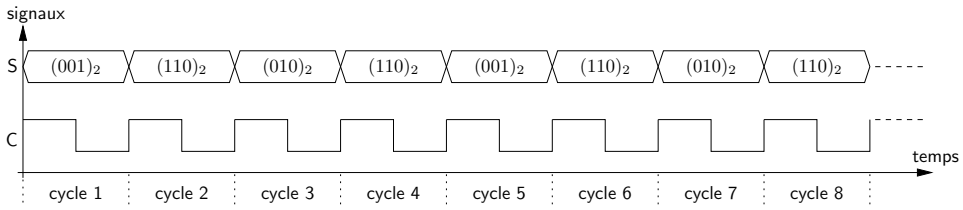
CC-TD-F - LIF6 Architecture matérielle et logicielle

Aucun document autorisé, durée 45 min, barême donné à titre indicatif : il pourra être modifié.
16 juin 2010

I Un circuit séquentiel

On souhaite mettre au point un circuit séquentiel permettant de générer la séquence de valeurs $(001)_2$, $(110)_2$, $(010)_2$, $(110)_2$, et la répéter de façon périodique. On note $S = (s_2, s_1, s_0)$ la sortie du circuit. Un chronogramme représentant l'évolution de la valeur de S au cours du temps sera par exemple le suivant :



Pour concevoir ce circuit séquentiel, on va le modéliser à l'aide d'un automate fini séquentiel. Il vous est demandé de justifier brièvement chacune de vos réponses.

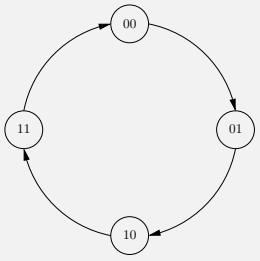
Q.I.1) - Quatre états sont suffisants pour réaliser l'automate fini demandé : pourquoi ?

Il suffit de 4 états distincts, puisque la séquence de valeurs à répéter sur la sortie est de periode 4.

Q.I.2) - Comme quatre états sont suffisants, on utilise un registre (à base de bascules flip-flops) à 2 bits pour le stockage de l'état courant $Q = (q_1, q_0)$ du circuit séquentiel. On choisit le codage suivant pour chacun des états de l'automate :

état Q	sortie S
00	001
01	110
10	010
11	110

Donnez une représentation graphique de l'automate fini demandé.



Q.I.3) - Soit F la fonction de transition de l'automate : donnez la table de vérité de $F(Q)$ en fonction de Q. Complétez pour cela le tableau de vérité suivant :

Q		$F(Q)$	
q_1	q_0	f_1	f_0
⋮	⋮	⋮	⋮

Q		$F(Q)$	
q_1	q_0	f_1	f_0
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

Q.I.4) - Exprimez f_1 et f_0 en fonction de q_1 et de q_0 à l'aide de formules booléennes simples.

$f_1(q_1, q_0) = q_1 \oplus q_0$ et $f_0(q_1, q_0) = \overline{q_0}$.

Q.I.5) - Complétez la table de vérité de s_2, s_1 et s_0 en fonction de q_1 et q_0.

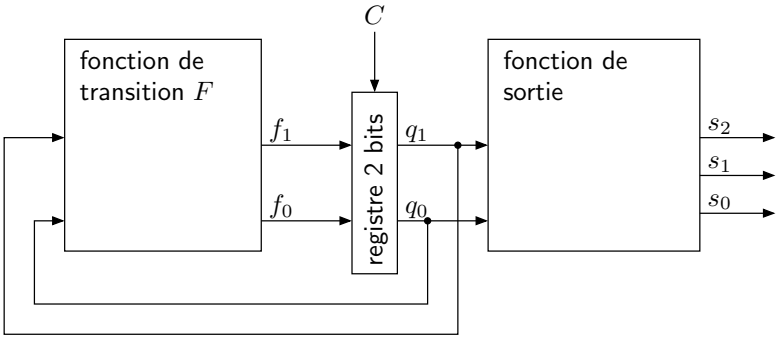
Q		$S(Q)$		
q_1	q_0	s_2	s_1	s_0
⋮	⋮	⋮	⋮	⋮

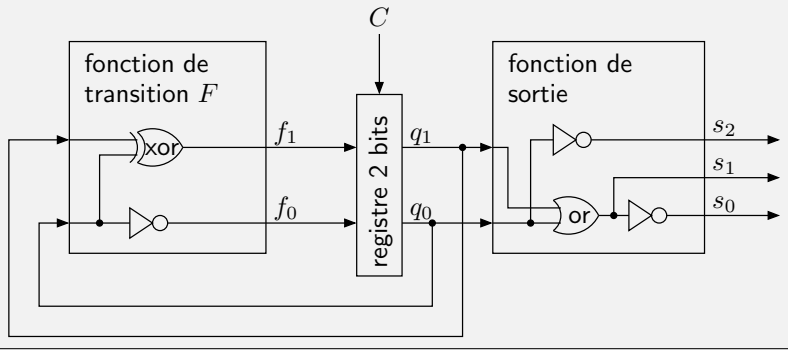
Q		$S(Q)$		
q_1	q_0	s_2	s_1	s_0
0	0	0	0	1
0	1	1	1	0
1	0	0	1	0
1	1	1	1	0

Q.I.6) - Exprimez s_2, s_1 et s_0 en fonction de q_1 et q_0 à l'aide de formules booléennes simples.

$s_2(q_1, q_0) = \overline{q_0}$, $s_1(q_1, q_0) = q_1 + q_0$ et $s_0(q_1, q_0) = \overline{q_1 + q_0}$.

Q.I.7) - Reproduisez, en le complétant, le logigramme ci-dessous, afin d'obtenir le circuit séquentiel demandé.

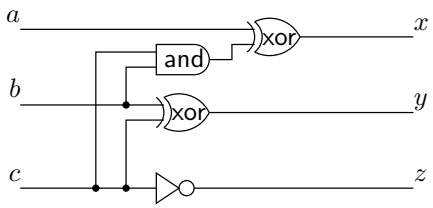




II Circuits combinatoires

II.1 Analyse d'un circuit combinatoire

On considère le circuit combinatoire suivant, dont les entrées sont a, b et c, et les sorties x, y et z :



Q.II.1) - Donnez des expressions booléennes pour x, y et z en fonction de a, b et c.

$x = a \oplus bc$, $y = b \oplus c$ et $z = \overline{c}$.

Q.II.2) - Complétez la table de vérité suivante :

a	b	c	x	y	z
⋮	⋮	⋮	⋮	⋮	⋮

a	b	c	x	y	z
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

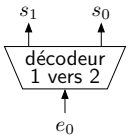
Q.II.3) - Quelle fonction réalise le circuit considéré ?

$(xyz)_2 = (abc)_2 + 1 \mod 8$.

II.2 Décodeurs

Un décodeur k bits est un circuits a k entrées $e_{k-1}, \ldots, e_0$ et 2^k sorties $s_{2^k-1}, \ldots, s_0$: la sortie $s_{(e_{k-1},\ldots,e_0)_2}$ dont l'indice est indiqué par les entrées est activée et toutes les autres restent inactives.

Q.II.4) - On représente de la manière indiquée ci-dessous un décodeur 1 vers 2.



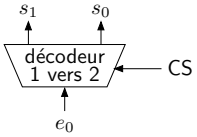
e_0	s_1	s_2
⋮	⋮	⋮

Complétez sa table de vérité, et donnez un logigramme pour un décodeur 1 vers 2.

On souhaite fabriquer un décodeur 2 vers 4 à partir de deux décodeurs 1 vers 2. Cela n'est pas facile à réaliser avec des décodeurs classiques, il faut leur ajouter une entrée supplémentaire CS (*Chip Select*). Le rôle de cette entrée est le suivant :

- quand CS=0, les sorties du décodeur restent à 0, quelles que soient les autres entrées ;
- quand CS=1, le décodeur se comporte comme un décodeur classique.

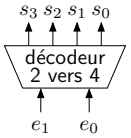
Q.II.5) - On représente de la manière indiquée ci-dessous un décodeur 1 vers 2 avec CS.



CS	e_0	s_1	s_2
⋮	⋮	⋮	⋮

Complétez sa table de vérité, et donnez un logigramme pour un décodeur 1 vers 2 avec CS.

Q.II.6) - On considère un décodeur 2 vers 4 classique, représenté ci-dessous : Dressez sa table de vérité.



e_0	e_1	s_3	s_2	s_1	s_0
⋮	⋮	⋮	⋮	⋮	⋮

Q.II.7) - En utilisant deux décodeurs 1 vers 2 avec CS, ainsi qu'un décodeur 1 vers 2 classique, proposez un circuit réalisant un décodeurs 2 vers 4 classique. Justifiez brièvement votre réponse en vous basant sur la table de vérité de la question précédente.

III Programmation en assembleur du LC-3

III.1 Somme des entiers de 1 à `n`

On considère le programme incomplet ci-dessous :

```
            .ORIG x3000     ; adresse de début de programme
; partie dédiée au code

            ; description du programme à implanter
            ;   int i;          // indice de boucle, qui sera maintenu dans le registre R0
            ;   int a;          // accumulateur, qui sera maintenu dans le registre R1
            ;   i = n;          // l'entier à l'adresse indiquée par l'étiquette n
            ;   a = 0;          // initialisation de l'accumulateur
            ;   while(i >= 1) {
            ;     a += i;       // on accumule i dans a
            ;     i--;          // on décrémente l'indice de boucle i
            ;   }
            ;   r = a;          // on stocke le résultat à l'adresse indiquée par l'étiquette r

            A COMPLETER

; partie dédiée aux données et au résultat
n:          .FILL #8        ; entrée n
r:          .BLKW #1        ; espace pour stocker le résultat
            .END
```

Q.III.1) - Complétez ce code, en vous conformant à la partie « `description du programme à implanter` » du listing. Ne reportez sur votre copie que le code correspondant à la partie « `A COMPLETER` ».

On peut compléter le programme par le code suivant :

```
            AND R1,R1,#0    ; on initialise R1 à l'entier 0
            LD R0,n         ; on initialise R0 à l'entier à l'adresse désignée par n
loop:       BRnz endloop    ; si la valeur dans R0 est négative ou nulle, on sort de la boucle
            ADD R1,R1,R0    ; on accumule le contenu de R0 dans R1
            ADD R0,R0,#-1   ; on décrémente l'indice de boucle
            BR loop         ; on saute au début de boucle
endloop:    ST R1,r         ; on stocke le résultat à l'adresse désignée par r
            HALT            ; fin du programme (TRAP x25)
```

III.2 Analyse d'une partie de programme

On considère le programme suivant :

```
            .ORIG x3000   ; adresse de début de programme
; partie dédiée au code
        LEA R6, stackend
        ; ... du code qui ne nous concerne pas ...
        ; début
        ADD R6,R6,#-2
        STR R1,R6,#1
        STR R2,R6,#0
        LDR R2,R6,#1
        LDR R1,R6,#0
        ADD R6,R6,#2
        ; fin
        ; ... du code qui ne nous concerne pas ...
; partie dédiée à la pile d'exécution
            .BLKW #25         ; emplacement réservé à la pile
stackend:   .FILL #0          ; fin de pile
            .END
```

Q.III.2) - Comment est utilisé généralement le registre `R6` ? Que fait l'instruction : `LEA R6, stackend` ?

Cette instruction charge l'adresse effective désignée par l'étiquette `stackend` dans le registre R6, utilisé comme pointeur de pile

Q.III.3) - Pour chaque instruction située entre les commentaires « `début` » et « `fin` », donnez un commentaire faisant référence à la pile d'exécution pour expliquer ce que fait l'instruction.

```
ADD R6,R6,#-2   ; Déplacement du sommet de pile
STR R1,R6,#1    ; Empilement de R1
STR R2,R6,#0    ; Empilement de R2
LDR R2,R6,#1    ; Dépilement du contenu de R1 dans R2
LDR R1,R6,#0    ; Dépilement du contenu de R2 dans R1
ADD R6,R6,#2    ; Restauration du sommet de pile
```

Q.III.4) - Que fait la partie du programme située entre les commentaires « `début` » et « `fin` » ?

Cette partie de programme échange le contenu des registres R1 et R2, en se servant de la pile pour le stockage intermédiaire des valeurs.

syntaxe	**action**	**nzp**	**codage**																
			opcode				arguments												
			F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
NOT DR,SR	DR <- not SR	*	1	0	0	1	DR			SR			1	1	1	1	1	1	
ADD DR,SR1,SR2	DR <- SR1 + SR2	*	0	0	0	1	DR			SR1			0	0	0		SR2		
ADD DR,SR1,Imm5	DR <- SR1 + SEXT(Imm5)	*	0	0	0	1	DR			SR1			1	Imm5					
AND DR,SR1,SR2	DR <- SR1 and SR2	*	0	1	0	1	DR			SR1			0	0	0		SR2		
AND DR,SR1,Imm5	DR <- SR1 and SEXT(Imm5)	*	0	1	0	1	DR			SR1			1	Imm5					
LEA DR,label	DR <- PC + SEXT(PCoffset9)	*	1	1	1	0	DR			PCoffset9									
LD DR,label	DR <- mem[PC + SEXT(PCoffset9)]	*	0	0	1	0	DR			PCoffset9									
ST SR,label	mem[PC + SEXT(PCoffset9)] <- SR		0	0	1	1	SR			PCoffset9									
LDR DR,BaseR,Offset6	DR <- mem[BaseR + SEXT(Offset6)]	*	0	1	1	0	DR			BaseR			Offset6						
STR SR,BaseR,Offset6	mem[BaseR + SEXT(Offset6)] <- SR		0	1	1	1	SR			BaseR			Offset6						
LDI DR,label	DR <- mem[mem[PC + SEXT(PCoffset9)]]	*	1	0	1	0	DR			PCoffset9									
STI SR,label	mem[mem[PC + SEXT(PCoffset9)]] <- SR		1	0	1	1	SR			PCoffset9									
BR[n][z][p] label	Si (cond) PC <- PC + SEXT(PCoffset9)		0	0	0	0	n	z	p	PCoffset9									
NOP	No Operation		0	0	0	0	0	0	0	0 0 0 0 0 0 0 0 0									
JMP BaseR	PC <- BaseR		1	1	0	0	0	0	0	BaseR			0 0 0 0 0 0						
RET (JMP R7)	PC <- R7		1	1	0	0	0	0	0	1	1	1	0 0 0 0 0 0						
JSR label	R7 <- PC; PC <- PC + SEXT(PCoffset11)		0	1	0	0	1	PCoffset11											
JSRR BaseR	R7 <- PC; PC <- BaseR		0	1	0	0	0	0	0	BaseR			0 0 0 0 0 0						
RTI	cf. interruptions		1	0	0	0	0 0 0 0 0 0 0 0 0 0												
TRAP Trapvect8	R7 <- PC; PC <- mem[Trapvect8]		1	1	1	1	0	0	0	0	Trapvect8								
Réservé			1	1	0	1													

Tab. 1 – Récapitulatif des instructions du LC-3