APPLICATION NOTE

ST7 UART EMULATION SOFTWARE

by Microcontroller Division Application Team

1 INTRODUCTION

All members of the STMicroelectronics ST7 Series of Microcontrollers feature a 16 bit timer with several possibilities such as output compares and input captures.

This note describes a technique for emulating an RS232 UART with the ST7 timer without any additional hardware. Only two pins are required for the serial communication.

The first part of this note will explain the protocol used for serial communication and how to adapt it for the ST7 timer. The other sections of this note describe more precisely how the program deals with transmitter mode and receiver mode.

Timings are used to illustrate the important points.

The user can easily adapt the example to his own application as only a small amount of code is required by the UART program.

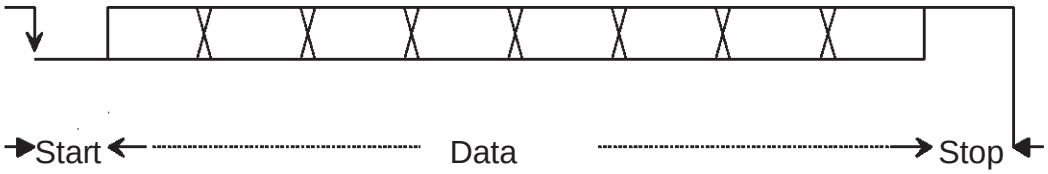
The software was tested by connecting a ST72251 to the serial port of a PC and communicating in all possible modes.

2 UART COMMUNICATIONS PROTOCOL

An UART is a two-way communications interface (Universal Asynchronous Receiver Transmitter).

A hardware UART is made of two cells and can work in FULL DUPLEX mode (transmitter and receiver together). The software module developed below supports full duplex functionality. However, it can be configured in transmitter mode only or receiver mode only. This application note briefly explains the UART protocol and how to configure the module for operation.

PROTOCOL:



VR02135B

1 Start bit corresponding to the beginning of the frame.

7 or 8 bits of data corresponding to the data to receive or transmit.

1 Stop bit corresponding to the end of the frame.

3 BAUD RATE

The Baud Rate is used to configure the transfer speed. For a hardware cell, the transfer speed is an input clock ratio formed by a pre-defined divider.

For the software module, we will use the timer to generate the different speeds, using both the internal clock and the timer clock divider. A pulse period can be calculated using the following formula:

Example:

If a 16 MHz crystal is used, the corresponding period is 62.5 ns but, as a clock divider is implemented within the ST7, the real period is 125 ns.

When using the timer, a prescaler is used and can be configured in the timer Control Register 2 (bits C0 and C1).

To sum up: CLOCK in = 16 MHz

1/16 MHz = 62.5 ns

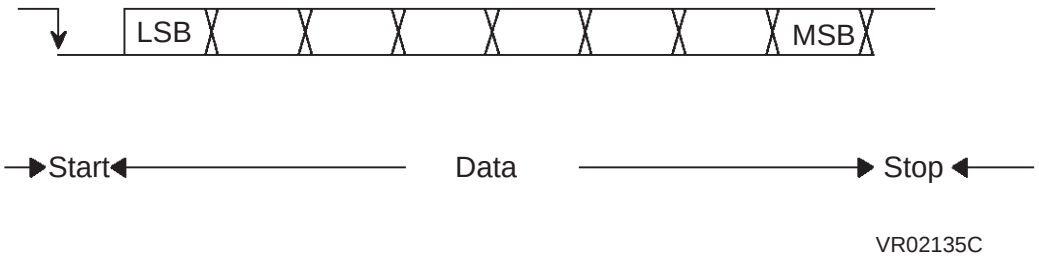
62.5 ns x 2 = 125 ns ST7 Core Internal clock (f_{CPU})

125 ns x 4 = 0.5 µs Timer prescaler divider

The pulse period on the line will be **0.5 µs** long.

So, if a 1200 baud rate is required for the application, the input value you will have to configure in the program will be:

1/1200 = 833 µs = 1666 x 0.5 µs or 682 hex.



In the software, two 8-bit registers (SH and SL) contain the speed value:

SH = 06 h

SL = 82 h

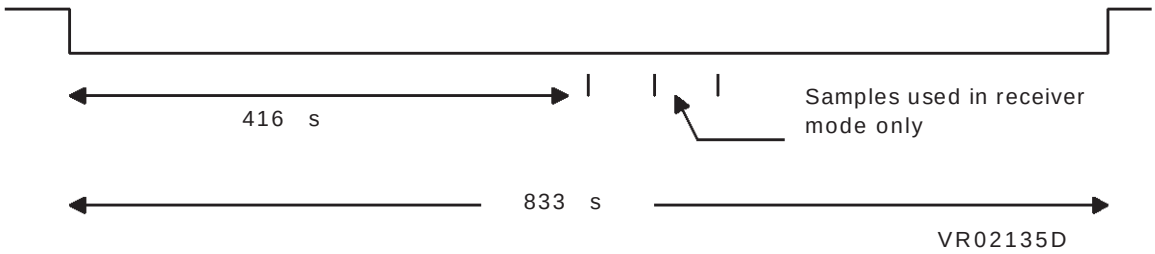
Again, in receiver mode, a half period is required to sample each data, so the next parameter you'll have to provide to the UART program is the Half Speed, coded over two 8-bit registers (HSH and HSL):

833 / 2 = 416 µs = 833 x 0.5 µs or 341 hex

and

HSH=03h

HSL=41h



STANDARD TRANSFER SPEED SUPPORTED:

6 standard speeds are supported by the UART program.

A simple calculation based on the previous one allows you to configure the program for each required speed. You can also check the constant.asm file which contains the hex value for each of the following baud rates.

1200 baud

2400 baud

4800 baud

9600 baud

19200 baud

28800 baud

Refer to the last chapter of this application note for more details on supported speeds.

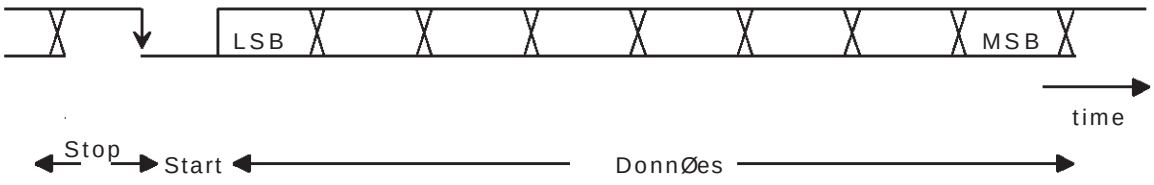
4 TRANSMITTER MODE

According to the previous calculation, each bit is transmitted on the line with the same duration. But, care must be taken about the protocol used here.

Instead of starting a communication with a start bit, when the UART function is called within the main program, we first generate a stop condition and, after the STOP bit length, we generate the start bit.

This way of proceeding insures that a correct start condition will be generated on the line whatever the previous pin level.

So, the frame looks like this, **with LSBs sent first**:



VR02135E

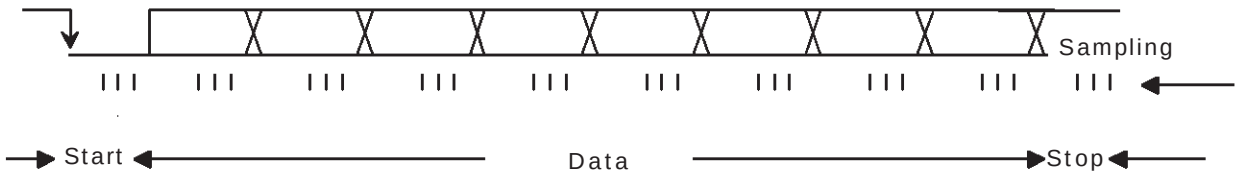
5 RECEIVER MODE

Receiver mode is more complicated than transmitter mode because of potential errors or noise created by the connections and wiring. So, a simple sampling function cannot ensure a proper bit decision.

A 2 to 1 majority voting system is used to determine the bit level on the line. Three samples are taken on the line at mid period and a decision is taken based on the sampling process.

A falling edge followed by a low level on the receive pin will start the process while a one level after the 9 first bits stands for a stop condition.

So, if a glitch occurs on the line, generating a falling edge detection, it won't be considered as a start bit unless it is followed by a low level for a bit duration.



VR02135G

PRINCIPLE:

Once the UART routine is called and when it is configured in receiver mode only, the input pin waits for a falling edge. Once an edge is detected, an interrupt is generated and the half duration of the bit (HSL and HSH registers) is loaded in the timer output compare registers (OCLR and OCHR). When the output compare interrupt occurs, the sampling process is done and a low level must be detected on the line (it's a start bit). If not, a flag is set meaning that the frame is incorrect and the software stops. But if yes, a full bit duration is added to the output compare value of the timer so that the same sampling process will be executed in the middle of the first data bit.

After all eight data bits have been received, the UART program tests if the bit received is at a high level for the stop bit. Once again, if not, the same flag (frame error) is used to warn that the frame is not good.

A flag indicates when the transmission is over and the data received is located in a RAM register.

EXAMPLES:

1. For a data bit: if two samples out of the three are ”0” then the data is a 0 but, as one of the sample was a 1, the Noise Frame flag in the status register is set.

2. For the Start bit: two samples out of the three taken must be zeros. If not, the frame error flag is set in the status register and the frame is incorrect.

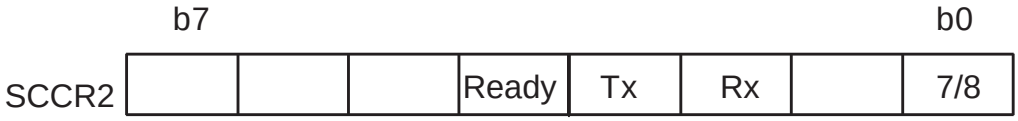
3. For the Stop bit: two samples out of the three taken must be ones. If not, the frame error flag is set in the status register and the frame is incorrect.

6 FUNCTIONAL DESCRIPTION

The UART is controlled through 6 registers. We’ve already seen 4 of them, SH, SL, HSL and HSH for the transmission speed and the two others are SCSR and SCCR2, respectively for Status Register and for Control Register.

CONTROL REGISTER SCCR2:

The register is described below:



VR02135H

bit 0 => 7/8 If this bit is set, the transmission or the reception is 7 bits long.

bit 1 => Sev If this bit is set, another byte will be transmitted on the line.

Care must be taken when you decide to set this bit, refer to the timing explanation later in this document.

bit 2 => Rx If this bit is set, the receiver function is enabled.

bit 3 => Tx If this bit is set, the transmitter function is enabled.

bit 4 => Ready This bit is read only. When set, it indicates that the UART can accept another byte. Refer to the timing explanation further in this document.

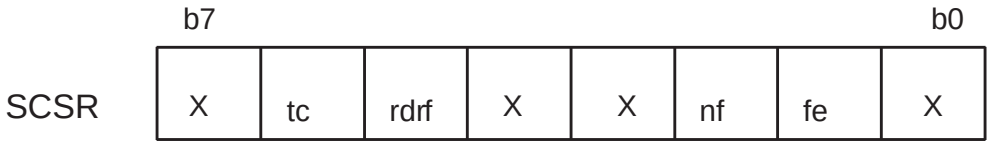
bit 5 to 7 Unused.

Notes:

1) The reset value of this register is 00h.

2) When Te = 1 and Re=1, full duplex mode is enabled and the transfer throughput cannot exceed 9600 baud.

STATUS REGISTER



VR02135I

bit 0 => X Unused

bit 1 => fe Frame Error: When set, this bit indicates that the reception is incorrect (start bit or stop bit invalid).

bit 2 => nf Noise Frame: When set, this bit indicates that some noise interfered on the line but that the result is still coherent.

bit 3 and 4 => X Unused.

bit 5 => rdrf Receive Data Register Full: When set, this flag indicates that all the bits of the data register are received.

bit 6 => tc Transmit Complete: When set, this flag indicates that a whole frame has been sent, including the stop bit and the start bit.

bit 7 => X Unused

The fe and nf flags are used to control the result of the communication.

The rdrf flag is used to control receiver mode (when set, the data can be read in the received data register).

The tc flag is used to control transmitter mode (when set, the data has been sent on the line).

DATA REGISTERS:

The UART has three data registers:

SCDAT_EMIT for the data that have been transmitted on the Tx line.

SCDAT_REC for the data that have been received on the Rx line.

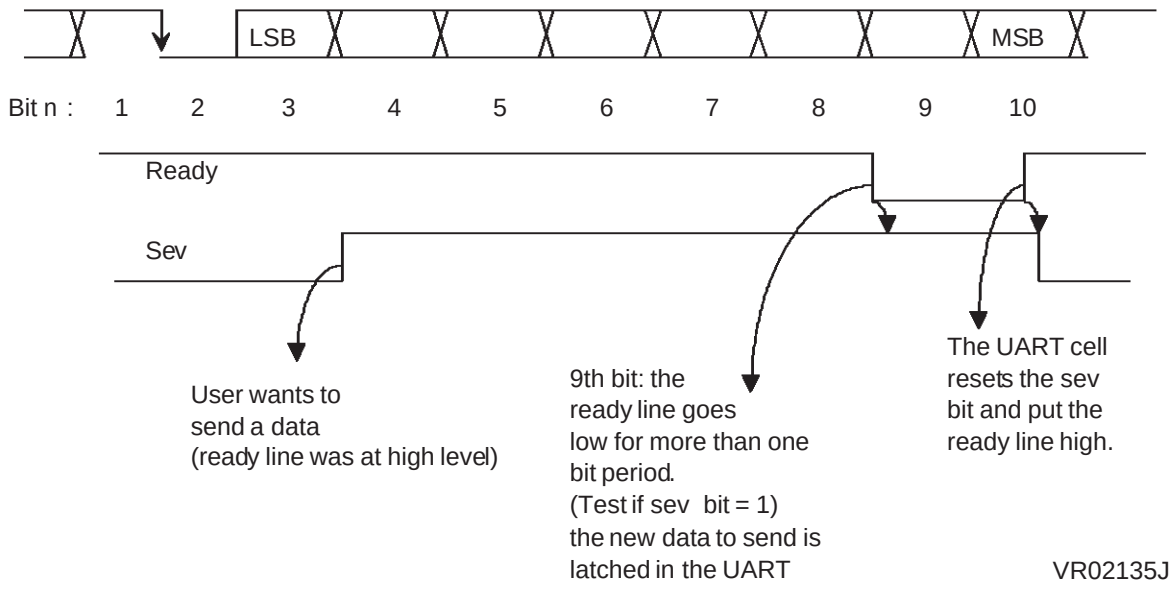
NEWDATA for the next byte to be transmitted on the Tx line.

PARTICULAR TIMINGS FOR SEV AND READY FLAGS:

In transmitter mode, the UART might have to send more than one byte.

The two flags Ready and SEV (for Several) in the SCCR2 register are used to handle a handshake protocol.

When the UART is sending bits on the Tx line, a shift register is used to serialise the data stream. However, the user must indicate to the UART if there's another byte to send afterwards. The timings are:



Between bit 1 and bit 9, the UART can accept another byte (READY = 1) at any time.

If the user wants to send a byte, he has to load the data in the NEWDATA register and set the SEV bit.

When bit 9 is being transferred on the line, the UART indicates that it cannot accept another byte (READY = 0) and tests the SEV flag. If set, then the content of NEWDATA will be loaded in the SCDAT_EMIT register for the next transfer. The UART resets the SEV bit and puts READY to high meaning that the user can send another byte if necessary.

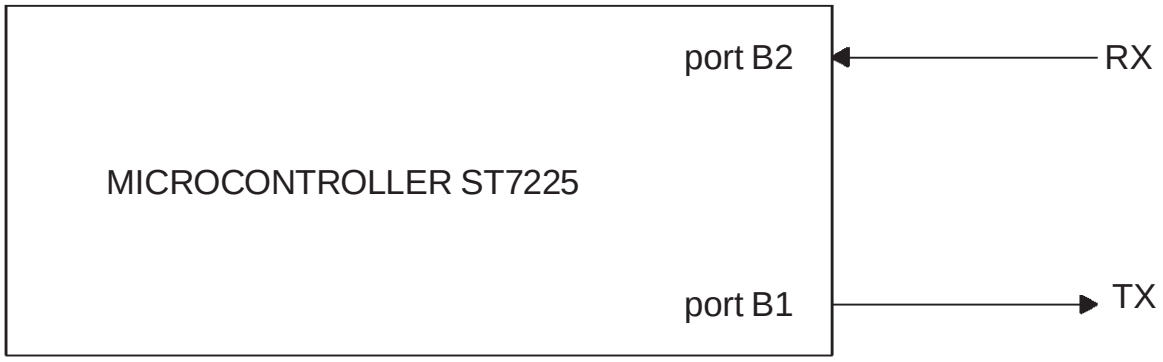
If reset, a stop is generated and further transmission can be achieved only by calling the UART routine again.

The 10th bit will be followed by a STOP and a START bit and after this, the byte will be transferred on the Tx line.

7 HARDWARE DESCRIPTION

The routine runs on the ST72251 microcontroller and uses one 16-bit timer. Three timer functions are used: the two output compares and the input capture 1 of timer A.

- 2 lines are used for Tx and Rx. Tx is the Output Compare pin 1 of timer A (PB1) and Rx is the Input Capture pin 2 of timer A (PB2).



VR02135K

Note: The routine uses all resources of Timer A. For correct handling, it's better not to use timer A for other tasks if you can use timer B instead.

8 SOFTWARE DESCRIPTION

To use the UART program, follow this procedure:

- Configure the speed registers in the constant.asm file according to the chosen mode.
- Configure the SCCR2 register to indicate which mode is enabled (transmitter only, receiver only, full duplex).
- Configure the PB1 pin in order to put PB1 at high level and reset the OLV1 bit of the TACR1 register (see main asm program for precise details).
- Load the data to transmit in the NEWDATA register, if transmitter mode is chosen.
- and finally, make the

  CALL UART

This is all you have to do in order to use the UART. The following chapter explains how the UART works but if you wish, you can skip this chapter and go directly to the example given at the end of this document.

TRANSMITTER MODE:

The first step in transmitter mode, is to initialise the PB1 pin to a high level.

However, a simple bit set (bset instruction) doesn't ensure a proper level on the pin.

As we are using the alternate function of the PB1 pin, the bset instruction will turn the I/O mode on but will not change the pin level properly.

To initialise it correctly, an output compare with OLVL1 = 1 must first be done.

By polling the OCF1 flag of Timer A status register (TASR) you can verify that the pin is correctly set and the rest of the program can continue.

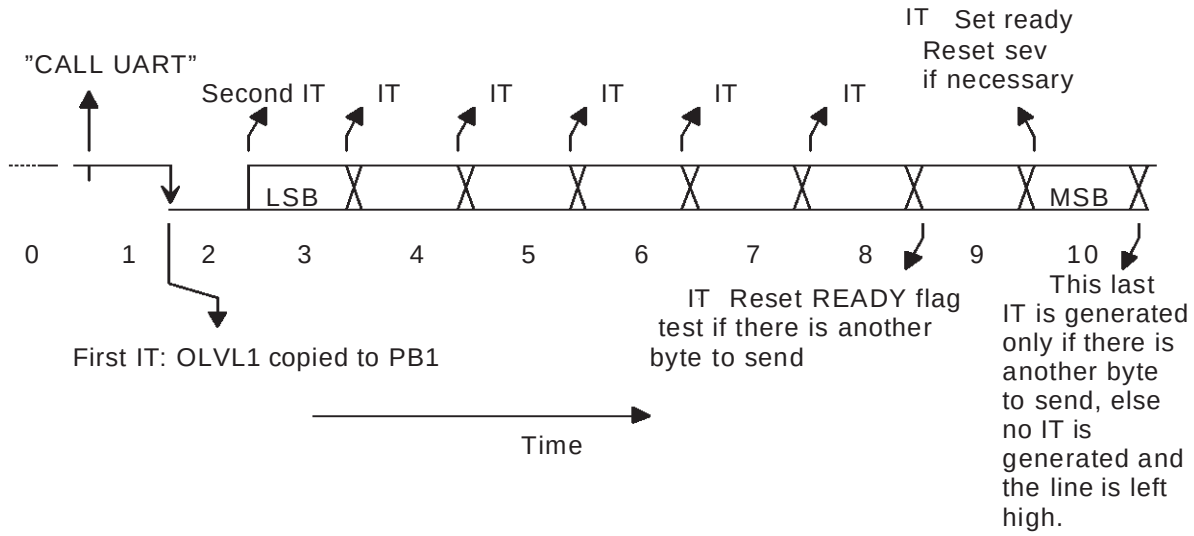
The second step of transmitter mode is to set the TE flag in the SCCR2 register and then to program the UART.

When the "CALL UART" has been done, the routine reads the alternate counter value and adds the bit period depending on the transmission speed and loads the calculated value in the output compare 1 registers. OLVL1 is reset in order to generate the start bit (remember that the sequence is STOP, START, DATA...). The main program can resume as the UART routine is, for now, over.

When the counter reaches the output compare value, a low level is applied on the line (OLVL1=0), the interrupt routine loads a new value (old output compare value + bit period) in the output compare registers and, the OLVL1 bit is set or reset depending on the first data bit (data bits are shifted serially in the carry).

When a frame has been transmitted, the TC flag is set and you can send another frame.

The following timing represents a full transmission.

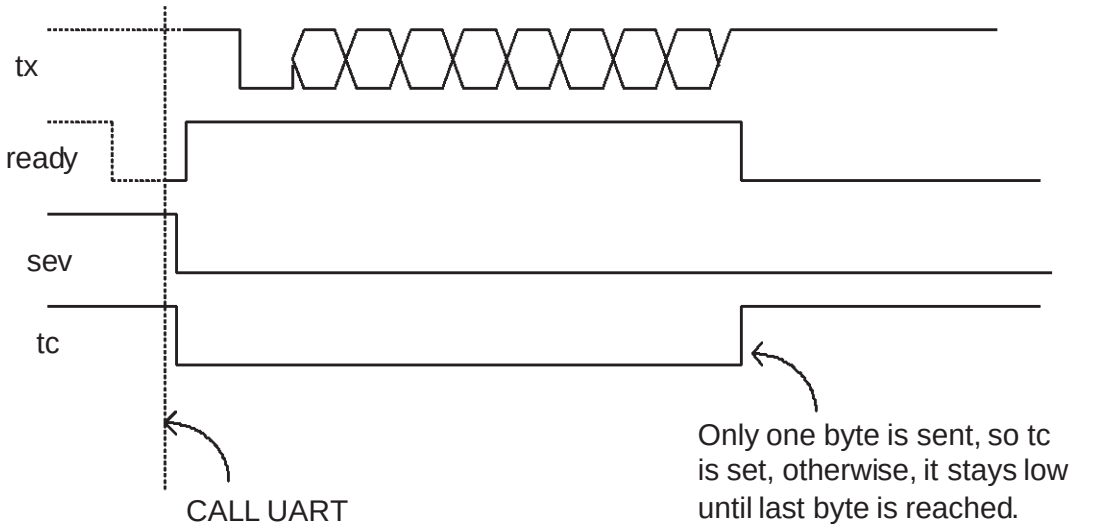


0: UART initialization,
PB1 pin = 1 with a first output compare.
OLV1=0 for the next output compare (= start bit).

1: STOP bit,

2: START bit. During the interrupt routine, the output compare registers are loaded with the next output compare value and OLV1 is configured according to the LSB value.

VR02135L



VR02135M

Notes:

– Between two interrupts, the main program is free for other tasks. The UART program works with interrupts only.

– All interrupts are OUTPUT COMPARE 1 interrupts (OCF1 flag).

– Interrupts coming from other devices will not disturb operations provided they are shorter than a bit duration minus the UART IT length, i.e provided that the interrupt length fits between two interrupts from the UART.

RECEIVER RUNNING MODE:

Once the UART has been configured in RECEIVER mode, the software waits for a falling edge on the PB2 pin which represents the start bit. To detect the falling edge, the input capture interrupt is turned on and the first interrupt routine is executed.

As soon as the falling edge is detected, the input capture routine disables the input capture interrupt on the pin (a high to low transition between two data bits must not be seen as a start bit!) and the half bit duration is added to the alternate counter value to load the output compare 2 registers.

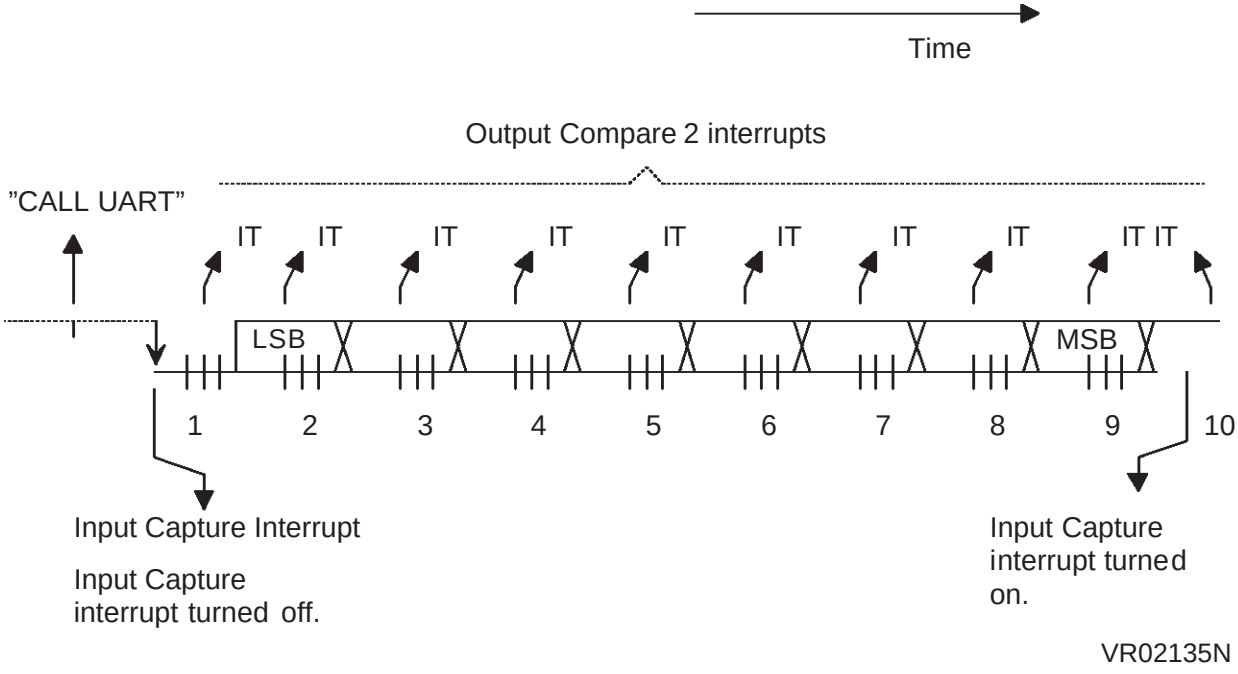
Once the first output compare 2 interrupt occurs, we are in the middle of the start bit, and the sampling process starts. If everything went right (2 or 3 samples at 0), the routine adds a full bit duration to the old output compare value and loads it in the output compare registers.

The next output compare 2 interrupt will occur in the middle of the first data bit, the sampling process will tell which level was received (0 or 1).

The process continues until the stop bit detection. The last interrupt triggers the input capture interrupt on the PB2 pin in order to detect the next start bit.

-At the end of the reception, the RDRF flag is set.

The following timing represents a full reception:



VR02135N

Notes:

– Between two interrupts, the main program is free for other tasks. The UART works with interrupts only.

– Interrupts coming from other peripherals (or external pins) should be disabled so as not to disturb the sampling process.

Note concerning full duplex mode:

Although receiver and transmitter mode can run at 19200 baud, full duplex mode is slower due to the interrupt routine length. In fact, each routine has its own execution time and if another interrupt occurs at this moment it cannot be served.

The following table gives the precise length of each routine.

From	To	Receiver	Full duplex	Transmitter	Worst Case
Call UART	RET instruction	68 CPU cycles	236 CPU cycles	174 CPU cycles	Word = 8 bits
ICAP2 interrupt	IRET instruction	78 CPU cycles	X	X	X
OUTPUT COMPARE 2 interrupt	IRET instruction	177 CPU cycles	X	X	Data processing, no special case (start or stop bit)
OUTPUT COMPARE 1 interrupt	IRET instruction	X	X	94 CPU cycles	Normal data processing

For example, the following is an example using the information given in the table:

If you are in receiver mode, the worst case you can encounter is when you are in the middle of the frame. Between the time you get the output compare 2 interrupt and when you exit the subroutine there are 177 CPU cycles.

So, this means that, if a CPU cycle is 125 ns long, the fastest transmission you can accept is:

177 x 125 ns = 22,125 µs between two data bits.

Consequently:

28800 baud <=> 34,72 µs

19200 baud <=> 52.08 µs

are both possible transmission rates.

In full duplex mode, two consecutive worst cases must be taken into account.

So the worst case that could occur is

177 + 94 = 271 CPU cycles = 33,875 µs

The closest standard speed to this worst case is:

19200 baud <=> 52.08 µs

9600 baud <=> 104,17 µs

which are both possible transmission rates.

The total number of bytes for the routine is: 393 bytes of ROM + 37 more bytes for the PB1 pin initialisation in the main program and 18 bytes in the RAM memory space for internal variables.

9 COMMUNICATION WITH A PC

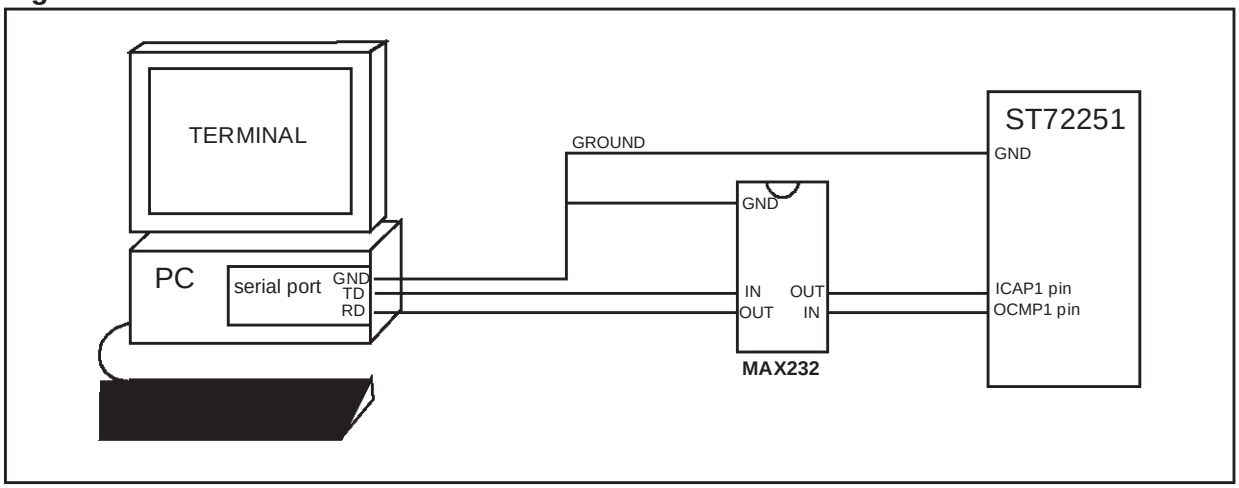
The ST7 emulated UART cannot be directly connected to a PC, as it uses the RS232 protocol.

The electrical and protocol characteristics of RS232 are different from those provided by the Timer I/O pins. In RS232 communication, the high level is typically +7V and the low level is typically -7V, while the Timer peripheral works at TTL levels (0, +5V).

Furthermore, the polarities are different. A '1' bit coming from the Timer corresponds to a '0' bit in RS232, and a '0' bit to a '1' bit. **This is true for all bits including the start and stop bits**.

So it is necessary to implement a conversion between the PC and the ST7. In the application, a MAX232 is used for this purpose. An overview schematic is presented below (Figure 1.).

Figure 1. Overview Schematic



Be sure that the three main devices (PC, ST7, MAX232) have the same electrical reference (GND).

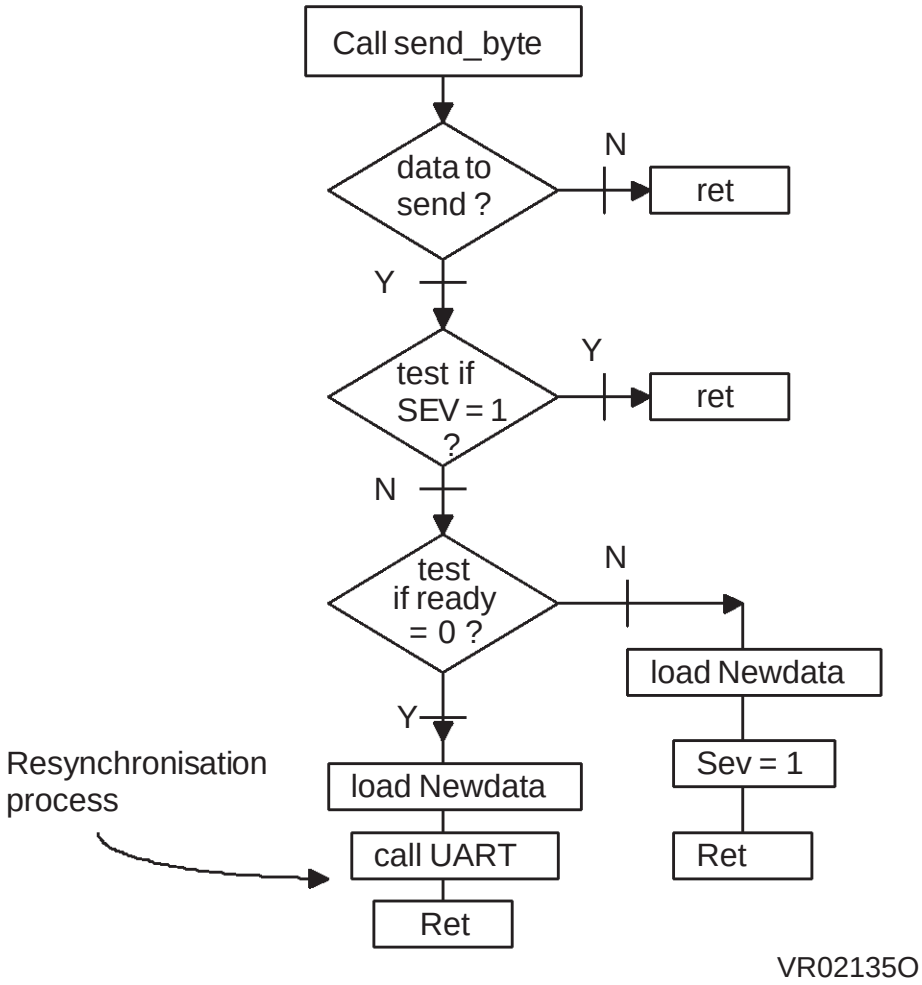
The Receive Data pin (RD) of the serial port of the PC must correspond to the OCMP1 pin of the ST7, and the Transmit Data pin (TD) to the ICAP1 pin.

Note for the main program example:

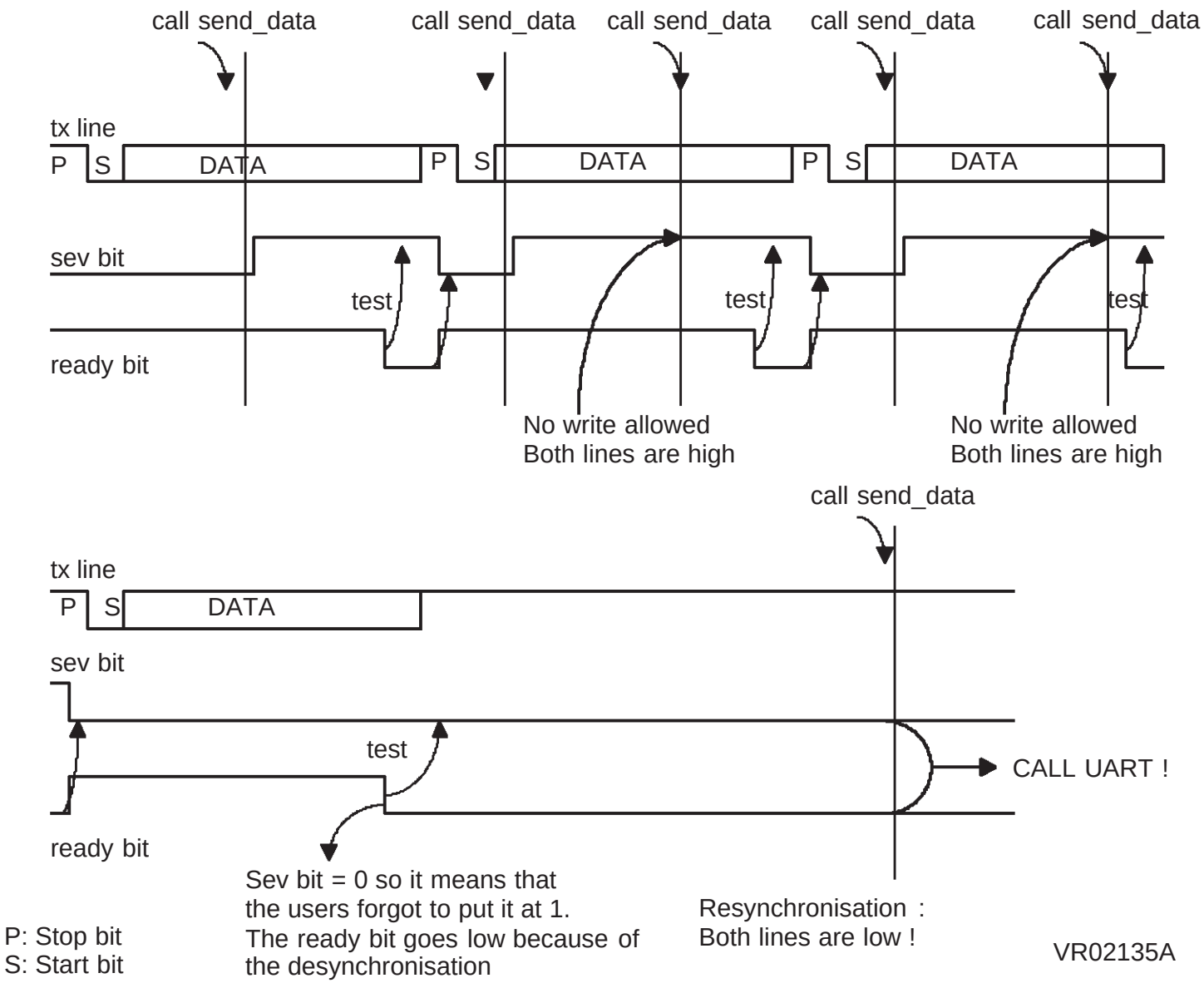
When using the UART in full duplex mode, there's no time left in the main program for other tasks. You must disable all other interrupt sources in order to make a correct full duplex transmission.

In transmitter mode, the following flowchart is given as an example for handling the different flags (tc, ready, sev).

An upper level subroutine called ”send_data” has been written. Use it in your main program if you want to send a data with the UART program.



To conclude, a transmission example is shown illustrating all cases with different states of the sev and ready bits.


call send_data
call send_data
call send_data
call send_data
call send_data
tx line
P
S
DATA
P
S
DATA
P
S
DATA
sev bit
test
test
test
ready bit
No write allowed
Both lines are high
No write allowed
Both lines are high
call send_data
tx line
P
S
DATA
sev bit
test
CALL UART !
ready bit
Sev bit = 0 so it means that
the users forgot to put it at 1.
The ready bit goes low because of
the desynchronisation
Resynchronisation :
Both lines are low !
P: Stop bit
S: Start bit
VR02135A

10 SOFTWARE

The assembly code given below is guidance only. The file cannot be used alone. The complete software can be found in the ST.COM website.
st7/

```
;******************************************************************************
; TITLE:         UART.ASM
; AUTHOR:        Microcontroller Division Applications Team
; DESCRIPTION:     Main program
;******************************************************************************

    TITLE  “UART.ASM”

;******************************************************************************
;                                          |
;    THIS PROGRAM PERFORMS a software U A R T IN TRANSMITTER OR RECEIVER modes    |
;    (<=> HALF DUPLEX UART)and also a FULL DUPLEX mode for fast data exchange  |
;                                          |
;******************************************************************************

; Start bit’s falling edge detected with the timer’s input capture function.
; |
; |
; |   Start <------------------Data ---------------------> Stop
; |    |                           |
; |----|   |-----|-----|-----|-----|-----|-----|-----|-----|-----|---------
; |---->|   | b0  b1  b2  b3  b4  b5  b6  b7 |
;    |_____|-----|-----|-----|-----|-----|-----|-----|-----|
;
; Samples |||  |||  |||  |||  |||  |||  |||  |||  |||  |||
;   |
; (only in receiver mode)


; The UART is controlled by two registers (SCCR2 and SCSR) and has three
; data registers (SCDAT_REC, SCDAT_EMIT and NEWDATA).
; The two control registers are described below while SCDAT_REC is the
; data register for incoming bytes, SCDAT_EMIT is the data register for outgoing
; byte and NEWDATA is the data register which holds the next value to transfert.
```

```
; SCCR2: Control register
;
; ------------------------
; |x|x|x|Rd|tx|rx|sev|7/8|
; ------------------------
;  ----  | | |  |  |
;   |    | | |  |  |
; Unused | | |  | Frame length: 1= 7 data bits in byte, 0= 8 data bits.
;      | | | Several bytes: If you want to send another byte after the
;      | | | one which is being transfered.
;      | | receive function turned on if set.
;      | transmit function turned on if set.
;     Ready flag: When set, the uart can accept another byte

; SCSR: Status register
;
; -----------------------------
; |tdre|tc|rdrf|x|x|nf|fe|Start|
; -----------------------------
;   | |  |    | |  |-> Start_bit : When set, the UART must generate a start bit
;   | |  |    | Frame error. When set, the frame is not valid
;   | |  |    Noise Frame. The flag is set when 1 of the three samples
;   | |  |    taken in the frame is different from the two others.
;   | | Receive data register full: When the byte is received the flag is set.
;   | Transmit completed: End of frame (with the stop bit)
;   transmit data register empty: Data register empty.
;
;-------------------------------------------------------------------------

    #include “st72251.inc” ; Insert register definition of the ST72251
    #include “constant.inc”

;*************************************************************************
*
;              USER PROGRAM
;*************************************************************************
    WORDS

    segment ‘rom’
;-------------------------------------------------------------------------
;              USER SOFTWARE
;-------------------------------------------------------------------------
.main
    ld   A,#$08      ; negative edge only
    ld   MISCR,A     ; normal mode
```

```
    ld   X,#$80
.raz  clr   (X)       ; Clear 62 bytes in RAM
    inc   X         ; for full duplex mode
    cp   X,#$FF       ; (to copy received data into RAM).
    jrne  raz

; Disable the timer interrupts
    ld   A,TASR     ; clear status flags
    clr   TAOC1LR
    clr   TAOC2LR
    rim             ; IT output compare will be enabled when oclr is
    clr   TAOC1HR     ; written.
    clr   TAOC2HR     ; stop timer output compare 1 and 2

; Init PB1 pin
    ld   A,#$80     ; put the Output compare pin 1 ON
    ld   TACR2,A    ; clock ratio = 4 <=> one bit = 1µs
    ld   A,#$01
    ld   TACR1,A    ; OLVL1=1 and no output compare interrupt
    ld   A,TAACHR   ; read alternate counter (high byte)
    ld   sochr,A    ; save value of high register
    ld   A,TAACLR   ; read alternate counter register (low byte)
    add   A,#80     ; add 80 cycles to the current value
    ld   soclr,A    ; to generate a first output compare
    ld   A,sochr    ; in order to set the alternate level to ONE.
    adc   A,#00
    ld   sochr,A    ; further compare inhibited
    ld   TAOC1HR,A
    ld   A,TASR     ; clear flag of timer1
    ld   A,soclr
    ld   TAOC1LR,A   ; start output compare
.pb1_init
    btjf  TASR,#6,pb1_init  ; wait for the output compare flag
    ld   TAOC1HR,A          ; further compare inhibited
                            ; The compare1 function will be restored
                            ; if transmission mode is selected.
; UART Initialisation

loopa  clr   Y
    clr   X
    ld   A,#$21
    ld   NEWDATA,A    ;first character to send (“!”)
    bset  SCCR2,#re   ; Receive mode selected
    bset  SCCR2,#te   ; Transmit mode selected
    call  uart
```

```
; ************************************************************
; If you want to test the receiver function of the uart program, remove
; the comma to jump to the read loop; (re bit=1 and te=0)

;   jp   loop_read
; ************************************************************

; ************************************************************
; If you want to test the full duplex function of uart program, remove
; the comma to jump to the duplex loop; (re bit=1 and te=1)

    jp   loop_duplex
; ************************************************************************


;************************************************************
; Sending of several independent characters to test the transmission
; function of uart program; (re bit=0 and te=1)
;************************************************************

loop5  btjf  SCCR2,#ready,loop5    ;wait for the ready flag
    bset  SCCR2,#1            ; once the flag is set, set flag for another byte
    ld   A,#'j'               ; and load register
    ld   NEWDATA,A
loop7  btjt  SCCR2,#ready,loop7    ; wait for the ready flag to go down
loop6  btjf  SCCR2,#ready,loop6    ; wait for the ready flag to go up
    bset  SCCR2,#1         ; once the flag is set, set flag for another byte
    ld   A,#'-'           ; and load register
    ld   NEWDATA,A
loop11 btjt  SCCR2,#ready,loop11   ; wait for the ready flag to go down
loop9  btjf  SCCR2,#ready,loop9    ; wait for the ready flag to go up
    bset  SCCR2,#1         ; once the flag is set, set flag for another byte
    ld   A,#'s'           ; and load register
    ld   NEWDATA,A
loop12 btjt  SCCR2,#ready,loop12   ; wait for the ready flag to go down
loop10 btjf  SCCR2,#ready,loop10   ; wait for the ready flag to go up
    bset  SCCR2,#1         ; once the flag is set, set flag for another byte
    ld   A,#$32          ; and load register
    ld   NEWDATA,A
loop8  btjt  SCCR2,#ready,loop8    ; wait for the ready flag to go down
    bres  SCCR2,#1             ; no other byte
loop2  btjf  SCSR,#tc,loop2        ; Wait for transmition complete flag

loop   jra  loop
```

```
;*********************************************************************
; If you want to test the transmission function of uart program and
; send more than one byte with the uart, you can still
; use this example of main program. An entire name is sent (“applications”).
;*********************************************************************
example
    ld   X,#11

begin  ld   A,(first_name,X)         ; Load first character to send
    ld   NEWDATA,A
    call  uart
lp   btjf  SCSR,#tc,lp
    bres  SCSR,#tc                   ; Clear tc flag
    DEC   X
    TNZ   X
    jreq  over
    jra   begin

over  ld   A,(first_name,X)          ; Sending of the last character
    ld   NEWDATA,A
    call  uart
lp2  btjf  SCSR,#tc,lp2
    bres  SCSR,#tc                   ; Clear tc flag

;*********************************************************************

;*********************************************************************
.loop_duplex
    btjt  SCCR2,#ready,load_data ; Wait for the ready flag
    btjt  SCCR2,#1,res_flag      ; Ready=0 and Sev=1 -> reset Sev flag
polling btjt  SCSR,#rdrf,read_data     ; Wait for the end of reception flag
    btjt  SCSR,#tc,end_transmit   ; Wait for the transmition complete flag
    jra  loop_duplex

res_flag
    bres  SCCR2,#1               ; Instructions executed only when Ready=0
    jra   polling                ; and Sev=1


load_data
    btjt  SCCR2,#1,exit         ; test if sev = 1. If so, wait for ready to go down
    bset  SCCR2,#1              ;else, set flag for another byte
    ld   A,TAACLR                ; and load register with counter’s low byte
    ld   NEWDATA,A               ; for a random value
    jrpl  end_transmit
```

```
exit  jra  polling

.end_transmit
    bres  SCSR,#tc                  ;clear flag
    jra  polling

.read_data
    inc  X
    ld   A,SCDAT_REC                ;store data in RAM
    ld   (SCDAT_REC,x),A
    bres  SCSR,#rdrf                ;clear frame received flag
    jra   polling


;*************************************************************


;*************************************************************
; The read loop shows how to receive three bytes from a master device.
;*************************************************************
loop_read
read1  btjf  SCSR,#rdrf,read1 ; Wait for the end of reception flag
    ld   A,SCDAT_REC
    ld   {SCDAT_REC+1},A
    clr   SCSR                  ;Read data and clear flag
read2  btjf  SCSR,#rdrf,read2  ; Wait for the end of reception flag
    ld   A,SCDAT_REC
    ld   {SCDAT_REC+2},A
    clr   SCSR                  ;Read data and clear flag
read3  btjf  SCSR,#rdrf,read3  ; Wait for the end of reception flag
    ld   A,SCDAT_REC
    ld   {SCDAT_REC+3},A
    bset  SCCR2,#ready

.loopr jra  loopr
;*************************************************************************
;           U A R T  SUBROUTINE
;*************************************************************************
.uart  btjt  SCCR2,#seven,seven_bits ; Test the transmission's length
    ld   a,#{8+2}                   ; The word's length is eight  bits
    jra   c_status
.seven_bits
    ld   a,#{7+2}                   ;The word's length is seven bits
```

```
.c_status
    bres  SCCR2,#1                  ; Reset SEV bit for main program
    bset  SCCR2,#ready              ; set ready bit for main program
    ld    cpt_bit_emit,a
    ld    cpt_bit_rec,a
    ld    nb_bit_emit,a
    ld    nb_bit_rec,a
    ld    a,NEWDATA                  ; get the new value to transfer
    ld    SCDAT_EMIT,a               ; in the transmitter's data register
    clr   SCSR                       ; clear all flags
    btjt  SCCR2,#re,receiver         ; check if receiver was requested

.transmiter_only                     ; No, then Half duplex transmitter only

; No receiver function requested (= transmitter function)

; In transmitter mode, the real sequence is:
;
;    |STOP|START|DATA7|DATA6|...|DATA0|
;
; rather then
;
;    |START|DATA7|DATA6|...|DATA0|STOP|
;
; so the next two decrementations stand for the stop bit.

    dec   nb_bit_emit
    dec   cpt_bit_emit

; Timer Initialization
    bset  TACR1,#6              ; enable the output compare interrupts
    bres  TACR1,#0              ; reset olv1 for the start bit
    ld    a,TAACHR              ; Read the current free running counter value
    ld    sochr,a               ; save value of high register
    ld    a,TAACLR
    add   a,#SL                 ; Add the bit duration and save the value
    ld    soclr,a
    ld    a,sochr
    adc   a,#SH
    ld    sochr,a               ; further compare inhibited
    ld    TAOC1HR,a       ; Load added value to the output compare 1 registers
    ld    a,TASR                ; clear flag of timer1
    ld    a,soclr
    ld    TAOC1LR,a             ; and start the output compare function
    ret
```

```
  .receiver
    dec   nb_bit_rec
    ld    a,#$C0                ; Input capture and output compare interrupts
    ld    TACR1,a               ; enabled.
    bres  TACR2,#1              ;Input Capture edge sensitivity= falling edge
                                ; test if we are in full_duplex mode
    btjt  SCCR2,#te,transmiter_only
    ret

;-------------------------------------------------------------------------
;           TIMER A INTERRUPT ROUTINE
;-------------------------------------------------------------------------
; Check which mode generated the timer A interrupt:
;    Output Compare1 (transmission mode)
;    Output compare2 (reception mode)
;    Input Capture  (reception mode)

.tima_rt
    btjf  TACR1,#7,continue     ; Test if the input capt interrupt is enabled
    btjt  TASR,#4,start_reception; Flag=1 so test input capture 2 flag
.continue                       ; else ICFE flag is reseted or it's not the right
                                ; edge on ICAP2
    btjt  TASR,#6,trans         ; so test output compare 1 flag
                                ;else output compare 2 flag
    btjt  TASR,#3,temp          ; jump to the receiver function
    iret
.temp  jp   recept

; re = 0  so transmit mode is selected

.trans
    btjt  SCSR,#start_bit,restart   ; test if start flag is set
    btjt  SCSR,#tdre,arret          ;if tdre is set transmissioncompleted
                                    ; so stop the timer
    ld    a,cpt_bit_emit            ; test which bit is sending on the line

    cp    a,#1                      ; test if there is only one bit left in the
    jreq  last_bit                  ; data register

    bset  SCCR2,#ready              ; uart can accept another byte
.send  sra   SCDAT_EMIT             ; shift the bits of the data register into
    jrc   one                       ; the carry. LSB first
    bres  TACR1,#0                  ; change OLVL1 to 0 for the next compare
    jra   fin
```

```
.one   bset   TACR1,#0                  ; change OLVL1 to 1 for the next compare

; load the new timer value

; New output compare value = Old output compare value + bit length value

.fin   dec    cpt_bit_emit
.fin1  ld     a,soclr                   ; add the duration value into save register
       add    a,#SL
       ld     soclr,a
       ld     a,sochr
       adc    a,#SH
       ld     sochr,a                    ; further compare inhibited
       ld     TAOC1HR,a
       ld     a,TASR                     ; clear flag of timer1
       ld     a,soclr
       ld     TAOC1LR,a                  ; start output compare
       iret


.last_bit
       bres   SCCR2,#ready               ; uart may read the NEWDATA register
       btjt   SCCR2,#1,one_more          ; test if there's another byte to transfer
       bset   SCSR,#tdre     ; no, so the data transmission register has been sent.
       bset   TACR1,#0             ; set olv1 to keep line at high level (stop bit).
       jp     fin

.one_more
       bset   TACR1,#0                   ; OLVL1 = 1 for the stop bit
       bset   SCSR,#start_bit            ; one more byte, so set start bit
       jp     fin                         ; another byte must be transfered

.restart
       bres   TACR1,#0                    ; OLVL1 = 0 for the start bit
       ld     a,nb_bit_emit              ; get frame length
       ld     cpt_bit_emit,a             ; reset cpt_bit with frame length
       bres   SCSR,#start_bit            ; clear start bit
       ld     a,NEWDATA
       ld     SCDAT_EMIT,a               ; get new byte to send
       bset   SCCR2,#ready                ; set ready bit for main program
       bres   SCCR2,#1                    ; reset sev bit to inform the main program
                                           ; that the new data had been latched.
       jp     fin1                         ; and wait for one period.

.arret                                  ; Stop output compare
```

```
    ld   a,TASR
    clr   TAOC1LR
    clr   TAOC1HR
    bset  SCSR,#tc                    ; The frame has been sent.
    iret

;-------------------------------------------------------------------------
;      RECEIVE MODE:                                    |
;       y register is incremented when a high level is detected on RX line |
;       x register is incremented when a low level is detected on RX line  |
;-------------------------------------------------------------------------

; Input capture condition

.start_reception
    ld   a,TAIC2HR              ; Read the input capture value
    ld   sochr2,a               ; MSB first and store value
    ld   a,TAIC2LR
    ld   soclr2,a
    add   a,#HSL                ; add the half duration value into save register
    ld   soclr2,a
    ld   a,sochr2
    adc   a,#HSH
    ld   sochr2,a               ; further compare inhibited
    ld   TAOC2HR,a
    ld   a,TASR                 ; clear timer's flag
    bres  TACR1,#7              ; disable input capture interrupt
    ld   a,nb_bit_rec           ; get frame length
    inc   a
    ld   cpt_bit_rec,a
    ld   a,soclr2
    ld   TAOC2LR,a              ; start output compare
    iret                        ; and wait for output compare2 interrupt


; Test the stop bit condition on the line

.stop0
    clr   x
    btjt  PBDR,#rx,ST10         ; check 1st sample
    bset  SCSR,#nf
    jp   ST100
.ST10  inc   x
.ST100 btjt   PBDR,#rx,ST11     ; check 2nd sample
    bset  SCSR,#nf              ; noise detected
```

```
    jp   ST101
.ST11  inc   x
.ST101 btjt  PBDR,#rx,ST12          ; check 3rd sample
    bset  SCSR,#nf                   ; noise detected
    jp   ST113
.ST12  inc   x
.ST113 cp    x,#2
    jrc   f_error
    jp   t_received
f_error bset  SCSR,#fe               ; If a majority of samples are “0”
    jp   t_received                  ; then it’s an error.


.recept
    push  x                          ; save user’s registers
    push  y
    dec   cpt_bit_rec                ; Test if we are receiving the last bit
    ld    a,cpt_bit_rec
    jreq  stop0                      ; if so, then test only “1” samples
    cp    a,nb_bit_rec             ; else, test if we are receiving the start bit
    jreq  start0                      ; and if so , test only “0” samples
                            ; If none of the above, we are receiving data bits
;
;   process data sample
;
    clr   y
    clr   x
    btjt  PBDR,#rx,T10               ; check 1st sample
    inc   x                          ; sample = 0 so increment X
    jrnc  T20
.T10  inc   y                         ; sample = 1 so increment Y

.T20  btjt  PBDR,#rx,T30             ; check 2nd sample
    inc   x
    jrnc  T40
.T30  inc   y                        ; second sample = 1 so increment Y

.T40  btjt  PBDR,#rx,T50             ; check 3rd sample
    inc   x
    jrnc  T60
.T50  inc   y
.T60  cp    y,#3                     ; if y= 3 and x = 0  a sample 1
    jrne  T61                        ; has been detected three times
    jp    decal                      ; sample 1 into the SCDAT register
.T61  cp    x,#3                     ; if x= 3 and y= 0 a sample 0
```

```
    jreq  T63                       ; has been detected three times
    bset  SCSR,#nf
.T62  ld   nbr_b1,y
    ld   a,x
    cp   a,nbr_b1                   ; compare x and y to determine
    jrc   decal                     ; the value of the bit received
.T63  rcf                           ; Reset carry
    jp   decal11
.decal scf                          ; Set carry
.decal11
    rrc   SCDAT_REC                  ; Shift carry into the data register
    jp   fin0                       ; 1st bit received = LSB


; Test the start bit condition on the line

.start0
    clr   x
    btjf  PBDR,#rx,S0                ; check 1st sample
    bset  SCSR,#nf
    jp   S00
.S0   inc   x
.S00  btjf  PBDR,#rx,S01             ; check 2nd sample
    bset  SCSR,#nf                  ; noise detected
    jp   S002
.S01  inc   x
.S002 btjf  PBDR,#rx,S02             ; check 3rd sample
    bset  SCSR,#nf                  ; noise detected
    jp   S003
.S02  inc   x
.S003 cp   x,#2
    jrnc  fin0                      ; If the majority of the samples are "1"
    bset  SCSR,#fe                   ; it's an error.
    pop   y                          ; restore user's registers
    pop   x                          ; and quit UART's routine
    iret


; Load the new timer value
.fin0  ld   a,soclr2                ; add the duration value into save register
    add   a,#SL
    ld   soclr2,a
    ld   a,sochr2
    adc   a,#SH
    ld   sochr2,a                   ; further compare inhibited
```

```
    ld   TAOC2HR,a
    ld   a,TASR                    ; clear flag of timer1
    ld   a,soclr2
    ld   TAOC2LR,a                 ; start output compare
    pop  y                         ; restore user's registers
    pop  x
    iret

.t_received
    bset  SCSR,#rdrf               ; The frame has been sent.

; Stop output compare
    ld   a,TASR                     ; clear interrupt flags
    clr  TAIC2LR                    ; Clear input capture flag
    clr  TAOC2LR
    clr  TAOC2HR                    ; Disable output compare
    ld   a,#$C0                ; Input capture and output compare interrupts
    ld   TACR1,a                    ; enabled.
    bres  TACR2,#1             ; Input Capture edge sensitivity= falling edge
    pop  y                          ; restore user's registers
    pop  x
    iret

;    ******************************************
;    *                      *
;    * INTERRUPT SUB-ROUTINES LIBRARY SECTION *
;    *                      *
;    ******************************************

dummy  iret

ext1_rt iret

ext0_rt iret

sw_rt  iret

spi_rt  iret

timb_rt iret

i2c_rt  iret
```

```
;*******************************************************************************
;            INTERRUPT AND RESTART VECTORS                *
;*******************************************************************************

    segment 'vectit'

             DC.W  not used    ;FFE0-FFE1h location
             DC.W  not used    ;FFE2-FFE3h location
.i2c_it      DC.W  dummy_rt    ;FFE4-FFE5h location
             DC.W  not used    ;FFE6-FFE7h location
             DC.W  not used    ;FFE8-FFE9h location
             DC.W  not used    ;FFEA-FFEBh location
             DC.W  not used    ;FFEC-FFEDh location
.timb_it     DC.W  dummy_rt    ;FFEE-FFEFh location
             DC.W  not used    ;FFF0-FFF1h location
.tima_it     DC.W  tima_rt     ;FFF2-FFF3h location
.spi_rt      DC.W  dummy_rt    ;FFF4-FFF5h location
             DC.W  not used    ;FFF6-FFF7h location
.ext1_it     DC.W  dummy_rt    ;FFF8-FFF9h location
.ext0_it     DC.W  dummy_rt    ;FFFA-FFFBh location
.softit      DC.W  dummy_rt    ;FFFC-FFFDh location
.reset       DC.W  main        ;FFFE-FFFFh location

                        ; This last line refers to the first line.
                        ; It used by the compiler/linker to determine code zone
          END           ; Be aware that the END directive should not
                         ; stand on the left of the page like the label names.
```

THE PRESENT NOTE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH A NOTE AND/OR THE USE MADE BY CUSTOMERS OF THE INFORMATION CONTAINED HEREIN IN CONNEXION WITH THEIR PRODUCTS.