USING THE ST626X SPI AS A UART

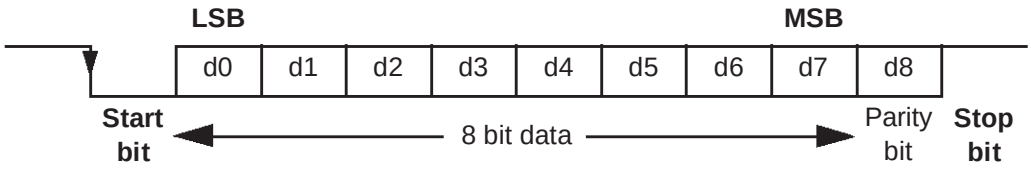
by 8-bit Micro Application Team

1 INTRODUCTION

The purpose of this note is to give indications on how to use ST626x SPI in order to perform UART serial communication. The principles of operation as well as limitations are described. As an example, reception and transmission at 9600 baud are presented. However, baud rates up to 19200 are supported.

2 UART OVERVIEW

A UART (Universal Asynchronous Receiver Transmitter) is an asynchronous communication interface. The protocol used in this application note is as follows:



A reception or transmission frame contains:

– **1 START BIT** that is a high to low transition (falling edge). The START BIT is valid if the line stays at level '0' during one SPI clock (one bit time period). It corresponds to the beginning of receive or transmit frame.

– **8 BITS of data** corresponding to the data to be received or transmitted. The LSB is input or output first.

– **1 PARITY BIT** or **2ND STOP BIT** or **1 EXTRA BIT**

– **1 STOP BIT** which is a '1' level during one bit time period. It corresponds to the end of frame.

The commonly used baud rates in asynchronous communication are: 1200, 2400, 4800, 9600, and 19200 baud. Transmission is started by sending a Start bit (a '0' for one bit time period) followed by the 8 data values (LSB first) and the value of the 9th bit. The output is then set to '1' for of one bit time period to generate a Stop bit. Reception is started when a falling edge is detected on the input pin. The 9-bit stream that follows is shifted in and the UART waits during one bit time period for the Stop bit.

Note: It is possible to receive and transmit more or less than the described 8 bits data + 1 parity bit while using the ST626x SPI as a UART. The number of bits per burst is largely up to the user.

3 ST626X SPI OVERVIEW

The SPI peripheral is a synchronous serial interface with programmable receiver and transmitter modes. It can operate either in Master or in Slave mode. It consists of 3 pins SCK (clock), SIN (data in) and SOUT (data out) which also provide alternate I/O pin functions (PC4, PC2, PC3). The clock polarity, clock phase and the number of bits per burst can be selected using the SPI Mode Control and Divide registers.

The SPI comprises the following 8-bit registers:

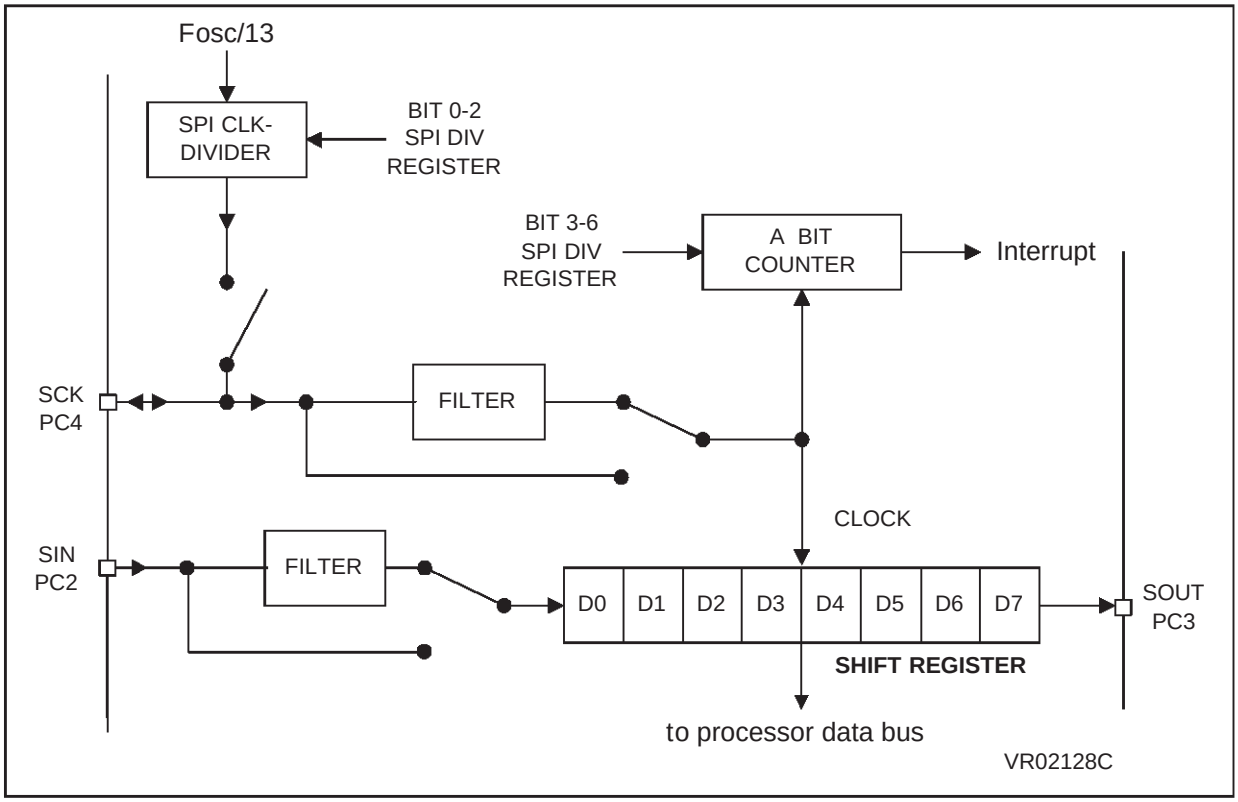
– an 8-bit Data/Shift register SPIDSR (E0h)

– a Divide register SPIDIV (E1h)

– a Mode Control register SPIMOD (E2h)

– a Miscellaneous register MISCR (DDh)

Figure 1. SPI Block Diagram



Data transfer to or from the SPI Shift register (SPIDSR, the case in this note) is started by setting SPRUN (bit 7 of SPIMOD register) to '1'. The SPI Shift register is simultaneously fed by the SIN pin and feeds the SOUT pin during both transmission and reception. The SPRUN bit is automatically cleared by the SPI cell at the end of transmission or reception and an interrupt request can be generated. It is associated with interrupt vector #2.

The division ratio between the core clock (F_{osc} divided by 13) and the clock supplied to the Shift register in Master mode is shown in Table 1.

Table 1. Selection of clock division ratio

SPIDIV BIT 0-2 (CD2-CD0)	Division Ratio (DIV)	Baud rates at Fosc = 4Mhz (4Mhz/13/DIV)	Baud rates at Fosc = 8Mhz (8Mhz/13/DIV)
0 0 0	1.00	307600	615300
0 0 1	2.00	153800	307600
0 1 0	4.00	76800	153800
0 1 1	8.00	38400	76800
1 0 0	16.00	19200	38400
1 0 1	32.00	9600	19200
1 1 0	64.00	2400	9600
1 1 1	256.00	1200	2400

4 OPERATING PRINCIPLES

ST626x SPI will be set in Master mode to use the internal clock division (SPCLK, bit 4 of SPI Mode Control Register is set to '1').

4.1 RECEIVER MODE (EX: 9600 BAUD AT FOSC=8MHZ)

The 3 SPI pins are then configured as follows:

– SCK in Output Push-pull '0'

– SIN in input with pull-up and interrupt (falling edge)

– SOUT in output push-pull '1'

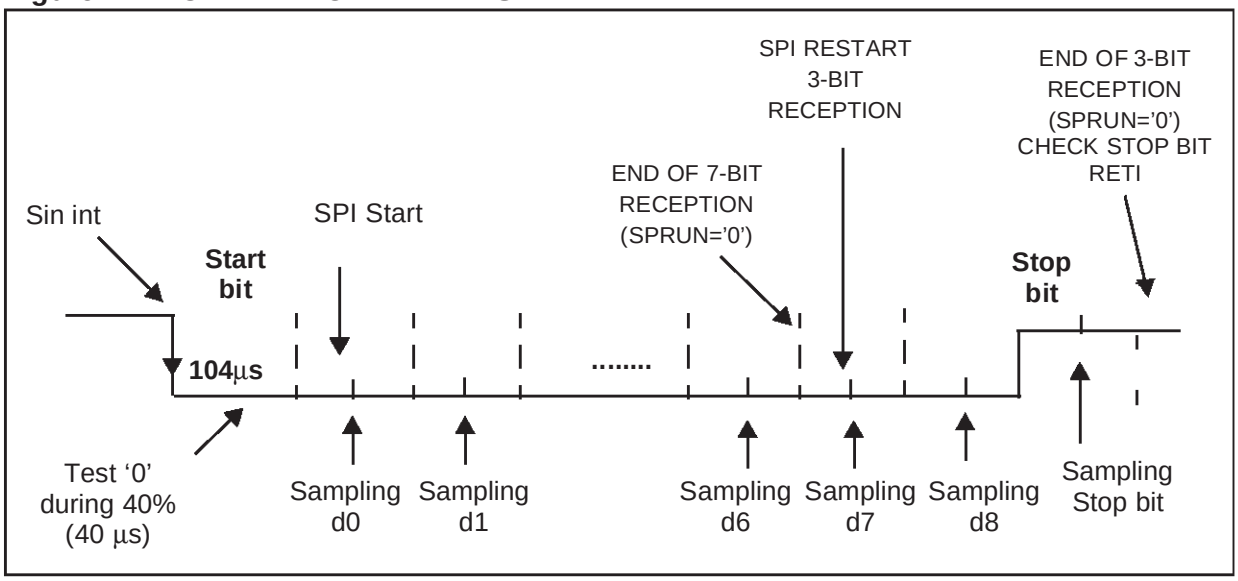
Set: MISC = 01h (SOUT switched to SPI output)

SPIDIV = 3Eh (DIV 64, 7 bits per burst)

SPIMOD = 18h (Master mode, Normal polarity and phase, SPI interrupt disabled, SPI start with SPRUN bit)

SPIDSR = 0FFh (To prevent shifting a level '0' on SOUT pin during reception)

Figure 2. RECEIVER MODE EVENTS



START BIT DETECTION:

Since the SIN pin (I/O PC2) is in input with pull-up and INTERRUPT (falling edge sensitive), when a high to low transition occurs on the RXD line, the program jumps to the PORT C interrupt vector (vector #2).

START BIT VALIDITY CHECK:

Inside the vector #2 interrupt routine, check if SIN is still at level '0' for around 40% of the clock period (ex: at 9600 baud, 40% of 104µs is 41µs => use the "jrs, SIN, enditspi" instruction 5 times for fosc = 8 Mhz). If yes, continue the reception, if not, exit the interrupt routine. The SIN pin is reconfigured as Input with pull-up WITHOUT interrupt.

FIRST 7-BIT DATA RECEPTION:

Use a delay routine to wait for the completion of 1.5-bit time period (from start bit detection) before starting the SPI (SPI registers must be previously configured during initialisation phase). 7 bits will be sampled at a rate of one bit time period. At the end of a 7-bit data reception, the SPRUN bit (bit7 of the SPIMOD register) is automatically reset. Poll for SPRUN='0'. Store the first 7-bit data received. Reload the SPI Data Register with 0FFh to prevent an output of '0' on the SOUT pin (spurious transmission start).

8th BIT, PARITY BIT AND STOP BIT RECEPTION:

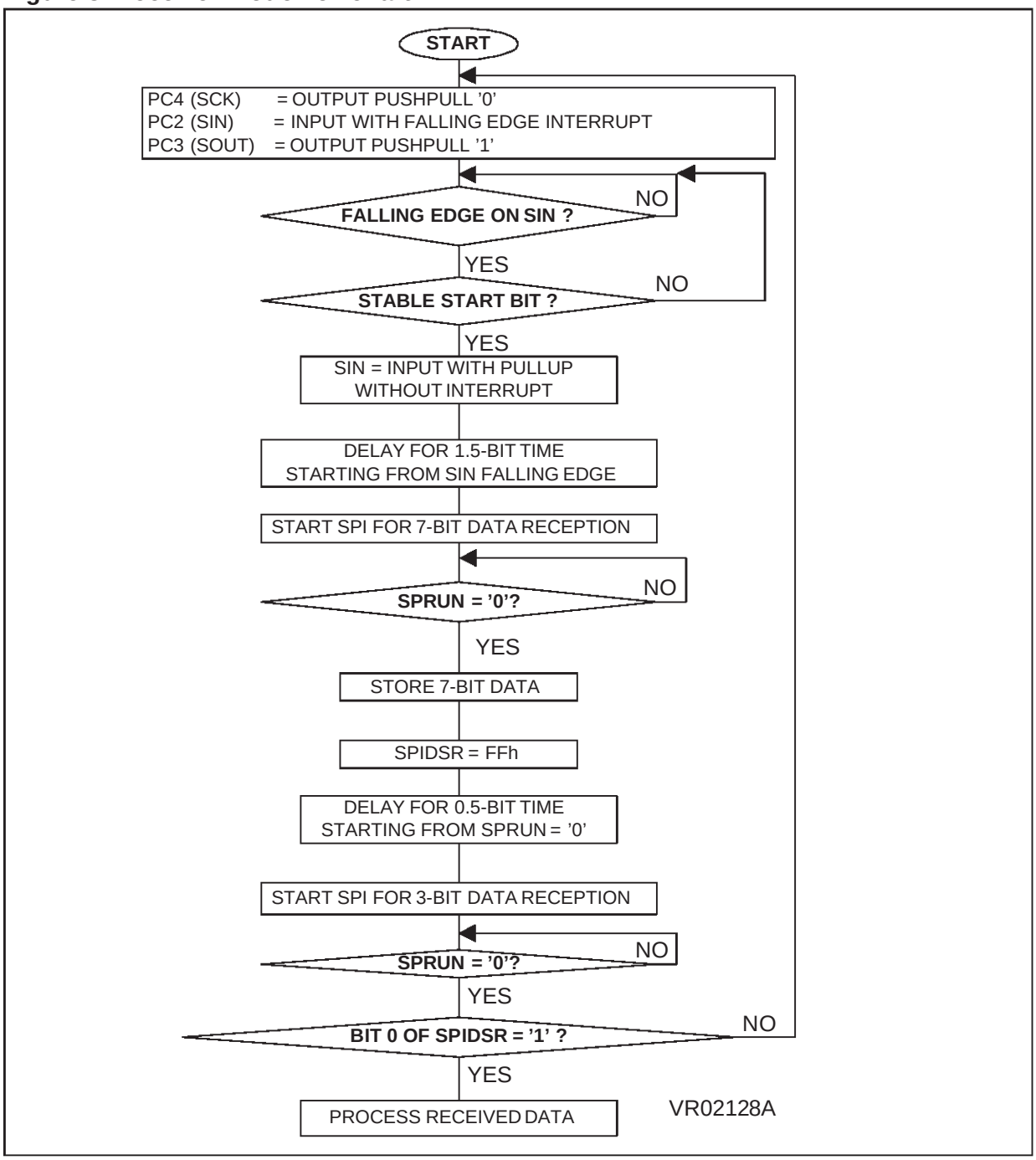
Clear the SPI interrupt and reconfigure the SPIDIV register to 1Eh (DIV 64, 3 BITS per BURST). Restart the SPI (SPRUN bit of SPIMOD register set to '1') after a temporisation of one half bit time period (from SPRUN='0' detection). Poll the SPRUN bit for the end of 3-bit reception.

STOP BIT CHECK:

Check if bit0 of SPIDSR register is ’1’. If yes, process the received data. If not, cancel reception (do not treat received data). Then exit the vector 2 interrupt routine.

The receiver mode flowchart is given below:

Figure 3. Receiver mode flowchart



4.2 TRANSMITTER MODE (EX: 9600 BAUD AT FOSC=8MHZ)

The 3 SPI pins are then configured as follows:

– SCK in output push-pull '0'

– SIN in input with pull-up

– SOUT in output push-pull '1'

Set:	MISC = 00h	(SOUT switched to PC3 I/O output)
	SPIDSR = XXh	(Data to be transmitted: LSB in bit7...MSB in bit0, LSB must be ;shifted out first)
	SPIDIV = 3Eh	(DIV 64, 7 BITS per BURST)
	SPIMOD = 18h	(Master mode, Normal polarity and phase, SPI interrupt disabled, ;SPI start with SPRUN bit)

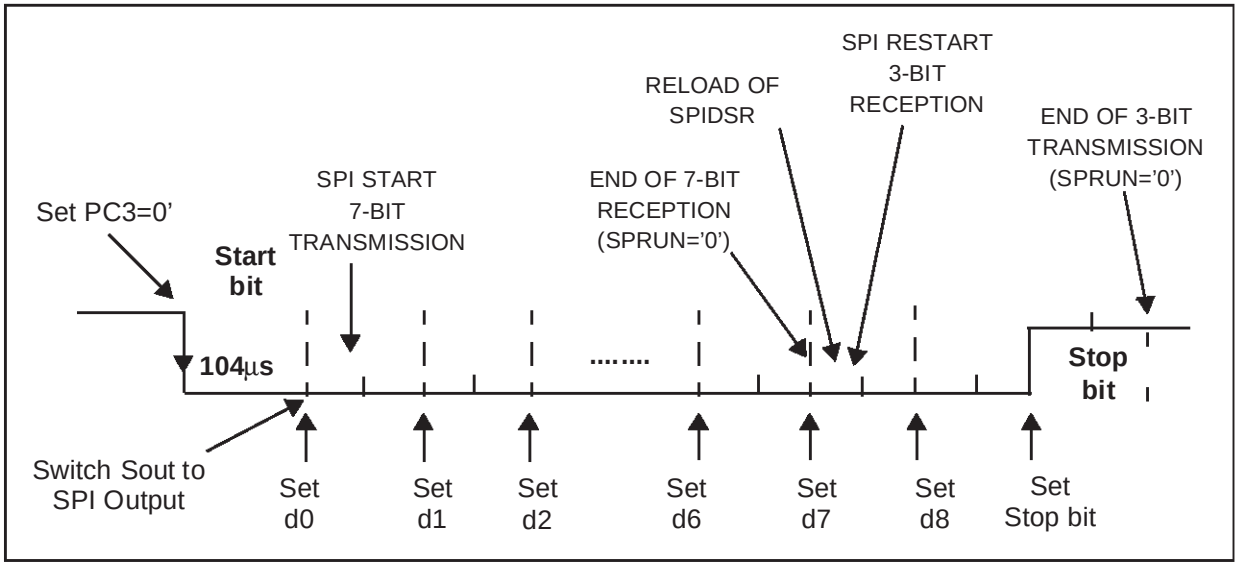
START BIT GENERATION:

Since the SOUT pin is switched to PC3 I/O output and is configured as output PUSHPULL '1'. By resetting the SOUT (PC3) data register, this will generate a High to Low transition (=> START BIT). This level '0' is kept during one bit time period.

7-BIT DATA TRANSMISSION:

Switch the SOUT pin to SPI output (MISC=01h) and start the SPI. 7 bits will be automatically transmitted. Poll the SPRUN bit in the SPIMOD register to synchronise with end of transmission (it automatically goes to '0'). The SPI interrupt is intentionally disabled during transmission and reception to avoid extra code to distinguish an end of reception, an end of transmission and a PORTC interrupt that are all mapped to vector #2.

Figure 4. Transmitter Mode Events

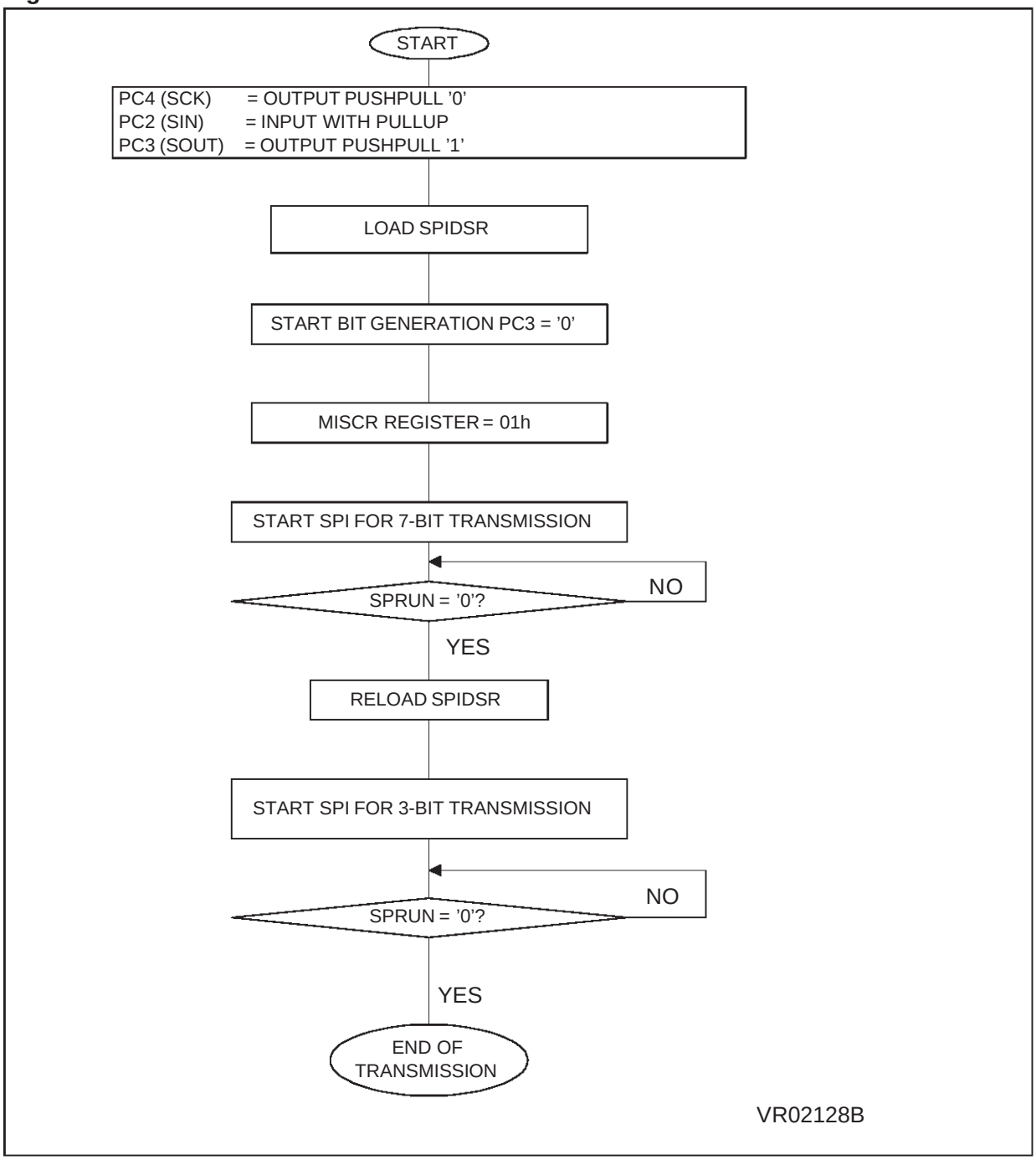


8th BIT, PARITY BIT AND STOP BIT TRANSMISSION:

Reconfigure the SPIDIV register to 1Eh (DIV 64, 3 BITS per BURST). RELOAD the SPI data shift register with new data (bit7/MSB of SPIDSR is for the 8th bit, bit6 is for parity bit and bit5 is for stop bit) and restart SPI (SPRUN bit of SPIMOD register set to '1'). Poll SPRUN bit of SPIMOD register to detect end of 3-bit transmission.

The transmitter mode flowchart is given below:

Figure 5. Transmitter Mode Flowchart



4.3 GENERAL REMARKS

1. Even though the ST626x can receive or transmit up to 15 bits per burst, there is only one 8-bit data shift/receive register SPIDSR. When more than 8 bits per burst is programmed, it is not possible to get or transmit the total number of bits using only the SPI peripheral. Other I/O pins have to be used. For example, one I/O can be connected to the SIN pin to transmit the 9th to 15th bit, one I/O can be connected to the SOUT pin to receive the 1st to the 8th bit and one I/O can be connected to SCK while in Master mode to synchronise bit sampling (reception) or bit setting (transmission) with UART clock. If such connections are not implemented, only the last 8 received bits can be retrieved during reception and only the first 8 bits can be correctly transmitted. To avoid this heavy configuration, to transmit 13 bits for instance, it is better to transmit one first burst of 7 bits and then a second burst of 6 bits instead of one single burst of 13 bits.
2. A certain number of instructions must be executed (start bit validity test, start/restart of SPI, switching of SOUT pin...) after start bit detection or after one burst transfer. The processing sequence usually needs minimum 5 instructions. Since one ST6 instruction usually takes 4 instruction cycles (i.e. 6.5µs at 8Mhz), this limits the possible transfer baud rates. The possible baud rates are:

SPIDIV BIT 0-2 (CD2-CD0)	Division Ratio	Baud rates at 4Mhz (4Mhz/13/DIV)	Baud rates at 8Mhz (8Mhz/13/DIV)
1 0 1	32.00	9600	19200
1 1 0	64.00	2400	9600
1 1 1	256.00	1200	2400

The SPI clock periods given by SPI division ratio are:

Baud rates	SPI clock period
19200	52 µs
9600	104 µs
2400	416 µs
1200	832 µs

These timings are used as reference for reception or transmission synchronisation.

3. It is mandatory to take the SPI timing diagrams into account (4 possible choices, depending on clock polarity and phase selection, please refer to the ST626x databook) in order not to generate spurious falling edge on the SOUT pin during reception, which can be understood by the other chip as a start of transmission. Since shifting data in through SIN will simultaneously shift data out through SOUT, when the first bit shifted in is a '0', a level '0' will be present on the SOUT pin after receiving 8 bits (with SPI configured as 8 bits per burst in reception mode, CPOL=0, CPHA=0). Therefore, care must be taken to load the SPI data/shift register with 0FFh before any reception and receive less than 8 bits per burst. For example, to receive 10 bits, one can receive a first burst of 7 bits and then a second burst of 3 bits instead of a first burst of 8 bits and a second of 2 bits.
4. For transmission, it is also better to transmit less than 8 bits per burst. During an 8-bit transmission process, the first shifted bit into the SPIDSR register should be a level '1' since the SIN pin should always be at level '1' during transmission (Half duplex mode). Therefore, after 8 bit transmission, a level '1' is present on the SOUT pin (with the SPI configured as 8 bit per burst transmission mode, CPOL=0, CPHA=0). If the Parity bit to be transmitted is '0', there will be a glitch on the SOUT pin until the SPIDSR register is reloaded. Care is then required to reload SPIDSR register as soon as possible after the first transmitted burst otherwise a wrong value will be read for the parity bit. To avoid this problem, for example, when transmitting 10 bits, it is safer to transmit a first burst of 7 bits (1st to 7th bit) and then a second burst of 3 bits (8th bit, parity bit and stop bit) instead of a first burst of 8 bits and a second of 2 bits. For the first burst of 7 bits, the SPIDSR register must nevertheless contain the first 8 bits in order to pre-position the 8th bit during the transition between the 2 bursts.
5. In this note, the SPI is started by setting the SPRUN bit (bit 7 of SPIMOD register) to '1'. Independently of selected the SPI divide ratio, the first active edge of the SPI clock, while in Master mode, will be present at SCK (PC4) pin, 9 or 10 Fosc clocks after SPRUN is set. This explains the chosen SPI start position in receiver and transmitter modes.

4.4 LIMITATIONS

1. ONLY HALF DUPLEX MODE IS POSSIBLE: transmission and reception can not be processed at the same time.
2. MAIN CONSTRAINT IN RECEIVER MODE: the CPU must be available during the whole reception i.e. NO INTERRUPT SHOULD OCCUR from the start to the end of reception otherwise the transfer will be desynchronised and the received data wrong. If maskable interrupts are needed in the code, the NMI pin must be used to set the reception as a NON-MASKABLE INTERRUPT routine. NMI is connected to the SIN pin. When a Start bit occurs, the high-to-low transition will generate an NMI and the MCU switches to NMI mode under which the reception will be processed. If another falling edges are received on the NMI pin, only one is latched and will be processed when the first NMI ends up. To distinguish a real start bit interrupt from a spurious NMI interrupt, the SIN input is read at the beginning of NMI interrupt routine. If it is a '0' level, this is a Start bit and the reception routine can be executed. If it is a '1' level, exit the NMI routine.
3. Receiver mode has a higher priority than transmitter mode which is the opposite case in a real UART. If a transmission is interrupted by a reception, the data transmitted will be corrupted. Therefore, software MUST monitor the priority using an I/O pin (ex: PC0) as a BUSY FLAG. Before transmitting data to the ST626x MCU, the external chip must check if the line is busy (ST626x transmitting) or not (PC0 set to '1'). If busy, it must wait until the end of the ST626x transmission. The ST626x micro should have the same behaviour: before transmitting to the external chip, it should check the busy flag.
4. A typical UART cell provides a 2-to-1 majority voting system inside the data reception mechanism to determine the logic state of the asynchronous incoming serial logic bit by taking 3 timed samples within a bit time. Using ST626x SPI as a UART does not provide this "2 to 1" sampling decision. The RXD line must therefore be stable (no fluctuation).

5 APPENDIX

A code example for 9600 baud rate communication is given below. The transmission and reception results are shown immediately after the code.

```
;*************************************************************************
;*
;*                  USING ST626X SPI AS UART
;*                  METHOD: SPI 7-BIT BURST + 3-BIT BURST
;*                  AT 9600 BAUD RATE with Fosc = 8Mhz
;*                  BASED ON SCK STARTS 9 CLOCKS AFTER SPRUN='1'
;*************************************************************************
          .W_ON
          .DP_ON

;|=========================================================
;|       Assembly method without linker : ast6 -s -l -m  uart6x.asm
;|                       (Need 626x.asm file)
;|=========================================================

;|=========================================================
;|                  Assembly directives
;|=========================================================
          .title "ST626X SPI AS UART"; This directive
                                ; defines a new title
          .vers "st6265"
          .romsize 4

;*************************************************************************
;*                  Data space needed registers
;*************************************************************************
count     .def  084h  ; Used for number of delay loop
spi_reg   .def  085h   ; b7:8th bit; b6:parity bit; b5:stop bit to
                       ;be transmitted during
                       ; 2nd 3-bit BURST
senddata   .def  086h ; The first 8-bit Data to be transmitted
                       ;(b7: MSB; b0:LSB).
spirec    .def  087h!m; Store the first 7 bits received.

;|=========================================================
;|                  ST626x registers
;|=========================================================
          .input "626X.asm"
```

```
;|============================================================
;|                  constants definition
;|============================================================

;SPIMOD USED BITS
;===============
sprun     .equ  7
spie      .equ  6
spin      .equ  3
spint     .equ  7

;SPI_REG USED BITS
=================
d7        .equ  7
d8        .equ  6
stopbit   .equ  5

;SPI PINS IN PORTC
================
scl       .equ  4
sout      .equ  3
sin       .equ  2

;*************************************************************************
;*                  PROGRAM SPACE
;*************************************************************************

;|============================================================
;|                  main program
;|============================================================
          .org 0080h

start     reti
          call init_spi

wavail
       jrr 0, drc, wavail; Wait for end of reception if processed
          ; jp uarttxd       ; Use this to debug transmission (jump to
                        ;transmission routine)
```

```
;*************************************************************************
;  ROUTINE TO DEBUG RECEPTION
;  NEED TO CONNECT ARTIMout/PB6 to Sin/PC2 to get reception frame
;*************************************************************************
receive
          ldi arlr, 00h      ; Connect ARTIMout/PB6 to Sin/PC2 in order
                        ;to start reception
          ldi arrc, 00h
          ldi arcp, 0BFh
          ldi arsc0, 00h
          ldi arsc1, 0C0h
          ldi armc, 0E0h

loop      jp loop        ; To debug reception


;*************************************************************************
;  ROUTINE TO DEBUG TRANSMISSION
;  SEND: Startbit='0';d0='1';d1='0';d2='1';d3='0';d4='1';d5='0';d6='1'
; d7='0' ;d8='0';Stopbit='1'
;*************************************************************************
uarttxd
       ldi drc, 008h      ; Line busy (pc0= '0') => can not receive NOW
          ldi senddata, 055h   ; Data to be transmitted (d0...d7)
          ldi spi_reg, 0BFh    ; b7: 8th bit (d7), b6: 9th bit (d8), b5:stopbit.
                        ;Here d7='1, d8='0, Stopbit='1'
          jrs 7, senddata, setd7; Program 8th bit into spi_reg which is
                        ;content of 2nd burst transmission
          ldi spi_reg, 03Fh    ; b7: 8th bit (d7), b6: 9th bit (d8),
                        ;b5:stopbit. Here d7='0, d8='0, Stopbit='1'
setd7     call transmit
          ldi drc, 09h       ; Free Busy flag line
          jp wavail


;*************************************************************************
;*            USED SUBROUTINES
**************************************************************************
```

```
;|============================================================
;|       SPI INIT ROUTINE
;|============================================================
init_spi
          ldi drc, 009h      ; Sout, pc0 in input without pullup. PC0 is
                             ;for the busy flag.
          ldi ddrc, 019h     ; Scl, Sout, pc0 in output opendrain '1' (Sout,
                             ; pc0), '0' (Scl)
          ldi orc, 01Dh      ; Sout (pc3)and pc0 in pushpull '1', SCL
                             ;(pc4) in pushpull '0'
                             ; Sin is in input pullup with interrupt
                             ; (falling edge)
          ldi spimod, 018h    ; Master mode, normal clock, no filter,
                             ;start by sprun, sin
                             ; SPI interrupt disabled because 1
                             ;spurious interrupt.
          ldi spidiv, 03Eh ; 9600 baud rate => 8Mhz/64*13,
                             ;Burst of 7 bits (104us/bit)
          ldi spidr, 0FFh     ; To make sure that when reception shift,
                             ;no spurious start bit

          ldi ior, 10h       ; Vector 2 falling edge - authorize
                             ;maskable interrupt
          ret
```

```
;|============================================================
;|       TRANSMISSION ROUTINE
;|============================================================
transmit
         call rotate
         ld spidr, A       ; Set the value to be transmitted
         ldi drc, 00h       ; Transmit Startbit i.e. '0' on Sout
         ldi count, 07h      ; Need 1 bit time (64cuc) from PC3='0'
                     ; falling edge to switch
                     ; Sout to SPI output
         call tempo      ; Set delay time
         nop             ; 2 extra cyc
         ldi misc, 01h     ; Switch Sout to spi output: here, we waited for
                   ;12cyc(process)+[8*6+2]cyc(tempo)+2cyc(nop)=64cyc
         ldi count, 01h       ; Wait for 0.5 bit time to start SPI
         call tempo       ; Set delay time
         nop              ; 6 extra cyc
         nop
      nop
         set sprun, spimod   ; Start spi => send d0...d7 (8bits)
                     ; Here, we waited from misc=01h:
                     ; 12cyc(process)+14cyc(tempo)+6cyc(nop)=32cyc
         ld A, spi_reg       ; Set 8th bit, 9th bit + stopbit (b5)
         ldi drc, 08h     ; Set Sout to '1' to prevent spurious txd start bit

wtxd1    jrs sprun, spimod, wtxd1
                     ; Wait for spi end of transmission interrupt
                     ; It occurs when Sck goes to 0 after the 8th bit.
         ld spidr, A    ; Set 8th, 9th and stop bits to spi data register
         ldi spidiv, 01Eh  ; 3 bits per burst
         nop             ; Extra 2 cyc
         ldi count, 00h   ; Set delay time: need to wait for 0.5 bit time before
restart
                   ; SPI.
         call tempo        ; From SPRUN='0', we waited:
                     ; 20cyc(process)+8cyc(tempo)+2cyc(nop)=30cyc
         set sprun, spimod   ; Wait for 0.5 bit time and start SPI again
```

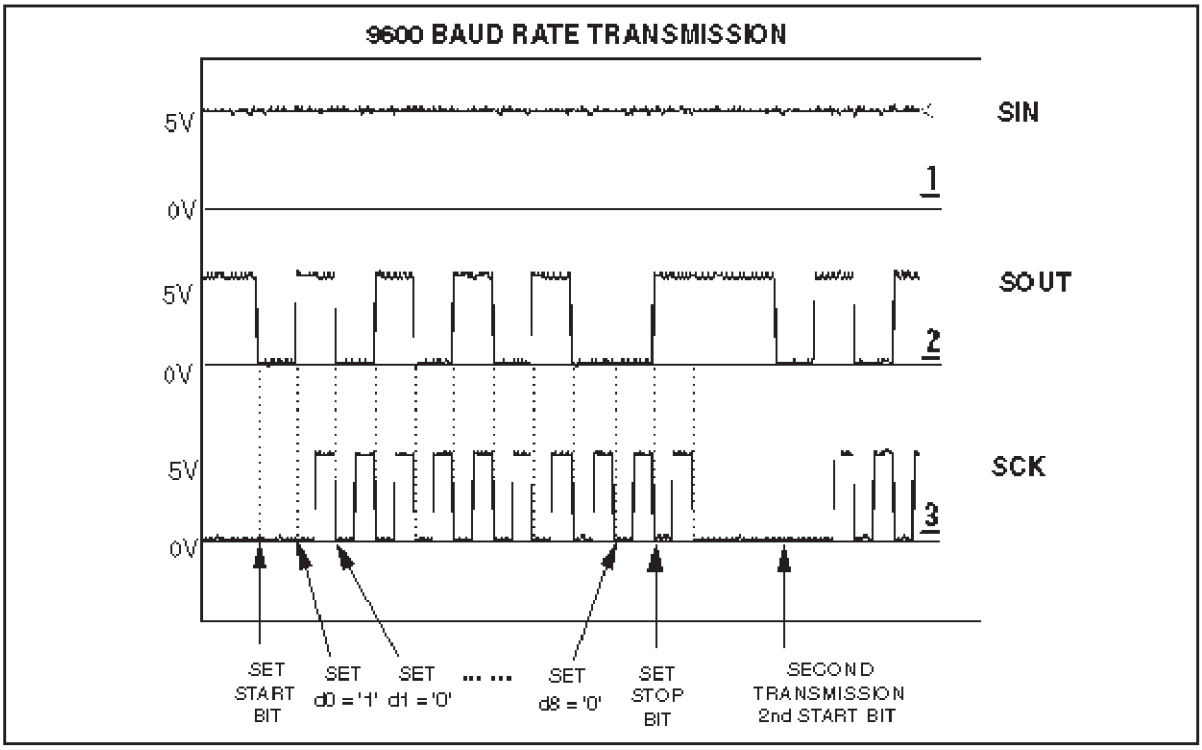
```
wtxd2     jrs sprun, spimod, wtxd2
                    ; Wait for spi end of transmission interrupt
                    ; 9th bit and stop bit transmitted
          ldi misc, 00h       ; Disactivate SPI
          ldi spidiv, 03Eh   ; 9600 baudrate => 8Mhz/64*13, Burst of 7 bits
                    ;(104us/bit)
          ldi spidr, 0FFh ; In case of reception: no spurious start bit on Sout
          ret

;|============================================================
;|        ROTATION subroutine
;|        70 cycles (113.75us at 8Mhz)
;|        b7,b6,b5,b4,b3,b2,b1,b0 => b0,b1,b2,b3,b4,b5,b6,b7
;|============================================================
rotate
          clr A
          jrr 0, senddata, bit1
          set 7, A          ; Can also use “sla A” because result of jrs is in
                    ; carry
bit1      jrr 1, senddata, bit2
          set 6, A
bit2      jrr 2, senddata, bit3
          set 5, A
bit3      jrr 3, senddata, bit4
          set 4, A
bit4      jrr 4, senddata, bit5
          set 3, A
bit5      jrr 5, senddata, bit6
          set 2, A
bit6      jrr 6, senddata, bit7
          set 1, A
bit7      jrr 7, senddata, endrot
          set 0, A
endrot    ret
```

```
;|============================================================
;|       DELAY routine
;|       9.75us for each loop
;|       total cycles= 6+(n*6)+2= [(n+1)*6 +2] cycles
;|       total: (count+1)*9.75us
;|============================================================
tempo
         ld A, count
         jrz tempend   ; If count=0 => no loop
tloop    dec A         ; 9.75us for each loop
         jrnz tloop
tempend
         ret


;|============================================================
;|       SECTION 1
;|============================================================
   .org 0800h

;|============================================================
;|             interrupt routine
;|============================================================
;|------------------------------------------------------------------------
-------------------------------------
;|       SPI, PORTC interrupt
;|       RECEPTION ROUTINE: 9600 baud; 7-bit reception+ 3-bit reception
;|------------------------------------------------------------------------
-------------------------------------
it_int2
         ldi drc, 008h       ; If receive startbit => set busy flag
         jrr spint, spidiv, it_rxd1   ; Check if interrupt from spi or not
         jp enditspi   ; SHOULD NOT HAVE SPI IT SINCE DISABLED
it_rxd1               ; First, receive start bit and prepare 8 bits
                      ; reception using SPI
         jrs sin, drc, enditspi  ; Test stability of startbit for around 40% of
period
         jrs sin, drc, enditspi ; Test stability of startbit
         jrs sin, drc, enditspi ; Test stability of startbit
         jrs sin, drc, enditspi ; Test stability of startbit
         jrs sin, drc, enditspi ; Test stability of startbit
                      ; Time spent for test: 34cyc out of 64cyc => 53%
nextrxd               ; We need 1.5-bit time from falling edge of start
                      ;bit (96cyc)
         res sin, orc     ; Set Sin (PC2) as input with pullup WITHOUT INT
```

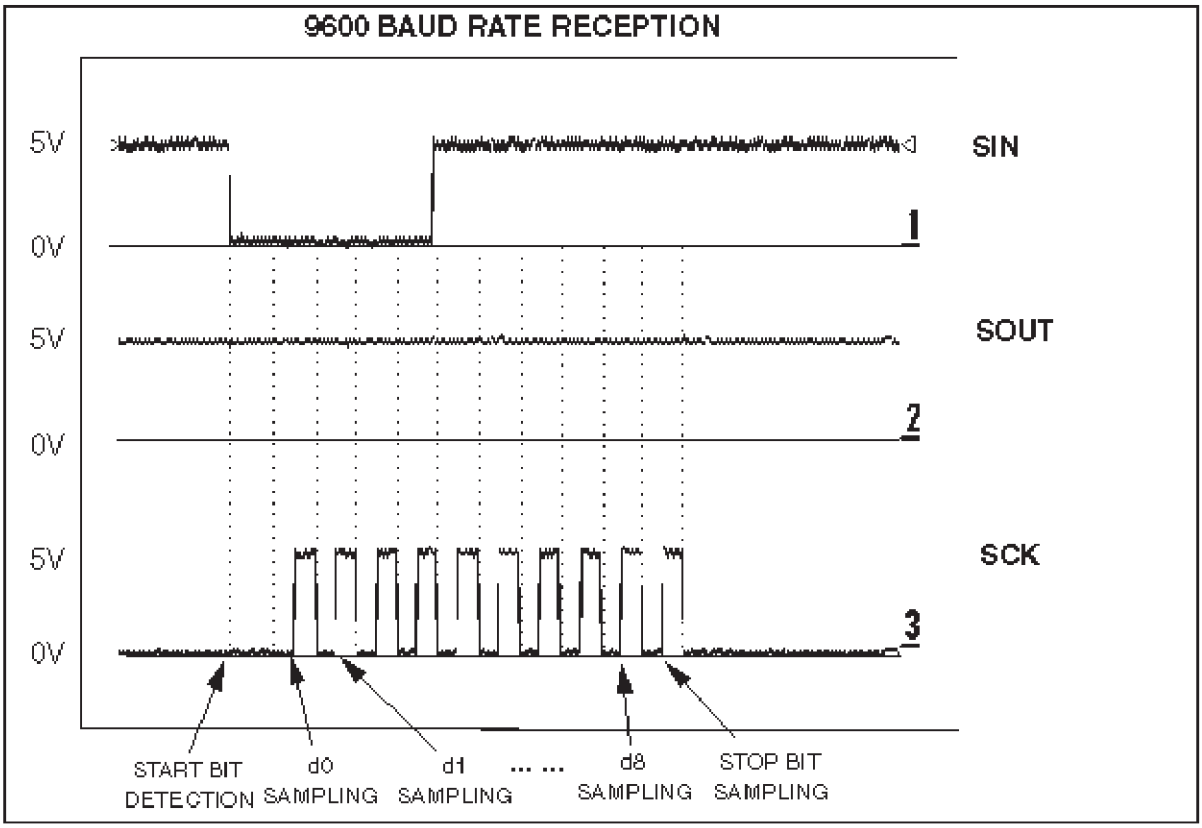
```
          ldi misc, 001h      ; Switch Sout to spi output
          ldi count, 05h     ; Set delay time
          call tempo      ; From it_int2, we waited for: 54cyc(process)
                         ;+38cyc(tempo)+4cyc(nop)=96cyc(need96cyc)
          nop              ; 4 extra cycles for delay
          nop

          set sprun, spimod  ; Start spi and SPI interrupt still disabled
                         ; => receive d0...d6 (7bits). From it_int2 to here,
                         ; we waited for: 96cyc
wrxd1  jrs sprun, spimod, wrxd1
                      ; Wait for end of 7 bits reception
it_rxd2                  ; Now, Wait 0.5-bit time and receive the 8th, 9th bit &
check stopbit.
          ldi spidiv, 1Eh     ; Receive NOW 3 bits
          ld A, spidr     ; Retrieve first 7 bits data
          ldi spidr, 0FFh    ; To avoid spurious TXD interrupt when shifting in/out
          ldi count, 00h    ; Dummy load for delay
          ; call tempo     ; Too many clocks even if count=00h
          ldi count, 00h    ; Dummy load for delay
          ldi count, 00h    ; Dummy load for delay
          nop              ; 2 extra nop
                         ; From SPRUN='0', we waited for
                         ; 16cyc(process)+12cyc(dummyldi)+2cyc(nop)
                         ; =30cyc (instead of 32cyc)
          set sprun, spimod   ; Start spi (SPI interrupt still disabled).
                         ; Receive d7,d8,stopbit

wrxd2     jrs sprun, spimod, wrxd2
                      ; Wait for end of 3-bit reception
          jrr 0, spidr, enditspi ; If no stop bit => DO NOT TREAT RECEIVED DATA
          ld spirec, A       ; Save to a special data register (not mandatory,
                         ; used for debug)
          ld A, spidr       ; Retrieve 2nd part

process;...              ; Call processing routine
enditspi
          ldi misc, 00h       ; Disactivate SPI
          ldi spidiv, 03Eh     ; Reconfigure SPI to 7 bits per burst.
          ldi spidr, 0FFh     ; Reload SPIDSR to 0FFh for next reception.
          set sin, orc     ; Set again Sin (PC2) as input with interrupt
          ldi drc, 009h   ; Release busy flag => line free again
          reti
```

```
;*************************************************************************
;*       Restart and interrupt Vectors
;*************************************************************************
     .org 0ff0h

int4      nop       ; Adc, Timer
          reti
int3      nop   ; Artimer
          reti
int2      jp it_int2  ; Spi, Portc
int1      nop       ; PortA, PortB
          reti

     .org 0ffch

nmi       nop
          reti
res       jp start
```

TRANSMISSION RESULT



RECEPTION RESULT



APPENDIX

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.