TEXAS INSTRUMENTS

MSP430

IrDA SIR Encoder/Decoder

Application Report

March 1999

Mixed Signal Products

SLAA044

Contents

List of Figures

List of Tables

MSP430 IrDA SIR Encoder/Decoder

Juergen Mayer

ABSTRACT

This report gives a short overview on the use of the MSP430x112 as an IrDA SIR encoder/decoder. The hardware is discussed, including a block diagram and an IrDA module schematic diagram. The next sections deal with programming and user interface issues. Appendix A presents applicable IrDA software.

1 Introduction

Infrared data transmission's recent popularity has been fueled by the need to exchange data between portable and fixed equipment. Furthermore, the arrival of the Infrared Data Association (IrDA) working standard offers a practical and cost-efficient protocol for data communications.

The MSP430 is a powerful microcontroller capable of handling both the target application and the IrDA serial infrared protocol. Its affordability, its 16-bit architecture, and its low-power consumption, makes this technology possible in medium and high-speed data transmission applications where cost issues are important or battery-powered equipment is required. This document describes the use of the MSP430x112 as an IrDA-encoder/decoder based on the IrDA demo board.

The IrDA transceiver module can be used for serial data communication between two PC's via the RS232. Each PC requires only a standard RS232 port and some conventional terminal software such as the Hyperterminal, which is provided by Windows 3.11, Windows95, or WindowsNT.

The same basic code can be used on other applications of the MSP430 family, including liquid-crystal display drivers, analog-to-digital converters, and hardware multipliers.

2 The IrDA Serial Infrared Physical Layer Specification

The IrDA physical layer specification is intended to define a half-duplex infrared communication link for exchanging data over a distance of up to 1 m. The full standard includes data rates up to 4 Mbit/s. However, in this note we cover only data rates between 2.4 kbit/s and 115.2 kbit/s.

An IrDA serial infrared interface must operate at a minimum of 9.6 kbit/s, with higher data rates optional. The signaling rate and pulse duration specifications are listed in Table 1.

Table 1. IrDA Signaling Rate and Pulse Duration Specification

DATA RATE	BIT TIME	IrDA PULSE DURATION MINIMUM	IrDA PULSE DURATION NOMINAL	IrDA PULSE DURATION MAXIMUM
2.4 kbit/s	416 µs	1.41 µs	78.13 µs	88.55 µs
9.6 kbit/s	104 µs	1.41 µs	19.53 µs	22.13 µs
19.2 kbit/s	52.0 µs	1.41 µs	9.77 µs	11.07 µs
38.4 kbit/s	26.0 µs	1.41 µs	4.88 µs	5.96 µs
57.6 kbit/s	17.3 µs	1.41 µs	3.26 µs	4.34 µs
115.2 kbit/s	8.68 µs	1.41 µs	1.63 µs	2.23 µs

The minimum pulse duration is the same for data rates up to 115.2 kbit/s. The reason is that the IrDA physical layer specification allows two kinds of pulse modulations: 3/16 of a bit duration pulse, or a minimum pulse duration of 1.63 µs minus a 0.22 µs tolerance.

With the transmission speed limited to 115.2kbit/s, there is no need to transmit a serial infrared interaction pulse (SIP) which would guarantee nondisruptive compatibility with slower infrared (IR) systems. This leads to a very simple structure of the IrDA frame for data rates up to 115.2 kbit/s.

For data rates up to 115.2 kbit/s, the electrical signal to the encoder/decoder unit is a serial bit stream. A logical 0 in the bit stream is represented by an IR pulse with the duration shown in Table 1, and a logical 1 is a bit period with no IR pulse.

The bit stream is organized into frames with a start bit, 8 data bits, and a stop bit, as shown in Figure 1.

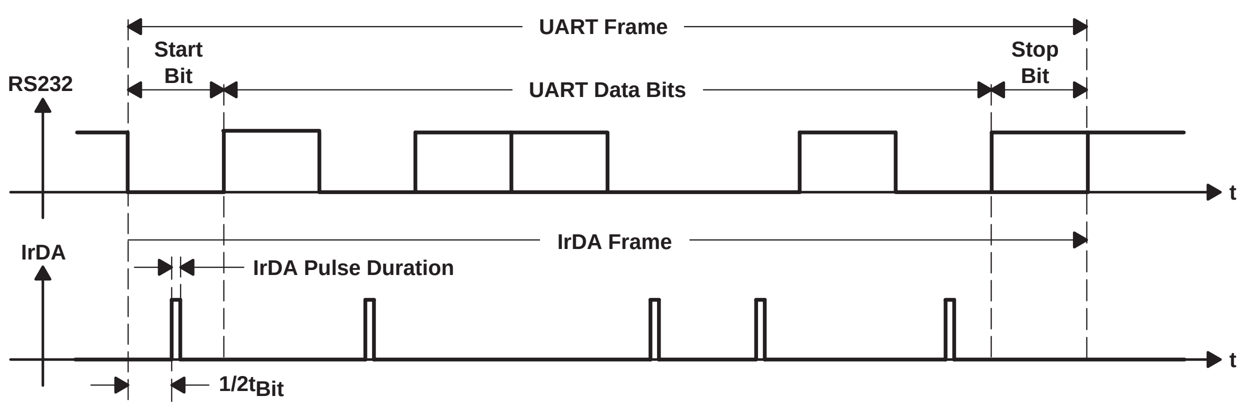

Figure 1. IrDA and UART Frames

The IrDA standard defines the IR pulse beginning at the center of the bit period with a length of 3/16 of a bit duration, or with a fixed pulse duration equivalent to a rate of 3/16 of 115.2 kbit/s.

Using the fixed pulse duration is recommended to reduce the power consumption in battery-powered applications.

3 The MSP430x112 – Low-End Ultra-Low Power Microcontroller

The MSP430x112 is the smallest member of the MSP430 family, a 16-bit ultra-low power microcontroller designed for battery powered applications. Each member of the family features a different set of peripheral modules. In contrast with most of the other derivatives of the MSP430 family. The x112 has no LCD driver and no AD converter.

The MSP430x112 consists of a powerful 16-bit timer with 3 capture/compare registers, fourteen I/O lines, and a new oscillator module.

The oscillator is designed to operate with no external components, with a low frequency crystal (32 kHz), or a high frequency crystal (up to 6 MHz), or can be driven by an external clock source up to 6 MHz.

4 MSP430x112 IrDA SIR Encoder/Decoder

4.1 Module Overview

Figure 2 shows the block diagram of the IrDA transceiver module. The heart of the module is a MSP430x112 which handles the encoding and decoding tasks as well as the communication to the RS232 interface. The TSLM1100 provides the required logical interface between the MSP430 and the IR signals. The IR transceiver is able to handle an IrDA Rev. 1.1 physical-layer compliant, bidirectional, half-duplex link. To reduce the power consumption, a fixed pulse-duration modulation of 1.63 µs nominal is implemented.

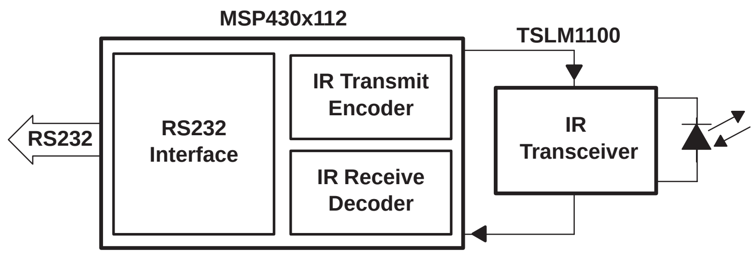

Figure 2. Block Diagram of the IrDA Module

The module can be directly connected to a conventional RS232 (DB9S connector) port. Due to the fact that the IrDA Physical Layer Specification defines a half-duplex link, the module must be initialized after reset to act as an IrDA receiver or transmitter—see section 4.3.2 User Interface.

4.2 MSP430x112 Software

The following section contains some general programming issues and a detailed discussion on the usage of TIMER_A to encode and decode the serial data stream.

A complete listing of the MSP430x112 software is shown in Appendix A.

4.2.1 General

The MSP430 must be initialized after system reset. The DCO modulation is disabled and the high frequency oscillator is enabled during initialization. This allows the application of a high frequency crystal pulse of up to 6 MHz to the Xin and Xout pins. This crystal generates the reference clock frequency used during serial data transmission. Alternatively, a 32 kHz crystal, in combination with a software FLL, can be used to lower system cost. This software FLL controls the DCO modulation register and adjusts the frequency based on the 32 kHz reference.

Next, the I/O ports are initialized and the jumper settings are checked. The corresponding counter value is stored in the variable *BAUDRATE* and used to determine the serial communication timing in both directions.

The initialization string is sent to the RS232. Since the status of the CTS (Clear To Send) line of the RS232 is not checked, we have to add a delay between each sent character. This prevents overflow of the RS232 input buffer at high data rates in some computers.

At this time, only the I/O Port P1.0 interrupt is enabled and ready to receive the initialization strings ^r or ^t. Depending on the user response, the MSP430 initializes all ports and interrupts to behave as a pure IrDA receiver or IrDA transmitter. Furthermore, it sends an acknowledgement via P1.1. This acknowledgement is displayed on the terminal window.

4.2.2 IrDA-SIR Encoder (TX)

If the user enters a ^t, the MSP430 acts as an encoder between the RS232 input port and the IR output port. The Timer_A and the ports are initialized as follows (see also schematic in chapter 5):

RS232 in (TXD)	⇒	Port P2.5
IrDA out (TSLM1100)	⇒	Port P2.4 (CCR2)

The half-period of bit duration is loaded into the period register CCR0, and Timer_A is set in the Up/Down mode. The capture/compare unit 2 is routed to port P2.4, and the corresponding capture compare register CCR2 is loaded with the value CCR0 minus the equivalent of the IR pulse duration of 1.63 µs.

During transmit, the capture/compare control register 2 (CCTL2) is set in the reset set mode. This means that every time the Timer_A register (TAR) reaches the value of CCR2, port P2.4 is set to low; if the TAR reaches the value CCR0, the port is set to high.

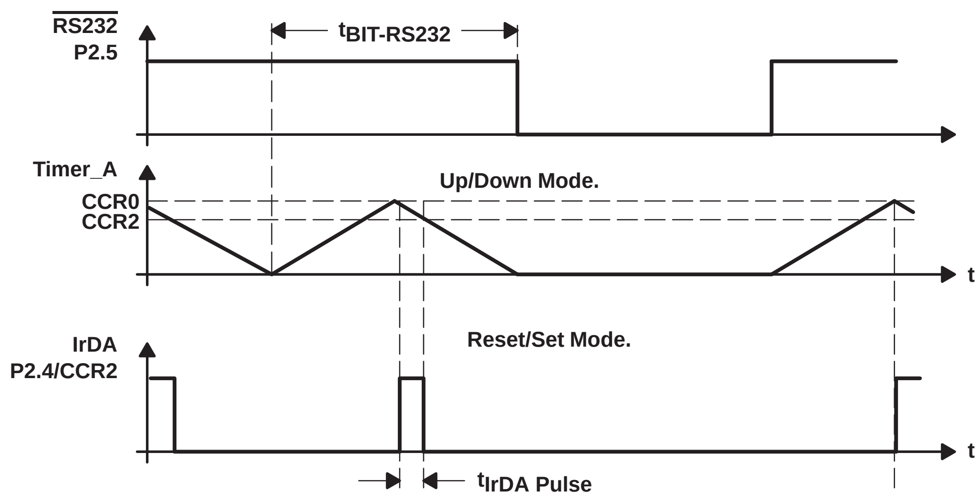

Figure 3. Timer_A Usage During RS232/IrDA Conversion

When Timer_A starts, it counts up to CCR0 and down to zero, and starts again counting up to CCR0. As long as Timer_A is running, it generates a 1.63 µs long pulse every bit period without CPU intervention.

The RS232 line driver (HIN232) produces an inverted serial bit stream at pin P2.5. A high level at P2.5 represents a logical 0, and vice versa. According to the IrDA Specification, a logical 0 on the RS232 line is represented by an IR pulse. This means that a rising edge at pin P2.5 must start Timer_A, and a falling edge must stop it. This sets the interrupt enable bit for port P2.5.

Each level shift on the RS232 line causes an interrupt, and the MSP430 starts to process the following interrupt service routine (ISR):

```
;*************************************************************************
; Interrupt routine TRANSMITTER
; RS232 (P2.5) –> IrDA (P2.4)
;*************************************************************************
TX_01  BIS       #04h,&TACTL   ;                                    5 cycles
       XOR R14,  &TACTL        ;start/stop Timer_A (up/down Mode)   4 cycles
       XOR.B     R13,&P2IES    ;toggle IR edge select 2.5           4 cycles
       BIC.B     #0FFH,&P2IFG  ;clear interrupt flags               5 cycles
       RETI                    ;                                    5 cycles
```

Six cycles after this interrupt event (the time is needed to save the processor status), the MSP430 enters the ISR. The timer must be stopped to change the contents of the Timer_A control register (TACTL).

Next, we start or stop the timer by toggling the mode bit in the TACTL register. Then the interrupt edge in the P2IES register is changed to produce an interrupt on the other edge.

The total ISR needs 23 + 6 cycles to enter the routine. Therefore, with a clock frequency of 3.6864 MHz (271.2 ns period) the ISR takes 29 x 271.2 ns, or 7.865 µs. During a 115.2 kbit/s serial communication, the minimum time between interrupts is 8.68 µs. The above interrupt service routine can fulfill this timing requirement.

4.2.3 IrDA-SIR Decoder (RX)

If the user enters an ^r, the MSP430 acts as a decoder between the IR input port and the RS232 output port. The Timer_A is halted and the ports are initialized as follows (see also schematic in chapter 5):

IrDA in (TSLM1100)	⇒	Port P1.2
RS232 out (RXD)	⇒	Port P1.1 (CCR0)

The period of the bit duration is loaded into the period register CCR0, and Timer_A is set (initialized) to count up to the value in CCR0, and restart the count to CCR0 again. The capture/compare unit 0 is routed to port P1.1.

During receive, the capture/compare control register (CCTL0) is in the set mode. This means that every time the Timer_A Register (TAR) reaches the value of CCR0, port P1.1 is set to high.

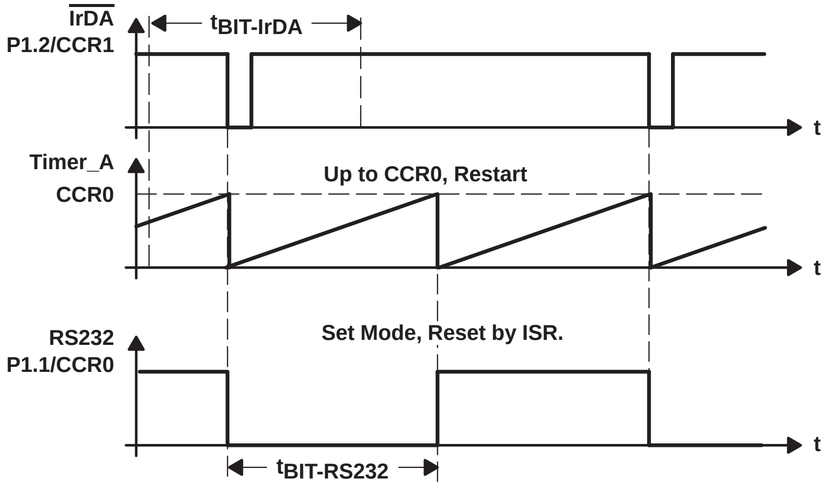

Figure 4. Timer_A Usage During IrDA/RS232 Conversion

If we start Timer_A, it counts up to CCR0, sets port P1.1 to high, and restarts from zero and counts up to CCR0 again. As long Timer_A is running, it generates a constantly high bit at port P1.1.

The IR transceiver generates the inverted IR signal at port P1.2. This means that a falling edge has to set port P1.1 to low and reset the timer. After that, Timer_A is synchronized with the IR signal. When the timer reaches the value CCR0 (RS232 bit duration) it automatically sets P1.1 to high and transmits a constant high level every time a new falling edge occurs.

```
;*******************************************************************************
; Interrupt routine RECEIVER
; IrDA (P1.2) –> RS232 (P1.1)
;*******************************************************************************
RX_01  CLR    &CCTL0       ;CC0 output mode / P1.1 low                   5 cycles
       BIS    #04h,&TACTL  ;reset Timer_A                                5 cycles
       MOV    #20h,&CCTL0  ;CCTL0 set P1.1 low and in ”mode–>P1.1 high”  5 cycles
       BIC    #01h,&CCTL1  ;reset interrupt flag CCTL1 !                 5 cycles
       RETI                ;                                             5 cycles
```

Each falling edge on port P1.2 causes an interrupt and the MSP430 starts to process the above interrupt service routine.

Six cycles after the interrupt event (falling edge on P1.2), the MSP430 enters the ISR. First, the P1.1 line is set to low and the CCR0 unit is set into output-only mode. Then, timer_A is reset by setting the CLR bit in the TACTL register, and the CCR1 interrupt flags must be reset.

The total ISR needs 25 + 6 cycles to enter the routine. Therefore, with a clock frequency of 3.6864 MHz (271.2 ns period), it needs 31×271.2 ns, or 8.407 µs. During a 115.2 kbit/s serial communication, the minimum time between interrupts is 8.68 µs. The above receiver interrupt service routine can fulfill this timing requirement.

Since every falling edge at P1.2 generates an interrupt and hence an RS232 low signal at P1.1, interference from a fluorescent lamp, or similar device, could generate bit errors. The time between the rising and falling edge can be measured by Capture/Compare Unit 2 and compared to the IrDA pulse duration to detect and eliminate such errors. To accomplish this, the CCR2 unit can be set to capture on the rising edge. As the CCR1 still captures on the falling edge and causes an interrupt, both captured counter values can be compared in the ISR. If the equivalent time is lower than 1.6 µs – 0.22 µs, the pulse is caused by interference and can be ignored. This requires a clock frequency higher than 3.68 MHz, or a limitation to lower data rates.

4.3 Installation

4.3.1 Data Rate Selection

The IrDA module can operate on six different data rates, ranging from 2.4 kbit/s to 115.2 kbit/s. A data rate must be selected before starting a communication session.

Table 2. Data Rate Settings

DATA RATE	JUMPER		
	J3	J2	J1
2.4 kbit/s	L	L	L
9.6 kbit/s	L	L	H
19.2 kbit/s	L	H	L
38.4 kbit/s	L	H	H
57.6 kbit/s	H	L	L
115.2 kbit/s	H	L	H

The data rate is set by the three jumpers J1, J2, and J3—see Table 3. A system reset will be performed if the jumper settings are changed during normal operation. The module can now be connected to the RS232 port on the PC and power can be applied. Make sure to use the same data rate setting on both PC terminals.

4.3.2 User Interface

A conventional terminal program such as Hyperterminal, which is provided by Windows 3.11, Windows95 or WindowsNT, can be used to communicate with the IrDA module.

The COM ports settings must be as follows:

Bits per second:	see Jumper J1 to J3 on IrDA module
Data bits:	7
Parity:	Even
Stop bits:	1
Flow control:	None

Make sure to use the same COM port settings on both PC terminals.

Pressing the RESET button on the IrDA module will display the following initialization string on the terminal window.

Figure 5. IrDA Initialization Message

Now it must be determined whether to set the module as a receiver or as a transmitter device.

When ^r is entered, the unit behaves as a receiver and displays an acknowledgement as shown in Figure 6.

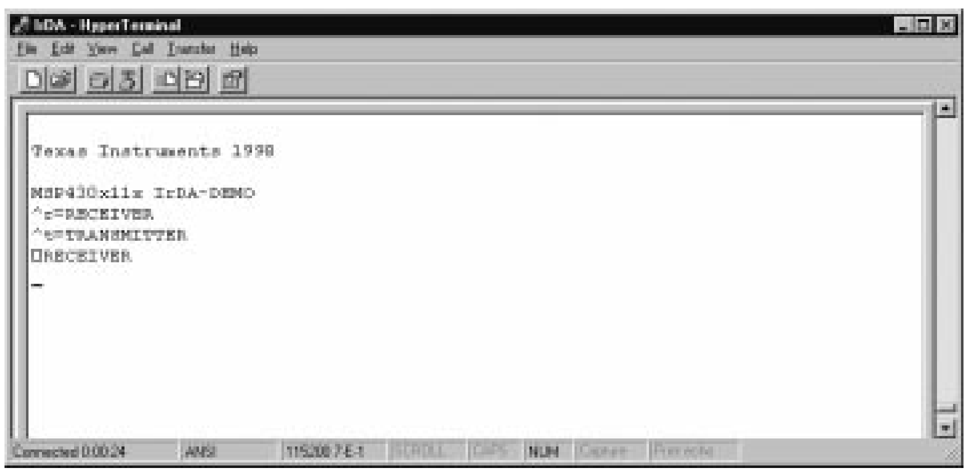

Figure 6. IrDA Initialization Acknowledgement – Receiver

To transmit data from one PC to an other, set up one IrDA module as a transmitter and the other as a receiver.

To reverse the direction of communication, just press the RESET button and restart the installation.

5 Schematic

Figure 7 shows the schematic of the IrDA module.

When a high frequency oscillator is used with a clock frequency above 3 MHz, the MSP430x112 requires a 5-V supply voltage. Due to the high power consumption of the RS232 line driver (HIN232), and to the fact that PC’s can not usually deliver sufficient supply current via the RS232, an external power supply or a battery may be required. Select the lowest practical crystal frequency to limit power consumption.

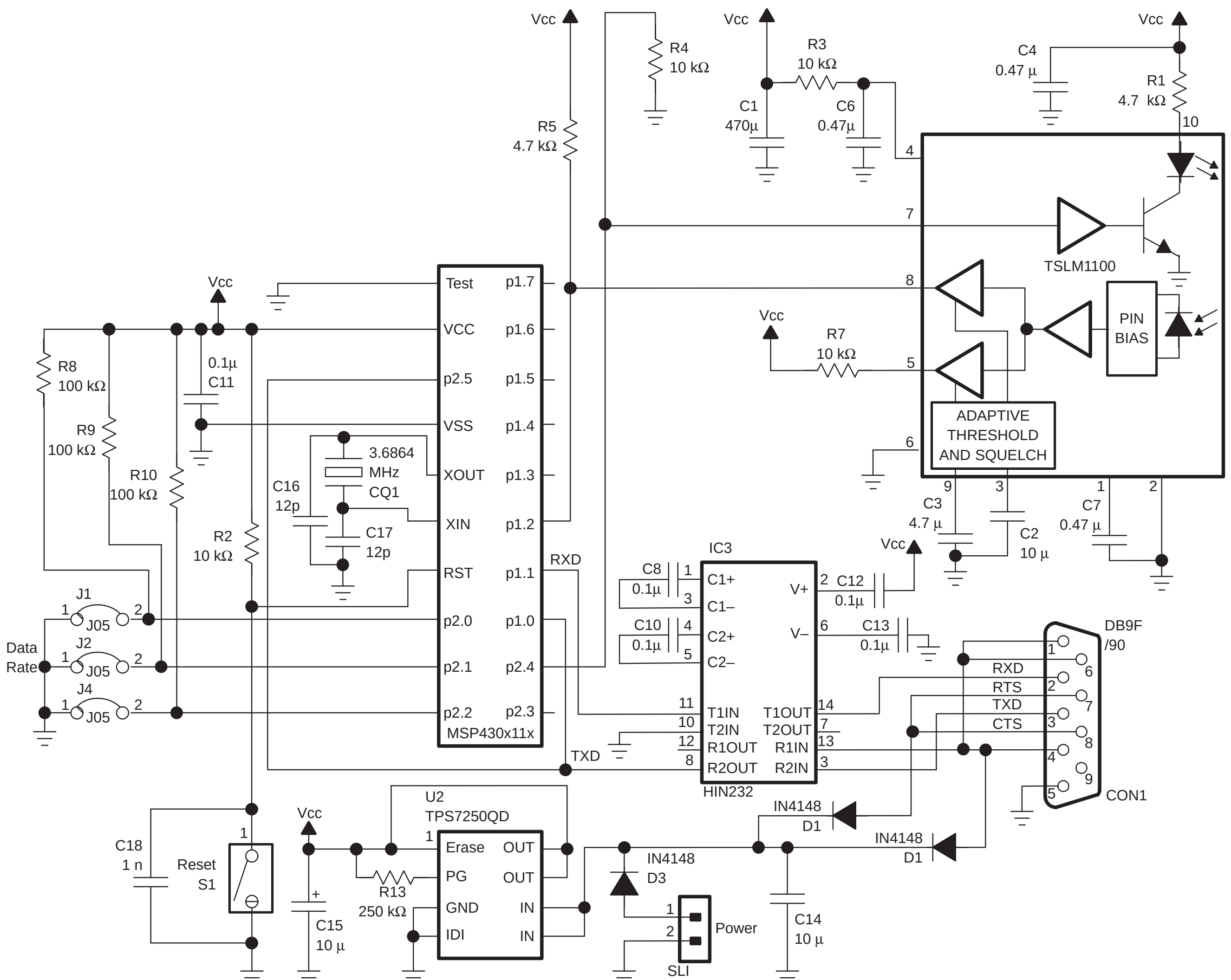

Figure 7. Schematic IrDA Module

The power supply for the TSLM1100 must be filtered to minimize noise from external sources. Capacitors C6 and C7 must be placed as close as possible to the TSLM1100 to achieve optimum noise immunity. For detailed application information, see the TSLM1100 data sheet.

Since the TSLM1100 is encapsulated in a light permeable plastic package, it is recommended to shield the device from sources of optical interference, especially fluorescent lamps and IR remote controls.

6 Conclusion

The IrDA specification is becoming increasingly popular as more and more applications require wireless data readout. This application note proves that low-power and wireless high-speed data communication can run hand-in-hand.

Using the MSP430C112 it is possible to build battery-powered applications including a wireless link communication up to 115.2 kbit/s.

The IrDA software presented in Appendix A can easily be adapted to any particular needs and implemented in the target application.

7 References

1. Texas Instruments, *Data Transmission Seminar 1998*, SLLDE01C Texas Instruments 1998.
2. Infrared Data Association, *Physical Layer Link Specification*, Version 1.1, 17.10.1995.
3. Texas Instruments, *Datasheet MSP430x110*, Texas Instruments 1998.
4. Texas Instruments, *MSP430 Family Architecture Guide and module Library*, Texas Instruments 1996.
5. Texas Instruments, *MSP430 Family Software User's Guide*, Texas Instruments 1994.
6. Texas Instruments, *Datasheet TSLM1100*, Texas Instruments 1997.

Appendix A A Software Listing

```
;***********************************************************************
; IrDA – Program MSP430x112          (C) TEXAS INSTRUMENTS 1998
;
; File Name:  IrDA11X.asm
; Project:    MSP430x112 IrDA Demo
; Originator: Jürgen Mayer  ( Texas Instruments Germany )
;
; Description: IrDA physical layer Rev. 1.1 – Encoder/Decoder
;              RX = RS232 –> IrDA
;              TX = IrDA  –> RS232
;              RX/TX controlled via ^T( = RS232 –> IrDA )
;                           and ^R( = IrDA  –> RS232)
;            Terminal settings: 1Start / 7Data–Bit / 1 Stop / even Parity
;
; Last Update: Rev. 2.1 / 28.05.98
;            Rev. 2.2 / 23.07.98
;***********************************************************************
SP_orig   .set   0300h     ; stackpointer
WDTCTL    .equ   0120h
WDTHold   .equ   080h
WDT_wrkey .equ   05A00h
IE1       .set   000h
IFG1      .set   002h
P1IN      .set   020h
P1OUT     .set   021h
P1DIR     .set   022h
P1IFG     .set   023h
P1IES     .set   024h
P1IE      .set   025h
P1SEL     .set   026h
P2IN      .set   028h
P2OUT     .set   029h
P2DIR     .set   02Ah
P2IFG     .set   02Bh
P2IES     .set   02Ch
P2IE      .set   02Dh
P2SEL     .set   02Eh
TACTL     .set   160h      ; Timer A
CCTL0     .set   162h
CCTL1     .set   164h
CCTL2     .set   166h
```

```
TAR       .set   170h
CCR0      .set   172h
CCR1      .set   174h
CCR2      .set   176h
DCOCTL    .set   056h                   ; DCO clock frequency control
BCSCTL1   .set   057h                   ; oscillator / clock control 1
BCSCTL2   .set   058h                   ; oscillator / clock control 2
TRANSMIT  .set   028h                   ; TX char ”^T”   – reverse, LSB first !
RECEIVE   .set   048h                   ; RX char ”^R”   – reverse, LSB first !
;*******************************************************************
; Main
;*******************************************************************
       .sect     ”ROM”, 0F000h
RESET  MOV       #SP_orig,SP            ;initialize stack pointer
       MOV       #(WDTHold+WDT_wrkey),&WDTCTL ; Stop Watchdog Timer
main_1 call      #init_sys
       CLR.B     STATUS_1
       CLR.B     STATUS_2
       CLR.W     BAUDRATE
       call      #init_Px
       call      #init_BAUDRATE
       MOV       BAUDRATE,R5            ;store Baudrate in R5
       RRA       R5                     ;divide by 2
       SUB       #02h,R5                ;adjust to timing
       MOV       R5,R8                  ;copy to R8
       MOV       #0,R9                  ;TEXT start...
       MOV       #28,R10                ;TEXT stop...
       call      #TEXT_OUT              ;write to PC...
       call      #init_RXTX
       MOV.B     STATUS_1,STATUS_2
       EINT                             ; Global interrupt enable
main_2
       MOV.B     &P2IN,R5               ;scan JUMPER –> R5
       BIC       #0FFF8H,R5
       MOV.B     R5,STATUS_1            ;store Baudrate
       CMP.B     STATUS_1,STATUS_2      ;any changes
       JEQ       main_2                 ;no
       DINT
       JMP       main_1                 ;restart
main_3 JMP       main_3
```

```
;*********************************************************************
; Reset : Initialize processor
;*********************************************************************
init_sys
      MOV.B  #00h,DCOCTL         ;disable DCO modulation
      BIS.B  #040h,BCSCTL1       ;1st: HF–osc. on => external crystal !!
      MOV.B  #0C8h,BCSCTL2       ;2nd: LFXT1 => MCLK / SMCLK
      CLR.B  IE1
      CLR.B  IFG1
      CLR    R5
      CLR R6
      CLR    R7
      CLR    R8
      CLR    R9
      CLR    R10
      CLR    R11
      CLR    R12
      CLR    R13
      CLR    R14
      CLR    R15
      RET
;*********************************************************************
; Initial BAUDRATE
;  – scan jumper
;  – set BAUDRATE
;**********************************************************************
init_BAUDRATE
      PUSH   R5
      CLR    R5
      MOV.B  &P2IN,R5            ;scan JUMPER –> R5
      BIC    #0FFF8H,R5
      MOV.B  R5,STATUS_1         ;store Baudrate
      RLA    R5
      ADD    #BAUD2400,R5
      MOV    @R5,BAUDRATE        ;load cycles in BAUDRATE
      POP    R5
      RET
;**********************************************************************
; Initial – Px
; SW – BAUDRATE – P2.0/1/2 input
; RX/TX – P1.0 = input –> IR edge select P1.0 HI\LO
; RX – P2.5 general input RS232
;          P2.4 module output IrDA
; TX – P1.2 general I/O <=> module input IrDA
;         P1.1 general I/O <=> module output RS232
;**************************************************************************
```

```
init_Px
      BIC.B  #0DH,&P1DIR            ;P1.0/2/3 input
      BIS.B  #02H,&P1DIR            ;P1.1 output
      BIC.B  #07H,&P1SEL            ;P1.0 general I/O ports
                                    ;P1.1/2 module ports during TX
      BIS.B  #02H,&P1OUT            ;set P1.1 – high
      BIC.B  #0FFH,&P1IFG           ;clear interrupt flags
      BIS.B  #001H,&P1IES           ;IR edge select P1.0 HI\LO
      BIS.B  #001H,&P1IE            ;Interrupt enable P1.0
      BIC.B  #10H,&P2OUT            ;set P2.4 – low
      BIC.B  #027H,&P2DIR           ;P2.0/1/2/5 input
      BIS.B  #010H,&P2DIR           ;P2.4 output
      BIC.B  #27H,&P2SEL            ;P2.0/1/2/5 general I/O Ports
      BIC.B  #0FFH,&P2IFG           ;clear interrupt flags
      RET
;***************************************************************************
; Initial ISR – RX/TX
; – P1.0 general input Pin RS232
; – Bit count in R6
;***************************************************************************
init_RXTX
      MOV    BAUDRATE,R5            ;store Baudrate in R5
      RRA    R5                     ;divide by 2
      SUB    #02h,R5                ;adjust to timing
      MOV    R5,R8                  ;copy to R8
      MOV    #08h,R6                ;load Bit counter in R6 ( n )
                                    ; –> see RX/TX Interrupt routine
      MOV    #25,R9                 ;TEXT start...
      MOV    #79,R10                ;TEXT stop...
      call   #TEXT_OUT              ;write to PC...
      RET
;*************************************************************************
; RX/TX Interrupt routine
; – input Pin P1.0
; – Baudrate R5/R8
; – Bit counter R6
; – DATA_in R7
;*************************************************************************
RXTX_01   MOV    R8,R8              ;NOP
          MOV    R8,R8              ;NOP
RXTX_02   MOV    R8,R5              ;load Baudrate in R5
RXTX_03   MOV    R8,R8              ;NOP
          DEC    R5
```

```
            JNZ    RXTX_03              ;sample line ?
        RRC     &P1IN                   ;P1.0 in Carry – LSB first !
        RLC     R7                      ;carry to DATA_in
        DEC     R6                      ;decrement Bit counter
        JNZ     RXTX_02                 ;last Bit ?
        CMP.B   #RECEIVE,R7             ; –> Receive ?
        JEQ     init_RX
        CMP.B   #TRANSMIT,R7            ; –> Transmit ?
        JEQ     init_TX
        BIC.B   #0FFH,&P1IFG            ;clear interrupt flag
        CLR     R7                      ;clr old DATA_in
        call    #init_RXTX
        RETI
init_TX CALL    #init_TATX              ;start RS232 –> IrDA transmission
        MOV     #66,R9                  ;TEXT start...
        MOV     #79,R10                 ;TEXT stop...
        call    #TEXT_OUT               ;write to PC...
        BIC.B   #0FFH,&P1IFG            ;clear interrupt flag
        BIC.B   #001H,&P1IE             ;Interrupt disable P1.0
        RETI
init_RX MOV     #53,R9                  ;TEXT start...
        MOV     #63,R10                 ;TEXT stop...
        call    #TEXT_OUT               ;write to PC...
        BIC.B   #0FFH,&P1IFG            ;clear interrupt flag
        BIC.B   #001H,&P1IE             ;Interrupt disable P1.0
        call    #init_TARX              ;start IrDA –> RS232 transmission
        RETI
;***************************************************************************
; Initial Timer_A  –  TRANSMITTER
; – TX mode  ( RS232 –> IrDA )
; – P2.5 general input Pin RS232
; – P2.4 CCR2 output Pin IrDA ( TSLM1100 )
;***************************************************************************
init_TATX
        MOV     #0200h,&TACTL           ;Prepare Timer_A ( MCLK,Timer halted...)
        CLR     CCTL1                   ;disable CCTL1 interrupt
        MOV     #0080h,&CCTL0           ;Capture/Compare Control 0
        MOV     BAUDRATE,&CCR0          ;Capture/Compare Register0 –> Period
        MOV     #00E0h,&CCTL2           ;Capture/Compare Control 2 –> operation mode
        MOV     BAUDRATE,R15
        SUB     #006h,R15               ;Subtract: impulse cycle > for up/down –Toggle !!
        MOV     R15,&CCR2               ;Capture/Compare Register2 –>Impulsduration 6h
        MOV     #030H,R14               ;start/stop Timer_A (up/down Mode)
```

```
          BIC.B  #0FFH,&P2IFG           ;clear interrupt flags port 2.x
          BIS.B  #020H,&P2IES           ;IR edge select P2.5 HI/LO
          BIS.B  #020H,&P2IE            ;Interrupt enable P2.5
          MOV    #020H,R13              ;toggle IR edge select 2.5 HI/LO <–> LO/HI
          BIS.B  #010H,&P2SEL           ;P2.4 module port
          RET
;*********************************************************************
; Initial Timer_A – RECEIVER
; – RX mode  ( IrDA –> RS232 )
; – P1.2 CCR1 input IrDA ( TSLM1100 ! )
; – P1.1 CCR0 output Pin RS232
;****************************************************************
init_TARX
          MOV    #0200h,TACTL           ;Prepare Timer_A ( MCLK,Timer halted...)
          MOV    #0000h,&CCTL0          ;CCTL0 in output mode to set P1.1 high
          BIS    #00004h,&CCTL0         ;P1.1 high
          BIS    #0020h,&CCTL0          ;CCTL0 in set mode –> P1.1 high
          MOV    BAUDRATE,&CCR0         ;load CCR0 with BAUDRATE
          RLA    CCR0                   ;BAUDRATE * 2
          BIS.B  #06H,&P1SEL            ;P1.1/2 module port –> CC1 !
          MOV    #8110h,CCTL1           ;–> cap. falling edge IrDA pulse + interrupt
          BIS.B  #010H,TACTL            ; start Timer_A
                                        ; 000h = halted
                                        ; 010h = up to CCR0 restart 0
                                        ; 020h = up to 65536
                                        ; 030h = up/down Mode
RET
;*****************************************************************
; Interrupt routine TRANSMITTER
; RS232 (P2.5) –> IrDA (P2.4)
;****************************************************************
TX_01     BIS    #04h,&TACTL
          XOR    R14,&TACTL             ;start/stop Timer_A (up/down Mode)
          XOR.B  R13,&P2IES             ;toggle IR edge select 2.5 HI/LO <–> LO/HI
          BIC.B  #0FFH,&P2IFG           ;clear interrupt flags
          RETI
;****************************************************************
; Interrupt routine RECEIVER
; IrDA (P1.2) –> RS232 (P1.1)
;****************************************************************
RX_01         CLR  &CCTL0               ;CC0 output mode / P1.1 low
              BIS    #004h,&TACTL       ;reset Timer_A
              MOV    #0020h,&CCTL0      ;CCTL0 set P1.1 low and in ”mode–>P1.1 high”
```

```
          BIC    #01h,&CCTL1         ;reset interrupt flag CCTL1 !
          RETI
;*************************************************************************
;Subroutine: write a string at TEXT(R9) to Terminal
; – output Pin P3.3 –> see init.
; – string start pos. R9 / stop pos. R10
;*************************************************************************
TEXT_OUT PUSH    R8
          PUSH   R7
          PUSH   R6
          PUSH   R5
          MOV    BAUDRATE,R5         ;store Baudrate in R5
          RRA    R5                  ;divide by 2
          SUB    #03h,R5             ;adjust to timing
TEXT_02   BIC.B  #02H,&P1OUT         ;write start bit –> LOW
          MOV.B  TEXT(R9),R7         ;load Data_byte
          MOV.B  #08H,R6             ;Bit count
          MOV.B  #08H,R6             ;just for....
          JMP    TEXT_03             ;....delay
TEXT_03   MOV    R8,R5               ;copy to R8
TEXT_04   MOV    R8,R8               ;NOP
          DEC    R5
          JNZ    TEXT_04             ;write next bit ?
          RLA.B  R7                  ;rotate in Carry
          JC     TEXT_H
          BIC.B  #02H,&P1OUT         ;write LOW bit
          DEC    R6
          JNZ    TEXT_03             ;next Byte ?
          JMP    TEXT_05
TEXT_H    BIS.B  #02H,&P1OUT         ;write HIGH bit
   DEC R6
          JNZ    TEXT_03             ;next Byte ?
TEXT_05   BIS.B  #02H,&P1OUT         ;write STOP bit –> HIGH
          MOV    #0F000h,R5          ;Delay between each character
TEXT_06   DEC    R5                  ; ...10ms
          JNZ    TEXT_06
          INC    R9
          CMP    R10,R9              ;x digits ?
          JNZ    TEXT_02             ;next Byte
          POP    R5
          POP    R6
          POP    R7
          POP    R8
          RET
```

```
;***********************************************************************
; Program Variables
;  – STATUS
;  – BAUDRATE    .WORD  #0200h
;***********************************************************************
   .sect  ”RAM”, 0200h
STATUS_1  .BYTE  #00H
STATUS_2  .BYTE  #00H
BAUDRATE  .WORD  #00H
;   .sect  ”ROM”             ;MCLK = 3.6864MHz / t=271.2ns
;BAUD2400 .WORD  #0300h      ; 1536/2 cycles
;BAUD9600 .WORD  #00C0h      ;  384/2 cycles
;BAUD1920 .WORD  #0060h      ;  192/2 cycles
;BAUD3840 .WORD  #0030h      ;   96/2 cycles
;BAUD5760 .WORD  #0020h      ;   64/2 cycles
;BAUD1152 .WORD  #0010h      ;   32/2 cycles
;   .sect  ”ROM”             ;MCLK = 4.194MHz / t=238.4ns
;BAUD2400 .WORD  #036Bh      ; 1750/2 cycles
;BAUD9600 .WORD  #00DAh      ;  436/2 cycles
;BAUD1920 .WORD  #006Dh      ;  218/2 cycles
;BAUD3840 .WORD  #0037h      ;  110/2 cycles
;BAUD5760 .WORD  #0024h      ;   72/2 cycles
;BAUD1152 .WORD  #0012h      ;   36/2 cycles
; sect ”ROM”                 ;MCLK = 5.0000MHz / t=200ns
;BAUD2400 .WORD  #0412h      ; 2083/2 cycles
;BAUD9600 .WORD  #0104h      ;  520/2 cycles
;BAUD1920 .WORD  #0082h      ;  260/2 cycles
;BAUD3840 .WORD  #0041h      ;  130/2 cycles
;BAUD5760 .WORD  #002Bh      ;   86/2 cycles
;BAUD1152 .WORD  #0016h      ;   43/2 cycles
   .sect  ”ROM”              ;MCLK = 6.144MHz / t=162.8ns
BAUD2400  .WORD  #0500h      ; 2560/2 cycles
BAUD9600  .WORD  #0140h      ;  640/2 cycles
BAUD1920  .WORD  #00A0h      ;  320/2 cycles
BAUD3840  .WORD  #0050h      ;  160/2 cycles
BAUD5760  .WORD  #0035h      ;  107/2 cycles
BAUD1152  .WORD  #001Ah      ;   53/2 cycles
;   .sect  ”ROM”             ;MCLK = 7.3728MHz / t=135.6ns
;BAUD2400 .WORD  #0600h      ; 3072/2 cycles
;BAUD9600 .WORD  #0180h      ;  768/2 cycles
;BAUD1920 .WORD  #00C0h      ;  384/2 cycles
;BAUD3840 .WORD  #0060h      ;  192/2 cycles
;BAUD5760 .WORD  #0040h      ;  128/2 cycles
;BAUD1152 .WORD  #0020h      ;   64/2 cycles
```

```
TEXT   .BYTE  005h      ;” ”
       .BYTE  0B1h      ;”CR1”
       .BYTE  050h      ;”CR2”
       .BYTE  02Bh      ;”T”
       .BYTE  0A6h      ;”e”
       .BYTE  01Eh      ;”x”
       .BYTE  087h      ;”a”
       .BYTE  0CFh      ;”s”
       .BYTE  005h      ;” ”
       .BYTE  093h      ;”I”
       .BYTE  077h      ;”n”
       .BYTE  0CFh      ;”s”
       .BYTE  02Eh      ;”t”
       .BYTE  04Eh      ;”r”
       .BYTE  0AFh      ;”u”
       .BYTE  0B7h      ;”m”
       .BYTE  0A6h      ;”e”
       .BYTE  077h      ;”n”
       .BYTE  02Eh      ;”t”
       .BYTE  0CFh      ;”s”
       .BYTE  005h      ;” ”
       .BYTE  08Dh      ;”1”
       .BYTE  09Ch      ;”9”
       .BYTE  09Ch      ;”9”
       .BYTE  01Dh      ;”8”
       .BYTE  005h      ;” ”
       .BYTE  0B1h      ;”CR1”
       .BYTE  050h      ;”CR2”
TXT_28 .BYTE  0B2h      ;”M”
       .BYTE  0CAh      ;”S”
       .BYTE  00Ah      ;”P”
       .BYTE  02Dh      ;”4”
       .BYTE  0CCh      ;”3”
       .BYTE  00Ch      ;”0”
       .BYTE  01Eh      ;”x”
       .BYTE  08Dh      ;”1”
       .BYTE  08Dh      ;”1”
       .BYTE  01Eh      ;”x”
       .BYTE  005h      ;” ”
       .BYTE  093h      ;”I”
       .BYTE  04Eh      ;”r”
       .BYTE  022h      ;”D”
       .BYTE  082h      ;”A”
```

```
       .BYTE  0B4h      ;”–”
       .BYTE  022h      ;”D”
       .BYTE  0A3h      ;”E”
       .BYTE  0B2h      ;”M”
       .BYTE  0F3h      ;”O”
       .BYTE  0B1h      ;”CR1”
       .BYTE  050h      ;”CR2”
TXT_50 .BYTE  07Bh      ;”^”
       .BYTE  04Eh      ;”r”
       .BYTE  0BDh      ;”=”
TXT_53 .BYTE  04Bh      ;”R”
       .BYTE  0A3h      ;”E”
       .BYTE  0C3h      ;”C”
       .BYTE  0A3h      ;”E”
       .BYTE  093h      ;”I”
       .BYTE  06Ah      ;”V”
       .BYTE  0A3h      ;”E”
       .BYTE  04Bh      ;”R”
       .BYTE  0B1h      ;”CR1”
TXT_62 .BYTE  050h      ;”CR2”
       .BYTE  07Bh      ;”^”
       .BYTE  02Eh      ;”t”
       .BYTE  0BDh      ;”=”
TXT_66 .BYTE  02Bh      ;”T”
       .BYTE  04Bh      ;”R”
       .BYTE  082h      ;”A”
       .BYTE  072h      ;”N”
       .BYTE  0CAh      ;”S”
       .BYTE  0B2h      ;”M”
       .BYTE  093h      ;”I”
       .BYTE  02Bh      ;”T”
       .BYTE  02Bh      ;”T”
       .BYTE  0A3h      ;”E”
       .BYTE  04Bh      ;”R”
       .BYTE  0B1h      ;”CR1”
TXT_78 .BYTE  050h      ;”CR2”
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Interrupt vectors
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
       .sect  ”Int_Vect”, 0FFE0h
       .word   RESET                  ; P0.2 t0 P0.7
       .word   RESET                  ; Basic Timer
       .word   RXTX_01                ; I/O Port P1
```

```
.word   TX_01       ; I/O Port P2
.word   RESET       ; Timer/Port
.word   RESET       ; no source
.word   RESET       ; no source
.word   RESET       ; no source
.word   RX_01       ; Timer_A/Timer Int.
.word   RESET       ; Timer_A/CAP/CMP Int.
.word   RESET       ; Watchdog
.word   RESET       ; no source
.word   RESET       ; P0.1
.word   RESET       ; P0.0
.word   RESET       ; NMI, Osc. fault
.word   RESET       ; POR, ext. Reset, Watchdog
```