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

;

; Author : ADI - Apps

;

; Date : 11/19/99

;

; File : SLAVuart.asm

;

; Hardware : ADuC824

;

; Description : This slave program transmits the numbers 11-20 in

; binary form continuously down the SPI serial port

; after receiving a clock signal.

;

; After the transmission of each byte the incoming

; byte is saved in order at an internal RAM address

; between #40h and #50h.

;

; This program can be used with the master program

; MASTuart.asm (which generates a clock signal for

; the slave)

;

; After the 16 input bytes have been stored in memory

; the values in memory are outputted up the UART to

; the PC where they can be viewed on screen by a

; program such as Hyperterminal. After each

; transmission up the UART the program is delayed for

; 1s before returning from the interrupt. It then

; waits for the next data byte from the SPI port

; which will arrive about 4s later if used with the

; Master program (MASTuart.asm).

;

; The Slave program (SLAVuart.asm) should be started

; after the master program (MASTuart.asm) but within

; the time delay of 5s in order that the slave

; program is synchronised by the first outputted

; clock of the master.

;

; The clock is inputted at sclock (pin 26)

; The data is outputted at MISO (pin 14)

; The data is inputted at sdata/MOSI (pin 27)

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

;

$MOD824 ;Use 8052 predefined Symbols

LED EQU P3.4

FLAG BIT 00H

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

; BEGINNING OF CODE

CSEG

ORG 0000H

JMP MAIN

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

; SPI INTERRUPT ROUTINE

ORG 003BH

CLR FLAG ; Clear flag to leave LOOP2

MOV @R1, SPIDAT ; move input into memory

INC R1 ; increment memory location so new

; data is stored in new address

CJNE R1, #50H, CONT ; reset memory location to 40h when

; memory location reaches 50h saving

; 16 bytes of data

CALL SNDUART

CONT: RETI

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

ORG 0060H ; Start code at address above interrupts

MAIN: ; Main program

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

MOV SPICON,#24h ; Initialise SPICON to have

; -Enable SPI serial port

; -slave mode select

; -CPOL=0, clk idling low

; -CPHA=1

; note: it is important to have CPHA in the master and the slave

; program equal, otherwise uncertainty will exist, as the input

; will be measued during its change of state, and not is at

; its final value.

MOV IEIP2, #01h ; Enable SPI interrupt

SETB EA ; Enable interrupts

MOV R1, #40h ; initialise R1 to 40 to store the

; input data from memory location 40

MOV R0, #0AH ; initialise R0 to 10

TRNSMT:

INC R0

MOV SPIDAT, R0 ; transmit the current value on R0

SETB FLAG ; set flag so that we wait here until

; the spi interrupt routine clears

; the FLAG

JB FLAG, $ ; stay here until flag is cleared

; by interrupt

; check if R0 is equal to 20. If so the number 20 has been

; transmitted and we should reset R0 to 10 to start transmission

; from 11 again.

MOV A, R0

CJNE A, #14H, TRNSMT ; if R0 is not 20, jump to TRNSMT

MOV R0, #0AH ; if R0=20 make R0=10 & jump to TRNSMT

JMP TRNSMT

; Transmit the values in locations 40h->50h up the UART wait for

; 1 seconds and then transmit and receive values to/from the Master

; again down the SPI port.

SNDUART:

CPL LED ;CPL LED with each transmission

MOV DPTR, #TITLE

CALL SENDSTRING ; write title block on screen

MOV R1, #40h ; move value at address 40 into R2

MOV A, @R1

MOV R2, A

NEXT: ; Put new value on a new line

MOV A, #10 ; Transmit a linefeed (= ASCII 10)

CALL SENDCHAR

MOV A, #13 ;Transmit a carriage return (=ASCII 13)

CALL SENDCHAR

MOV A, R2 ; Transmit R2 i.e. value @ address R1

CALL SENDVAL

INC R1 ; Increment address

MOV A, @R1

MOV R2, A ; R2 holds the value @ addrR1

MOV A, R1 ; Check if at address 50h

CJNE A, #50h, NEXT ; if not jump to Next

JMP WAIT1S ; if so wait 1s and repeat

WAIT1S: MOV A, #10 ; wait for time less than master to

; synchronise with the master

CALL DELAY

MOV R1, #40h ; store new inputs at address 40h again

RET

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

; 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

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

; DELAY

DELAY: ; Delays by 100ms \* A

; 100mSec based on 1.573MHZ Core Clock

MOV R2,A ; Acc holds delay variable

DLY0: MOV R3,#50 ; Set up delay loop0

DLY1: MOV R4,#131 ; 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

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

TITLE: DB 10,10,13,'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_',10,13

DB 'Analog Devices MicroConverter ADuC824',10,13

DB ' SPI SLAVE Demo Routine',10,13

DB ' Data Stored in Memory in Hex Form',10,13,0

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

END