Electronique numérique

Valentin Gies

Seatech - 3A
Université de Toulon (UTLN)

Plan du Chapitre introductif

Le comptage binaire

- Comptage en base 2.
- Deux *éléments binaires* ou *bits* (bit=Binary digIT) : 0 et 1
- Adapté à la technologie *tout ou rien* : transistors MOS
- 8 bits = 1 octet $\Rightarrow 2^8 = 256$ valeurs
- Les octets sont souvent représentés en base 16 (hexadecimal) par groupe de 4 bits :
  ex. $0d134 = 0b10000110 = 0x86$

Conversions binaire ou hexadécimal vers base 10

- Conversion vers la base 10 :
  - $0b10010 = 1 * 2^4 + 0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 = 18$
  - $0x1A6 = 1 * 16^2 + 10 * 16^1 + 6 * 16^0 = 422$
  - $0o345 = 3 * 8^2 + 4 * 8^1 + 5 * 8^0 = 229$

- Conversion vers la base 2 ou 16 (soustractions successives des plus grandes puissances)
  - $456 = 2^8 + 200 = 2^8 + 2^7 + 72 = 2^8 + 2^7 + 2^6 + 8 = 2^8 + 2^7 + 2^6 + 2^3 = 0b111001000$
  - $456 = 16^2 + 200 = 16^2 + 12 * 16^1 + 8 = 16^2 + 12 * 16^1 + 8 * 16^0 = Ox1C8$

Codes numériques

- Code binaire pur :
  Code pondéré par des puissances de 2.
  - $0b11010010 =$
    $1*2^7+1*2^6+0*2^5+1*2^4+0*2^3+0*2^2+1*2^1+0*2^0 = 210$
  - Ne permet pas la représentation des nombre négatifs
  - Permet de représenter des nombres de 0 à 2^N-1

- Code binaire en complément à 2 :
  Le bit de poids fort a un poids négatif.
  - $0b11010010 =$
    $-1*2^7+1*2^6+0*2^5+1*2^4+0*2^3+0*2^2+1*2^1+0*2^0 = -46$
  - Si le bit de poids fort est négatif :
    $0b11010010 = -0b00101101 - 1 = -45 - 1 = -46$
  - Permet de représenter des nombres négatifs :
  - Permet de représenter des nombre de -2^{N-1} à $2^{N-1}-1$
  - Format de sortie de nombreux capteurs !

Codes numériques

Il existe d'autres codes moins utilisés

- Code Binaire codé décimal (BCD) :
  Code pondéré avec les poids $1, 2, 4, 8, 10, 20, 40, 80, 100$
  - $0b11010010 =$
    $1 * 80 + 1 * 40 + 0 * 20 + 1 * 10 + 0 * 8 + 0 * 4 + 1 * 2 + 0 * 1 = 132$
  - Code utilisé pour les afficheurs LCD

- Code Gray (ou code binaire réfléchi) :
  On ne change qu'un seul bit entre deux nombres consécutifs

Décimal	Gray			
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	0	1	0
4	0	1	1	0
5	0	1	1	1
6	0	1	0	1
7	0	1	0	0

Décimal	Gray			
8	1	1	0	0
9	1	1	0	1
10	1	1	1	1
11	1	1	1	0
12	1	0	1	0
13	1	0	1	1
14	1	0	0	1
15	1	0	0	0

Codes numériques utilisés en transmission

- Codes p parmi n :
  Codes possédant n éléments binaires dont p sont à 1 :
  $C_n^p = \frac{n!}{p!(n-p)!}$ possibilités
- Code 2 parmi 5

Déc.	Télécoms				
	0	1	2	3	6
1	1	1	0	0	0
2	1	0	1	0	0
3	1	0	0	1	0
4	0	1	0	1	0
5	0	0	1	1	0
6	1	0	0	0	1
7	0	1	0	0	1
8	0	0	1	0	1
9	0	0	0	1	1
0	0	1	1	0	0

  - Permet une vérification de l'intégrité des données tranmises
  - Ne permet pas la correction des erreurs éventuelles
  - Code utilisé dans les centraux téléphoniques (*cross bar*)
  - Les codes barres utilisent également un code proche du 3 parmi 9.

Codes numériques utilisés en transmission

- Codes à contrôle de parité :
  On ajoute au mot de n bits transmis un bit supplémentaire de parité :
  - $0b10110011 \Rightarrow 0b110110011$
  - $0b10110111 \Rightarrow 0b010110111$
  - Permet la détection d'une erreur dans le données transmises
  - Il existe une possibilité de correction si on contrôle la parité temporelle dans des paquets de données également.
- Code de Hamming : permet la correction d'erreur moyennant l'ajout de 3 bits pour 4 bits transmis.

Codes alpha-numériques

- Le code ASCII :
  Sert à coder les chiffres, lettres et caractères spéciaux

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

L'algèbre de Boole

- Algèbre au sens mathématique du terme
- Elle est basées sur les opérateurs :
  - *OU* (noté $+$)
  - *ET* (noté .)
- Elle permet de décrire n'importe quelle fonction logique non séquentielle.
- La dimension temporelle n'est pas présente.

L'algèbre de Boole

Table de vérité et représentation graphique de l'opérateur OU :

In A	In B	Output
0	0	0
0	1	1
1	0	1
1	1	1

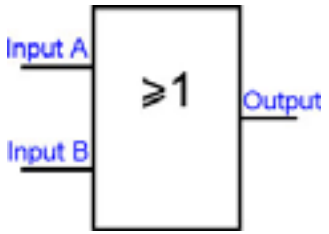
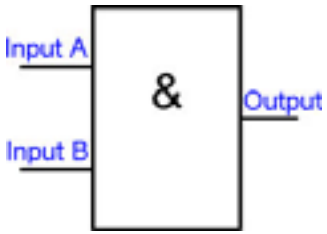


Table de vérité et représentation graphique de l'opérateur ET :

In A	In B	Output
0	0	0
0	1	0
1	0	0
1	1	1



L'algèbre de Boole

Propriétés	**OU**	**ET**
Identité	$e + 0 = e$	$e.1 = e$
Elément neutre	$e + 0 = e$	$e.1 = e$
Elément absorbant	$e + 1 = 1$	$e.0 = 0$
Idempotence	$e + e = e$	$e.e = e$
Complémentation	$e + \overline{e} = 1$	$e.\overline{e} = 0$
Involution	$\overline{\overline{e}} = e$	$\overline{\overline{e}} = e$
Commutativité	$e + f = f + e$	$e.f = f.e$
Associativité	$e + (f + g) = (e + f) + g$	$e.(f.g) = (e.f).g$
Distributivité	$e + (f.g) = (e + f).(e + g)$	$e.(f+g) = e.f+e.g$
Absorption	$e + e.f = e$	$e.(e + f) = e$
De Morgan	$\overline{e + f} = \overline{e}.\overline{f}$	$\overline{e.f} = \overline{e} + \overline{f}$

L'algèbre de Boole : Théorème de *De Morgan*

- Un théorème fondamental :
  - Le complément d'un produit est égal à la somme des compléments : $\overline{e.f} = \overline{e} + \overline{f}$
  - Le complément d'une somme est égal au produit des compléments : $\overline{e + f} = \overline{e}.\overline{f}$
- Conséquence :
  - L'algèbre de Boole peut être mise en oeuvre électroniquement en utilisant uniquement des portes logiques NAND ou des portes logiques NOR.
  - Propriété qui sert de base à la conception de la partie combinatoire des FPGA

Représentation des fonctions logiques : Table de vérité

Exemple : le *Full Adder*, permet de sommer deux bits et la retenue issue d'un additionneur précédent.

Carry input	Bit a	Bit b	Carry output	Sum bit output
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Représentation des fonctions logiques : Tableaux de Karnaugh

Exemple : le *Full Adder*, permet de sommer deux bits et la retenue issue d'un additionneur précédent.

Sum bit

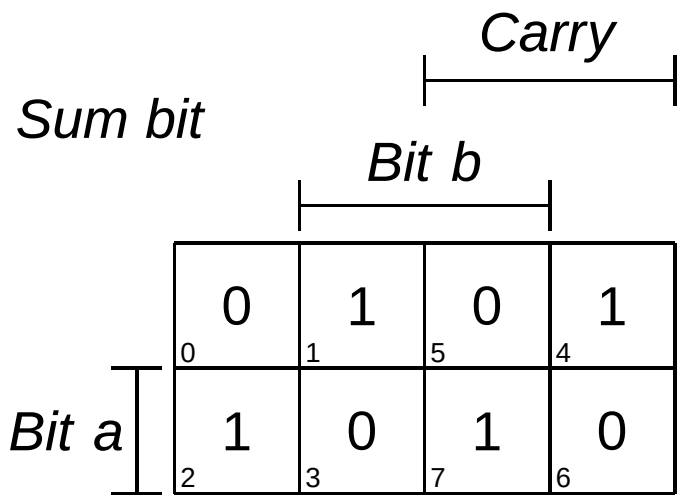
Bit a \ Carry, Bit b	00	01	11	10
0	0 (0)	1 (1)	0 (5)	1 (4)
1	1 (2)	0 (3)	1 (7)	0 (6)

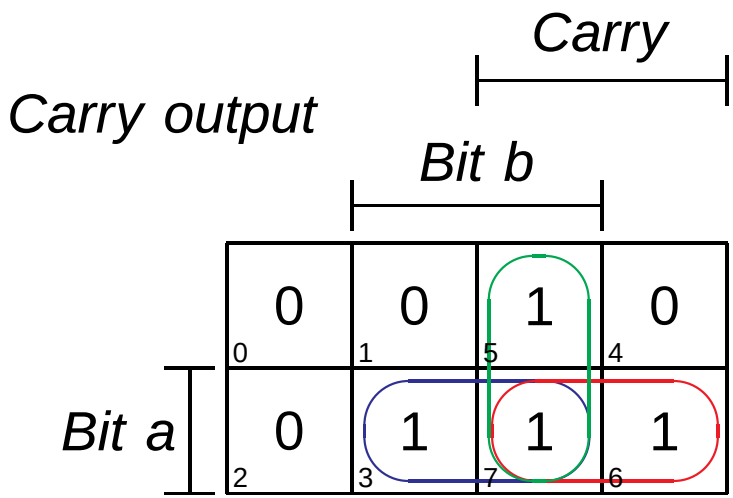
Carry output

Bit a \ Carry, Bit b	00	01	11	10
0	0 (0)	0 (1)	1 (5)	0 (4)
1	0 (2)	1 (3)	1 (7)	1 (6)

⇒ **Permet de simplifier les expressions combinatoires par regroupements**

Représentation des fonctions logiques : Tableaux de Karnaugh





Pas de regroupements possibles

$$Sum = a.\overline{b}.\overline{carry} + \overline{a}.b.\overline{carry} + a.b.carry + \overline{a}.\overline{b}.carry$$

3 regroupements possibles

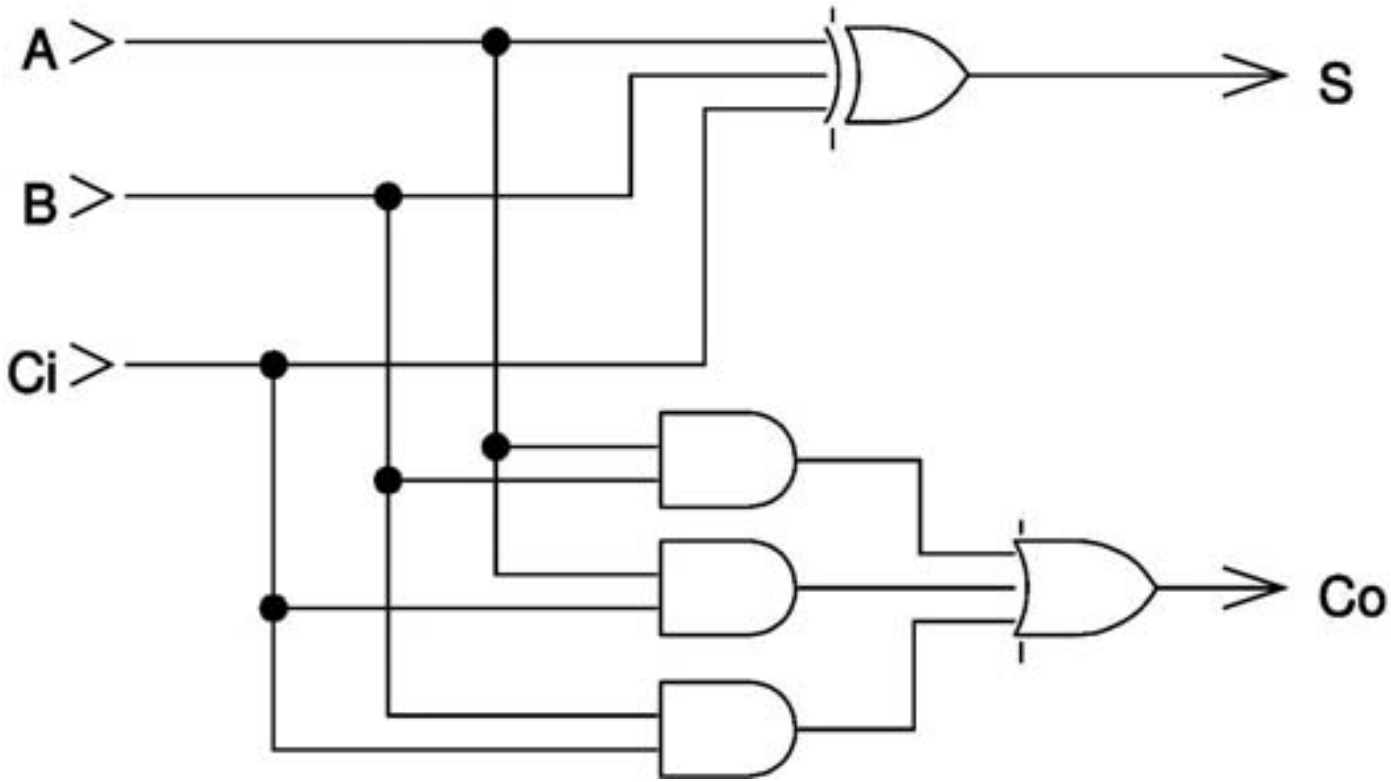
$$Carry\ Output = a.b + a.carry + b.carry$$

Simplification en logique combinatoire

- Tableaux de Karnaugh : efficaces jusqu'à 4 variables, au delà trop complexe.
- Méthodes algorihtmiques :
  - Algorithme de Quine-MacCluskey : recherche des implicants majeurs, très gourmand en calcul (*NP complet*)
  - Méthode de Petrick : simple à implanter en programmation informatique
  - Heuristique Expresso
    - Développée par IBM en 1986
    - Non optimale mais très proche, et peu couteuse en temps et en mémoire.
    - Utilisée dans les logiciels de synthèse automatique (Altera, Xilinx...)

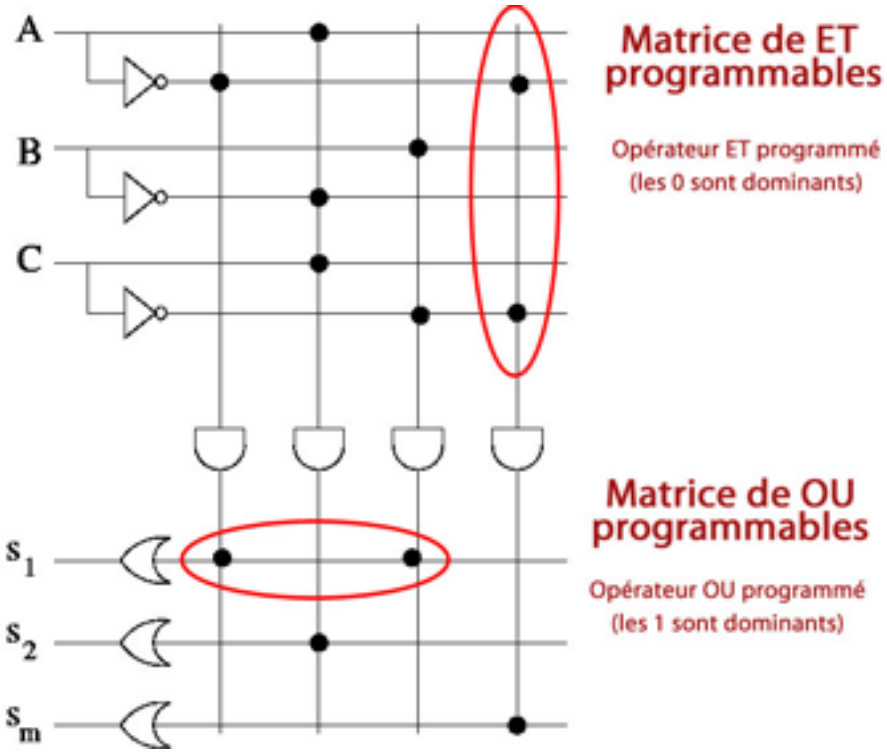
Implantation des fonctions logiques combinatoires : portes logiques

Exemple : le *Full Adder*, permet de sommer deux bits et la retenue issue d'un additionneur précédent.



Implantation des fonctions logiques combinatoires : composants programmables

Structure générique : tableau de portes *OU* et *ET* séparées par un suiveur (FPGA)



Il existe des structures où seules les parties *ET* ou *OU* sont programmables : les *PAL* (*ET* programmables) ou des *PROM* (*OU* programmables)

Fonctions réalisables en logique combinatoire

Sont réalisables en logique combinatoire, toutes les fonctions ne dépendant pas du temps :

- Unité arithmétique et logique : addition, soustraction, ET OU, OU exclusif, multiplication...
- Comparateurs
- Multiplexeurs

Les fonctions dépendant des états temporellement précédents nécessitent l'usage de la logique séquentielle.

Circuits séquentiels

Un circuit séquentiel si l'état de ses sorties dépend des entrées mais également des états antérieurs des entrées ou sorties. Ces circuits permettent entre autre :

- La mémorisation
- Le décalage
- Le comptage

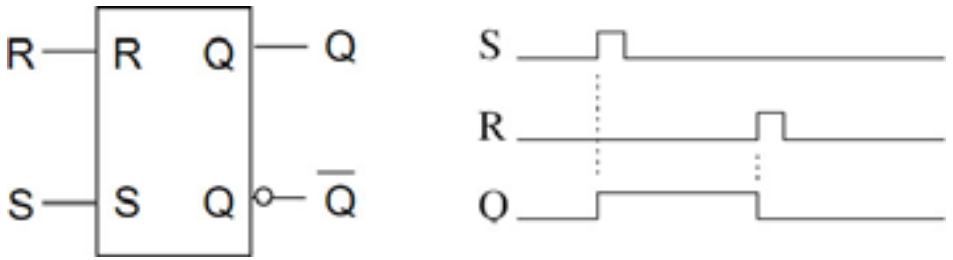
Ils reposent sur les composants principaux suivant :

- Les bascules
- Les registres
- Les compteurs

Le plus souvent, ces circuits sont synchronisés à l'aide d'une horloge.

Mémorisation : les bascules

La bascule RS (Reset - Set) :



- La sortie change d'état de manière asynchrone dès que les entrées S ou R sont actives.
- Lorsque les entrées sont à 0, la sortie mémorise son état précédent.
- La configuration $R = S = 1$ a des effets dépendant de l'implantation électronique.

Mémorisation : les bascules

Une bascule RS peut être réalisée à l'aide de deux portes *NOR* (*Non OU*) ou de deux portes *NAND* et d'inverseurs :

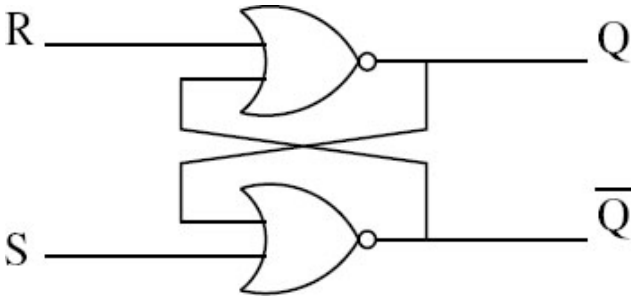


fig: bascule RS *NOR*

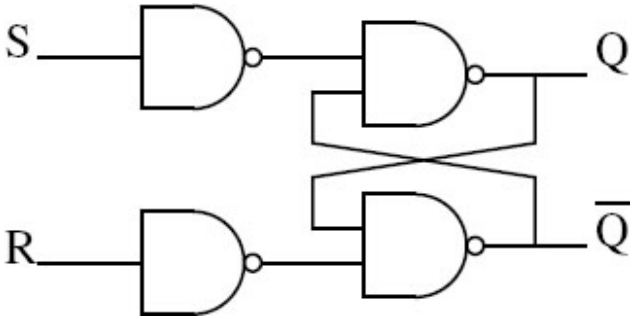


fig: bascule RS *NAND*

- Dans le cas d'une bascule RS *NOR* :
  $Q = \overline{R + (\overline{S + Q})} = \overline{R}.(S + Q)$, $\Rightarrow$ Reset prioritaire si $R = S = 1$
- Dans le cas d'une bascule RS *NAND* :
  $Q = \overline{\overline{S}.(\overline{R}.Q)} = S + (\overline{R}.Q)$, $\Rightarrow$ Set prioritaire si $R = S = 1$

Mémorisation : les bascules

La bascule RS :
Rappel de la slide précédente :

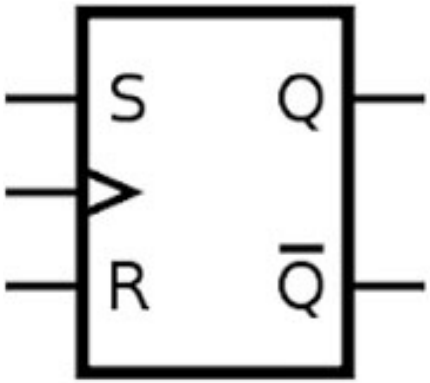
- Dans le cas d'une bascule RS *NOR* :
  $Q = \overline{R + (\overline{S + Q})} = \overline{R}.(S + Q)$
- Dans le cas d'une bascule RS *NAND* :
  $Q = \overline{\overline{S}.(\overline{\overline{R}.Q})} = S + (\overline{R}.Q)$

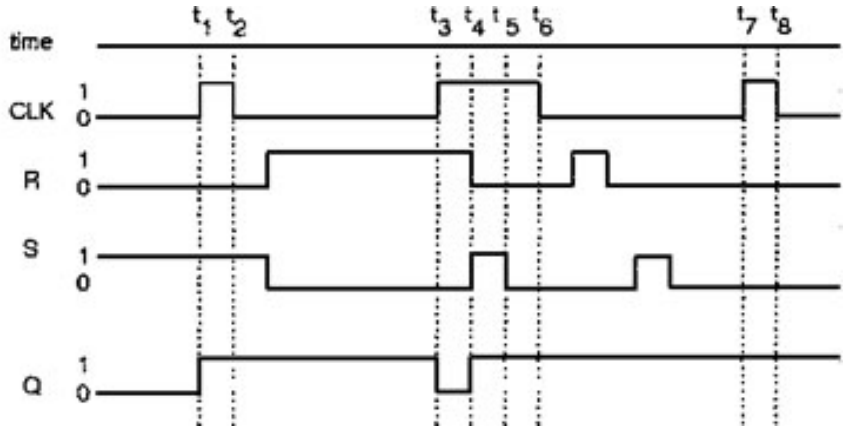
On peut vérifier que pour les deux types de bascule *RS* :

- Quand $R = S = 0$, $Q = Q$
- Quand $R = 1$ et $S = 0$, alors $Q = 0$
- Quand $R = 0$ et $S = 1$, alors $Q = 1$

Mémorisation : les bascules

La bascule RS synchrone :





- La sortie change d'état de manière synchrone dès que les entrées *S* ou *R* sont actives et que l'horloge est active.
- Lorsque les entrées ou l'horloge sont à 0, la sortie mémorise son état précédent.
- Si l'horloge est à 1 et que les 2 entrées *R* et *S* oscillent, alors la sortie oscille.

Mémorisation : les bascules

Une bascule RS synchrone peut être réalisée à l'aide de 2 portes *NOR* et 2 portes *AND* ou de 4 portes *NAND* :

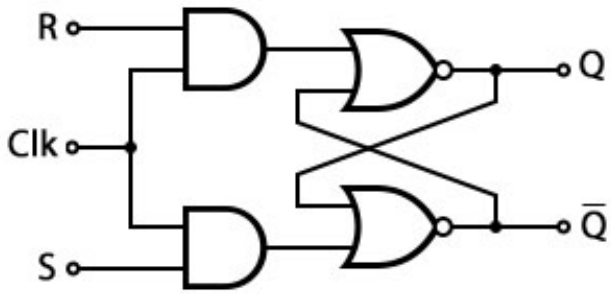


fig: bascule RS synchrone *NOR*

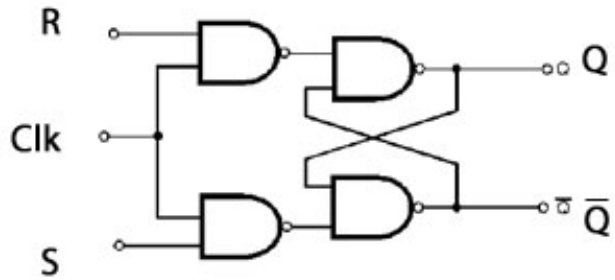


fig: bascule RS synchrone *NAND*

Mémorisation : le verrou (*latch*)

Un verrou ou bascule *D* est destiné à maintenir en sortie l'état d'une entrée à l'instant où le verrou est activé :

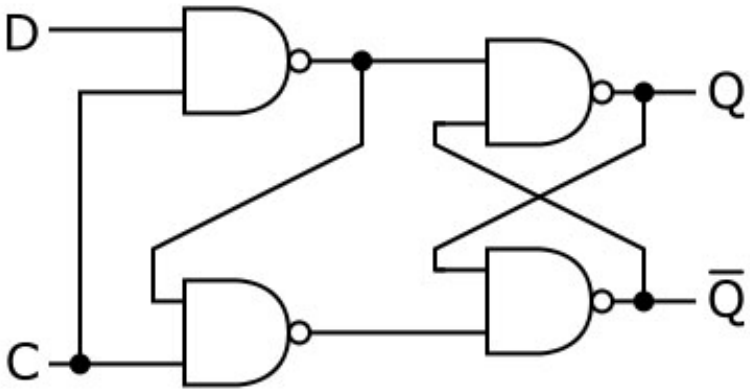


fig: Verrou (Bascule *D*)

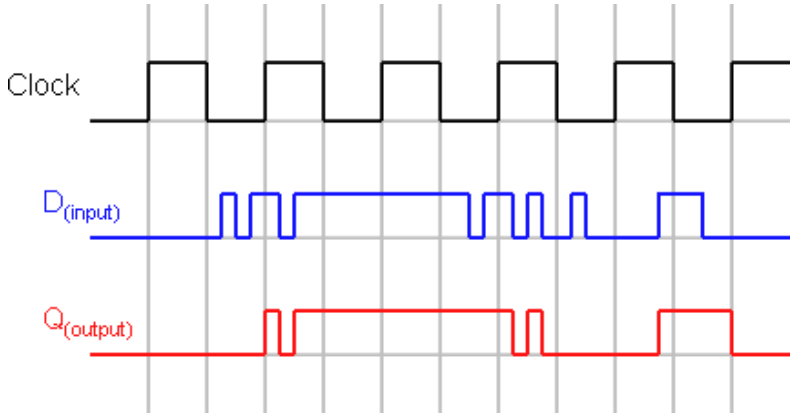


fig: Chronogramme associé

- Un verrou est une bascule RS synchrone ayant un nombre limité d'état possible en entrée.
- Si l'horloge est à 1 et que l'entrées oscille, alors la sortie oscille.

Mémorisation : les bascules fonctionnant sur fronts d'horloge

La bascule JK :

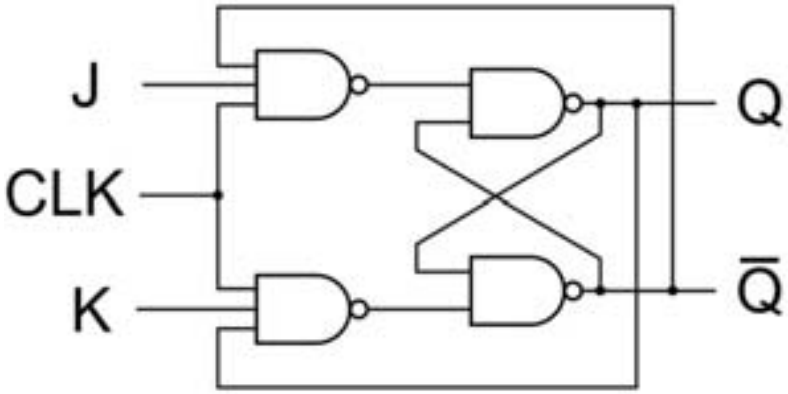


fig: Bascule *JK*

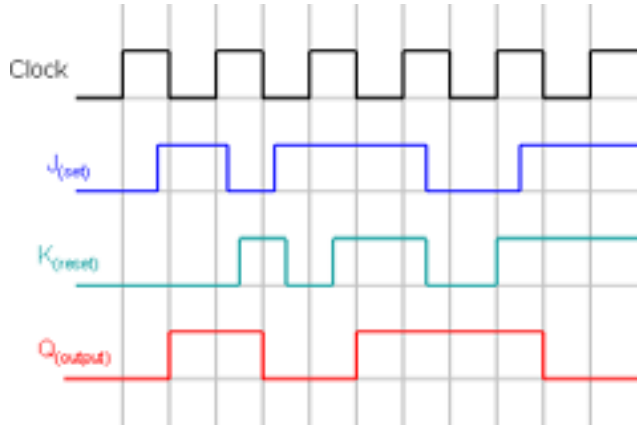


fig: Chronogramme associé

- La sortie ne peut changer d'état que sur les fronts descendants d'horloge.
- Pas d'oscillation possible en sortie : fonctionnement plus déterministe
- Q est inversée si $J = K = 1$ au moment du front d'horloge.

Mémorisation : les bascules fonctionnant sur fronts d'horloge

Le verrou (bascule D) synchrone :

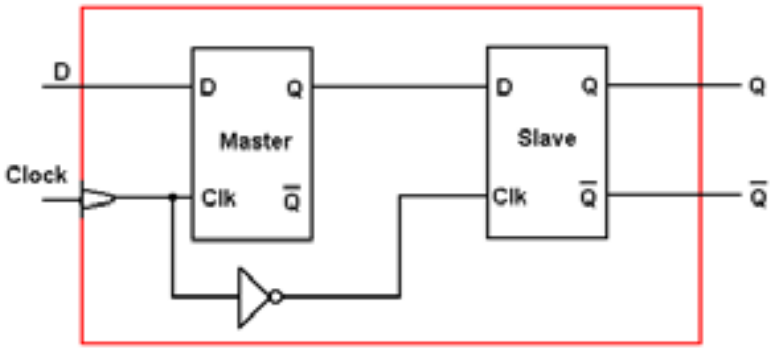


fig: Verrou (Bascule *D*) synchrone

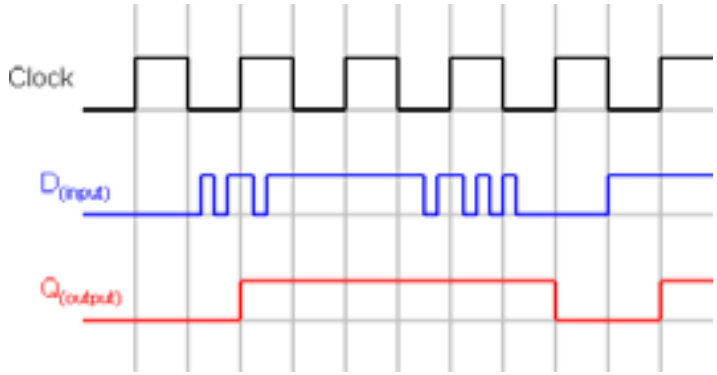


fig: Chronogramme associé

- La sortie ne peut changer d'état que sur les fronts montants d'horloge.
- Pas d'oscillation possible en sortie : fonctionnement plus déterministe
- Idéal pour stocker un bit mémoire.

Mémorisation : les registres

Un registre de mémorisation est un ensemble de bascules *D* synchrones mises en parallèle et permettant de stocker simultanément plusieurs bits :

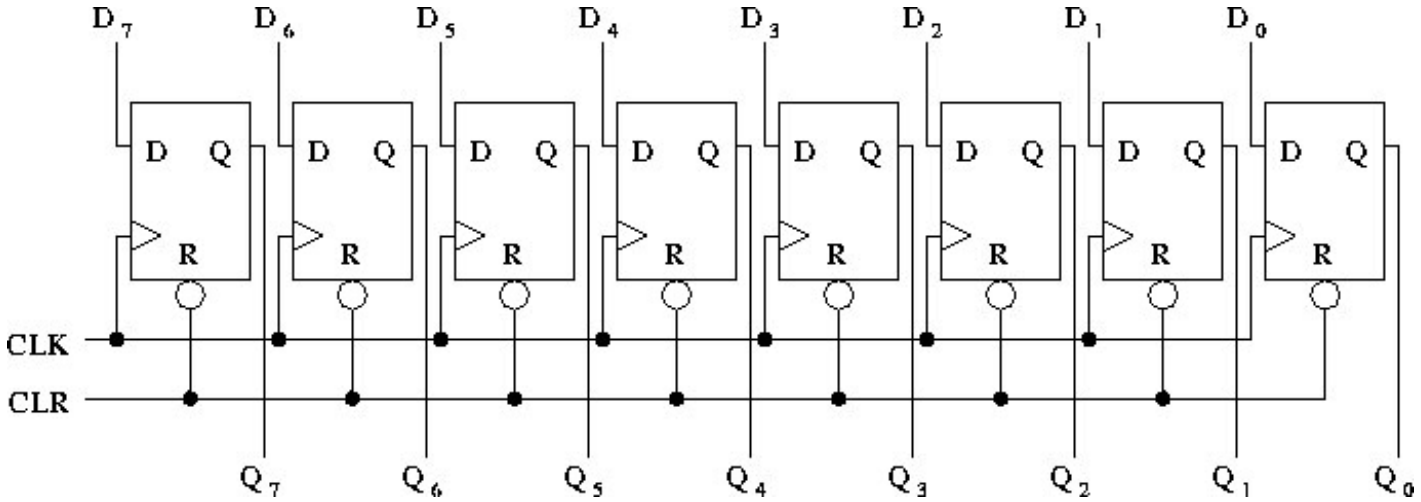


fig: Registre mémoire

- Les registres sont une des structures de base des microcontrôleurs. Ils ont une taille dépendant des bus de données de celui-ci : 8 bits, 16 bits, 32 bits ou 64 bits pour les plus courants.

Transmission et calcul : les registres à décalage

Un registre à décalage (*shift register*) est un ensemble de bascules *D* synchrones mises en série et permettant de propager des bits :

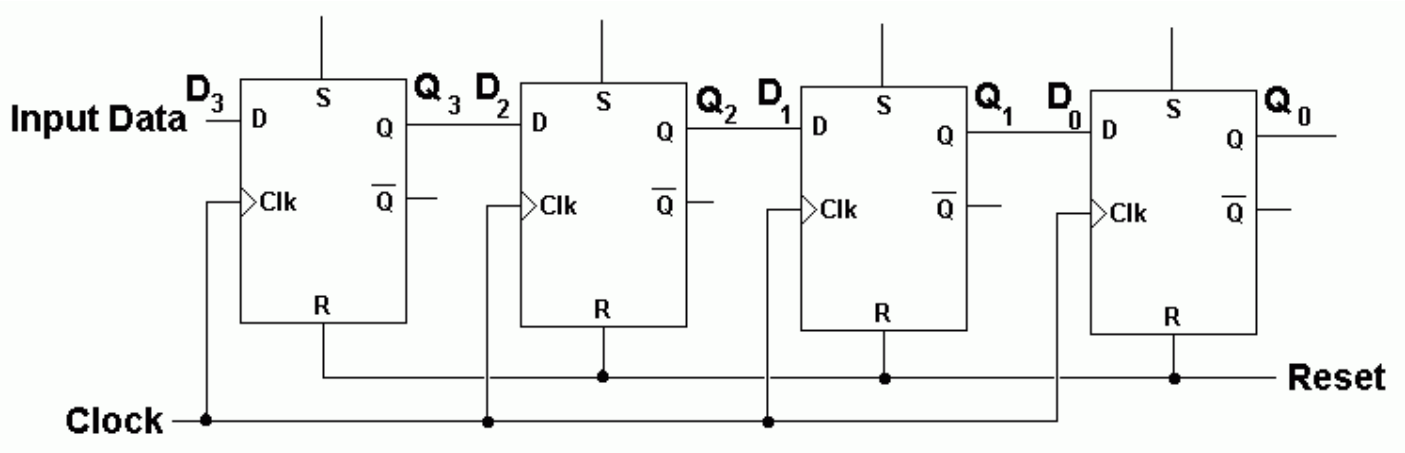


fig: Registre à décalage

- Le registre à décalage permet d'avoir à disposition les données d_t, d_{t-1}, d_{t-2}, d_{t-3} jusqu'à d_{t-n}, n étant la profondeur du registre.
- Il est utilisé pour des calculs tels que les produits scalaires utilisant des données passées. C'est un des éléments de base des *DSP*.

Transmission de données : les registres parallèle/série

- **Un registre à écriture parallèle et lecture série** permet d'envoyer des octets en provenance d'un bus de données 8 bits sur une liaison mono-filaire série.
- **Un registre à écriture série et lecture parallèle** permet de récupérer des octets en provenance d'une laison série mono-filaire pour les lire sur un bus de données 8 bits.

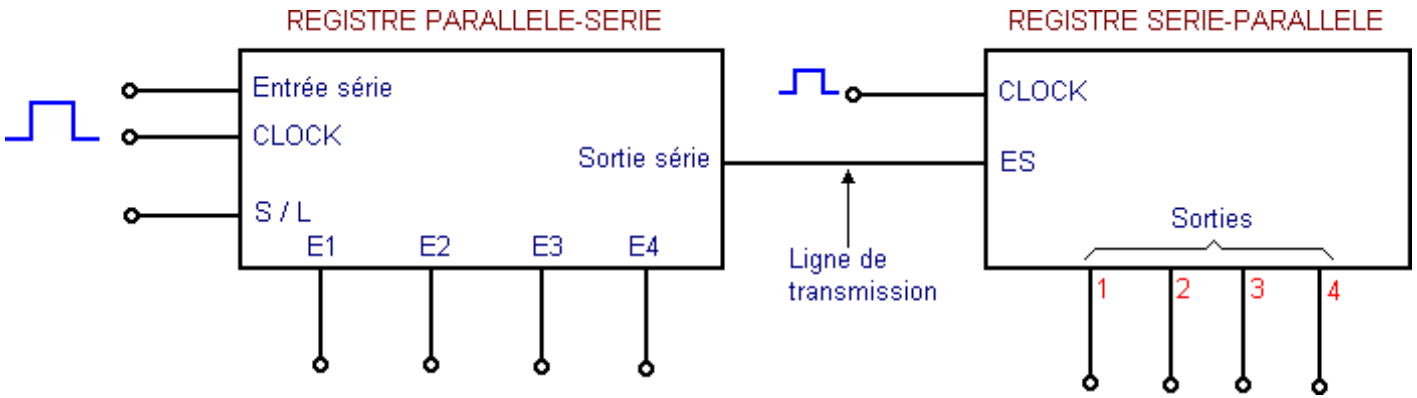


fig: Registres parallèle/série et série/parallèle

Compteurs synchrones

Les compteurs synchrones sont un autre élément majeur des microcontrôleurs : couplés à des comparateurs, ils permettent de réaliser des temporisations appelées *timers*.

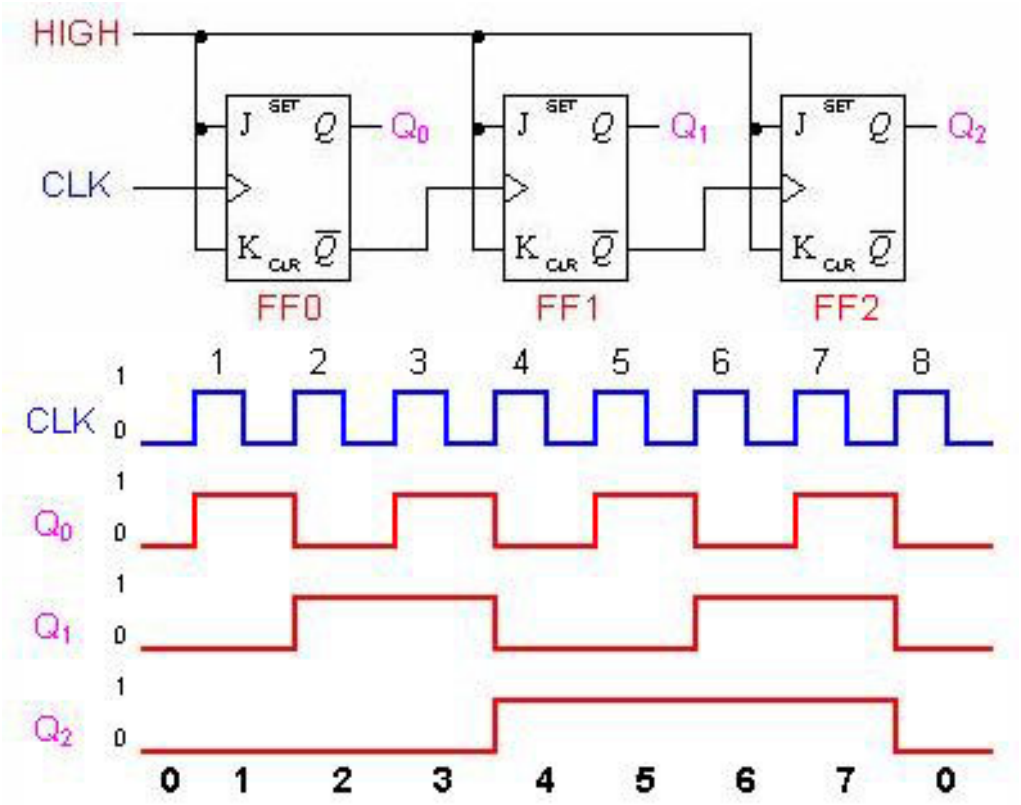


fig: Compteur synchrone réalisé à base de bascules JK

Conclusion

La logique combinatoire et séquentielle permettent de réaliser les fonctions essentielles suivantes :

- Opérateurs logiques booléens et de calcul élémentaire : utilisé dans les ALU des microcontrôleursμC.
- Registres de mémorisation : utilisés dans les μC.
- Registres à décalage : utilisés dans les périphériques de transmission numérique des μC.
- Compteurs : utilisés dans les timers de μC.

L'ensemble des fonctions vues dans ce cours permettent de réaliser les éléments de base d'un microcontrôleur.
Dans la suite, et avant de détailler le μC, nous allons voir comment ces fonctions sont réalisées du point de vue électronique.

Questions ?

- Questions
- Contact : vgies@hotmail.com