Application Note

Software UART for the Z86E02

AN004102-0502

ZiLOG WORLDWIDE HEADQUARTERS • 910 E. HAMILTON AVENUE • CAMPBELL, CA 95008
TELEPHONE: 408.558.8500 • FAX: 408.558.8300 • www.ZiLOG.com

This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Headquarters
910 E. Hamilton Avenue
Campbell, CA 95008
Telephone: 408.558.8500
Fax: 408.558.8300
www.ZiLOG.com

Windows is a registered trademark of Microsoft Corporation.

Information Integrity

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZiLOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZiLOG Sales Office to obtain necessary license agreements.

Document Disclaimer

Table of Contents

Acknowledgments

Project Lead Engineer

Mathias Loehr

Application and Support Engineer

Mathias Loehr

System and Code Development

Mathias Loehr

Software UART for the Z86E02

General Overview

This Application Note describes how to implement a software-emulated universal asynchronous receiver transmitter (UART) function on the Z86 family of low-cost 8-bit microcontrollers. This particular UART function is half-duplex, event-driven, and supports an 8-N-1 protocol using an RS-232 interface.

Optionally, a ninth data bit can be enabled. Baud rates from 1200 to 57600 are supported. The software features full initialization and a basic application for both a receiver and a transmitter. The primary goals for this software UART are speed and reliability.

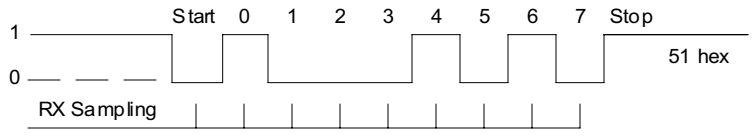
Discussion

Theory of Operation

The UART protocol is based on the EIA RS-232C standard, published in 1969. That standard was popular with the introduction of personal computers and it is one of the most commonly used serial interfaces. Originally defined as a 25-pin interface with several modem handshake and control signals, the basic interface requires only three lines, Receive (RX), Transmit (TX), and GND. In this constellation, a handshake is executed in software by transmitting special XON and XOFF characters, or by using a ninth data bit as a separator between command and data. In most MCU applications, the half-duplex communication is sufficient, meaning that each side is either a receiver or a transmitter at any given time.

While the transmission speed of the RS-232C cable was formerly limited to 19200 baud for a cable length of 50 feet, advanced specifications and line drivers allow much higher baud rates today. The underlying UART protocol, however, is still the same. Figure 1 illustrates that protocol.

Figure 1. Basic 8-bit UART Protocol



The TX idle state of the UART is High. A High-to-Low transition of the Start bit initiates the transmission. Eight or nine data bits follow before the Stop bit pulls High again. In the case of 19200 baud, each bit time is approximately 52 µs. Asynchronous operation means that the clock is not transmitted. The receiver must operate with the same baud rate, usually derived from a local oscillator. It synchronizes on the falling Start edge and samples the incoming data in their bit middle.

Note: The actual signal levels on a PC serial connector are inverted by line drivers on both sides and provide symmetrical levels of around ±12V.

Results of Operation

The Software UART supports the basic format 8-N-1. That format is 8 data bits, no parity and 1 stop bit. Optionally a 9-bit mode can be enabled to receive or transmit nine data bits. The ninth bit can be used by software as a separator between command or address and data. The achieved baud rates depend on RX or TX mode and the clock frequency, as shown in Table 1.

Table 1. Tested Baud Rates for RX and TX Modes vs. Clock Frequencies

RX-4MHz	1200	2400	4800	9600			
RX-8MHz	1200	2400	4800	9600	19200		
RX-12MHZ	1200	2400	4800	9600	19200	38400	
RX-16MHz		2400	4800	9600	19200	38400	
TX-4MHz	1200	2400	4800	9600	19200		
TX-8MHz	1200	2400	4800	9600	19200	38400	
TX-12MHz	1200	2400	4800	9600	19200	38400	57600
TX-16MHz		2400	4800	9600	19200	38400	57600

Communication is half-duplex. In RX mode, the program waits to receive characters and stores them in a 16-byte RAM buffer. In TX mode, the program continuously sends out an ASCII message string, stored in code memory.

Options

Several options at assembly or runtime must be selected to adapt the program to the desired operation. Table 2 contains a list of those options.

Table 2. Options at Assembly and Run Time

Name	Description	When	Where
XTAL	Oscillator frequency/1000 in standard mode. Range 4-16 MHz, odd values allowed. Default is 8000 (8MHz).	Assembly time	CONSTANTS DEF.
BAUD	Baud rate: ONE_TWO to FIFTY_SEVEN valid settings depend on mode and frequency shown in Table 1. When XTAL and BAUD are specified, the program selects all the appropriate timings. Default is NINETEEN_TWO.	Assembly time	CONSTANTS_DEF
RAM_TOP	Top of RAM for MCU other than Z86E02. Default is 3F.	Assembly time	CONSTANTS_DEF
ROM_TOP	Top of ROM for MCU other than Z86E02. Default is 01FF.	Assembly time	CONSTANTS_DEF
MODE	RX or TX	Run time	After code label *start*
NINE_BIT_MODE	ON or OFF	Run time	After code label *start*
WAIT_STOP	ON or OFF. Wait for Stop bit, verify it, and set FRAME_ERR accordingly.	Assembly time	CONDITIONAL ASSEMBLY SWITCH

Further, there are device options to be selected when the One-Time Programmable (OTP) is programmed or a ROM code is transmitted for mask production. The selection is given in the header of the listing.

Initialization

Out of Reset, the program sets P0 to output mode and outputs a High level at P00, which is the TX pin. P2 is set to push-pull output for optional data display. A RAM validation routine checks if three special patterns are still valid. If yes, the following RAM initialization is skipped and all RAM buffers and variables are kept. This routine stores valid information, in the case of spurious undervoltage resets, if V_{CC} goes below V_{BO} but not below the RAM retention voltage.

The Stack Pointer is set to the top of RAM, the timers are disabled, and T1 is loaded with BAUD/PBD for the bit time delay. BAUD incorporates the clock frequency and PBD is the T1 prescaler value.These values are computed during assembly time, if BAUD exceeds the value of 255. Finally, the RX interrupt for P31

is enabled and the program enters the RX or TX application, depending on the setting of MODE.

Transmit Mode

The transmission is realized in the subroutine `serial_out`. Before calling the subroutine, the byte to be transmitted must be loaded into DATA. The Technical Support section contains the flowchart for the transmit routine

The routine first saves present interrupts and then enables a Timer 1 interrupt, thereby ensuring that the RX interrupt and other user interrupts are disabled during transmission. The transmit routine assembles the complete bit sequence of Start, Data, and Stop into a 16-bit shift register, made up by DATA and DATA2. This assembly allows the fastest baud rates, because the 9-bit mode decision is kept outside the transmission loop. After bit assembly, the bit counter is set and T1 is started for bit-time delay. The bit is shifted via Carry into a P0 shadow register and the program enters HALT mode to wait for the T1 interrupt. T1 service is just an IRET instruction. After return, the bit is output at P00. This method allows a precise timing without any jitter. After the Stop bit is output, the routine restores the interrupts and returns to main.

The main application in TX mode reads an ASCII message string, stored in code memory at label `tx_rom`. The message string is terminated with Carriage Return and Line Feed for display on a PC monitor. The TX application runs in a loop.

Receive Mode

Because the receive event is asynchronous in nature, the Port interrupt pin P31 is used for RX. The appropriate port interrupt service routine (ISR) is diagrammed in the Technical Support section. The ISR first waits for nearly half a bit time. This delay is realized with DJNZ instructions and there is a short-cut for the highest baud rate at a given clock frequency. The RX pin is read to validate the Start bit. If RX is not zero, the program exits after clearing any spurious P31 interrupt requests. If the Start bit is valid, T1 is started with bit-time delay and the bit counter is set to 8. The program polls the T1 IRQ bit for speed reasons. After timeout, RX pin is sampled and shifted into DATA until the complete byte is received. If 9-bit mode is enabled the program waits for another T1 timeout, reads the RX pin and sets NINTH accordingly. The receive character available flag RX_AVAIL is set to `1`, T1 is stopped, and the program returns to main after clearing spurious P31 interrupt requests. User interrupts are inherently disabled during P31 receive service.

The software must continuously check the RX_AVAIL flag to save a received byte or react on it, before it is overwritten by the next reception. To allow at least a full bit time for check loop plus reaction, the receive routine does not wait for the Stop bit and returns early in the second half of the most recent data bit. This fact must be considered for handshakes and data flow control. For lower baud rates, the

early return might not be required, and the user may want to wait for the Stop bit, verify it, and set the flag FRAME_ERR accordingly. This option is provided by the conditional assembly switch WAIT_STOP, as described in Table 2.

The RX application stores received bytes in a 16-byte RAM buffer at RX_BUF. When the buffer is full, it is overwritten from the beginning. Additionally, each received byte is output at P2.

Hardware

The software UART can be built using the schematics in the Technical Support section.The schematic shows a single +5V supply setup, where the Z8 is connected by a MAX-232A line driver to a SUBD-9 serial connector. Only RX and TX lines are used. The circuit can be connected to a PC, running a terminal program with setting *no handshake*.

Figure 4, in the Technical Support section, shows a setup without line drivers, which can be used for in-system communication or for transmission via short cables. Both Z8 MCUs must be programmed with the same baud rate, one as transmitter and one as receiver and the RX and TX lines are crossed. Beginning with a cable length of 10cm, wave reflections occur and can result in significant overshoots and undershoots. Because P31 RX input is also used for OTP programming, allowing 12V input levels, P31 RX input does not feature an input protection diode to V_{CC}. This diode is external, in addition to an RC input filter of 1 µs to attenuate possible overshoots.

The oscillator buildup capacitors in both schematics are connected to V_{CC} instead of GND. For the targeted frequency range of 4-16MHz, the RF sees no difference between V_{CC} and GND. The V_{CC} connection allows a short and direct PCB path, resulting in an improved EFT behavior.

Summary

Reaffirmation of Results

The Software UART supports the most common UART protocol 8-N-1 and, optionally, a nine-bit mode. The concentration on the basic format results in efficient code and high baud rates. Special efforts were put into the reliability of the application, such as jitter-free transmission, glitch filter in the receive path, RAM data protection against spurious VBO and ESD, and a PC wraparound protection with SW-Reset. On the hardware side, the connection of the oscillator buildup capacitors to V_{CC} help achieve optimal system performance.

Technical Support

Source Code

```
; ***************************************************************************
*
*       Module Name:    SW-UARTX.ASM V1.2
*       Copyright:      ZILOG (c)1999
*       Date:           OCT 7, 1999
*       Created by:     Mathias Loehr - ZILOG Germany
*       Modified by:
*       Compiler:       ZDS V3.00B1
*       Code size:      294 bytes
*       Checksum:       59B3 (Project-Settings-Debugger-Pad Memory FF)
*
*       Description:
*       Asynchronous Serial RS-232 Interface for Z86E02 and higher
*       Half-Duplex Operation: Receive (RX) or Transmit (TX)
*
*       Selectable Baudrates:
*       RX-4MHz:        1200,2400,4800,9600
*       RX-8MHz:        1200,2400,4800,9600,19200
*       RX-12MHz:       1200,2400,4800,9600,19200,38400*
*       RX-16MHz:       2400,4800,9600,19200,38400
*                       *crystal <0.5%
*
*       TX-4MHz:        1200,2400,4800,9600,19200
*       TX-8MHz:        1200,2400,4800,9600,19200,38400
*       TX-12MHz:       1200,2400,4800,9600,19200,38400,57600
*       TX-16MHz:            2400,4800,9600,19200,38400,57600
*
*       Format:         1 START bit, 8 DATA bits, optional 9. DATA bit,
*                       NO PARITY and 1 STOP bit.
*       Operation:      T1 in continuous mode is used for bit-time delay
*                       Register DATA is used for RX or TX buffer
*
*       Target Device:  Z86E02 and Z86E04/08/30/31/33/34/40/44/733/743
*       Oscillator:     4-16MHz crystal or resonator, tolerance max. +-1%
*
*       Device Options: AUTOLATCHES OFF         YES
*                       LOW EMI                 NO
*                       32KHZ-OSC               NO
*                       RC-OSC                  NO
*                       PERMANENT WDT           NO
*                       TESTMODE DISABLE        NO
*
*       Defaults:       8MHz XTAL, 19200 Baud TX, 8 DATA, NO PARITY, 1 STOP
*
*       Application:    RX-Mode - 16 byte receive buffer
*                       TX-Mode - Send ROM-message
*
*       Mode Options:   - RX/TX operation mode is set by MODE in "start"
*                       - 9-bit mode is enabled by NINE_BIT_MODE in "start"
*                       - Oscillator Frequency is set by XTAL in CONSTANTS DEF
*                       - Baudrate is set by BAUD in CONSTANTS DEF
*                       - Top of MCU RAM is set by RAM_TOP in CONSTANTS DEF
```

```
*                       - Top of MCU ROM is set by ROM_TOP in CONSTANTS DEF
*                       - Wait for STOP bit function is set by WAIT_STOP
*                         in CONDITIONAL ASSEMBLY SWITCHES
*
*       Other Features:  The built-in RAM pattern validation keeps valid
*                        RAM data in case of (spurious) Undervoltage Resets.
*                        Undervoltage (VBO) Reset can be also caused by ESD.
*
*************************************************************************

; 1. HARDWARE DESCRIPTION

;       Receive (RX):   P31
;       Transmit(TX):   P00

; Note:       Add clamping diode between P31 and VCC and RC input filter
10k,100pF)
;       at P31, if operated without line driver.


; 2. CONDITIONAL ASSEMBLY SWITCHES:

              GLOBALS  on              ;symbols globally available to linker

ON            EQU      1               ;
OFF           EQU      0               ;
WAIT_STOP     EQU      OFF             ;ON/OFF. Wait for Stop bit, verify it
                                       ;and set FRAME_ERR accordingly

; 3. SEGMENT DEFINITIONS

              DEFINE   ram_e02, space=rfile, org=04 ;RAM range 04-3F
              DEFINE   main_e02                     ;relocatable ROM 000C-01FA
              DEFINE   rollover_e02, org=01FB       ;non-relocatable ROM 01F
                                                    ;01FF


; 4. CONSTANTS DEFINITIONS

XTAL          EQU      8000            ;SELECT CRYSTAL FREQUENCY HERE!
                                       ;Range: 4000-16000=4-16MHz (OSC/2 mode)
                                       ;4MHz Low EMI in OSC/1 mode = 8000

FIFTY_SEVEN   EQU      175*XTAL/8/10000        ;T=17,4us (57600 Baud)
                                                ; TCLK=XTAL/8000
THIRTY_EIGHT  EQU      26*XTAL/8/1000          ;T=26,0us (38400 Baud)
NINETEEN_TWO  EQU      52*XTAL/8/1000          ;T=52,1us (19200 Baud)
NINE_SIX      EQU      104*XTAL/8/1000         ;T=104,2us (9600 Baud)
FOUR_EIGHT    EQU      208*XTAL/8/1000         ;T=208,3us (4800 Baud)
TWO_FOUR      EQU      416*XTAL/8/1000         ;T=416,6us (2400 Baud)
ONE_TWO               EQU     833*XTAL/8/1000  ;T=833,2us (1200 Baud)
                                               ;(49,9 rounds down to 49)

BAUD          EQU      NINETEEN_TWO    ;SELECT BAUDRATE HERE!
                                       ;
       IF     BAUD > 1023
PBD           EQU      8               ;T1 Prescaler setting for bittime delay
       ELSEIF BAUD > 511
```

```
PBD             EQU     4               ;
        ELSEIF BAUD > 255
PBD             EQU     2               ;
        ELSE
PBD             EQU     1               ;
        ENDIF

BIT0            EQU     %01             ;Bit 0 mask
BIT1            EQU     %02             ;Bit 1 mask
BIT2            EQU     %04             ;Bit 2 mask
BIT3            EQU     %08             ;Bit 3 mask
BIT4            EQU     %10             ;Bit 4 mask
BIT5            EQU     %20             ;Bit 5 mask
BIT6            EQU     %40             ;Bit 6 mask
BIT7            EQU     %80             ;Bit 7 mask

RAM_TOP                 EQU     %3F     ;SELECT TOP OF RAM HERE!
RAM_BOT                 EQU     %04     ;bottom of RAM
ROM_TOP                 EQU     %01FF   ;SELECT TOP OF ROM HERE!

DATA_LENGTH     EQU     8               ;basic data bits (fixed)
RX              EQU     0               ;RX mode
TX              EQU     1               ;TX mode

PATTERN1        EQU     %5A             ;RAM validation pattern
PATTERN2        EQU     %A5             ;RAM validation pattern
PATTERN3        EQU     %F0             ;RAM validation pattern


; 5. NON-RELOCATABLE I/O-PORTS, RAM & STATUS REGISTER DEFINITIONS
;    (LOCAL VARIABLES)

; user-defined port & bank 0 registers (4-bit addressing mode)
PORT_GRP        EQU     %00             ;port register group
_P0             EQU     R0              ;Port 0 R/W
_P2             EQU     R2              ;Port 2 R/W
_P3             EQU     R3              ;Port 3 R+W
_BIT_CNT        EQU     R4              ;serial bit counter
_DATA           EQU     R5              ;receive/transmit data
_DATA2          EQU     R6              ;transmit data 2
_NINTH          EQU     R7              ;9. data bit, usable by SW as
                                        ;data/command switch
_NINE_BIT_MODE          EQU     R8      ;Nine Bit Mode ON/OFF
_RXH            EQU     R9              ;used in "ser_in" RX-ISR
_RX_PTR                 EQU     R10     ;receive buffer pointer
_TX_CNT                 EQU     R11     ;transmit byte counter
_TX_PTR                 EQU     RR12    ;pointer to transmit table in ROM
_TX_PTR_H       EQU     R12             ;MSB at even address
_TX_PTR_L       EQU     R13             ;LSB at odd address

; user-defined control registers (4-bit addressing mode)
CTRL_GRP        EQU     %F0             ;control register group
_SIO            EQU     R0              ;sio register (only on classic Z8)
_TMR            EQU     R1              ;timer mode register
_T1             EQU     R2              ;timer/counter 1
_PRE1           EQU     R3              ;prescaler 1
_T0             EQU     R4              ;timer/counter 0
_PRE0           EQU     R5              ;prescaler 0
```

```
_P2M            EQU     R6              ;port 2 mode register
_P3M            EQU     R7              ;port 3 mode register
_P01M           EQU     R8              ;port 0/1 mode register
_IPR            EQU     R9              ;interrupt priority register
_IRQ            EQU     R10             ;interrupt request register
_IMR            EQU     R11             ;interrupt mask register
_FLAGS          EQU     R12             ;flag register
_RP             EQU     R13             ;register pointer
_SPH            EQU     R14             ;stack pointer High byte
_SPL            EQU     R15             ;stack pointer Low byte


; 6. RELOCATABLE RAM DEFINITIONS
;    (GLOBAL VARIABLES)

                SEGMENT ram_e02         ;general purpose RAM 04-3F

BIT_CNT                 DS      1       ;04 - serial bit counter
DATA            DS      1               ;05 - receive/transmit data
DATA2           DS      1               ;06 - transmit data 2
NINTH           DS      1               ;07 - 9. data bit, usable by SW as
                                        ;     data/command switch
NINE_BIT_MODE   DS      1               ;08 - 1=ON, 0=OFF
RXH             DS      1               ;09 - various use in serial_in and
serial_out
RX_PTR          DS      1               ;0A - receive buffer pointer
TX_CNT          DS      1               ;0B - transmit byte counter
TX_PTR          BLKW    1               ;0C - pointer to transmit table in ROM
                DS      1               ;0E - unused
                DS      1               ;0F - unused

FRAME_ERR       DS      1               ;10 - Frame Error Flag, 0=ok, 1=bad
BAUD_SH                 DS      1       ;11 - Baudrate Shadow
P0_SH           DS      1               ;12 - Port 0 Shadow
MODE            DS      1               ;13 - RX/TX Operation Mode
RX_AVAIL        DS      1               ;14 - 1 = receive character available
                                        ;     0 = no data received
SIGNATURE1      DS      1               ;15 - RAM signature
SIGNATURE2      DS      1               ;16 - RAM signature
SIGNATURE3      DS      1               ;17 - RAM signature
                DS      1               ;18 - unused
                BLKB    7               ;19-1F unused
RX_BUF          BLKB    16              ;20-2F Receive Buffer
RX_BUF_END                              ;30
USER_STACK      BLKB    16              ;30-3F reserved for stack

; Max. Stack Depth: 2 Call + 1 INT = 7 bytes
; Top of Stack: RAM_TOP


; 7. Z8 RESET & INTERRUPT VECTORS

                VECTOR  IRQ0 = dummy_isr        ;P32 int, falling edge
                VECTOR  IRQ1 = dummy_isr        ;P33 int, falling edge
                VECTOR  IRQ2 = serial_in        ;P31 int, falling edge
                VECTOR  IRQ3 = dummy_isr        ;P32 int, rising edge
                VECTOR  IRQ4 = dummy_isr        ;T0 (not present on E02)
                VECTOR  IRQ5 = timer1           ;T1 int
```

```
;             VECTOR   RESET = main             ;Reset vector


*****************************************************************************
* Application Name:    SW-UARTX
*****************************************************************************

            SEGMENT    main_e02                 ;
main:
               di                                ;
               srp      #CTRL_GRP                ;select Control register bank
               ld       _P3M,#BIT0               ;P3 digital inputs
                                                 ;P2 Push-Pull
               ld       _P2M,#00000000b          ;P2 all outputs
               ld       P0_SH,#BIT0              ;TX=1 (P00 mark state)
               ld       P0,P0_SH                 ;output on Port 0
               ld       _P01M,#BIT2              ;internal stack, P0 output
               cp       SIGNATURE1,#PATTERN1     ;check RAM pattern 1
               jr       nz,ram_is_invalid
               cp       SIGNATURE2,#PATTERN2     ;check RAM pattern 2
               jr       nz,ram_is_invalid
               cp       SIGNATURE3,#PATTERN3     ;check RAM pattern 3
               jr       z,ram_is_ok              ;if RAM is intact skip RAM init
ram_is_invalid:
               ld       _SPL,#RAM_BOT            ;RAM bottom
init_ram:
               clr    @_SPL                      ;clear RAM byte
               inc    _SPL                       ;point to next byte
               cp     _SPL,#RAM_TOP+1            ;top of RAM exceeded?
               jr     nz,init_ram                ;leaves SPL at RAM_TOP+1

               ld       SIGNATURE1,#PATTERN1     ;write special test patterns
               ld       SIGNATURE2,#PATTERN2     ;into contiguous RAM locations
               ld       SIGNATURE3,#PATTERN3     ;
ram_is_ok:
               ld       _SPL,#RAM_TOP+1          ;init stack pointer
               clr      TMR                      ;disable timers
               ld       _PRE1,#PBD<<2+BIT0+BIT1  ;Modulo-N T1 mode (autoreload)
                                                 ;PSC=PBD. TCLK=SCLK/4
               ld       _T1,#BAUD/PBD            ;T1 bit-time delay
               ld       BAUD_SH,#BAUD/PBD        ;Baudrate Shadow

               ld       _IPR,#%2A                ;int. priority: 3>5>2>0>4>1
                                                 ;reserve IRQ3 for highest int
               ld       _IMR,#00000100b          ;enable IRQ2 (RX-INT)
               clr      IRQ                      ;clear spurious IRQ´s
               ei                                ;enable interrupts globally

start:
               srp      #PORT_GRP                ;Port Register Group
               ld       MODE,#TX                 ;SELECT RX/TX MODE HERE!
               ld       _NINE_BIT_MODE,#OFF      ;SELECT NINE BIT MODE HERE!
                                                 ;ON or OFF
               cp       MODE,#RX                 ;RX mode?
               jr       z,rx_mode       ;


; Transmits a whole message, stored bytewise in ROM at tx_rom
```

```
tx_mode:
              ld      _TX_CNT,#(tx_end-tx_rom) ;rom table length
              ld      _TX_PTR_H,#>tx_rom       ;load MSB of register pair
              ld      _TX_PTR_L,#<tx_rom       ;load LSB of register pair
tx_lp:        ldc     _DATA,@_TX_PTR           ;get constant from ROM table
              ld      _NINTH,#0                ;optional 9. data bit
              call    serial_out               ;transmit one character
              incw    _TX_PTR                  ;point to next character
              djnz    _TX_CNT,tx_lp            ;transmit whole message
              jr      start                    ;


; Inits RX mode, receives 16 bytes and stores in RAM buffer at RX_BUF.
; Buffer is overwritten when full.
; Received character is also displayed at P2.

rx_mode:
              ld      _RX_PTR,#RX_BUF          ;
wait_rx:
              cp      RX_AVAIL,#1              ;character received?
              jr      nz,wait_rx               ;
              clr     RX_AVAIL                 ;reset flag immediately
              ld      _P2,_DATA                ;..save/process data
              ld      @_RX_PTR,_DATA           ;  within one bit-time
              inc     _RX_PTR                  ;
              cp      _RX_PTR,#RX_BUF_END      ;end of buffer exceeded?
              jr      nz,wait_rx               ;wait for next char
              jr      start                    ;


; SUBROUTINES

*****************************************************************************
* Transmit Data
*
* Returns:    none. DATA and NINTH are destroyed.
*
* Entry Values:       DATA - byte to be transmitted
*                     NINTH - bit 0 holds the optional 9. data bit
*                     P0_SH - shadow register of P0
*
*
* Description:        DATA+DATA2 form a 16-bit shift register for serial
                      transmission LSB is transmitted first.
*                    T1 is setup in continous mode to generate the bit-time
                      delay.Other interrupts are disabled during transmission.
*
* Notes:      jitter-free bit synchronisation to falling START edge by
*             entering HALT and waiting for T1 vector interrupt.
*
*****************************************************************************

serial_out:
              di                               ;
              push    IMR                      ;save user interrupt mask
              ld      IMR,#BIT5                ;enable T1 interrupt only
```

```
                or      P0_SH,#BIT0             ;TX=1 (Idle mode = mark level)
                ld      _P0,P0_SH               ;output on P00

                ld      _DATA2,#1               ;shift in STOP bit
                cp      _NINE_BIT_MODE,#ON      ;9-bit mode on?
                jr      nz,serial_out1          ;branch, if not
                sra     NINTH                   ;9. bit into Carry
                rlc     DATA2                   ;shift in 9. data bit
serial_out1:
                rcf                             ;START into Carry
                rlc     DATA                    ;START into LSB, D7 into Carry
                rlc     DATA2                   ;now all bits are in

                ld      _BIT_CNT,#DATA_LENGTH+2 ;sync on first T1 interrupt
                add     _BIT_CNT,_NINE_BIT_MODE ;
                ei                              ;
                ld      TMR,#BIT2+BIT3          ;load and enable T1
send_lp:
                rrc     DATA2                   ;LSB into Carry
                rrc     DATA                    ;Carry into MSB, LSB into Carry
                rlc     P0_SH                   ;get Carry into LSB
tx_sync:
                nop                             ;
                halt                            ;wait on T1 interrupt
                ld      _P0,P0_SH               ;output at P00
                rrc     P0_SH                   ;restore P0_SH
                djnz    _BIT_CNT,send_lp        ;loop delay = 54+40=94 SCLK

                clr     TMR                     ;disable T1
                di                              ;disable interrupts
                pop     IMR                      ;restore user interrupt mask
                ei                              ;
                ret                             ;


;*******************************************************************************
; Receive data (Interrupt Service)
;
; Returns:      DATA            received data byte, LSB first
;               NINTH           optional 9. data bit in LSB
;               RX_AVAIL 0 = nothing received
;                               1 = character is available
;               FRAME_ERR       0 = ok
;               (optional)      1 = invalid STOP bit detected
; Entry Values:         none
;
; Description:          Falling START edge causes P31-IRQ2 service.
;               After nearly a half bit-time RX is sampled again to validate
;               the START bit. Immediate exit in case of glitch or noise.
;               Otherwise IRQ5 is enabled and T1 is setup in continous mode
;               for bit-time delay. T1 is polled for timeout and received
;               bits are shifted into DATA.
;               Routine does not wait for the STOP bit and exits in the
;               second half of the last data bit.
;               RX_AVAIL is set, T1 is disabled and IRQ-bit2 is cleared.
;
; Response:     P31-IRQ2: 26 + 0.5x longest instruction = 36 +-10 SCLK
```

```
*              T1-IRQ5:  2+22/2 = 13 +-11 SCLK
* Jitter:      Total sampling jitter: +-21 SCLK
*
*******************************************************************************

serial_in:
               cp      BAUD_SH,#66              ;check BAUD
               jr      ult,validate_start       ;short cut for fast baudrates
               ld      _RXH,#(BAUD/6)-(120/12)  ;Half-bit delay count in DJNZ
half_bit:
               djnz    _RXH,half_bit            ;12 SCLK - execute delay
validate_start:
               tm      P3,#BIT1                 ;RX=0?
               jr      nz,rx_exit               ;if zero, START bit is valid!
               ld      TMR,#BIT2+BIT3           ;load and enable T1 (bit time)

               or      IMR,#BIT5                ;enable T1 Interrupt (polling)
               clr     NINTH                    ;clear bit 9
               ld      _BIT_CNT,#DATA_LENGTH    ;number of basic data bits
receive_lp:
               tcm     IRQ,#BIT5                ;bit-time elapsed - IRQ5=1?
               jr      nz,receive_lp            ;
sample_data:
               ld      _RXH,_P3                 ;sample P31
               rr      RXH                      ;shift right
               rr      RXH                      ;P31 into Carry
               rrc     DATA                     ;Carry into MSB
               and     IRQ,#~BIT5               ;reset IRQ5
               djnz    _BIT_CNT,receive_lp      ;loop delay: 86+13=99+-11 SCLK

               cp      _NINE_BIT_MODE,#ON       ;9-bit mode on?
               jr      nz,rx_ok
wait_bit9:
               tcm     IRQ,#BIT5                ;bit-time elapsed - IRQ5=1?
                 jr    nz,wait_bit9             ;
               tm      P3,#BIT1                 ;RX=0?
               jr      z,rx_ok                  ;
               ld      _NINTH,#1                ;set data bit 9
rx_ok:
               ld      RX_AVAIL,#1              ;set data available flag

               IF      WAIT_STOP=ON
wait_STOP_bit:
               tcm     IRQ,#BIT5                ;bit-time elapsed - IRQ5=1?
               jr      nz,wait_STOP_bit         ;
               tcm     P3,#BIT1                 ;STOP=1?
               jr      z,frame_ok
               ld      FRAME_ERR,#1             ;indicate frame error
               ENDIF
frame_ok:
               clr     TMR                      ;disable T1
               and     IMR,#~BIT5               ;disable T1 interrupt
rx_exit:
               and     IRQ,#~(BIT2+BIT5)        ;clear P31 and T1 requests
               iret                             ;ok exit

*******************************************************************************
```

```
* Timer 1 Interrupt Service Routine
*
* Execution:   26 (latency) + 14 (IRET) = 40 SCLK
*
* Notes:       If this ISR shall be used by other tasks, the T1 interrupt
*              in serial_out may be changed from vector to polling mode.
*
;*****************************************************************************

timer1:
               iret                              ;


;*****************************************************************************
*      Interrupt Service for unused interrupt vectors
;*****************************************************************************

dummy_isr:
               iret


;*****************************************************************************
*      Table #1 - TX message string
;*****************************************************************************

tx_rom:
               .ASCII   "SW-UARTX V1.2 (c)1999 ZILOG"
               DB       %0D                      ;Carriage Return
               DB       %0A                      ;Newline
tx_end:


; RELOCATABLE PC ROLLOVER PROTECTION
; recommended SW-Reset to protect from wrap-around opcode fetches into
; interrupt vector table.

               SEGMENT  rollover_e02             ;trap routine at the end of ROM
               ORG      ROM_TOP-4
               nop                               ;synchronize
               nop                               ;synchronize
               jp       main                     ;SW-Reset
```

Flow Charts

Figure 2. Transmit Flow Chart

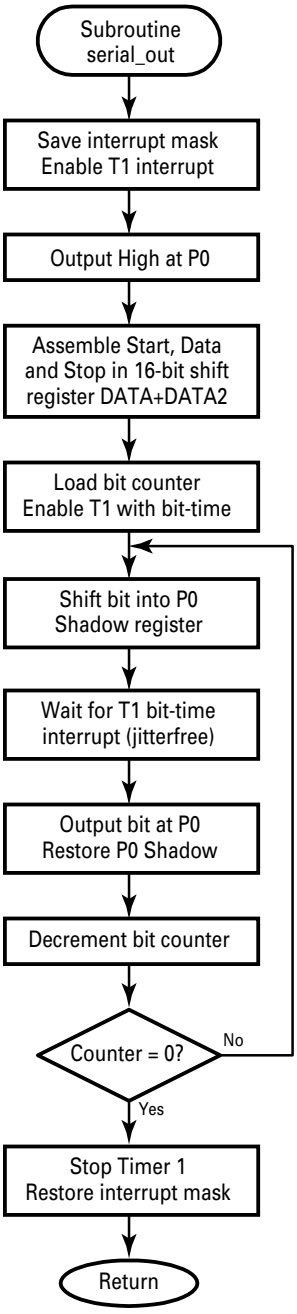
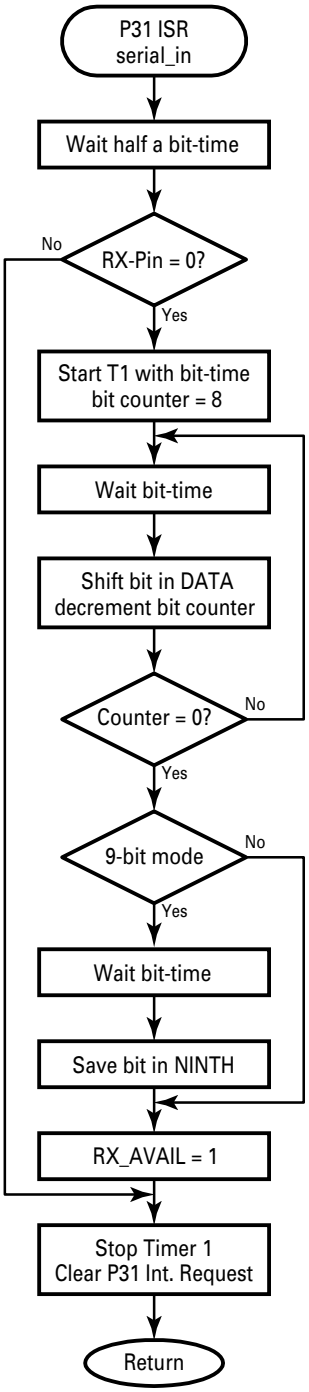
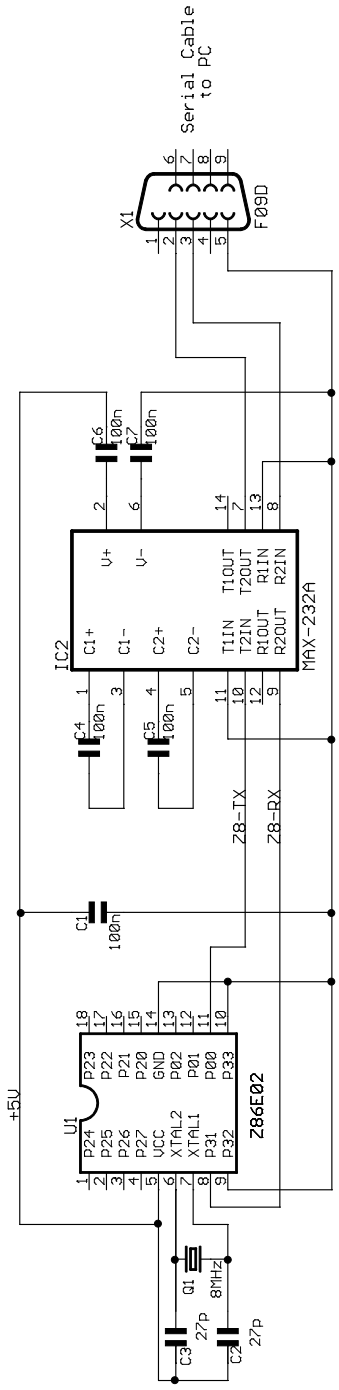


Figure 3. Receive Flow Chart



Schematic

Figure 4. RS-232 SetupFigure 4



Test Procedure

Equipment Used

The following test equipment was used for testing:

- Prototype board according to schematic
- Personal computer with Windows 95
- Z86CCP01ZEM, called CCP emulator
- Digital oscilloscope (required during debug phase)
- Clock generator (required during debug phase)

General Test Setup and Execution

Test with OTP

The board is equipped with an OTP Z86E02, programmed with the CCP emulator and the program defaults 8MHz XTAL, 19200 TX, 9-bit mode Off and WAIT_STOP Off. The OTP programming procedure is given in the Appendix.

The board is connected by a 9-pin serial cable to a standard PC running Windows 95. The Windows HyperTerminal is called and setup for *direct* connection via COM1, 19200 baud, 8-N-1 and No Protocol. When the board is supplied with +5V, the line message `SW-UARTX V1.2 © 1999 ZILOG` is displayed across the screen. A complete setup for HyperTerminal is detailed in the Appendix. Other terminal programs may be used as well, if they provide corresponding settings.

Test with an Emulator

The CCP emulator is used for the test of all baud rates at clock frequencies of 4MHz, 8MHz and 12MHz (16-MHz operation requires a Z86C50 emulator; selected CCP emulators may also work at 16MHz.The board is connected to the emulator by an ICE cable, and the V_{CC} jumper on the emulator is set to supply the board. The software calls the ZiLOG Developer Studio (ZDS), and a project is created for the target MCU. The settings for assembler and linker remain at their default values. A test for different baud rates and clock frequencies can be performed by changing values for the BAUD and XTAL variables. However, please note that the crystal on the emulator must be exchanged accordingly. Because HyperTerminal does not synchronize or error-check the transmission in this setting, a program halt and subsequent recontinuance of program execution may lead to an improper display.

RECEIVE mode is tested in the same setup. MODE, BAUD and XTAL are set before compiling. The ASCII representation for each pressed key becomes visible

in the Z8 RAM buffer, starting at 20h. To test a continuous data stream, a text file with 16 characters can be setup and transmitted from HyperTerminal.

Test Results

All settings according to baud rates and clock frequencies in Table 1 test successfully. A clock tolerance of±1% is sufficient for all settings, except 38400 RX @ 12MHz, which requires a tolerance of ±0.5%.

The basic timing for transmission and reception includes the following variance per bit:

1. Clock frequency tolerance, as provided by a crystal or resonator: <1%
2. T1 integer value to bit timing deviation: 0.16%
3. BAUD integer rounding for odd crystal frequency: <0.8%

These variances are added and multiplied by the number of transmitted bits (for example, 10). The resulting word variance must be <50%, because this variance is the maximum shift from the middle of the bit cell to its margin. In practice, the word variance should be reasonably lower than 50% for the following reasons:

- The transmitter also exhibits a clock variance
- Receiver bit sampling is not exactly in bit middle
- There is jitter in the transmitter and receiver timing

The SW_UARTX receiver samples in the bit middle (50% of the bit cell) for all baud rates, except for the highest setting at each frequency. 38200 baud at 16MHz and all corresponding baud rates sample at round 60%. 38200 baud at 12MHz samples at approximately 70% of the bit cell. Additionally, there is jitter from P31 interrupt and T1 IRQ polling, resulting in a worst-case measurement of ±21 SCLK. These values were recorded during the debug phase with a frequency generator and a digital scope.

References

Gilbert Held, Data Communications Networking Devices, John Wiley & Sons, 1993.

Z86C36 Product Specification, ZiLOG, Inc., 1999.

Appendix

Setup HyperTerminal

Connect: Direct to COM1 or any available COM port

Configure:

- Bits per second: 19200
- Data Bits: 8
- Parity: None
- Stop Bits 1
- Flow Control: None

Settings:

- Function, arrow and control keys act as: Windows Keys
- Emulation: Auto detect
- Back-scroll buffer lines: 0

ASCII Setup: ASCII Receiving:

- Force incoming data to 7-bit ASCI
- Wrap lines that exceed terminal width

OTP Programming Procedure

- Install ZDS version 3, and copy `SW_UARTX.ASM` and `SW_UARTX.ZWS` into the working directory
- IF file `ZWS` is not available, create `FILE` ➤ `NEW PROJECT` for the Z86E02 and Z86CCP01ZEM emulator
- ELSE navigate to `FILE` ➤ `OPEN PROJECT` and answer YES to move the project into the directory path
- BUILD the downloadable file for the debugger
- Navigate to PROJECT ➤ SETTINGS ➤ DEBUGGER ➤ PAD MEMORY, and check value `FF` for the correct checksum
- Connect the CCP emulator, power up, and click the RESET debug icon
- Click the OTP icon

- Select DEVICE Z86E02 and the corresponding TOPMARK, (for example, SL1925)
- Select the following PROGRAMMING OPTIONS:
  - AUTOLATCHES DISABLE: YES
  - LOW EMI: NO
  - RC-OS: NO
  - PERMANENT WDT: NO
  - EPROM/TESTMODE DISABLE: NO
- Select SERIALIZATION *none*
- Insert an empty OTP into the emulator programming socket and perform BLANK CHECK
- PROGRAM the device and note the checksum