AN1445
APPLICATION NOTE

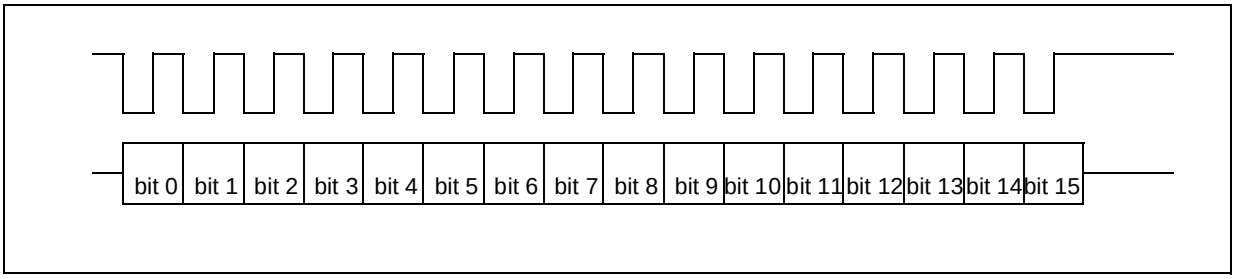
USING THE ST7 SPI TO EMULATE A 16-BIT SLAVE

By Microcontroller Division Applications

INTRODUCTION

This application note describes how to emulate a 16-bit slave SPI using an ST7 microcontroller with an on-chip 8-bit SPI.

Figure 1. 16-bit SPI frame



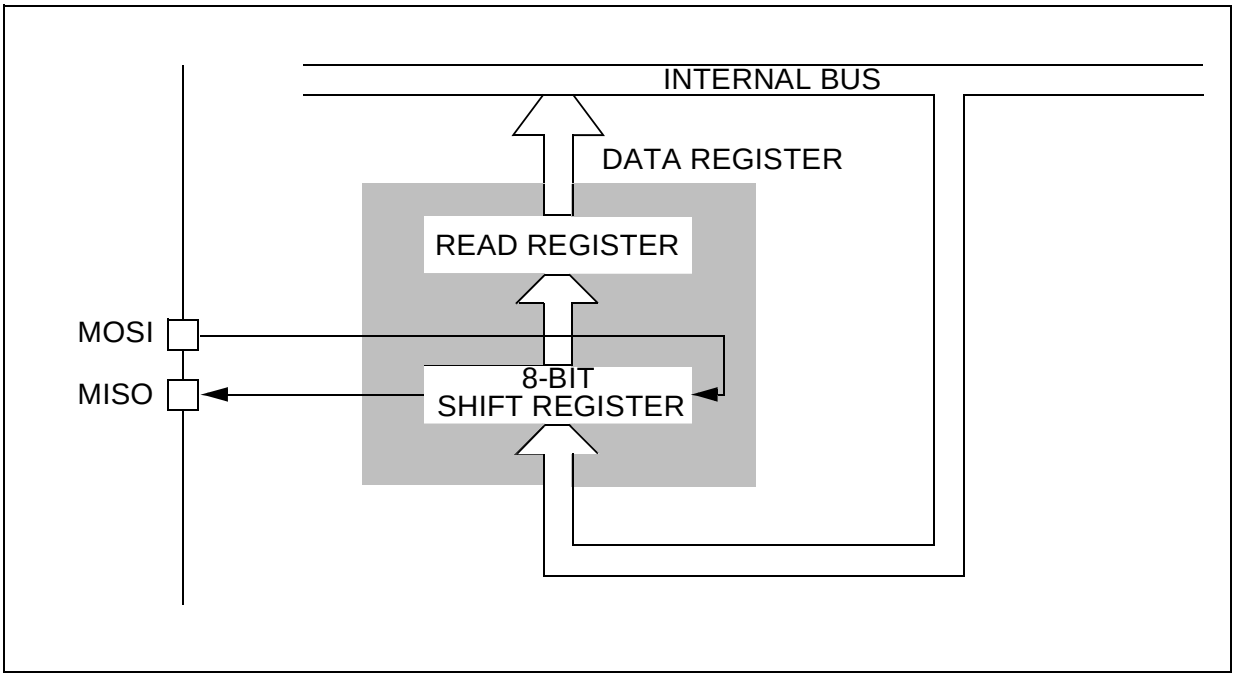
1 PRINCIPLE

The ST7 SPI cell has a double buffer for receiving data using two 8-bit registers: a read register and a shift register (see Figure 2.). The application software accesses the read register to retrieve the received data. The 8-bit shift register is managed by hardware to receive the 8 bits of each byte. As each bit is received, it is shifted into the shift register. During byte reception, the read register is not changed. It contains the previously received byte which can still be read by software. At the end of byte reception, the 8-bit shift register is copied into the read register.

This double buffering makes it possible to receive 16-bit words. At the end of reception of the first byte, the shift register is copied into the read register, the SPIF flag is set and an interrupt can be generated. The next in-coming byte will be received in the shift register while the first byte is available in the read register. In order not to lose any bits, the software must be fast enough to read the first byte before the end of the reception of the second one.

Note: The SPISR (SPI Status Register) is also called SPICSR (SPI Control/Status Register) depending on which ST7 microcontroller device you use. In this application note, we'll use the name SPISR for the status register.

Figure 2. Data Register Block diagram



2 SOFTWARE

Figure 3. C code (COSMIC C Compiler) for the interrupt routine

```
@interrupt @nostack void SPI_Interrupt(void)
{
volatile TwoBytes twobytesDR;

  if (ValBit(SPISR,SPIF)) // if a byte is received (SPIF=1) + first step to clear int flags
  {
    twobytesDR.b_form.low=SPIDR; //1st byte storage
    while(ValBit(SPISR,SPIF));
    twobytesDR.b_form.high=SPIDR; //2nd byte storage
  }
  else // then MODF flag caused the interrupt -> add your own code here
  {
    SPICR=0xC4; // second step to clear MODF flag: write access to SPICR (init value)
  }
}
```

Figure 4. Disassembled interrupt routine code

```
_SPI_Interrupt:
        btjf _SPISR,#7,L501
        ld a,_SPIDR
        ld _SPI_Interrupt$L-1,a
L311:
        btjt _SPISR,#7,L311
        ld a,_SPIDR
        ld _SPI_Interrupt$L-2,a
        iret
L501:
        ld a,#196
        ld _SPICR,a
        iret
```

where:

```
typedef unsigned char u8;   /* unsigned 8 bit type definition */
typedef signed char   s8;   /* signed 8 bit type definition */
typedef unsigned int u16;  / * unsigned 16 bit type definition */
typedef signed int s16;    /* signed 16 bit type definition */
typedef unsigned long u32; / * unsigned 32 bit type definition */
typedef signed long s32;   / * signed 32 bit type definition */

typedef union {            /* unsigned 16 bit type for 8 & 16 */
     u16 w_form;           /* bit accesses: 16> var.w_form   */
     struct {              /* 8> var.b_form.high/low     */
         u8 high, low;
     } b_form;
 } TwoBytes;
```

Note: On some devices, another flag called OVR (overrun) can also cause an SPI interrupt to occur. In this case, you will have to add some code to the interrupt routine to handle this.

3 MAXIMUM BAUD RATE

The baud rate is limited by the speed of the ST7 SPI hardware, which runs at 2 MHz maximum (for a master @ f_{CPU} = 8 MHz). We can calculate if this baud rate allows enough time for the software to read the data register before the end of the reception of the second byte.

The time needed to release the data register is 4 microseconds at 2 MBaud. The time needed by the software consists of:

- The interrupt context switching: worst case: 2,75us

  Firstly, the current instruction must finish: worst case: multiply instruction: 12 cycles = 1,5us @ 8 MHz internal. Secondly the hardware needs 10 cycles = 1,25µs @ 8 MHz internal frequency to save the context before jumping to the interrupt routine.

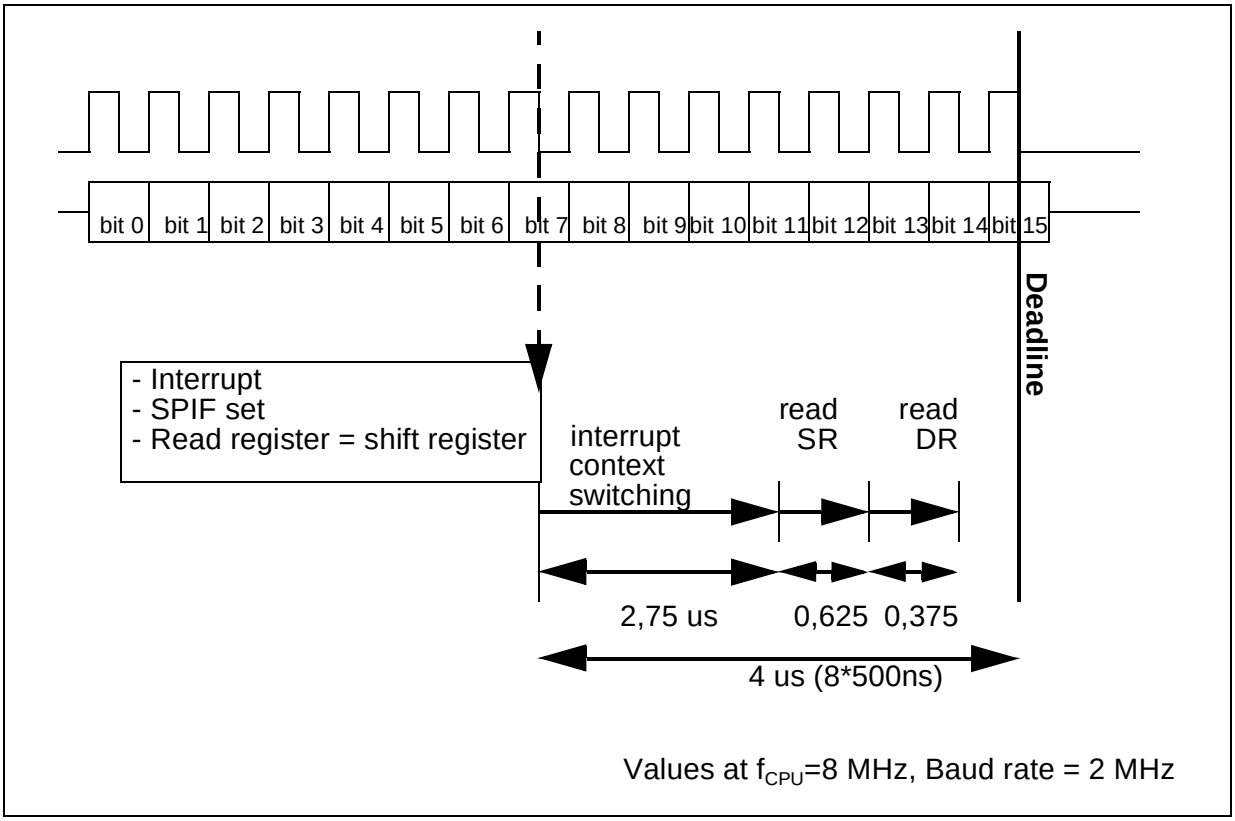
- A read access to the status register: 0,625µs

  btjf _SPISR,#7,L501 takes 5 cycles = 0,625µs

- A read access to the data register: 0,375µs

  ld a,_SPIDR takes 3 cycles = 0,375µs

Figure 5. Data Reception Timing Analysis



The time needed by the software is 30 cycles = 3,75us < 4us.

In the case of concurrent interrupts, the SPI interrupt must always be the first served. The only way to ensure this happens is to have only the SPI interrupt enabled. Otherwise, if another interrupt is served between two received words, an overrun condition will occur.

In the case of nested interrupts, the SPI interrupt only needs to have the highest priority, to be always immediately served.

CONCLUSION: The maximum baud rate, which can be reached is 2 MBaud.

4 MEASUREMENTS

The goal of the measurement is to verify the above theory and to see whether the software correctly receives both bytes. What is especially important is the time needed to read the data register, which releases the shift register for the reception of the 2nd byte. To measure this time an I/O port pin is used and cleared after reading the data register. The interrupt routine is therefore slightly modified for the measurement (Figure 6.). The timing conditions are: SPI baud rate = 2 Mbaud and f_{CPU} = 8 MHz.

Figure 6. Measurement Software

C code

```
if (ValBit(SPISR,SPIF))
{
  twobytesDR.b_form.low=SPIDR;
PBDR=0x00;
  while(!ValBit(SPISR,SPIF));
  twobytesDR.b_form.high=SPIDR;
  PBDR=0xFF;
}
else
{
  SPICR=0xC4;
}
```

Assembly code

```
_SPI_Interrupt:
   btjf _SPISR,#7,L522
   ld a,_SPIDR
   ld _SPI_Interrupt$L-1,a
   clr _PBDR
L332:
   btjf _SPISR,#7,L332
   ld a,_SPIDR
   ld _SPI_Interrupt$L-2,a
   ld a,#255
   ld _PBDR,a
   iret
L522:
   ld a,#196
   ld_SPICR,a
   iret
```

The time to be measured is the time taken by the “LD a,SPIDR” instruction (Ax in Figure 7.). The point we can measure is the transition when the port pin goes low (Bx in Figure 7.). The time between both is the time needed by “ld _SPI_Interrupt$L-1,a“and “clr PBDR”, which is 4cy+5cy=1,125us. So to Ax is 1,125us before Bx.

The measurement shows that Ax is always before the deadline. This proves that the 8-bit SPI can be used to receive 16-bit SPI messages at speeds up to 2MBaud.

Figure 7. Read time measurement