage
- ST_CP** : horloge du registre de stockage
- DS** : entrée série des données
- Q0–Q7** : sorties parallèles
- OE** : activation des sorties (active à l'état bas, connectée à GND)
- MR** : remise à zéro (active à l'état bas, connectée au +5 V ici)
- Q7'** : sortie série (utilisée pour chaîner plusieurs 74HC595)

On souhaite développer une classe C++ permettant de simplifier l'utilisation d'un registre 74HC595 depuis une Raspberry Pi.

```
class Chc595Exp {
private:
    uint8_t _sclk, _latch, _sda, _byte;
    /*
     * Prepare sending bits: _sclk & _latch must be LOW for at least 1us
     */
    void prepare();
    /*
     * Latch data to output
     */
    void update();
    /*
     * shift bits
     */
    void write8Bits();
public:
    /*
     * Specifies BCM pins
     */
    Chc595Exp(uint8_t sclk, uint8_t latch, uint8_t sda);
    /*
     * Write a byte to parallel output
     */
    void writeByte(uint8_t b);
};
```



```
virtual ~CHc595Exp();  
};
```

Rôle des attributs

- **_sclk**, **_latch**, **_sda** : numéros BCM des broches connectées respectivement à **SH_CP**, **ST_CP**, et **DS** du 74HC595.
- **_byte** : octet stockant la valeur à transmettre au registre.

Travail à réaliser

Écrire la définition de chacune des méthodes suivantes :

1. **CHc595Exp(uint8_t sclk, uint8_t latch, uint8_t sda)**

Le constructeur :

- enregistre les numéros BCM des broches,
- configure les broches en mode *sortie*,
- met **SH_CP**, **ST_CP** et **DS** à leur état de repos (LOW).

2. **void prepare()**

Prépare la transmission en mettant les broches **SH_CP** et **ST_CP** à 0 durant au moins **1 µs**.

3. **void write8Bits()**

Transmet les 8 bits contenus dans **_byte** vers les mémoires internes du registre :

- positionne **DS** selon chaque bit,
- génère un front montant sur **SH_CP** pour décaler.

4. **void update()**

Applique le contenu des mémoires internes sur les sorties :

- génère un front montant sur **ST_CP**.

5. **void writeByte(uint8_t b)**

Envoie la valeur **b** sur les sorties Q0–Q7 :

1. stocker **b** dans **_byte**;



2. appeler **prepare()** ;
3. appeler **write8Bits()** ;
4. appeler **update()**.