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

;

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

;

; Date : Oct 2000

;

; File : i2cslave.asm

;

; Hardware : ADuC812 (commented out = ADuC816/ADuC824)

;

; 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

;

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

$MOD812 ; use ADuC812 & 8052 predefined symbols

;$MOD816

;$MOD824

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

; 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 for all the interrupts

FIRST BIT 01h ; flag for recieve mode interrupt

TR BIT 02h ; flag for transmit mode interrupt

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,TRANSMIT ; if slave-transmitter get data ready

RECEIVE:

CLR TR ; FLAG to indicate that this time

; we were receiving (not trans'ting)

JB FIRST, ENDINT1 ; no need to store the address

SETB GO ; reception complete

MOV INPUT, I2CDAT ; store data recieved in INPUT

JMP ENDINT1 ; ERROR => end interrupt

TRANSMIT:

SETB TR ; FLAG to indicate that this time

; we were transmitting (not receiving)

SETB GO ; transmission complete

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

; JMP ENDINT2 ; Note: On the ADuC824/816 the read or

; write of I2CDAT register

; automatically clears i2ci. If

; I2CI is cleared twice then the

; microconverter will hang.)

ENDINT1:

CLR I2CI ; clear I2C interrupt bit (812 only)

ENDINT2:

CLR FIRST ; address has already been recieved

RETI

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

; MAIN PROGRAM

ORG 0060h

MAIN:

; configure the UART ADuC812

MOV SCON,#52h ; configure UART for 9600baud..

MOV TMOD,#20h ; ..assuming 11.0592MHz crystal

MOV TH1,#-3

SETB TR1

; configure the UART ADuC824/ADuC816

; MOV RCAP2H,#0FFh ; config UART for 9830baud

; MOV RCAP2L,#-5 ; (close enough to 9600baud)

; MOV TH2,#0FFh

; MOV TL2,#-5

; MOV SCON,#52h

; MOV T2CON,#34h

;configure and enable interrupts

MOV IE2,#01h ; enable I2C interrupt

; MOV IEIP2,#01h ; enable I2C interrupt

SETB EX0 ; enable INT0

SETB IT0 ; INT0 edge triggered

SETB EA ; allow all the interrupts

;initialise settings

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

MOV I2CCON,#00h ; slave mode

CLR GO ; clear flag used in the interrupt

MOV OUTPUT,#0 ; TX 0 as default

SETB LED

RESET:

SETB FIRST ; first byte recieved will be the

; address => no need to store it.

JNB GO,$ ; wait for the interrupt. If it is in

; recieve mode, it will wait here for

; a second interrupt, when it

; recieves the data bit

CLR GO ; flag cleared for the next interrupt

JB TR,RESET ; if the slave has just transmitted,

; wait for another interrupt. If it has

; recieved a data byte send it out the UART

SENDUART:

CPL LED ; LED changes each time one byte has been

; recieved and another transmitted

MOV A,INPUT ; send value recieved out the UART

CALL SENDVAL

MOV A,#10

CALL SENDCHAR ; send LF + CR

MOV A,#13

CALL SENDCHAR

JNB RI, RESET ; 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 RESET ; 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