Applications d’éclairage par LED à faible consommation : technique de gestion d’alimentation

par Ezana Haile, ingénieur d'application principal de la division Analog & Interface Products de Microchip Technology Inc.

L’équilibre entre l’efficacité lumineuse des diodes électroluminescentes (LED) et la puissance électrique d’une application d’éclairage donnée constitue une spécification essentielle pour déterminer la qualité de la lumière visible émise par une matrice de LED pilotée électroniquement. En effet, sur des applications à faible consommation, la source de courant étant limitée, il se peut qu’il ne soit pas possible d’allumer toutes les LED de la matrice simultanément pour obtenir le maximum de luminosité. Afin d’obtenir une luminosité optimale, la dissipation énergétique des LED doit être gérée efficacement, depuis l’alimentation faible consommation. Pour ce faire, il faut utiliser une technique de gestion d’alimentation consistant à n’alimenter qu’une rangée de LED de la matrice pendant un temps très bref. L’alternance entre les rangées de LED, allumées tour à tour, reste invisible à l’œil nu, tandis qu’on obtient l’intensité lumineuse requise.

La puissance disponible et l’intensité lumineuse requise pour l’application doivent être préalablement établies pour pouvoir déterminer le nombre de LED à allumer pendant un intervalle de temps donné. Il convient de comparer l’intensité lumineuse et les caractéristiques du courant direct pour pouvoir sélectionner les LED possédant le niveau d’intensité requis. Une fois le nombre de LED requis défini, il faut calculer combien d’entre elles peuvent être alimentées en même temps selon un intervalle de temps donné, en divisant le courant total requis pour les LED par le courant disponible, tel que ci-dessous :

Nbr de LED dans une rangée = Courant total requis pour les LED / Courant total disponible à la source

La fréquence à laquelle la rangée de LED doit être allumée/éteinte doit être définie de telle sorte que la durée d’allumage soit assez longue pour illuminer totalement la rangée. La durée d’extinction, quant à elle, est fixée en fonction du temps nécessaire pour que la luminosité de la rangée commence à faiblir visuellement. La durée d’extinction limite le nombre de rangées de LED supplémentaires pouvant être pilotées via la gestion d’intervalles de temps.

Cette technique peut être mise en œuvre pour un coût modique à l’aide d’une source d’horloge, de bascules numériques pour piloter les rangées de LED, et d’une porte OU pour la détection d’une condition de démarrage, avec un simple commutateur marche/arrêt. La figure 1 montre le schéma électronique d’une bascule D permettant de contrôler les quatre rangées de LED.

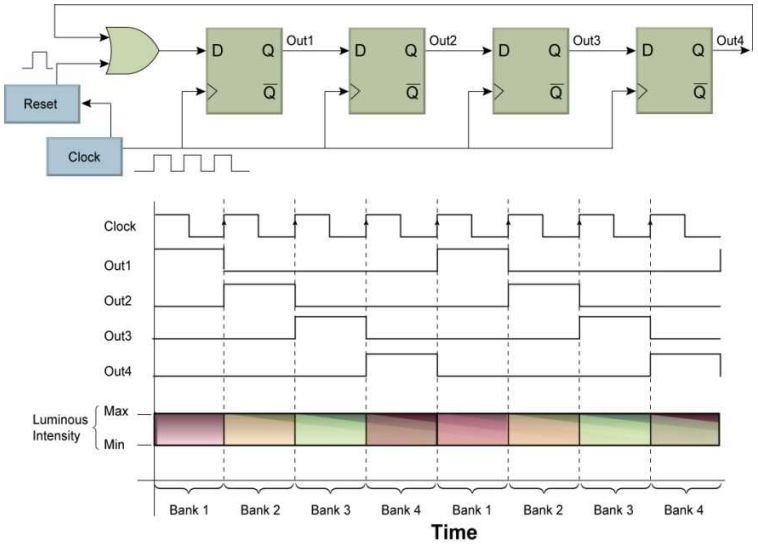


Figure 1 : Éclairage par LED à intervalles avec bascule numérique

Au démarrage, l'état de la bascule requiert une impulsion de démarrage pour pouvoir changer. Ce signal doit intervenir au moins une fois par cycle d'horloge, pour qu'il puisse être détecté par la première bascule lors du front montant de l'horloge. La durée du signal de démarrage doit être brève et ne peut excéder un cycle d'horloge, sous peine que les deux premières sorties de la bascule « tombent » au même moment, et comme la source de courant est limitée, l'application d'éclairage ne fonctionnera pas correctement. Avec cette configuration, chaque rangée de LED est totalement allumée à chaque front montant de l'horloge, bien que, visuellement, l'œil humain voie toutes les LED s'allumer simultanément.

Cependant, cette technique est répétitive et n'offre pas la flexibilité recherchée par les développeurs. Avec cette méthode, seuls deux états sont possibles : allumé ou éteint. C'est un circuit qui peut convenir pour des applications tel que le rétro-éclairage par LED, mais si la gradation de l'intensité lumineuse ou l'affichage de motifs sont requis, mieux vaut avoir recours à un circuit intégrant un microcontrôleur, qui offrira la flexibilité idéale tout en ayant un impact minimum sur le coût total de la solution. Ce genre de circuit est facile à construire, requiert peu de composants (le microcontrôleur se chargeant de piloter chaque rangée de LED), et peut également détecter les instructions fournies par l'utilisateur pour le contrôle de l'intensité lumineuse et la sélection des motifs.

Un microcontrôleur 8 bits économique ayant peu de broches, du type de ceux de la famille des PIC10F ou des PIC12F de Microchip, doté d'une barrette de ports d'E/S tel que le MCP23018 de Microchip, constitue une solution au bon rapport coût/efficacité. Les barrettes d'extension d'E/S peuvent également se révéler utiles pour le pilotage des LED, si le circuit d'éclairage est distant par rapport au microcontrôleur.

Les barrettes d'extension d'E/S permettent d'ajouter des ports d'E/S à un microcontrôleur. Dans cette application, le microcontrôleur commande les ports de la barrette d'extension d'E/S via le protocole I^2C™, pour allumer et éteindre les LED, et les broches d'E/S du microcontrôleur

peuvent être utilisées pour détecter les instructions fournies par l'utilisateur via les bouton-poussoir, ou pour contrôler l'intensité lumineuse grâce au convertisseur analogique-numérique (CAN) intégré (servant à détecter un niveau potentiométrique).

Les barrettes d’extension d’E/S peuvent être dotées de sorties à collecteur ouvert ou symétrique *(push-pull)*. Les microcontrôleurs fonctionnant à 3,3 V ou à une tension inférieure, une barrette d’extension d’E/S à sorties à collecteur ouvert se prête bien à cette application. Avantage : les LED peuvent avoir une tension de fonctionnement de 5 V ou supérieure tandis que le microcontrôleur et la barrette d’extension d’E/S sont alimentés à une tension inférieure. Le MCP23018 est une barrette d'extension d'E/S 16 bits à sortie à collecteur ouvert et interface I^2C. La figure 2 montre le schéma d’un circuit de barrettes d’extension d’E/S avec sortie à collecteur ouvert avec rappel à une tension supérieure à la tension d'entrée du microcontrôleur.

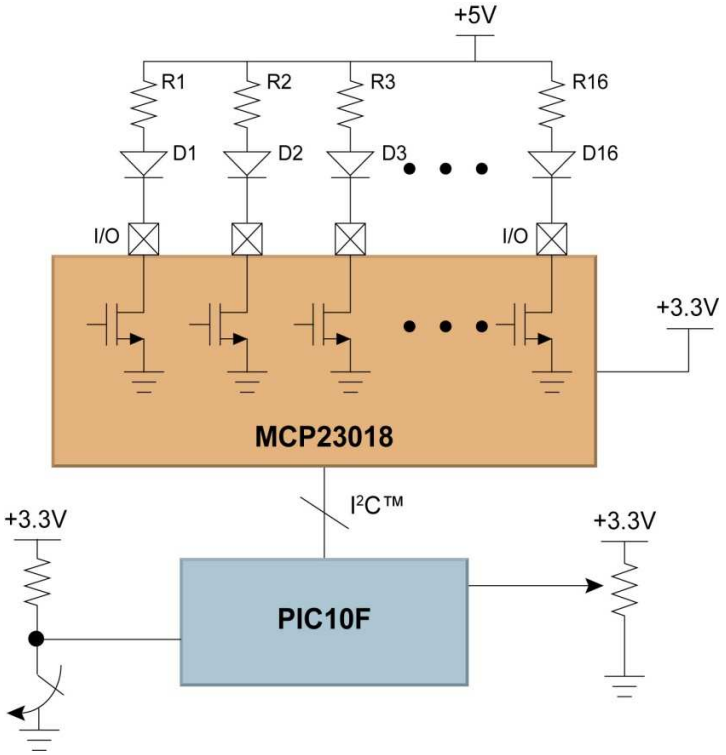


Figure 2 : Solution d’éclairage intégrant un microcontrôleur économique avec barrette d'extension d'E/S

Quand le port d’E/S est réglé à l’état bas, la tension du port de la barrette d’extension est égale à 0 V et le courant circule, ce qui polarise la LED en sens direct et l’allume. La résistance de polarisation de la LED limite le courant envoyé vers la LED en fonction de l’intensité lumineuse requise, et fonctionne également comme résistance de rappel à l’état haut *(pull-up)* pour la sortie à collecteur ouvert.

Quand le port de sortie de la barrette d’extension d’E/S est à l’état haut, la sortie à collecteur ouvert est inactive ou à l’état de haute impédance, et la tension au port de la barrette d’extension d’E/S est apportée au + 5 V par la résistance de rappel à l’état haut. La LED est éteinte car le courant ne circule pas. Autre avantage d’une sortie à collecteur ouvert : quand le port est à l’état de haute impédance, la LED met plus longtemps à s’éteindre à cause de la

capacité parasite, ce qui permet à la rangée de LED suivante de rester allumée pendant légèrement plus longtemps que dans d’autres applications similaires avec sortie symétrique.

Les ports d'E/S du MCP23018 peuvent piloter jusqu'à 16 LED. La quantité de courant consommée par le port d'E/S quand une LED est allumée à plein régime peut être limitée par le module de pilotage des sorties de la barrette d'extension. La tension à l'état bas du port E/S est spécifiée pour 0,6 V au maximum à 8,5 mA. Si le courant est supérieur à 8,5 mA, ce niveau de tension à l'état bas augmente de manière exponentielle. Le courant maximum absolu est 25 mA.

Si le courant de source est limité à 5 V/ 50 mA, par exemple, et que 2 mA sont alloués au microcontrôleur, à la barrette d'extension d'E/S et aux résistances utilisées pour la détection des entrées de l'utilisateur, alors le reste du courant disponible peut être utilisé pour l'éclairage par LED. Quatre LED peuvent être pilotées par rangée si l'intensité lumineuse des LED à environ 10 mA convient pour l'application. Dans cet exemple, la valeur de la résistance de limitation du courant sera d'environ 440 Ω.

La temporisation montrée à la figure 1 peut être réalisée à l'aide d'un petit programme pour microcontrôleur. Par exemple, dans l'exemple de code 1, le sous-programme principal peut être une boucle `while` infinie. À des intervalles définis par le microcontrôleur (Timer0), une routine d'interruption (ou ISR) envoie une commande I^2C à la barrette d'extension d'E/S pour allumer une seule rangée de LED. La routine d'interruption comptabilise (ou enregistre) le statut de la rangée de LED, c'est-à-dire celui de celle qui est en train de s'allumer, ainsi que l'état du port correspondant de la barrette d'extension (haut ou bas). Au début, ou après un redémarrage du microcontrôleur, à la première interruption Timer0, la rangée 1 est allumée. Puis, à la seconde interruption, la rangée 1 est éteinte et la rangée 2 est allumée. Lors de la 3ème interruption, la rangée 2 est éteinte et la rangée 3 est allumée. Enfin, à la 4ème interruption, la rangée 3 est éteinte et la rangée 4 est allumée ; la variable du compteur de rangées est vidée. Lors de l'interruption suivante, le cycle est répété en allumant la rangée 1 et en éteignant la rangée 4. Les autres variables, telles que `Command_byte`, `Address_pointer_bank1n2` et `Address_pointer_bank3n4` sont des constantes définies en en-tête. Avec une implémentation de ce type, le circuit produit en sortie l'intensité lumineuse requise tandis que visuellement il semble que toutes les LED s'allument simultanément à partir de la source d'alimentation disponible.

```
void main (Void){
        initialize();    //initialize the PICmicro peripherals including Timer0, and I/O expander
        peripherals
        bank_counter = 0;    //Clear the bank counter variable
        while(1){}    //infinite while loop
}
Void Interrupt int_service(void){
        TurnOffAll_LEDs();        //subroutine to turn off all previously lit LEDs
        I2C_start();    //I2C protocol – start signal subroutine
        I2C_send(Comand_byte);  //I2C protocol – send byte subroutine

        If(bank_counter == bank1){          // bank1 is a constant defined as 0
                I2C_send(Address_pionter_bank1n2);  //Send peripheral register address pointer
```

```
                I2C_send('0000 1111');  //I2C protocol – send bank status (bank 1 on and bank 2
        off)
                bank_counter = bank_counter + 1;
        }
        If(bank_counter == bank2){      // bank1 is a constant defined as 1
                I2C_send(Address_pionter_bank1n2);  //Send peripheral register address pointer
                I2C_send('1111 0000');  //I2C protocol – send bank status (bank 2 on and bank 1
        off)
                bank_counter = bank_counter + 1;
        }
        If(bank_counter == bank3){      // bank1 is a constant defined as 2
                I2C_send(Address_pionter_bank3n4);  //Send peripheral register address pointer
                I2C_send('0000 1111');  //I2C protocol – send bank status (bank 3 on and bank 4
        off)
                bank_counter = bank_counter + 1;
        }
        If(bank_counter == bank4){      // bank1 is a constant defined as 3
                I2C_send(Address_pionter_bank3n4);  //Send peripheral register address pointer
                I2C_send('1111 0000');  //I2C protocol – send bank status (bank 4 on and bank 3
        off)
                bank_counter = 0;
        }
        I2C_stop();    //I2C protocol – stop signal subroutine
}
```

Exemple de code 1 : Lignes de code d'une routine d'interruption du microcontrôleur

Contrôle de l'intensité et affichage de motifs LED

On peut facilement afficher des motifs intéressants à l'aide d'une application utilisant le microcontrôleur. Les états des sorties de la barrette d'extension sont chargés depuis 2 octets de RAM de variables, nommés respectivement `Bank1n2_pattern` et `Bank3n4_pattern`. Chaque quartet (ou *nibble*) de chaque octet correspond à une rangée de LED. Une table de correspondance établie au préalable comprend les suites de 0 et de 1 correspondant à chaque rangée.

Pour surveiller en continu l'état allumé/éteint des interrupteurs à bouton-poussoir, une déclaration IF THEN est ajoutée au code du microcontrôleur : c'est le sous-programme principal de la boucle infinie `while`. Quand l'interrupteur est pressé pendant un moment, l'un des motifs de la table de correspondance est chargé dans les 2 octets de RAM, nommés `Bank1n2_pattern` et `Bank3n4_pattern`. Quand l'interruption Timer0 se produit, le nouveau motif est envoyé à la barrette d'extension d'E/S et les LED sont allumées conformément au motif. Le sous-programme principal parcourt la table de correspondance pendant que le bouton-poussoir reste enfoncé par l'utilisateur. Pour un affichage clignotant, il suffit d'envoyer le complément du motif précédemment envoyé. Par exemple, si le motif de la rangée 1 est « 0101 », son complément est « 1010 », tel qu'on peut le voir dans l'exemple de code 2, qui est un fragment tiré de l'exemple de code 1.

```
*
*
*
If(bank_counter == bank1){        // bank1 is a constant defined as 0
        I2C_send(Address_pionter_bank1n2);  //Send peripheral register address pointer
        Bank1n2_pattern = ~Bank1n2_pattern; //complement the variable using '~'
        I2C_send((Bank1n2_pattern | '0000 1111'));  //I2C protocol – send bank status
                                                    //(use OR '|' to set bank 1 on and bank 2 off)
        bank_counter = bank_counter + 1;
}
*
*
*
```

Exemple de code 2 : Affichage de motifs

Cependant, une pause doit être introduite dans le sous-programme d'interruption, avant les déclarations IF THEN pour l'allumage/extinction des rangées, afin que l'alternance des motifs soit visible par l'œil humain. Pour ce faire, on utilise un drapeau `Delay_ON` de sorte que lors de l'interruption suivante, un compteur décompte le nombre d'interruptions devant se produire avant la fin de la pause (ou délai). La valeur attribuée au délai peut être sélectionnée par l'utilisateur à l'aide d'un potentiomètre, situé à la connexion de la languette centrale à l'entrée du convertisseur analogique-numérique (CAN) intégré au microcontrôleur. Les données numériques du CAN sont converties entre un délai minimum et un délai maximum grâce à la détection des 4 bits supérieurs, ce qui fait 16 niveaux différents au total. Un rapport plus précis peut être réglé en détectant les 5 bits supérieurs, soit 32 niveaux au total. Le délai maximum correspond à la vitesse la plus lente à laquelle les LED peuvent clignoter, tandis qu'avec un réglage sur le délai minimum, toutes les LED semblent allumées à pleine puissance. Le schéma de temporisation de la figure 3 montre la position du délai, t_{DELAY}.

```
*
*
*
Void Interrupt int_service(void){
        TurnOffAll_LEDs();      //subroutine to turn off all previously lit LEDs
        If (Delay_ON == ON){    // ON is a constant defined as 1
                Update_Timer0_Counter(Timer0delay_interval);      //subroutine to update
Timer0
                                        // Timer0delay_interval is a constant to set the minimum
delay
                Delay_Counter = Delay_Counter – 1;          //count down the number of
interrupts for delay
                If(Delay_Counter == 0){
                        Delay_ON = OFF;    // Clear the delay flag for the next interrupt
                        Delay_Counter  = Get_Delay_Counter();      //subroutine to detect user
input
                                        //and set the delay counter variable
                }else{
                        Delay_ON = ON; // leave delay flag ON
                        Return;         //exit the interrupt service routine
                }
        }

        I2C_start();    //I2C protocol start signal subroutine
        I2C_send(Comand_byte);  //I2C protocol – send byte subroutine
        If(bank_counter == bank1){      // bank1 is a constant defined as 0
                I2C_send(Address_pionter_bank1n2);  //Send peripheral register address pointer
                Bank1n2_pattern = ~Bank1n2_pattern; //complement the variable using '~'
```

```
                I2C_send((Bank1n2_pattern | '0000 1111'));  //I2C protocol – send bank status
                                                            //(use OR '|' to set bank 1 on and bank 2 off)
                bank_counter = bank_counter + 1;
                Delay_ON = ON; // turn on the delay flag
        }
*
*
*
```

Exemple de code 3 : Implémentation d'un délai pour l'affichage des motifs lumineux

Pour contrôler la durée de l'intervalle de temps existant entre l'allumage de chaque rangée, le contrôle de l'intensité lumineuse utilise la modulation en largeur d'impulsion PWM *(pulse width modulation)*. La durée de l'interruption Timer0 possède deux valeurs : une pour les longues durées et une pour les courtes durées, toutes deux proportionnelles au pourcentage de PWM fixé à l'aide d'un potentiomètre rotatif ; la languette centrale est connectée à l'entrée du CAN. Ce niveau de résolution est ajustable en sélectionnant les 4 ou 5 bits supérieurs du CAN. La position du compteur Timer0 est ajustée par rapport aux données du CAN, selon une échelle où 100 % de PWM est égal à la position maximum du compteur, complètement allumé, et où le 0 % est le minimum ou le niveau le plus bas d'intensité. L'exemple de code 4 montre la relation existant entre les pourcentages de PWM et les 16 niveaux, et la figure 3 montre le schéma de temporisation pour la durée du signal PWM, t_{PWM_LOW} et t_{PWM_HIGH}.

```
Void get_PWM_ratio(void) {
        Double PotScale;        //local variable to store ADC output scale
        Double PWM_Percentage;     //local variable to store PWM change percentage

        PotScale = (ADRESH)/16 + 1;          //Scale ADC output high byte
        PWM_Percentage = 1/16 * PotScale;//Scale the output from from 0 to 1, equivalent to
        100%
        PWM_High = Frequency_counter * PWM_Percentage;       //set PWM high Timer0 value
        PWM_Low = Frequency_counter * (1 – PWM_Percentage);          //set PWM low
        Timer0 value
                        //PWM_High and PWM_Low are global variables,
                        //and Frequency_counter is a constant Timer0 value to set frequency.
}
```

Exemple de code 4 : Sous-programme pour le calcul des pourcentages de PWM

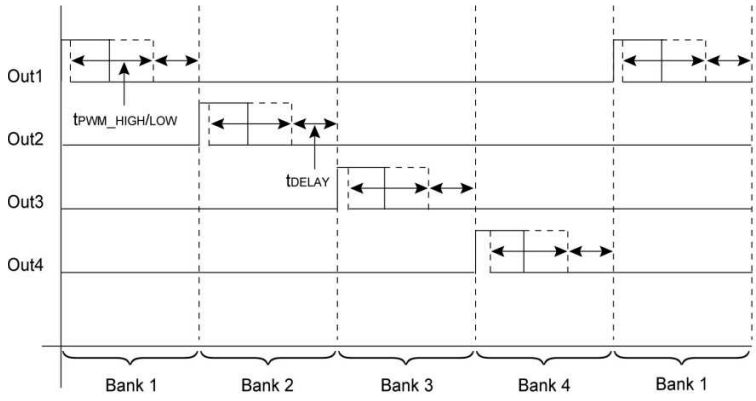


Figure 3 : Schéma de temporisation pour le signal PWM en sortie et la pause

La routine d'interruption doit mettre à jour les positions du compteur Timer0 en prévision des prochaines interruptions. Elle doit également détecter si la durée du signal PWM indiquée

correspond à l’état haut ou bas. C’est pourquoi certaines instructions doivent être ajoutées pour détecter le niveau potentiométrique servant au contrôle de l’intensité et pour convertir proportionnellement la position du compteur Timer0 par rapport aux valeurs `PWM_High` et `PWM_low` tel qu’on peut le voir dans l’exemple de code 4. L’exemple de code 5 montre comment programmer le réglage du PWM, si une déclaration IF THEN est utilisée pour détecter l’état du PWM.

```
        *
        *
        *
        If(bank_counter == bank1){   // bank1 is a constant defined as 0
                If (PWM_High_Low_flag == OFF){   //check PWM status flag
                        get_PWM_ratio();      //Detect user input for PWM ratio
                        Timer0_counter = 65535 – PWM_High;  //set Timer0 counter variable
                                Update_Timer0_Counter(Timer0_counter);       //subroutine to
update Timer0

                        I2C_send(Address_pionter_bank1n2);  //Send peripheral register address
                pointer
                        Bank1n2_pattern = ~Bank1n2_pattern; //complement the variable using
                '~'
                        I2C_send((Bank1n2_pattern | '0000 1111'));  //I2C protocol – send bank
                status
                                                        //(use OR '|' to set bank 1 on and bank 2 off)
                        PWM_High_Low_flag = ON;   //Set flag
                        Delay_ON = OFF;

                }else{
                        Timer0_counter = 65535 – PWM_Low;  //set Timer0 counter variable
                        Update_Timer0_Counter(Timer0_counter);       //subroutine to update
Timer0

                        I2C_send(Address_pionter_bank1n2);  //Send peripheral register address
                pointer
                        I2C_send('1111 1111'));  //I2C protocol – turnoff all LEDs
                        bank_counter = bank_counter + 1;
                        PWM_High_Low_flag = OFF;  //Clear flag
                        Delay_ON = ON;
                }
        }
        *
        *
        *
```

Exemple de code 5 : Lignes de code pour le réglage du pourcentage de PWM

Cette méthode peut aussi être utilisée sur un microcontrôleur 8 bits milieu de gamme avec mémoire programmable supplémentaire tel que ceux de la famille des PIC16F de Microchip. Ce type de microcontrôleur doit permettre au sous-programme principal de gérer les motifs lumineux plus complexes, tels que les cascades lumineuses. Le module Timer1 du

microcontrôleur peut être utilisé pour faire varier la durée quand les deux octets de RAM sont mis à jour pour un motif de cascade.

Les développeurs sont sans cesse en quête de nouveaux moyens de réduire les coûts sans pour autant compromettre les performances du système. Il existe de nombreuses méthodes pour piloter efficacement les rangées de LED d’un écran LCD rétro-éclairé ou des applications à motifs lumineux, et produire une illumination optimale. Sur les applications à faible consommation, les LED peuvent être pilotées en gérant l’intervalle de temps entre chaque rangée de LED. En outre, les microcontrôleurs avec peu de broches associés à des barrettes d'extension d'E/S constituent pour les solutions d’éclairage une alternative économique offrant plus de souplesse de conception.