INTRODUCTION

The goal of this application note is to present a basic software driver to emulate a master SPI full duplex communication through standard ST7 I/O ports.

1 SPI COMMUNICATION DEFINITION

The software SPI communication is based on three signal lines:
- CLK: Clock line
- MO: Master output data line
- MI: Master input data line

These three signals allow synchronous full duplex master communication.

PHASE AND POLARITY

The timing relationship between clock and data signal is fixed. The polarity can be reversed easily inverting the falling and rising edges of the clock in the software. Concerning the phase a deeper modification has to be done reversing the read and the write data order in front of the clock signal. The chosen implemented configuration is given by the Figure 1.

Figure 1. Clock / Data Sampling

![Clock / Data Sampling Diagram]

CLK
MO

Read data bit from MI

Write data bit on MO
2 SPI SOFTWARE COMMUNICATION DRIVER IMPLEMENTATION

2.1 HARDWARE CONFIGURATION

The ST7 software SPI communication driver test hardware is composed by a single ST72251 microcontroller (see Figure 2.) running with $f_{CPU}$ at 8 MHz.

Three I/O ports (PC0, PC1 and PC2) are used as SPI communication lines (resp. MI, MO, CLK). As it is a full duplex communication, the MO and MI lines are connected together to allow a pseudo simultaneous data write/read from/to the ST7 memory.

The signal can also be checked through an oscilloscope.

Figure 2. ST7 Software SPI Communication Driver Test Configuration

2.2 SOFTWARE DRIVER ALGORITHM

Based on the previous hardware description, the driver test software is composed by a transmission of an 8-byte ROM buffer on MO line concurrently received on the MI line and then stored in a 8-byte RAM buffer. This parallel sequence is possible thanks to the full duplex capability of the communication.

**FULL DUPLEX 8-BIT COMMUNICATION DRIVER**

The principle of this driver is to use the ST7 Carry flag of the CC register as the data slicer. The data byte to transmit is given in A register and the receive data byte is returned in the same A register. The flowchart shown in Figure 3. explains this algorithm.

**SPEED**

The maximum communication speed is computed from two parameters:
– $f_{CPU}$ which gives the CPU cycle time
– the software instruction sequence which composes the driver described in Figure 3.

This maximum speed can be decreased by the software inserting waiting steps.
As the ST7 is running at 8 Mhz internal frequency, the maximum speed is given by the number of ST7 instruction cycles needed for this communication (see Figure 4.) taking into account a 50% ratio of high/low level on the bus.

\[ f_{SPI} = \frac{f_{CPU}}{\left(\frac{32 + 35}{2}\right)} = 238, 8KHz \]

**Figure 4. Communication Speed**

Write data bit on MO  Read data bit from MI

\[ T_{PERIODE} = 32\sim35 \text{ cycles} \]
3 SOFTWARE

The assembly code given below is for guidance only. For missing label declaration please refer to the register label description of the datasheet or the ST web software library ("ST72251.inc" file...).

```assembly
TITLE "SPI_SW.ASM"
MOTOROLA
INCLUDE "ST72251.inc" ; ST72251 registers and memory mapping file

;******************************************************************************
; Macro definitions
;******************************************************************************
#defineSPI_MI PCDR,#0 ; SPI MI data line ... PC0 : Input floating.
#defineSPI_MO PCDR,#1 ; SPI MO data line ... PC1 : Output PP.
#defineSPI_CLK PCDR,#2 ; SPI clock line .....PC2 : Output PP.

;******************************************************************************
; RAM SEGMENT
;******************************************************************************
BYTES ; following addresses are 8 bit length
segment byte at 80-FF 'ram0'
.buf_in DS.B 8 ; Input data buffer.

;******************************************************************************
; ROM SEGMENT
;******************************************************************************
WORDS
segment 'rom'
.buf_out DC.B $FF,$F0,$F0,$CC,$33,$AA,$55,$00 ; Constant output data buffer.
```
**SOFTWARE**

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

; -----------------------------------------------
; ROUTINE NAME : SPI_SwMComm
; INPUT/OUTPUT : Data byte to transmit / Data byte received
; DESCRIPTION : Send-receive a data byte with fullduplex SPI Master protocol.
; COMMENTS : Data write on high clock level and read on low one.
; -----------------------------------------------

; SPI_SwMComm ;IN: A=Data / OUT: A=Data
LD X,#$08 ; X reg contains the nb+1 of bit to be transmitted.
.spi_1 RLC A ; Transfer next bit from A in the carry and load receive one.
JRC spi_2 ; Transmit data bit from Carry flag to the I/O port.
BRES SPI_MO
JRT spi_3
.spi_2 BSET SPI_MO
.spi_3 BSET SPI_CLK
BTJT SPI_MI,spi_4 ; Copy data bit from line to carry flag.
.spi_4 BRES SPI_CLK
DEC X ; One more bit transmitted.
JRNE spi_1 ; Data byte is not yet all transmit => Next bit.
RLC A
RET

; ******************************************************
; * MAIN-ROUTINES SECTION*
; ******************************************************

.main LD A,#$06 ; Configure: PC0 in Input floating mode,
LD PCDDR,A ; PC1,PC2 in Output Push-pull.
LD PCOR,A
CLR Y
.next LD A,(buff_out,Y) ; Load data byte to send from ROM output buffer.
CALL SPI_SwMComm ; Compute the full duplex SPI communication.
LD (buff_in,Y),A ; Save receive data byte in RAM input buffer.
INC Y ; Next bytes to transmit and receive.
CP Y,#$08
JRNE next
.end JRA end ; Infinity main loop.
;   **********************************************
;   * INTERRUPT SUB-ROUTINES LIBRARY SECTION *
;   **********************************************

.dummy_rt iret ; Empty subroutine. Go back to main (iret instruction).
.spi_rt iret

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
.tim0_it DC.W dummy_rt ; FFF2-FFF3h location
.spi_it DC.W spi_rt ; FFF4-FFF5h location
   DC.W not used ; FFF6-FFF7h location
.exti1_it DC.W dummy_rt ; FFF8-FFF9h location
.exti0_it DC.W dummy_rt ; FFFA-FFFFh location
.softit DC.W dummy_rt ; FFFC-FFFFh location
.reset DC.W main ; FFFE-FFFFh location

END
;*** (c) 1997 STMicroelectronics *************** END OF FILE ****
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.