;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;

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

;

; Date : 12 October 1999

;

; File : UARTIO.hex

;

; Hardware : any 8051 based microcontroller or MicroConverter

;

; Description : standard UART I/O subroutines. total size of this

; code when assembled is 155 bytes. routines for use

; external to this file are:

;

; SENDSTRING - sends a string of characters

; SENDCHAR - sends a single character

; SENDVAL - sends a byte as 2 ASCII characters

; HEX2ASCII - converts from HEX to ASCII

; ASCII2HEX - converts from ASCII to HEX

; GETCHAR - gets a single character

; GETVAL - gets a byte as 2 ASCII characters

;

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

; SENDSTRING

SENDSTRING: ; sends ASCII string to UART starting at location

; DPTR and ending with a null (0) value

PUSH ACC

PUSH B

CLR A

MOV B,A

IO0010: MOV A,B

INC B

MOVC A,@A+DPTR

JZ IO0020

CALL SENDCHAR

JMP IO0010

IO0020: POP B

POP ACC

RET

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

; SENDCHAR

SENDCHAR: ; sends ASCII value contained in A to UART

JNB TI,$ ; wait til present char gone

CLR TI ; must clear TI

MOV SBUF,A

RET

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

; SENDVAL

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.

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

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

; HEX2ASCII

HEX2ASCII: ; converts A into the hex character representing the

; value of A's least significant nibble

ANL A,#00Fh

CJNE A,#00Ah,$+3

JC IO0030

ADD A,#007h

IO0030: ADD A,#'0'

RET

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

; ASCII2HEX

ASCII2HEX: ; converts A from an ASCII digit ('0'-'9' or 'A'-'F')

; into the corresponding number (0-15). returns C=1

; when input is other than an ASCII digit,

; indicating invalid output (returned as 255).

CLR C

SUBB A,#'0'

CJNE A,#10,$+3

JC IO0050 ; if '0'<=char<='9', return OK

CJNE A,#17,$+3

JC IO0040 ; if '9'<char<'A', return FAIL

SUBB A,#7

CJNE A,#10h,$+3

JC IO0050 ; if 'A'<=char<='F', return OK

CJNE A,#42,$+3

JC IO0040 ; if 'F'<char<'a', return FAIL

SUBB A,#20h

CJNE A,#10h,$+3

JC IO0050 ; if 'a'<=char<='f', return OK..

IO0040: CLR C ; ..else return FAIL

MOV A,#0FFh

IO0050: CPL C

RET

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

; GETCHAR

GETCHAR: ; waits for a single ASCII character to be received

; by the UART. places this character into A.

JNB RI,$

MOV A,SBUF

CLR RI

RET

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

; GETVAL

GETVAL: ; waits for two ASCII hex digits to be received by

; the UART. returns the hex value in A.

PUSH B

PUSH 0

IO0060: CLR RI

CALL GETCHAR ; first nibble

MOV 0,A ; store received char

CALL ASCII2HEX

JC IO0060 ; if not '0' thru 'F', don't accept

SWAP A ; swap nibbles

MOV B,A ; store nibble in B

MOV A,0 ; echo received char

CALL SENDCHAR

IO0070: CLR RI

CALL GETCHAR ; second nibble

MOV 0,A ; store received char

CALL ASCII2HEX

JC IO0070 ; if not '0' thru 'F', don't accept

ORL A,B ; combine nibbles

MOV B,A ; store results in B

MOV A,0 ; echo received char

CALL SENDCHAR

MOV A,B ; final result

POP 0

POP B

RET