AN1264
APPLICATION NOTE

Serial Communication RS232 with ST52x420

Author: G. Grasso

INTRODUCTION

This application note presents a standard RS232 serial communication between ST52x420 and a PC or another microcontroller. The assembler code provided at the end of the document could be easily arranged to meet user specifications.

The software was optimized in order to allow the user to embed these RX and TX subroutines in his main program. Alternatively the software can be written by using the FUZZYSTUDIO™ 4.0 in a graphical environment instead of assembler instructions.

RS232 PROTOCOL

This asynchronous technique is the most used implementation of a communication channel between a PC and a low cost external device. The reason is due to the low numbers of copper wires and to the high EMI immunity also for long distance connections.

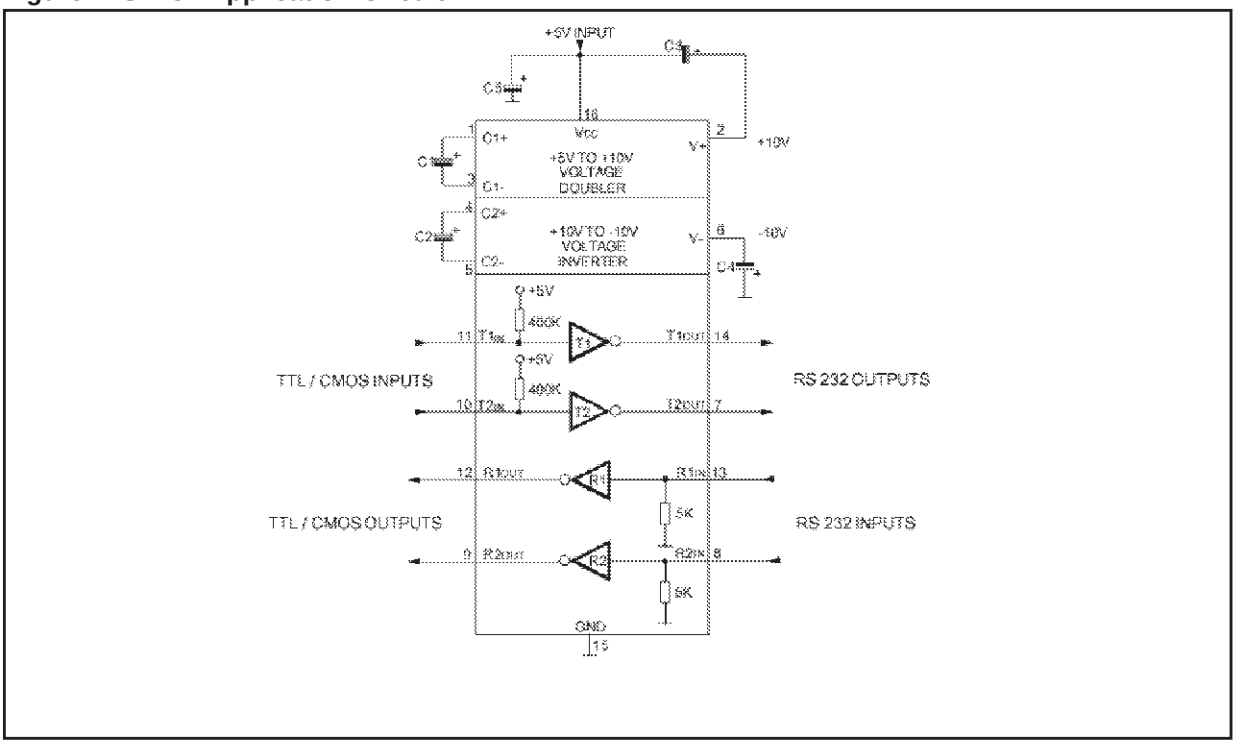
The complete standardized RS232 protocol uses some synchronisation/handshaking pins (DTA, DTR,...etc) that allow a more powerful communication but, in this case, the electrical connection requires a 25-pin connector.

Among several ways to implement a serial communication by using the RS232 standard, a very utilized simplification is the half-duplex communication on 3 wires. Half-duplex specification means that a communication is possible in both directions but not simultaneously. The baud rate can vary among different values but it should be the same for TX and RX. In this case, the electrical connection needs at least 3 wires that can be arranged into a 9 pins connector (standard DB9 connector of a Personal Computer).

From an electrical point of view, the PC serial port transmits a '1' as -3 to -25 Volts and a '0' as +3 to +25 Volts. Therefore the serial port can have a maximum swing of 50V compared to the parallel port which has a maximum swing of 5V. Due to this, cable losses and the inducted noise are not so problematic as for parallel cables. On the other hand, these electrical levels impose the use of level translators when a PC is interfaced to a digital device as a CMOS/TTL microcontroller.

Common level translators are the MAX232, ST232, etc which include a Charge Pump generating +10V and -10V from a single 5V supply. These I.C. also include two receivers and two transmitters in the same package.

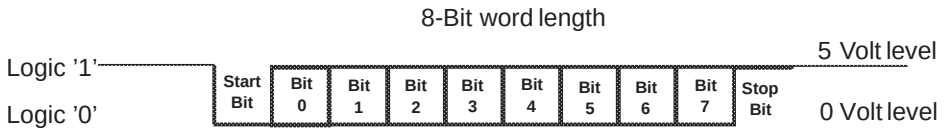
Figure 1. ST232 Application Circuit



Of course, if the serial communication links two microcontrollers or, in general, two digital CMOS/TTL devices it is not necessary to use a level translator.

From a logic point of view, the RS232 protocol consists in a sequence of bits that are arranged in a temporal 'frame' as shown in figure 2.

Figure 2. RS232 Data Frame



Since a clock signal is not sent with the data, each frame is synchronized using its Start Bit, and an internal clock on each side of the serial link.

A frame is always composed with two control bits (2 temporal slots) and 8 data bits (8 temporal slots) and optionally with other bits (parity, ninth bit). The time duration of each slot depends from the baud-rate chosen. For instance, 9600 baud (9600 bit/sec) means a time duration of 100 µs for each bit. This must be taken into account when the internal clock of the transmitter/reiceiver is initialized.

A transmission starts with a Start Bit which is '0'. Then, each bit is sent down the line, one at a time. The LSB (Least Significant Bit) is sent first. A Stop Bit '1' is then appended to the signal to end the transmission.

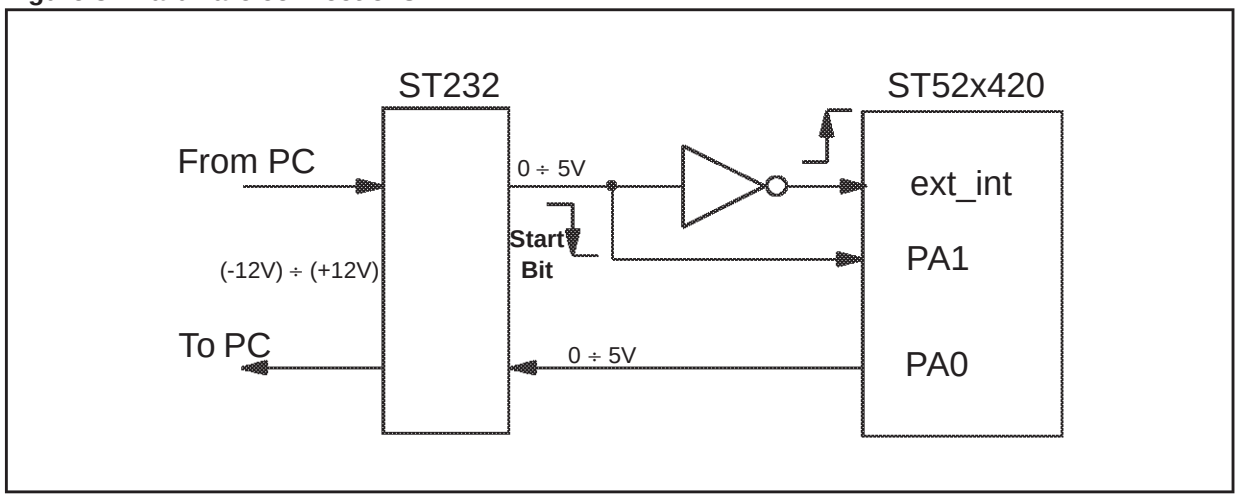
SOFTWARE DESCRIPTION

The assembler code here described is intended as a subprogram that can be added to the user main program in order to perform also the RS232 serial communication. The assembler code is shared into two different “CALL” subroutines that are invoked from the ‘main’.

The utilized resources in ST52x420 are only 3 pins and an internal Timer. The Timer can be reconfigured “on-fly” after each transmission/reception in order to employ it for other purposes.

Since the “start-bit” has to be detected from the ST52x420 (receiver) when the PC is transmitting, it is suggested to use the External Interrupt pin of the ST52x420 to recognize immediately the start of communication. The hardware configuration is shown in figure 3.

Figure 3. Hardware connections



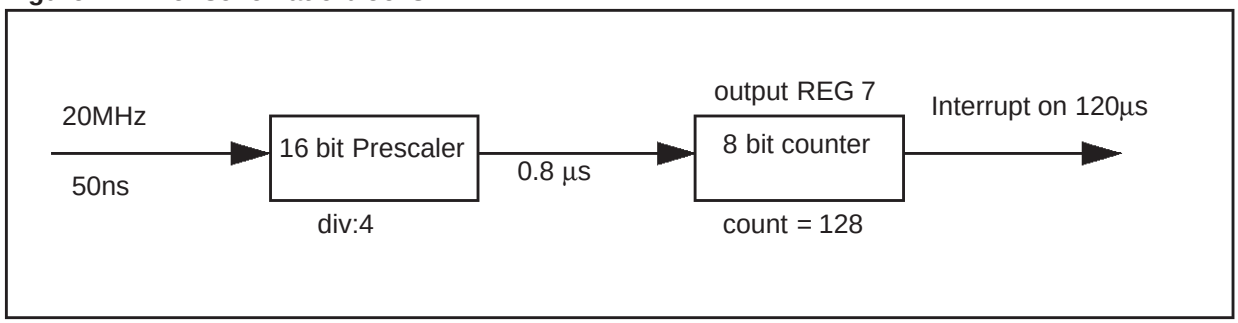
After the voltage translation, the signal coming from the PC can be read directly on the pin PA1 of ST52x420. A second input pin (ext_int) is used to detect the falling edge of the start bit of the data frame. In this case, an inverter is used because the pin ‘ext_int’ of ST52x420 is sensitive only to the rising edges of a signal. If the ext_int pin is used for other purposes, the falling edge of the start-bit can be detected in ‘polling mode’ directly on PA1. Likewise, the transmission from ST52x420 to PC is carried out on the PA0 pin.

Now, let us analyze the assembler code referred in the appendix. The code begins with the allocation of the interrupt vectors (0, 1, 2, ..) at predefined labels. The interrupt service routines are written at the end of the document, but these can be located anywhere. After these 5 lines the CPU and peripherals are initialized. To understand these configurations please refer to the registers description in the ST52x420 datasheet. Each configuration register is written with a “LDCR” instruction. This instruction will cause the Load of a ‘Conf_reg’ from a Ram register.

Following in the analysis, the “main” block is shown. The function of the main block is to produce a reply of the data received towards the transmitter. Infact, the main sequentially invokes two subroutines for the reception and transmission of the same data.

Before to analyze the subroutines it is useful to observe the Timer2 configuration. ST52x420 provides three different autoreload timers with a 16-bit prescaler for each one. These Timers can be configured as independent PWM generators or general purposes Timers. In the current application, Timer2 is used to count a fixed time period in order to clock each bit duration.

Figure 4. Timer schematic blocks



With a Baud-rate of 9600 the Timer has to count a duration of 100 µs for each bit (double for 4800 Baud, .. etc). To achieve this, a division ratio of '16' has been chosen for the prescaler. The "ldrc 0 4 , ldcr 11 0" instructions will write the binary '0000 0100' into the Conf_reg11. (see Table 9.7 in the St52x420 Data-Sheet).

By using a clock frequency for the device of 20 MHz (T=50ns) the output period of the Timer 2 Prescaler will be:

$$Tpsc = 50\ ns \times 16 = 0.8\ \mu s$$

Therefore a pulse wave with 0.8µs period will feed the 8-bit counter witch will count between 0.8 µs up to 204 µs (1 ÷ 255). In the subroutine, the Timer2 counter register (output register 7) will be loaded with '40' and '128' in order to count for '32 µs' and '102 µs' for the reason that will be clarified later.

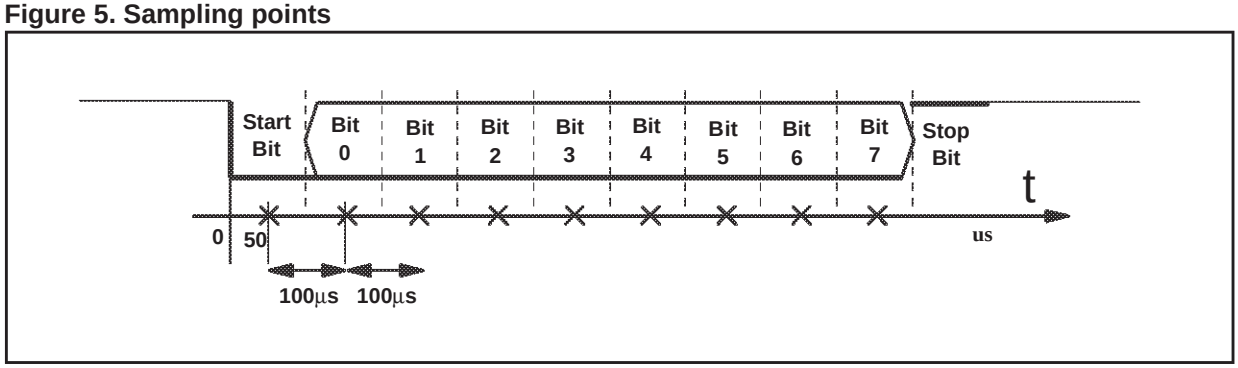
A second configuration register (Conf_Reg10) for the Timer 2 is the control register. By writing in this register the user can handle the start/stop, reset and interrupts signals of Timer2. The instructions "ldrc 0 64, ldcr 10 0" will set the peripheral in stop and reset mode and will enable the interrupt on the falling edge of the timerout (see Table 9.6 of DataSheet).

RECEPTION TASK

In the procedure implemented, the start bit of the coming data is detected by using the external interrupt. For this purpose, a 'flag' (RAM 3) is issued in the Ext_Int service routine. The RX subroutine checks if the flag is issued and starts with the sampling of the data on PA1.

Since the first coming bit (100 µs) is the start bit, this does not need to be sampled. Timer 2 is then set in order to count for 50 µs. This means that the Timer 2 will provide an interrupt at the center of the start bit.

Figure 5. Sampling points



The first data sampling will be performed after 100 µs from the center of the start bit. This will read the logic level of the Bit0 in its center. The same for the other bits until Bit7. Each time, the sampled bit is shifted to left in the data byte. Of course the sampled data byte contains as MSB the first bit sampled Bit0, therefore the received byte has to be mirrored.

TRANSMISSION TASK

Transmission is carried out on pin PA0. To set/reset this pin the user has to write the correct value on the PORT_A output register by using a “ldpr” instruction. The subroutine task starts by lowering the PA0 line for 100 µs; after that, each bit of the byte (to be transmitted) is isolated and sent to PA0 for 100 µs.

The stop bit is sent to PA0 as last bit for the same time duration, therefore the line is released at the high level.

CONCLUSION

This application note is intended to guide the user in the ST52x420 assembler code. The RX/TX algorithm could be enhanced and modified in order to meet the user’s requirements.

A second way to organize the algorithm is the use of the graphical environment FUZZYSTUDIO™ 4.0 which allows a fast program development, debugging, simulation also for novice designers.

REFERENCES

[1] ST52x420 - Datasheet, STMicroelectronics, 2000

[2] FUZZYSTUDIO™ 4.0 - User Manual, STMicroelectronics, 2000

[3] ST232 Datasheet, STMicroelectronics, 1999

APPENDIX: *ST52X420 ASSEMBLER CODE*

```
;************************************** ****************************
;
;   PURPOSE:        RS232 with ST52x420
;
;   AUTHOR: Fuzzy Logic Application Group
;

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

;************************************** ************************
;* transmission RS232 frame=9600baud(100us /bit) data=8bit,
;* 1 bit stop, 1 bit start
;* Transmit on PA0 (pin 25) and Receive on PA1 (pin24)
;* Receive the start bit on ”EXT_INT” (pin 5)
;************************************** **********************

irq 0 AD_INT
irq 1 TIM0
irq 2 TIM1
irq 3 TIM2
irq 4 EXT_INT

;******** Peripheral and Chip configurations ********

       ldrc 0 17   ; load RAM0(”tmp”) with ”0001 0001”
       ldcr 0 0    ; move on reg_conf0 the RAM0 content
                   ; interrupt mask ”nu(MSB)|nu|nu|tim2|tim1|tim0|AD|ext_int”
                   ; all masked unless ext_int e tim2

       ldrc 0 27   ; priority configuration 0001 1011
       ldcr 1 0    ; from the top: TIM2, TIM1, TIM0, ADC(lowest)

       ldrc 0 00001111b
       ldcr 2 0    ; watchdog configuration
                   ; 9375*50ns*500= 234ms count
       wdtslp      ; watchdog disabled

       ldrc 0 0
       ldcr 3 0    ; ADC configuration
                   ; default settings
       ldrc 0 254
       ldcr 4 0    ; Port A configuration ”1111 1110”
                   ; Only PA0 is output
       ldrc 0 0
```

```
       ldcr 5 0    ; PWM-Timer 0 configuration
                   ; default settings
       ldrc 0 0
       ldcr 6 0    ; PWM-Timer 0 configuration
                   ; default settings
       ldrc 0 0
       ldcr 7 0    ; PWM-Timer 0 configuration
                   ; default settings
       ldrc 0 0
       ldcr 8 0    ; PWM-Timer 1 configuration
                   ; default settings
       ldrc 0 0
       ldcr 9 0    ; PWM-Timer 1 configuration
                   ; default settings
       ldrc 0 64
       ldcr 10 0   ; PWM-Timer 2 ”tim/pwm(MSB)|INT|pe/ne/both/nu|nu|start/
                     stop|nu|timRST”
                   ; tim2 is: reset,stop,INT on falling of timout,no PWM
       ldrc 0 4
       ldcr 11 0   ; PWM-Timer 2 configuration ”PSC=0000 0100”
                   ; outpsc=0,8us (timerout ”pulse type”)
       ldrc 0 0
       ldcr 12 0   ; Port A mode configuration
                   ; configured as I/O , not as timers_out
       ldrc 0 0
       ldcr 13 0   ; Port B direction configuration
                   ; all pins output
       ldrc 0 0
       ldcr 14 0   ; Port B mode configuration
                   ; all pins digital
       ldrc 0 255
       ldcr 15 0   ; Port C direction configuration
                   ; all pins input
       ldrc 0 255
       ldcr 16 0   ; Port C mode configuration
                   ; pins as portC in input, not timerout(PC0 e’ l’INT)

       ldrc 0 255  ; RAM0=1111 1111
        ldpr 0 0   ; set all pins PA


;********* Variables definition ***********
; RAM0(”tmp”)  temporary var
; RAM1(”datotx”) data to transmit
; RAM2(”count”)  count sent digit
; RAM3(”flag”)
; RAM4(”datorx”) data received
```

```
; RAM5(”buffa”)  portA buffer

;********* MAIN ***********
start: ldrc 4 0    ; clear ”datorx”
       ldrc 3 0    ; reset ”flag”

call rx
       mirror 4     ; mirror ”datorx” I^ entry became LSB

       ldrc 2 8    ; init ”count”=8
       ldrc 0 0
       ldrr 1 4    ; ”datorx”=”datotx”

       call tx

       ldrc 0 17   ; load ”tmp” with ”0001 0001”
       ldcr 0 0    ; enable EXT_INT

       jp start

;********* END MAIN ***********

;======== reception CALL ===============
rx:

wt:    ldrc 0 255
       sub 0 3     ; Sub RAM0 with ”flag”. Z is set if ”flag”=255
       jpnz wt     ; infinite loop on Transmitter START_BIT

       ldrc 0 16   ; load ”tmp” with ”0001 0000”
       ldcr 0 0    ; disable EXT_INT

       ldrc 0 40   ; Tim2_counter=56 (56 x 0.8us=45us)
       ldpr 7 0    ; 49us==> samples in the middle of next bit
       ldrc 0 69
       ldcr 10 0   ; tim2 starts (with INT)

       waiti       ; skip start_bit INT

       ldrc 0 64
       ldcr 10 0   ; tim2 stop counting

       ldrc 0 128  ; Tim2_counter=120 (120 x 0.8us=96 us)
         ldpr 7 0  ; I^ sample at 150us, II^ sample at 250us ..etc

       ldrc 0 69
       ldcr 10 0   ; tim2 start counting
```

```
       ldrc 2 7    ; ”count”=7
next:  waiti
       ldri 5 9    ; Read PA; move port_A contents in ”buffa”
       ldrc 0 2    ; ”tmp”=0000 0010=mask to isolate PA1
        and 0 5    ; AND between ”tmp” and ”buffa”
       asr   0     ; LSB”tmp”=last bit received
        add 4 0    ; adds ”datorx” with last bit received
       asl 4       ; shift ”datorx” left

       dec 2       ; decrement ”count”
       jpnz next

       waiti
       ldri 5 9    ; Read PA; move port_A contents in ”buffa”
       ldrc 0 2    ; ”tmp”=0000 0010=mask to isolate PA1
        and 0 5    ; AND between ”tmp” and ”buffa”
       asr 0       ; LSB”tmp”=last bit received
        add 4 0    ; adds ”datorx” with last bit received

       waiti
       ldrc 0 64
        ldcr 10 0  ; tim2 stop counting

       ret
;========= END CALL ==========

;========== Transmission CALL ========

tx:     ldpr 0 0   ; reset pin PA: Start bit

       ldrc 0 125
        ldpr 7 0   ; Tim2_counter=128 (130 x 0.8us=104 us)

       ldrc 0 69
        ldcr 10 0  ; tim2 start counting

loop:  waiti       ; wait 100us for timer2_INT

       ldrc 0 1    ; RAM0=0000 0001 is the mask to isolate LSB
       and 0 1     ; AND between ”tmp” and LSB”datotx”
       ldpr 0 0    ; PA0=LSB of ”datotx”
       asr 1       ; shift ”datotx” for next INT

       dec 2       ; decrement ”count”
       jpnz loop
```

```
       waiti
       ldrc 0 1    ; Stop bit (Hi)
       ldpr 0 0    ; set PA0

       waiti

       ldrc 0 64
         ldcr 10 0  ; tim2 stop counting

       ret
;========= END CALL ==========


;**** INTs Subroutines *********

EXT_INT:
       ldrc 3 255 ;”flag”=255
       reti

AD_INT: reti

TIM0:  reti

TIM1:  reti

TIM2:  reti
```