Masques, bits, logique et arithmétique

J.-M Friedt

FEMTO-ST/département temps-fréquence

jmfriedt@femto-st.fr

transparents à `jmfriedt.free.fr`

27 février 2021

Affichage de valeurs numériques : hexadécimal

On suppose avoir une fonction qui affiche une chaîne de caractères.
Quelle est la définition d'une chaîne de caractères en C ?
Comment en connaître la fin ? [1]
On veut afficher en hexadécimal des nombres codés sur 8, 16 et 32 bits.
Approche naïve :

```
#include <stdio.h>
int main()
{char c[256];
 int32_t a=10;
 sprintf(c,"%d",a);
}
```

donne un exécutable qui occupe 1840 octets de programme

```
$ avr-gcc -mmcu=atmega32u4 source.c
$ avr-objcopy -Obinary a.out a.bin
-rwxr-xr-x 1 jmfriedt jmfriedt  1840 Jan 13 16:54 a.bin
```

1. voir `affiche_chaine.c`

Remplacer sprintf() – 8 bits

Soit un entier sur 8 bits `c` : remplir la chaine de caractères `buffer` avec les chiffres représentant `c` en hexadécimal (utiliser masques et divisions : 5 lignes)

`man ascii`

	2	3	4	5	6	7
0:		0	@	P	‘	p
1:	!	1	A	Q	a	q
2:	"	2	B	R	b	r
3:	#	3	C	S	c	s
4:	$	4	D	T	d	t
5:	%	5	E	U	e	u
6:	&	6	F	V	f	v
7:	’	7	G	W	g	w
8:	(	8	H	X	h	x
9:	)	9	I	Y	i	y
A:	*	:	J	Z	j	z
B:	+	;	K	[	k	{
C:	,	<	L	\	l	|
D:	-	=	M	]	m	}
E:	.	>	N	^	n	~
F:	/	?	O	_	o	DEL

	30	40	50	60	70	80	90	100	110	120
0:		(	2	<	F	P	Z	d	n	x
1:		)	3	=	G	Q	[	e	o	y
2:		*	4	>	H	R	\	f	p	z
3:	!	+	5	?	I	S	]	g	q	{
4:	"	,	6	@	J	T	^	h	r	|
5:	#	-	7	A	K	U	_	i	s	}
6:	$	.	8	B	L	V	‘	j	t	~
7:	%	/	9	C	M	W	a	k	u	DEL
8:	&	0	:	D	N	X	b	l	v	
9:	’	1	;	E	O	Y	c	m	w	

Remplacer sprintf() – 8 bits

Soit un entier sur 8 bits `c` : remplir la chaine de caractères `t` avec les chiffres représentant `c` en hexadécimal

```
#ifdef PC
#include <stdio.h>
#endif

void c(char c,char *t)
{t[0]=((c&0xf0)>>4)+'0';if (t[0]>'9') t[0]+=7;
 t[1]=(c&0x0f)+'0';     if (t[1]>'9') t[1]+=7;
 t[2]=0;
}

int main()
{char t[256];
 char cc=0x42;
 c(cc,t);
#ifdef PC
 printf("%s\n",t);
#else
 affiche(t);
#endif
}
```

```
$ avr-gcc -mmcu=atmega32u4 source.c
$ avr-objcopy -Obinary a.out a.bin
-rwxr-xr-x 1 jmfriedt jmfriedt 430 Jan 14 13:53 a.bin
```

Code compilable sur AVR **et PC** : `gcc -DPC fichier.c`

Remplacer sprintf() – 16 bits

Soit un entier sur 16 bits `s` : remplir la chaine de caractères `buffer` avec les chiffres représentant `s` en hexadécimal (2 lignes)

Remplacer sprintf() – 16 bits

Soit un entier sur 16 bits `s` : remplir la chaine de caractères `t` avec les chiffres représentant `s` en hexadécimal

```
#ifdef PC
#include <stdio.h>
#define affiche printf
#else
#define affiche
#endif

// bien penser a finir par l'octet de poids faible (d'@ la plus forte) car on
// ajoute \0 a la fin de la chaine
void c(char c,char *t)
{t[0]=((c&0x0f0)>>4)+'0';if (t[0]>'9') t[0]+=7;
 t[1]=(c&0x0f)+'0';      if (t[1]>'9') t[1]+=7;
 t[2]=0;
}

void s(short s,char *t)
{c((s&0xff00)>>8,&t[0]);
 c(s&0xff,&t[2]);
}

int main()
{char t[256];
 char cc=0x42;
 short ss=0x5678;
 long ll=0x12345678;
 c(cc,t);affiche("%s\n",t);
 s(ss,t);affiche("%s\n",t);
}
```

Remplacer sprintf() – 32 bits

Soit un entier sur 32 bits `l` : remplir la chaine de caractères `buffer` avec les chiffres représentant `l` en hexadécimal (2 lignes)

Remplacer sprintf() – 32 bits

Soit un entier sur 32 bits `l` : remplir la chaine de caractères `t` avec les chiffres représentant `l` en hexadécimal

```
#ifdef PC
#include <stdio.h>
#define affiche printf
#else
#define affiche
#endif

void c(char c,char *t)
{t[0]=((c&0xf0)>>4)+'0';if (t[0]>'9') t[0]+=7;
 t[1]=(c&0x0f)+'0';      if (t[1]>'9') t[1]+=7;
 t[2]=0;
}

void s(short s,char *t)
{c((s&0xff00)>>8,&t[0]);
 c(s&0xff,&t[2]);
}

void l(long l,char *t)
{s((l&0xffff0000)>>16,&t[0]); // idem que t
 s(l&0xffff,&t[4]);
}

int main()
{char t[256];
 char cc=0x42;
 short ss=0x5678;
 long ll=0x12345678;
 c(cc,t);affiche("%s\n",t);
 s(ss,t);affiche("%s\n",t);
 l(ll,t);affiche("%s\n",t);
}
```

-rwxr-xr-x 1 jmfriedt jmfriedt 662 Jan 13 17 :08 a.bin

Échanger deux variables

Proposer une fonction qui prend en argument deux entiers `a` et `b`, et intervertit leur contenu

Échanger deux variables

Proposer une fonction qui prend en argument deux entiers `a` et `b`, et intervertit leur contenu

Est-ce que ce programme fonctionne ?

```
#include <stdio.h>

void f(int a,int b)
{int tmp;
 printf("%d %d -> ",a,b);
 tmp=a; a=b; b=tmp;
 printf("%d %d\n",a,b);
}

int main()
{int a=1,b=2;
 printf("%d %d\n",a,b);f(a,b);printf("%d %d\n",a,b);
}
```

Échanger deux variables

Proposer une fonction qui prend en argument deux entiers `a` et `b`, et intervertit leur contenu

Est-ce que ce programme fonctionne ?

```
#include <stdio.h>

void f(int a,int b)
{int tmp;
 printf("%d %d -> ",a,b);
 tmp=a; a=b; b=tmp;
 printf("%d %d\n",a,b);
}

int main()
{int a=1,b=2;
 printf("%d %d\n",a,b);f(a,b);printf("%d %d\n",a,b);
}
```

donne

```
1 2
1 2 -> 2 1
1 2
```

Passage par valeurs et non passage par pointeur ⇒ `a` et `b` du programme principal sont inchangées

Échanger deux variables

Proposer une fonction qui prend en argument deux entiers `a` et `b`, et intervertit leur contenu

```
#include <stdio.h>

void f(int *a,int *b)
{int tmp;
 printf("%d %d -> ",*a,*b);
 tmp=*a; *a=*b; *b=tmp;
 printf("%d %d\n",*a,*b);
}

int main()
{int a=1,b=2;
 printf("%d %d\n",a,b);f(&a,&b);printf("%d %d\n",a,b);
}
```

donne

```
1 2
1 2 -> 2 1
2 1
```

Passage par pointeur $\Rightarrow$ `a` et `b` du programme principal sont modifiées

Échanger deux variables sans variable intermédiaire

Proposer une solution pour échanger `a` et `b` sans passer par une variable intermédiaire `tmp`

Échanger deux variables sans variable intermédiaire

Proposer une solution pour échanger `a` et `b` sans passer par une variable intermédiaire `tmp`

```
int main()
{int a=1,b=2;
 printf("%d %d\n",a,b);
 a^=b;
 b^=a;
 a^=b;
 printf("%d %d\n",a,b);
}
```

En effet :

1. a=0, b=0 : XOR(0,0)=0 donc on garde toujours 0 ;
2. a=1, b=1 : a=XOR(1,1)=0 → b=XOR(0,1)=1 → a=XOR(0,1)=1
3. a=0, b=1 : a=XOR(0,1)=1 → b=XOR(1,1)=0 → a=XOR(1,0)=1
4. a=1, b=0 : a=XOR(1,0)=1 → b=XOR(0,1)=1 → a=XOR(1,1)=0

valide pour chaque bit, donc valide pour tous les bits. CQFD

Démonstration :

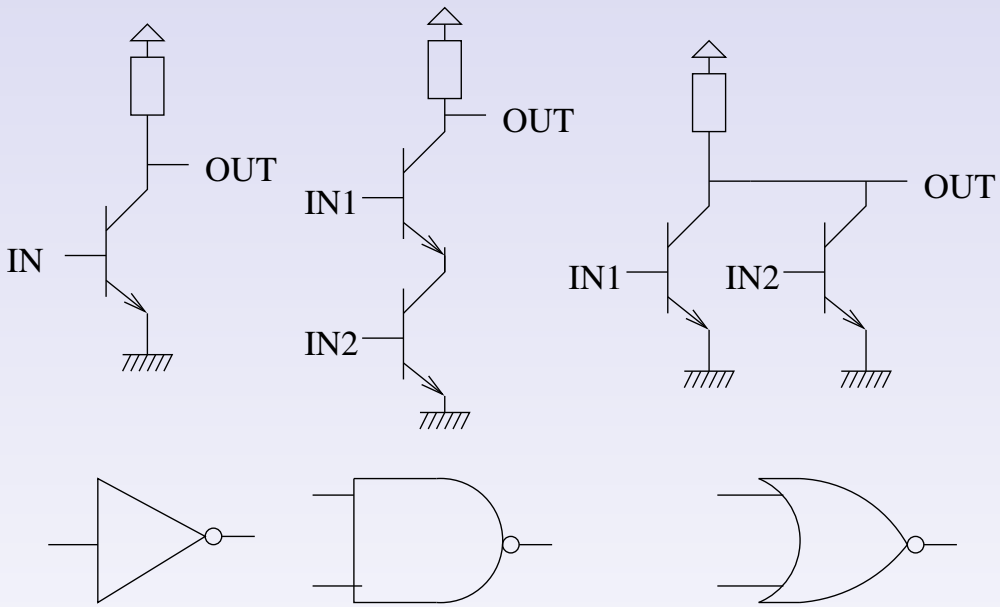
XOR est associatif ((a^b)^c=a^(b^c)) et commutatif (a^b=b^a)

1. a=a^b;b=b^a;a=a^b; se réduit (1er terme) en ...
2. b=b^(a^b) et a=(a^b)^(b^(a^b));
3. b=b^(b^a); en utilisant la commutativité (1er terme)
4. b=(b^b)^a; en utilisant l'associativité (1er terme)
5. b=(0)^a; en utilisant l'identité a^0=a (1er terme)
6. b=a; en utilisant l'identité a^0=a (1er terme)
7. a=(b^b)^(a^a)^b; devient a=b; pour le deuxième terme

Économie d'un octet au détriment des opérations logiques additionnelles

Transistors et logique

Exemple avec transistors NPN ($I_{CE} > 0$ si $V_B > 0$)



Logique → arithmétique : XOR

XOR répond à la même logique que la somme binaire
$(0+0=0,\ 1+0=1,\ 1+1=0^1)$
Comment synthétiser la table logique de XOR à partir des portes vues auparavant

Rappel : théorème de DeMorgan

$$\boxed{\overline{A \cdot B} = \overline{A} + \overline{B} \text{ et } \overline{A + B} = \overline{A} \cdot \overline{B}}$$

avec $\cdot$ la fonction ET et $+$ la fonction OU

Logique → arithmétique : XOR

XOR répond à la même logique que la somme binaire
$(0+0=0,\ 1+0=1,\ 1+1=0^1)$
Comment synthétiser la table logique de XOR à partir des portes vues auparavant

$$\begin{array}{c|cc} \wedge & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 1 & 0 \end{array} = \text{NOT}\left(\begin{array}{c|cc} ? & 0 & 1 \\ \hline 0 & 1 & 0 \\ 1 & 0 & 1 \end{array}\right) =$$

$$\text{NOT}\left(\left(\begin{array}{c|cc} \& & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \end{array}\right) \text{OR} \left(\begin{array}{c|cc} \&(\text{NOT},\text{NOT}) & 0 & 1 \\ \hline 0 & 1 & 0 \\ 1 & 0 & 0 \end{array}\right)\right)$$

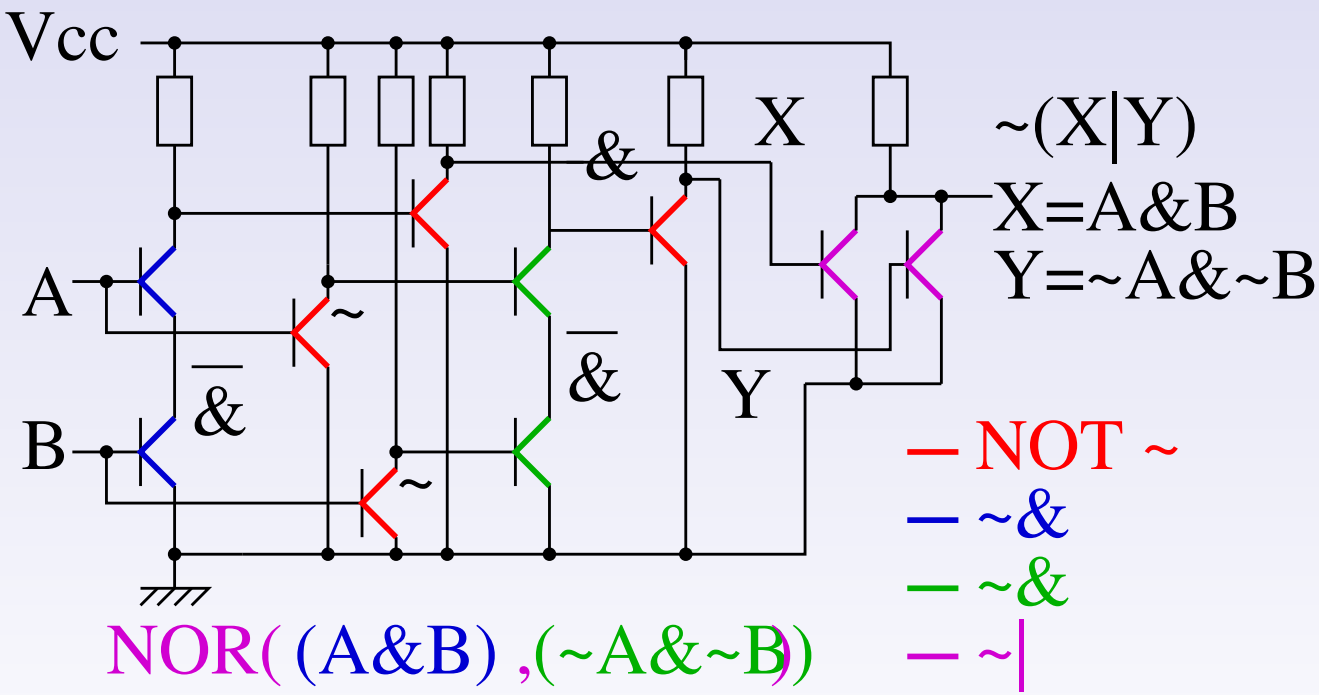
$$= \text{NOR}((A\&B),(\overline{A}\&\overline{B}))$$

En d'autres termes, $\overline{(A\cdot B)+\overline{A}\cdot\overline{B}}$ qui devient
$\overline{A\cdot B}\cdot\overline{\overline{A}\cdot\overline{B}} = (\overline{A}+\overline{B})\cdot(\overline{\overline{A}}+\overline{\overline{B}}) = {}^2\overline{A}\cdot A+\overline{A}\cdot B+\overline{B}\cdot A+\overline{B}\cdot B = {}^3\overline{A}\cdot B+\overline{B}\cdot A$

2. $\overline{\overline{X}} = X$
3. $\overline{A}\cdot A = 0$

Logique → arithmétique : XOR

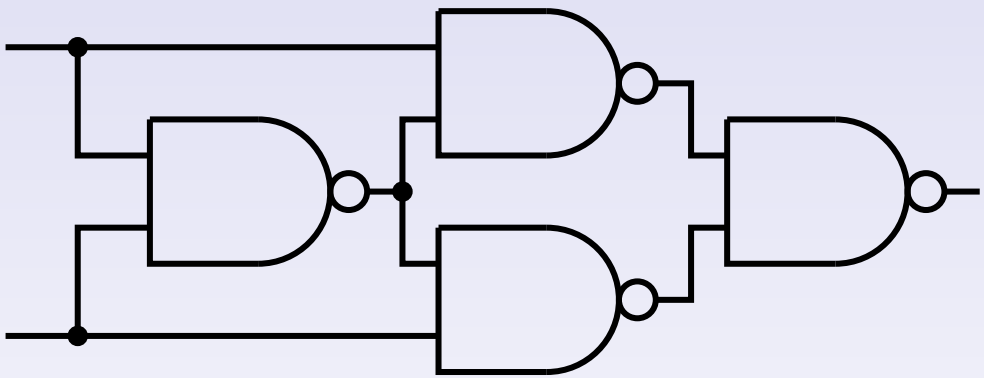
XOR répond à la même logique que la somme binaire
$(0+0=0,\ 1+0=1,\ 1+1=0^1)$
Comment synthétiser la table logique de XOR à partir des portes vues auparavant



Solution à 10 transistors

Logique → arithmétique : XOR

XOR répond à la même logique que la somme binaire
$(0+0=0,\ 1+0=1,\ 1+1=0^1)$
Comment synthétiser la table logique de XOR à partir des portes vues auparavant



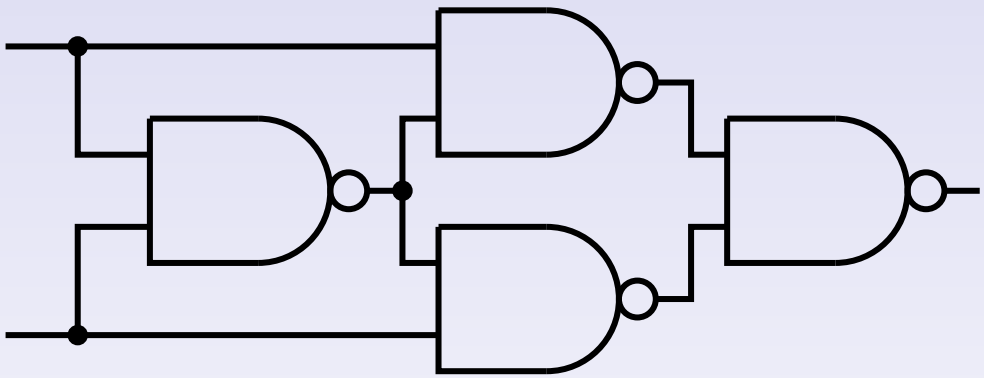
Solution à 8 transistors puisque

$$\overline{A\cdot(\overline{A\cdot B})}\cdot\overline{B\cdot(\overline{A\cdot B})}=\overline{A\cdot(\overline{A\cdot B})}+\overline{B\cdot(\overline{A\cdot B})}=\ ^2A\cdot(\overline{A\cdot B})+B\cdot(\overline{A\cdot B})$$

2. $(\overline{\overline{X}}=X)$

Logique → arithmétique : XOR

XOR répond à la même logique que la somme binaire
$(0+0=0,\ 1+0=1,\ 1+1=0^{1})$
Comment synthétiser la table logique de XOR à partir des portes vues auparavant



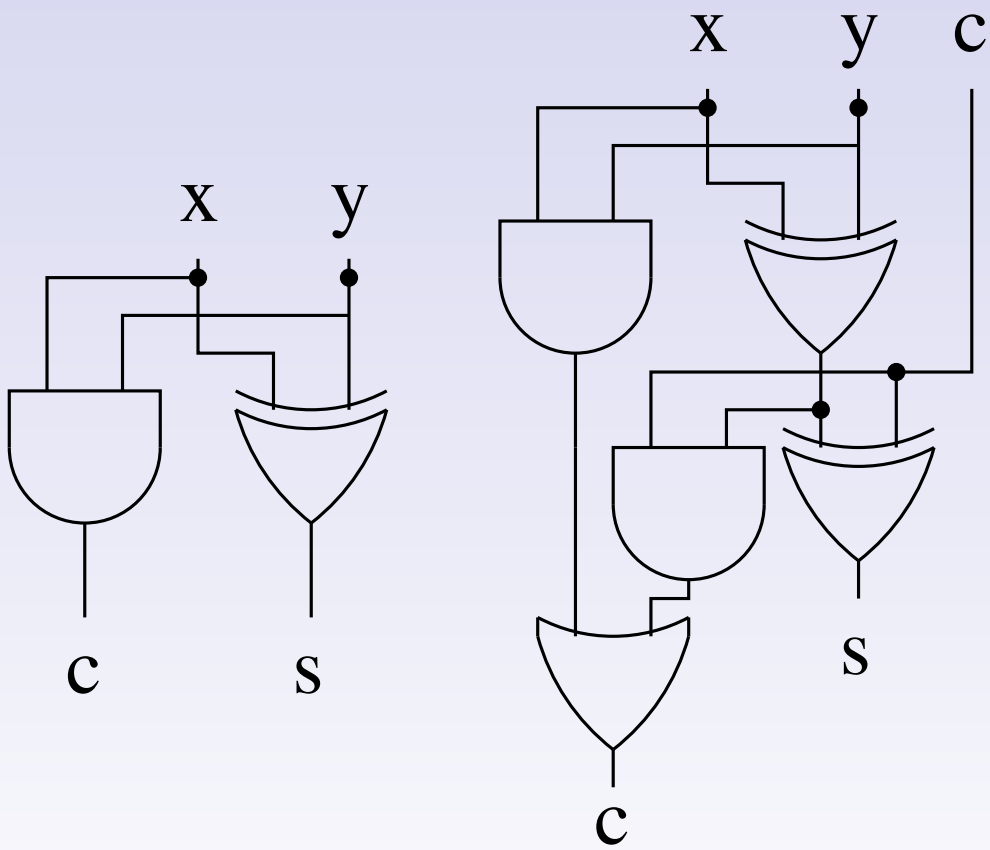
Solution à 8 transistors puisque

$$\overline{\overline{A\cdot(\overline{A\cdot B})}\cdot\overline{B\cdot(\overline{A\cdot B})}}=\overline{\overline{A\cdot(\overline{A\cdot B})}}+\overline{\overline{B\cdot(\overline{A\cdot B})}}=\ ^{2}A\cdot(\overline{A\cdot B})+B\cdot(\overline{A\cdot B})=$$
$$A\cdot(\overline{A}+\overline{B})+B\cdot(\overline{A}+\overline{B})=A\cdot\overline{A}+A\cdot\overline{B}+B\cdot\overline{A}+B\cdot\overline{B}=\ ^{3}A\cdot\overline{B}+B\cdot\overline{A}\quad\text{CQFD}$$

2. $(\overline{\overline{X}}=X)$
3. $X\cdot\overline{X}=0$

Transistors et arithmétique

Half adder (XOR puis retenu par AND) et full adder (XOR avec retenue)



c est carry, s est somme

Multiplication

Comment multiplier par 15 ?

Multiplication

Comment multiplier par 15 ?

*15=*16-1 et *16 s'écrit <<4.

Démonstration avec gcc pour processeur ARM :

```
int main()
{volatile int d,c=42;
 d=c*15;
}
```

devient par

```
arm-none-eabi-gcc -O2 -c t.c
arm-none-eabi-objdump -dSt t.o
```

la séquence d'opcodes

```
  14:   e0633203        rsb     r3, r3, r3, lsl #4
  18:   e58d3000        str     r3, [sp]
```

qui décalle de 4 vers la gauche (multiplication par 16) et soustraction de r3 (*16-1)

Exercice : comment multiplier par 10 ? par 20 ?

Exercice : comment est-ce que `gcc` **compile**

```
for (k=0;k<(20*10);k++) {...}
```

?