SGS-THOMSON MICROELECTRONICS

APPLICATION NOTE

ST75C502 - BULK DELAY MANAGEMENT

1 - INTRODUCTION

The purpose of this application note is to describe the way the user must handle the interrupt reserved for Bulk delay managment.

A V.32bis modem use echo canceller technology, it substract from its received signal an estimation of its own signal echoed by the PSTN. As the transmition can have a very long delay, especialy while using satellite (up to 1.4 Second), its is mandatory to memorise all the signal that have been send during the last 1.4 second. To reduce the size of the memory needed, instead of storing the signal, we just store the symbols that were transmitted. However one can see that, if we want to handle two satellites hops, it is necessary to have a 1.4 second * 2400 symbols by second = 3360 symbols. Each symbol can be packed using a single byte, so the size of this BULK memory is 3360 bytes.

In the ST75C502, instead of adding a 4K byte inside the DSP, just to be used like a FIFO, to store and recall one symbol (Byte) each baud (0.4ms), the Bulk Delay can be implemented using the Host interface Memory. We assume that the host processor have enough memory to allocate a 4K byte inside its own Data Memory.

2 - INITIALIZATION

Prior to any operation, the user must assign the Bulk Delay (bulk_delay_line) inside its data space. The length of the bulk delay is depending of the Maximum Round Trip Delay (MAX_BULK_DELAY) that we want to handle.

```
#define MAX_BULK_DELAY 3360        /* 1.4 Second Maximum Roud Trip Delay */

unsigned char bulk_delay_line[MAX_BULK_DELAY];   /* Symbol's Storage Area */
```

Code : 1 /* "C" Global Declaration */

For further understanding we define few prototype functions :
- ST75c5x_read: read a DUAL RAM location.
- ST75c5x_write: write a DUAL RAM location.
- ST75c5x_send_cci_command: send a CCI Command to the ST75C502.

```
/* Read a DUAL RAM Location: return code is the contain of the RAM */
unsigned char ST75c5x_read(unsigned char address);

/* Write a DUALRAM Location: write data at address              */
void ST75c5x_write(unsigned char address, unsigned char data);

/* Send a CCI Command to the ST75C502                           */
unsigned char ST75c5x_send_cci_command(unsigned char opcode, unsigned char param[4]);
```

Code : 2 /* "C" Prototype Function */

An Example of implementation of ST75c5x_send_cci_command is given at the end of this application note.

The mechanism implemented inside the ST75C502 assumes that the address of the bulk_delay_line is a 16 bit word (short int) and that each byte of the bulk_delay_line are located continiously (+1). At the begining we must initialize this mechanism by giving it the two addresses **BA_ADDR** and **TO_ADDR**.

We assume that the **"C"** compiler contains two functions to convert a pointer into its physical address and an address (as a short int) into a pointer. Let define these two prototypes:

```
/* Convert a Pointer into a Physical Address */
short int PTR_ADDR( unsigned char *var);

/* Convert a Physical Address into a Pointer */
unsigned char *ADDR_PTR( short int var);
```

Code : 3 /* "C" Pointer to Interger Conversion Prototype */

After the CONF command used to select the V.32bis mode of operation, we have to initialize the bulk delay mechanism with a BULK command:

```
/* Global declaration */
#define  CCI_BULK  0x22      /* CCI Bulk Opcode */

unsigned char ST75c5x_init_bulk()

{

        /* local declaration */
        short int i, base_addr, top_addr;
        unsigned char param[4];

        /* Get Physical address for Base of Bulk_Delay_Line */
        i = PTR_ADDR(&bulk_delay_line[0]);
        /* Be sure this number is on a 8 bytes boundary */
        while ((i%8)!=0) i++;
        base_addr = i;

        /* Get Physical address for Top of Bulk_Delay_Line */
        i = PTR_ADDR(&bulk_delay_line[MAX_BULK_DELAY-1]);
        /* Be sure this number is on a 8 bytes boundary */
        while (((i+1)%8)!=0) i--;
        top_addr = i;

        /* Prepare Parameters for sending Command */
        param[0]=(unsigned char) base_addr % 256;
        param[1]=(unsigned char) base_addr > 8;
        param[2]=(unsigned char) top_addr % 256;
        param[3]=(unsigned char) top_addr > 8;

        /* Send CCI Command */
        return( ST75c5x_send_cci_command( CCI_BULK, param ));

}
```

Code : 4 /* "C" Bulk initialization */

There is no particular timming to respect between the **CONF** command, the **HSHK** command and the **BULK** command. However, to work properly, the **BULK** command must be send before the Echo canceller is started (CA-AC transition in answer mode, R1 detection in originate mode).

We can also send the **BULK** command in other mode than V.32 (or V.32 autobaud) this will not have any effect.

At that steep we must known if we want to proceed the bulk delay managment by pooling or by interrupt. As the Interrupt task is very simple we recommand the use of an interrupt; however just pooling the **SYMSTA DUAL RAM** Location will give the same results.

If we use interrupt we must enable the interrupt bit inside the **ITMASK** register, this will allow the ST75C502 to generate a signal on its **SINTR** Pin.

```
/* Enable ST75c5x bulk Interrupt */
#define  ADDR_ITMASK     0x4F   /* ST75c5x Interrupt Mask */
#define  DUAL_EN_BULK_IT 0x02   /* Enable Bulk Interrupt Bit */

void ST75c5x_enable_bulk_it()
{
      ST75c5x_write(ADDR_ITMASK,
       ST75c5x_read(ADDR_ITMASK) | DUAL_EN_BULK_IT));
}
```

Code : 5 /* "C" Enable Bulk Interrupt */

3 - MAIN TASK

Each 8 symbols (3.3ms) it is mandatory to serve the Bulk delay mechanism, otherwise an error occurs that will be signaled into the **SYSERR** bit 2 **(ERR_SYM)**.

The following routine is just the part of the interrupt mandatory to serve the Bulk Delay. Its suppose that the Interrupt **(ITSRCR)** source have been correctely decoded and that the other interrupts (Error, Command, Status, Data_Tx, Data_Rx) are well served.

```
/* Global Declaration                              */

#define  ADDR_SYMSTA         0x0F   /* Symbol Buffer Status    */
#define  ADDR_SYMADT         0x10   /* Symbol Tx Buffer Pointer */
#define  ADDR_SYMADR         0x12   /* Symbol Rx Buffer Pointer */
#define  ADDR_SYMBUF         0x14   /* Symbol Buffer           */

#define  ADDR_ITSRCR         0x50   /* Interrupt Source Byte   */
#define  DUAL_CLR_IT_BULK    0x41   /* Clear IT1               */

/* !!!! Only Part of the Interrupt !!!! */

          /* Local Declaration                    */
          short int addr;        /* Local address      */
          unsigned char *p;      /* Local Pointer      */
          unsigneg char i;       /* Local Loop Counter  */

          /* Read Interrupt Source */
          if (ST75c5x_read(ADDR_ITSRCR)&DUAL_IT_BULK)) {
             /**** The BULK Service is Required *****/

             /* Read First Address */
             addr=(short int) ST75c5x_read(ADDR_SYMADR);
             addr+=(ST75c5x_read(ADDR_SYMADR+1)<8));
             p=ADDR_PTR(addr); /* Convert into a Pointer    */

             /* Move from DUAL RAM to bulk_delay_line      */
             for (i=0;i<8;i++) ST75c5x_write (ADDR_SYMBUF+i, *p++);

             /* Clear the Bulk Interrupt Pending Bit       */
             ST75c5x_write (DUAL_CLR_IT_BULK, 0);

/* !!!! Continu processing with the other interrupts !!!! */
```

Code 6 : /* "C" Interrupt Bulk Managment */

4 - APPENDIX

```
/* Global Definition of DUAL RAM Address            */
#define ADDR_COMSYS      0x00  /* Command Word       */
#define ADDR_COMPAR      0x01  /* Parameters         */
#define ADDR_SYSERR      0x08  /* Error Status       */

/* OPTIONAL: ERROR Return Codes                     */
#define DUAL_ERR_NREADY  0x01  /* ST75c5X not Ready  */
#define DUAL_ERR_IOCD    0x02  /* Incorrect Opcode   */
#define DUAL_ERR_IPRM    0x04  /* Incorrect Parameter*/
#define CCI_ERR_MASK     0x18  /* Mask for IOCD or IPRM */
#define CCI_ERR_MASKIO   0x08  /* Mask for IOCD      */

/* Send a CCI Command to the ST75c5x                */

unsigned char ST75c5x_send_cci_command( unsigned char opcode, unsigned char param[4])
{

        unsigned char i; /* local */

        /* OPTIONAL: Test if the ST75c5x is ready to Execute a command */
        if (ST75c5x_read(ADDR_COMSYS)!=0x00) return(DUAL_ERR_NREADY);

        /* Write Parameters                         */
        for (i=0;i<=3;i++) ST75c5x_write(ADDR_COMPAR+i, param[i]);

        /* Last Write opcode to start transfer      */
        ST75c5x_write(ADDR_COMSYS, opcode);

        /* Wait until COMSYS Empty                  */
        while (ST75c5x_read(ADDR_COMSYS)!=0x00) /* wait */;

        /* OPTIONAL: Read the Error Status to check if the command was successfull */
        i = (ST75c5x_read(ADDR_SYSERR)&CCI_ERR_MASK);

        /* OPTIONAL: test if CCI Error              */
        if (i!=0) {
          if (i&CCI_ERR_MASKIO) return(DUAL_ERR_IOCD);
          else return(DUAL_ERR_IPRM);
          }

        return (0);

}
```

Code : 7 /* "C" ST75c5x_send_cci_command Example of Implementation */