Annexe_TD

TABLEAU ALPHABETIQUE DES INSTRUCTIONSASSEMBLEUR HC11

Mnemonic	Opération	Description	Mode adressage	Code machine complet		taille	Cycles
				Code Mnémonic	Opérande		
ABA	Add A et B	A + B → A	INH	1B		1	2
ABX	Add B to X	IX + (00 : B) → IX	INH	3A		1	3
ABY	Add B to Y	IY + (00 : B) →IY	INH	18 3A		2	4
ADCA	Add with Carry to A	A + M + C → A	A IMM	89	ii	2	2
			A DIR	99	dd	2	3
			A EXT	B9	hh ll	3	4
			A IND,X	A9	ff	2	4
			A IND,Y	18 A9	ff	3	5
ADCB	Add with Carry to B	B + M + C → B	B IMM	C9	ii	2	2
			B DIR	D9	dd	2	3
			B EXT	F9	hh ll	3	4
			B IND,X	E9	ff	2	4
			B IND,Y	18 E9	ff	3	5
ADDA	Add Memory to A	A + M → A	A IMM	8B	ii	2	2
			A DIR	9B	dd	2	3
			A EXT	BB	hh ll	3	4
			A IND,X	AB	ff	2	4
			A IND,Y	18 AB	ff	3	5

Mnemonic	Opération	Description	Mode adressage	Code machine complet		taille	Cycles
				Code Mnémonic	Opérande		
ADDB	Add Memory to B	B + M → B	B IMM	CB	ii	2	2
			B DIR	DB	dd	2	3
			B EXT	FB	hh ll	3	4
			B IND,X	EB	ff	2	4
			B IND,Y	18 EB	ff	3	5
ADDD	Add 16-Bit to D	D + (M : M + 1) → D	IMM	C3	ii kk	3	4
			DIR	D3	dd	2	5
			EXT	F3	hh ll	3	6
			IND,X	E3	ff	2	6
			IND,Y	18 E3	ff	3	7
ANDA	AND A with Memory	A • M → A	A IMM	84	ii	2	2
			A DIR	94	dd	2	3
			A EXT	B4	hh ll	3	4
			A IND,X	A4	ff	2	4
			A IND,Y	18 A4	ff	3	5
ANDB	AND B with Memory	B • M → B	B IMM	C4	ii	2	2
			B DIR	D4	dd	2	3
			B EXT	F4	hh ll	3	4
			B IND,X	E4	ff	2	4
			B IND,Y	18 E4	ff	3	5

Mnemonic	Opération	Description	Mode adressage	Code machine complet		taille	Cycles
				Code Mnémonic	Opérande		
ASL	Arithmetic Shift Left		EXT	78	hh ll	3	6
			IND,X	68	ff	2	6
			IND,Y	18 68	ff	3	7
ASLA	Arithmetic Shift Left A		A INH	48		1	2
ASLB	Arithmetic Shift Left B		B INH	58		1	2
ASLD	Arithmetic Shift Left D		INH	05		1	3
ASR	Arithmetic Shift Right		EXT	77	hh ll	3	6
			IND,X	67	ff	2	6
			IND,Y	18 67	ff	3	7
ASRA	Arithmetic Shift Right A		A INH	47		1	2
ASRB	Arithmetic Shift Right B		B INH	57		1	2
BCC	Branch if Carry Clear	? C = 0	REL	24	rr	2	3
BCLR	Clear Bit(s)	M • (mm) → M	DIR	15	dd mm	3	6
			IND,X	1D	ff mm	3	7
			IND,Y	18 1D	ff mm	4	8
BCS	Branch if Carry Set	? C = 1	REL	25	rr	2	3
BEQ	Branch if = Zero	? Z = 1	REL	27	rr	2	3
BGE	Branch if ≥ Zero	? N ⊕ V = 0	REL	2C	rr	2	3
BGT	Branch if> Zero	? Z + (N ⊕ V) = 0	REL	2E	rr	2	3

Mnemonic	Opération	Description	Mode adressage	Code machine complet		taille	Cycles
				Code Mnémonic	Opérande		
BHI	Branch if Higher	? C + Z = 0	REL	22	rr	2	3
BHS	Branch if Higher or Same	? C = 0	REL	24	rr	2	3
BITA	Bit(s) Test A with Memory	A • M	A IMM	85	ii	2	2
			A DIR	95	dd	2	3
			A EXT	B5	hh ll	3	4
			A IND,X	A5	ff	2	4
			A IND,Y	18 A5	ff	3	5
BITB	Bit(s) Test B with Memory	B • M	B IMM	C5	ii	2	2
			B DIR	D5	dd	2	3
			B EXT	F5	hh ll	3	4
			B IND,X	E5	ff	2	4
			B IND,Y	18 E5	ff	3	5
BLE	Branch if A Zero	? Z + (N ⊕ V) = 1	REL	2F	rr	2	3
BLO	Branch if Lower	? C = 1	REL	25	rr	2	3
BLS	Branch if Lower or Same	? C + Z = 1	REL	23	rr	2	3
BLT	Branch if < Zero	? N ⊕ V = 1	REL	2D	rr	2	3
BMI	Branch if Minus	? N = 1	REL	2B	rr	2	3
BNE	Branch if not = Zero	? Z = 0	REL	26	rr	2	3
BPL	Branch if Plus	? N = 0	REL	2A	rr	2	3
BRA	Branch Always	? 1 = 1	REL	20	rr	2	3
BRCLR	Branch if Bit(s) Clear	? M • mm = 0	DIR	13	dd mm rr	4	6
			IND,X	1F	ff mm rr	4	7
			IND,Y	18 1F	ff mm rr	5	8

Mnemonic	Opération	Description	Mode adressage	Code machine complet		taille	Cycles
				Code Mnémonic	Opérande		
BRN	Branch Never	? 1 = 0	REL	21	rr	2	3
BRSET	Branch if Bit(s) Set	? ($\overline{M}$) • mm = 0	DIR	12	dd mm rr	4	6
			IND,X	1E	ff mm rr	4	7
			IND,Y	18 1E	ff mm rr	5	8
BSET	Set Bit(s)	M + mm M	DIR	14	dd mm	3	6
			IND,X	1C	ff mm	3	7
			IND,Y	18 1C	ff mm	4	8
BSR	Branch to Subroutine	Appel module en relatif	REL	8D	rr	2	6
BVC	Branch if Overflow Clear	? V = 0	REL	28	rr	2	3
BVS	Branch if Overflow Set	? V = 1	REL	29	rr	2	3
CBA	Compare A to B	A – B	INH	11		1	2
CLC	Clear Carry Bit	0 → C	INH	0C		1	2
CLI	Clear Interrupt Mask	0 → I	INH	0E		1	2
CLR	Clear Memory Byte	0 → M	EXT	7F	hh ll	3	6
			IND,X	6F	ff	2	6
			IND,Y	18 6F	ff	3	7
CLRA	Clear Accumulator A	0 → A	A INH	4F		1	2
CLRB	Clear Accumulator B	0 → B	B INH	5F		1	2
CLV	Clear Overflow Flag	0 → V	INH	0A		1	2

Mnemonic	Opération	Description	Mode adressage	Code machine complet		taille	Cycles
				Code Mnémonic	Opérande		
CMPA	Compare A to Memory	A – M	A IMM	81	ii	2	2
			A DIR	91	dd	2	3
			A EXT	B1	hh ll	3	4
			A IND,X	A1	ff	2	4
			A IND,Y	18 A1	ff	3	5
CMPB	Compare B to Memory	B – M	B IMM	C1	ii	2	2
			B DIR	D1	dd	2	3
			B EXT	F1	hh ll	3	4
			B IND,X	E1	ff	2	4
			B IND,Y	18 E1	ff	3	5
COM	Ones Complement Memory Byte	$FF – M → M	EXT	73	hh ll	3	6
			IND,X	63	ff	2	6
			IND,Y	18 63	ff	3	7
COMA	Ones Complement A	$FF – A → A	A INH	43		1	2
COMB	Ones Complement B	$FF – B → B	B INH	53		1	2
CPD	Compare D to Memory 16-Bit	D – M : M + 1	IMM	1A 83	ii kk	4	5
			DIR	1A 93	dd	3	6
			EXT	1A B3	hh ll	4	7
			IND,X	1A A3	ff	3	7
			IND,Y	CD A3	ff	3	7
CPX	Compare X to Memory 16-Bit	IX – M : M + 1	IMM	8C	ii kk	3	4
			DIR	9C	dd	2	5
			EXT	BC	hh ll	3	6
			IND,X	AC	ff	2	6
			IND,Y	CD AC	ff	3	7
CPY	Compare Y to Memory 16-Bit	IY – M : M + 1	IMM	18 8C	ii kk	4	5
			DIR	18 9C	dd	3	6
			EXT	18 BC	hh ll	4	7
			IND,X	1A AC	ff	3	7
			IND,Y	18 AC	ff	3	7
DAA	Decimal Adjust A	Adjust Sum to BCD	INH	19		1	2
DEC	Decrement Memory Byte	M – 1 → M	EXT	7A	hh ll	3	6
			IND,X	6A	ff	2	6
			IND,Y	18 6A	ff	3	7

Mnemonic	Opération	Description	Mode adressage	Code machine complet		taille	Cycles
				Code Mnémonic	Opérande		
DECA	Decrement Accumulator A	A – 1 → A	A INH	4A		1	2
DECB	Decrement Accumulator B	B – 1 → B	B INH	5A		1	2
DES	Decrement Stack Pointer	SP – 1 → SP	INH	34			3
DEX	Decrement Index Register X	IX – 1 → IX	INH	09		1	3
DEY	Decrement Index Register Y	IY – 1 → IY	INH	18 09		2	4
EORA	Exclusive OR A with Memory	A ⊕ M → A	A IMM	88	ii	2	2
			A DIR	98	dd	2	3
			A EXT	B8	hh ll	3	4
			A IND,X	A8	ff	2	4
			A IND,Y	18 A8	ff	3	5
EORB	Exclusive OR B with Memory	B ⊕ M→ B	B IMM	C8	ii	2	2
			B DIR	D8	dd	2	3
			B EXT	F8	hh ll	3	4
			B IND,X	E8	ff	2	4
			B IND,Y	18 E8	ff	3	5
FDIV	Fractional Divide 16 by 16	D / IX → IX; r → D	INH	03		1	41
IDIV	Integer Divide 16 by 16	D / IX → IX; r → D	INH	02		1	41
INC	Increment Memory Byte	M + 1 → M	EXT	7C	hh ll	3	6
			IND,X	6C	ff	2	6
			IND,Y	18 6C	ff	3	7
INCA	Increment Accumulator A	A + 1 → A	A INH	4C		1	2
INCB	Increment	B + 1 → B	B INH	5C		1	2
	Accumulator						
	B						

Mnemonic	Opération	Description	Mode adressage	Code machine complet		taille	Cycles
				Code Mnémonic	Opérande		
INS	Increment Stack Pointer	SP + 1 ⇒ SP	INH	31		1	3
INX	Increment Index Register X	IX + 1 ⇒ IX	INH	08		1	3
INY	Increment Index Register Y	IY + 1 ⇒ IY	INH	18 08		2	4
JMP	Jump	Saut absolu inconditionnel	EXT	7E	hh ll	3	3
			IND,X	6E	ff	2	3
			IND,Y	18 6E	ff	3	4
JSR	Jump to Subroutine	Appel module en absolu	DIR	9D	dd	2	5
			EXT	BD	hh ll	3	6
			IND,X	AD	ff	2	6
			IND,Y	18 AD	ff	3	7
LDAA	Load Accumulator A	M ⇒ A	A IMM	86	ii	2	2
			A DIR	96	dd	2	3
			A EXT	B6	hh ll	3	4
			A IND,X	A6	ff	2	4
			A IND,Y	18 A6	ff	3	5
LDAB	Load Accumulator B	M ⇒ B	B IMM	C6	ii	2	2
			B DIR	D6	dd	2	3
			B EXT	F6	hh ll	3	4
			B IND,X	E6	ff	2	4
			B IND,Y	18 E6	ff	3	5
LDD	Load Double Accumulator D	M ⇒ A, M + 1 ⇒ B	IMM	CC	ii kk	3	3
			DIR	DC	dd	2	4
			EXT	FC	hh ll	3	5
			IND,X	EC	ff	2	5
			IND,Y	18 EC	ff	3	6

Mnemonic	Opération	Description	Mode adressage	Code machine complet		taille	Cycles
				Code Mnémonic	Opérande		
LDS	Load Stack Pointer	M : M + 1 ⇒ SP	IMM	8E	ii kk	3	3
			DIR	9E	dd	2	4
			EXT	BE	hh ll	3	5
			IND,X	AE	ff	2	5
			IND,Y	18 AE	ff	3	6
LDX	Load Index Register X	M : M + 1 ⇒ IX	IMM	CE	ii kk	3	3
			DIR	DE	dd	2	4
			EXT	FE	hh ll	3	5
			IND,X	EE	ff	2	5
			IND,Y	CD EE	ff	3	6
LDY	Load Index Register Y	M : M + 1 ⇒ IY	IMM	18 CE	ii kk	4	4
			DIR	18 DE	dd	3	5
			EXT	18 FE	hh ll	4	6
			IND,X	1A EE	ff	3	6
			IND,Y	18 EE	ff	3	6
LSL	Logical Shift Left		EXT	78	hh ll	3	6
			IND,X	68	ff	2	6
			IND,Y	18 68	ff	3	7
LSLA	Logical Shift Left A		A INH	48		1	2
LSLB	Logical Shift Left B		B INH	58		1	2
LSLD	Logical Shift Left Double		INH	05		1	3

Mnemonic	Opération	Description	Mode adressage	Code machine complet		taille	Cycles
				Code Mnémonic	Opérande		
LSR	Logical Shift Right		EXT	74	hh ll	3	6
			IND,X	64	ff	2	6
			IND,Y	18 64	ff	3	7
LSRA	Logical Shift Right A		A INH	44		1	2
LSRB	Logical Shift Right B		B INH	54		1	2
LSRD	Logical Shift Right Double		INH	04		1	3
MUL	Multiply 8 by 8	A* B ⇒ D	INH	3D		1	10
NEG	Two's Complement Memory Byte	0 – M ⇒ M	EXT	70	hh ll	3	6
			IND,X	60	ff	2	6
			IND,Y	18 60	ff	3	7
NEGA	Two's Complement A	0 – A ⇒ A	A INH	40		1	2
NEGB	Two's Complement B	0 – B ⇒ B	B INH	50		1	2
NOP	No operation	No Operation	INH	01		1	2
ORAA	OR Accumulator A (Inclusive)	A + M ⇒ A	A IMM	8A	ii	2	2
			A DIR	9A	dd	2	3
			A EXT	BA	hh ll	3	4
			A IND,X	AA	ff	2	4
			A IND,Y	18 AA	ff	3	5
ORAB	OR Accumulator B (Inclusive)	B + M ⇒ B	B IMM	CA	ii	2	2
			B DIR	DA	dd	2	3
			B EXT	FA	hh ll	3	4
			B IND,X	EA	ff	2	4
			B IND,Y	18 EA	ff	3	5

Mnemonic	Opération	Description	Mode adressage	Code machine complet		taille	Cycles
				Code Mnémonic	Opérande		
PSHA	Push A onto Stack	A→Stack, SP = SP – 1	A INH	36		1	3
PSHB	Push B onto Stack	B→Stack, SP = SP – 1	B INH	37		1	3
PSHX	Push X onto Stack (Low First)	IX→Stack, SP = SP – 2	INH	3C		1	4
PSHY	Push Y onto Stack (Low First)	IY→Stack, SP = SP – 2	INH	18 3C		2	5
PULA	Pull A from Stack	SP = SP + 1, Stack→A	A INH	32		1	4
PULB	Pull B from Stack	SP = SP + 1, Stack→B	B INH	33		1	4
PULX	Pull X From Stack (High First)	SP = SP + 2, Stack→IX	INH	38		1	5
PULY	Pull Y from Stack (High First)	SP = SP + 2, Stack→IY	INH	18 38		2	6
ROL	Rotate Left		EXT	79	hh ll	3	6
			IND,X	69	ff	2	6
			IND,Y	18 69	ff	3	7
ROLA	Rotate Left A		A INH	49		1	2
ROLB	Rotate Left B		B INH	59		1	2
ROR	Rotate Right		EXT	76	hh ll	3	6
			IND,X	66	ff	2	6
			IND,Y	18 66	ff	3	7
RORA	Rotate Right A		A INH	46		1	2
RORB	Rotate Right B		B INH	56		1	2

Mnemonic	Opération	Description	Mode adressage	Code machine complet		taille	Cycles
				Code Mnémonic	Opérande		
RTI	Return from Interrupt	A la fin d'une	INH	3B		1	12
RTS	Return from Subroutine	A la fin d'un module Retour à l'appelant	INH	39		1	5
SBA	Subtract B from A	A – B → A	INH	10		1	2
SBCA	Subtract with Carry from A	A – M – C → A	A IMM	82	ii	2	2
			A DIR	92	dd	2	3
			A EXT	B2	hh ll	3	4
			A IND,X	A2	ff	2	4
			A IND,Y	18 A2	ff	3	5
SBCB	Subtract with Carry from B	B – M – C → B	B IMM	C2	ii	2	2
			B DIR	D2	dd	2	3
			B EXT	F2	hh ll	3	4
			B IND,X	E2	ff	2	4
			B IND,Y	18 E2	ff	3	5
SEC	Set Carry	1 → C	INH	0D		1	2
SEI	Set Interrupt Mask	1 → I	INH	0F		1	2
SEV	Set Overflow Flag	1 → V	INH	0B		1	2
STAA	Store Accumulator A	A → M	A DIR	97	dd	2	3
			A EXT	B7	hh ll	3	4
			A IND,X	A7	ff	2	4
			A IND,Y	18 A7	ff	3	5
STAB	Store Accumulator B	B → M	B DIR	D7	dd	2	3
			B EXT	F7	hh ll	3	4
			B IND,X	E7	ff	2	4
			B IND,Y	18 E7	ff	3	5
STD	Store Accumulator D	A → M, B → M + 1	DIR	DD	dd	2	4
			EXT	FD	hh ll	3	5
			IND,X	ED	ff	2	5
			IND,Y	18 ED	ff	3	6
STOP	Stop Internal Clocks		INH	CF		1	2

Mnemonic	Opération	Description	Mode adressage	Code machine complet		taille	Cycles
				Code Mnémonic	Opérande		
STS	Store Stack Pointer	SP → M : M + 1	DIR	9F	dd	2	4
			EXT	BF	hh ll	3	5
			IND,X	AF	ff	2	5
			IND,Y	18 AF	ff	3	6
STX	Store Index Register X	IX → M : M + 1	DIR	DF	dd	2	4
			EXT	FF	hh ll	3	5
			IND,X	EF	ff	2	5
			IND,Y	CD EF	ff	3	6
STY	Store Index Register Y	IY → M : M + 1	DIR	18 DF	dd	3	5
			EXT	18 FF	hh ll	4	6
			IND,X	1A EF	ff	3	6
			IND,Y	18 EF	ff	3	6
SUBA	Subtract Memory from A	A – M → A	A IMM	80	ii	2	2
			A DIR	90	dd	2	3
			A EXT	B0	hh ll	3	4
			A IND,X	A0	ff	2	4
			A IND,Y	18 A0	ff	3	5
SUBB	Subtract Memory from B	B – M → B	A IMM	C0	ii	2	2
			A DIR	D0	dd	2	3
			A EXT	F0	hh ll	3	4
			A IND,X	E0	ff	2	4
			A IND,Y	18 E0	ff	3	5
SUBD	Subtract Memory from D	D – M : M + 1 → D	IMM	83	ii kk	3	4
			DIR	93	dd	2	5
			EXT	B3	hh ll	3	6
			IND,X	A3	ff	2	6
			IND,Y	18 A3	ff	3	7
SWI	Software Interrupt	Retour au moniteur	INH	3F		1	14
TAB	Transfer A to B	A → B	INH	16		1	2
TAP	Transfer A to CC Register	A → CCR	INH	06		1	2
TBA	Transfer B to A	B → A	INH	17		1	2

Mnemonic	Opération	Description	Mode adressage	Code machine complet		taille	Cycles
				Code Mnémonic	Opérande		
TEST	TEST (Only in Test Modes)	Address Bus Counts	INH	00		1	*
TPA	Transfer CC Register to A	CCR → A	INH	07		1	2
TST	Test for Zero	M – 0	EXT	7D	hh ll	3	6
	Or Minus		IND,X	6D	ff	2	6
			IND,Y	18 6D	ff	3	7
TSTA	Test A for Zero or Minus	A – 0	A INH	4D		1	2
TSTB	Test B for Zero or Minus	B – 0	B INH	5D		1	2
TSX	Transfer Stack Pointer to X	SP + 1 ⇒ IX	INH	30		1	3
TSY	Transfer Stack Pointer to Y	SP + 1 ⇒ IY	INH	18 30		2	4
TXS	Transfer X to Stack Pointer	IX – 1 ⇒ SP	INH	35		1	3
TYS	Transfer Y to Stack Pointer	IY – 1⇒ SP	INH	18 35		2	4
WAI	Wait for Interrupt	Stack Regs & WAIT	INH	3E		2	**
XGDX	Exchange D with X	IX ⇒ D, D ⇒ IX	INH	8F		1	3
XGDY	Exchange D with Y	IY ⇒ D, D ⇒ IY	INH	18 8F		2	4

Convertisseur Analogique-numérique :

But : passer du domaine de l'analogique (variables continues) au domaine du numérique (échantillonnage).

Deux registres de commande permettent de piloter ce **CAN** :

- le registre **OPTION**, d'offset **$39**, avec les 2 bits **CSEL** (bit6) et **ADPU** (bit7):

OPTION	ADPU	CSEL	IRQE	DLY	CME	0	CR1	CR0
RESET	0	0	0	1	0	0	0	0

● **bit 6 : CSEL** (AD/EE charge pump **C**lock **SEL**ect)
permet de choisir l'horloge utilisée par le convertisseur à pompage (pour le Convertisseur
Analogique/Numérique et pour la programmation de l'EEPROM).
CSEL = 0 : c'est E qui est pris (à faire si E ³ 1,5 MHz)
CSEL = 1 : c'est une cellule RC interne qui est utilisée.

● **bit 7 : ADPU** (**AD P**ower **U**p)
ADPU = 0 : le CAN est éteint et n'est pas utilisable,
ADPU = 1 : permet d'utiliser le CAN.
La mise à 1 de ce bit doit être faite avant chaque utilisation du CAN.
Au reset le registre **OPTION** contient **$10**, donc seul **DLY** est positionné.

- le registre **ADCTL** (pour **A**nalog **D**igital **C**on**T**ro**L**) d'offset **$30** :

ADCTL	CCF	0	SCAN	MULT	CD	CC	CB	CA
RESET	0	0	X	X	X	X	X	X

● **bits 3 à 0** : CD CC CB CA : ils permettent de piloter le multiplexeur d'entrée et donc de fixer la voie qui sera utilisée par le CAN suivant le tableau :

CD	CC	CB	CA	ENTREE CHOISIE	RESULTAT DANS
0	0	0	0	PE0	ADR1
0	0	0	1	PE1	ADR2
0	0	1	0	PE2	ADR3
0	0	1	1	PE3	ADR4
0	1	0	0	PE4	ADR1
0	1	0	1	PE5	ADR2
0	1	1	0	PE6	ADR3
0	1	1	1	PE7	ADR4
1	0	0	0	RESERVE	ADR1
1	0	0	1	RESERVE	ADR2
1	0	1	0	RESERVE	ADR3
1	0	1	1	RESERVE	ADR4
1	1	0	0	VRH	ADR1
1	1	0	1	VRL	ADR2
1	1	1	0	VRH/2	ADR3
1	1	1	1	RESERVE	ADR4

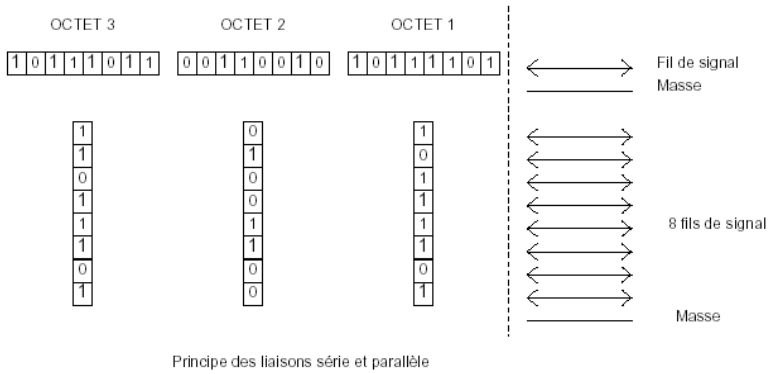
Certaines lignes de ce tableau sont utilisées pour des phases de test du circuit par MOTOROLA.
Sur la dernière colonne de ce tableau, les registres **ADR1** (offset **$31), ADR2** (offset **$32), ADR3** (offset **$33)** et **ADR4** (offset **$34)** sont ceux où le résultat de la conversion sera placé et qui devront être lus pour l'obtenir.
La conversion est faite en **binaire pur** (**VRL** donne **$00** et **VRH** donne **$FF** : pleine échelle).

● **bit 4 : MULT** (**MULT**iple channel).
A la base, le convertisseur analogique / numérique a été conçu pour balayer les voies d'entrées par paquet de quatre, (canal 1 à canal 4) conformément au chronogramme précédent de la figure ci-dessous.
MULT = 1 : le balayage est effectué sur les voies référencées par les bits CD et CC.
Par exemple, si **CD CC** = 0 0 alors ce sont les voies **PE0** à **PE3** qui sont converties.
MULT = 0 : une seule entrée est convertie, car 4 acquisitions successives sont lancées sur la même voie repérée par CD à CA.

● **bit 5 : SCAN** (**SCAN** control).
SCAN = 1 : les conversions ont lieu en continu (mode scanning) et les registres de résultats **ADRi** sont ainsi régulièrement rafraîchis avec les bonnes valeurs. Il n'y a pas de risque que la valeur lue soit erronée, car l'actualisation des registres de résultats est toujours faite en dehors des phases de lecture de ces registres par l'unité centrale.
SCAN = 0 : **un seul cycle** de 4 conversions (selon le bit **MULT**) est **lancé par une écriture** dans le registre **ADCTL** (pas de mode scanning). Avant de lire le (ou les) résultat(s) dans le (ou les) registre(s) adéquat(s) **ADRi**, il faut attendre que ces quatre conversions soient effectuées (au risque de lire des valeurs erronées si la conversion n'est pas terminée) en surveillant le **bit 7 CCF** du registre **ADCTL** que nous allons maintenant présenter.

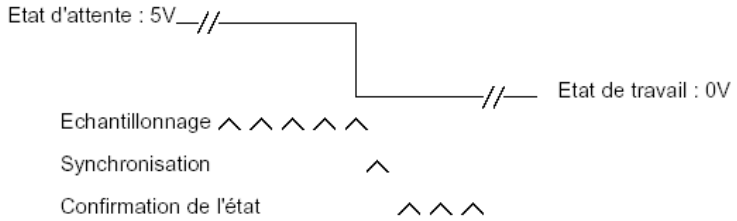
● **bit 7 : CCF** (**C**onversion **C**omplete **F**lag : conversion terminée)
C'est un bit d'état qui est **mis à 0 à chaque écriture** dans **ADCTL**, ce qui, on vient de le voir, a pour effet de lancer un cycle de 4 conversions. **A la fin de ces conversions, CCF passe à 1**, indiquant que ce cycle est terminé, et qu'on peut venir consulter les résultats si on le souhaite. (dans le fonctionnement en mode scanning, les cycles de conversions s'enchaînent automatiquement, même si CCF=1 (inutile de relancer un nouveau cycle de conversion).
Pour remettre à 0 le bit **CCF** (et interrompre le cycle de conversion en cours, en mode scanning ou pas), **il faut écrire dans le registre ADCTL : on lance ainsi un nouveau cycle de conversion.**

Communication série :
Si l'on désire transférer des informations entre deux appareils informatiques (entre deux ordinateurs, entre un système superviseur et un automate programmable, entre un ordinateur et un périphérique par exemple) on dispose de deux grands procédés de communication, comme indiqué sur la figure ci-dessous.
-la transmission série : dans ce cas, les données sont transmises sous forme d'une **trame d'octets**, dont les bits sont transmis les uns derrière les autres, suivant un rythme bien établi.
La principale fonction d'un composant gestionnaire de l'interface série sera d'assurer une transformation parallèle-série pour les données à l'émission et série-parallèle à la réception (grâce à des registres à décalage), puisqu'à l'intérieur du système informatique les données sont traitées globalement par blocs de 8, 16 ou 32 bits. Nous verrons très rapidement que d'autres fonctionnalités viendront se rajouter.
La **rapidité d'une telle liaison est relativement faible**, mais il faut **peu de fils** pour transmettre: ici, typiquement deux fils suffisent au minimum, le premier servant de référence, le second pour le signal effectif.
Exemples : liaison V24, liaison USB, réseaux informatiques avant modulation ...

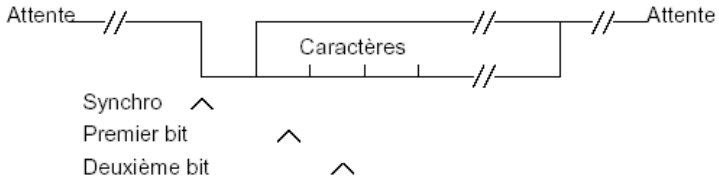


Principe des liaisons série et parallèle

Les liaisons série sont elles même divisées en deux grandes familles : La liaison **SCI** est une liaison **asynchrone** (les données séries peuvent arriver n'importe quand sur la liaison), tandis que la liaison **SPI** est **synchrone** (nécessite une horloge de synchronisation).
Généralités sur la communication série asynchrone :
Le fonctionnement **asynchrone** correspond à un mode de transmission dont la synchronisation se fait caractère par caractère, ceux-ci pouvant arriver ou être transmis à n'importe quel instant. Pour pouvoir détecter sur une entrée asynchrone qu'il y a un début de réception de caractère, on va définir deux états :
- **5V** pour la phase d'attente (ou état de repos appelé **MARK**),
- **0V** pour signaler l'arrivée d'un caractère (état de travail appelé **SPACE**).
Le schéma de la figure ci-dessous résume le principe de la synchronisation par caractère :



Cette détection est effectuée par le composant gestionnaire de la liaison série, qui signalera alors dans un registre prévu à cet effet et accessible au programmeur, l'arrivée d'un caractère.
Il pourra même, si on lui en donne l'autorisation par logiciel, provoquer une interruption vers le micro/processeur/contrôleur.
Le format d'une transmission asynchrone est donc le suivant :

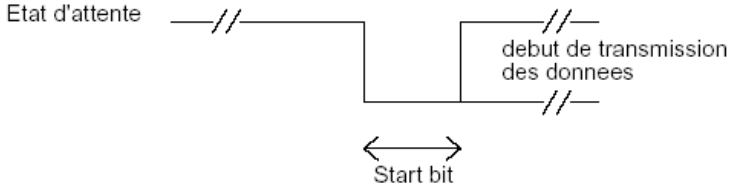


Chaque bit est ainsi acquis par échantillonnage successif de l'état situé au milieu de la durée de ce bit, grâce à une horloge interne au circuit U.A.R.T. et resynchronisée par le bit de départ.
A condition bien entendu que **les vitesses de transmission soient les mêmes** côté émetteur et côté récepteur.
Paramètres de la liaison série asynchrone
Ce sont eux qui définissent le format de la liaison (qui doit être le même côté émetteur et côté récepteur).

- **Bit de départ (START BIT)**

Lorsque la réception est **autorisée**, une voie de réception asynchrone teste en permanence le passage de l'état 5V à l'état 0V pour détecter un début de transmission.

On définit une durée pendant laquelle le signal restera à un état 0V : cette première étape de la transmission d'un caractère s'appelle **le bit de départ (start bit).**
Ainsi, chaque caractère se trouvera précédé d'un start bit, comme sur la figure ci-dessous.

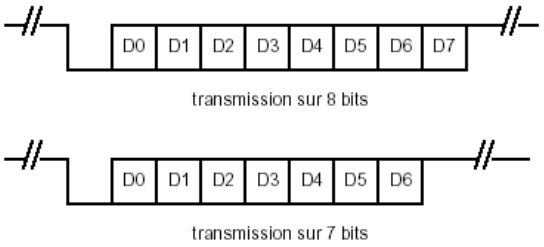


- **Vitesse de liaison**

Nous venons de voir que l'horloge de récupération des bits était synchronisée par le front descendant du start bit.
Pendant toute la durée d'une trame, l'horloge interne devra ensuite indiquer l'emplacement du milieu des bits pour pouvoir les échantillonner à cet instant là, où le niveau du bit est le mieux défini.
Notons cependant que certains U.A.R.T. échantillonnent l'entrée asynchrone pendant toute la durée du bit (typiquement de 2 à 64 fois en fonction de la vitesse de transmission), ce qui améliore sensiblement l'immunité par rapport au bruit et divers parasites industriels.
Cette horloge correspond donc à une vitesse de transmission des informations en mode asynchrone. Ces vitesses sont définies en nombre de bits par seconde ou **Bauds.**
Les périphériques spécialisés pour la transmission ou réception asynchrone peuvent en général travailler à des vitesses comprises entre 50 et 19200 bauds. Ces différentes valeurs sont normalisées, les vitesses successives étant calculées, au moins pour les plus élevées, en les divisant ou multipliant par 2 (**Exemple** : 19200 = 9600*2).

- **Nombre de bits de données**

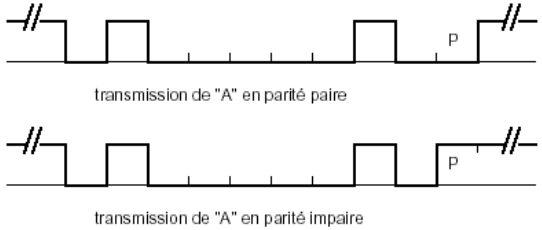
Le nombre de bits d'une transmission est le nombre de bits **utiles** de chaque caractère proprement dit. La plupart des U.A.R.T. sont paramétrables par logiciel et offrent le plus souvent le choix entre 5 et 8 bits utiles par caractère. C'est le type d'information à échanger qui déterminera le nombre de bits utiles à transmettre:
- si l'on désire ne transmettre que des caractères A.S.C.I.I., ceux-ci sont alors codés sur 7 bits de **$00** (caractère Null) à **$7F**,
- si l'on souhaite transmettre des informations binaires donc codées sur un octet il faudra alors 8 bits de données utiles.
Le premier bit à être transmis est le bit **L.S.B**.



- **Parité :**

Lorsque l'on transmet des informations à distance, il est souhaitable que celles-ci soient reçues correctement, c'est-à-dire **sans erreur**. Il faut donc pouvoir détecter l'intégrité de chaque caractère reçu. Pour cela, on peut rajouter, après chaque caractère, un bit spécial dit **bit de parité (Parity Bit).**
Il existe 2 types de parité :
- la **parité paire (even parity : 4** lettres à even **)**, lorsque le nombre de bits à 1 du caractère **et** de ce bit de parité, est **pair**,
- la **parité impaire (odd parity : 3** lettres à odd **)**, lorsque le nombre de bits à 1 du caractère **et** de ce bit de parité, est **impair**.

Attention : rien à voir avec la parité arithmétique (multiple de 2 ou pas) !!!
Soit par exemple à transmettre le caractère A.S.C.I.I."A", codé **$41** = % 0**1**00 000**1** en parité paire sur 8 bits : le bit de parité **P** vaudra **0** (2 bits à 1). En parité impaire, ce bit **P** vaudrait **1** (3 bits à 1).
Les chronogrammes sont représentés ci-dessous, sur la figure ci-dessous :

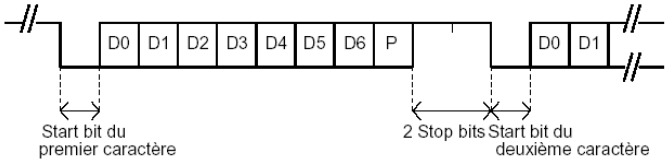


A la réception, on utilise le même type de parité qu'à l'émission.
En comptant le nombre de bits à 1, on pourra détecter une violation de parité si un bit (et plus généralement un nombre impair de bits) a changé d'état : le bit de parité reçu n'aura pas la bonne valeur, l'erreur sera détectée.

Il est clair que la parité a ses limites de détection car si un nombre pair de bits s'inversent dans un octet, le bit de parité restera correct et l'erreur ne sera pas détectée.

- **Bit stop**

Ils sont rajoutés après le bit de parité, ce qui permet un retour systématique de la ligne à l'étatde repos entre chaque caractère (voir figure ci-dessous).

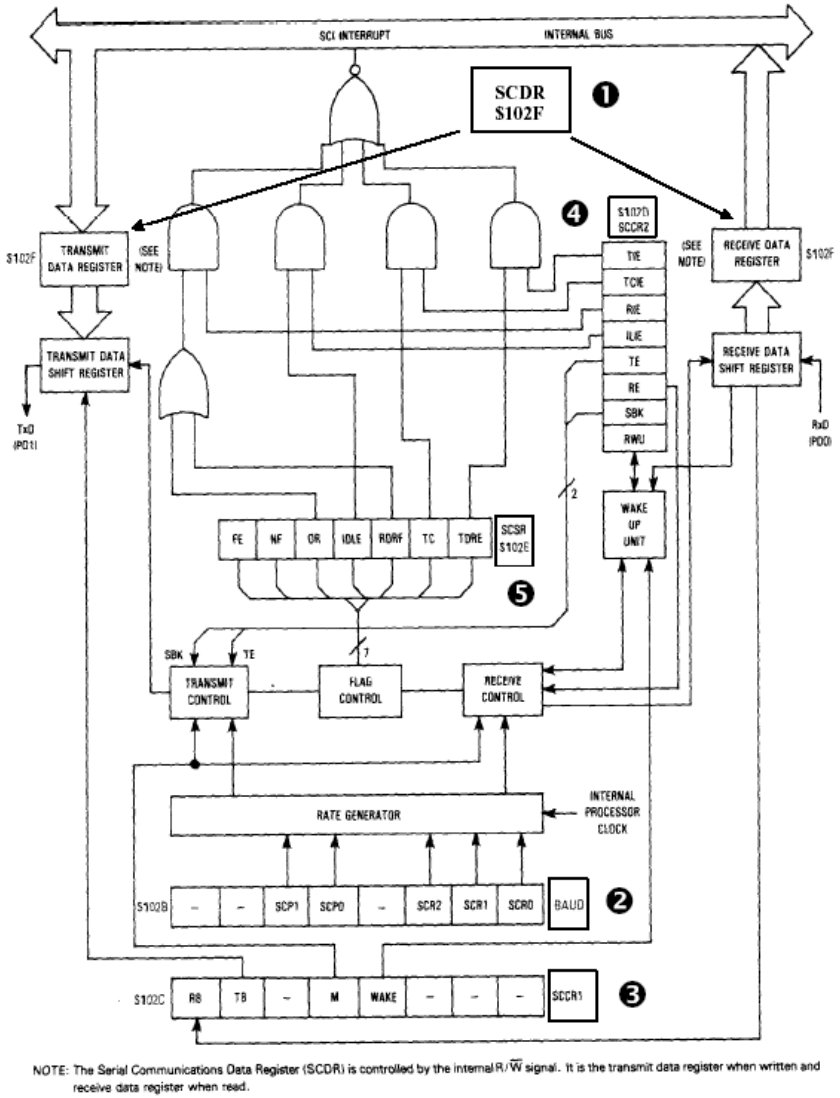


Vu côté émetteur, le nombre de **bits d'arrêt (stop bit)** donne le temps minimum d'attente pour la transmission d'un nouveau caractère.

Vu côté récepteur, ce sera le temps minimum d'attente **avant** une nouvelle scrutation d'un start bit.

Registres concernés par la liaison SCI du HC11

Le synoptique général du bloc SCI du HC11 est donné ci-dessous,



NOTE: The Serial Communications Data Register (SCDR) is controlled by the internal R/$\overline{W}$ signal. It is the transmit data register when written and receive data register when read.

Comme le montre la figure VII.29. la broche pour **RXD** (**R**éception des **D**onnées) est la ligne **PD0** du **PORTD** et la broche pour **TXD** (**T**ransmission des **D**onnées) est la ligne **PD1** du **PORTD.**

Au total 5 registres internes du HC11 sont dédiés à la commande de la liaison série, numérotés de 1 à 5 sur la figure ci-dessous. Nous allons maintenant les présenter.

1. SCDR (Serial Communications Data Register) : offset $2F,

Il est en fait constitué de 2 "sous-registres" à la **même adresse physique** :

- en **lecture**, on accède au **RDR** (**R**eceive **D**ata **R**egistrer) : c'est le registre où sont stockés un par un, les caractères qui arrivent sur la liaison série via l'entrée **RXD** et le convertisseur série/parallèle,
- en **écriture**, on accède au **TDR** (**T**ransmit **D**ata **R**egistrer) : c'est le registre où il faut envoyer un par un les caractères à transmettre sur la liaison série via le convertisseur parallèle/série et la sortie **TXD**.

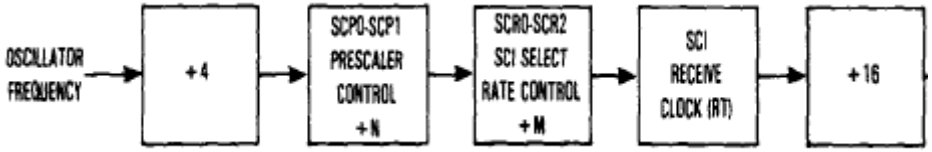
2. BAUD (BAUD Rate Register) : offset $2B,

Il permet de choisir la **vitesse de transmission** en Bauds.

BAUD	TCLR	0	SCP1	SCP0	RCKB	SCR2	SCR1	SCR0
Reset	0	0	0	0	0	U	U	U

– **Bit 7** : **TCLR** (**T**est **CL**ea**R** baud rate counters) et **Bit 3** : **RCKB** (SCI test baud rate clock check) sont 2 bits de test à laisser à 0.

- **Bits 5 à 4** : **SCP1- SCP0** (**SC**i baud rate **P**rescaler) et**Bits 3 à 0** : **SCR2 - SCR0** (**SC**i baud **R**ate Select) permettent de choisir la vitesse de la liaison en établissant **une chaîne de diviseurs** en cascade à partir de la fréquence du quartz du HC11, comme indiqué sur la figure ci-dessous:



Au final, le tableau fourni sur la figure VII.31. ci-dessous permet de choisir la valeur à utiliser pour les bits **SCPi** et **SCRi** en fonction :

- de la fréquence du quartz,
- de la vitesse souhaitée.

Prescaler Selects					Prescale Divide	Baud Set Divide	Crystal Frequency (MHz)					
							4.00	4.9152	8.00	10.00	12.00	16.00
							Bus Frequency (MHz)					
SCP1	SCP0	SCR2	SCR1	SCR0			1.00	1.23	2.00	2.50	3.00	4.00
0	0	0	0	0	1	1	62500	76800	125000	156250	187500	250000
0	0	0	0	1	1	2	31250	38400	62500	78125	93750	125000
0	0	0	1	0	1	4	15625	19200	31250	39063	46875	62500
0	0	0	1	1	1	8	7813	9600	15625	19531	23438	31250
0	0	1	0	0	1	16	3906	4800	7813	9766	11719	15625
0	0	1	0	1	1	32	1953	2400	3906	4883	5859	7813
0	0	1	1	0	1	64	977	1200	1953	2441	2930	3906
0	0	1	1	1	1	128	488	600	977	1221	1465	1953
0	1	0	0	0	3	1	20833	25600	41667	52083	62500	83333
0	1	0	0	1	3	2	10417	12800	20833	26042	31250	41667
0	1	0	1	0	3	4	5208	6400	10417	13021	15625	20833
0	1	0	1	1	3	8	2604	3200	5208	6510	7813	10417
0	1	1	0	0	3	16	1302	1600	2604	3255	3906	5208
0	1	1	0	1	3	32	651	800	1302	1628	1953	2604
0	1	1	1	0	3	64	326	400	651	814	977	1302
0	1	1	1	1	3	128	163	200	326	407	488	651
1	0	0	0	0	4	1	15625	19200	31250	39063	46875	62500
1	0	0	0	1	4	2	7813	9600	15625	19531	23438	31250
1	0	0	1	0	4	4	3906	4800	7813	9766	11719	15625
1	0	0	1	1	4	8	1953	2400	3906	4883	5859	7813
1	0	1	0	0	4	16	977	1200	1953	2441	2930	3906
1	0	1	0	1	4	32	488	600	977	1221	1465	1953
1	0	1	1	0	4	64	244	300	488	610	732	977
1	0	1	1	1	4	128	122	150	244	305	366	488
1	1	0	0	0	13	1	4808	5908	9615	12019	14423	19231
1	1	0	0	1	13	2	2404	2954	4808	6010	7212	9615
1	1	0	1	0	13	4	1202	1477	2404	3005	3606	4808
1	1	0	1	1	13	8	601	738	1202	1502	1803	2404
1	1	1	0	0	13	16	300	369	601	751	901	1202
1	1	1	0	1	13	32	150	185	300	376	451	601
1	1	1	1	0	13	64	75	92	150	188	225	300
1	1	1	1	1	13	128	38	46	75	94	113	150

Les deux registres suivants sont des registres de contrôle.

3. SCCR1 (Serial Communications Control Register 1) : offset $2C,

Pour la longueur des caractères et le mode de "réveil".

SCCR1	R8	T8	0	M	WAKE	0	0	0
Reset	U	U	0	0	0	0	0	0

- **Bit 4** : **M** (**M**(ode) : SCI character length)
  = 0 : 1 start bit, **8** bits de données, 1 stop bit,
  = 1 : 1 start bit, **9** bits de données, 1 stop bit.
- **Bit 7** : **R8** (**R**eceive data bit **8**)
  Si M = 1, lieu de stockage du 9ème bit du caractère reçu.
- **Bit 6** : **T8** (**T**ransmit data bit **8**)
  Si M = 1, lieu de stockage du 9ème bit du caractère à transmettre.
- **Bit 3** : **WAKE** (WAKE up method select)

= 0 : réveil de la réception si la ligne **RXD** est inactive (niveau haut) pendant la durée d'au moins un caractère (**IDLE**),
= 1 : réveil de la réception si le **MSB** du caractère reçu (8éme ou 9éme bit) est 1 (**ADRESS MARK**).

4. SCCR2 (Serial Communications Control Register 1) : offset $2D,

Pour la mise en route de la SCI et les masques locaux d'interruption de la SCI.

SCCR2	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
Reset	0	0	0	0	0	0	0	0

● **Bit 7** : **TIE** (**T**ransmit **I**nterrupt **E**nable)
= 0 : pas de fonctionnement sous interruption de **registre de transmission vide**,
= 1 : autorisation des interruptions quand le registre de transmission est **vide** (le caractère à transmettre est passé dans le convertisseur parallèle/série) et donc on peut écrire un nouveau caractère dans le registre de transmission (le caractère suivant à émettre).

● **Bit 6** : **TCIE** (**T**ransmit **C**omplete **I**nterrupt **E**nable)
= 0 : pas de fonctionnement sous interruption de transmission **complète**,
= 1 : autorisation des interruptions quand le convertisseur parallèle/série est **vide**, c'est-à-dire quand le caractère en cours d'émission est **totalement** et **complètement** transmis sur **TXD** et que le registre de transmission est aussi vide.

● **Bit 5** : **RIE** (**R**eceive **I**nterrupt **E**nable)
= 0 : pas de fonctionnement sous interruption de **registre de réception plein**,
= 1 : autorisation des interruptions quand le registre de réception est **plein** (un nouveau caractère vient d'arriver sur **RXD** et il est disponible dans le **registre de réception).** On peut venir le lire.

● **Bit 4** : **ILIE** (**I**dle **L**ine **I**nterrupt **E**nable)
= 0 : pas de fonctionnement sous interruption de **détection de ligne inactive**,
= 1 : autorisation des interruptions quand la ligne de réception **RXD** reste **au repos** (niveau haut) pendant la durée d'au moins un caractère (10 ou 11 bits suivant le bit **M** de **SCCR1**).

● **Bit 3** : **TE** (**T**ransmit **E**nable)
= 0 : **émetteur désactivé** (la ligne **PD1** peut être récupérée comme ligne de **PORTD** sous le contrôle de **DDRD1**),
=1 : **validation** (mise en route) de **l'émetteur,** on peut l'utiliser normalement.

● **Bit 2** : **RE** (**R**eceive **E**nable)
= 0 : **récepteur désactivé**, (la ligne **PD0** peut être récupérée comme ligne de **PORTD** sous le contrôle de **DDRD0**),
=1 : **validation** (mise en route) du **récepteur,** on peut l'utiliser normalement.

● **Bit 1** : **RWU** (**R**eceiver **W**ake **U**p)
= 0 : la SCI fonctionne normalement,
=1 : met le **récepteur en sommeil.** Il sera réveillé suivant la méthode définie par le bit **WAKE** de **SCCR1**.

● **Bit 0** : **SBK** (**S**end **BR**eak)
= 0 : la SCI fonctionne normalement,
=1 : envoi de « BREAK » sur la ligne **TXD** (séquence de 10 ou 11 bits à 0).

Le dernier registre est un registre d'état (status register qui contient des flags).

5. SCSR (Serial Communications Status Register) : offset $2E,

SCSR	TDRE	TC	RDRF	IDLE	OR	NF	FE	0
Reset	1	1	0	0	0	0	0	0

● **Bit 7** : **TDRE** (**T**ransmit **D**ata **R**egister **E**mpty)
Il passe à 1 pour indiquer que le registre de transmission est **vide** (le caractère à transmettre est passé dans le convertisseur parallèle/série).
En mode **sans interruption** (si le bit **TIE** du **SCCR2** est à 0), on pourra détecter ce passage à 1 de **TDRE** en le scrutant (pooling sur **TDRE**) et ensuite on peut écrire un nouveau caractère dans le registre de transmission (le caractère suivant) ce qui aura pour effet de faire passer ce bit **TDRE** à 0.
En mode **interruption** (si le bit **TIE** du **SCCR2** est à 1), le passage à 1 de **TDRE** déclenche une interruption de la SCI. (vecteur **$FFD6-$FFD7** en mode circuit seul ou étendu).
Pour acquitter cette interruption, il faut refaire passer ce bit **TDRE** à 0 (car sinon on serait en interruption permanente) en lisant le registre **SCSR** (avec **TDRE** à 1) puis en écrivant le prochain caractère à transmettre dans le registre de transmission. Tout ceci dans la routine d'interruption.

● **Bit 6** : **TC** (**T**ransmit **C**omplete)
Il passe à 1 pour indiquer que le convertisseur parallèle/série est **vide**, c'est-à-dire quand le caractère en cours d'émission est **totalement** et **complètement** transmis sur **TXD** et que le registre de transmission est aussi vide.
En mode **sans interruption** (si le bit **TCIE** du **SCCR2** est à 0), on pourra détecter ce passage à 1 de **TC** en le scrutant (pooling sur **TC**) et ensuite on peut écrire un nouveau caractère dans le registre de transmission (le caractère suivant) ce qui aura pour effet de faire passer ce bit **TC** à 0.

En mode **interruption** (si le bit **TCIE** du **SCCR2** est à 1), le passage à 1 de **TC** déclenche une interruption de la SCI. (vecteur **$FFD6-$FFD7** en mode circuit seul ou étendu).
Pour acquitter cette interruption, il faut refaire passer ce bit **TC** à 0 (car sinon on serait en interruption permanente) en lisant le registre **SCSR** (avec **TC** à 1) puis en écrivant le prochain caractère à transmettre dans le registre de transmission. Tout ceci dans la routine d'interruption.

● **Bit 5** : **RDRF** (**R**eceive **D**ata **R**egister **F**ull)

Il passe à 1 pour indiquer que le registre de réception est **plein** (un nouveau caractère vient d'arriver sur **RXD** et il est disponible dans le **registre de réception).**
En mode **sans interruption** (si le bit **RIE** du **SCCR2** est à 0), on pourra détecter ce passage à 1 de **RDRF** en le scrutant (pooling sur **RDRF**) et ensuite on peut lire ce nouveau caractère dans le registre de réception (le caractère reçu) ce qui aura pour effet de faire passer ce bit **RDRF** à 0.
En mode **interruption** (si le bit **RIE** du **SCCR2** est à 1), le passage à 1 de **RDRF** déclenche une interruption de la SCI. (vecteur **$FFD6-$FFD7** en mode circuit seul ou étendu).
Pour acquitter cette interruption, il faut refaire passer ce bit **RDRF** à 0 (car sinon on serait en interruption permanente) en lisant le registre **SCSR** (avec **RDRF** à 1) puis en lisant ce caractère reçu dans le registre de réception. Tout ceci dans la routine d'interruption.

● **Bit 4** : **IDLE** (**IDLE** line detect)

Il passe à 1 pour indiquer que la ligne de réception **RXD** reste **au repos** (niveau haut) pendant la durée d'au moins un caractère (10 ou 11 bits suivant le bit **M** de **SCCR1**).
En mode **sans interruption** (si le bit **ILIE** du **SCCR2** est à 0), on pourra détecter ce passage à 1 de **IDLE** en le scrutant (pooling sur **IDLE**) et ensuite on peut lire le registre de réception (qui ne contient pas de caractère, lecture « à vide ») ce qui aura pour effet de faire passer ce bit **IDLE** à 0.
En mode **interruption** (si le bit **ILIE** du **SCCR2** est à 1), le passage à 1 de **IDLE** déclenche une interruption de la SCI. (vecteur **$FFD6-$FFD7** en mode circuit seul ou étendu).
Pour acquitter cette interruption, il faut refaire passer ce bit **ILIE** à 0 (car sinon on serait en interruption permanente) en lisant le registre **SCSR** (avec **ILIE** à 1) puis en lisant le registre de réception (lecture « à vide »). Tout ceci dans la routine d'interruption.

● **Bit 3** : **OR** (**O**ver **R**un error)

Il passe à 1 pour indiquer un **débordement** à la réception : un nouveau caractère arrive alors que le précédent n'a pas été lu.

● **Bit 2** : **NF** (**N**oise **F**lag)

Il passe à 1 pour indiquer que du **« bruit »** est détecté sur la ligne.

● **Bit 1** : **FE** (**F**raming **E**rror)

Il passe à 1 pour indiquer que le bit de stop n'a pas été détecté sur le caractère reçu. Ce peut être une erreur de format entraînant une « désynchronisation » de la liaison.

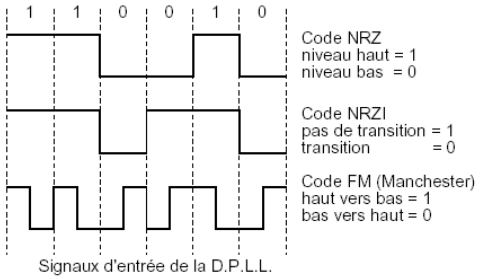
La liaison série synchrone SPI du HC11

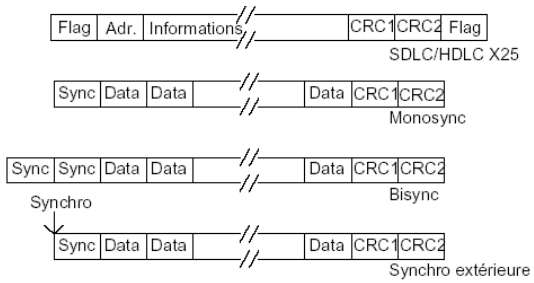
Généralités sur les liaisons séries synchrones

Le terme **synchrone** doit être interprété de la façon suivante :
à tout instant, il existe un lien temporel entre l'émetteur et le récepteur.
Pratiquement cela signifie qu'il existe une référence de temps commune et permanente (tout au moins dans la phase de connexion) entre les deux systèmes. Cette référence peut être :
- soit fournie par une horloge véhiculée **en plus** des signaux informationnels proprement dits (solution utilisée en pratique pour les très courtes distances),
- soit en fabricant, au niveau du récepteur, une horloge extraite du message lui même grâce à un type de codage adéquat. Au moyen d'une **boucle à verrouillage de phase numérique** ou **D.P.L.L.** (**D**igital **P**hase **L**ocked **L**oop) intégrée dans le composant gestionnaire de la liaison synchrone et pilotée en absence de signal reçu par une horloge (fréquence libre de la D.P.L.L.) de période 16 à 32 fois plus faible que la durée nominale d'un bit, il est possible de synthétiser en quelques cycles, une autre horloge calée sur les données reçues en recherchant les fronts montants ou descendants sur le signal reçu (cas des codages N.R.Z.I. ou F.M. par exemple),
- soit à un niveau supérieur, réalisée par des séquences de bits identifiables (flags). C'est le cas notamment des protocoles **SDLC/HDLC** (**S**erial or **H**igh **D**ata **L**ink **C**ontrol) utilisés dans les réseaux **X25**,
- soit présente sous la forme d'octets spécifiques (dits de synchro) dans les protocoles orientés octets.

En pratique, il existe des protocoles avec un seul caractère de synchro (Monosync) ou deux caractères de synchro (Bisync),
- soit un signal externe.

Les chronogrammes de la figure ci-dessous résument quelques unes de ces différentes possibilités :





Principaux protocoles synchrones

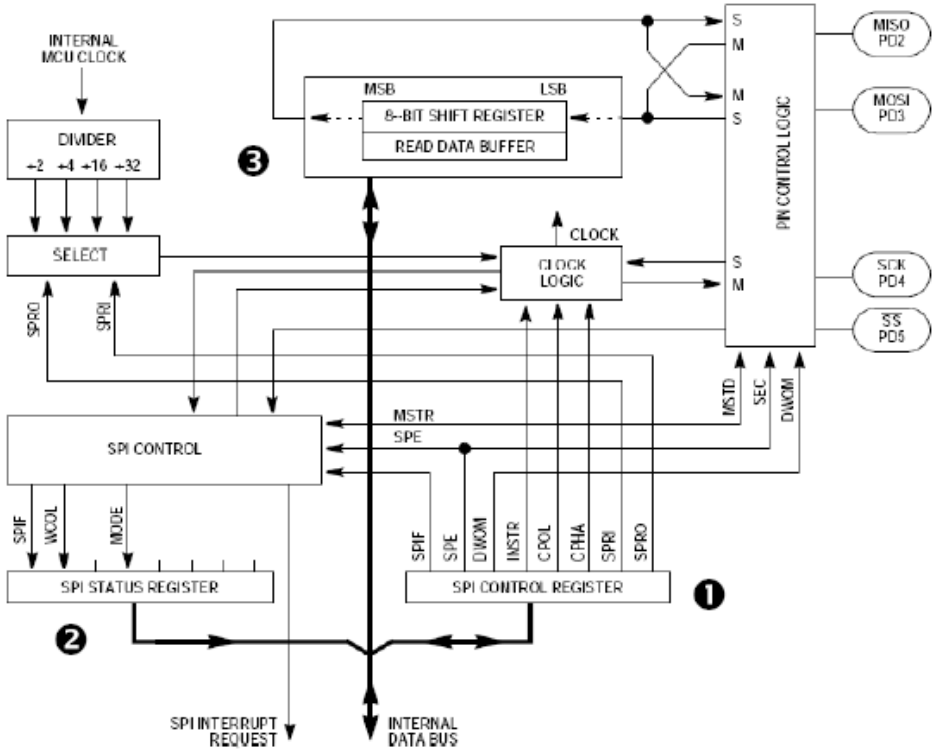
De façon générale, les liaisons synchrones permettent **des vitesses de transmission plus élevées** que les liaisons série asynchrones (typiquement jusqu'à plusieurs Mégabits/seconde).

Registre concernés par la liaison SPI du HC11

La **SPI** (**S**erial **P**eripheral **I**nterface) est une interface de type série synchrone qui permet la connexion (en **Full Duplex**) sous forme série de plusieurs circuits en mode **Maître/Esclave** (**Master/Slave**).

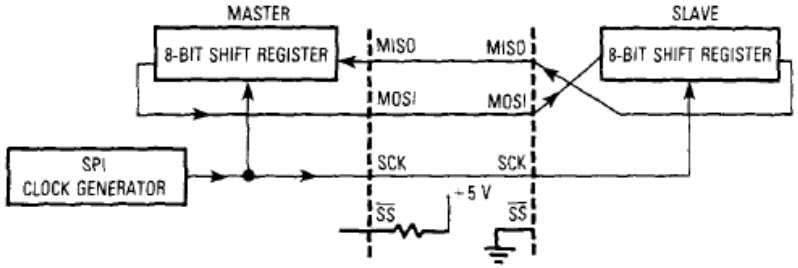
Dans ce mode c'est le maître qui s'occupe totalement de la communication, l'(ou les) esclave(s) répondant aux requêtes du maître.

Le synoptique général du bloc SPI du HC11 est donné ci-dessous,



On y voit en haut sur la droite les 4 pattes du HC11 réservées pour cet usage : les lignes **PD2** à **PD5** du **PORTD**.

Le rôle et la signification de ces 4 signaux est représenté sur la figure suivante, montant un exemple de connexion entre un maître (à gauche) et son esclave (à droite).



- ligne **SCK** (signal d'horloge) : c'est une sortie pour le maître et une entrée pour l'esclave car c'est bien **le maître qui génère l'horloge**.
Elle permet la synchronisation et l'échantillonnage des échanges de données.
- ligne **MOSI** (**M**aster **O**utput **S**lave **I**nput) : sortie chez le maître et entrée chez l'esclave, elle assure le passage des données du maître vers l'esclave,
- ligne **MISO** (**M**aster **I**nput **S**lave **O**utput) : entrée chez le maître et sortie chez l'esclave, elle assure le passage des données de l'esclave vers le maître.
Ainsi, à chaque coup d'horloge, le maître et l'esclave s'échangent un bit (le **MSB** en premier).
Après huit coups d'horloge le maître a transmis un octet à l'esclave et vice-versa.
On voit d'ailleurs sur le schéma de la figure VII.34. précédente que l'ensemble des 2 registres à décalage 8 bits forment **l'équivalent d'un registre à décalage 16 bits unique** partagé entre le maître et l'esclave : l'émission par le maître d'un octet vers l'esclave provoque la réception par le maître de l'octet contenue dans le registre de l'esclave (le registre à décalage du maître « se vide » dans celui de l'esclave pendant que celui de l'esclave « remplit » celui du maître).
Mais c'est toujours le maître qui décide des transferts, il doit émettre un octet, pour en recevoir un autre.
- ligne SS (**S**lave **S**elect) :
● pour un esclave, cette ligne est **obligatoirement une entrée**, et elle doit être au niveau logique **bas lors des transferts.**
● pour un maître :
- si elle est configurée en **entrée,** elle doit être maintenue au niveau **haut** (par une résistance de pull up **) pour autoriser les transferts.** Si elle passe à 0, cela signifie qu'un esclave veut devenir maître à sa place et donc que le maître doit cesser de transmettre. Cette situation provoque le passage à 1 du bit **MODF** (MODe Fault) dans le registre de status **SPSR** de la SPI (voir plus loin).
- par contre, si elle configurée en **sortie,** elle devient une broche de sortie d'intérêt général (PD5) et est ignorée par l'interface SPI.
Dans le cas d'un système à un maître et plusieurs esclaves, ceux-ci vont tous utiliser l'horloge du maître et vont partager les lignes **MOSI** et **MISO**. Par contre aucun adressage des esclaves n'est possible, il faudra une ligne de sélection SS par esclave générée par une logique de décodage pilotée par le maître (les esclaves non sélectionnés auront leurs sorties **MISO** en **haute-impédance**, seul l'esclave sélectionné pourra recevoir/envoyer un octet du/vers le maître).
Le maître peut aussi faire du **broadcast** (envoyer un octet à tous les esclaves en même temps en n'en sélectionnant aucun). Bien sûr, dans ce cas, il ne recevra en retour aucun octet.

L'inconvénient de la liaison SPI, c'est qu'il n'y a pas d'acquittement de la part de l'esclave dans le cas d'un échange d'octet : le maître peut parler « dans le vide » sans le savoir.
Avant d'utiliser la liaison SPI, il faut initialiser le registre de direction du **PORTD** : **DDRD**.
● bit **DDRD2** : donne le sens de la ligne **MISO** (**PD2**).
Pour un maître, la ligne **MISO** est toujours une entrée quelque soit la valeur de **DDRD2**.
Pour un esclave, **DDRD2** = 1 pour avoir cette ligne en sortie. Si **DDRD2** = 0 sur un esclave mise de cette sortie en haute-impédance dans le cas de systèmes multi-esclaves par exemple.
● bit **DDRD3** : donne le sens de la ligne **MOSI** (**PD3**).
Pour un maître, **DDRD3** = 1 pour avoir cette ligne en sortie. Si **DDRD3** = 0 pour un maître, aucune donnée ne sera émise.
Pour un esclave, la ligne **MOSI** est toujours une entrée quelque soit la valeur de **DDRD3**.
● bit **DDRD4** : donne le sens de la ligne **SCK** (**PD4**).
Pour un maître, on doit mettre **DDRD4** = 1 pour configurer **SCK** en sortie.
Pour un esclave, la ligne **SCK** est toujours une entrée quelque soit la valeur de **DDRD4**.
● bit **DDRD5** : donne le sens de la ligne SS (**PD5**).
Pour un maître, on doit mettre **DDRD5** = 1 pour configurer SS en sortie d'usage général. Si **DDRD5** = 0 sur un maître, cette ligne devient une entrée de détection de défauts de transmission (bit **MODF**).
Pour un esclave, la ligne SS est toujours une entrée quelque soit la valeur de **DDRD5**.
Comme on peut le voir sur le schéma précédent de la figure VII.33., **trois** registres internes du HC11 servent à l'utilisation et à la programmation de la liaison SPI. Nous allons maintenant les détailler.
_ **SPCR** (**S**erial **P**eripheral **C**ontrol **R**egister) : offset **$28**
C'est le registre de contrôle qui permet de choisir le mode de fonctionnement de l'interface et donc de l'initialiser correctement en fonction des besoins de l'application :

SPCR	SPIE	SPE	DWOM	MSTR	CPOL	CPHA	SPR1	SPR0
Reset	0	0	0	0	0	1	U	U

● **Bit 7** : **SPIE** (**S**erial **P**eripheral **I**nterrupt **E**nable)

= 0 : pas de fonctionnement sous interruption **sur fin de transfert** (bit **SPIF**) ou écrituredans le registre à décalage pendant le transfert (bit **WCOL**)
= 1 : autorisation des interruptions quand **SPIF = 1**.

● **Bit 6** : **SPE** (**S**erial **P**eripheral **E**nable)
= 0 : **SPI désactivée** (les lignes **PD2** à **PD5** peuvent être récupérées comme lignes d'E/S de **PORTD** sous le contrôle de **DDRD2** à **DDRD5**),
=1 : **validation** (mise en route) de la SPI, on peut l'utiliser normalement.

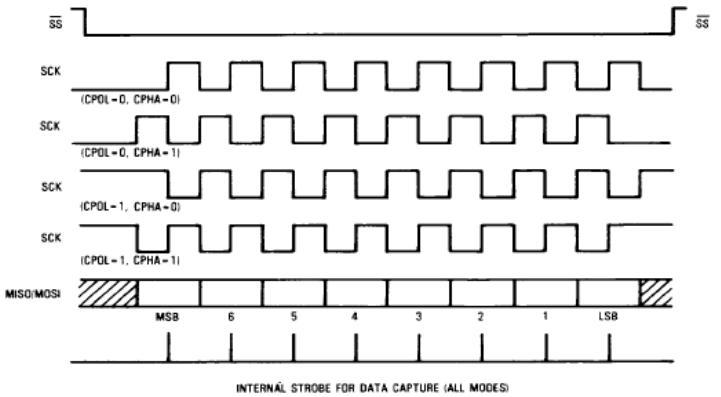
● **Bit 5 : DWON** (port **D W**ire **O**r **M**ode option) affecte ensemble les 6 lignes de **PORTD** suivant la logique :
= 0 : ces 6 lignes sont des sorties CMOS normales en « totem-pôle »,
= 1 : ces 6 lignes sont des sorties en **drain ouvert**.
Cela permet de faire des **« ou câblés »** en reliant plusieurs sorties de ce type ensemble, après bien sûr les avoir reliées chacune au 5V par une résistance de « pull up ».

● **Bit 4 : MSTR** (**M**a**ST**e**R** mode select)
= 0 : mode **esclave**,
= 1 : mode **maître**.

● **Bit 3 : CPOL** (**C**lock **POL**arity)
La ligne **SCK** sert à synchroniser et à échantillonner les bits pendant un échange. En dehors des échanges, cette ligne est dite au repos. Il y a donc, 2 états de repos possibles :
= 0 : **SCK** est au repos à l'état **bas**,
= 1 : **SCK** est au repos à l'état **haut**.

● **Bit 2 : CPHA** (**C**lock **PHA**se) Donne la « phase » (retard) de l'horloge.
= 0 : échantillonnage des bits sur le **premier** front de l'horloge : front **montant** si **CPOL** = 0 et front **descendant** si **CPOL** = 1.
= 1 : échantillonnage des bits sur le **deuxième** front de l'horloge : front **descendant** si **CPOL** = 0 et front **montant** si **CPOL** = 1.

Au final, en fonction des 4 combinaisons possibles pour les bits **CPOL** et **CPHA**, on obtient 4 modes d'échantillonnage des bits, comme le représentent les chronogrammes de la figure suivante :



Noter le fait que la ligne SS reste à 0 (active au niveau bas) pendant toute la durée d'un échange.
La configuration de la SPI est souvent nommée dans les documentations techniques sous forme du nombre (appelé alors le **mode**) dont le bit de poids fort est **CPOL** et le bit de poids faible **CPHA**, comme sur le tableau suivant :

CPOL	CPHA	Mode
0	0	0
0	1	1
1	0	2
1	1	3

Les modes 0 et 3 sont identiques en transmission, seul change l'état de la ligne d'horloge au repos. C'est pour cette raison que de nombreux périphériques peuvent être capables de dialoguer soit en mode 0 et 3, soit en mode 1 et 2.

● **Bit 1** à **Bit 0 : SPR1/SPR0** (**S**erial **P**eripheral **R**ate selection)
Ces 2 bits permettent de choisir la vitesse (fréquence de **SCK**) de la liaison SPI sur le maître (n'ont aucun effet sur l'esclave) comme indiqué ci-dessous :

SPR1	SPR0	E divisé par	soit pour un quartz à 8 MHz (E = 2 MHz)
0	0	2	1 MHz
0	1	4	500 kHz
1	0	16	125 kHz
1	1	32	62,5 kHz

- **SPSR** (**S**erial **P**eripheral **S**tatus **R**egister) : offset **$29**
C'est le registre de status de la SPI.

SPSR	SPIF	WCOL	0	MODF	0	0	0	0
Reset	0	0	0	0	0	0	0	0

● **Bit 7** : **SPIF** (**SPI** complete **F**lag)
Il passe à 1 pour indiquer que le transfert d'un octet est terminé.
En mode **sans interruption** (si le bit **SPIE** du **SPCR** est à 0), le maître et l'esclave pourront détecter ce passage à 1 de **SPIF** en le scrutant (pooling sur **SPIF**) et ensuite le maître et l'esclave pourront lire l'octet reçu dans le registre **SPDR**, ce qui aura pour effet de faire repasser ce bit **SPIF** à 0. Puis le maître et l'esclave pourront éventuellement écrire un nouvel octet dans le **SPDR** (l'octet suivant à émettre), mais **c'est l'écriture du maître** dans le registre **SPDR** qui d**émarrera** le transfert suivant.
En mode **interruption** (si le bit **SPIE** du **SPCR** est à 1), le passage à 1 de **SPIF** déclenche une interruption de la SPI. (vecteur **$FFD8-$FFD9** en mode circuit seul ou étendu).
Pour acquitter cette interruption, maître et esclave doivent faire refaire passer ce bit **SPIF** à 0 (car sinon on serait en interruption permanente) en lisant le registre **SPSR** (avec **SPIF** à 1) puis en lisant l'octet reçu dans le registre **SPDR**. Et éventuellement, pour le maître et l'esclave écrire l'octet suivant à transmettre dans le **SPDR**. Tout ceci dans la routine d'interruption du maître et de l'esclave.
● **Bit 6** : **WCOL** (**W**rite **COL**lision)
Il passe à 1 pour indiquer une tentative d'écriture dans le registre **SPDR** pendant une phase de transmission.
Pour le remettre à 0, il faut lire le registre **SPSR** (avec **WCOL** à 1) puis en lire le registre **SPDR** (lecture « à vide »).
● **Bit 4** : **MODF** (**MOD**e **F**ault)
Il passe à 1 pour indiquer que plusieurs maîtres ont pris le contrôle en même temps de la SPI, un esclave voulant par exemple devenir maître. (ce qui a lieu si la ligne SS d'un maître passe à 0). Il se passe alors :
- génération d'une interruption SPI (vecteur **$FFD8-$FFD9** en mode circuit seul ou étendu),
- le bit **SPE** est mis à 0, mettant hors service la SPI,
- le bit **MSTR** du maître en cours est mis à 0, ce qui le force en mode esclave,
- les 4 bits **DDRDi** de la **SPI** sont tous mis à 0.
Le bit **MODF** est effacé en lisant le registre **SPSR** (avec **MODF** à 1) puis en **écrivant dans le registre SPCR** pour réinitialiser une nouvelle configuration. Il faudra aussi penser à gérer les bits **DDRDi** compatibles avec cette nouvelle configuration.
- **SPDR** (**S**erial **P**eripheral **D**ata I/O **R**egister) : offset **$2A**
C'est le registre de données dans lequel **le maître** et **l'esclave** viendront **lire** l'octet reçu.
Mais aussi dans lequel **maître** et **esclave** pourront **écrire** l'octet à transmettre.
En mode maître, une écriture dans ce registre lance le transfert. Lorsque le transfert est terminé, ce registre contient la donnée renvoyée par l'esclave, que le maître pourra lire.
En mode esclave, la donnée placée dans ce registre sera renvoyée vers le maître quand ce dernier exécutera un transfert. Lorsque le transfert sera terminé ce registre contiendra la donnée envoyée par le maître, que l'esclave pourra lire.