How To Program The Z80 Serial I/O (SIO) and CTC

Document Version 2.1

Date: 2010-03-31

Contents

1 Terminal Mode.................................................................2
  1.1 Desired Communication Mechanism..........................2
  1.2 SIO Device Structure and external wiring...............3
    1.2.1 Wiring..............................................................3
  1.3 Programming..........................................................5
    1.3.1 Header............................................................5
    1.3.2 Interrupt Vector Table.................................5
    1.3.3 Initializing the SIO.......................................6
    1.3.4 Initializing the CTC......................................7
    1.3.5 Initializing the CPU.....................................7
    1.3.6 Hardware Flow Control..............................8
    1.3.7 Disabling SIO RX-channel..........................8
    1.3.8 Interrupt Service Routines..........................9
    1.3.9 Transmission of a character to the host........10

2 File Transfer Mode....................................................11
  2.1 Desired Communication Mechanism......................11
  2.2 Programming........................................................12
    2.2.1 Header........................................................12
    2.2.2 Interrupt Vector Table................................12
    2.2.3 Initializing the CTC....................................12
    2.2.4 Initializing the CPU....................................13
    2.2.5 X-Modem File Transfer............................13
      2.2.5.1 Host triggered transfer and setup........13
      2.2.5.2 Subroutines..........................................17

3 Z80 IC equivalents table............................................18

4 Useful Links.............................................................19

5 Further Reading........................................................19

6 Disclaimer...............................................................19
Preface

The Z80 SIO is the most powerful I/O device of the Z80 product family. The official ZiLOG-datasheet gives a good overall view of all the features of this device but lacks a tutorial like approach and programming examples in assembly language.

This document aims to make the Z80 processor system popular again since a lot of valuable literature and expertise has vanished from the public because of more sophisticated processor architectures of the present.

Remarkably ZiLOG still produces the ICs of the Z80 family – since the late seventies!

Part one of this document describes how to program the SIO so that it communicates with a PC in asynchronous terminal mode whereas part two focuses on the block transfer mode used for file transmission. Also slightly touched in this document is the CTC programming and implementation of an interrupt mechanism.

There is no special focus on hardware issues like device selection, pin characteristics or ratings. Please refer to the official ZiLOG datasheets at www.zilog.com or www.z80.info.

The code examples shown here provide by far not the best performance and robustness. Therefore I appreciate every hint or critics to improve the quality of this document.

1 Terminal Mode

1.1 Desired Communication Mechanism

We want to program the SIO for asynchronous RS232 terminal mode with these parameters:

- Baudrate: 9600 Baud/sec
- Stopbits: 1
- Startbits: 1
- Character length: 8 bit
- Parity: none

In terminal mode the host computer (in our case the PC with any terminal program like Minicom or Hyper Terminal and an RS232 interface) communicates with the client (the Z80-SIO) character based via a so called Null-Modem-Cable. The host transmits one or more characters to the client, whereupon the client echoes this character back to the host and processes it. The host displays the echoed character on its screen. If the host does not “hear” the echoed character the communication is faulty.
Special attention is to be paid to the flow control scheme which is hardware based. In general this is called “RTSCTS” or just “hardware flow control”. This method allows transmission of all 8-bit-characters (so called binary mode) and prevents overrunning of one of the peers in case one of them is to slow. In this document I assume the host PC is much faster than the client.

The wiring of the null modem cable used here has following connections between its female 9 pin D-Sub connectors:

1 – 4 / 4 -1  cross wired DTR and DCD
2 – 3 / 3 – 2  cross wired TxD and RxD
5 – 5  signal ground (GND)
7 – 8 / 8 – 7  cross wired RTC and CTS
9 – 9  ring indicator (RI, not used here)

1.2 SIO Device Structure and external wiring

Figure 1 shows the block diagram of the device with the blocks and signals needed for our example outlined in red. Figure 2 shows the data paths within the SIO. Marked in red are the blocks we need for asynchronous mode.

1.2.1 Wiring

Data and control: These are the Z80 bus signals D[7:0], A[1:0], /RD, /IOREQ, /RESET, /CE and CLK.

Interrupt Control Lines: /M1, /INT connected to CPU, IEI and IEO daisy chained to other periphery

Serial Data: TxD and RxD going towards host computer¹

Channel Clocks: TxCA and RxCA driven by CTC channel output TO0

Modem or other Controls: CTS, RTS, DTR, DCD used for hardware flow control

miscellaneous: /SYNC not used, pulled high by 10k resistor
/Wait/Ready comes out of the device. It is to be connected to the WAIT-Input of the Z80-CPU. By asserting this signal the SIO tells the CPU to wait until the SIO has completed a character transfer. For this example we do not make use of this connection.

¹ Usually these signals are not connected directly to the host but via diver devices like MAX232, 1488, 1489 or similar level converters.
Figure 1: Block Diagram

Figure 2: Data Path
1.3 Programming

Tree problems have to be solved: initializing the SIO, implementing the interrupt mechanism, echoing the received character, transmitting a character and turning on/off the SIO RX-channel in certain situations.

1.3.1 Header

The header shown below defines the hardware addresses of the data and control port of your SIO and the address of your CTC channel 0. My hardware here uses the addresses 0x4, 0x6 and 0x0.

```assembly
SIO_A_D  equ  4h
SIO_A_C  equ  6h
CH0      equ  0h
```

Text 1: header

1.3.2 Interrupt Vector Table

Every time the SIO receives a character it requests an interrupt causing the CPU to jump to the memory address specified by the term RX_CHA_AVAILABLE. Special receive conditions like receiver buffer overrun cause a jump to location SPEC_RX_CONDITION.

```assembly
INT_VEC:
    org  0Ch
    DEFW RX_CHA_AVAILABLE
    org  0Eh
    DEFW SPEC_RX_CONDITION
```

Text 2: SIO interrupt vector table
1.3.3 Initializing the SIO

First we have to configure the SIO using the sequence shown in Text 3. For detailed information on the purpose of certain registers and control bits please read the SIO datasheet. We operate the SIO in interrupt mode “interrupt on all received characters”.

```
SIO_A_RESET:
    ;set up TX and RX:
    ld a,00110000b ;write into WR0: error reset, select WR0
    out (SIO_A_C),A

    ld a,018h ;write into WR0: channel reset
    out (SIO_A_C),A

    ld a,004h ;write into WR0: select WR4
    out (SIO_A_C),A

    ld a,44h ;44h write into WR4: clkx16, 1 stop bit, no parity
    out (SIO_A_C),A

    ld a,005h ;write into WR0: select WR5
    out (SIO_A_C),A

    ld a,0E8h ;DTR active, TX 8bit, BREAK off, TX on, RTS inactive
    out (SIO_A_C),A

    ld a,01h ;write into WR0: select WR1
    out (SIO_B_C),A

    ld a,00000100b ;no interrupt in CH B, special RX condition affects vect
    out (SIO_B_C),A

    ld a,02h ;write into WR0: select WR2
    out (SIO_B_C),A

    ld a,0h ;write into WR2: cmd line int vect (see int vec table)
    ;bits D3,D2,D1 are changed according to RX condition
    out (SIO_B_C),A

    ld a,01h ;write into WR0: select WR1
    out (SIO_A_C),A

    ld a,00110000b ;interrupt on all RX characters, parity is not a spec RX condition
    ;buffer overrun is a spec RX condition
    out (SIO_A_C),A

SIO_A_EI:
    ;enable SIO channel A RX
    ld a,003h ;write into WR0: select WR3
    out (SIO_A_C),A

    ld a,0C1h ;RX 8bit, auto enable off, RX on
    out (SIO_A_C),A

    ;Channel A RX active
    RET
```

Text 3: configure the SIO
1.3.4 Initializing the CTC

The CTC channel 0 provides the receive and transmit clock for the SIO.

```
INL_CTC:

;init CH0
;CH0 provides SIO A RX/TX clock
ld A,00000111b ; int off, timer on, prescaler=16, don't care ext. TRG edge,
; start timer on loading constant, time constant follows
; sw-rst active, this is a ctrl cmd
out (CH0),A
ld A,2h ; time constant defined
out (CH0),A ; and loaded into channel 0

; TO0 outputs frequency=CLK/2/16/(time constant)/2
; which results in 9600 bits per sec
```

Text 4: configuring the CTC channel 0

1.3.5 Initializing the CPU

The CPU is to run in interrupt mode 2. See Text 5 below. This has to be done after initializing SIO and CTC.

```
INT_INI:

ld A,0
ld I,A ; load I reg with zero
im 2 ; set int mode 2
ei ; enable interrupt
```

Text 5: set up the CPU interrupt mode 2
### 1.3.6 Hardware Flow Control

In order to signal the host whether the client is ready or not to receive a character the RTS line coming out of the client (and driving towards the host) needs to be switched. As earlier said I assume the host is much faster than the client, that why I do not implement a routine to check the CTS-line coming from the host.

```assembly
A_RTS_OFF:
    ld a,005h ;write into WR0: select WR5
    out (SIO_A_C),A
    ld a,0E8h ;DTR active, TX 8bit, BREAK off, TX on, RTS inactive
    out (SIO_A_C),A
    ret

A_RTS_ON:
    ld a,005h ;write into WR0: select WR5
    out (SIO_A_C),A
    ld a,0EAh ;DTR active, TX 8bit, BREAK off, TX on, RTS active
    out (SIO_A_C),A
    ret
```

**Text 6: signaling the host go or nogo for reception**

### 1.3.7 Disabling SIO RX-channel

When certain conditions arise it might by important to disable the receive channel of the SIO (see routine in Text 7).

```assembly
SIO_A_DI:
    ;disable SIO channel A RX
    ld a,003h ;write into WR0: select WR3
    out (SIO_A_C),A
    ld a,0C0h ;RX 8bit, auto enable off, RX off
    out (SIO_A_C),A
    ;Channel A RX inactive
    ret
```

**Text 7: Disabling the SIO**
1.3.8  Interrupt Service Routines

Upon reception of a character the routine RX_CHA_AVAILABLE shown in Text 8 is executed. Here you get the character set by the host. In the end of this routine, the character is sent back to the host by a simple single command

```
out (SIO_A_D),A
```

**Note:** In this example we backup only register AF. Depending on your application you might be required to backup more registers like HL, DE, CD, ...

Routine SPEC_RX_CONDITION is executed upon a special receive condition like buffer overrun. In my example the CPU is to jump at the warmstart location 0x0000.

```
RX_CHA_AVAILABLE:
push AF ;backup AF
call A_RTS_OFF
in A,(SIO_A_D) ;read RX character into A

; A holds received character
;do something with the received character

;echo character to host
out (SIO_A_D),A

call TX_EMP

ei ;see comments below

call A_RTS_ON ;see comments below

pop AF ;restore AF
Reti

SPEC_RX_CONDITION:

jp 0000h
```

**Text 8: character received routine**

**Note:** The code written in red might be required if you want the CPU to be ready for another interrupt (ei) and to give the host a go for another transmission (call A_RTS_ON).

I recommend to put these two lines not here but in your main program routine that processes the characters received by the SIO. This way you process one character after another and avoid overrunning your SIO RX buffer.
1.3.9 Transmission of a character to the host

In general transmitting of a character is done by the single command 
out (SIO_A_D),A

as written in Text 8. To make sure the character has been sent completely the transmit buffer needs to be checked if it is empty. The general routine to achieve this is shown in Text 9.

<table>
<thead>
<tr>
<th>TX_EMP:</th>
</tr>
</thead>
<tbody>
<tr>
<td>; check for TX buffer empty</td>
</tr>
<tr>
<td>sub a</td>
</tr>
<tr>
<td>inc a</td>
</tr>
<tr>
<td>out (SIO_A_C),A</td>
</tr>
<tr>
<td>in A,(SIO_A_C)</td>
</tr>
<tr>
<td>bit 0,A</td>
</tr>
<tr>
<td>jp z,TX_EMP</td>
</tr>
<tr>
<td>ret</td>
</tr>
</tbody>
</table>

**Text 9: transmitting a character to host**
2 File Transfer Mode

2.1 Desired Communication Mechanism

We want to program the SIO for asynchronous RS232 X-Modem protocol with these parameters:

Baudrate: 9600 Baud/sec
Stopbits: 1
Startbits: 1
Character length: 8 bit
Parity: none

In difference to the character based mode described in part 1 (Terminal Mode) blocks of 128 byte size are to be transferred over the Null-Modem-Cable from the host PC to the client, the Z80-machine. I choose the X-Modem protocol due to its robustness and easy feasibility. Typical terminal programs like HyperTerminal, Kermit or Minicom do support the X-Modem protocol.

Of course you can also transfer a file via character based mode but the transfer will take much more time.

Regarding the device structure, Null-Modem-Cable, wiring and flow-control please refer to section 1.1 on page 2 and 1.2 on page 3.

A web link to the description of the X-Modem protocol can be found in section 4 on page 19.

Note: For this mode the connection of the CPU pin /WAIT and the SIO pin /Wait/Ready is required. Please see section 1.2.1 on page 3.
2.2 Programming

Four problems have to be solved: initializing the SIO, implementing the interrupt mechanism, requesting the host to start the X-Modem transfer and load the file to a certain RAM location.

2.2.1 Header

The header shown below defines the hardware addresses of the data and control port of your SIO and the address of your CTC channel 0. My hardware here uses the addresses 0x4, 0x6 and 0x0. Further on there is a RAM location defined for counting bad blocks while the file is being transferred.

```
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIO_A_D</td>
<td>4h</td>
</tr>
<tr>
<td>SIO_A_C</td>
<td>6h</td>
</tr>
<tr>
<td>CH0</td>
<td>0h</td>
</tr>
<tr>
<td>temp0</td>
<td>1015h</td>
</tr>
</tbody>
</table>
```

: holds number of unsuccessful block transfers/block during download

Text 10: header

2.2.2 Interrupt Vector Table

Every time the SIO receives the first byte of a block it requests an interrupt causing the CPU to jump to the memory address specified by the term BYTEAVAILABLE. This is the interrupt mode: interrupt on first character. Special receive conditions like receiver buffer overrun cause a jump to location SPEC_BYTE_COND. The latter case aborts the transfer.

```
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_VEC:</td>
<td></td>
</tr>
<tr>
<td>org</td>
<td>1Ch</td>
</tr>
<tr>
<td>DEFW</td>
<td>BYTEAVAILABLE</td>
</tr>
<tr>
<td>org</td>
<td>1Eh</td>
</tr>
<tr>
<td>DEFW</td>
<td>SPEC_BYTE_COND</td>
</tr>
</tbody>
</table>
```

Text 11: SIO interrupt vector table

2.2.3 Initializing the CTC

Please read section 1.3.4 on page 7.
2.2.4 Initializing the CPU

Please read section 1.3.5 on page 7.

2.2.5 X-Modem File Transfer

The assembly code of this module is described in the following sections. Due to its complexity I split it into parts shown in Text 12, 13 and 14 whose succession must not be mixed. For detailed information on the purpose of certain registers and control bits please read the SIO datasheet.

2.2.5.1 Host triggered transfer and setup

The host PC initiates the transfer. Using Minicom for example you press CTRL-A-S to get into a menu where you select the x-modem protocol and afterward into the file menu to select the file to be sent to the client. The procedure is similar with HyperTerminal.

After that the host waits for a NAK character sent by the client.

Now you should run the code shown below in Text 12 on your Z80-machine. This code initializes the SIO for interrupt mode “interrupt on first received character”.

```
; set up TX and RX:
ld  a,018h ; write into WR0: channel reset
out  (SIO_A_C),A

ld  a,004h ; write into WR0: select WR4
out  (SIO_A_C),A

ld  a,44h ; 44h write into WR4: clkx16,1 stop bit, no parity
out  (SIO_A_C),A

ld  a,005h ; write into WR0: select WR5
out  (SIO_A_C),A

ld  a,0E8h ; DTR active, TX 8bit, BREAK off, TX on, RTS inactive
out  (SIO_A_C),A

ld  a,01h ; write into WR0: select WR1
out  (SIO_B_C),A

ld  a,00000100b ; no interrupt in CH B, special RX condition affects vect
out  (SIO_B_C),A

ld  a,02h ; write into WR0: select WR2
out  (SIO_B_C),A

ld  a,10h ; write into WR2: cmd line int vect (see int vec table)
out  (SIO_B_C),A

Text 12: setup 1
```
Now we do some settings for bad block counting, the first block number to expect and the RAM destination address of the file to receive from the host. See Text 13. The destination address setting is red colored. From this RAM location onwards the file is to be stored. In my example I use address 0x8000. Depending on your application you should change this value.

Text 13: setup 2

```
sub A
ld (temp0),A ; reset bad blocks counter
ld C,1h ; C holds first block nr to expect
ld HL,8000h ; set lower destination address of file

call SIO_A_EI
call A_RTS_ON

call TX_NAK ; NAK indicates ready for transmission to host
```

Text 14 shows the code section that prepares the CPU for the reception of the first byte of a data block. The line colored red makes the CPU waiting for an interrupt which is caused by the SIO. The belonging interrupt service routine is shown in Text 15.

Once a block has been received, the checksum is verified and possible bad blocks counted. The same data block is transferred maximal 10 times whereupon the transfer is aborted.
TEXT 14: Receive Data Block

REC_BLOCK:

; set block transfer mode
ld a,21h ; write into WR0 cmd4 and select WR1
out (SIO_A_C),A
ld a,10101000b ; wait active, interrupt on first RX character
out (SIO_A_C),A ; buffer overrun is a spec RX condition

ei
call A_RTS_ON
halt ; await first rx char
call A_RTS_OFF

ld a,01h ; write into WR0: select WR1
out (SIO_A_C),A
ld a,00101000b ; wait function inactive
out (SIO_A_C),A

; check return code of block reception (e holds return code)
ld a,e
cp 0 ; block finished, no error
jp z,l_210

cp 2 ; eot found
jp z,l_211

cp 3 ; chk sum error
jp z,l_613

ld a,010h
jp l_612

l_210:
call TX_ACK ; when no error
inc c ; prepare next block to receive
sub A
ld (temp0),A ; clear bad block counter
jp REC_BLOCK

l_211:
call TX_ACK ; on eot
ld A,01h
jp l_612

l_613:
call TX_NAK ; on chk sum error
scf
ccf ; clear carry flag
ld DE,0080h ; subtract 80h
sbc HL,DE ; from HL, so HL is reset to block start address

ld A,(temp0) ; count bad blocks in temp0
inc A
ld (temp0),A
op 09h
jp z,l_612 ; abort download after 9 attempts to transfer a block
jp REC_BLOCK ; repeat block reception

l_612:
DLD_END:

ret
BYTE_AVAILABLE:

EXP_SOH_EOT:
  _l_205:  in A, (SIO_A_D) ; read RX byte into A
            cp 01h ; check for SOH
            jp z, _l_205 ; check for EOT
            cp 04h
            jp nz, _l_205
            ld e, 2h
            reti

; await block number

EXP_BLK_NR:
  _l_205:  in A, (SIO_A_D) ; read RX byte into A
            cp C ; check for match of block nr
            jp nz, _l_205
            cp 01h ; check for SOH
            jp z, _l_205
            cp 04h ; check for EOT
            jp nz, _l_2020
            ld e, 2h
            reti

; await complement of block number

EXP_CPL_BLK_NR:
  _l_205:  in A, (SIO_A_D) ; read RX byte into A
            cp E ; check for cpl of block nr
            jp nz, _l_205
            cp 01h ; check for SOH
            jp z, _l_205
            cp 04h ; check for EOT
            jp nz, _l_2020
            ld e, 2h
            reti

; await data block

EXP_DATA:
  _l_205:  in A, (SIO_A_D) ; read RX byte into A
            ld (HL), A
            add A, D
            ld D, A
            inc HL
            djnz EXP_DATA ; loop until block finished

EXP_CHK_SUM:
  _l_205:  in A, (SIO_A_D) ; read RX byte into A
            cp a, 045h ; for debug only
            jp z, _l_2020
            ld e, 3h
            reti

; return when block received completely

;-------------------Int routine on RX overflow-----------------------

SPEC_BYTE_COND:

; in case of RX overflow prepare abort of transfer

Text 15: Interrupt Service Routine
2.2.5.2 Subroutines

Important for the X-Modem protocol is the sending of the Acknowledge and the Not-Acknowledge character to the host machine. For all other routines used in the code above please refer to sections 1.3.3 on page 6 and 1.3.6 on page 8.

```
TX_NAK:
  ld a,15h  ;send NAK 15h to host
  out (SIO_A_D),A
  call TX_EMP
  RET

TX_ACK:
  ld a,6h  ;send AK to host
  out (SIO_A_D),A
  call TX_EMP
  RET
```

Text 16: Acknowledge / Not-Acknowledge
3 Z80 IC equivalents table

An overview of ICs of the famous Z80 family gives Table 1.

<table>
<thead>
<tr>
<th>device</th>
<th>equivalent type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z80-CPU</td>
<td>BU18400A-PS (ROHM)</td>
</tr>
<tr>
<td></td>
<td>D780C-1 (NEC)</td>
</tr>
<tr>
<td></td>
<td>KP1858BM1/2/3 / KR1858BM1/2/3 (USSR)</td>
</tr>
<tr>
<td></td>
<td>LH0080 (Sharp)</td>
</tr>
<tr>
<td></td>
<td>MK3880x (Mostek)</td>
</tr>
<tr>
<td></td>
<td>T34VM1 / T34BM1 (USSR)</td>
</tr>
<tr>
<td></td>
<td>TMPZ84C00AP-8 (Toshiba)</td>
</tr>
<tr>
<td></td>
<td>UA880 / UB880 / VB880D (MME)</td>
</tr>
<tr>
<td></td>
<td>Z0840004 (ZiLOG)</td>
</tr>
<tr>
<td></td>
<td>Z0840006 (ZiLOG)</td>
</tr>
<tr>
<td></td>
<td>Z80ACPUD1 (SGS-Ates)</td>
</tr>
<tr>
<td></td>
<td>Z84C00AB6 (SGS-Thomson)</td>
</tr>
<tr>
<td></td>
<td>Z84C00 (ZiLOG)</td>
</tr>
<tr>
<td></td>
<td>Z8400A (Goldstar)</td>
</tr>
<tr>
<td></td>
<td>U84C00 (MME)</td>
</tr>
<tr>
<td>Z80-SIO</td>
<td>UA8560, UB8560 (MME)</td>
</tr>
<tr>
<td></td>
<td>Z0844004 (ZiLOG)</td>
</tr>
<tr>
<td></td>
<td>Z8440AB1 (ST)</td>
</tr>
<tr>
<td></td>
<td>Z0844006 (ZiLOG)</td>
</tr>
<tr>
<td></td>
<td>Z84C40 (ZiLOG)</td>
</tr>
<tr>
<td></td>
<td>U84C40 (MME)</td>
</tr>
<tr>
<td>Z80-PIO</td>
<td>Z0842004/6 (ZiLOG)</td>
</tr>
<tr>
<td></td>
<td>UA855 / UB855 (MME)</td>
</tr>
<tr>
<td></td>
<td>Z84C20 (ZiLOG)</td>
</tr>
<tr>
<td></td>
<td>U84C20 (MME)</td>
</tr>
<tr>
<td>Z80-CTC</td>
<td>Z84C30 (ZiLOG)</td>
</tr>
<tr>
<td></td>
<td>U84C30 (MME)</td>
</tr>
<tr>
<td></td>
<td>UA857 / UB857 (MME)</td>
</tr>
</tbody>
</table>

Table 1: Z80 equivalents
4 Useful Links

- A complete embedded Z80 system can be found at http://www.train-z.de/train-z
- The Free and Open Productivity Suite OpenOffice at http://www.openoffice.org
- Z80 assembler for Linux and UNIX at http://www.unix4fun.org/z80pack/
- The powerful communication tool Kermit at http://www.columbia.edu/kermit/
- The Z80 history at http://en.wikipedia.org/wiki/Zilog_Z80

- EAGLE - an affordable and very efficient schematics and layout tool at http://www.cadsoftusa.com/

5 Further Reading

I recommend to read these books:

“Using C-Kermit” / Frank da Cruz, Christine M. Gianone / ISBN 1-55558-108-0 (english)
“C-Kermit : Einführung und Referenz” / Frank da Cruz, Christine M. Gianone / ISBN 3-88229-023-4 (german)

6 Disclaimer

This tutorial is believed to be accurate and reliable. I do not assume responsibility for any errors which may appear in this document. I reserve the right to change it at any time without notice, and do not make any commitment to update the information contained herein.

---------------

My Boss is a Jewish Carpenter