FUJITSU

Application Note

Notes on using Serial IO driven by EI²OS

Vers. 1.0 by Holger Lösche

Introduction

If EI²OS (Extended Intelligent IO Service) and SIO (Serial IO) are used together, several points have to be taken in consideration:

- When does start the SIO transfer?
- When and how is the SIO interrupt request reset?
- When and how does the EI²OS reset the SIO interrupt request?

Following explanation supposes that the global interrupt enable bit I is set and the interrupt control register of SIO has set an interrupt priority higher than the currently set priority in ILM. I and ILM belong to PS (processor state register). “Higher” priority means lower number.

Serial IO provides two modes. The first one starts every transmission by setting the STRT bit of the SMCS register. The second one starts by writing/reading a data byte to/from SDR (Serial Data Register). The second mode has to be set, if EI²OS is used. The EI²OS does not use any “trigger” bit. It just writes bytes/words from any address to any address. Therefore, only the mode that starts by writing/reading data can be used.

If the last bit has been sent/received by SIO, a interrupt is requested. If the SIE bit (Serial Interrupt enable) in the SMCS register is set, either the Interrupt handler or the EI²OS is executed. This depends on the ISE (I²OS enable) bit of the corresponding interrupt control register. This bit has to be set before starting transfer and automatically reset by EI²OS, when the EI²OS counter (DCT) has reached zero.

The SIO interrupt request is cleared by writing/reading SDR. Clearing the interrupt request is independent of the SIE register. That means the request can be cleared no matter if the EI²OS or the interrupt handler have been executed.

If the EI²OS is executed after completion the transmission of one byte, it automatically writes/reads the next byte to/from SDR. Due to this the interrupt request is cleared and the next transmission is started.

An interrupt or another EI²OS cannot interrupt the EI²OS. However, the start of EI²OS can be delayed due to a currently executed EI²OS or interrupt handler of same or higher priority. Therefore, a transmission request (for more than one byte) might result in slightly separated bytes.

Configuration

There are several ways to configure SIO and EI²OS to get it working. They can effect different performance and side effects.

The best way might be to initialise SIO and EI²OS providing a pending interrupt. If data have to be transferred, the EI²OS descriptor is updated and finally the interrupts for SIO are enabled. That causes the EI²OS immediately to be started:

```
extern volatile
union {
    struct {
        unsigned int  BAP : 16;
        unsigned char BAPB:  8;

        unsigned char SE  :  1;
        unsigned char DIR :  1;
        unsigned char BF  :  1;
        unsigned char BW  :  1;
        unsigned char IF  :  1;
        unsigned char     :  3;

        unsigned int  IOA : 16;
        unsigned int  DCT : 16;
    } reg;
    struct {
        unsigned char BAPL;
        unsigned char BAPM;
        unsigned char BAPH;
        unsigned char ISCS;
        unsigned char IOAL;
        unsigned char IOAH;
        unsigned char DCTL;
        unsigned char DCTH;
    } byte;
} ISD0;

volatile __direct unsigned char SDRval[12] = {1,3,7,15,31,63,127};
#define NUMBYTES 3
```

This union specifies the structure of the EI²OS descripotr

This are the data bytes to transmit

```
void InitSIO(void){
        ISD0.reg.BAPB = ((unsigned long)((unsigned char __far *)SDRval))>>16;
        ISD0.reg.IOA = (unsigned int)&SDR;
        ISD0.reg.IF = 1;       /* IO address fixed */
        ISD0.reg.BW = 0;       /* transfer length is Byte */
        ISD0.reg.BF = 0;       /* buffer pointer updated */
        ISD0.reg.DIR= 1;       /* buffer -> IO */
        ISD0.reg.SE = 0;       /* no end request */

        SMCS_STOP = 1;         /* stop it */
        CDCR = 0xF8;           /* prescaler divider 8 */
        SMCS_MODE = SIO_MODE;  /* start condition */
        SMCS_BDS =  0;         /* LSB first */
        SMCS_SMD =  4;         /* 31,25 kHz */
        SMCS_SOE =  1;         /* serial data output */
        SMCS_SCOE = 1;         /* serial clock output */
        SMCS_STOP = 0;         /* normal operation */

        SMCS_STRT = 1;         /* enable, if started by data */
        while (SMCS_BUSY);     /*  START triggers trans, even in mode 1 */

        PDR4_PD40 = 1;
        DDR4_DD40 = 1;
        PDR4_PD41 = 1;
        DDR4_DD41 = 1;
}
```

Pre-configure EI²OS

configure SIO

ports to show on oscilloscope

Since STRT has to be set in order to enable transfer at all (even if MODE is „1“), this causes the transmission of one very first„dummy“ byte. If the serial clock output is enabled afterwards (instead of before), this only delays this „dummy“ byte. Therefore, set SDR to a usefull value, if necessary. The „dummy“ transmission finally requests an interrupt. Therefore, SIE should not be set otherwise SIOInt() would be called immediately. If SIE is not set, the pending interrupt request can be used to start the actual transfer later.

```
        InitSIO();              /* initialize resource
        __set_il(7);            /* set ILM to 7, allow all levels
                                /* allow all interrupt levels
        __EI();                 /* enable interrupts at all


        while (1) {
            PDR4_PD41 = 0;      /* set external chip select for oscilloscope */

            ISD0.reg.DCT = NUMBYTES;
            ISD0.reg.BAP = (unsigned int)SDRval;
            ICR11 = 5 | 0x08;           /* level 5, EI²OS mode */


            SMCS_SIE = 1;       /* enable interrupts and start */


            while (PDR4_PD41 == 0); /* wait for end (chip select = 1) */

            for (ctr = 1000; ctr; ctr--); /* just to have a gap on oscilloscope */
        }
```

This is a part of the main program, which sends three bytes periodically.

set buffer address, byte number and EI²OS

start transfer

The end of transfer of all bytes can also be polled by:

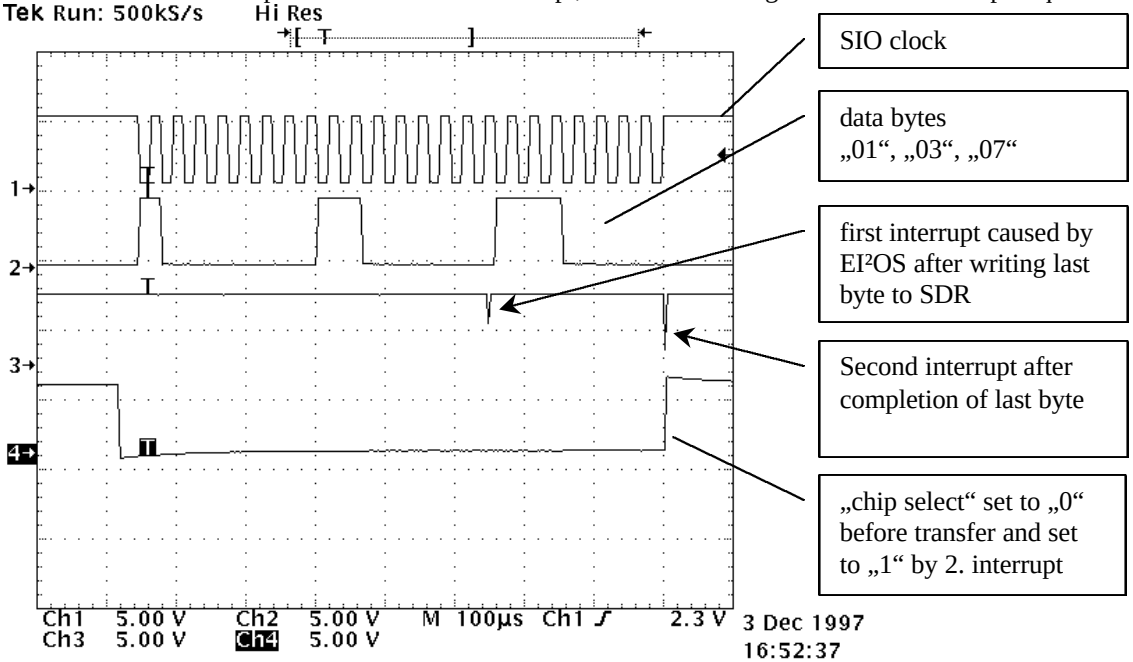
```
  if (ICR11_S == 0x01 && !SMCS_BUSY) /* EIIOS count complete and nothing on line */
```

When EI²OS has sent the last byte, it will decrement the EI²OS counter (DCT) to zero. Then, the interrupt handler is called. Since the previously sent byte is still on transmission, the SIO is still busy. After completion of the last byte the interrupt handler is called again. After completion of the last byte the handler has to disable SIO interrupts. This means that the interrupt request is still pending (it is cleared by writing/reading data to/from SDR). This pending interrupt can be used to start the next transfer by enabling SIO interrupts (address, counter and interrupt control register have to be set before).

```
__interrupt
void SIOInt(void)
{
        PDR4_PD40 = 0;          /* show interrupt on oscillos
        if (!SMCS_BUSY)
                SMCS_SIE = 0;   /* disable interrupt */
        PDR4_PD41 = 1;          /* disable "chip select" */
        PDR4_PD40 = 1;          /* show end of interrupt on o
}
```

interrupt handler called either by EI²OS or by common interrupt controller

Please note that the first interrupt is caused by EI²OS. The EI²OS counter (DCT) and the EI²OS enable flag (ISE) in the interrupt control register are cleared automatically before executing the interrupt handler the first time. The second interrupt is a common SIO interrupt, since the ISE flag was zero on interrupt request.



SIO clock

data bytes „01“, „03“, „07“

first interrupt caused by EI²OS after writing last byte to SDR

Second interrupt after completion of last byte

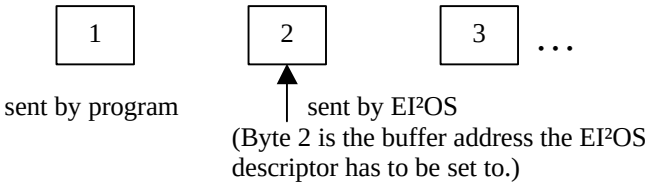
„chip select“ set to „0“ before transfer and set to „1“ by 2. interrupt

First Variation of Configuration

In the recommended configuration (first method) the buffer address of the descriptor is set to the first byte to be transmitted. The counter is set to the number of bytes to be transmitted and the transfer is initiated by enabling SIO interrupt, which forces EI²OS to send the first byte. The transfer is finished after the second interrupt. The first variation (second method) refers to the start of transfer.

The transmission can also be started by writing/reading data to/from SDR by the program itself and enabling SIO interrupts afterwards. In this case the next interrupt request occurs after completion the first byte. If ISE is enabled, EI²OS is started. It will send the byte that the EI²OS descriptor points to.

Therefore, the descriptor buffer address has to be set to the byte following the byte already sent by the program itself.



Consequently, the EI²OS counter has to be set to the number of all bytes to send minus one. Since nought must not be set (it would cause 65,536 bytes to be transferred), the minimum number of bytes is two (one sent by program and one sent by EI²OS).

In the end this method only requires more code than the recommended method and does not provide better performance.

Second Variation of Configuration

The third method refers to the end of transfer.

The program can also abort the transfer by stopping the SIO within the first (EI²OS) interrupt. However, if the interrupt is delayd by any reason (e.g. higher interrupt was currently executed), there might be some bits sent by the SIO. Remember that the first interrupt is initiated by EI²OS, when the last byte has been started to sent.

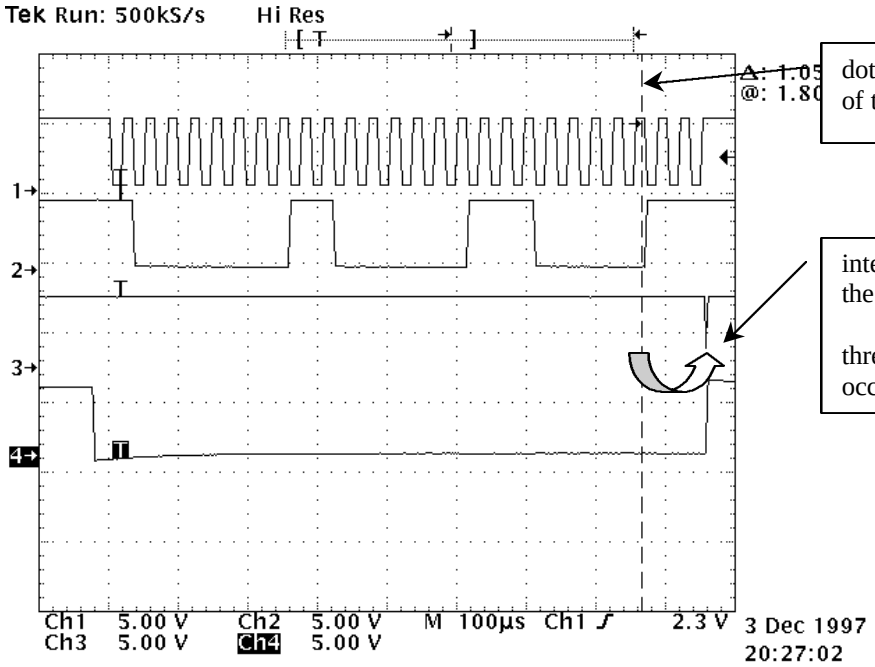
```
__interrupt
void SIOInt(void)
{
        int ctr;
        for (ctr = 100; ctr; ctr--);  /* simulate delay */

        PDR4_PD40 = 0;

        SMCS_STOP = 1;          /* stop SIO */
        SMCS_SIE = 0;           /* disable interrupt */

        PDR4_PD40 = 1;
        PDR4_PD41 = 1;
}
```

loop simulates interrupt delay




```
        while (1) {
            PDR4_PD41 = 0;     /* chip select for oscilloscope */

            ISD0.reg.DCT = NUMBYTES;
            ISD0.reg.BAP = (unsigned int)&SDRval[1];
            ICR11 = 5|0x08;           /* level 5, EIIOS */

            SMCS_STOP = 0;
            SDR = *SDRval;     /* reset last request */
            SMCS_STRT = 1;     /* start */

            SMCS_SIE = 1;      /* enable interrupts */

            while(PDR4_PD41 == 0);    /* wait for end */
            for (ctr = 1000; ctr; ctr--); /* delay next block */
        }
```

set buffer address to second byte

This three lines are necessary to enable SIO again.

It includes start condition as described in method 2.

Conclusion

Method 1 is highly recommended. It provides best performance, requires smallest code size and avoids side effects. Therefore, initialise the SIO and EI²OS only once and keep the interrupt request pending. So, the EI²OS descriptor has only to be updated and the SIO interrupt has only to be enabled in order to start the transfer of data.

Dec 1997, HL