COMPTEUR DE PASSAGES À GAL

Les composants programmables autorisent une flexibilité et une densité de fonctionnalités surfacique supérieure aux composants logiques traditionnels. Parmi ceux-ci, les microcontrôleurs demandent en général une approche plus «informaticienne» alors que les réseaux logiques programmables PAL, GAL, FPGA sont plus proches de la philosophie de l'électronicien. Cette application de comptage de passages met en œuvre des GAL simples 16V8 avec une étude effectuée sous ABEL, de façon à approfondir notre connaissance et l'utilisation de ce langage (voir ERP n°567).

L'APPLICATION

L'application envisagée est un compteur de passages. Un ensemble de deux capteurs infrarouge perçoit le passage d'une personne et en fonction de son sens de passage incrémente et décrémente un compteur qui décodé affiche le nombre de personnes sur un afficheur 7 segments. Les méthodes pour résoudre un tel projet sont assez nombreuses mais pour des raisons didactiques d'utilisation du langage ABEL, nous choisirons une méthode qui n'est peut-être pas la plus simple mais du moins la plus riche en possibilités.

Comme pour toute création, il nous faut rapidement une évaluation des composants demandés :

- un décodeur BCD -> 7 segments pour l'affichage
- un compteur-décompteur BCD
- une logique de contrôle pour détecter le sens de passage.

On peut rapidement remarquer sur la **figure 1**, le schéma de cette application. On retrouve rapidement les différents éléments constitués chacun d'une GAL.

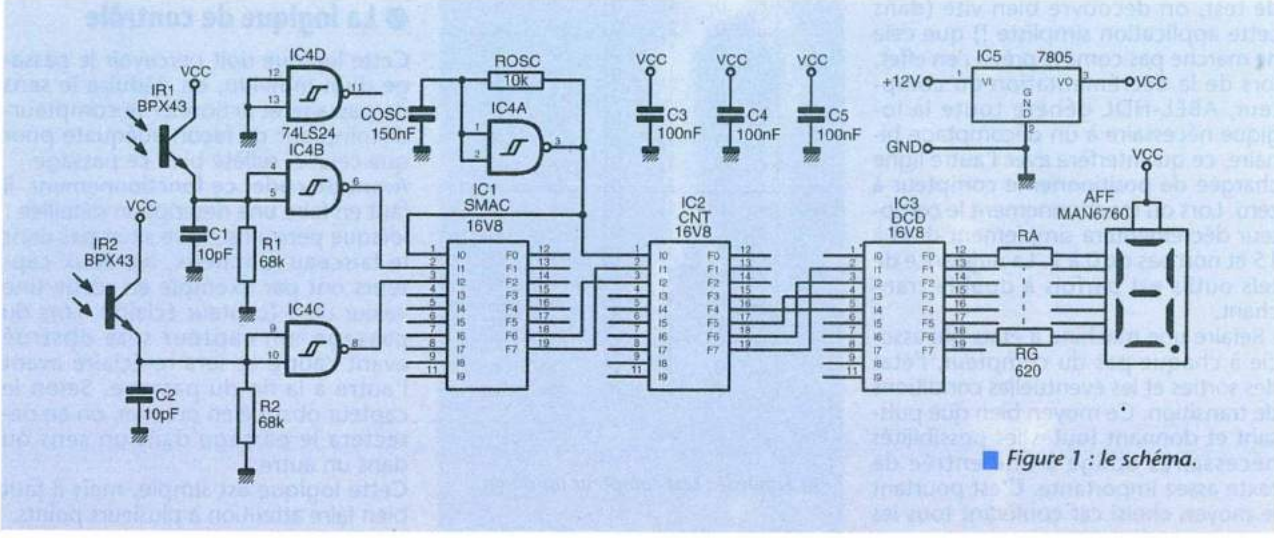


Figure 1 : le schéma.

Le décodeur BCD

Ce sous-ensemble est chargé de prendre un code BCD sur 4 bits et de le transformer en 7 bits pour un affichage 7 segments. Le procédé est donc purement combinatoire, et nous allons donc le coder en ABEL-HDL avec une table de vérité qui à chaque code d'entrée fournit le code 7 segments.
On retrouve sur la **figure 2** le texte d'une telle table de vérité, elle se passe de commentaires !
Notons cependant qu'il est tout a fait possible d'intégrer dans ces équations un terme qui éteindrait l'afficheur pour des zéros non significatifs lors de la mise en série de tels circuits (au cas ou l'on envisagerait de compter par exemple les visiteurs d'un musée au dessus de 9 personnes !).

```
module count
title 'compteur binaire'

        Q3,Q2,Q1,Q0 PIN istype 'reg';
        Clk     PIN ;
        Cnt     =       [Q3,Q2,Q1,Q0];
equations
        Cnt.c=Clk;
        Cnt:=Cnt+1;
end
```

Figure 3

Le compteur-décompteur BCD

Cette application n'est pas simplement combinatoire. Le composant est dans un état déterminé et réagit à des stimuli par le passage vers un autre état. Il s'agit bien évidemment d'une machine et pour coder un fonctionnement de ce type, nous avons plusieurs possibilités :
- ABEL-HDL donne des moyens pour pouvoir intégrer directement des compteurs comme décrit sur la **figure 3**, mais et il y a un mais que seule l'expérience discerne, il ne s'agit généralement d'applications que sur les compteurs binaires. Dans notre cas, le BCD est mal géré lors de la décrémentation. Sur la **figure 4**, on a par exemple une première tentative de créer un tel compteur : la structure de déclarations parait correcte, mais lors de la phase de test, on découvre bien vite (dans cette application simpliste !) que cela ne marche pas comme prévu : en effet, lors de la décrémentation du compteur, ABEL-HDL génère toute la logique nécessaire à un décomptage binaire, ce qui interfère avec l'autre ligne chargée de positionner le compteur à zéro. Lors du fonctionnement le compteur décrémentera simplement de 0 à 15 et non pas de 0 à 9. La puissance de tels outils est parfois à double tranchant.
- Refaire une machine à états qui associe à chaque pas du compteur, l'état des sorties et les éventuelles conditions de transition. Ce moyen bien que puissant et donnant toutes les possibilités nécessaires oblige à une entrée de texte assez importante. C'est pourtant le moyen choisi car conférant tous les éléments pour gérer les transitions difficiles d'un compteur binaire.
Notre compteur sera donc représenté par son état qui sera ses données de sortie. En fonction de 3 signaux d'entrée, on permettra le passage vers d'autres états. On choisit rapidement comme signaux de commande, un signal Reset (non utilisé dans notre application mais présent dans un but purement didactique), un signal Enable permettant de compter et un signal Up/Down spécifiant le sens de comptage (**figure 5**).

```
module segment7
title 'afficheur 7 segments'

        D0,D1,D2,D3     pin     2,3,4,5;
        a,b,c,d,e,f,g   pin     13,12,14,15,16,17,18 istype 'com';

        bcd     =[D3,D2,D1,D0];
        led     =[a,b,c,d,e,f,g];

        ON,OFF  =0,1;

truth_table
(       bcd     ->[     a,      b,      c,      d,      e,      f,      g       ])
        0       ->[     ON,     ON,     ON,     ON,     ON,     ON,     OFF     ];
        1       ->[     OFF,    ON,     ON,     OFF,    OFF,    OFF,    OFF     ];
        2       ->[     ON,     ON,     OFF,    ON,     ON,     OFF,    ON      ];
        3       ->[     ON,     ON,     ON,     ON,     OFF,    OFF,    ON      ];
        4       ->[     OFF,    ON,     ON,     OFF,    OFF,    ON,     ON      ];
        5       ->[     ON,     OFF,    ON,     ON,     OFF,    ON,     ON      ];
        6       ->[     ON,     OFF,    ON,     ON,     ON,     ON,     ON      ];
        7       ->[     ON,     ON,     ON,     OFF,    OFF,    OFF,    OFF     ];
        8       ->[     ON,     ON,     ON,     ON,     ON,     ON,     ON      ];
        9       ->[     ON,     ON,     ON,     ON,     OFF,    ON,     ON      ];
end
```

Figure 2 : table de vérité BCD → 7 segments.

```
module compteur
title 'compteur decimal Up/Down'

        Clk,UD,Ena,Clr      pin 1,2,3,4;
        Q3,Q2,Q1,Q0         pin 16,17,18,19 istype 'reg';

        Cnt     =       [Q3,Q2,Q1,Q0];

        Ck,X,Z,P        = .C. , .X., .Z., .P.;
equations
        Cnt.c = Clk;
        WHEN (Clr==0) & (Ena==1) & (UD==1) THEN Cnt:=Cnt.fb+1;
        WHEN (Clr==0) & (Ena==1) & (UD==0) THEN Cnt:=Cnt.fb-1;
        WHEN (Clr==0) & (Ena==0) THEN Cnt:=Cnt.fb;
        WHEN Clr==1 THEN Cnt:=0;

test_vectors
(       [Clk,   Clr,    Ena,    UD]     ->      Cnt     )
        [.c.,   0,      0,      .x.]    ->      .x.;
        [.c.,   0,      1,      1]      ->      .x.;
        [.c.,   0,      1,      1]      ->      .x.;
        [.c.,   0,      1,      1]      ->      .x.;
        [.c.,   0,      1,      1]      ->      .x.;
        [.c.,   0,      1,      1]      ->      .x.;
        [.c.,   0,      1,      1]      ->      .x.;
        [.c.,   0,      1,      1]      ->      .x.;
        [.c.,   0,      1,      1]      ->      .x.;
        [.c.,   0,      1,      1]      ->      .x.;
        [.c.,   0,      0,      1]      ->      .x.;
        [.c.,   0,      0,      1]      ->      .x.;
        [.c.,   0,      0,      1]      ->      .x.;
        [.c.,   0,      0,      1]      ->      .x.;
        [.c.,   0,      0,      1]      ->      .x.;
        [.c.,   0,      0,      1]      ->      .x.;
        [.c.,   0,      0,      1]      ->      .x.;
        [.c.,   0,      0,      1]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
        [.c.,   0,      1,      0]      ->      .x.;
end
```

Figure 4 : test compteur up-down.

La logique de contrôle

Cette logique doit percevoir le passage d'un individu, en déduire le sens de passage et actionner le compteur-décompteur de façon adéquate pour que celui-ci reflète bien ce passage.
Avant de coder ce fonctionnement, il faut en faire une description détaillée : lorsque personne ne se situe pas dans le faisceau lumineux, les deux capteurs ont par exemple en sortie une valeur de 1 (capteur éclairé). Lors du passage, un capteur sera obstrué avant l'autre et sera re-éclairé avant l'autre à la fin du passage. Selon le capteur obstrué en premier, on en détectera le passage dans un sens ou dans un autre.
Cette logique est simple, mais il faut bien faire attention à plusieurs points.

Si on ne prend en compte qu'un type de configuration pour compter ou décompter le passage, on s'expose à des cas problématiques. En considérant les sorties des capteurs tels que (0,1), cela peut être quelqu'un qui commence à rentrer ou qui finit de sortir.
D'une autre façon, en considérant les séquences suivantes, divers cas particuliers peuvent se présenter :

1er cas	(1,1) ;	2e cas	(1,1) ;
	(0,1) ;		(0,1) ;
	(0,0) ;		(1,1) ;
	(0,1) ;		
	(1,1) ;		

Dans le premier cas, une personne a complètement traversé la barrière de faisceaux, s'est arrêté en plein milieu en obstruant les deux capteurs et ensuite a fait demi-tour.
Dans le deuxième cas, une personne est entrée dans le faisceau du premier capteur et avant d'avoir obstrué le deuxième faisceau a fait demi-tour.
Ceux qui ont une certaine habitude des machines à états auront vu qu'un cas particulier n'a pas été pris en compte. En effet la séquence

(1,1) ;
(0,0) ;

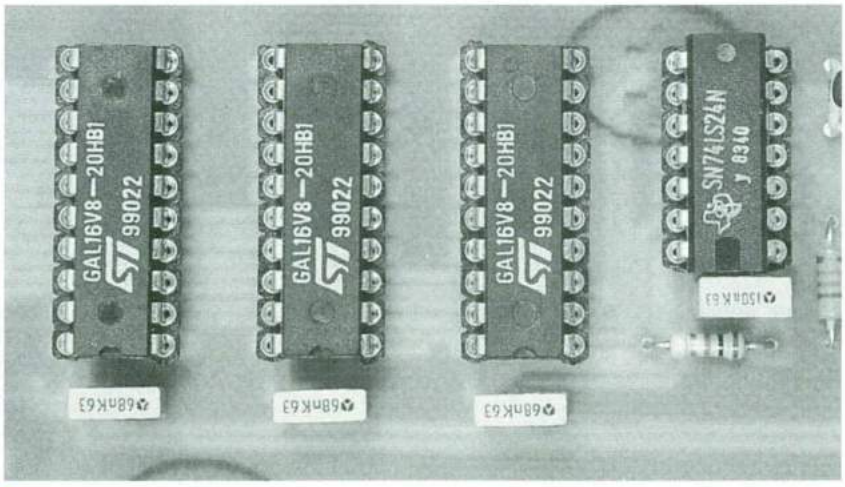
supposerait que quelqu'un traverse les faisceaux en même temps ce qui supposerait soit qu'il tombe, soit qu'il monte du sol ! On peut penser aussi que cela arrive lorsque deux personnes traversent exactement en même temps le faisceau en sens contraire mais là on cherche la petite bête !
Ce cas tout de même serait bien géré, car la partie dans laquelle il se relèverait serait bien comptabilisée. En effet, nous ne déclencherons le comptage ou le décomptage que dans des cas bien précis que nous allons décrire dans l'encadré ci-dessous.
Une fois cette logique établie, il faut l'intégrer dans un composant physique. Pour ce fonctionnement, une machine à états est le type le plus adapté.
On dispose en entrée des deux signaux provenant des capteurs. Pour notre machine à états, on va utiliser un registre pour indiquer le sens de déplacement initial, et il va nous servir ensuite pour spécifier le comptage où le décomptage au compteur. Ensuite 2 registres servent de mémoire aux signaux d'entrée, et servent ainsi à déterminer le sens de déplacement et l'état dans lequel le contrôleur se trouve. Lors de la phase de commande du compteur/décompteur, on fournit un signal Enable. Or celui-ci ne peut être une sortie de registre, car lors du déroulement de la machine à états du contrôleur, il changerait de valeur exactement au même front d'horloge que lors de l'activation du compteur. On utilise donc une possibilité du langage ABEL-HDL qui permet de spécifier des signaux asynchrones (ou plutôt combinatoires) à l'intérieur d'une machine à états. Ce signal Count sera donc positionné avant l'arrivée du front montant du compteur et permettra un fonctionnement optimum (**figure 6**).
Certains diront qu'il aurait été plus simple de relier la patte d'horloge du compteur directement à ce signal. On touche là tout une polémique sur le fait ou non de n'utiliser,que des applications synchrones. Une petite explication est nécessaire pour tous ceux que ce domaine intéresse ou qui seront à plus ou moins long terme destinés à s'en préoccuper.
Les circuits logiques devenant de plus en plus complexes, les méthodes de

```
module compteur
title 'compteur decimal Up/Down'

    Clk,UD,Ena,Clr     pin 1,2,3,4;
    Q3,Q2,Q1,Q0    pin 16,17,18,19 istype 'reg';

    Ck,X,Z,P       = .C. , .X., .Z., .P.;

equations
    [Q3,Q2,Q1,Q0].c = Clk;

state_diagram [Q3,Q2,Q1,Q0]
    State 0:    IF !Clr THEN
                    IF Ena THEN
                        IF UD THEN
                            1
                        ELSE
                            9
                    ELSE
                        0
                ELSE
                    0;

    State 1:    IF !Clr THEN
                    IF Ena THEN
                        IF UD THEN
                            2
                        ELSE
                            0
                    ELSE
                        1
                ELSE
                    0;

    State 2:    IF !Clr THEN
                    IF Ena THEN
                        IF UD THEN
                            3
                        ELSE
                            1
                    ELSE
                        2
                ELSE
                    0;

    State 3:    IF !Clr THEN
                    IF Ena THEN
                        IF UD THEN
                            4
                        ELSE
                            2
                    ELSE
                        3
                ELSE
                    0;

    State 4:    IF !Clr THEN
                    IF Ena THEN
                        IF UD THEN
                            5
                        ELSE
                            3
                    ELSE
                        4
                ELSE
                    0;

    State 5:    IF !Clr THEN
                    IF Ena THEN
                        IF UD THEN
                            6
                        ELSE
                            4
                    ELSE
                        5
                ELSE
                    0;

    State 6:    IF !Clr THEN
                    IF Ena THEN
                        IF UD THEN
                            7
                        ELSE
                            5
                    ELSE
                        6
                ELSE
                    0;

    State 7:    IF !Clr THEN
                    IF Ena THEN
                        IF UD THEN
                            8
                        ELSE
                            6
                    ELSE
                        7
                ELSE
                    0;

    State 8:    IF !Clr THEN
                    IF Ena THEN
                        IF UD THEN
                            9
                        ELSE
                            7
                    ELSE
                        8
                ELSE
                    0;

    State 9:
                IF !Clr THEN
                    IF Ena THEN
                        IF UD THEN
                            0
                        ELSE
                            8
                    ELSE
                        9
                ELSE
                    0;

«Ensure return from illegal state
    State 10:   GOTO 0;
    State 11:   GOTO 0;
    State 12:   GOTO 0;
    State 13:   GOTO 0;
    State 14:   GOTO 0;
    State 15:   GOTO 0;

end
```

Figure 5 : le compteur retenu.

(1,1)	ou	(1,1)
on a une rentrée		on a une sortie
(1,0)		(0,1)
(0,0)		(0,0)
(0,1)		(1,0)
si rentrée incrémentation du compteur		si sortie décrémentation du compteur
(1,1)		(1,1)

Les conditions de comptage et décomptage.



Les trois GAL.

conception et de test ont impliqué des outils tels que des simulateurs, des compilateurs pour des composants de plus en plus intégrés comme des FPGAs. Malheureusement et comme tout n'est jamais parfait, il faut parfois procéder à des séances de test plus ou moins complexes et longues. Les fréquences utilisées réservent alors de nombreuses surprises sur le fonctionnement d'un montage à priori simple. Considérons simplement le montage de la **figure 7** : il s'agit simplement d'un compteur binaire à 3 étages. Or il y a deux possibilités pour produire un tel montage, la **figure 8** et la **figure 9**. Si on considère que le temps de propagation des circuits combinatoires (tcom) et des circuits à registre (treg) est le même pour les deux types de

```
module cntPas
title 'controleur du compteur de passage'

    S0,S1       pin 17,16  istype 'reg';
    UD          pin 19 istype 'reg';
    Count       pin 18 istype 'com';
    Clk,LedIn,LedOut        pin 1,2,3;

    Leds    =    [LedIn,LedOut];
    Entre   =    [1,0];
    Sort    =    [0,1];
    Milieu  =    [1,1];
    Personne =   [0,0];

    S       =    [UD,S1,S0];
    WaitIn  =    0;
    In0     =    1;
    In1     =    2;
    In2     =    3;
    WaitOut     =    4;
    Out0    =    5;
    Out1    =    6;
    Out2    =    7;

equations
    S.c=Clk;
state_diagram S
    STATE WaitIn:
        CASE
            Leds==Entre:    In0 WITH Count=0; ENDWITH;
            Leds==Sort:     Out0 WITH Count=0; ENDWITH;
            Leds==Personne: WaitIn WITH Count=0; ENDWITH;
            Leds==Milieu:   WaitIn WITH Count=0; ENDWITH;
        ENDCASE

    STATE In0:
        CASE
            Leds==Entre:    In0 WITH Count=0; ENDWITH;
            Leds==Sort:     WaitIn WITH Count=0; ENDWITH;
            Leds==Personne: WaitIn WITH Count=0; ENDWITH;
            Leds==Milieu:   In1 WITH Count=0; ENDWITH;
        ENDCASE

    STATE In1:
        CASE
            Leds==Entre:    In0 WITH Count=0; ENDWITH;
            Leds==Sort:     In2 WITH Count=0; ENDWITH;
            Leds==Personne: WaitIn WITH Count=0; ENDWITH;
            Leds==Milieu:   In1 WITH Count=0; ENDWITH;
        ENDCASE

    STATE In2:
        CASE
            Leds==Entre:    WaitIn WITH Count=0; ENDWITH;
            Leds==Sort:     In2 WITH Count=0; ENDWITH;
            Leds==Personne: WaitIn WITH Count=1; ENDWITH;
            Leds==Milieu:   In1 WITH Count=0; ENDWITH;
        ENDCASE

    STATE WaitOut:
        CASE
            Leds==Entre:    In0 WITH Count=0; ENDWITH;
            Leds==Sort:     Out0 WITH Count=0; ENDWITH;
            Leds==Personne: WaitOut WITH Count=0; ENDWITH;
            Leds==Milieu:   WaitOut WITH Count=0; ENDWITH;
        ENDCASE

    STATE Out0:
        CASE
            Leds==Entre:    WaitOut WITH Count=0; ENDWITH;
            Leds==Sort:     Out0 WITH Count=0; ENDWITH;
            Leds==Personne: WaitOut WITH Count=0; ENDWITH;
            Leds==Milieu:   Out1 WITH Count=0; ENDWITH;
        ENDCASE

    STATE Out1:
        CASE
            Leds==Entre:    Out2 WITH Count=0; ENDWITH;
            Leds==Sort:     Out0 WITH Count=0; ENDWITH;
            Leds==Personne: WaitOut WITH Count=0; ENDWITH;
            Leds==Milieu:   Out1 WITH Count=0; ENDWITH;
        ENDCASE

    STATE Out2:
        CASE
            Leds==Entre:    Out2 WITH Count=0; ENDWITH;
            Leds==Sort:     WaitOut WITH Count=0; ENDWITH;
            Leds==Personne: WaitOut WITH Count=1; ENDWITH;
            Leds==Milieu:   Out1 WITH Count=0; ENDWITH;
        ENDCASE

test_vectors
(   [Clk,  LedIn,  LedOut] ->    [S,    Count]      )
    [.c.,  0,   0]   ->   [WaitIn,  .x.   ];
    [.c.,  0,   0]   ->   [WaitIn,  .x.   ];
    [.c.,  1,   0]   ->   [In0,  .x.   ];
    [.c.,  1,   1]   ->   [In1,  .x.   ];
    [.c.,  0,   1]   ->   [In2,  .x.   ];
    [.c.,  1,   1]   ->   [In1,  .x.   ];
    [.c.,  1,   0]   ->   [In0,  .x.   ];
    [.c.,  0,   0]   ->   [WaitIn,  .x.   ];
    [.c.,  1,   0]   ->   [In0,  .x.   ];
    [.c.,  1,   1]   ->   [In1,  .x.   ];
    [.c.,  0,   1]   ->   [In2,  .x.   ];
    [.c.,  0,   0]   ->   [WaitIn,  .x.   ];
    [.c.,  0,   0]   ->   [WaitIn,  .x.   ];
    [.c.,  0,   1]   ->   [Out0,  .x.   ];
    [.c.,  1,   1]   ->   [Out1,  .x.   ];
    [.c.,  1,   0]   ->   [Out2,  .x.   ];
    [.c.,  1,   1]   ->   [Out1,  .x.   ];
    [.c.,  0,   1]   ->   [Out0,  .x.   ];
    [.c.,  0,   0]   ->   [WaitOut,  .x.   ];
    [.c.,  0,   1]   ->   [Out0,  .x.   ];
    [.c.,  1,   1]   ->   [Out1,  .x.   ];
    [.c.,  1,   0]   ->   [Out2,  .x.   ];
    [.c.,  0,   0]   ->   [WaitOut,  .x.   ];

end
```

Figure 6 : contrôleur de passage.

réalisations, les deux circuits ont un comportement assez différent lorsque la fréquence s'élève.
Dans la figure 8, le compteur fonctionnera correctement tant que tfreq>tcom+treg, tandis que le circuit de la figure 9 ne fonctionnera que si tfreq>(tcom+treg)xnombre d'étages car le temps de propagation entre les différents étages sera suffisamment long pour que le premier registre change alors que le dernier registre du précédent comptage ne se soit pas encore positionné.
Cet exemple est certes trivial, mais il le devient beaucoup moins dès que le nombre de composants augmente un tant soit peu et il faut alors étudier très sérieusement les différents signaux d'arrivée et de sortie de ces portions asynchrones. Tous les chemins doivent être pris en compte, ce qui devient vite assez décourageant vu la somme de travail demandé.
Les machines à états synchrones sont elles beaucoup plus faciles à simuler et à certifier. Tant qu'on est sous la fréquence limite du moins bon composant, le montage a de bonnes chances de fonctionner. Ce type de montages comme nous l'avons utilisé ici, bien que demandant parfois plus de composants est le plus sur moyen de réussir du premier coup son application.
Le produit de DATA-IO est aussi livré avec un simulateur, qui nous permet de vérifier rapidement le bon fonctionnement de notre contrôleur.

LE CIRCUIT D'INTERFACE INFRAROUGE

Notre application utilise un ensemble de deux faisceaux infrarouges pour détecter le passage d'une personne.
L'émetteur infrarouge est simplement constitué d'une diode infrarouge.
Les récepteurs sont quant à eux constitués de phototransistors. Un étage d'adaptation à l'aide d'une porte trigger de schmitt permet de mettre en forme ces divers signaux et de les adapter à un niveau logique. Une porte trigger est elle-même utilisée pour fournir l'horloge aux différentes GALs.

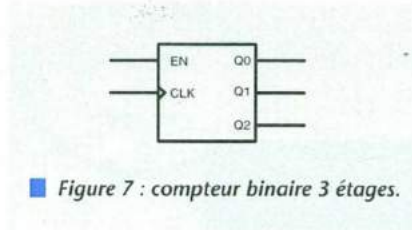


Figure 7 : compteur binaire 3 étages.

LA RÉALISATION

Sur la **figure 10**, vous disposez du tracé du circuit imprimé de ce compteur de passages. Il est sur une seule face et bénéficie de la programmabilité des composants utilisés : on peut faire des connexions toutes droites, ce qui est un plaisir toujours apprécié par l'électronicien. Le routage se fait directement à l'intérieur de la GAL par la déclaration des pattes d'entrées-sorties. Les fichiers de programmation JEDEC sont disponibles sur le serveur.

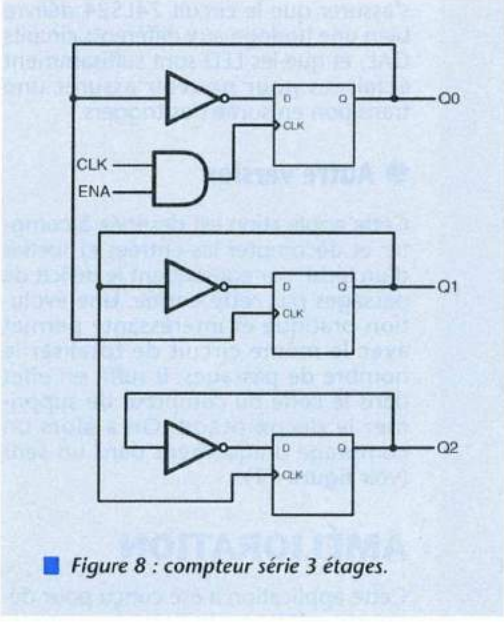


Figure 8 : compteur série 3 étages.

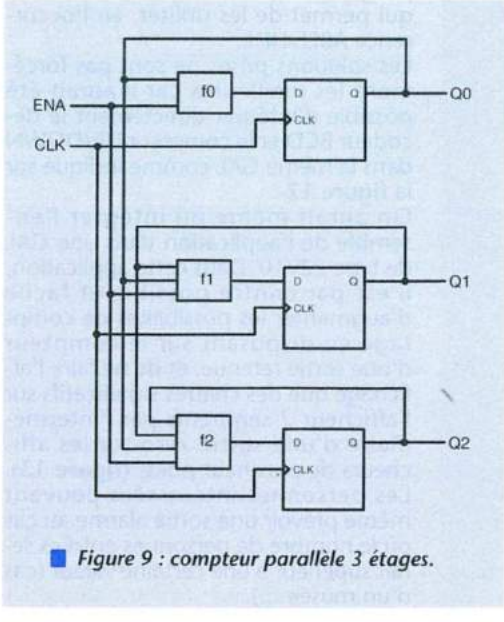


Figure 9 : compteur parallèle 3 étages.

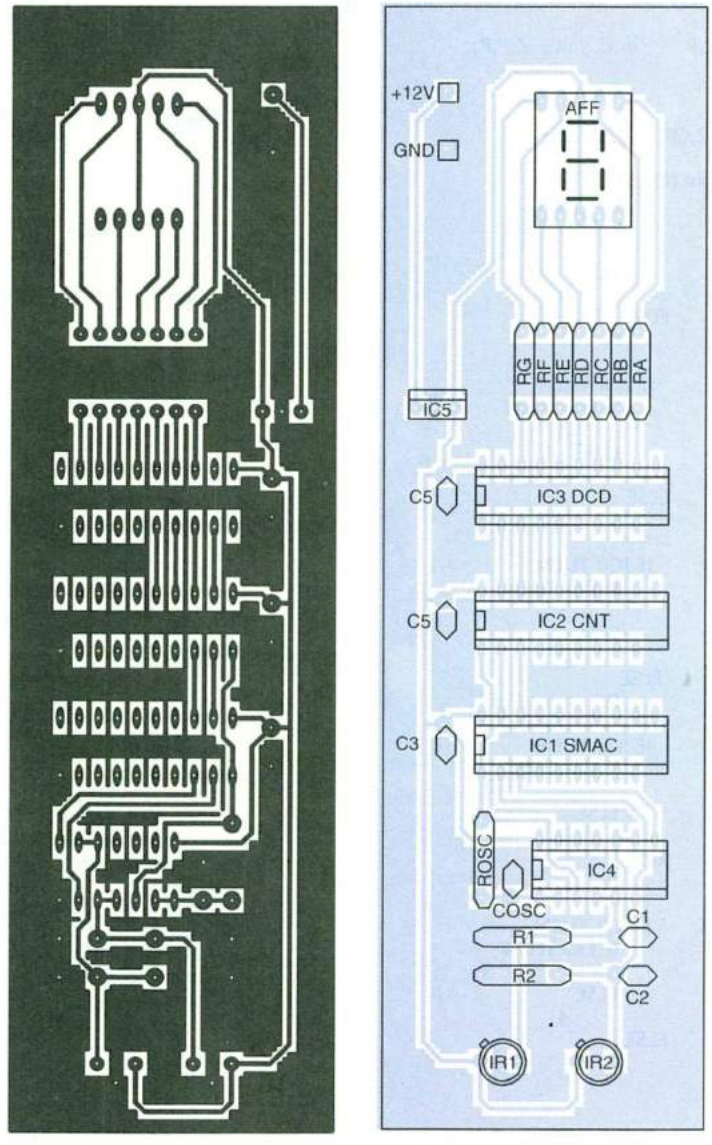
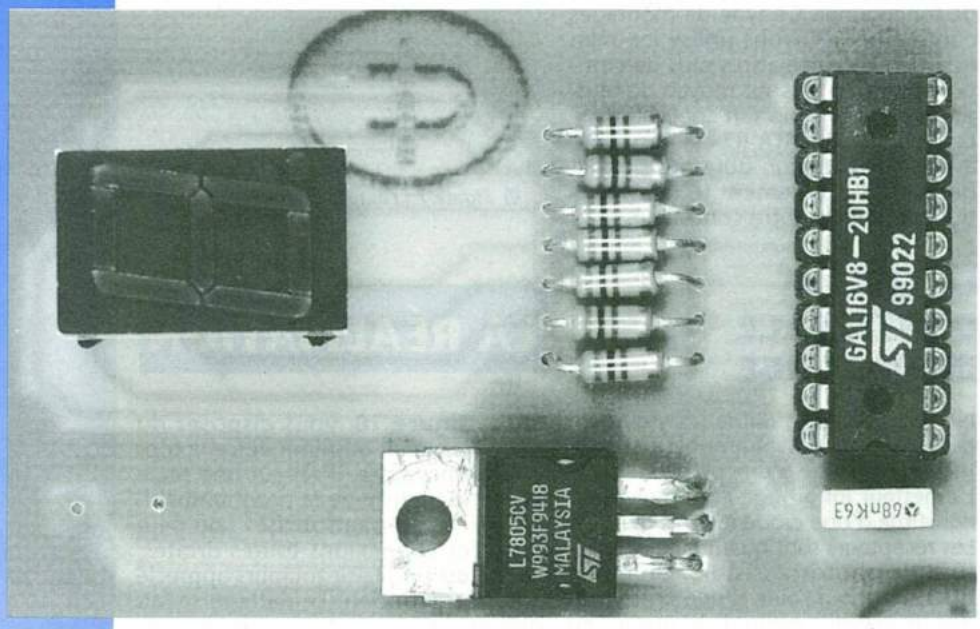


Figure 10 : circuit imprimé et implantation du compteur de passages.



NOMENCLATURE

Résistances :

RA, RF : 620 Ω
ROSC : 10 kΩ
R1, R2 : 68 kΩ

Condensateurs :

COSC : 150 nF
C1, C2 : 10 pf
C3, C4, C5 : 100 nF

Circuits intégrés :

IC1, IC2, IC3 : GAL 16V8
IC4 : 74LS24
IC5 : 7805
IR1, IR2 : BPX43
Afficheur : MAN6760 ou équivalent

```
module compteur
title 'compteur decimal Up'

    Clk,Ena,Clr      pin 1,3,4;
    Q3,Q2,Q1,Q0      pin 16,17,18,19 istype 'reg';

    Ck,X,Z,P         = .C. , .X., .Z., .P.;


equations
    [Q3,Q2,Q1,Q0].c = Clk;

state_diagram [Q3,Q2,Q1,Q0]
    State 0:    IF !Clr THEN
                    IF Ena THEN
                        1
                    ELSE
                        0
                ELSE
                    0;

    State 1:    IF !Clr THEN
                    IF Ena THEN
                        2
                    ELSE
                        1
                ELSE
                    0;

    State 2:    IF !Clr THEN
                    IF Ena THEN
                        3
                    ELSE
                        2
                ELSE
                    0;

    State 3:    IF !Clr THEN
                    IF Ena THEN
                        4
                    ELSE
                        3
                ELSE
                    0;

    State 4:    IF !Clr THEN
                    IF Ena THEN
                        5
                    ELSE
                        4
                ELSE
                    0;

    State 5:    IF !Clr THEN
                    IF Ena THEN
                        6
                    ELSE
                        5
                ELSE
                    0;

    State 6:    IF !Clr THEN
                    IF Ena THEN
                        7
                    ELSE
                        6
                ELSE
                    0;

    State 7:    IF !Clr THEN
                    IF Ena THEN
                        8
                    ELSE
                        7
                ELSE
                    0;

    State 8:    IF !Clr THEN
                    IF Ena THEN
                        9
                    ELSE
                        8
                ELSE
                    0;

    State 9:
                IF !Clr THEN
                    IF Ena THEN
                        0
                    ELSE
                        9
                ELSE
                    0;

«Ensure return from illegal state
    State 10:   GOTO 0;
    State 11:   GOTO 0;
    State 12:   GOTO 0;
    State 13:   GOTO 0;
    State 14:   GOTO 0;
    State 15:   GOTO 0;

end
```

Figure 11 : compteur totalisateur.

Le test

Le test est immédiat pourvu que les connexions soient correctes. On peut s'assurer que le circuit 74LS24 délivre bien une horloge aux différents circuits GAL, et que les LED sont suffisamment éclairées pour pouvoir assurer une transition en sortie des triggers.

Autre version

Cette application est destinée à compter et décompter les entrées et sorties d'un local, par conséquent le déficit de passages par cette entrée. Une évolution pratique et intéressante permet avec le même circuit de totaliser le nombre de passages. Il suffit en effet dans le code du compteur de supprimer le décomptage. On a alors un comptage uniquement dans un sens (voir **figure 11**).

AMÉLIORATION

Cette application a été conçu pour démontrer les possibilités des circuits programmables et surtout de l'outil qui permet de les utiliser, en l'occurrence ABEL-HDL.
Les solutions prises ne sont pas forcément les meilleures car il aurait été possible d'intégrer directement le décodeur BCD et le compteur UP/DOWN dans la même GAL comme indiqué sur la **figure 12**.
On aurait même pu intégrer l'ensemble de l'application dans une GAL de type 22V10. Dans cette application, il est par contre possible et facile d'augmenter les possibilités de comptage en disposant sur le compteur d'une sortie retenue, et de ne faire l'affichage que des chiffres significatifs sur l'afficheur 7 segments par l'intermédiaire d'une sortie zéro sur les afficheurs de plus haut poids (**figure 13**).
Les personnes intéressées peuvent même prévoir une sortie alarme au cas où le nombre de personnes entrées serait supérieur à une certaine valeur (cas d'un musée ...).

```
module cnt7seg
title 'compteur up/down 7 segments'
        a,b,c,d,e,f,g      pin
19,18,17,16,15,14,13 istype 'reg';
    clk,ena,ud        pin 1,3,2;

    S     =[a,b,c,d,e,f,g];

    S0    =[0,0,0,0,0,0,0];
    S1    =[0,0,1,0,1,0,0];
    S2    =[0,0,0,0,0,1,0];
    S3    =[0,0,0,1,0,0,0];
    S4    =[0,0,0,0,0,1,0];
    S5    =[0,0,1,0,0,0,0];
    S6    =[0,0,0,0,1,0,0];
    S7    =[0,0,0,0,0,0,1];
    S8    =[0,1,0,0,1,0,0];
    S9    =[0,0,0,0,0,1,0];

equations
    S.c=clk;

state_diagram S;
    state S0:
        IF ena THEN
            IF ud THEN
                S1
            ELSE
                S9;
        ELSE
            S0;
    state S1:
        IF ena THEN
            IF ud THEN
                S2
            ELSE
                S0;
        ELSE
            S1;
    state S2:
        IF ena THEN
            IF ud THEN
                S3
            ELSE
                S1;
        ELSE
            S2;
    state S3:
        IF ena THEN
            IF ud THEN
                S4
            ELSE
                S2;
        ELSE
            S3;
    state S4:
        IF ena THEN
            IF ud THEN
                S5
            ELSE
                S3;
        ELSE
            S4;
    state S5:
        IF ena THEN
            IF ud THEN
                S6
            ELSE
                S4;
        ELSE
            S5;
    state S6:
        IF ena THEN
            IF ud THEN
                S7
            ELSE
                S5;
        ELSE
            S6;
    state S7:
        IF ena THEN
            IF ud THEN
                S8
            ELSE
                S6;
        ELSE
            S7;
    state S8:
        IF ena THEN
            IF ud THEN
                S9
            ELSE
                S7;
        ELSE
            S8;
    state S9:
        IF ena THEN
            IF ud THEN
                S0
            ELSE
                S8;
        ELSE
            S9;
end
```

Figure 12 : compteur-décompteur 7 segments.

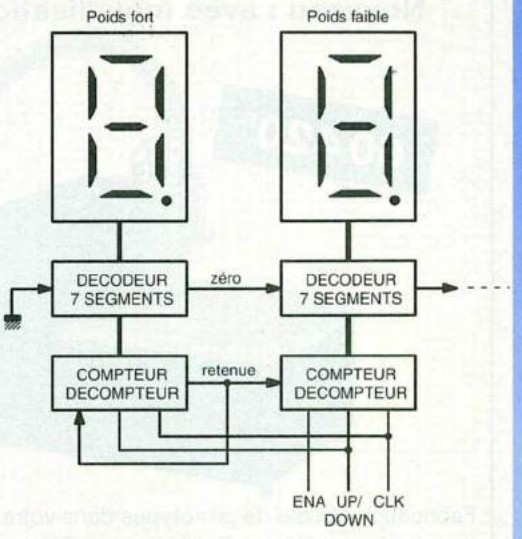


Figure 13 : évolution à n chiffres.

CONCLUSION

L'application présentée ici se veut essentiellement didactique. Nous avons abordé les problèmes posés par un ensemble sur les composants logiques et leurs applications, ceci par l'intermédiaire d'un langage de description. Ce concept permet une flexibilité importante et qui, pour autant qu'on se donne le temps de le maîtriser, donne à l'électronicien même amateur des possibilités énormes.

De plus, le langage ABEL-HDL est proche de la logique électronique avec ses concepts de machines à états, équations logiques, ce qui ne force pas l'utilisateur à devenir un informaticien chevronné.

P. DE CARVALHO

Digimétrie

Instrumentation Informatique et Communication pour l'Industrie et le Laboratoire

MATERIEL pour PC- bus ISA XT/AT et PC/104

Réseaux industriels

_ **AT-FullFIP2 :** Interface pour réseau FIP (nouvelle version fullfip2-64 MHz).
_ **PC-ACOM 485/ISO :** Interface pour liaison série 485 avec isolation en option.

Traitement du signal (basé sur le**DSP** 56001 de Motorola)

_ **PC-DSP 56K :** Carte processeur de signal DSP (**D**igital **S**ignal **P**rocessing).
_ **PC-DSP 56K - AD /-ADC16 :** DSP + AD / DA 14 bits ou 2 AD 16 bits. 44K éch. / sec.
_ **PC-DSP 56K - ISO/ -ST :** DSP + Port isolé RS 485 ou 2 AD/ DA 16bits-100KHz + MIDI

Acquisition de données

_ **AT-LAB12B /R /D :** ADC 12Bits, 16Voies, 100 ou (/R) 800K échantillons /s.Gains , DMA et IRQ prog. + E / S TTL + Timer + (/D) 2 voies DAC.
_ **AT-LAB16B /D :** ADC 16Bits,100K échantillons /s.Identique à AT-LAB12B.
_ **PC-LAB:** ADC 12Bits, 16Voies, 4Gains prog. + E / S TTL + Timer + 2 DAC.
_ **PC-ADC 12B 16V 4G :** ADC 12Bits, 16Voies, 4Gains prog. + E / S TTL + Timer
_ **PC-ADC 12B 8V /D :** ADC 12Bits 8Voies+ E / S TTL + Timer + (/D) 2 voies DAC.
_ **Micro-LAB/D:** ADC 12Bits 8Voies+ E / S TTL + Timer + (/D) 1 voie DAC . bas coût.
_ **PC-DAC 12B 4V/2V :** Convertisseur DA 12Bits 4 ou 2 Voies + E / S TTL + Timer.
_ **EX-MUX 32V/ISO :** Carte externe d'extension 32 à 256 voies AD avec isolation en option.
_ **EX-RELAIS16:** Carte externe d'extension 16 relais 400Volts - 6Amp.
_ **PC-COMPT :** 3 décompteurs 24 bits + 2 timers 8254 + Zone de wrapping.
_ **PC-ROPTO :** 16 relais REED, 32 optocoupleurs + 24 E / S TTL + Timer.
_ **PC-OCTOTIM :** 8 Timers 8254 + Zone de wrapping .
_ **PC-IOT :** 96 E / S TTL bufferisées + Timer + Zone de wrapping.
_ **PC-PIT :** 48 E / S TTL + Timer + Zone de wrapping.
_ **PC-TOR48 :** 48 E/S TTL compatible OPTO22 + Timer.
_ **PC-TIM :** 6 décompteurs 16 bits entièrement programmable (CLK,GATE,OUT).
_ **PC104-AD12 /-AD16 :** Carte format PC/104,ADC 12 ou 16Bits,100K échantillons /s.
_ **PC104 -IO48:** Carte format PC/104, 48 E/S TTL compatible opto 22.

LOGICIEL pour PC-DOS et Windows

Bibliothèques Réseau

_ **DigiTools - FIP :** Configurateur de réseau FIP pour notre carte FIP
_ **DigiTools - JBus :** Gestion de notre carte série sous protocole J-Bus

Bibliothèques d'acquisition en C ou Pascal

_ **DigiTools - DSP :** Routines d' E / S et de traitement du signal pour nos cartes **DSP**
_ **DigiTools - I / O :** Routines d' Entrée / Sortie pour nos cartes d' Entrée / Sortie
_ **DigiTools - Motor :** Routines de contrôle d'axes pour notre carte PC-Motor

Logiciels d'acquisition de données

_ **DigiView - DSP :** Filtrage et analyse spectrale pour nos cartes **DSP**
_ **DigiView - I / O :** Acquisition de données pour nos cartes d' Entrée / Sortie
_ **DigiView - Icon :** Générateur d'applications guidé par icônes

Paquets applicatifs

_ **Vocalix :** Matériel et logiciel pour traitement vocal complet
_ **Neuronix :** Expert Neuronal pour l' industrie

DIGIMETRIE
30, rue E. Renan
66000 Perpignan
- FRANCE - U.E -
Tél : (33) 68.66.54.48
Fax : (33) 68.50.27.85

Nom:
Société:
Adresse:

Catalogue gratuit sur demande

ERP 6/95
rdp-03/94