106061151 劉安得 HW3

*1. (Software Design) (30%)*

*Serial communication*

1. *Write a main program that will first initialize the 8051 serial port in* ***mode 1*** *with 9600 baud, odd parity, non-interrupt-driven, and then call a subroutine called* ***INLINE*** *to read from serial port a line of ASCII codes terminated with a carriage return character code (0DH) and store it in a line buffer with an extra null character (00H) for its termination. It will then call a subroutine* ***OUTSTR*** *to send out the received line in the line buffer via serial port. Assume the line buffer is residing in the* ***external data memory*** *beginning at address* ***60H****with the length of the line buffer limited to 40 bytes. When calling both subroutines, the* ***R0*** *is used as the pointer to the line buffer.*

CR EQU 0DH ;ASCII code for carriage return

MOV SCON, #70H ;mode 1, SM2=1, REN=1

ORL PCON, #80H ;SMOD = 1

MOV TMOD, #20H ;T1 mode 2

MOV TH1, #0F9H ;9.6 Kbps, autoreloadvalue

SETB TR1 ;start T1

LOOP: MOV R0, #60H

LCALL INLINE

MOV R0, #60H

LCALL OUTSTR

LJMP LOOP

INLINE: LCALL INCHR ;read a character

CJNE R0, #87H, BELOW40

LJMP NULL

BELOW40: CJNE A, #CR, INPUT ;end of string?

NULL: MOV A, #00H ;terminate string by null

MOVX @R0, A ;save char in buffer memory

RET

INPUT: MOVX @R0, A ;save char in buffer memory

INC R0 ;move buffer pointer

AJMP INLINE ;continue

OUTSTR: MOVX A, @R0 ;get char from buffer

INC R0 ;move buffer pointer

CJNE A, #00H, OUTPUT ;end of string?

MOV A, #CR ;yes

LCALL OUTCHR ;send a CR character

RET

OUTPUT: LCALL OUTCHR ;send char

AJMP OUTSTR ;continue

INCHR: JNB RI, $ ;wait for character

CLR RI

MOV A, SBUF ;get received byte

CLR A.7 ;strip off parity bit

RET

OUTCHR: MOV C, P ;parity bit in C

CPL C ;change to odd parity

MOV A.7, C ;add to character

JNB TI, $ ;TX empty?

CLR TI

MOV SBUF, A ;send character

CLR A.7 ;strip off parity bit

RET ;A unchanged

1. *Repeat (a) with serial port enabled in interrupt-driven mode.*

CR EQU 0DH ;ASCII code for carriage return

ORG 0000H

LJMP Main

ORG 0023H

LJMP SPISR ;serial port vector

ORG 0030H ;main entry point

Main: MOV SCON, #70H ;mode 1, SM2=1, REN=1

ORL PCON, #80H ;SMOD = 1

MOV TMOD, #20H ;T1 mode 2

MOV TH1, #0F9H ;9.6 Kbps, autoreloadvalue

SETB TR1 ;start T1

MOV IE,#90H ;enable SP interrupt

MOV R0, #60H

SJMP $ ;do nothing

SPISR: JNB RI, OUTSTR

LCALL INCHR ;read a character

CJNE R0, #87H, BELOW40

LJMP NULL

BELOW40: CJNE A, #CR, INPUT ;end of string?

NULL: MOV A, #00H ;terminate string by null

MOVX @R0, A ;save char in buffer memory

MOV R0, #60H

SETB TI

RETI

INPUT: MOVX @R0, A ;save char in buffer memory

INC R0 ;move buffer pointer

RETI

OUTSTR: MOVX A, @R0 ;get char from buffer

INC R0 ;move buffer pointer

CJNE A, #00H, OUTPUT ;end of string?

MOV A, #CR ;yes

MOV R0, #60H

OUTPUT: LCALL OUTCHR ;send a CR character or char

RETI

INCHR: CLR RI

MOV A, SBUF ;get received byte

CLR A.7 ;strip off parity bit

RET

OUTCHR: MOV C, P ;parity bit in C

CPL C ;change to odd parity

MOV A.7, C ;add to character

CLR TI

MOV SBUF, A ;send character

CLR A.7 ;strip off parity bit

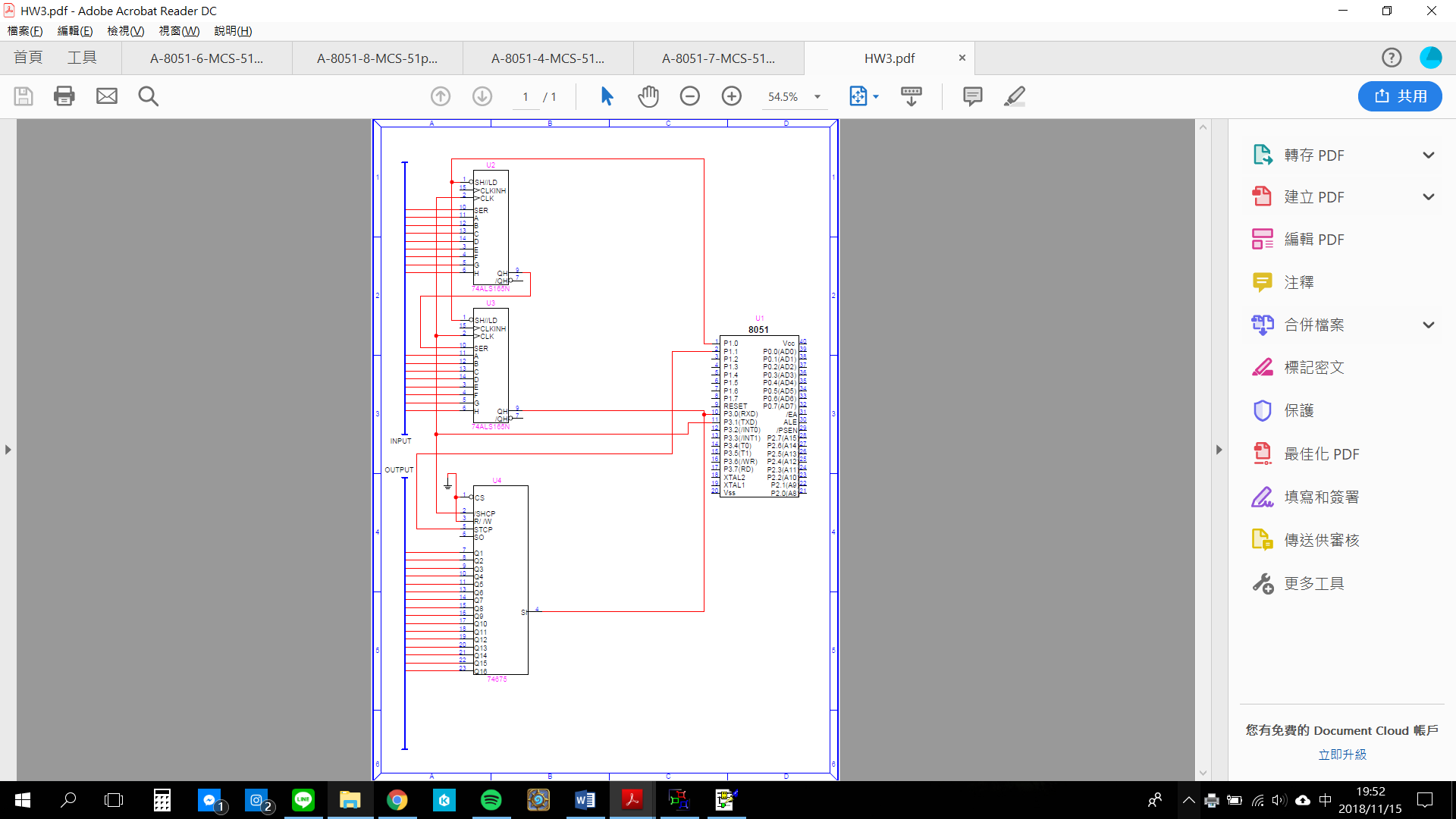
RET ;A unchanged

*2. (Hardware schematic design and software design) (30%)*

*I/O expansion using* ***shift registers of serial por****t*

*We want to use serial port in* ***mode 0*** *to expand 8051 I/O pins. Assume that we want to have 16 extra output pins with each bit value stored in bit-addressable memory 20H, 21H and 16 extra input pins read into bit-addressable memory 22H, 23H using serial-in-parallel-out and parallel-in-serial-out shift registers connected to 8051 serial port for I/O expansion, respectively. The I/O pins should be updated at least 100 times per second.*

1. *Please pick the proper TTL shift registers and ICs whenever needed, study their datasheets, and draw a simplified schematic diagram showing the necessary connection of them with 8051.*



1. *Design an 8051 program for fulfilling this task by first designing your flow chart, examining its logical validity, then designing the codes.*

…………..

…………..

Parallel-in serial-out

Serial-in parallel-out

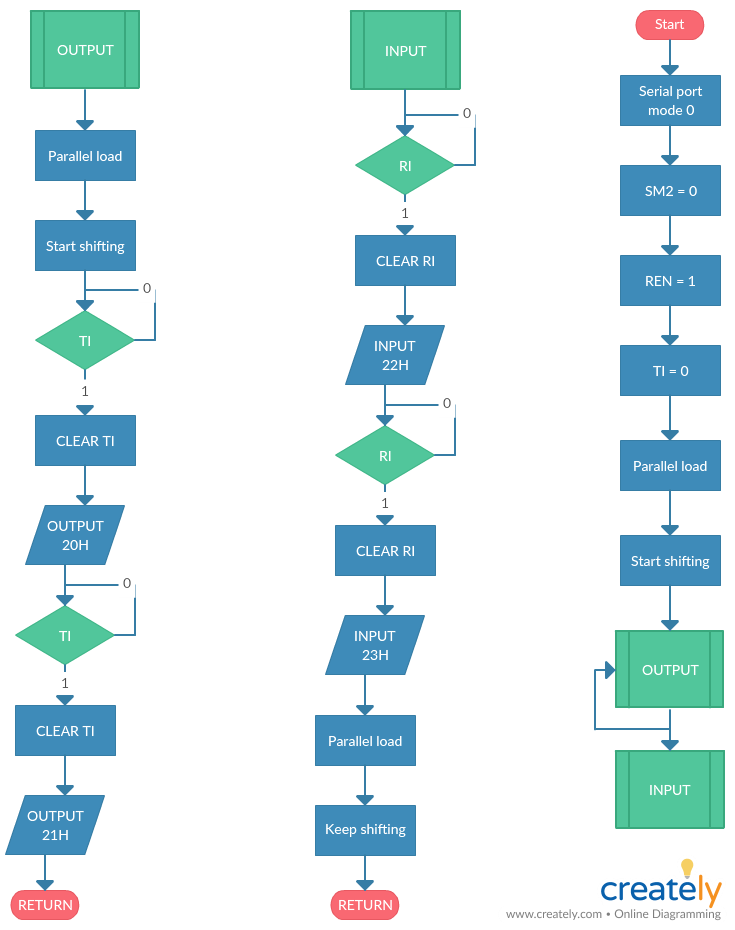
Tx

Rx

16 inputs

16 outputs

8051



MOV SCON, #12H ;mode 0, SM2=0, REN=1

SETB P1.0

CLR P1.0

OUTPUT: JNB TI, $ ;TX empty?

CLR TI

SETB P1.1

CLR P1.1

MOV SBUF, 20H

JNB TI, $ ;TX empty?

CLR TI

MOV SBUF, 21H

INPUT: JNB RI, $

CLR RI

MOV 22H, SBUF

JNB RI, $

CLR RI

MOV 23H, SBUF

SETB P1.0

CLR P1.0

LJMP OUTPUT

*3. (Hardware and software design) (40%)*

*I/O expansion using* ***multiprocessor communication***

8051 Slave 1

P0 P1 P2

Tx Rx

8051 Slave 2

P0 P1 P2

Tx Rx

8051 Slave 3

P0 P1 P2

Tx Rx

8051 Slave 4

P0 P1 P2

Tx Rx

Tx

Rx

8051 Master

*As shown above, we have an 8051 master and 4 8051slaves connected in multiprocessor communication configuration using serial ports. Each slave has its id as shown. Slave 1 and 2 provide 6 x 8 = 48 input pins; slave 3 and 4 provide 48 output pins.**The states of those 96 I/O pins are to be stored at master’s bit-addressable memory starting at bit address 00H ~ bit address 5FH (i.e., byte address 20H ~ 2BH). Please design codes for master and each slave (slave 1 and 2 may have similar code, so do slave 3 and 4) to fulfill the task. You should provide:*

* *a subroutine to initialize master and slave 8051’s and configure the input, output ports,*
* *a subroutine to read in all input ports and placed them in master’s internal bit-addressable bits,*
* *a subroutine to output from masters bit-addressable bits to slave 3 and 4 ports, and*
* *a main program that can continuously handle those I/O pins.*

1. *Design flowcharts or pseudo codes for Init subroutine, Input subroutine, and Output subroutine for 8051 master and slaves.*
2. master

INIT()

BEGIN

[serial port in mode 2]

[sm2 = 0]

[ren = 1]

[ti = 1]

[smod = 0]

RETURN()

END

INPUT()

BEGIN

WHILE [slave\_id <= 6] DO BEGIN

[wait for ti]

[clear ti]

[transmit address byte (slave\_id)]

[tb8 = 1]

[wait for ri]

[clear ri]

[receive data byte]

[slave\_id ++]

END

RETURN()

END

OUTPUT()

BEGIN

WHILE [slave\_id <= 12] DO BEGIN

[wait for ti]

[clear ti]

[transmit address byte (slave\_id)]

[tb8 = 1]

[wait for ti]

[clear ti]

[transmit data byte]

[tb8 = 0]

[slave\_id ++]

END

RETURN()

END

1. slave\_input

INIT()

BEGIN

[serial port in mode 2]

[sm2 = 1]

[ren = 1]

[ti = 1]

[smod = 0]

RETURN()

END

INPUT()

BEGIN

[wait for ri]

[clear ri]

[receive address byte]

IF [address == slave\_id] THEN BEGIN

[wait for ri]

[clear ri]

[transmit data byte from P0,1,2]

END

RETURN()

END

1. slave\_output

INIT()

BEGIN

[serial port in mode 2]

[sm2 = 1]

[ren = 1]

[ti = 0]

[smod = 0]

RETURN()

END

INPUT()

BEGIN

[wait for ri]

[clear ri]

[receive address byte]

IF [address == slave\_id] THEN BEGIN

[clear sm2]

[wait for ri]

[clear ri]

[transfer data byte to P0,1,2]

[set sm2]

END

RETURN()

END

1. *Finally, based on (a), design the main and serial port programs for the master and each slave 8051s.*
2. Master

Init: MOV SCON, #52H ;mode 2, SM2=0, REN=1

ANL PCON, #7FH ;SMOD = 0

LOOP: MOV R0, #20H

LJMP INPUT

LJMP OUTPUT

LJMP LOOP

INPUT: CJNE R0, #26H, ABOVE26

JNB TI, $ ;TX empty?

CLR TI

MOV SBUF, R0

SETB TB8

JNB RI, $

CLR RI

MOV @R0, SBUF ;get received byte

INC R0

LJMP INPUT

ABOVE26: RET

OUTPUT: CJNE R0, #2CH, ABOVE2C

JNB TI, $ ;TX empty?

CLR TI

MOV SBUF, R0

SETB TB8

JNB TI, $ ;TX empty?

CLR TI

MOV SBUF, @R0

CLR TB8

INC R0

LJMP OUTPUT

ABOVE2C: RET

1. Slave\_1

Init: MOV SCON, #72H ;mode 2, SM2=1, REN=1

ANL PCON, #7FH ;SMOD = 0

LOOP: LJMP INPUT

LJMP LOOP

INPUT: JNB RI, $

CLR RI

MOV A, SBUF ;get received byte

CJNE A, #20H, SKIP1

JNB TI, $ ;TX empty?

CLR TI

MOV SBUF, P0 ;get received byte

RET

SKIP1: CJNE A, #21H, SKIP2

JNB TI, $ ;TX empty?

CLR TI

MOV SBUF, P1 ;get received byte

RET

SKIP2: CJNE A, #22H, EXIT

JNB TI, $ ;TX empty?

CLR TI

MOV SBUF, P2 ;get received byte

EXIT: RET

1. Slave\_2

Init: MOV SCON, #72H ;mode 2, SM2=1, REN=1

ANL PCON, #7FH ;SMOD = 0

LOOP: LJMP INPUT

LJMP LOOP

INPUT: JNB RI, $

CLR RI

MOV A, SBUF ;get received byte

CJNE A, #23H, SKIP1

JNB TI, $ ;TX empty?

CLR TI

MOV SBUF, P0 ;get received byte

RET

SKIP1: CJNE A, #24H, SKIP2

JNB TI, $ ;TX empty?

CLR TI

MOV SBUF, P1 ;get received byte

RET

SKIP2: CJNE A, #25H, EXIT

JNB TI, $ ;TX empty?

CLR TI

MOV SBUF, P2 ;get received byte

EXIT: RET

1. Slave\_3

Init: MOV SCON, #70H ;mode 2, SM2=1, REN=1

ANL PCON, #7FH ;SMOD = 0

LOOP: LJMP OUTPUT

LJMP LOOP

OUTPUT: JNB RI, $

CLR RI

MOV A, SBUF ;get received byte

CJNE A, #26H, SKIP1

CLR SM2

JNB RI, $

CLR RI

MOV P0, SBUF

SETB SM2

RET

SKIP1: CJNE A, #27H, SKIP2

CLR SM2

JNB RI, $

CLR RI

MOV P1, SBUF

SETB SM2

RET

SKIP2: CJNE A, #28H, EXIT

CLR SM2

JNB RI, $

CLR RI

MOV P2, SBUF

SETB SM2

EXIT: RET

1. Slave\_4

Init: MOV SCON, #70H ;mode 2, SM2=1, REN=1

ANL PCON, #7FH ;SMOD = 0

LOOP: LJMP OUTPUT

LJMP LOOP

OUTPUT: JNB RI, $

CLR RI

MOV A, SBUF ;get received byte

CJNE A, #29H, SKIP1

CLR SM2

JNB RI, $

CLR RI

MOV P0, SBUF

SETB SM2

RET

SKIP1: CJNE A, #2AH, SKIP2

CLR SM2

JNB RI, $

CLR RI

MOV P1, SBUF

SETB SM2

RET

SKIP2: CJNE A, #2BH, EXIT

CLR SM2

JNB RI, $

CLR RI

MOV P2, SBUF

SETB SM2

EXIT: RET