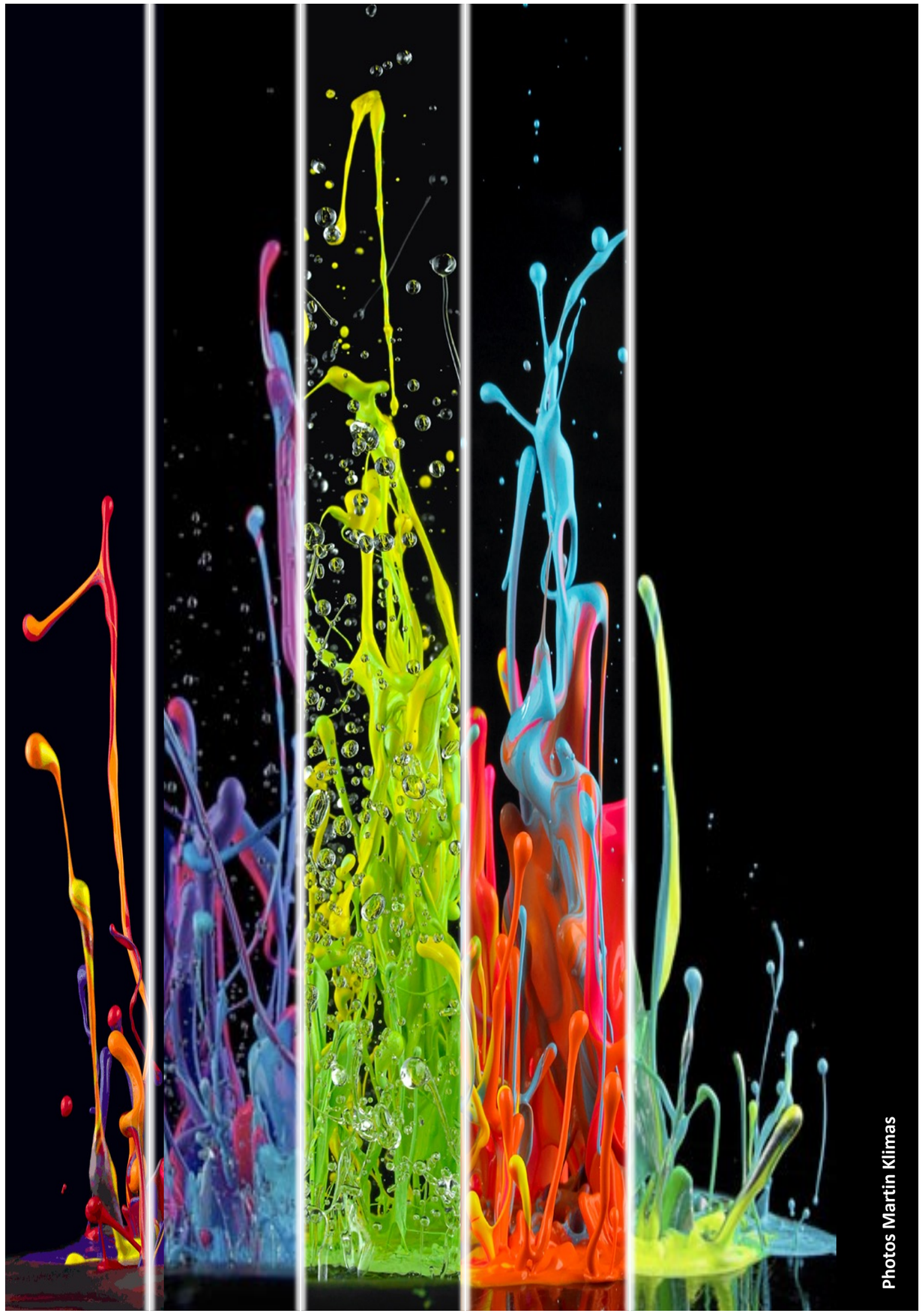


ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Programmation

Révision

LabVIEW & Matlab

série & corrigé

ME 2e semestre

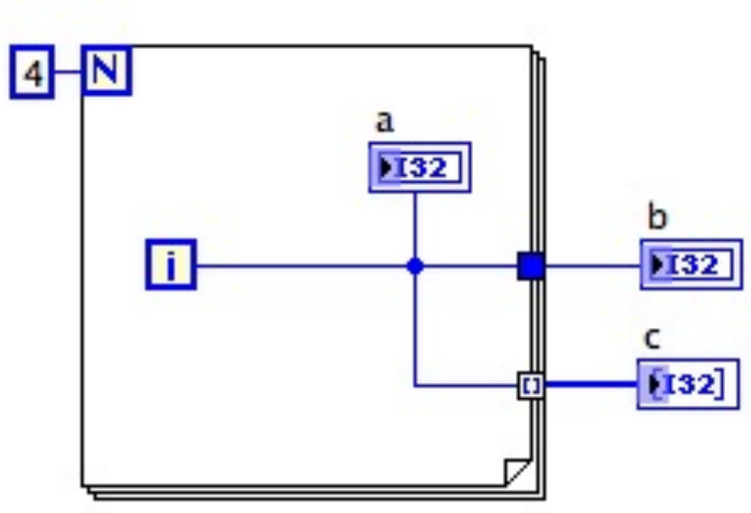
rev. 2022.1

Christophe Salzmann

Laboratoire d'Automatique

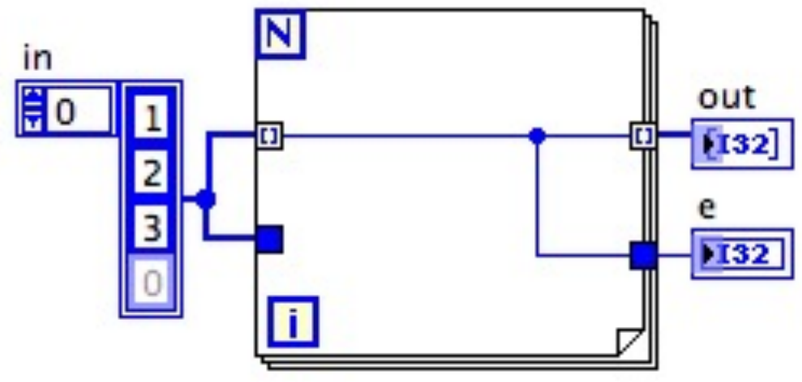
Rappel auto-indexing

- nombre d'itérations (constant)
- indice de boucle, de 0 à N-1
- *auto indexing,* les valeurs sont mémorisées dans un *array* et retournées **à la fin**
- pas d'*indexing*, uniquement la dernière valeur est disponible **à la fin**



- Dans le front panel, est-ce que a,b,c changent au cours de l'execution du VI?

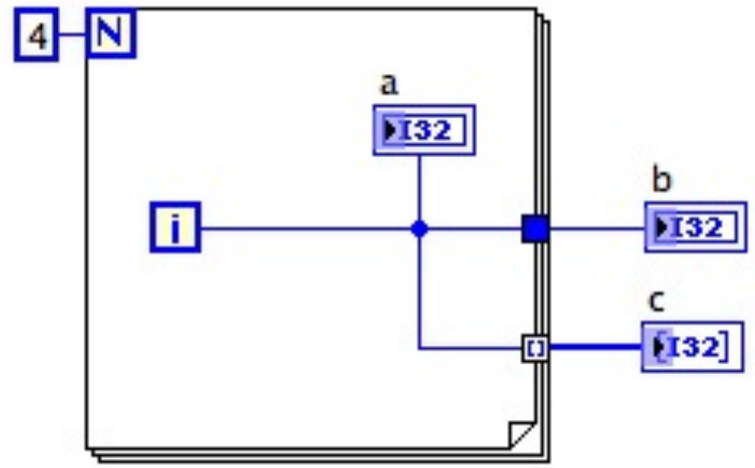
- a,b,c *final* = ?
- a=b ? pourquoi?
- N=?



- N=? Pourquoi?
- out=?
- e=?

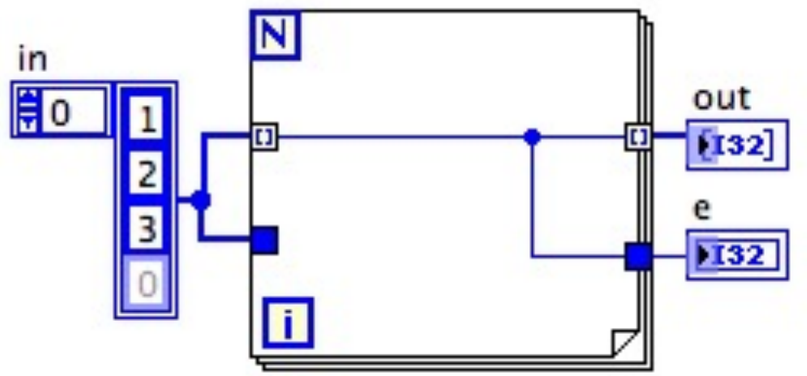
Rappel auto-indexing

- nombre d'itérations (constant)
- indice de boucle, de 0 à N-1
- *auto indexing,* les valeurs sont mémorisées dans un *array* et retournées **à la fin**
- pas d'*indexing*, uniquement la dernière valeur est disponible **à la fin**



- Dans le front panel, est-ce que a,b,c changent au cours de l'execution du VI?
  Seulement a change car il se trouve à l'intérieur de la boucle for.

- a,b,c *final*, la boucle est éxécutée 4x, i de 0 à 3, a prend les valeurs 0,1,2,3, a final=3, b=3, c=[0,1,2,3]
- a=b -> vrai, car relié au même fil
- N= 4, toujours

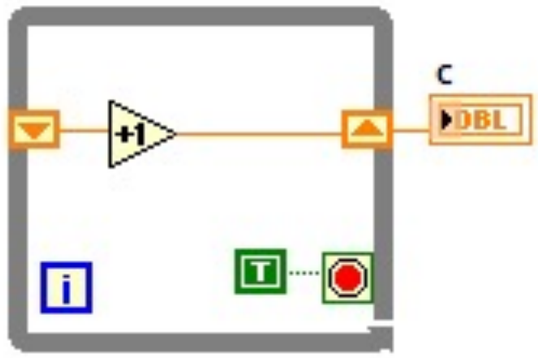


- N= 3, car il y a 3 éléments dans le vecteur *in*, N vaut toujours 3
- out=auto-indexing actif, out = in = [1,2,3]
- e= pas d'auto-indexing, vaut la dernier valeur du tableau *in* -> 3

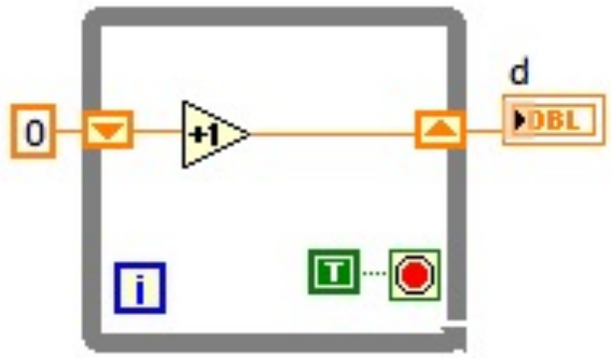
Rappel shift registers

shift register = variable locale temporelle, mémorise la dernière valeur entrée

- *mémorise la valeur courant qui sera disponible dans* *à la prochaine itération*
- recupère la valeur entrée lors de l'exécussion précédente (k-1)
- initialisation, défini la valeur de la variable pour le temps k
- recupère les valeurs entrées lors des exécussions précédentes (k-1,k-2,k-3)



Après 3 exécutions du VI
- c =?

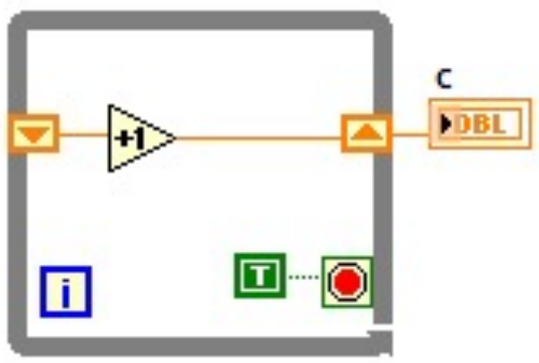


Après 3 exécutions du VI
- d =?

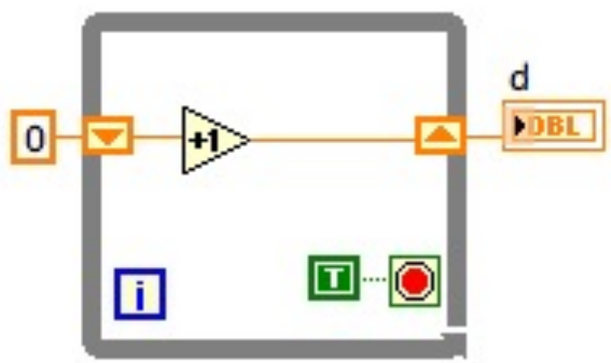
Rappel shift registers

shift register = variable locale temporelle, mémorise la dernière valeur entrée

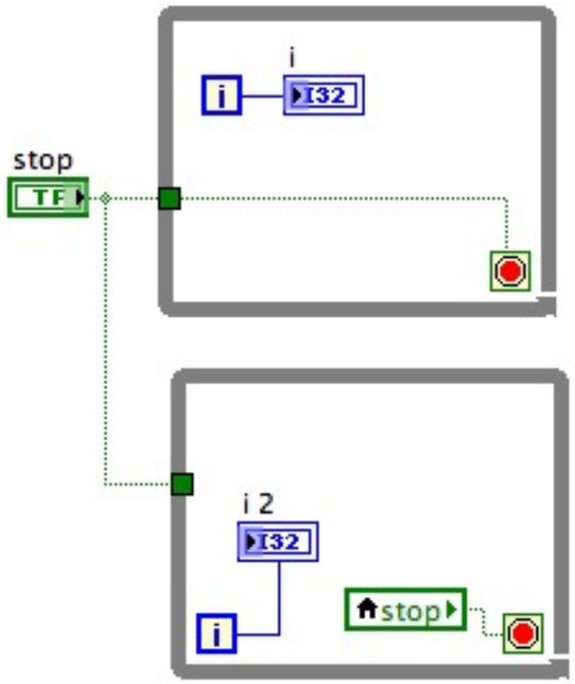
- *mémorise la valeur courant qui sera disponible dans* *à la prochaine itération*
- recupère la valeur entrée lors de l'exécussion précédente (k-1)
- initialisation, défini la valeur de la variable pour le temps k
- recupère les valeurs entrées lors des exécussions précédentes (k-1,k-2,k-3)



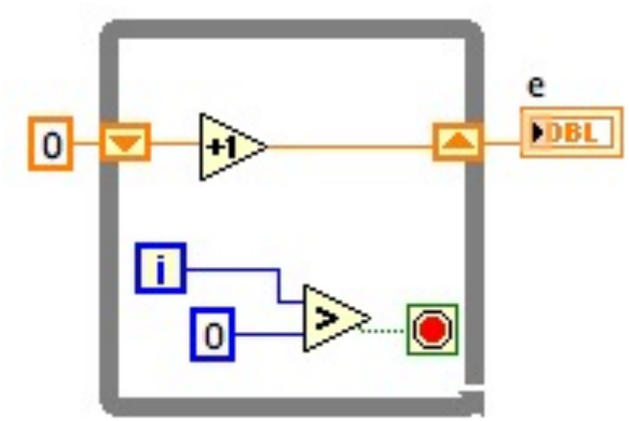
Après 3 exécutions du VI
Il n'y a pas de valeur initiale pour le shift register, donc la 1ere exécution, il vaut 0, après la 1ere exécution nous aurons c=0+1=1.
À la 2e execution, le shift register vaut la dernière valeur entrée, i.e. 1 -> c=1+1=2
À la 3e execution, le shift register vaut la dernière valeur entrée, i.e. 2 -> c=2+1=3
Note: la boucle itère 1 seule fois, true étant connecté sur la condition d'arrêt.



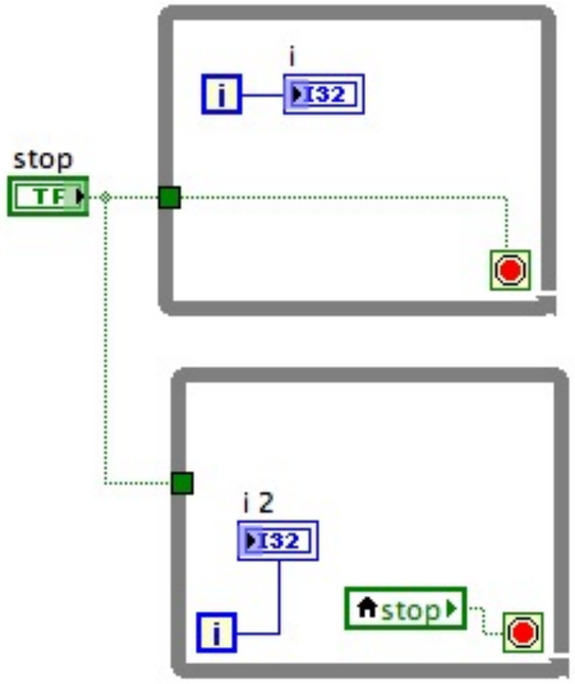
Après 3 exécutions du VI
Il y a une valeur initiale pour le shift register, donc à chaque exécution le shift register sera remis à 0.
Après la 1ere exécution nous aurons c=0+1=1.
À la 2e execution, le shift register est ré-initialisé -> c=0+1=1
À la 3e execution,idem 2e, c=0+1=1
Note: la boucle itère 1 seule fois, true étant connecté sur la condition d'arrêt.



Est-ce que la *while loop* du haut va s'arrêter?
Est-ce que la *while loop* du bas va s'arrêter?

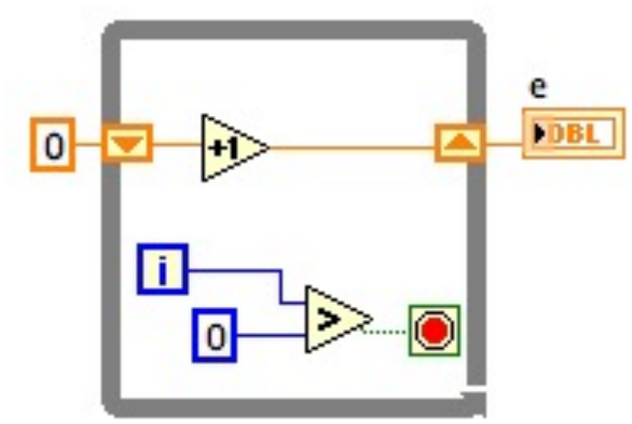


Combien de fois la boucle while est-elle éxécutée?
e=?

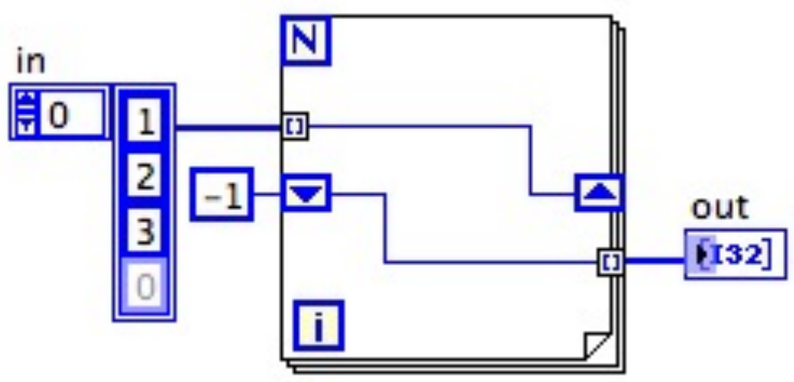


Est-ce que la *while loop* du haut va s'arrêter?
Oui, si le bouton "stop" est à true avant d'entrer dans la boucle.
Non, dans le cas contraire, c'est une boucle infinie, le bouton stop n'est pas à l'intérieur de la boucle et donc pas "visible"

Est-ce que la *while loop* du bas va s'arrêter?
Oui, dans tout les cas, la variable locale à l'intérieur de la boucle à un effet similaire à avoir le bouton à l'intérieur de celle-ci.

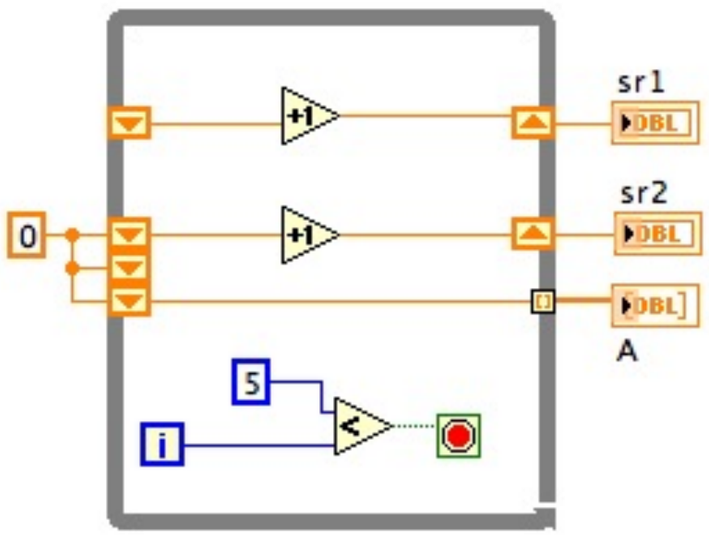


Combien de fois la boucle while est-elle éxécutée?
2 fois,
i=0-> 0>0 -> faux, continue,
i=1 -> 1>0 -> vrai -> stop
e= $0_{\text{initial}} + 1_{\text{iteration1}} + 1_{\text{iteration2}} = 2$



N=?

out=?



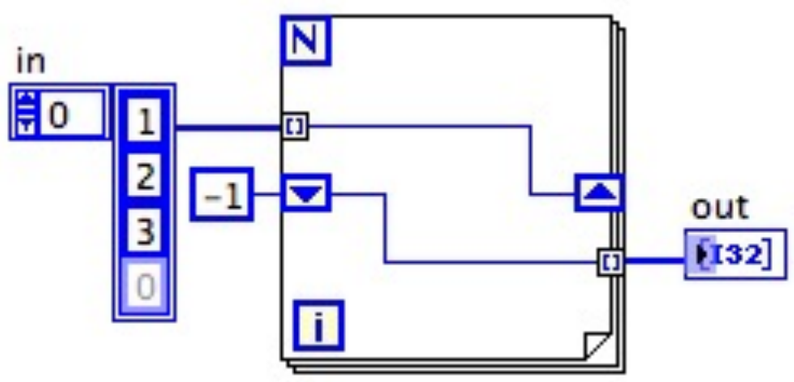
Après la première exécution, que valent

sr1=?

sr2=?

A=?

Idem après la 2e exécution ?



N=3

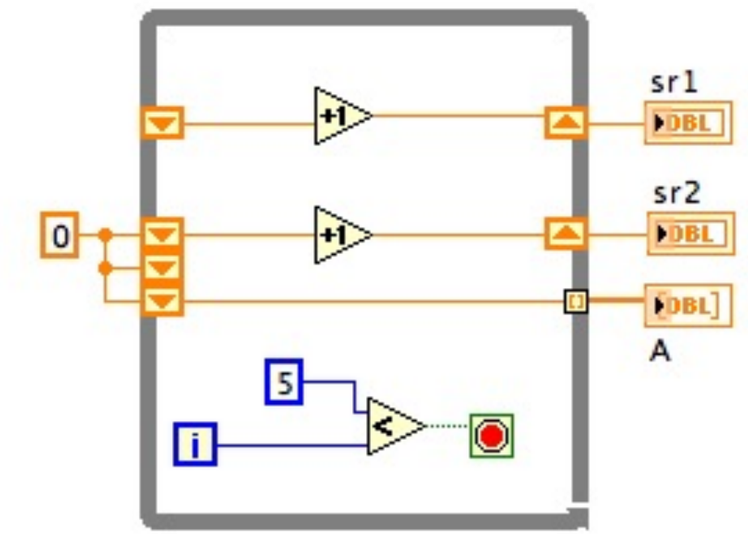
Out= la boucle for est exécutée 3 fois, le shift register est initialisé à -1, l'auto-indexing est activé.

A la première itération -1 est mémorisé localement par l'auto-indexing, le 1er élément du tableau est mis dans le shift register (à droite), elle sera disponible (à gauche) à la prochaine itération.

A la 2e itération, le '1' mis dans le shift register est lu et mémorisé est par l'auto-indexing. 2 est mis dans le shift register

A la 3e itération, le '2' mis dans le shift register est lu et mémorisé est par l'auto-indexing.

La boucle s'arrête. Out = valeurs mémorisées dans l'auto-indexing=[-1,1,2]



Après la première exécution, que valent

Nombres d'itérations: 5<0,faux, 5<1, faux,..5<5,faux,

5<6, vrai-> **stop**, i=0,1,2,3,4,5,6 -> 7x

sr1= $0_{initial}$ + $1_{i=0}$+$1_{i=1}$+1 +1+1+1+$1_{i=6}$ = 7

sr2= $0_{initial}$ + $1_{i=0}$+$1_{i=1}$+1 +1+1+1+$1_{i=6}$ = 7

A=connecté au shift register qui à 3 itérations de retard, initial values 0, -> A= [0,0,0,1,2,3,4]

Idem après la 2e exécution ?

7 est mémorié dans le shift register de la précédente exécution.

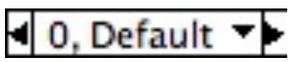
sr1= $7_{initial}$ + $1_{i=0}$+$1_{i=1}$+1 +1+1+1+$1_{i=6}$ = ~~7~~ **14**

sr2= $0_{initial}$ + $1_{i=0}$+$1_{i=1}$+1 +1+1+1+$1_{i=6}$ = 7

A=connecté au shift register qui à 3 itérations de retard, initial values 0, -> A= [0,0,0,1,2,3,4]

Rappel case structure = switch

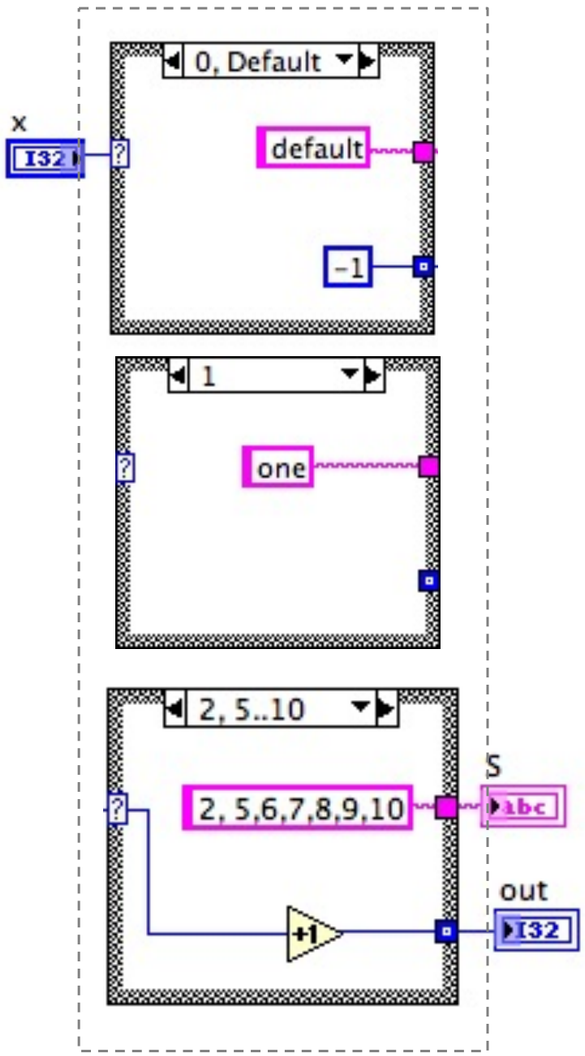
- si terminal non connecté, valeur par défaut (0, ou false ou '')



"Default" recupère toutes les entrées non-définies

Case selector peut être une valeur, des valeurs séparées pas des ','

ou des *ranges* fermés '1..5' ou ouverts '22..'



Si x = 8,

S=?

out=?

Si x = 6,

S=?

out=?

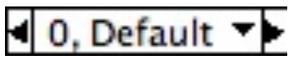
Si x = 1,

S=?

out=?

Rappel case structure = switch

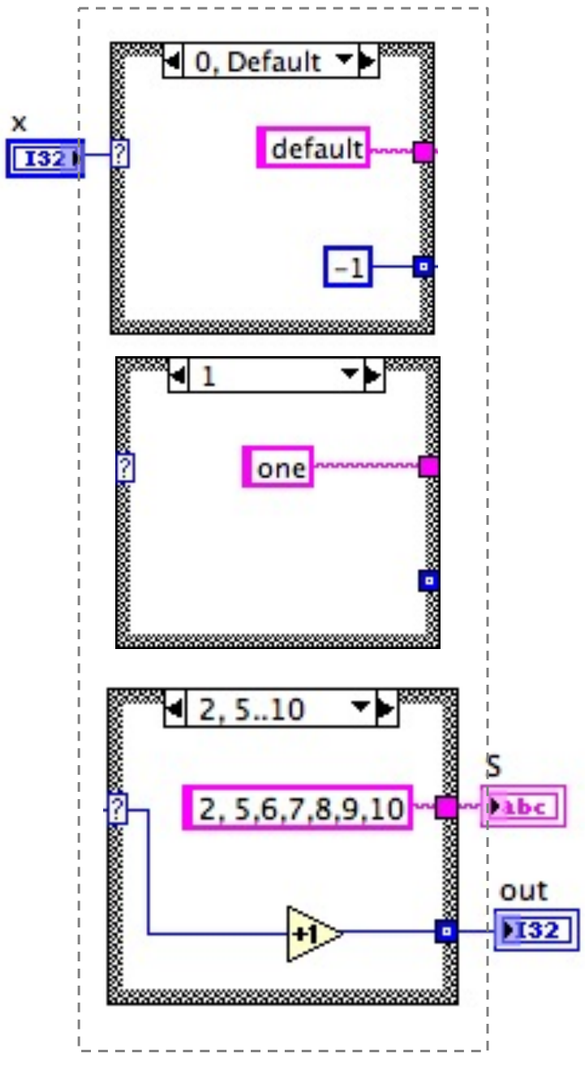
- si terminal non connecté, valeur par défaut (0, ou false ou '')



"Default" recupère toutes les entrées non-définies

Case selector peut être une valeur, des valeurs séparées pas des ','

ou des *ranges* fermés '1..5' ou ouverts '22..'

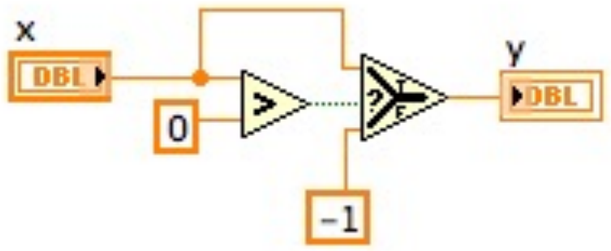


Si x = 8, entre 5 et 10 -> 3e frame
S= "2, 5,6,7,8,9,10"
out= x +1 = 8 + 1 = 9, '?' *contient* x = 8

Si x = 6, entre 5 et 10 -> 3e frame
S= "2, 5,6,7,8,9,10"
out= x +1 = 6 + 1 = 7, '?' *contient* x = 6

Si x = 3, aucun frame-> default -> 1e frame
S= "default "
out= -1

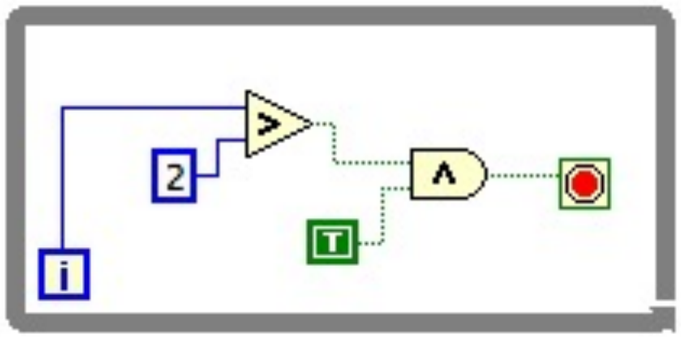
Si x = 1, 2e frame
S= "one"
out= valeur par défaut = 0



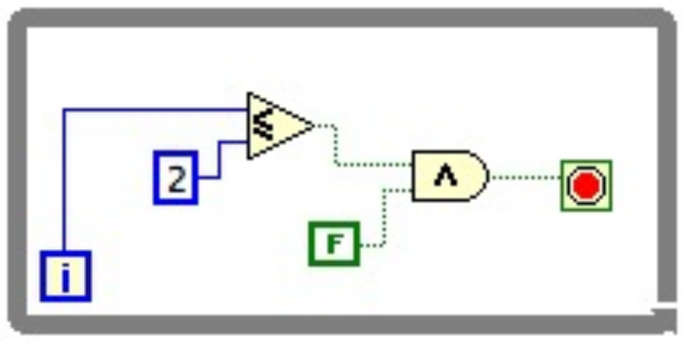
Si x=0, y=?

Si x=10, y=?

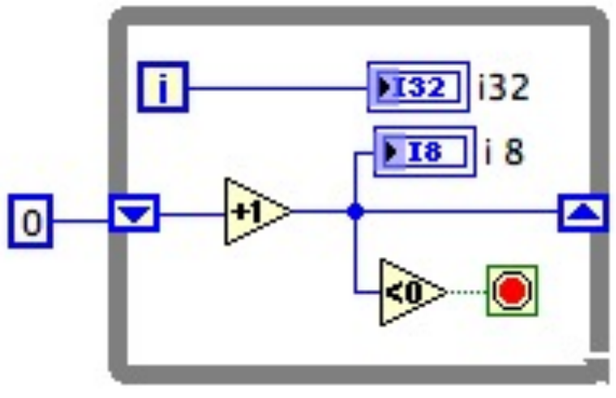
Si x=-21, y=?



Combien de fois s'exécute la boucle *while* ?



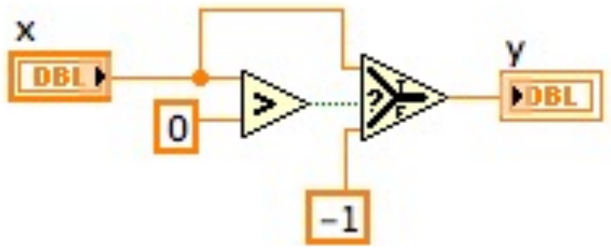
Combien de fois s'exécute la boucle *while* ?



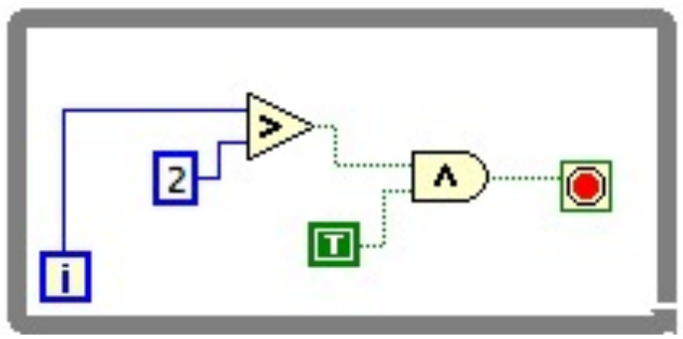
Combien de fois s'exécute la boucle *while* ?

i32=?

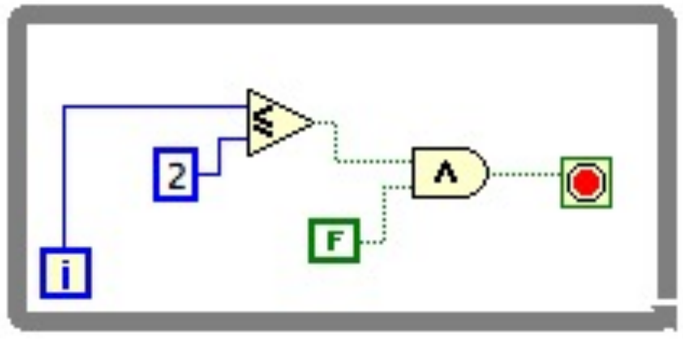
i8=?



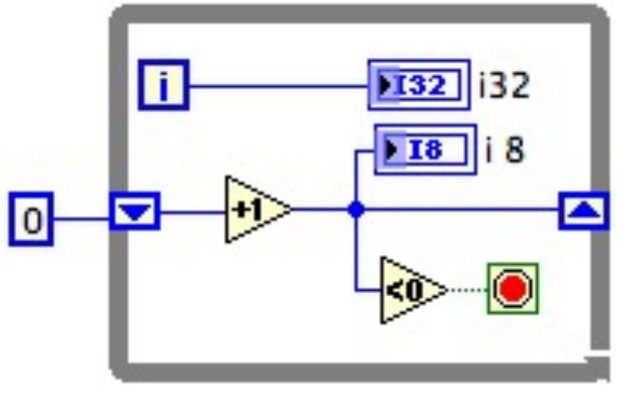
Si x=0, 0>0 faux, y= -1
Si x=10, 10>0 vrai, y= 10
Si x=-21, -21>0 faux, y= -1



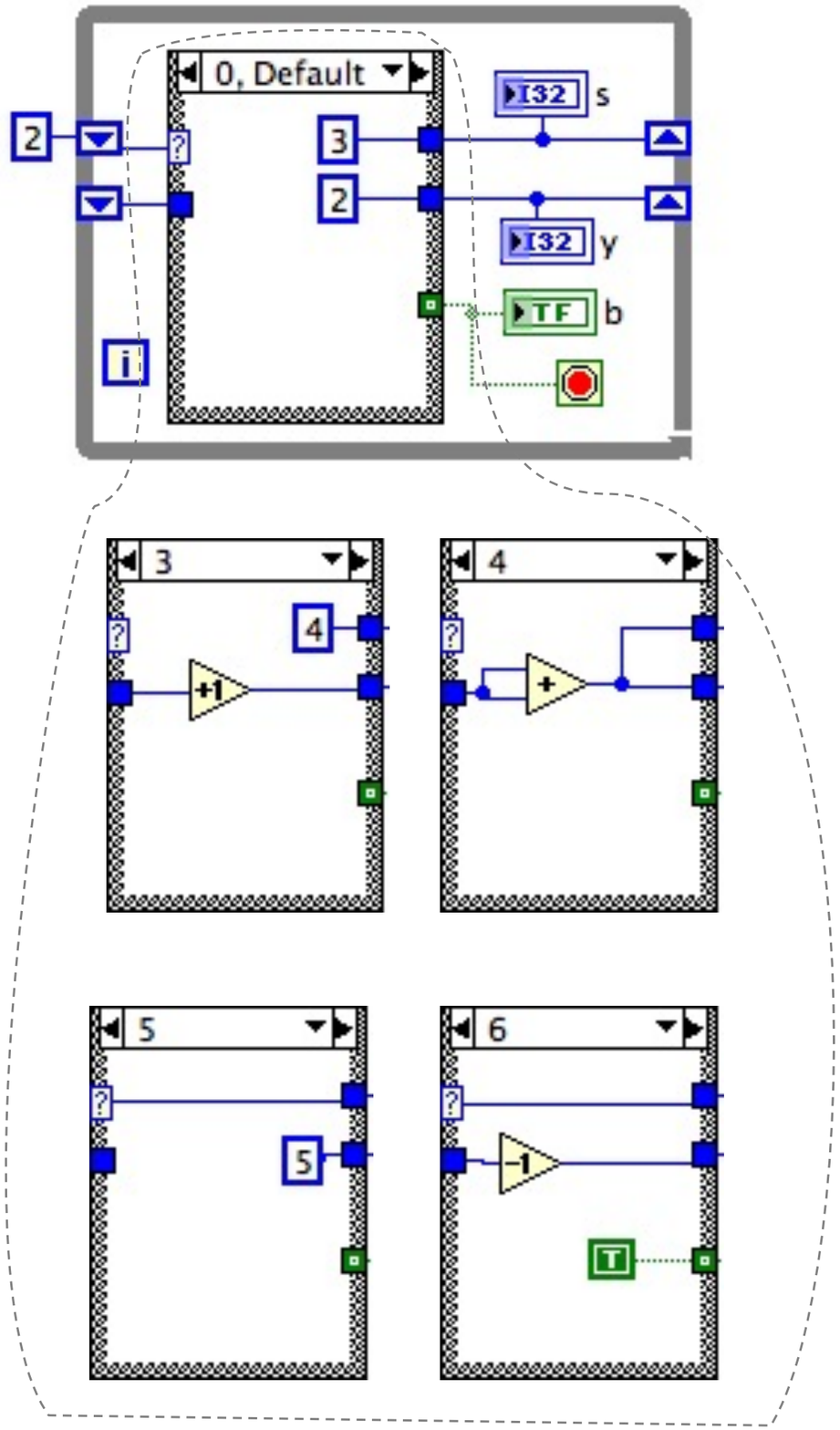
Combien de fois s'exécute la boucle *while* ? 4 x
i= 0; 0>2 -> faux; faux et vrai = faux - > continue
i= 1; 1>2 -> faux; faux et vrai = faux - > continue
i= 2; 2>2 -> faux; faux et vrai = faux - > continue
i= 3; 3>2 -> vrai; vraiet vrai = vrai- > stop



Combien de fois s'exécute la boucle *while* ? infini
i= 0; 0≤2 -> vrai; vrai et faux = faux - > continue
i= 1; 1≤2 -> vrai; vrai et faux = faux - > continue
La condition x et faux sera toujours faux -> boucle infinie

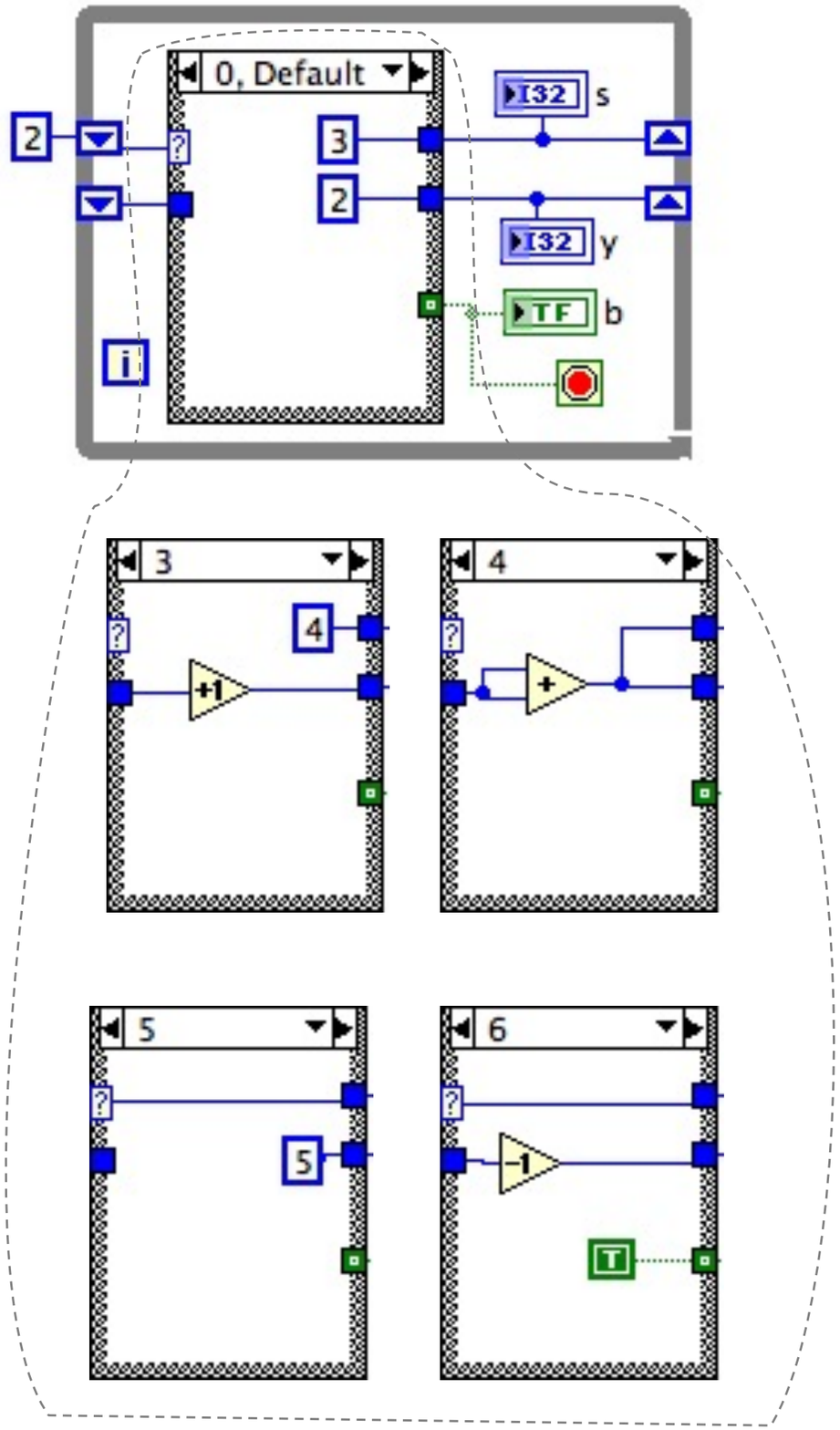


Combien de fois s'exécute la boucle *while* ? 128 x,
i32= 127, i8=-128
Le shift register est un integer codé sur 8 bit (I8), 2^8=256 valeurs,
réparties de -128 à +127, le shift register prend les valeurs suivantes:
$0_{initial}$, 1, 2, .. 127, -128 et $i_0<0$ faux, $i_1<0$ faux, $i_{127}<0$ faux, -> continue
$i_{-128}<0$ vrai -> stop



Quelles sont les valeurs successives de

- s = ?
- y = ?
- b = ?



Quelles sont les valeurs successives de

- s = ? y = ? b = ?

Shift register initialisé à 2, pas de frame 2
=>default -> s=3,y=2, b pas connecté –> default b= false,
3 est mémorisé dans le shift register

S-r =3 -> frame 3, s=4,y=2+1=3, b pas connecté –> default b= false,
4 est mémorisé dans le shift register

S-r =4 -> frame 4, s=3+3=6, y=3+3=6, b pas connecté –> default b= false,
6 est mémorisé dans le shift register

S-r =6 -> frame 6, s=6, y=6-1=5, b= true, -> STOP

Note: le frame 5 n'est jamais exécuté!

Dans Matlab sans employer de boucle

- Créer le vecteur [1000 995 990 … 10 5 0]
- Créer le vecteur [1/2; 3/4; 5/6; 7/8; 9/10; … ; 99/100]
- Calculer le produit des éléments du vecteur X plus grand que 1

Avec a= [1 2 3], b=[4 5 6]

- X = a+b
- Y = [a'; b']
- Z = b(a(1,3))

- w=sum(linspace(1,3,3))

Avec m = [1 2 3; 4 5 6]

- m(5) = m(2,3)
- m(:,[1 3])=m(:,[3 1])
- X=sum([m(3) m(2,2)])

Dans Matlab sans employer de boucle

- Créer le vecteur [1000 995 990 … 10 5 0]

  x1 = 1000:-5:0;

- Créer le vecteur [1/2; 3/4; 5/6; 7/8; 9/10; … ; 99/100]

  x2 = (1:2:99) ./ (2:2:100) % notez le "**./**" pour la division élément par éléments!

- Calculer le produit des éléments du vecteur X plus grand que 1

  x3 = prod(X(find(X>1))) % find(x>1) retourne les idx, x(find(x>1)) les valeurs, et prod calcul le produit

Avec a= [1 2 3], b=[4 5 6]

- X = a+b **X = [5 7 9]** % addition éléments par éléments
- Y = [a', b'] **Y = [1 4; 2 5; 3 6]**
- Y = [a'; b'] **Y = [1 2 3 4 5 6]'** %vecteur colonne!
- Z = b(a(1,3)) **Z = 6** % a(1,3) = 3, b(3) = 6,

- w=sum(linspace(1,3,3)) **w = 6** % linspace(1,3,3)= [1,2,3], -> 6

Avec m = [1 2 3; 4 5 6]

- m(5) = m(1,3) **m = [1 2 3; 4 5 6]** % **m** ne change pas, car m(5) == m(1,3)
- m(:,[1 3])=m(:,[3 1]) **m = [3 2 1; 6 5 4]** % colonnes 1 et 3 inversées
- X=sum([m(3) m(2,2)]) **X = 7** % m(3)=2, m(2,2) =5

Dans Matlab

Avec a= [1 2; 3 4]

- a(:,:,3)=[8 9;10 11]

Avec a= [1 2 3], b=[4 5 6]

- W = a'*b
- X = a*b
- Y = a.*b
- Z = a*b'

Dans Matlab

Avec a= [1 2; 3 4]

- a(:,:,3)=[8 9;10 11] % ajoute une 3e dimension à la matrice *a*

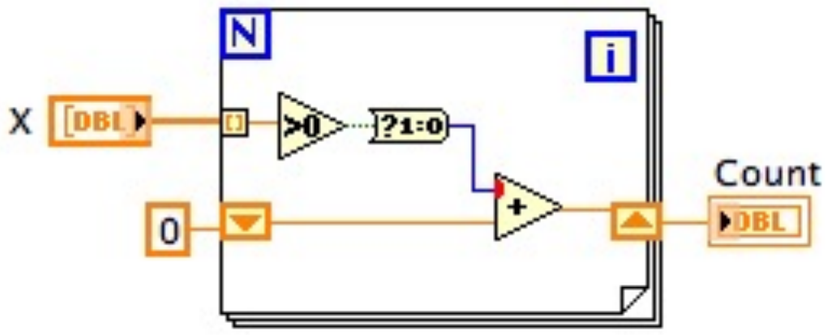
a(:,:,1) = [1 2 ; 3 4]

a(:,:,2) = [0 0 ; 0 0] % non défini, alors valeur(s) par défaut

a(:,:,3) = [8 9; 10 11]

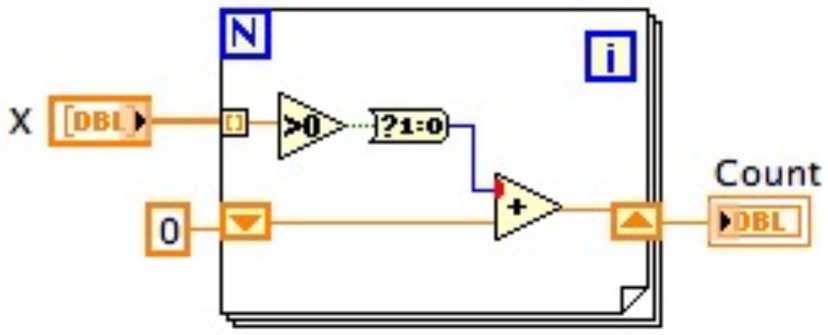
Avec a= [1 2 3], b=[4 5 6]

- W = a'*b **W =[4 5 6; 8 10 12; 12 15 18]**
- X = a*b Error using *, Inner matrix dimensions must agree.
- Y = a.*b **Y =[4 10 18]**
- Z = a*b' **Z= 32**



Ecrire le code matlab équivalent (sans boucle)

Ecrire les codes matlab (sans boucle) et C équivalents



Ecrire le code matlab équivalent (sans boucle)

'?1:0' tranforme un boolean en 0 ou 1

count = lenght(find(X>0))

Ecrire les codes matlab (sans boucle) et C équivalents

p = prod(Y)

```
p=1; // 1 élément neutre de la multiplication
for (i=0; i < sizeof(Y); i++) {
            p = p * Y[i];
   }
```

Regardez également les exemples traités en classe!