;======================================================================

;

; Author : ADI - Apps www.analog.com/MicroConverter

;

; Date : April 2004

;

; File : i2cslave.asm

;

; Hardware : ADuC845

;

; Description : Code for a slave in an I2C system. This code will

; continuously receive and transmit a byte over the I2C

; interface, then send the received byte out the UART,

; then check if a character had been entered in the UART.

; If so, it will send the ASCII value of the character

; entered to the slave, the next time it transmits a byte.

;

; Reference : Tech Note, uC001: "MicroConverter I2C Compatible

; Interface" find it at www.analog.com/microconverter

;

;======================================================================

$MOD845

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

; DEFINE VARIABLES IN INTERNAL RAM

BYTECNT DATA 30h ; byte counter for I2C routines

INPUT DATA 31h ; data recieved from master

OUTPUT DATA 32h ; data to be transmitted to master

GO BIT 00h ; flag to wait for interrupts

FIRST BIT 01h ; flag to indicate first receive Int

LED EQU P3.4 ; P3.4 drives the LED on eval board

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

; BEGINNING OF CODE

CSEG

ORG 0000h

JMP MAIN

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

; INT0 ISR

ORG 0003h

INC OUTPUT

RETI

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

; I2C ISR

ORG 003Bh

JB I2CTX, SLAVE\_TRANSMITTER

SLAVE\_RECEIVER:

JB FIRST, ENDINT1 ; if first INT then wait for next int

SETB GO ; reception complete

MOV INPUT, I2CDAT ; store data received in INPUT

JMP ENDINT1

SLAVE\_TRANSMITTER:

SETB GO ; transmission complete

MOV I2CDAT, OUTPUT ; move data to be transmitted into I2CDAT

JMP ENDINT2 ;

ENDINT1:

clr i2ci ; Manually clear the I2C interrupt

ENDINT2:

CLR FIRST ; address has already been received

RETI

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

; MAIN PROGRAM

ORG 0060h

MAIN:

; configure UART for 9600 using Timer3

MOV T3CON,#83h

MOV T3FD,#12h

MOV SCON,#52h

;configure and enable interrupts

MOV IEIP2,#01h ; enable I2C interrupt

SETB EX0 ; enable INT0

SETB IT0 ; INT0 edge triggered

SETB EA ; allow all the interrupts

;initialize settings

MOV I2CADD,#044h ; slave address is 44h

MOV I2CCON,#00h ; slave mode (default=>not necessary)

CLR GO ; clear flag to wait for interrupt

; GO is set once data is TX'd or RX'd

SETB FIRST ; FIRST is cleared after receiving the

; first SLAVE receiver interrupt

MOV OUTPUT,#0 ; first byte to be transmitted is 40h

CLR LED

WAITFORDATA:

JNB GO,$ ; ----- wait for i2c interrupt ------

; If it is in receive mode, it will

; wait here for a second interrupt (as

; the first interrupt only contains the

; slave address in I2CDAT).

; In transmit mode the tranmission will

; occur after the first interrupt.

SETB FIRST ; re-initialise flags

CLR GO

JB I2CTX,WAITFORDATA

; if the slave has just transmitted then

; wait to receive a byte

; if the slave has just received then

; send input up the UART

SENDUART:

CPL LED ; LED changes each time one byte has been

; received and another transmitted

MOV A,INPUT ; send value received out the UART

CALL SENDVAL

MOV A,#10

CALL SENDCHAR ; send LF + CR

MOV A,#13

CALL SENDCHAR

JNB RI, WAITFORDATA ; repeat (unless UART data received)

; WHEN UART DATA RECEIVED, MOVE DATA TO I2C OUTPUT...

MOV OUTPUT, SBUF ; update OUTPUT byte to new value

CLR RI ; must clear RI

JMP WAITFORDATA ; back to main loop

;======================================================================

; SUBROUTINES

;======================================================================

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

; SENDCHAR

; sends ASCII value contained in A to UART

SENDCHAR:

JNB TI,$ ; wait 'til present char gone

CLR TI ; must clear TI

MOV SBUF,A

RET

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

; HEX2ASCII

; converts A into the hex character representing the value of A's

; least significant nibble

HEX2ASCII:

ANL A,#00Fh

CJNE A,#00Ah,$+3

JC IO0030

ADD A,#007h

IO0030: ADD A,#'0'

RET

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

; SENDVAL

; converts the hex value of A into two ASCII chars, and then spits

; these two characters up the UART. does not change the value of A.

SENDVAL:

PUSH ACC

SWAP A

CALL HEX2ASCII

CALL SENDCHAR ; send high nibble

POP ACC

PUSH ACC

CALL HEX2ASCII

CALL SENDCHAR ; send low nibble

POP ACC

RET

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

END