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

;

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

;

; Date : May 2002

;

; File : i2Cmstr.asm

;

; Hardware : ADuC814

;

; Description : Code for a master 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

;

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

$MOD814

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

; DEFINE VARIABLES IN INTERNAL RAM

BITCNT DATA 30h ; bit counter for I2C routines

SLAVEADD DATA 31h ; slave address for I2C routines

INPUT DATA 32h ; data recieved from the slave

OUTPUT DATA 33h ; data to be transmitted to slave

NOACK BIT 00h ; I2C no acknowledge flag

ERR BIT 00h ; I2C error flag

LED EQU P3.4

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

; BEGINNING OF CODE

CSEG

ORG 0000h

JMP MAIN

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

; INT0 ISR

ORG 0003h

INC OUTPUT

RETI

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

; MAIN PROGRAM

ORG 0060h

MAIN:

; configure the UART ADuC812s

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

MOV RCAP2L,#-7 ; (close enough to 9600baud)

MOV TH2,#0FFh

MOV TL2,#-7

MOV SCON,#52h

MOV T2CON,#34h

; enable i2c pins on 812s

MOV CFG814,#01H

; configure & enable interrupts

SETB EX0 ; enable INT0

SETB IT0 ; INT0 edge triggered

SETB EA ; allow all the interrupts

; initialise settings

MOV SLAVEADD,#88H ; clear RW bit

MOV I2CCON,#0A8h ; sets SDATA & SCLOCK, and

; selects master mode

MOV OUTPUT,#0 ; TX 0 as default

CLR NOACK

CLR ERR

RXTXLOOP:

; code for a read mode ( master recieves one byte from slave )

CALL RCVDATA ; sends start bit

; sends address byte

; checks acknowledge

; receives byte into ACC

; checks ACK

; sends stop bit

; code for write mode ( master transmits one byte to slave )

CALL SENDDATA ; sends start bit

; sends address byte

; checks acknowledge

; transmits ACC

; checks ACK

; sends stop bit

; Check for Error message

JB ERR,SENDERR ; if error, send error message

; Transmit received byte (INPUT) up UART to PC (hyperterminal)

MOV A,INPUT ; put value received into ACC

CALL SENDVAL ; send value received out the UART

JMP SKIP

SENDERR:

CALL ERROR ; send error message out the UART

CLR ERR ; clear error flag

SKIP:

MOV A,#10 ; send LF+CR

CALL SENDCHAR

MOV A,#13

CALL SENDCHAR

; Toggle LED (1s delay so that LED can be seen toggle)

MOV A, #10

CALL DELAY

CPL LED

; Check for new OUTPUT

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

; If UART data received, then save to OUTPUT

MOV OUTPUT,SBUF ; update OUTPUT byte to new value

CLR RI ; must clear RI

JMP RXTXLOOP ; back to main loop

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

; SUBROUTINES

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

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

; SENDDATA

; Send all the sequence to the slave (slave address + data (OUTPUT))

SENDDATA:

; send start bit

CALL STARTBIT ; acquire bus and send slave address

; send slave address

MOV A, SLAVEADD

CALL SENDBYTE ; sets NOACK if NACK received

JB NOACK, STOPSEND ; if no acknowledge send stop

; send OUTPUT byte

MOV A, OUTPUT

CALL SENDBYTE ; sets NOACK if NACK received

STOPSEND:

CALL STOPBIT ; sends stop bit

JNB NOACK, SENDRET ; if slave sends NACK send error

SETB ERR ; sets the error flag

SENDRET:

RET

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

; RCVDATA

; receives one or more bytes of data from an I2C slave device.

RCVDATA:

INC SLAVEADD ; Set RW for reception

; send start bit

CALL STARTBIT ; acquire bus and send slave address

; send slave address

MOV A, SLAVEADD

CALL SENDBYTE ; sets NOACK if NACK received

DEC SLAVEADD ; returns SLAVEADD to 88h (after INC)

JB NOACK, STOPRCV ; Check for slave not responding.

CALL RCVBYTE ; Receive next data byte.

MOV INPUT,A ; Save data byte in buffer.

STOPRCV:

CALL STOPBIT

JNB NOACK, RCVRET ; if slave sends NACK send error

SETB ERR ; sets the error flag

RCVRET:

RET

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

; STARTBIT

; Sends the start bit to initiate an I2C communication

STARTBIT:

SETB MDE ; enable SDATA pin as an output

CLR NOACK

CLR MDO ; low O/P on SDATA

CLR MCO ; start bit

RET

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

; STOPBIT

; Sends the stop bit to end an I2C transmission

STOPBIT:

SETB MDE ; to enable SDATA pin as an output

CLR MDO ; get SDATA ready for stop

SETB MCO ; set clock for stop

SETB MDO ; this is the stop bit

RET

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

; SENDBYTE

; Send 8-bits in ACC to the slave

SENDBYTE:

MOV BITCNT,#8 ; 8 bits in a byte

SETB MDE ; to enable SDATA pin as an output

CLR MCO ; make sure that the clock line is low

SENDBIT:

RLC A ; put data bit to be sent into carry

MOV MDO,C ; put data bit on SDATA line

SETB MCO ; clock to send bit

CLR MCO ; clear clock

DJNZ BITCNT,SENDBIT ; jump back and send all eight bits

CLR MDE ; release data line for acknowledge

SETB MCO ; send clock for acknowledge

JNB MDI,NEXT ; this is a check for acknowledge

SETB NOACK ; no acknowledge, set flag

NEXT: CLR MCO ; clear clock

RET

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

; RCVBYTE

; receives one byte of data from an I2C slave device. Returns it in A

RCVBYTE:

MOV BITCNT,#8 ; Set bit count.

CLR MDE ; to enable SDATA pin as an input

CLR MCO ; make sure the clock line is low

RCVBIT:

SETB MCO ; clock to recieve bit

CLR MCO ; clear clock

MOV C,MDI ; read data bit into carry.

RLC A ; Rotate bit into result byte.

DJNZ BITCNT,RCVBIT ; Repeat until all bits received.

; recieved byte is in the accumulator

SETB MDE ; Data pin =Output for NACK

SETB MDO ; Send NACK (always send NACK for

; last byte in transmission)

SETB MCO ; Send NACK clock.

CLR MCO

RET

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

; DELAY

; DELAY ROUTINE FOR THE ADuC812/ADuC816/ADuC824

DELAY: ; Delays by 100ms \* A

; ADuC812 100ms based on 11.0592MHz Core Clock

; ADuC824 100ms based on 1.573MHz Core Clock

MOV R2,A ; Acc holds delay variable

;DLY0: MOV R3,#200 ; Set up delay loop0

;DLY1: MOV R4,#229 ; Set up delay loop1

DLY0: MOV R3,#22H ; Set up delay loop0

DLY1: MOV R4,#0FFH ; Set up delay loop1

DJNZ R4,$ ; Dec R4 & Jump here until R4 is 0

; wait here for 131\*15.3us=2ms

DJNZ R3,DLY1 ; Dec R3 & Jump DLY1 until R3 is 0

; Wait for 50\*2ms

DJNZ R2,DLY0 ; Dec R2 & Jump DLY0 until R2 is 0

; wait for ACC\*100ms

RET ; Return from subroutine

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

; ERROR

; this subroutine is run if a NACK is received from the slave

ERROR:

MOV A,#45h

CALL SENDCHAR ; send the letter E out the UART

RET

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

; 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