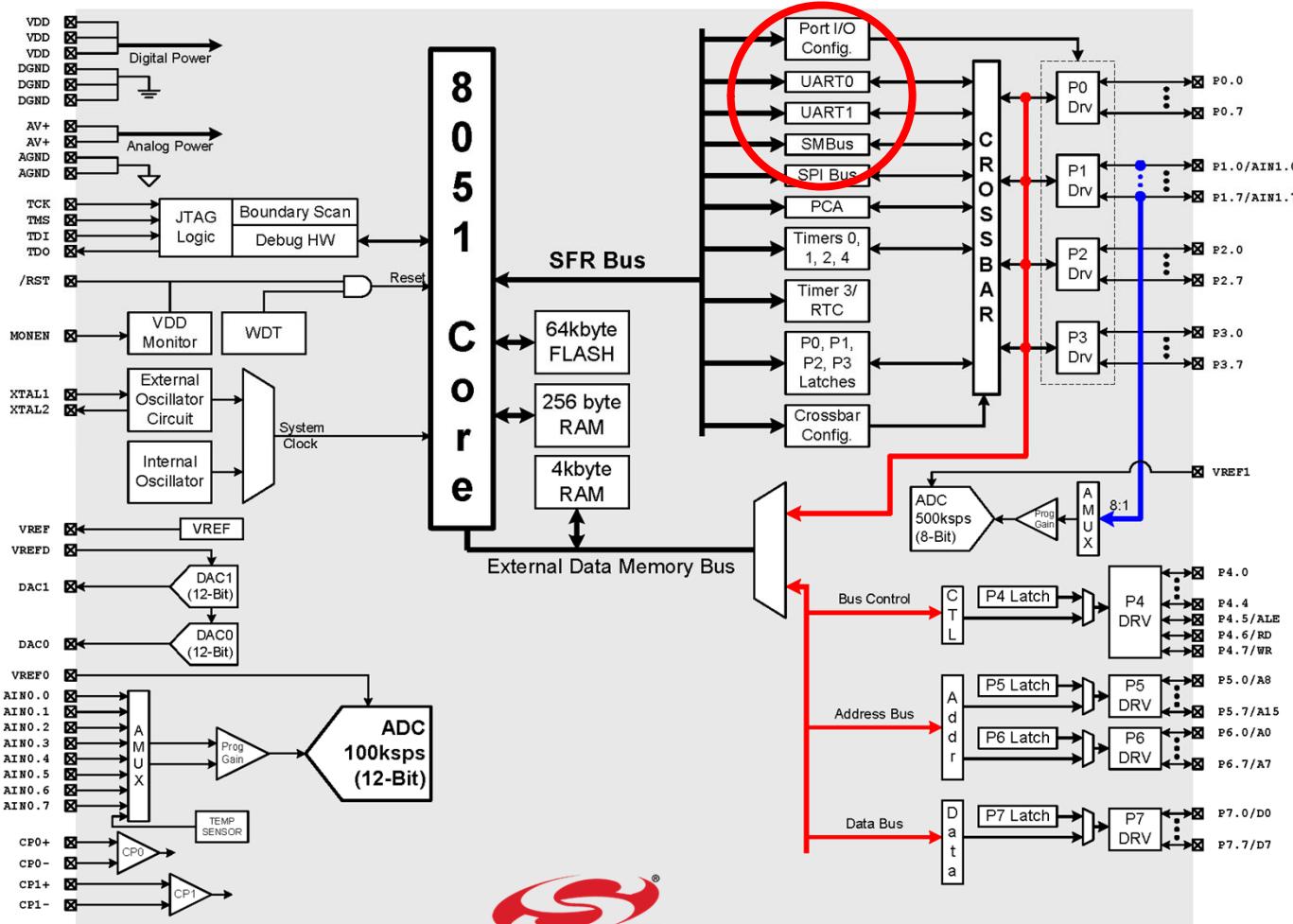


4ETI Bases des systèmes embarqués

A06 – UART0 dans le 8051F020

Version 2022 -- ver 25/11/2022 13:59

Les UARTs dans le 8051F020



2 UARTS Disponibles
Fonctionnement en
Synchrone
unidirectionnel
ou Asynchrone bi-
directionnel



UART et 8051F020

Présence de 2 UART (UART0 et UART1)

Ces 2 dispositifs fonctionnent globalement de la même manière.

Il faudrait plutôt parler d'USART avec S pour synchronous

2 modes de fonctionnement possibles:

- Mode Synchrone
- Mode Asynchrone



Modes de Fonctionnement des UARTs

Table 20.1. UART0 Modes

Mode	Synchronization	Baud Clock	Data Bits	Start/Stop Bits
0	Synchronous	SYSCLK / 12	8	None
1	Asynchronous	Timer 1 or 2 Overflow	8	1 Start, 1 Stop
2	Asynchronous	SYSCLK / 32 or SYSCLK / 64	9	1 Start, 1 Stop
3	Asynchronous	Timer 1 or 2 Overflow	9	1 Start, 1 Stop

LE mode le plus utilisé
Notamment en
« Debug »

Ajout possible d'un 9 ième bit pour gérer une parité

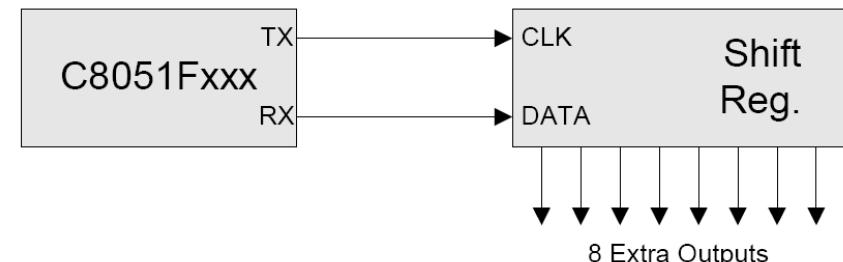
Mode Multi-processor Possible avec transmission d'octets d'adresse et de données sur la liaison série. Mais avantageusement remplacé par des protocoles plus performants tels que l'I2C (SMBus)

Fonctionnement des UARTs en synchrone

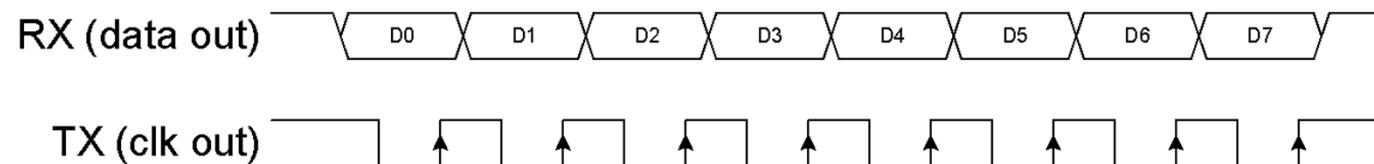


Fonctionnement en Half Duplex exclusivement

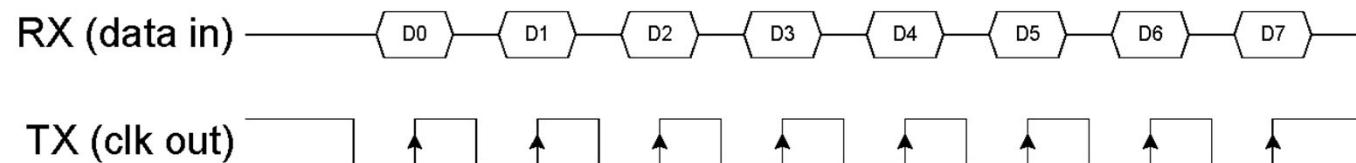
- Solution plus guère utilisée pour communiquer en systèmes
- Mais, c'est une solution basique pour piloter de nombreuses sorties (ou entrées) à l'aide de registres à décalage chainés



MODE 0 TRANSMIT



MODE 0 RECEIVE

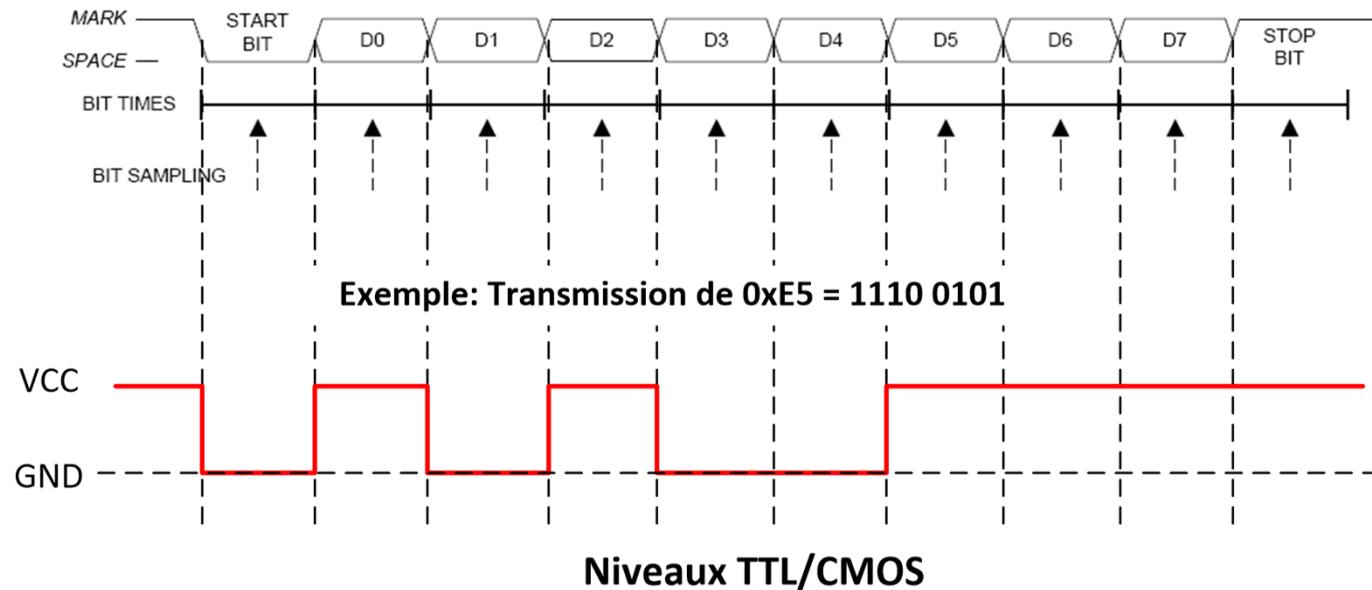




Fonctionnement des UARTs en asynchrone

- Mode Asynchrone 8 ou 9 bits
 - 8 bits: communication type RS232 (7bits + parité ou 8 bits)
 - 9 bits : communication type RS232 avec parité (8 + 1) ou communication multi-processeurs

Chronogramme Transmission de données par UART



Etude de l'UART0: utilisation dans un mode standard: asynchrone – 8bits

Inventaire des registres de UART0

Figure 20.8. SCON0: UART0 Control Register

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value	
SM00/FE0	SM10/RXOV0	SM20/TXCOL0	REN0	TB80	RB80	TI0	RI0	00000000	SFR Address:
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0		0x98
Bits7-6: The function of these bits is determined by the SSTAT0 bit in register PCON. If SSTAT0 is logic 1, these bits are UART0 status indicators as described in Section 20.3 . If SSTAT0 is logic 0, these bits select the Serial Port Operation Mode as shown below. SM00-SM10: Serial Port Operation Mode:									

1 registre de contrôle (configuration + état)

Figure 20.9. SBUF0: UART0 Data Buffer Register

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value	
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	00000000	SFR Address:
Bits7-0: SBUF0.[7:0]: UART0 Buffer Bits 7-0 (MSB-LSB) This SFR accesses two registers; a transmit shift register and a receive latch register. When data is written to SBUF0, it goes to the transmit shift register and is held for serial transmission. Writing a byte to SBUF0 is what initiates the transmission. A read of SBUF0 returns the contents of the receive latch.									

Registres utilisés dans les modes 2 et 3: Transmission Multiprocesseurs
Inutiles dans le cas du TP

Figure 20.10. SADDR0: UART0 Slave Address Register

R/W	Reset Value								
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	00000000	SFR Address:

Figure 20.11. SADEN0: UART0 Slave Address Enable Register

R/W	Reset Value								
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	00000000	SFR Address:

Registre de contrôle SCON0 Bit 7 6 5

Figure 20.8. SCON0: UART0 Control Register

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
SM00/FE0 Bit7	SM10/RXOV0 Bit6	SM20/TXCOL0 Bit5	REN0 Bit4	TB80 Bit3	RB80 Bit2	TI0 Bit1	RI0 Bit0	00000000
								SFR Address: 0x98

Bits7-6: The function of these bits is determined by the SSTAT0 bit in register PCON. If SSTAT0 is logic 1, these bits are UART0 status indicators as described in [Section 20.3](#). If SSTAT0 is logic 0, these bits select the Serial Port Operation Mode as shown below.

SM00-SM10: Serial Port Operation Mode:

SM00	SM10	Mode
0	0	Mode 0: Synchronous Mode
0	1	Mode 1: 8-Bit UART, Variable Baud Rate
1	0	Mode 2: 9-Bit UART, Fixed Baud Rate
1	1	Mode 3: 9-Bit UART, Variable Baud Rate

**Mode Configuration
SSTAT0 = 0 dans PCON**

Configuration

Bit5: SM20: Multiprocessor Communication Enable.
If SSTAT0 is logic 1, this bit is a UART0 status indicator as described in [Section 20.3](#).
If SSTAT0 is logic 0, the function of this bit is dependent on the Serial Port Operation Mode.

Mode 0: No effect.

Mode 1: Checks for valid stop bit.

0: Logic level of stop bit is ignored.

1: RI0 will only be activated if stop bit is logic level 1.

Modes 2 and 3: Multiprocessor Communications Enable.

0: Logic level of ninth bit is ignored.

1: RI0 is set and an interrupt is generated only when the ninth bit is logic 1 and the received address matches the UART0 address or the broadcast address.

Registre PCON

Figure 12.15. PCON: Power Control

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
SMOD0	SSTAT0	Reserved	SMOD1	SSTAT1	Reserved	STOP	IDLE	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address. 0x87

Bit7:

SMOD0: UART0 Baud Rate Doubler Enable. *Pour le timer 1 uniquement*

This bit enables/disables the divide-by-two function of the UART0 baud rate logic for configurations described in the UART0 section.

0: UART0 baud rate divide-by-two enabled.

1: UART0 baud rate divide-by-two disabled.

Bit6:

SSTAT0: UART0 Enhanced Status Mode Select.

This bit controls the access mode of the SM20-SM00 bits in register SCON0.

0: Reads/writes of SM20-SM00 access the SM20-SM00 UART0 mode setting.

1: Reads/writes of SM20-SM00 access the Framing Error (FE0), RX Overrun (RXOV0), and TX Collision (TXCOL0) status bits.

Bit5:

Reserved. Read is undefined. Must write 0.



Registre SCON0 Bit 4 3 2 1 0

Configuration

Etat

- Bit4: REN0: Receive Enable.
This bit enables/disables the UART0 receiver.
0: UART0 reception disabled.
1: UART0 reception enabled.
- Bit3: TB80: Ninth Transmission Bit.
The logic level of this bit will be assigned to the ninth transmission bit in Modes 2 and 3. It is not used in Modes 0 and 1. Set or cleared by software as required.
- Bit2: RB80: Ninth Receive Bit.
The bit is assigned the logic level of the ninth bit received in Modes 2 and 3. In Mode 1, if SM20 is logic 0, RB80 is assigned the logic level of the received stop bit. RB8 is not used in Mode 0.
- Bit1: TI0: Transmit Interrupt Flag.
Set by hardware when a byte of data has been transmitted by UART0 (after the 8th bit in Mode 0, or at the beginning of the stop bit in other modes). When the UART0 interrupt is enabled, setting this bit causes the CPU to vector to the UART0 interrupt service routine. This bit must be cleared manually by software
- Bit0: RI0: Receive Interrupt Flag.
Set by hardware when a byte of data has been received by UART0 (as selected by the SM20 bit). When the UART0 interrupt is enabled, setting this bit causes the CPU to vector to the UART0 interrupt service routine. This bit must be cleared manually by software.



Registre SBUF – 2 registres en un seul!



Figure 20.9. SBUF0: UART0 Data Buffer Register

R/W	Reset Value 00000000	SFR Address: 0x99							
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0		

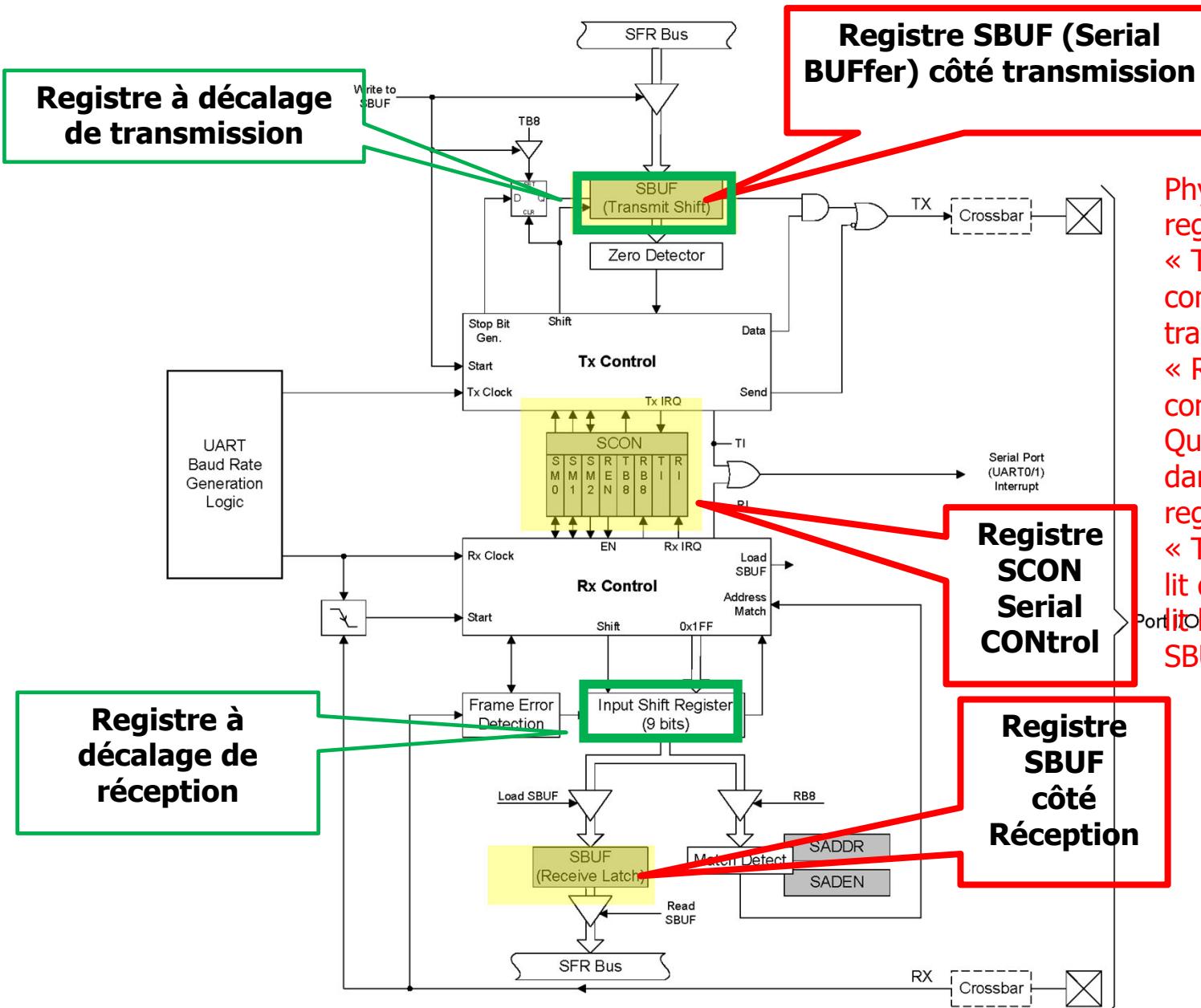
Bits7-0: SBUF0 [7:0]: UART0 Buffer Bits 7-0 (MSB-LSB)
This SFR accesses two registers; a transmit shift register and a receive latch register. When data is written to SBUF0, it goes to the transmit shift register and is held for serial transmission. Writing a byte to SBUF0 is what initiates the transmission. A read of SBUF0 returns the contents of the receive latch.

Il existe physiquement 2 registres SBUF:

- 1 registre SBUF accessible en écriture uniquement qui sert de buffer de transmission
- 1 registre SBUF accessible en lecture uniquement qui sert de buffer de réception

Mais il n'occupe qu'une seule adresse dans l'espace mémoire SFR

Block Diagram UART0



Registre SBUF (Serial BUFFer) côté transmission

Physiquement, il existe 2 registres SBUF: un SBUF « Transmission » qui va contenir la donnée à transmettre et un SBUF « Réception » qui va contenir la donnée reçue. Quand le processeur écrit dans SBUF, il écrit dans le registre SBUF « Transmission », quand il lit dans le registre SBUF, il lit le contenu du registre SBUF « réception »

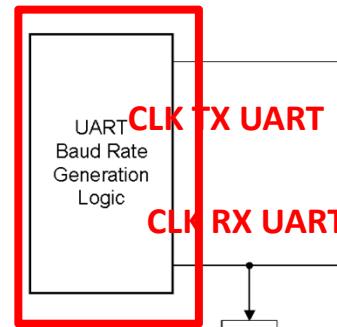
Registre SCON Serial CONtrol

Registre SBUF côté Réception

L'UART0 et ses horloges



Les générateurs d'horloge sont à l'extérieur de l'UART
Possibilité d'avoir des horloges différentes pour la transmission et la réception



C'est la fréquence des horloges internes de l'UART qui fixe:

- la vitesse de transmission sur la liaison

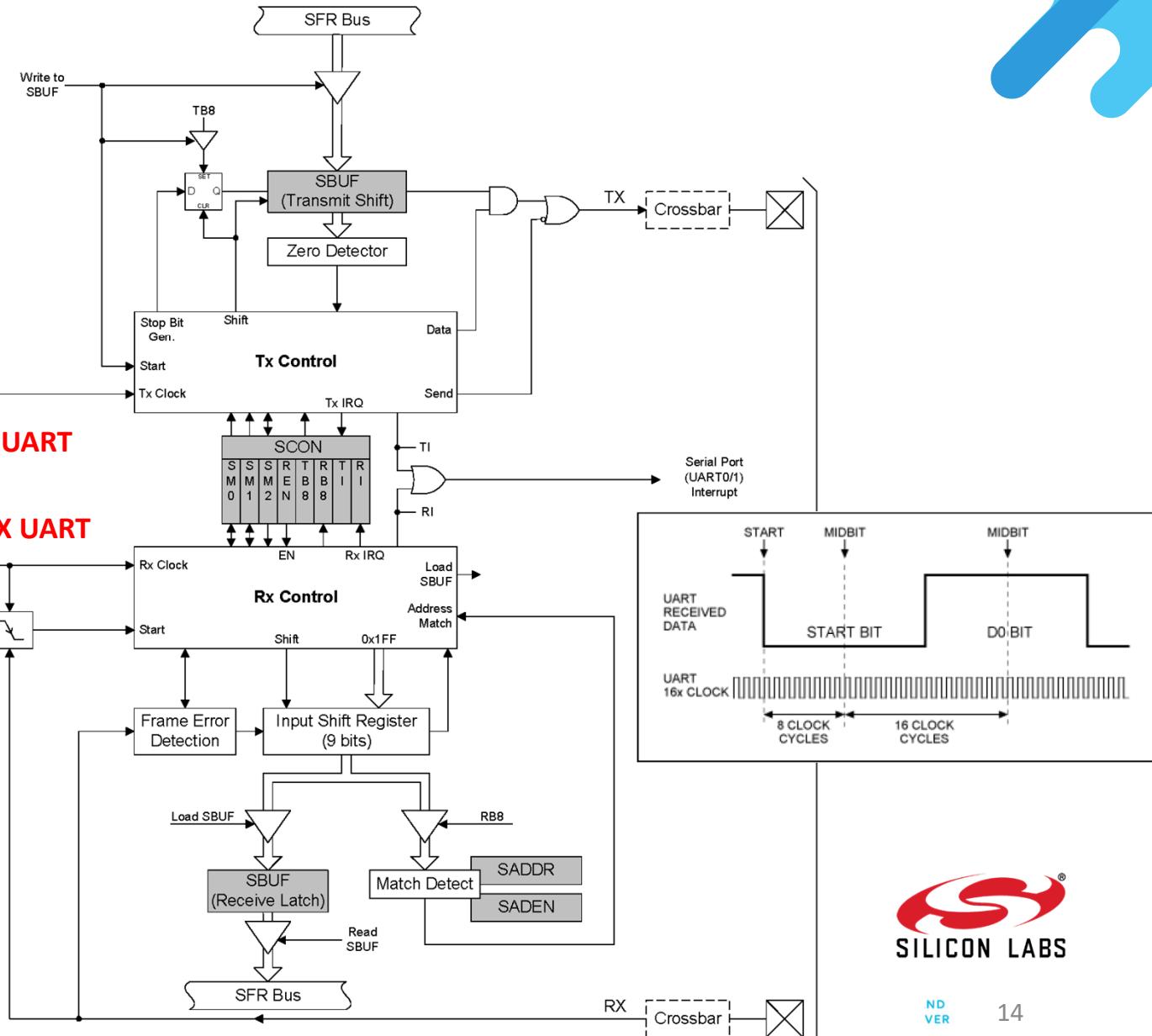
BaudRate TX=

$F_{CLK_TX_UART} / 16$

- La cadence de réception

BaudRate RX=

$F_{CLK_RX_UART} / 16$

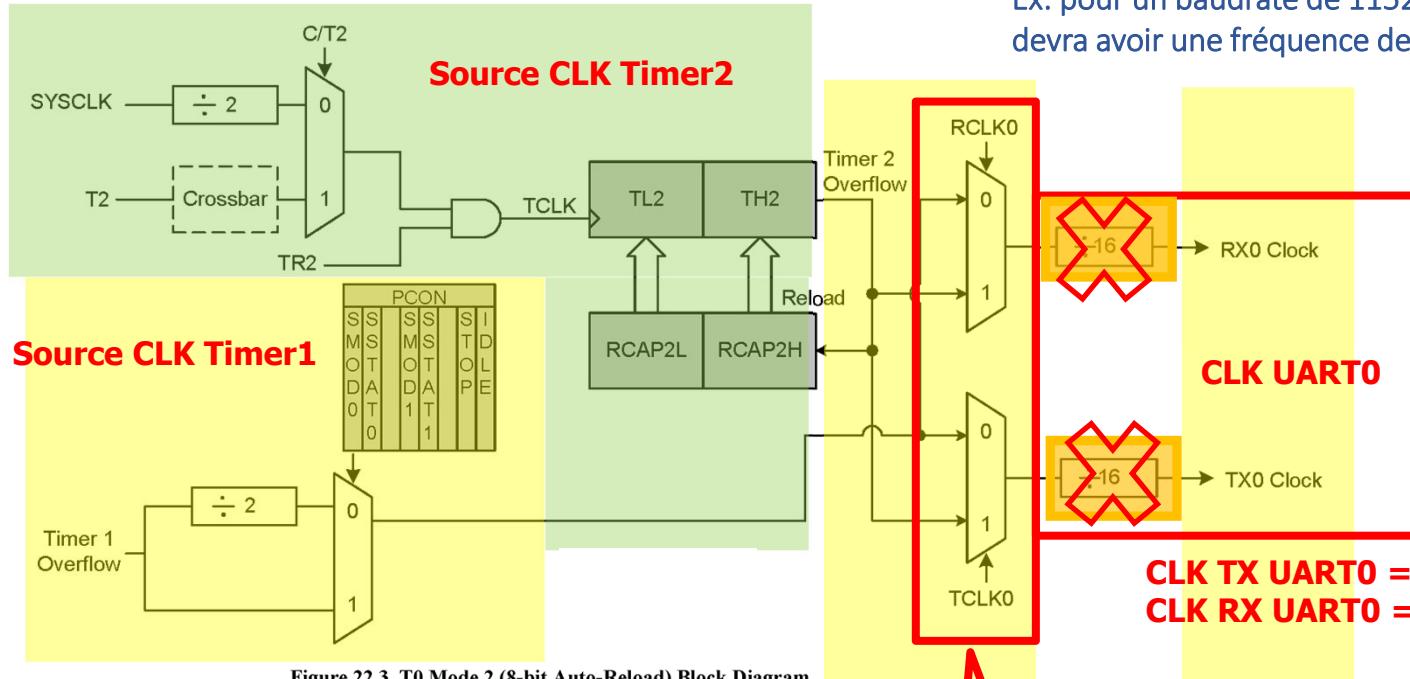


ND
VER

14

Génération d'horloge pour les UART

Figure 22.13. T2 Mode 2 Block Diagram



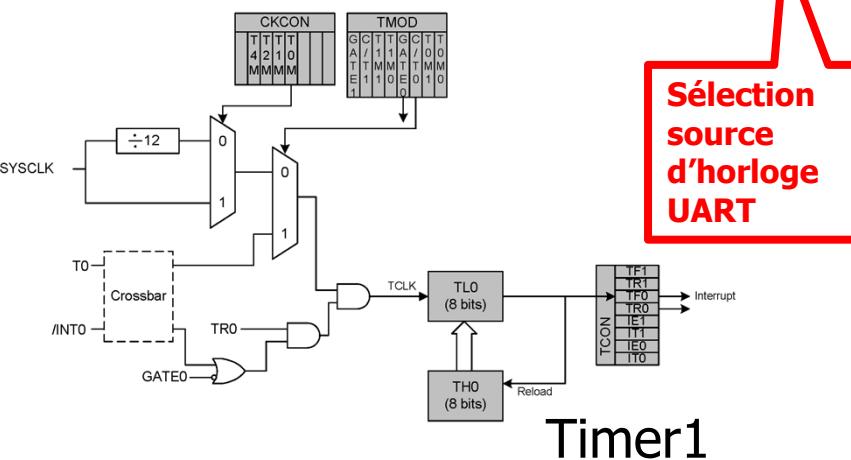
Les horloges à fournir à l'UART doivent avoir des fréquences égales à 16 fois les vitesses de Transmission/Réception (Contraintes de construction)
Ex: pour un baudrate de 115200, l'horloge devra avoir une fréquence de 1,8432Mhz

Représentation ambiguë sur ce schéma:
L'horloge à fournir à l'UART doit être 16 fois supérieure à la vitesse de transmission sur la ligne

$$\text{CLK TX UART0} = 16 \times \text{TX BaudRate}$$

$$\text{CLK RX UART0} = 16 \times \text{RX BaudRate}$$

- Pour l'UART0, 2 sources possibles d'horloge: Timer2 ou/et Timer 1
- Pour l'UART1, 2 sources possibles d'horloge: Timer4 ou/et Timer 1



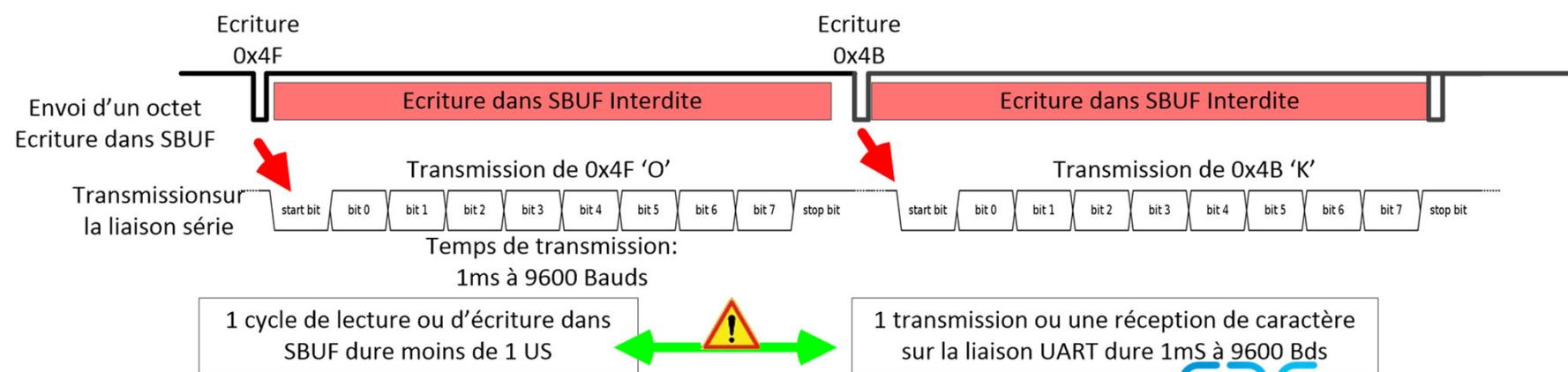
Problématique: Transmission de caractères sur la liaison série



Pour envoyer un caractère sur la liaison série, il « suffit » d'écrire dans SBUF.

Exemple: on souhaite produire l'affichage du message « OK » sur un terminal de commande tel que Putty.

- Cela revient à envoyer conséutivement sur la liaison série les octets de valeur 0x4F ('O') et 0x4B ('K')
- Dans un premier temps on écrit la première valeur à transmettre (0x4F) dans SBUF
- Immédiatement la transmission débute, et le contenu de SBUF subit des décalages bit à bit pour transférer les informations sur la broche TX.
- Cette opération de décalages dure $(1 + 8 + 1) / \text{BaudRate}$ soit environ 1ms pour un BaudRate de 9600
- Durant toute cette durée, **SBUF n'est pas disponible** pour recevoir un autre octet.
- L'écriture dans second octet (0x4B) doit donc être différée d'une durée égale ou supérieure à la durée de transmission d'un octet.
- Si cette condition n'est pas respectée, le résultat est imprévisible



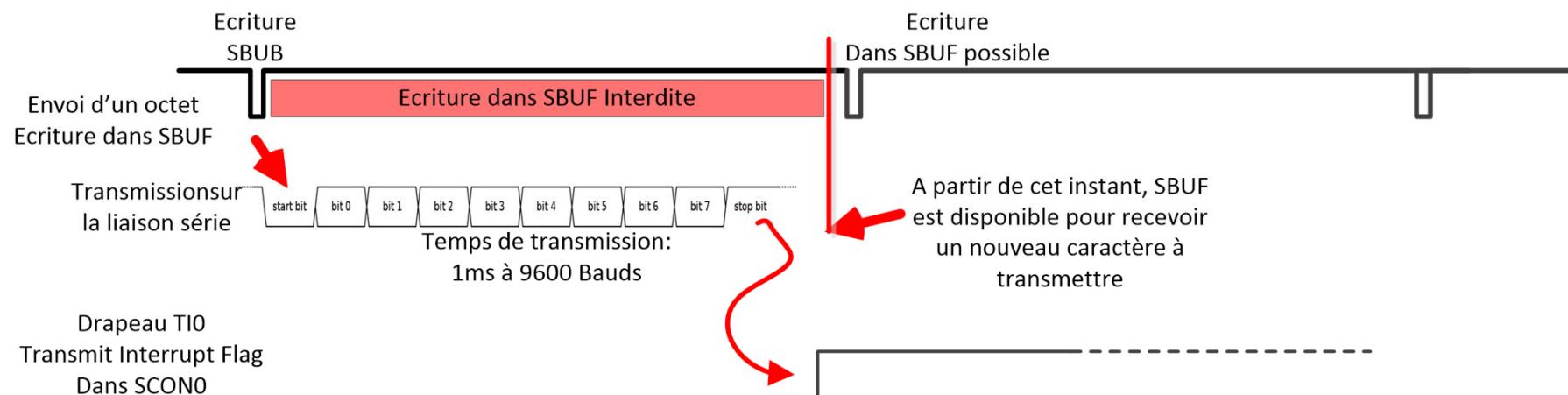
Solution: Transmission de caractères sur la liaison série



Avant d'écrire dans SBUF, il faut s'assurer que SBUF (côté transmission) est « libre », c'est-à-dire s'assurer que le dernier octet écrit a été complètement transmis sur la liaison série.

Les méthodes:

- La « rustique »: attente par temporisation (si elle est recevable pour tester une liaison série (en TP par exemple...), elle n'est pas admissible dans une application « professionnelle »).
- La scrutation : on va surveiller un drapeau qui est chargé de signaler que la transmission du caractère en cours est terminée et que SBUF est donc disponible. Ce **drapeau (TI0)** se trouve dans le registre de contrôle de l'UART. Ne pas oublier de le remettre à zéro....
- L'interruption : ce même drapeau, TI0 est susceptible de provoquer le déclenchement d'une interruption pour signaler que la transmission en cours est terminée et que SBUF est donc disponible.



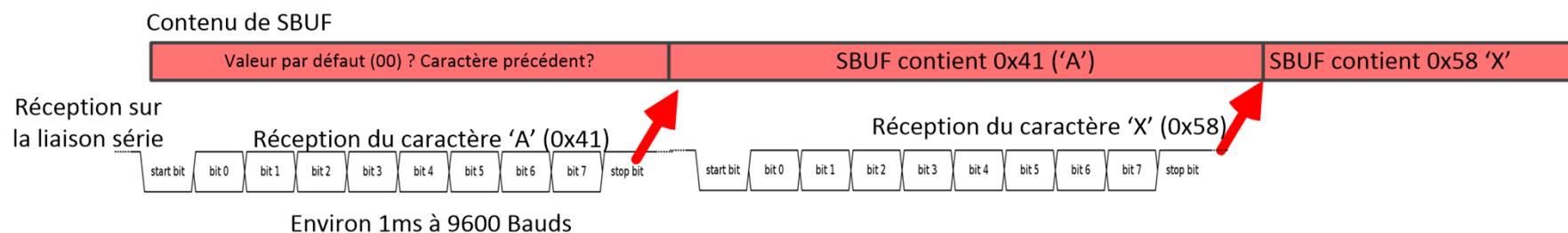
Problématique: Réception de caractères sur la liaison série



Pour récupérer un hypothétique caractère reçu, il « suffit » de lire SBUF

Fonctionnement de la réception dans l'UART:

Lorsque qu'un caractère arrive dans l'UART (une suite de 10 bits, car 1 bit de start, 8 bits de données et un bit de stop), il est reconstitué en un octet dans un registre à décalage, puis lorsque l'octet est entièrement reconstitué, il est transféré dans le registre SBUF de réception. Cette valeur se maintiendra jusqu'à la réception d'un prochain caractère.



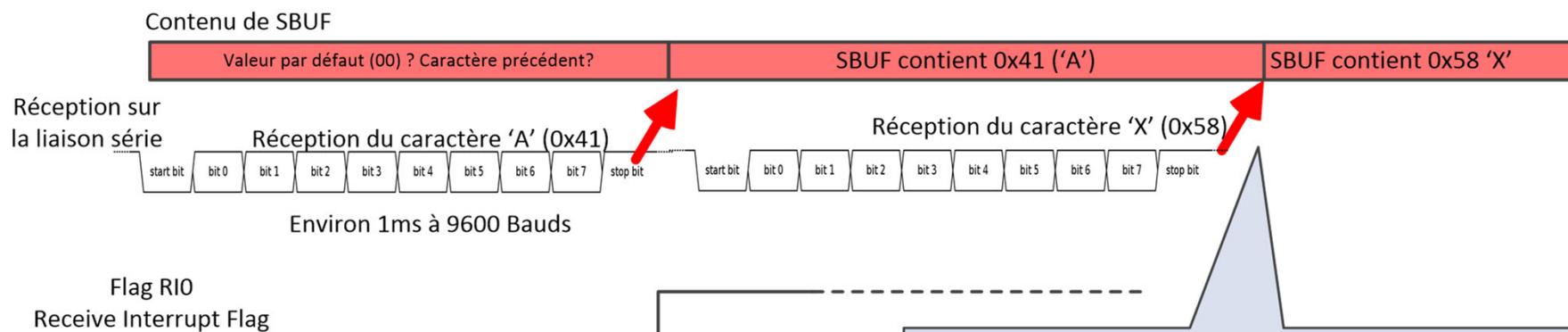
Solution: Réception de caractères sur la liaison série



Avant de lire SBUF il faut s'assurer que SBUF contient bien un caractère nouvellement reçu, si SBUF est « vide » (pas de caractère reçu) ne pas lire, et pareillement ne pas lire plusieurs fois le même caractère reçu

Les méthodes:

- La scrutation : on va surveiller un drapeau qui est chargé de signaler qu'un caractère a été reçu et qu'il est donc disponible dans SBUF. Ce **drapeau (RIO)** se trouve dans le registre de contrôle de l'UART.
- L'interruption : Ce drapeau RIO est susceptible de provoquer le déclenchement d'une interruption pour signaler qu'un nouveau caractère a été reçu et qu'il est disponible dans SBUF.

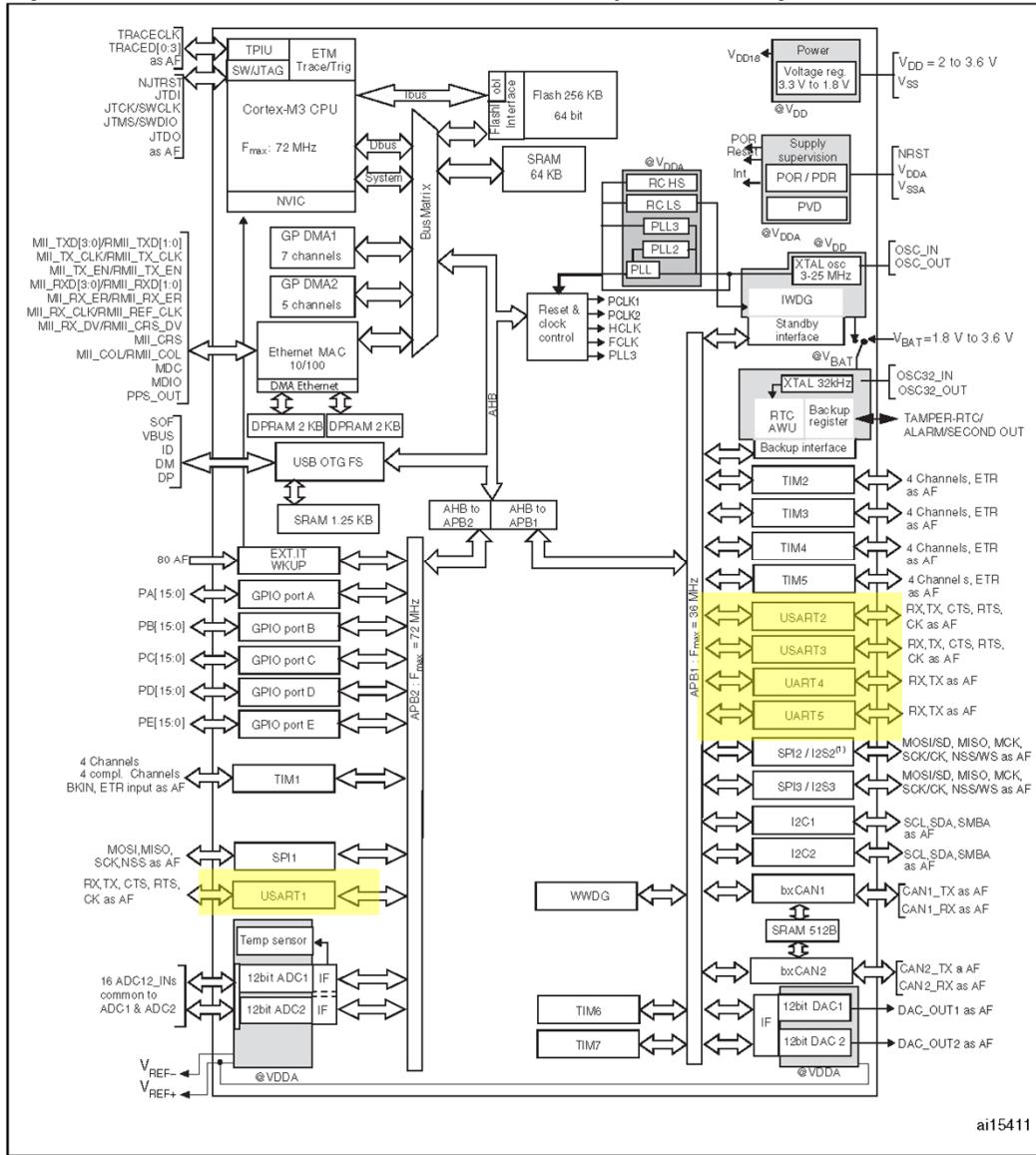


Le mot « Interrupt » dans le nom d'un drapeau, ne signifie pas l'utilisation obligatoire en interruption

En fait si RIO n'est pas remis à zéro, le contenu de SBUF ne sera pas modifié, et conservera la valeur du caractère précédent à priori non lu

Autres horizons: Les UARTs dans le STM32F107 –

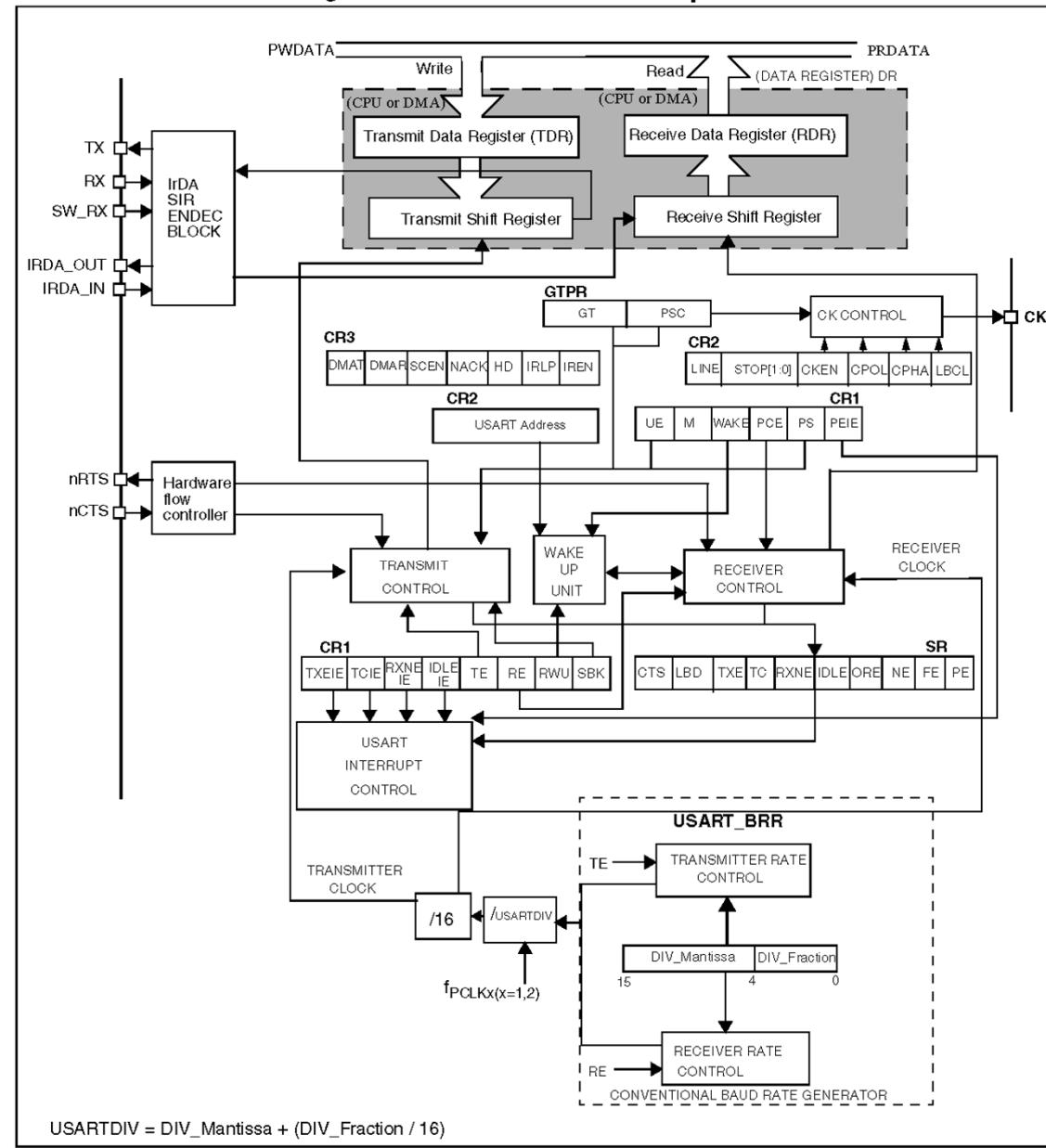
Figure 1. STM32F105xx and STM32F107xx connectivity line block diagram



- 5 UARTS
- Support Li IrDA
- Sync / Async
- Buffer de données via DMA

STM32
F107

27.3.1 USART block diagram USART character description



- 5 UARTS
- Support LIN, IrDA
- Sync / Async
- Buffer de données via DMA

STM32
F107



Exemple de Mise en œuvre de l'UART0 en mode scrutation

2 types de fonctions à coder

- **Fonctions de configuration (appelées une seule fois)**

```
➤void CFG_clock_UART(void) ;  
➤Void CFG_uart0_mode1(void) ;
```

Initialiser l'UART0 pour permettre des réceptions transmissions asynchrones à 9600 bauds avec 8 bits de données, sans parité.

- **Fonctions d'utilisation**

```
➤char Getchar(void) ; réception de caractère par l'UART  
➤void Putchar(char c) ; émission de caractère par l'UART
```

Dépendances avec les configurations globales



Configurations globales du microcontrôleur:

- ✗ Les sources de « Reset »
- ✗ Le chien de garde
- ✓ L' (ou les) horloge(s) - on prendra **impérativement** **SYSCLK = 22,1184 MHz**
- ✗ Les mémoires
- ✗ La gestion de la puissance
- ✓ L'affectation et la configuration des broches d'entrées-sorties (GPIOs).

C'est le quartz externe à 22,1184 MHz, qui peut garantir une précision de l'horloge meilleure que 2% (0,5% environ)

Liaison avec d'autres périphériques

- Timer 1
- Timer 2

Table 14.1. Internal Oscillator Electrical Characteristics

VDD = 2.7V to 3.6V; T_a = -40°C to +85°C

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
Internal Oscillator Frequency	OSCICN.[1:0] = 00	1.5	2	2.4	MHz
	OSCICN.[1:0] = 01	3.1	4	4.8	
	OSCICN.[1:0] = 10	6.2	8	9.6	
	OSCICN.[1:0] = 11	12.3	16	19.2	
Internal Oscillator Current Consumption (from VDD)	OSCICN.2 = 1		200		µA

Oscillateur interne:
précision +/- 20% à
pleine échelle en
température

Configurer l'UART0



1. Générer une horloge de pilotage de l'UART0.

- Choix possible Timer 1 ou Timer 2 -> on choisit le Timer1
- Fréquence à générer: $9600 \times 16 = 153600\text{Hz}$

Car l'horloge qui pilote le fonctionnement interne de l'UART doit être 16 fois plus rapide que le débit sur la liaison série.

2. Configurer l'UART proprement dite



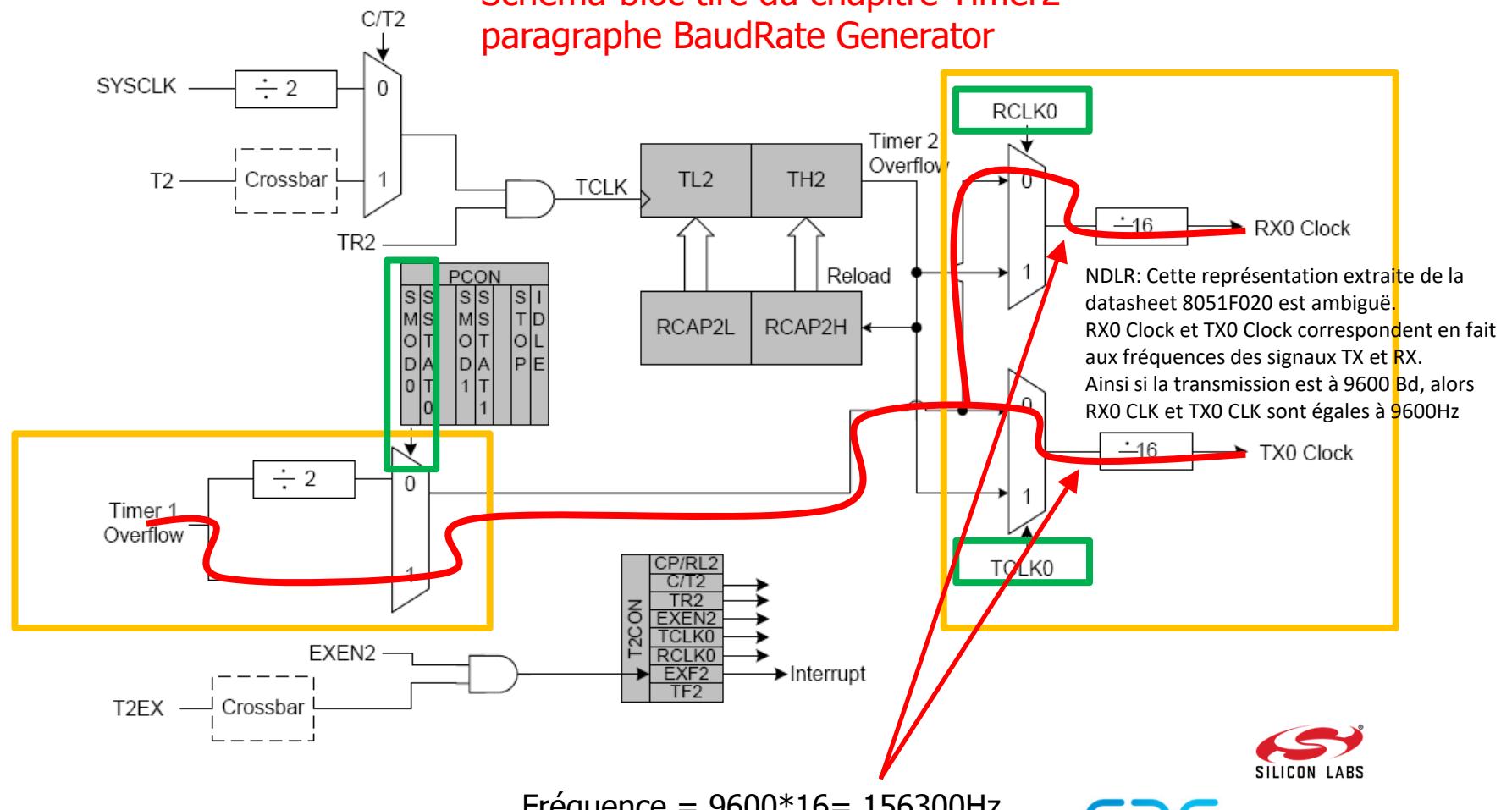
Pourquoi ce choix?
Si un Timer qui fonctionne en auto-rechargement sur 8 bits peut suffire, autant l'utiliser et laisser les Timers 16 bits à d'autres tâches...

Config Horloge UART0



Comme source d'horloge, l'UART0 peut fonctionner avec le Timer 2 ou le Timer 1, le choix a été fait de fonctionner avec le Timer 1

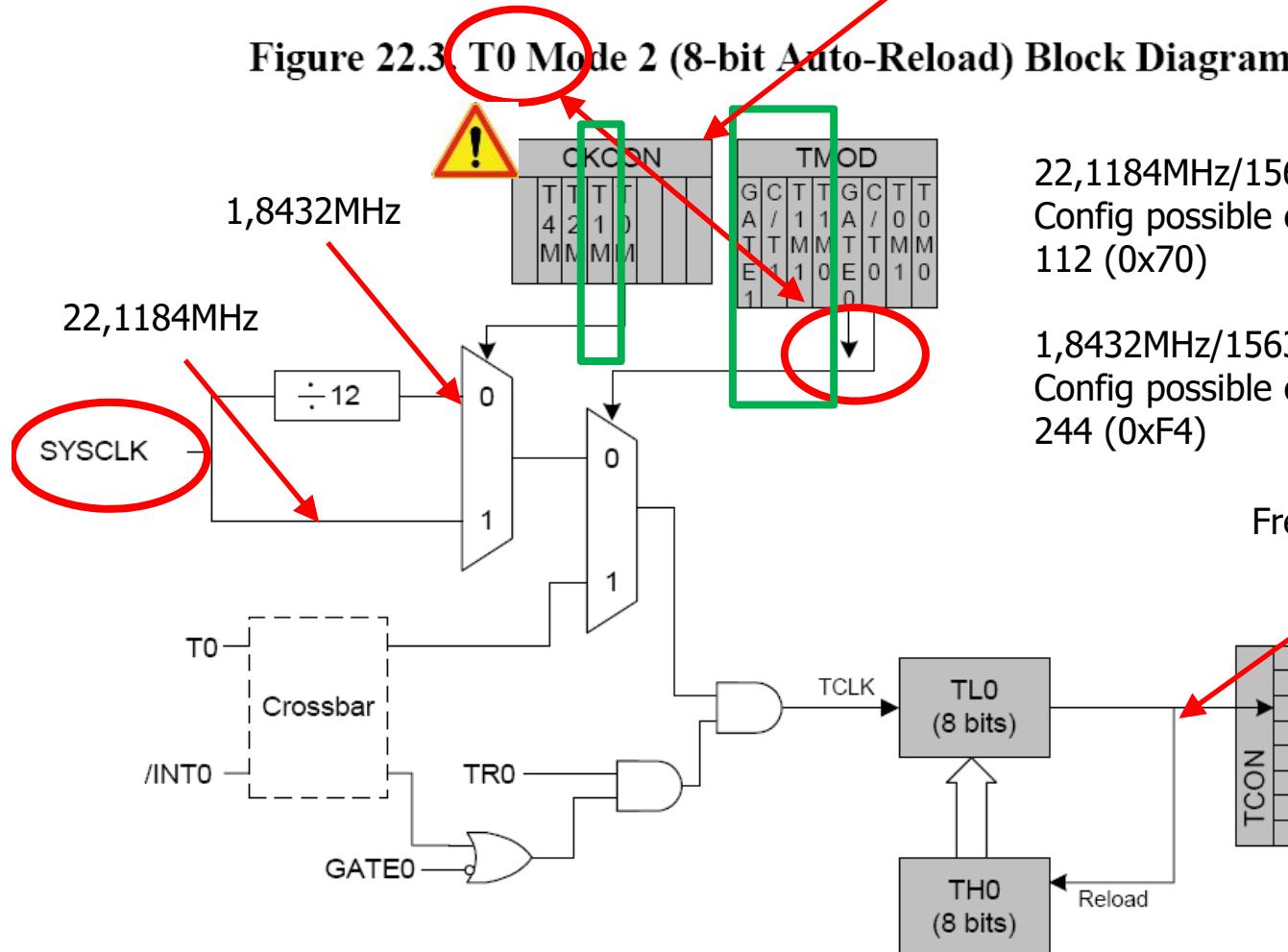
Schéma-bloc tiré du chapitre Timer2 –
paragraphe BaudRate Generator



Config Timer 1 pour UART0

Attention, la figure représente les configurations pour le Timer0.
Or on travaille sur Timer1

Dans CKCON configurer T1M (au lieu de T0M)
Dans TMOD configurer Gate1 et C/T1 au lieu de Gate0 et C/T0



$$22,1184\text{MHz}/156300 = 144$$

Config possible car <256 Reload = 256 -144 = 112 (0x70)

$$1,8432\text{MHz}/156300 = 12$$

Config possible car <256 Reload = 256 -12 = 244 (0xF4)

$$\text{Fréquence} = 9600 * 16 = 156300\text{Hz}$$



Registre CKCON



Figure 22.1. CKCON: Clock Control Register

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value 00000000
-	T4M	T2M	T1M	T0M	Reserved	Reserved	Reserved	SFR Address: 0x8E
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	
X Ne pas modifier!!	Bit7: UNUSED. Read = 0b, Write = don't care.							
X Ne pas modifier!!	Bit6: T4M: Timer 4 Clock Select. This bit controls the division of the system clock supplied to Timer 4. This bit is ignored when the timer is in baud rate generator mode or counter mode (i.e. C/T4 = 1). 0: Timer 4 uses the system clock divided by 12. 1: Timer 4 uses the system clock.							On doit modifier 1 seul bit du registre, on ne connaît pas par ailleurs l'état des autres bits du registre, donc: Masquage obligatoire
1 Ne pas modifier!!	Bit5: T2M: Timer 2 Clock Select. This bit controls the division of the system clock supplied to Timer 2. This bit is ignored when the timer is in baud rate generator mode or counter mode (i.e. C/T2 = 1). 0: Timer 2 uses the system clock divided by 12. 1: Timer 2 uses the system clock.							
	Bit4: T1M: Timer 1 Clock Select. This bit controls the division of the system clock supplied to Timer 1. 0: Timer 1 uses the system clock divided by 12. 1: Timer 1 uses the system clock.							Choix Timer1 CLK = SYSCLOCK
X Ne pas modifier!!	Bit3: T0M: Timer 0 Clock Select. This bit controls the division of the system clock supplied to Counter/Timer 0. 0: Counter/Timer uses the system clock divided by 12. 1: Counter/Timer uses the system clock.							
Bits2-0:	Reserved. Read = 000b, Must Write = 000.							

Registre TMOD



Figure 22.6. TMOD: Timer Mode Register

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
GATE1	C/T1	T1M1	T1M0	GATE0	C/T0	T0M1	T0M0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address:
								0x89

Pas accessible bit à bit!!

Bit7: GATE1: Timer 1 Gate Control.
0
 0: Timer 1 enabled when TR1 = 1 irrespective of /INT1 logic level.
 1: Timer 1 enabled only when TR1 = 1 AND /INT1 = logic 1.

Bit6: C/T1: Counter/Timer 1 Select.
0
 0: Timer Function: Timer 1 incremented by clock defined by T1M bit (CKCON.4).
 1: Counter Function: Timer 1 incremented by high-to-low transitions on external input pin (T1).

Bits5-4: T1M1-T1M0: Timer 1 Mode Select.
 These bits select the Timer 1 operation mode.

On doit modifier 4 bits du registre relatifs au Timer1, on ne connaît pas par ailleurs l'état des 4 autres bits du registre dédiés au timer0, donc:
Masquage obligatoire

T1M1	T1M0	Mode
0	0	Mode 0: 13-bit counter/timer
0	1	Mode 1: 16-bit counter/timer
1	0	Mode 2: 8-bit counter/timer with auto-reload
1	1	Mode 3: Timer 1 inactive

Registre TCON



Figure 22.5. TCON: Timer Control Register

R/W	Reset Value							
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: (bit addressable) 0x88

Bit7: TF1: Timer 1 Overflow Flag.
Set by hardware when Timer 1 overflows. This flag can be cleared by software but is automatically cleared when the CPU vectors to the Timer 1 interrupt service routine.

0
0: No Timer 1 overflow detected.
1: Timer 1 has overflowed.

Bit6: TR1: Timer 1 Run Control.
0: Timer 1 disabled.
1
1: Timer 1 enabled.

Bit5: TF0: Timer 0 Overflow Flag.
Set by hardware when Timer 0 overflows. This flag can be cleared by software but is automatically cleared when the CPU vectors to the Timer 0 interrupt service routine.

X



Code Config CLK UARTO

Fonction CFG_clock_UART - Utilisation du Timer 1

Timer1 Stoppé (précaution) - Action sur TR1 de TCON

Flag Timer1 effacé - - Action sur TF1 de TCON

Config CKCON - | T1M: Timer1 ClockSelect - CLK Timer = Sysclk

Config TMOD - Timer1 configuré en timer 8 bit avec auto-reload

Programmation du registre de rechargement TH1

Initialisation du registre Timer TL1 - Nécessaire?

Dévalidation de l'interruption Timer1 - Action sur registre IE

Timer1 démarré - - Action sur TR1 de TCON

Configurer l'UART0



Générer une horloge de pilotage de l'UART0.

- Choix possible Timer 1
- Fréquence à générer: $9600 \times 16 = 153600\text{Hz}$

2. Configurer l'UART – Agir sur SCON0



Config SCON0 Bits 7....5

Figure 20.8. SCON0: UART0 Control Register

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
SM00/FE0 Bit7	SM10/RXOV0 Bit6	SM20/TXCOL0 Bit5	REN0 Bit4	TB80 Bit3	RB80 Bit2	TI0 Bit1	RI0 Bit0	00000000 SFR Address: 0x98

Bits7-6: The function of these bits is determined by the SSTAT0 bit in register PCON. If SSTAT0 is logic 1, these bits are UART0 status indicators as described in [Section 20.3](#). If SSTAT0 is logic 0, these bits select the Serial Port Operation Mode as shown below.

01 SM00-SM10: Serial Port Operation Mode:

SM00	SM10	Mode
0	0	Mode 0: Synchronous Mode
0	1	Mode 1: 8-Bit UART, Variable Baud Rate
1	0	Mode 2: 9-Bit UART, Fixed Baud Rate
1	1	Mode 3: 9-Bit UART, Variable Baud Rate

1 Bit5: SM20: Multiprocessor Communication Enable. If SSTAT0 is logic 1, this bit is a UART0 status indicator as described in [Section 20.3](#). If SSTAT0 is logic 0, the function of this bit is dependent on the Serial Port Operation Mode.

Mode 0: No effect.

Mode 1: Checks for valid stop bit.

0: Logic level of stop bit is ignored.

1: RI0 will only be activated if stop bit is logic level 1.

Modes 2 and 3: Multiprocessor Communications Enable.

0: Logic level of ninth bit is ignored.

1: RI0 is set and an interrupt is generated only when the ninth bit is logic 1 and the received address matches the UART0 address or the broadcast address.

Mode Configuration
SSTAT0 = 0
dans PCON

Config SCON0 Bits 4....0



Bit4:	REN0: Receive Enable. This bit enables/disables the UART0 receiver. 1 0: UART0 reception disabled. 1: UART0 reception enabled.
Bit3: Non utilisés	TB80: Ninth transmission Bit. The logic level of this bit will be assigned to the ninth transmission bit in Modes 2 and 3. It is not used in Modes 0 and 1. Set or cleared by software as required.
Bit2: Non utilisés	RB80: Ninth Receive Bit. The bit is assigned the logic level of the ninth bit received in Modes 2 and 3. In Mode 1, if SM20 is logic 0, RB80 is assigned the logic level of the received stop bit. RB8 is not used in Mode 0.
Bit1:	TI0: Transmit Interrupt Flag. Set by hardware when a byte of data has been transmitted by UART0 (after the 8th bit in Mode 0, or at the beginning of the stop bit in other modes). When the UART0 interrupt is enabled, setting this bit causes the CPU to vector to the UART0 interrupt service routine. This bit must be cleared manually by software 0
Bit0:	RI0: Receive Interrupt Flag. Set by hardware when a byte of data has been received by UART0 (as selected by the SM20 bit). When the UART0 interrupt is enabled, setting this bit causes the CPU to vector to the UART0 interrupt service routine. This bit must be cleared manually by software. 0

Configuration Registre T2CON_ Bits 7-6-5

On configure ici la source d'horloge pour l'UART: Timer2 ou Timer 1

Figure 22.14. T2CON: Timer 2 Control Register

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value			
TF2	EXF2	RCLK0	TCLK0	EXEN2	TR2	C/T2	CP/RL2	00000000			
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: (bit addressable) 0xC8			
Bit7:	TF2: Timer 2 Overflow Flag. Set by hardware when Timer 2 overflows. When the Timer 2 interrupt is enabled, setting this bit causes the CPU to vector to the Timer 2 interrupt service routine. This bit is not automatically cleared by hardware and must be cleared by software. TF2 will not be set when RCLK0 and/or TCLK0 are logic 1.	X	Bit6:	EXF2: Timer 2 External Flag. Set by hardware when either a capture or reload is caused by a high-to-low transition on the T2EX input pin and EXEN2 is logic 1. When the Timer 2 interrupt is enabled, setting this bit causes the CPU to vector to the Timer 2 Interrupt service routine. This bit is not automatically cleared by hardware and must be cleared by software.	X	Bit5:	RCLK0: Receive Clock Flag for UART0. Selects which timer is used for the UART0 receive clock in modes 1 or 3. 0: Timer 1 overflows used for receive clock. 1: Timer 2 overflows used for receive clock.	0	Bit4:	TCLK0: Transmit Clock Flag for UART0. Selects which timer is used for the UART0 transmit clock in modes 1 or 3. 0: Timer 1 overflows used for transmit clock. 1: Timer 2 overflows used for transmit clock.	0

Registre PCON



Figure 12.15. PCON: Power Control

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
SMOD0	SSTAT0	Reserved	SMOD1	SSTAT1	Reserved	STOP	IDLE	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0x87

- Bit7: **SMOD0: UART0 Baud Rate Doubler Enable.**
This bit enables/disables the divide-by-two function of the UART0 baud rate logic for configurations described in the UART0 section.
1
0: UART0 baud rate divide-by-two enabled.
1: UART0 baud rate divide-by-two disabled.
- Bit6: **SSTAT0: UART0 Enhanced Status Mode Select.**
This bit controls the access mode of the SM20-SM00 bits in register SCON0.
0
0: Reads/writes of SM20-SM00 access the SM20-SM00 UART0 mode setting.
1: Reads/writes of SM20-SM00 access the Framing Error (FE0), RX Overrun (RXOV0), and TX Collision (TXCOL0) status bits.
- Bit5: Reserved. Read is undefined. Must write 0.

Code Config UART0

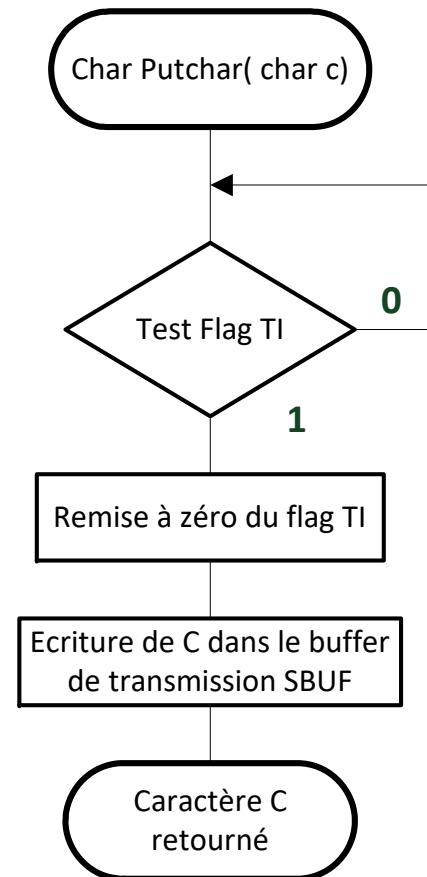


Fonction CFG_uart0_mode1

L'ordre de ces opérations est importante (surtout sur PCON/SCON0)

- Sélection de la source d'horloge (Timer1) pour l'UART - Action sur T2CON
- Configuration de PCON - UART0 Baud Rate Doubler Disabled et accès à SM20-SM00 de SCON0
- Configuration de SCON0 - Mode 1 - Check Stop bit - Réception validée

Code de char putchar(char)



Durée max de la
scrutation: le temps de
transmission d'un
caractère si la
configuration est
correcte; infinie en cas
de mauvaise
configuration!!

CORRECTION

Code Putchar

```
*****  
//PUTCHAR  
*****  
  
char Putchar(char c)  
{  
    while (!TI0) : // Traitement UART pas prête à transmettre  
        // Risque de blocage!!!  
    TI0 = 0;          // Raz flag octet transmis  
    SBUF0 = c;        // Traitement UART prête à transmettre  
    return c;  
}
```

Contrainte sur config
de SCON0?

Après la configuration, si TI est mis à zéro, alors on ne pourra jamais sortir du while, car TI ne passera jamais à 1, étant donné que l'on n'a pas pu transmettre le premier caractère...

Intérêt de placer le test du drapeau TI avant l'envoi dans SBUF?
Quelles sont les conséquences si on inverse l'ordre?

Config SCON0 Bits 4....0

Si utilisation du Putchar
précédemment codé
« Amorçage » de la transmission

- Bit4: REN0: Receive Enable.
This bit enables/disables the UART0 receiver.
1
0: UART0 reception disabled.
1: UART0 reception enabled.
- Bit3: TB80: Ninth Transmission Bit
X
The logic level of this bit will be assigned to the ninth transmission bit in Modes 2 and 3. It is not used in Modes 0 and 1. Set or cleared by software as required.
- Bit2: RB80: Ninth Receive Bit.
X
The bit is assigned the logic level of the ninth bit received in Modes 2 and 3. In Mode 1, if SM20 is logic 0, RB80 is assigned the logic level of the received stop bit. RB8 is not used in Mode 0.
- Bit1: TI0: Transmit Interrupt Flag.
1!!
Set by hardware when a byte of data has been transmitted by UART0 (after the 8th bit in Mode 0, or at the beginning of the stop bit in other modes). When the UART0 interrupt is enabled, setting this bit causes the CPU to vector to the UART0 interrupt service routine. This bit must be cleared manually by software.
- Bit0: RI0: Receive Interrupt Flag.
0
Set by hardware when a byte of data has been received by UART0 (as selected by the SM20 bit). When the UART0 interrupt is enabled, setting this bit causes the CPU to vector to the UART0 interrupt service routine. This bit must be cleared manually by software.

UART0 avancée



Frame Error: Erreur de Trame: Stop bit non valide

RX Overrun: le caractère reçu n'a pas été lu à temps

TX Collision: écriture dans SBUF avant la fin de la transmission en cours

Des drapeaux supplémentaires dans SCON0 permettent la détection d'erreurs....

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
SM00/FE0	SM10/RXOVR0	SM20/TXCOL0	REN0	TB80	RB80	TI0	RI0	
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	

20.3. Frame and Transmission Error Detection

Frame error detection is available in the following modes when the SSTAT0 bit in register PCON is set to logic 1. Note: The SSTAT0 bit must be logic 1 to access any of the status bits (FE0, RXOVR0, and TXCOL0). To access the UART0 Mode Select bits (SM00, SM10, and SM20), the SSTAT0 bit must be logic 0.

All Modes:

The Transmit Collision bit (TXCOL0 bit in register SCON0) reads ‘1’ if user software writes data to the SBUF0 register while a transmit is in progress. Note that the TXCOL0 bit also functions as the SM20 bit when the SSTAT0 bit in register PCON is logic 0.

Modes 1, 2, and 3:

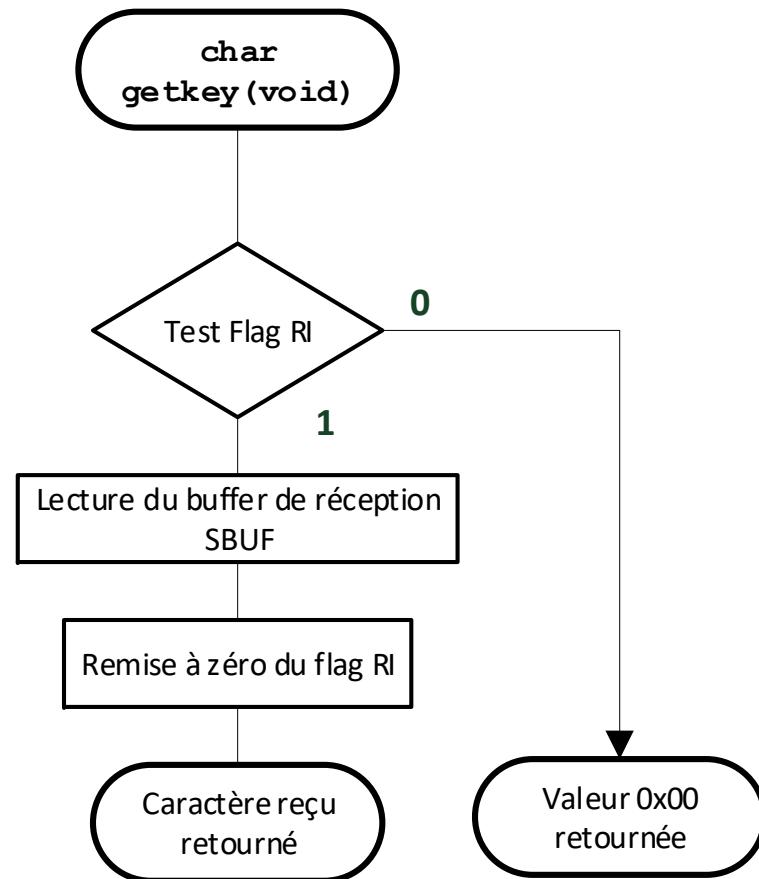
The Receive Overrun bit (RXOVR0 in register SCON0) reads ‘1’ if a new data byte is latched into the receive buffer before software has read the previous byte. Note that the RXOVR0 bit also functions as the SM10 bit when the SSTAT0 bit in register PCON is logic 0.

The Frame Error bit (FE0 in register SCON0) reads ‘1’ if an invalid (low) STOP bit is detected. Note that the FE0 bit also functions as the SM00 bit when the SSTAT0 bit in register PCON is logic 0.

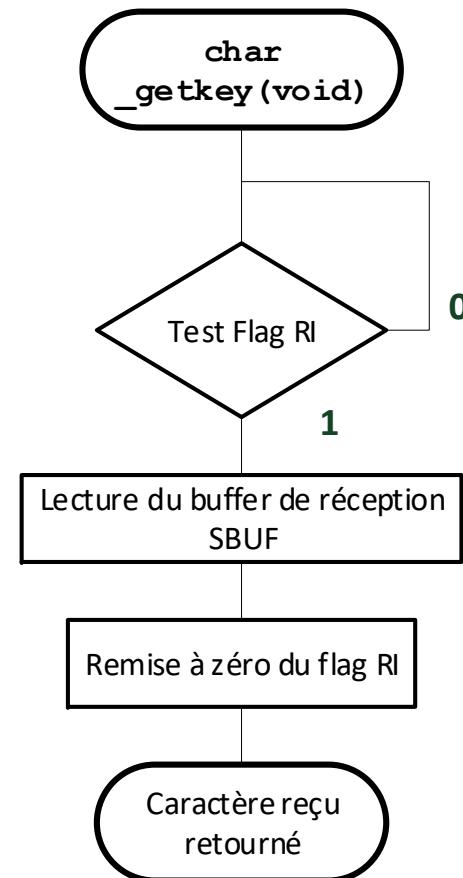
Code de `char getkey(void)` ou `_getkey(void)`



Version non bloquante `getkey()`



Version bloquante `_getkey()`
Utilisée par les fonctions telles que `getchar`, `gets`, `scanf`





Pause - Critique Getkey/Putchar

Getchar:

Problème?

Solution?

Putchar:

Problème?

Solution?

CORRECTION

Critique getkey/putchar



getkey:

Problème?

1. Réception d'une valeur de 0 à 0xFF impossible (la valeur 0 est exclue)
2. Pour ne pas perdre de caractère, la vitesse de scrutation doit être rapide ($86\mu\text{s}$ à 115200Bd)

Solution?

1. Retourner un entier 16 bits (par exemple, poids fort: valeur reçue et poids faible: validité de la valeur)
2. Fonctionnement en interruption

putchar:

Problème?

- Boucle infinie si Timer mal configuré
- Cas d'une transmission d'une chaîne de caractères? Temps d'exécution important et beaucoup de temps perdu à attendre le flag de fin de transmission

Solution?

- Timeout (Cf TP)
- Fonctionnement en Interruption – Mise en oeuvre d'une gestion par buffer.

Gestion UART par Interruption



Remarque: l'interruption UART peut être déclenchée par 2 sources: le drapeau TI0 qui indique l'achèvement d'une transmission et le drapeau RI0 qui indique la réception d'un caractère.

Conséquence: détection impérative de la source de l'interruption dans le code d'interruption.

Exemple de code d'interruption UART0:

```
If (RI0 == 1)
{
    RI0 = 0;
    traitement réception }

If (TI0 == 1)
{
    TI0 = 0;
    traitement transmission }
```

Question: peut-on gérer l'UART par interruption dans une direction d'échange et en scrutation dans l'autre direction d'échange?

Par exemple: réception par interruption et transmission par scrutation

Approche Gestion UART par Interruption



Règle de base: un programme d'interruption doit être le plus bref possible – pas question de gérer l'application dans cette interruption

- Configuration Timer et UART rigoureusement identique
- Abandon des fonctions de scrutation Putchar et Getchar
- Autoriser l'interruption UART0 et autoriser globalement les interruptions
- Contenu de la fonction d'interruption IST_UART0:
 - Tester Interruption causée par une réception (RI=1)?
 - Si oui: récupérer le caractère reçu, pour le mettre où? Char en Variable globale? Tableau de char?
 - Effacer drapeau RI.
 - Tester Interruption causée par une fin de transmission (TI=1)?
 - Si oui: écrire dans SBUF le caractère à transmettre, où se trouve-t-il?? Char en Variable globale? Tableau de char?
 - Effacer drapeau TI.

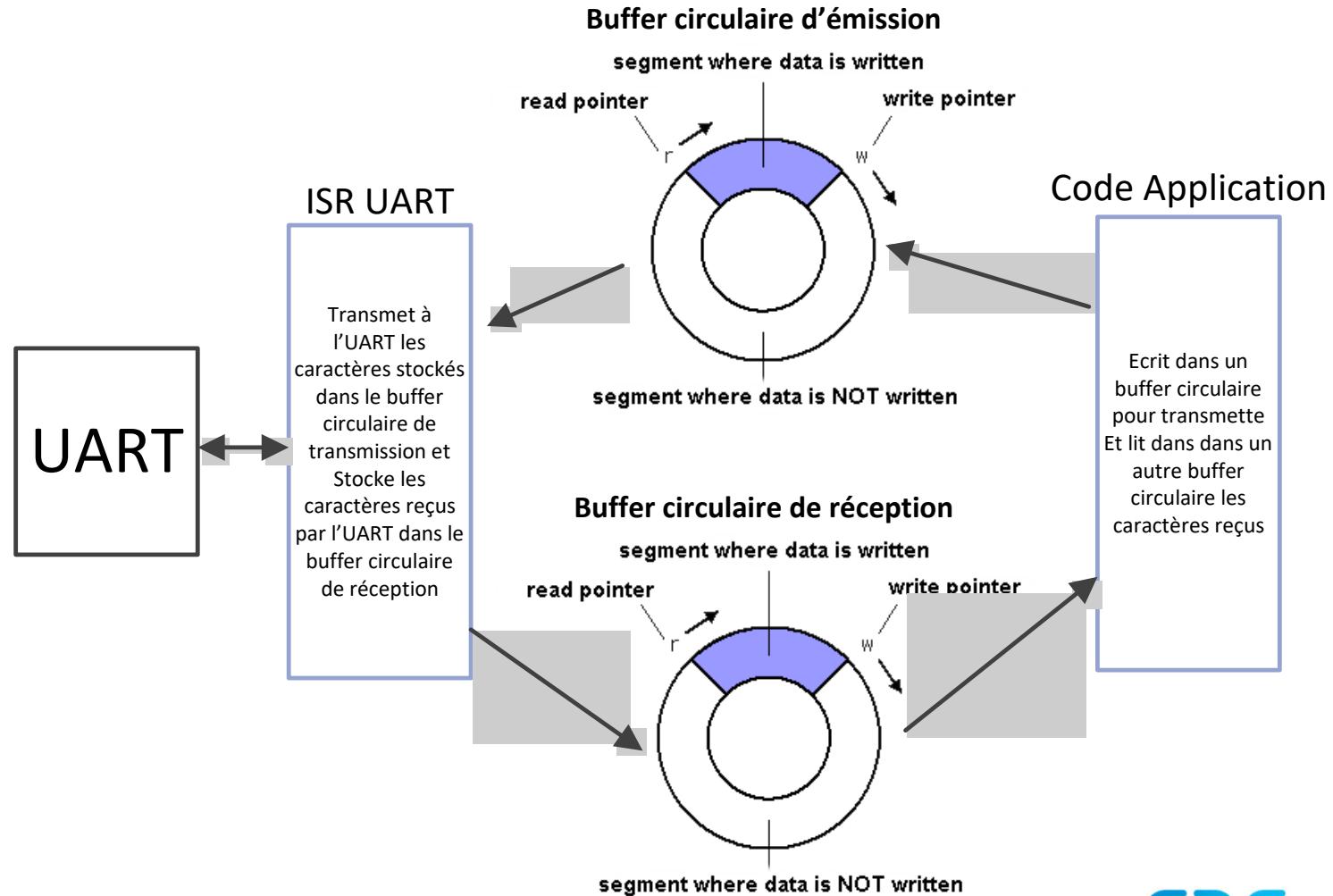
Approche Gestion UART par Interruption – Questions annexes



Règle de base (bis) : un programme d'interruption doit être le plus bref possible – pas question de gérer l'application dans cette interruption (la seule exception est celle d'une interruption timer utilisée comme base de temps).

- Cas d'un caractère à transmettre – comment déclencher l'interruption? Mise à 1 de TI par logiciel? Risqué si transmission en cours → Savoir si une transmission est en cours ou pas...
- Transmission d'une chaîne de caractères, comment faire?
- Cas de la réception – réception d'une chaîne de caractères, comment traiter?

Gestion UART par Interruption et buffer circulaire



Fichiers Configurables: Gestion des Entrées/Sorties



Basic I/O (Doc Keil)

- The following files contain the source code for the low-level stream I/O routines. When you use µVision IDE, you can simply add the modified versions to the project.
 - C Source File Description
-
- **PUTCHAR.C** Used by all stream routines that output characters. **You may adapt this routine to your individual hardware** (for example, LCD or LED displays).
The default version outputs characters via the serial interface. An **XON/XOFF** protocol is used for flow control. Line feed characters ('\n') are converted into carriage return/line feed sequences ('\r\n').
 - **GETKEY.C** Used by all stream routines that input characters. **You may adapt this routine to your individual hardware** (for example, matrix keyboards). The default version reads a character via the serial interface. No data conversions are performed..

Dans la librairie stdio, putchar et _getkey sont les fonctions de base pour toutes les fonctions de gestion d'entrées-sorties telles que **printf** et **scanf**. Par défaut, **putchar** et **getkey** utilisent l'UART0. **getkey** est bloquant. Si vous avez des fonctions putchar et/ou getkey dans votre code, elles remplaceront automatiquement les fonctions d'origine à la compilation.



LIVE AND
DISCOVER

Contact

François JOLY
Tél. : 04 72 43 13 36
francois.joly@cpe.fr

www.cpe.fr