

**Réalisation d’une Serrure Electronique**

***I. Introduction***

Nous souhaitons réaliser une serrure électronique. La clef est fixée par un code à 4 chiffres que doit taper l’utilisateur. Si le code utilisateur est correct alors on déclenche l'ouverture de la serrure. Ce type de serrure est utilisé par exemple dans les hôtels pour de petits coffres forts permettant aux clients de laisser des objets précieux dans leur chambre. La serrure électronique est connectée à un clavier 16 touches. Un bus 16 bits ***BusExt*** relie le clavier à la serrure électronique. Chaque touche est connectée à la partie électronique de la serrure grâce à un fils du bus ***BusExt***. La connexion est montrée sur la figure 1.

1

2

3

A

4

5

6

B

7

8

9

C

F

0

E

D

16

BusExt

(de 0 à F)

Clavier

Serrure

Ouverture

Figure 1 : Interface clavier-serrure

Tant qu'aucune touche n'est enfoncée tous les fils du bus sont au niveau haut. L'appui sur une des touches met le fil qui la relie au codeur au niveau bas tandis que les autres fils restent au niveau haut.

La table suivante montre l’état de *BusExt* lors de l'appui sur les différentes touches du clavier. Vous remarquerez la position du bit au niveau bas en fonction de la position de la touche enfoncée sur le clavier.

|  |  |
| --- | --- |
| **Touche du clavier enfoncée** | **Etat du bus BusExt** |
| 0 | 1101 1111 1111 1111 |
| 1 | 1111 1111 1111 1110 |
| 2 | 1111 1111 1111 1101 |
| 3 | 1111 1111 1111 1011 |
| 4 | 1111 1111 1110 1111 |
| 5 | 1111 1111 1101 1111 |
| 6 | 1111 1111 1011 1111 |
| 7 | 1111 1110 1111 1111 |
| 8 | 1111 1101 1111 1111 |
| 9 | 1111 1011 1111 1111 |
| A | 1111 1111 1111 0111 |
| B | 1111 1111 0111 1111 |
| C | 1111 0111 1111 1111 |
| D | 0111 1111 1111 1111 |
| E | 1011 1111 1111 1111 |
| F | 1110 1111 1111 1111 |

Table 1- Table des états de BusExt en fonction de la touche enfoncée

***II. Partie 1 : Implémentation de la partie électronique d’une serrure en technologie FPGA***

L’architecture globale du système est présentée par la figure 2.

Détection Caractère

Codeur

Mémoire

Comparateur 16 bits

4

BusInt

16

BusExt

DetectCar

16

BusCode

Ouverture

Figure 2 : Architecture globale du système

Dans cette partie, vous allez créer chaque bloc constituant la serrure. Vous allez réaliser leur description structurelle en VHDL et validerez, à chaque fois, leur fonctionnement en effectuant une simulation.

1. ***Codeur***

Un codeur traduisant le code émis par le clavier (cf. table 1) en code binaire qui sera utilisé dans la serrure. Le bus ***BusExt*** provient du clavier et est en entrée du codeur. Le résultat du codage sur 4 bits sera mis sur le bus ***BusInt.*** Le codage retenu est indiqué dans la Table 2.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Etat du bus BusExt** | **BusInt** | **Detect** |
|  | 1111 1111 1111 1111 | Z | 0 |
| **0** | 1101 1111 1111 1111 | 0000 | 1 |
| **1** | 1111 1111 1111 1110 | 0001 | 1 |
| **2** | 1111 1111 1111 1101 | 0010 | 1 |
| **3** | 1111 1111 1111 1011 | 0011 | 1 |
| **4** | 1111 1111 1110 1111 | 0100 | 1 |
| **5** | 1111 1111 1101 1111 | 0101 | 1 |
| **6** | 1111 1111 1011 1111 | 0110 | 1 |
| **7** | 1111 1110 1111 1111 | 0111 | 1 |
| **8** | 1111 1101 1111 1111 | 1000 | 1 |
| **9** | 1111 1011 1111 1111 | 1001 | 1 |
| **A** | 1111 1111 1111 0111 | 1010 | 1 |
| **B** | 1111 1111 0111 1111 | 1011 | 1 |
| **C** | 1111 0111 1111 1111 | 1100 | 1 |
| **D** | 0111 1111 1111 1111 | 1101 | 1 |
| **E** | 1011 1111 1111 1111 | 1110 | 1 |
| **F** | 1110 1111 1111 1111 | 1111 | 1 |

Table 2 - Codeur

Codeur

16

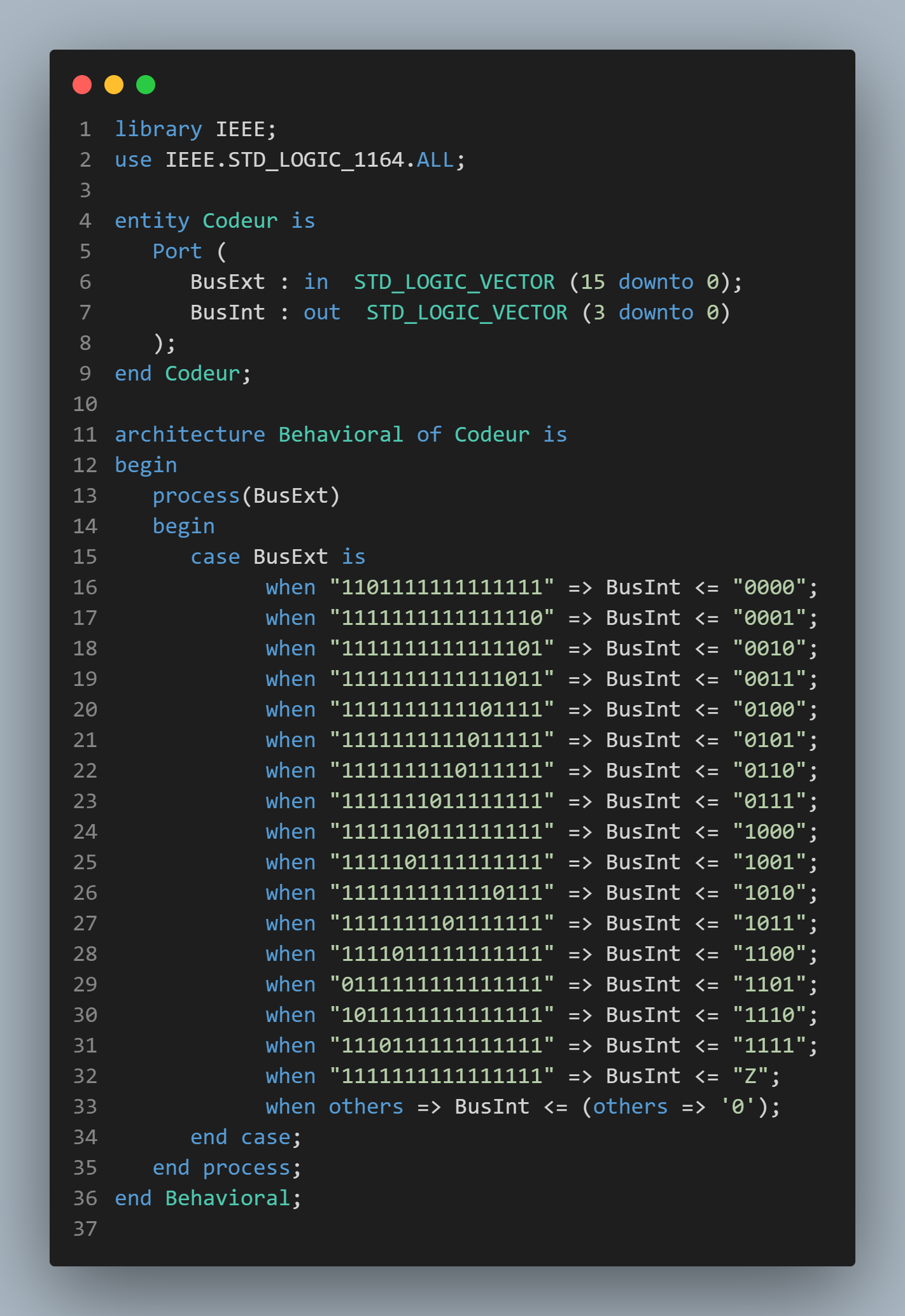
BusExt

(de 0 à F)

4

BusInt

Figure 3 - Codeur

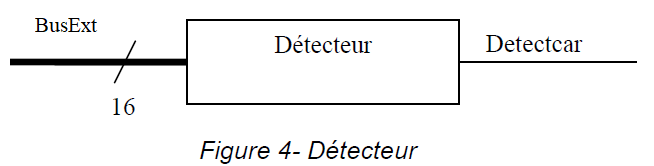
**Description VHDL du Codeur :**

Ce module possède un port d'entrée « BusExt » de type STD\_LOGIC\_VECTOR à 16 bits, et un port de sortie « BusInt » de type STD\_LOGIC\_VECTOR à 4 bits.

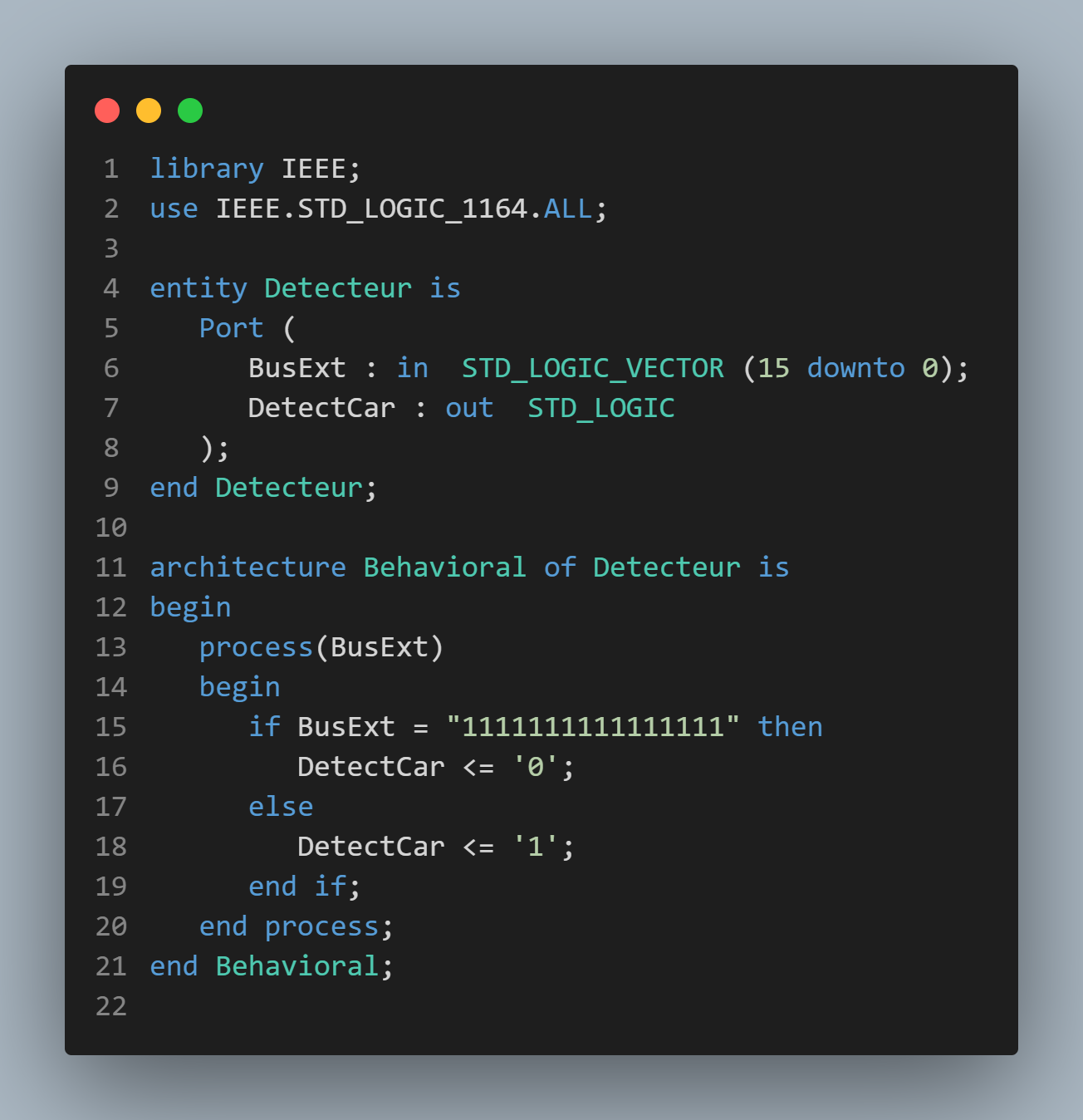
À l'intérieur de l'architecture comportementale, un processus est déclenché par le port d'entrée. Dans le processus, une instruction « case » est utilisée pour mapper l'entrée 16 bits à la sortie 4 bits correspondante selon les règles de conversion données. Si l'entrée ne correspond à aucun des cas spécifiés la sortie est 0.

1. ***Détection***

Un détecteur est un composant qui détecte l'appui sur une touche et positionne le signal « ***DetectCar »*** à 1. Si aucune pression sur une touche n’a été détectée (voir table 2), le signal de sortie est mis à 0.



**Description VHDL du Détecteur :**



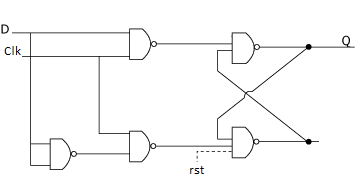
Ce module possède un port d'entrée  « BusExt » de type STD\_LOGIC\_VECTOR à 16 bits, et un port de sortie « Detectar » de type STD\_LOGIC.

À l'intérieur de l'architecture comportementale, un processus est déclenché par le port d'entrée. Dans le processus, une instruction if est utilisée pour vérifier si l'entrée est égale à "1111111111111111". Si c'est le cas, la sortie « DetectCar » est mise à 0, sinon « DetectCar » est mis à 1.

1. ***Bascule D active sur le front montant***

La bascule D est une bascule comportant uniquement une entrée de données : D. La valeur de l'entrée est recopiée sur la sortie à chaque front d'horloge. Cette bascule permet d'assurer un état de sortie stable entre deux fronts d'horloge, et ainsi d'ignorer toute valeur transitoire apparaissant sur son entrée au cours d'un cycle d'horloge. On ajoute parfois un signal reset afin de pouvoir forcer la valeur initiale à la mise sous tension.

Pour mettre une bascule D active sur front montant et possédé un signal reset *rst* qui remettra à zéro de façon asynchrone la sortie Q on doit réaliser le montage ci-dessous :



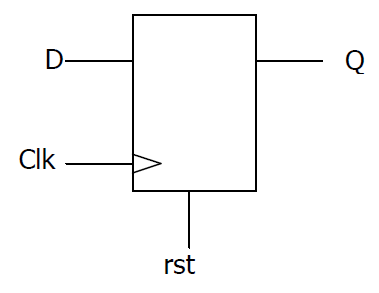
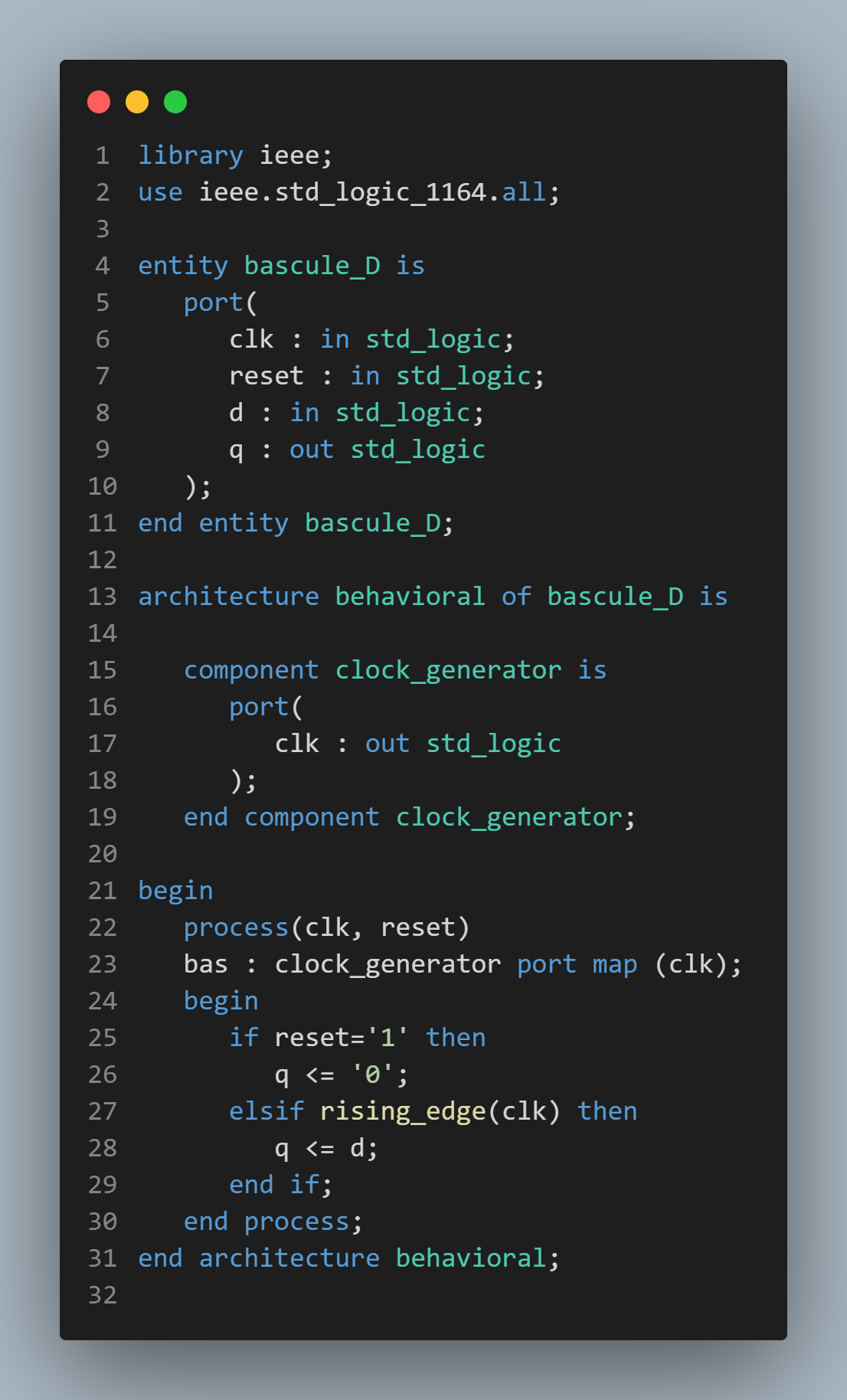




Figure 5 : bascule D active sur niveau

**Description VHDL du Bascule D active à front montant avec reset :**

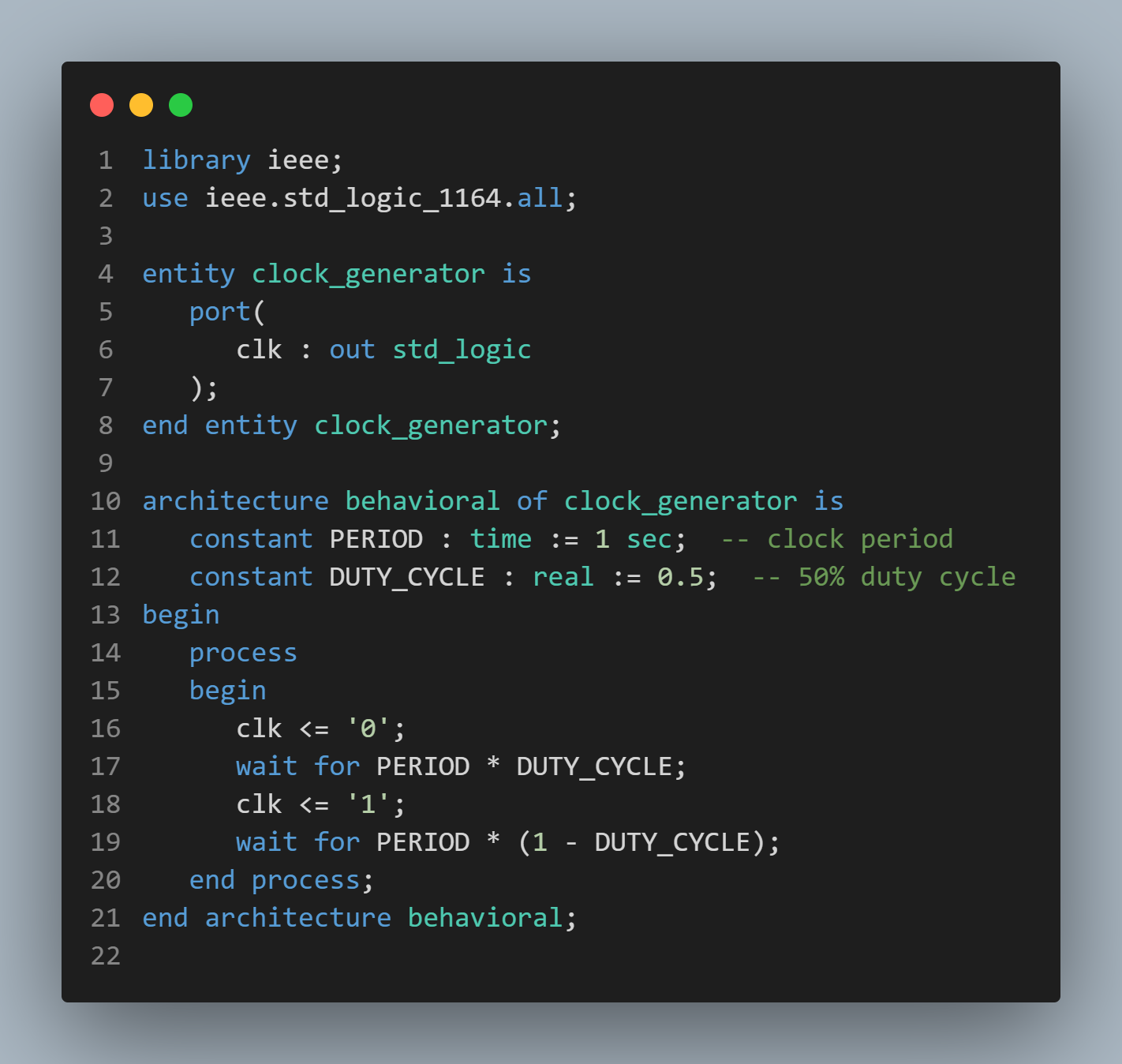
****

Ce code définit une entité appelée Bascule\_D avec quatre ports : clk, reset, d et q. Les entrées clk et reset sont de type std\_logic et les entrées d et q sont de type std\_logic.

Le comportement de la bascule est implémenté dans l'architecture comportementale à l'aide d'un processus sensible aux entrées clk et reset. Le processus vérifie d'abord la valeur de l'entrée de réinitialisation. Si la réinitialisation est '1', la sortie q est mise à '0'. Si reset vaut '0', le processus attend un front montant sur l'entrée clk, puis règle la sortie q sur la valeur de l'entrée d.

Ce code implémentera une bascule D active sur le front montant de l'horloge et dotée d'une entrée de réinitialisation et d'une entrée D. La sortie Q reflétera la valeur de l'entrée D sur le front montant de l'horloge, sauf si l'entrée de réinitialisation est active, auquel cas la sortie Q sera mise à '0'.

**Description VHDL du CLOCK:**

****

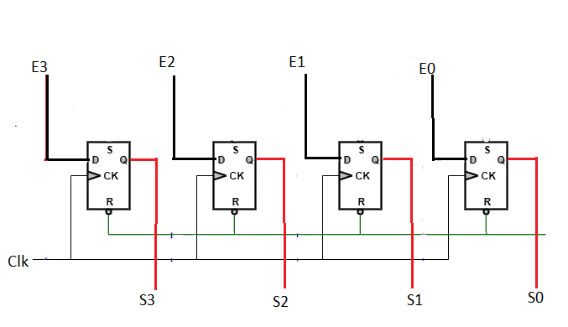
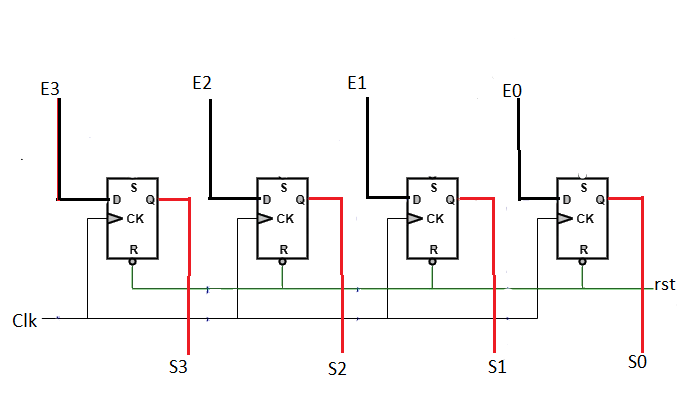
Ce code définit une entité appelée clock\_generator avec un seul port de sortie clk de type std\_logic. Le comportement du générateur d'horloge est implémenté dans l'architecture comportementale à l'aide d'un processus qui bascule la valeur de la sortie clk à une fréquence de 1 Hz et un rapport cyclique de 50 %.

Pour modifier la fréquence de l'horloge, vous pouvez ajuster la valeur de la constante PERIOD. Par exemple, pour générer une horloge avec une fréquence de 2 Hz, vous pouvez régler PERIOD sur 0,5 sec. Pour modifier le rapport cyclique, vous pouvez ajuster la valeur de la constante DUTY\_CYCLE. Par exemple, pour générer une horloge avec un rapport cyclique de 75 %, vous pouvez définir DUTY\_CYCLE sur 0,75.

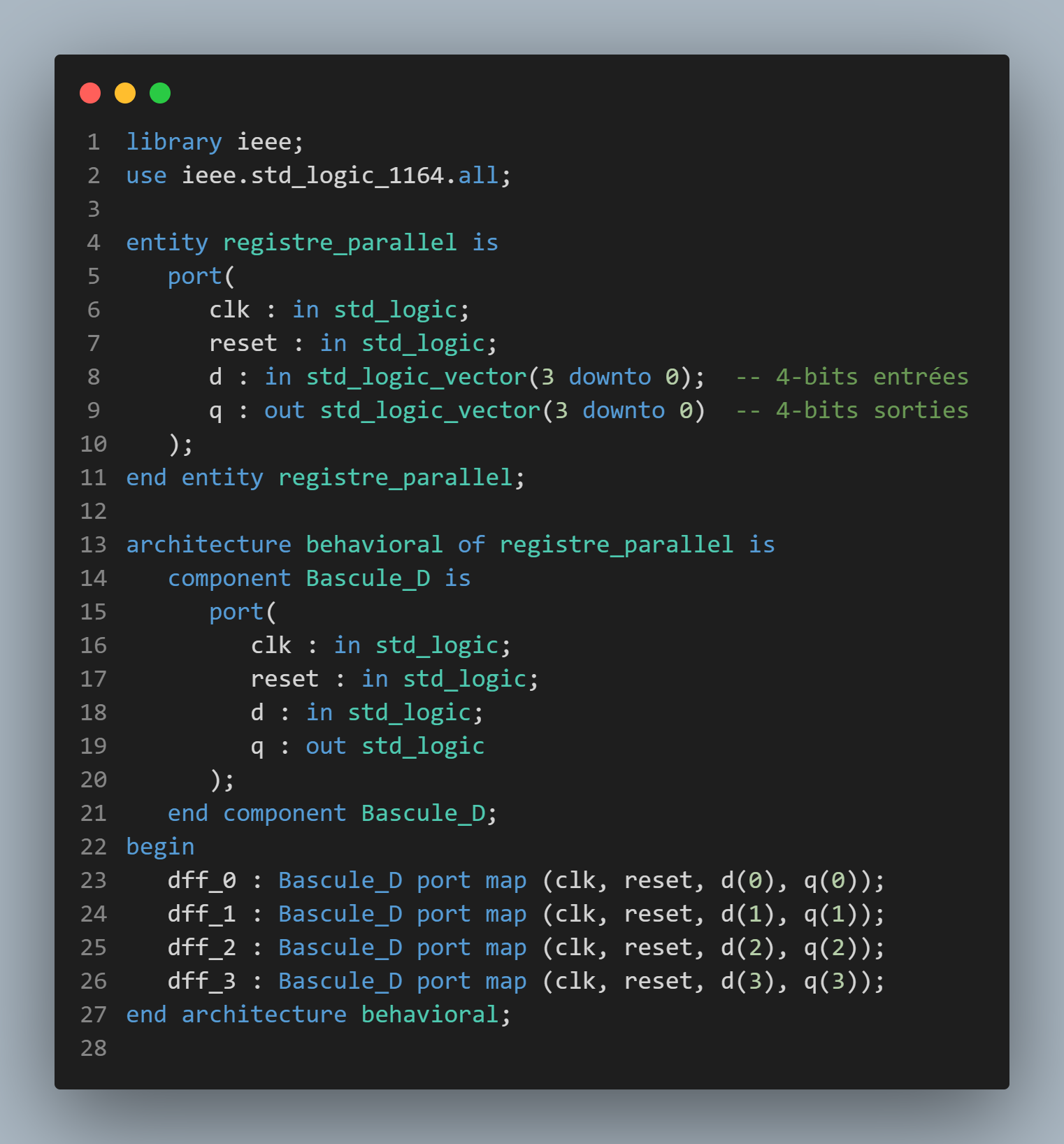
Nous pouvons ensuite utiliser ce signal d'horloge comme entrée clk de la bascule D dans le code précédent

1. ***Registre à chargement parallèle***

Un registre parallèle 4 bits constitué avec des bascules D. Ces bascules possèdent des entrées de forçage au niveau au niveau 0 (R = reset) comme ont étudié dans la section précédente. Ces entrées sont actives quand elles sont au niveau logique 0 et elles ont priorité sur les autres entrées.  
A chaque front montant de l'horloge (transition du niveau 0 au niveau 1), la valeur du bit présent sur l'entrée D de la bascule est recopié sur sa sortie Q.



**Description VHDL d’un registre parallèle avec les bascules D :**

****

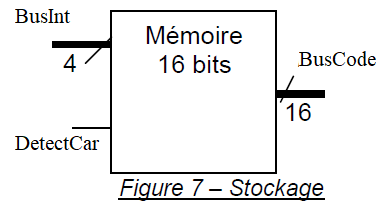
Ce code définit une entité appelée registre\_parllel avec quatre ports d'entrée : clk, reset et d, et quatre ports de sortie : q. Les entrées clk et reset sont de type std\_logic, et les entrées d et q sont de type std\_logic\_vector avec quatre bits (3 à 0).

Le comportement du registre est implémenté dans l'architecture comportementale en utilisant une boucle for pour instancier quatre instances de la bascule D définie dans l'exemple précédent, et en les connectant en série. Les entrées clk et reset sont partagées entre les quatre bascules, et les entrées et sorties d et q sont connectées aux bits correspondants des vecteurs d'entrée et de sortie d et q.

Ce code implémentera un registre parallèle à 4 bits qui décale les valeurs du vecteur d'entrée d dans le vecteur de sortie q sur le front montant de l'horloge, sauf si l'entrée de réinitialisation est active, auquel cas le vecteur de sortie q sera défini sur '0'.

1. ***Stockage du code***

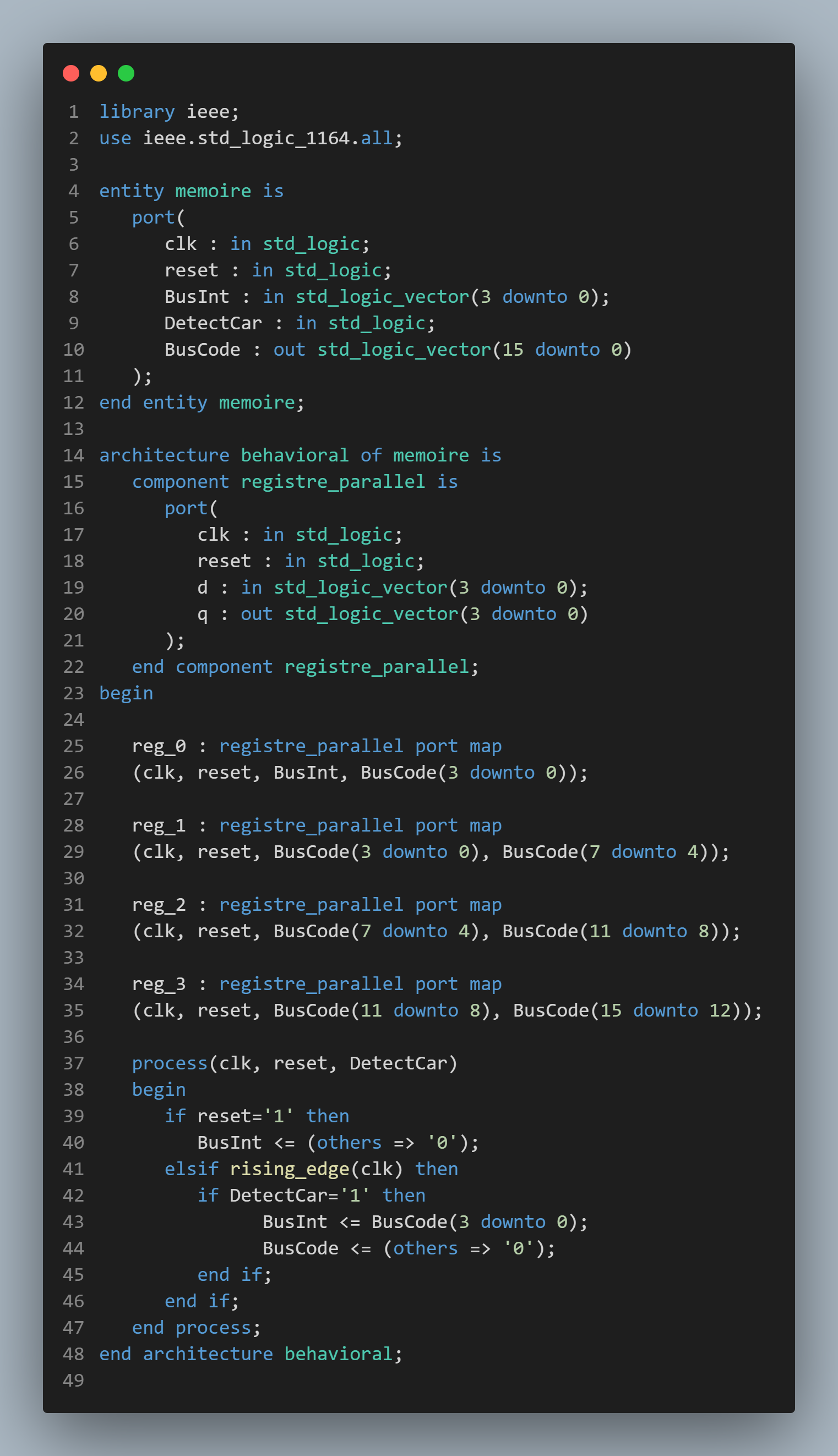
Les caractères saisis par l'utilisateur sont stockés dans une mémoire composée de 4 registres lecture/écriture parallèles 4 bits actifs sur front montant. Quand le signal ***DetectCar*** passe à l'état haut, on capture la valeur du signal présent sur la sortie du codeur ***BusInt*** dans le premier registre et les valeurs précédentes contenues dans les registres sont décalées d'un registre. Le bus ***BusCode*** contient les valeurs des quatre registres. Les 4 premiers bits contiennent la valeur du premier registre (le chiffre de la touche enfoncée en dernier, dernier chiffre du code d’ouverture), les 4 bits suivants du bus contiennent la valeur du second registre et ainsi de suite jusqu’aux 4 bits de poids fort qui contiennent la valeur du 4ème registre (le chiffre de la touche enfoncée en premier, le premier chiffre du code).



**Description VHDL du composant mémoire :**

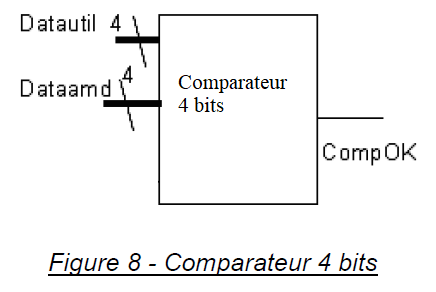
Ce code définit une entité appelée « mémoire » avec quatre ports d'entrée : « clk », « reset », « BusInt » et « DetectCar », et un port de sortie : « BusCode ». Les entrées « clk » et « reset » sont de type std\_logic, l'entrée « BusInt » est de type std\_logic\_vector avec quatre bits (de 3 à 0) et l'entrée « DetectCar » est de type std\_logic. La sortie « BusCode » est de type std\_logic\_vector avec 16 bits (15 jusqu'à 0).

Le comportement du composant « mémoire » est implémenté dans l'architecture comportementale à l'aide de quatre instances du registre parallèle à 4 bits défini dans la section précédente. Les entrées « clk » et « reset » sont partagées entre les quatre registres.

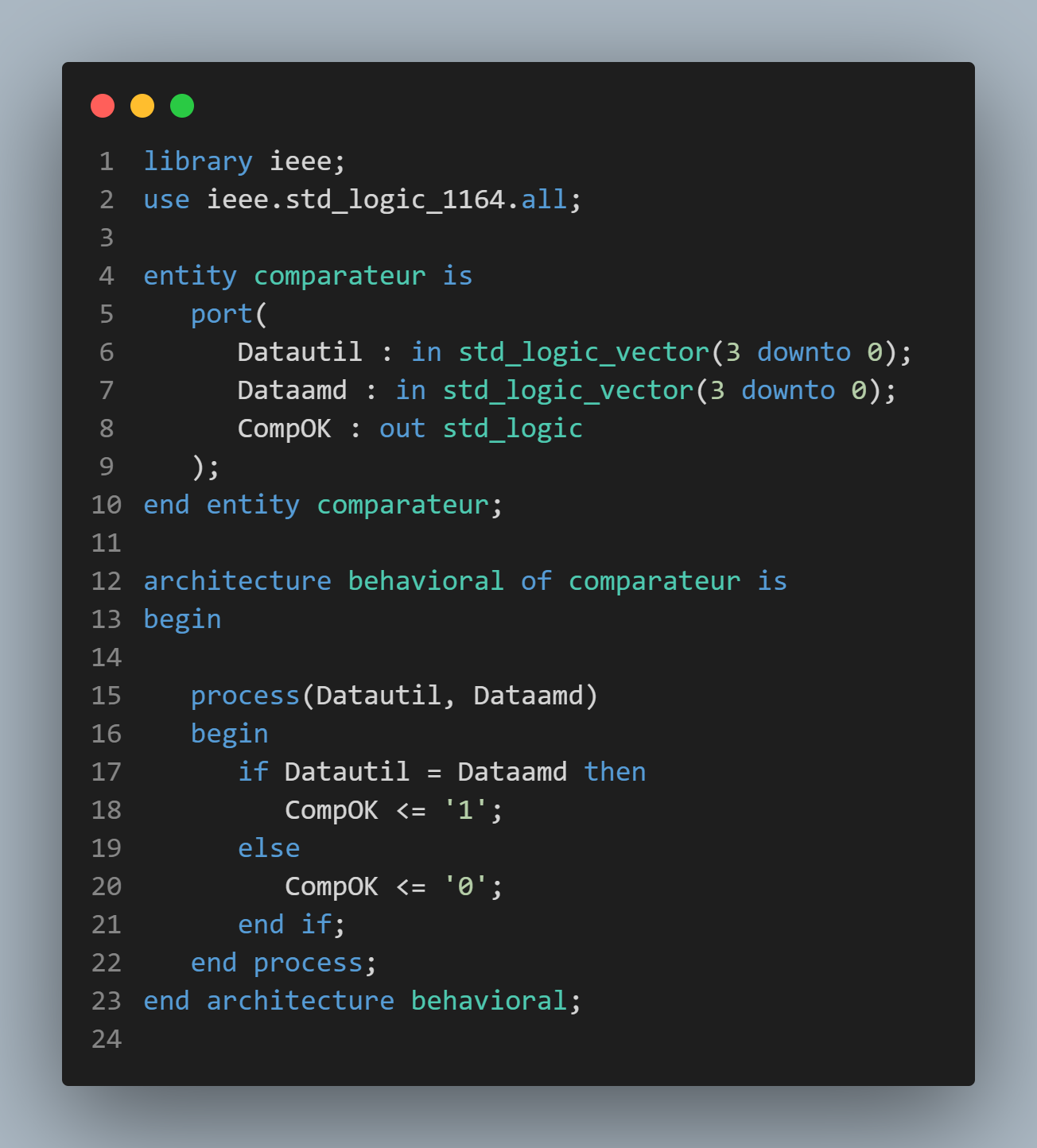
****

1. ***Comparateur 4 bits***

Un comparateur qui détecte l’égalité de deux valeurs sur 4 bits en entrée. Lorsque les signaux ***Datautil*** et ***Dataamd*** sont égaux alors le signal ***CompOK*** passe à 1, sinon ***CompOK*** est égal à 0.



**Description VHDL du comparateur :**

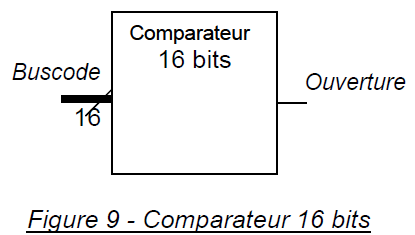
****

Ce code définit une entité appelée comparateur avec deux ports d'entrée : « Datautil » et « Dataamd », et un port de sortie : « CompOK. » Les entrées « Datautil » et « Dataamd » sont de type std\_logic\_vector avec quatre bits (de 3 à 0) et la sortie « CompOK » est de type std\_logic.

Le comportement du comparateur est implémenté dans l'architecture comportementale à l'aide d'un processus qui compare les entrées « Datautil » et « Dataamd ». Si les entrées sont égales, la sortie « CompOK » est mise à '1'. Si les entrées ne sont pas égales, la sortie « CompOK » est mise à '0'.

1. ***Comparateur 16 bits***

Le comparateur 16 bits est composé de 4 comparateurs 4 bits, vus précédemment. L'entrée ***Datautil*** de chaque comparateur est connectée à 4 bits du bus d'entrée du comparateur 16 bits ***BusCode***. Ce bus est le bus de sortie des registres de la mémoire (cf Figure 2) et contient le code tapé par l’utilisateur. Le bus ***Dataamd*** de chaque comparateur 4 bits contient la valeur d'un des caractères de la clef que vous fixerez à 1234. Lorsque le code entré par l'utilisateur est égal au code de la clef, le signal ***ouverture*** passe à 1, sinon le signal ***ouverture*** reste à 0

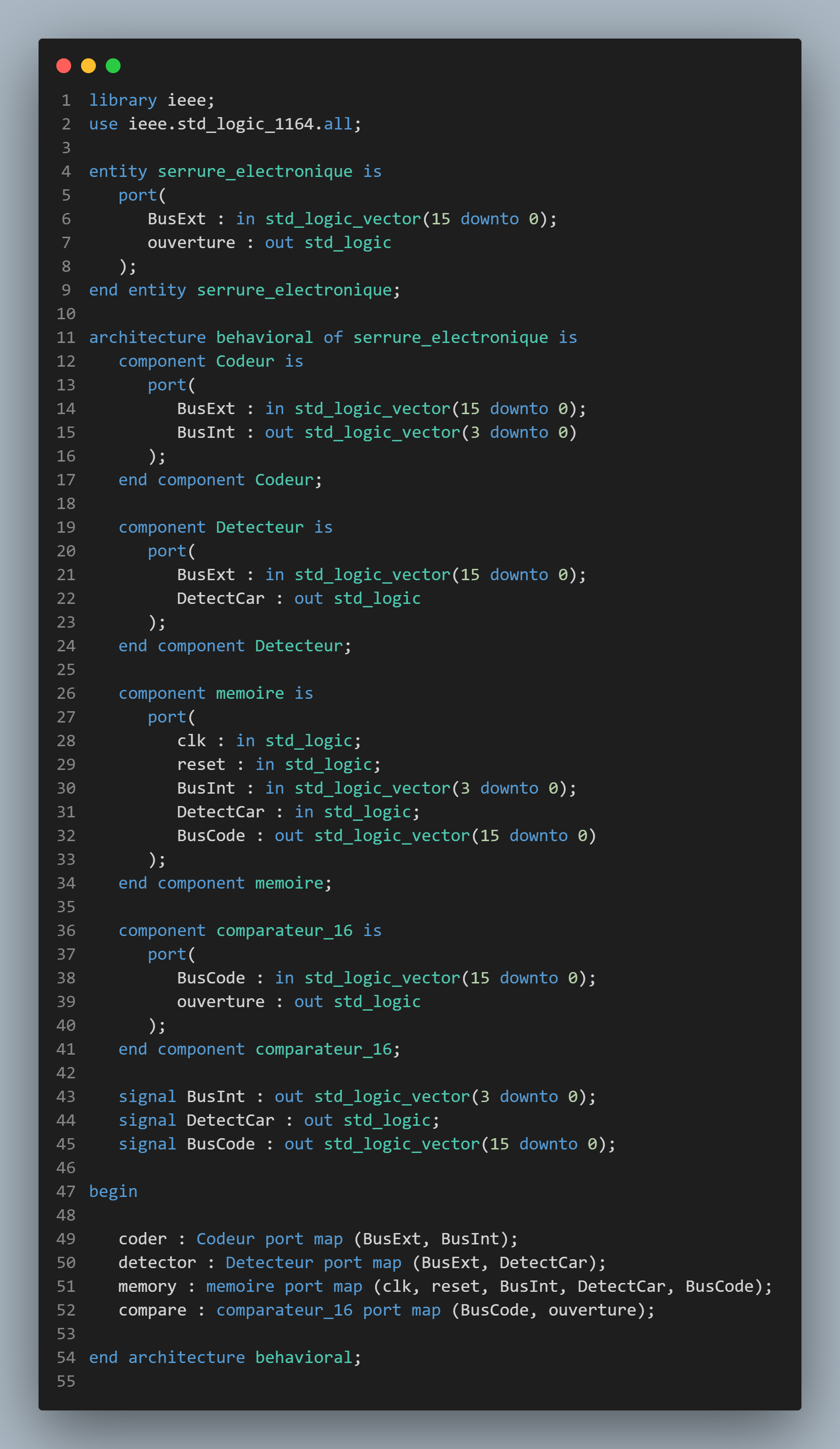


**Description VHDL du comparateur à 16 bits :**



1. ***Assemblage***

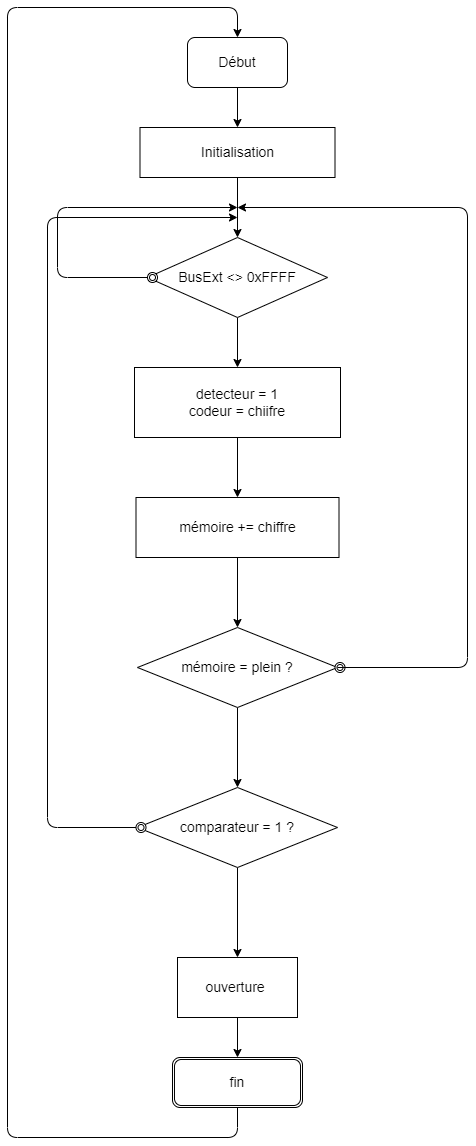
L’assemblage des différents codes précédents assure le fonctionnement de notre serrure électronique, ci-dessous le code de l’application principale en VHDL :



***III. Partie 2 : Implémentation sur carte STM32 (carte Blue Pill)***

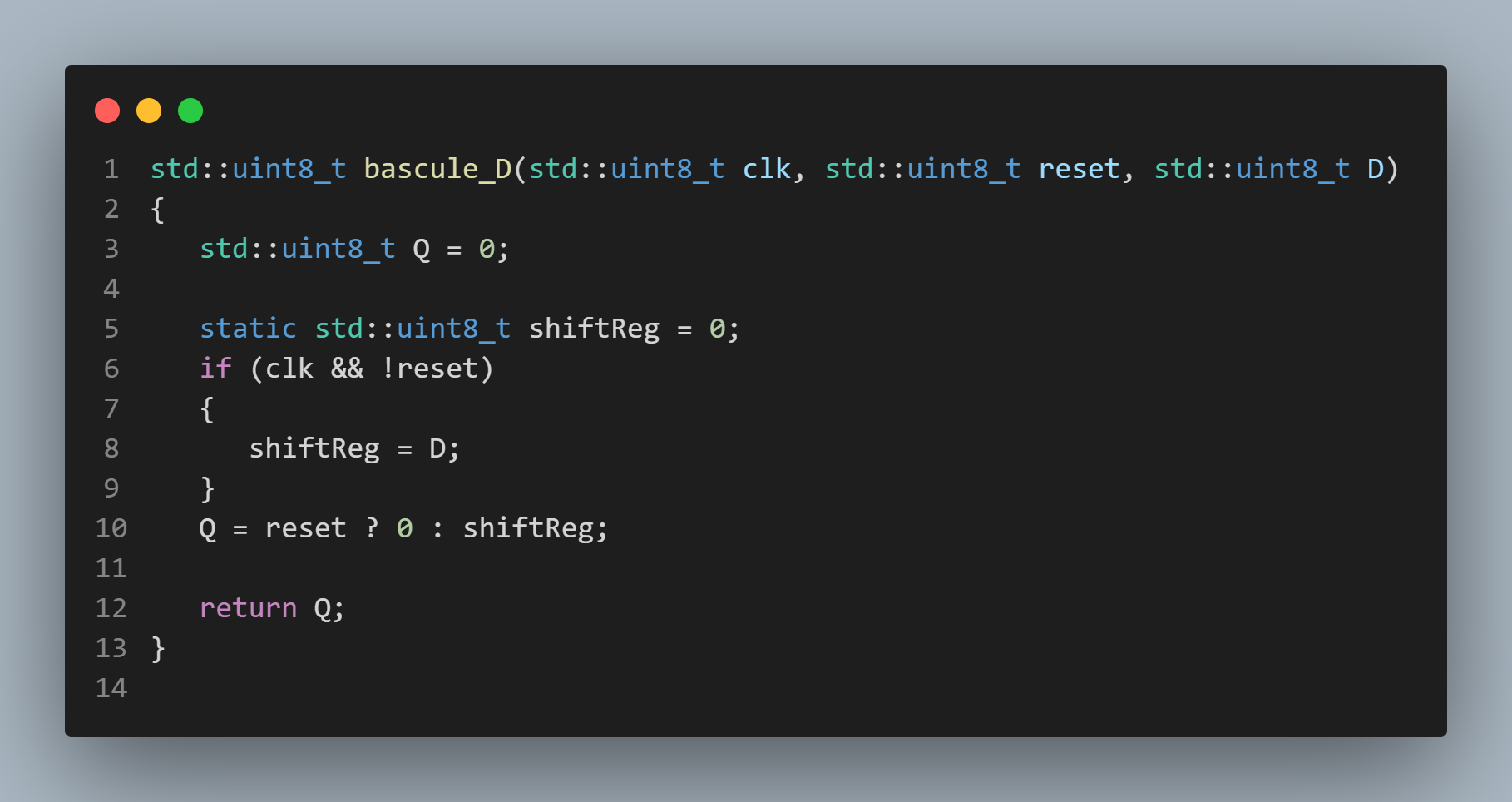
Pour l’implémentation du système sur la carte STM32 ,suivre le même principe de fonctionnement cité dans la première partie.

Organigramme :

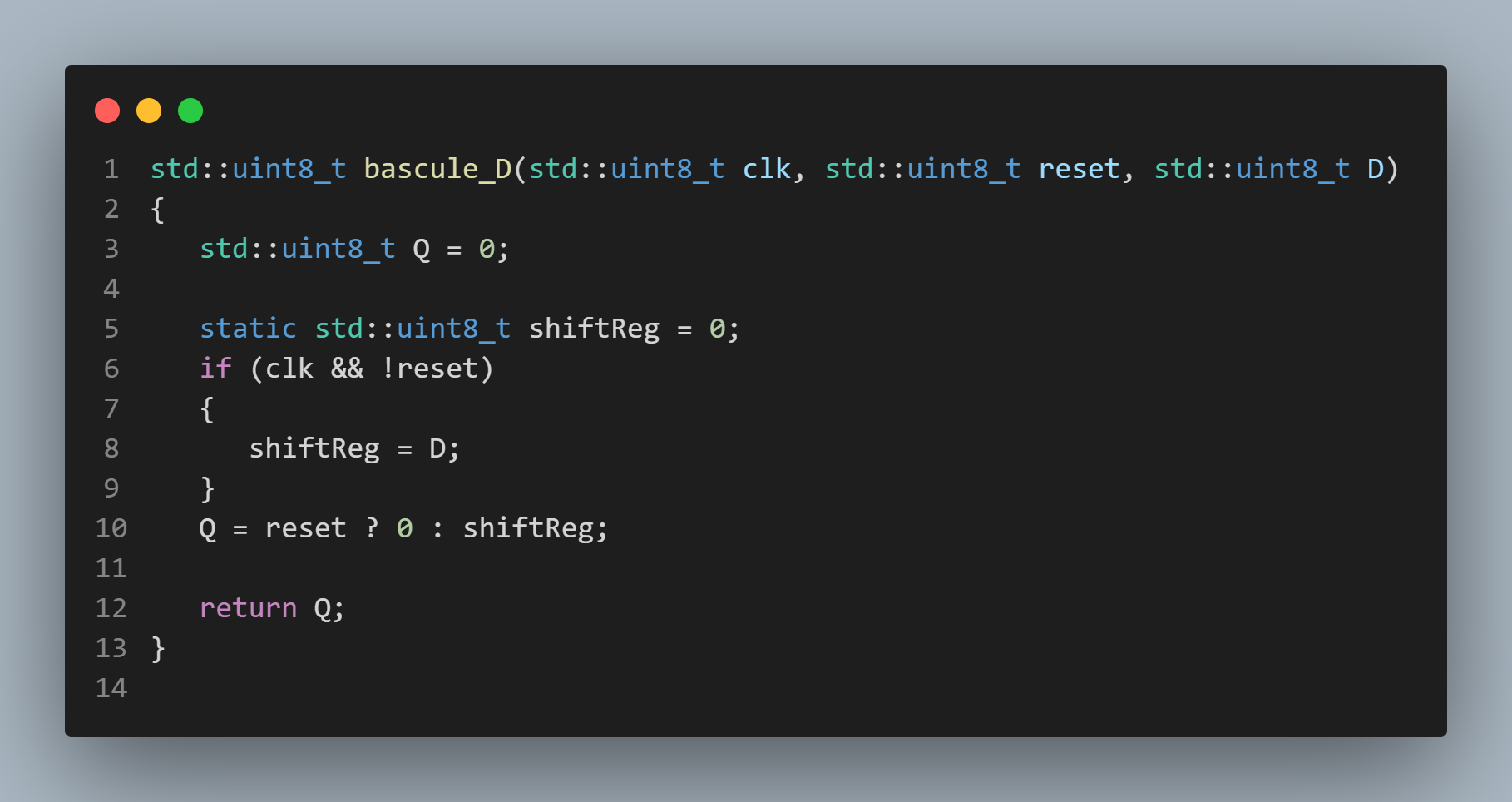


Ci-joint les codes cpp pour les implémentés dans la carte Blue Pill :

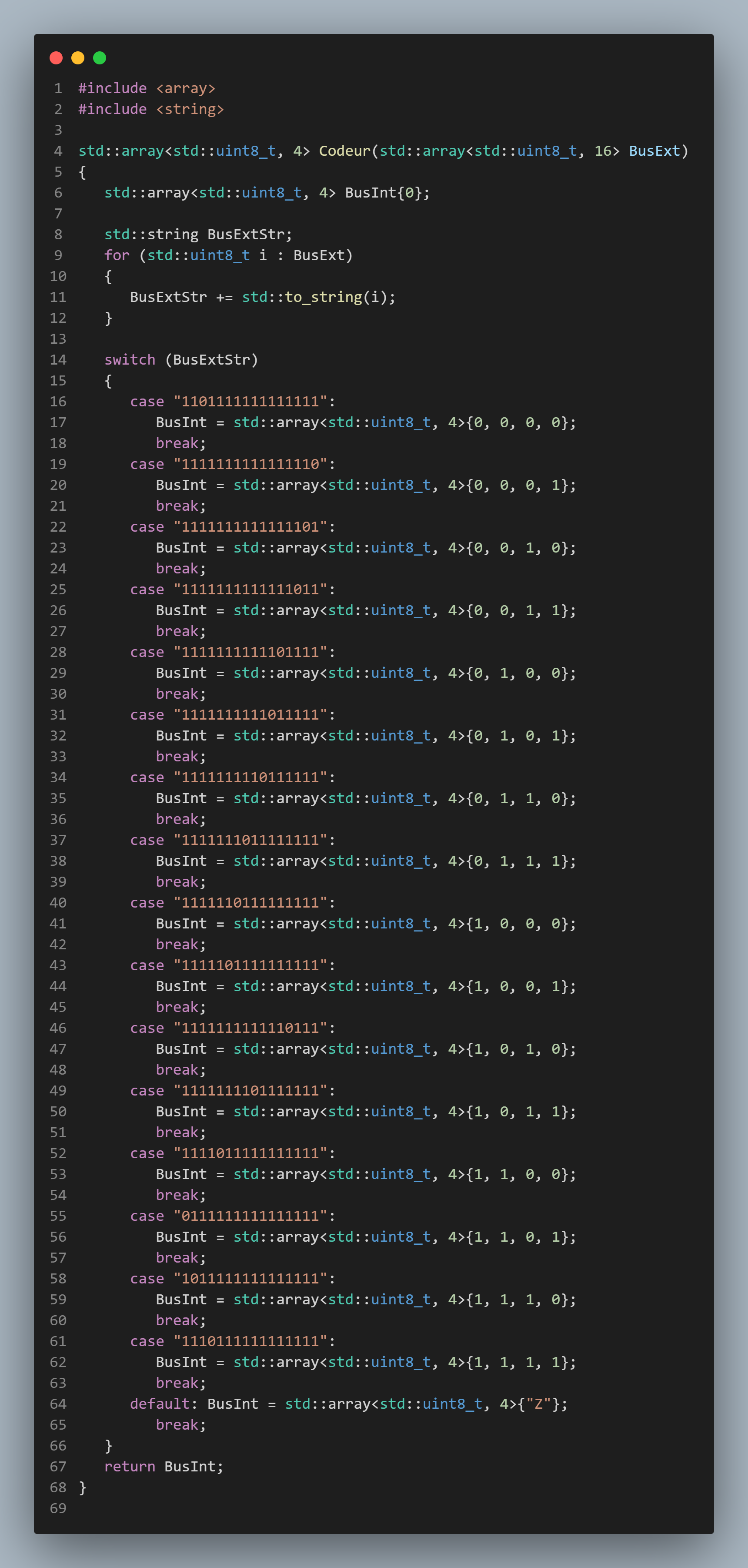
**La fonction d’une bascule D en cpp :**



**La fonction du CLOCK en cpp :**

****

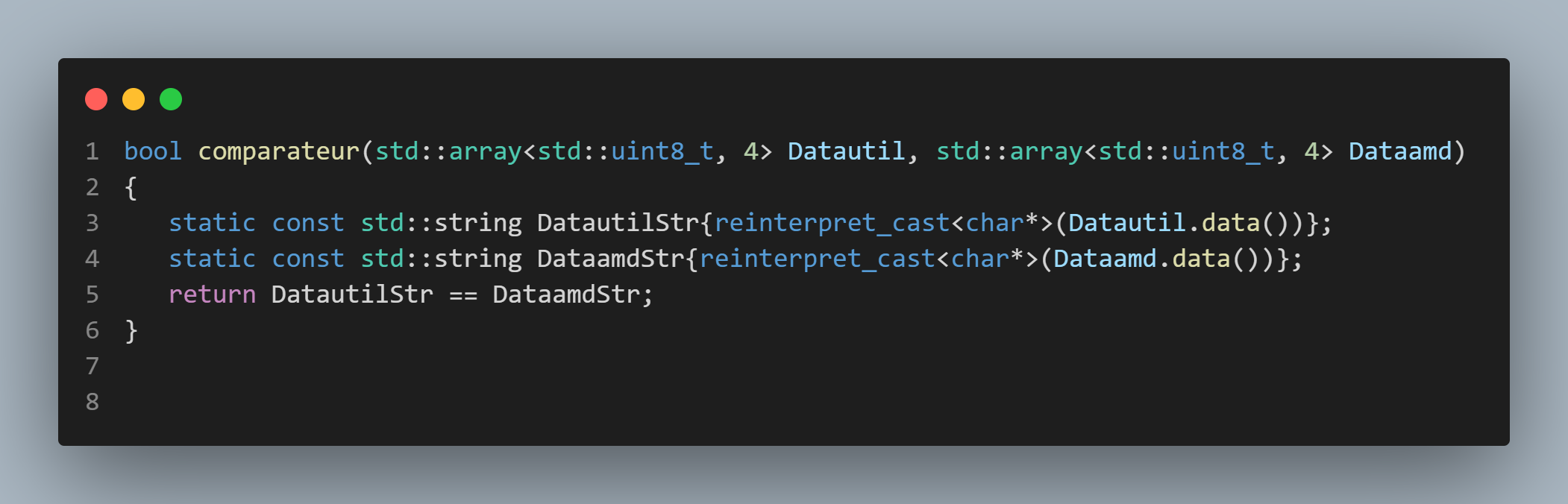
**La fonction du codeur en cpp :**

****

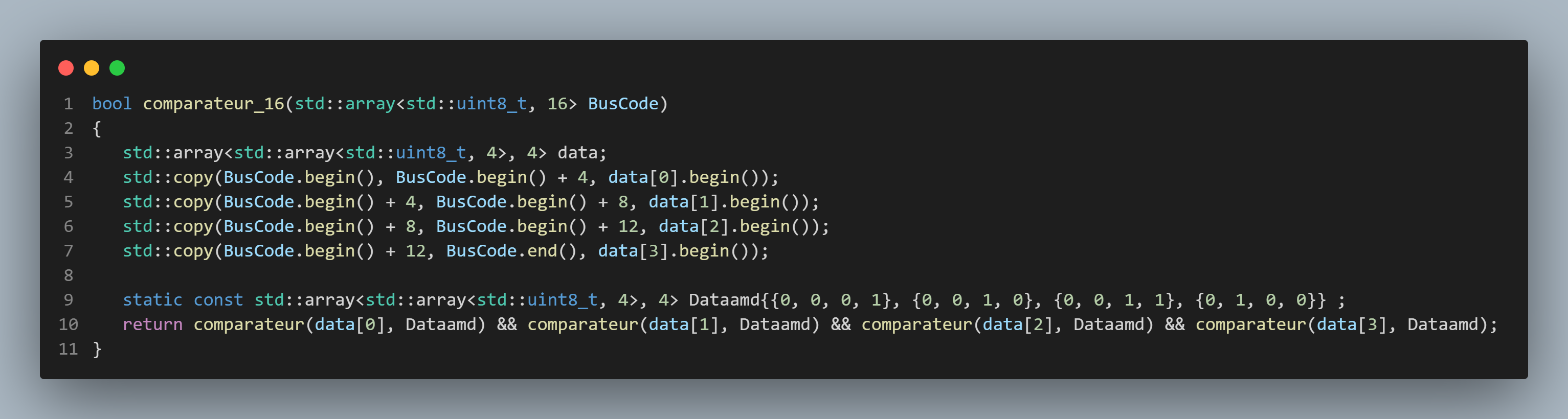
**La fonction du detecteur :**

****

**La fonction du comparateur à 4 bits:**

****

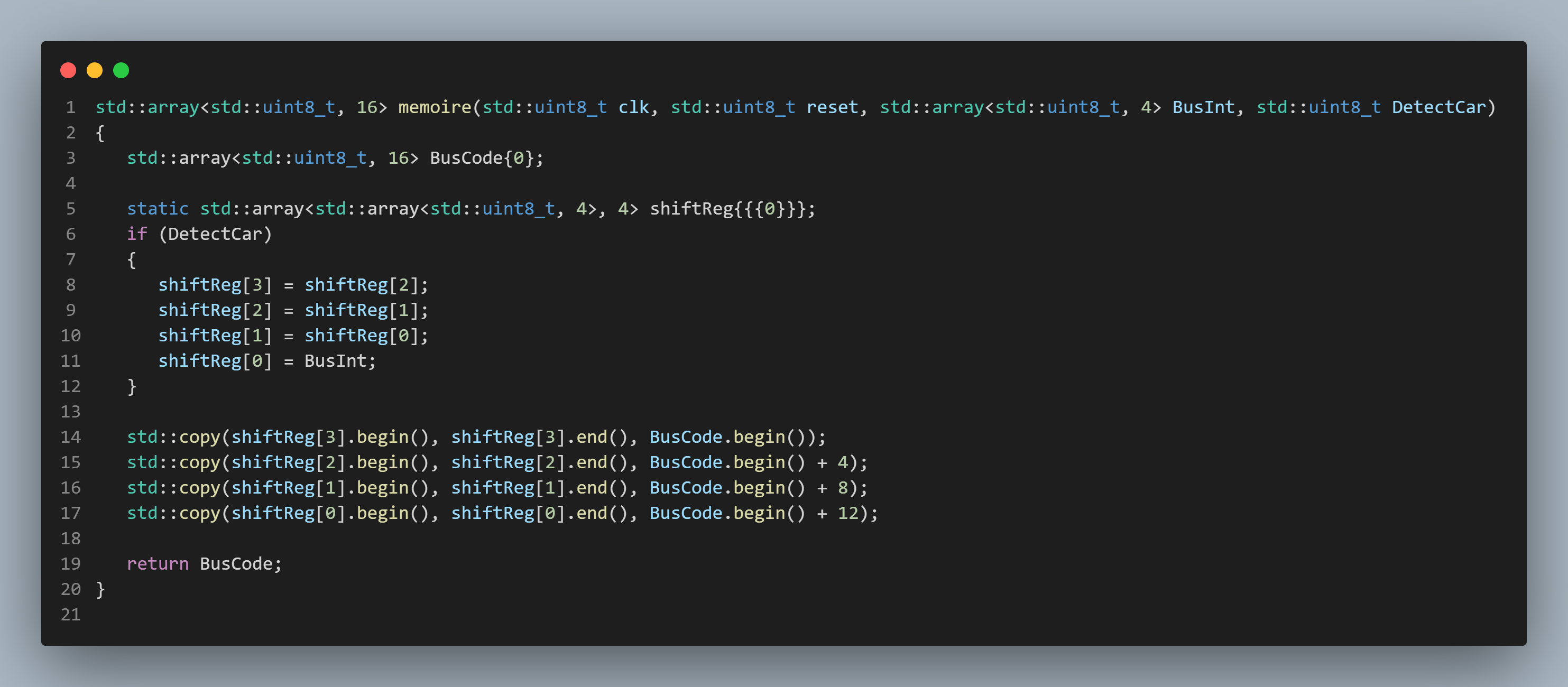
**La fonction du comparateur à 16 bits :**

****

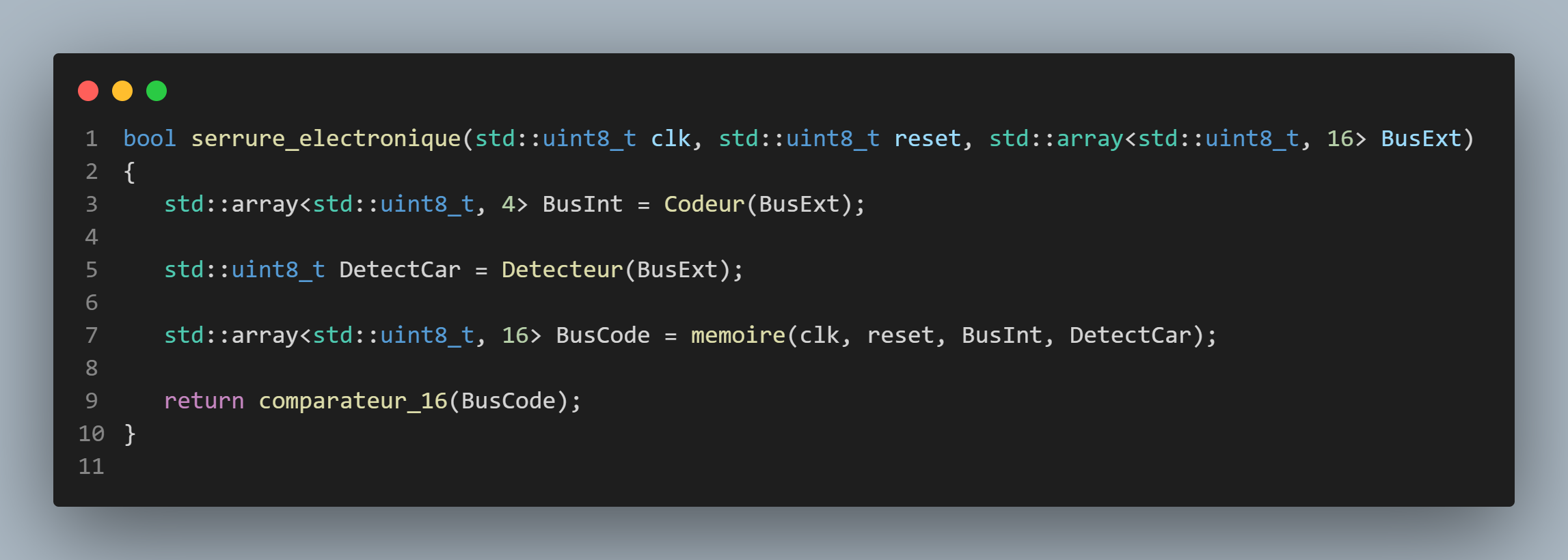
**La fonction du registre parallèle :**

****

**La fonction du mémoire :**

****

**La fonction principale de la serrure électronique :**

****

**Conclusion**

Il est important de signaler que ce mini projet a présenté l’occasion de consolider notre formation par la pratique de nos connaissances en système embarqué.

Ce travail, nous a permis d’avoir une idée générale sur le formalisme ainsi que le système de ventilation d’un local, à travers les étapes suivantes : On commence par la description de mini projet.

Par la suite, nous avons entamé une étude théorique de projet où nous avons présenté les différents outils de travail. Et finissons par le développement du projet.

Cette démarche nous a permis de bien étudier le besoin et de proposer un programme de à travers le simulateur mbed.

Finalement ce projet présente un profit à l’échelle personnelle puisqu’il nous a permis de consolider nos connaissances en microcontrôleur et interface.