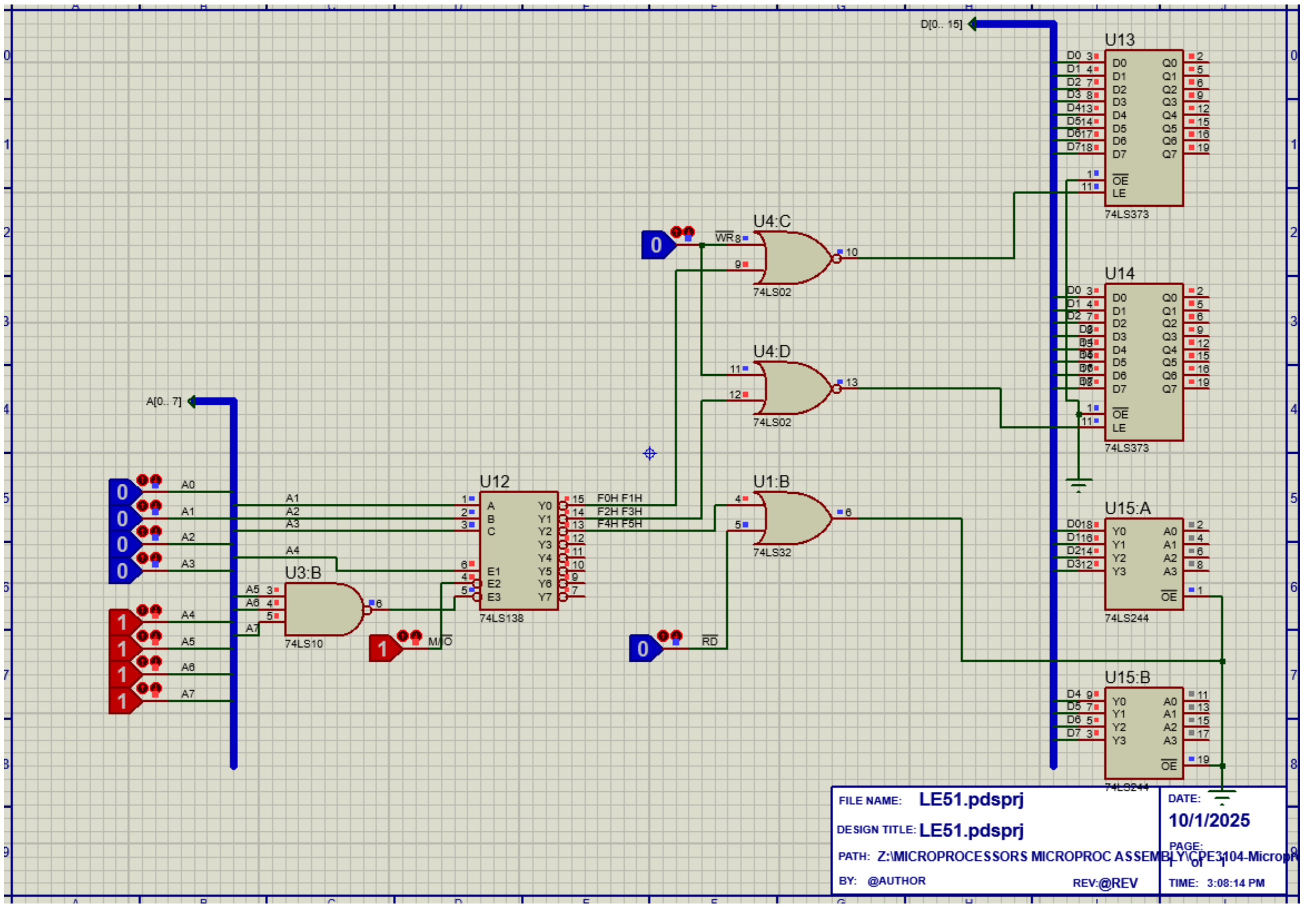
Laboratory Report

Laboratory Exercise No.:	5	**Date Performed:**	Sept. 29, 2025
Laboratory Exercise Title:	I/O Interfacing		
Name of Student:	Arnold Joseph C. Najera Jr.	**Document Version:**	1

Activity #1



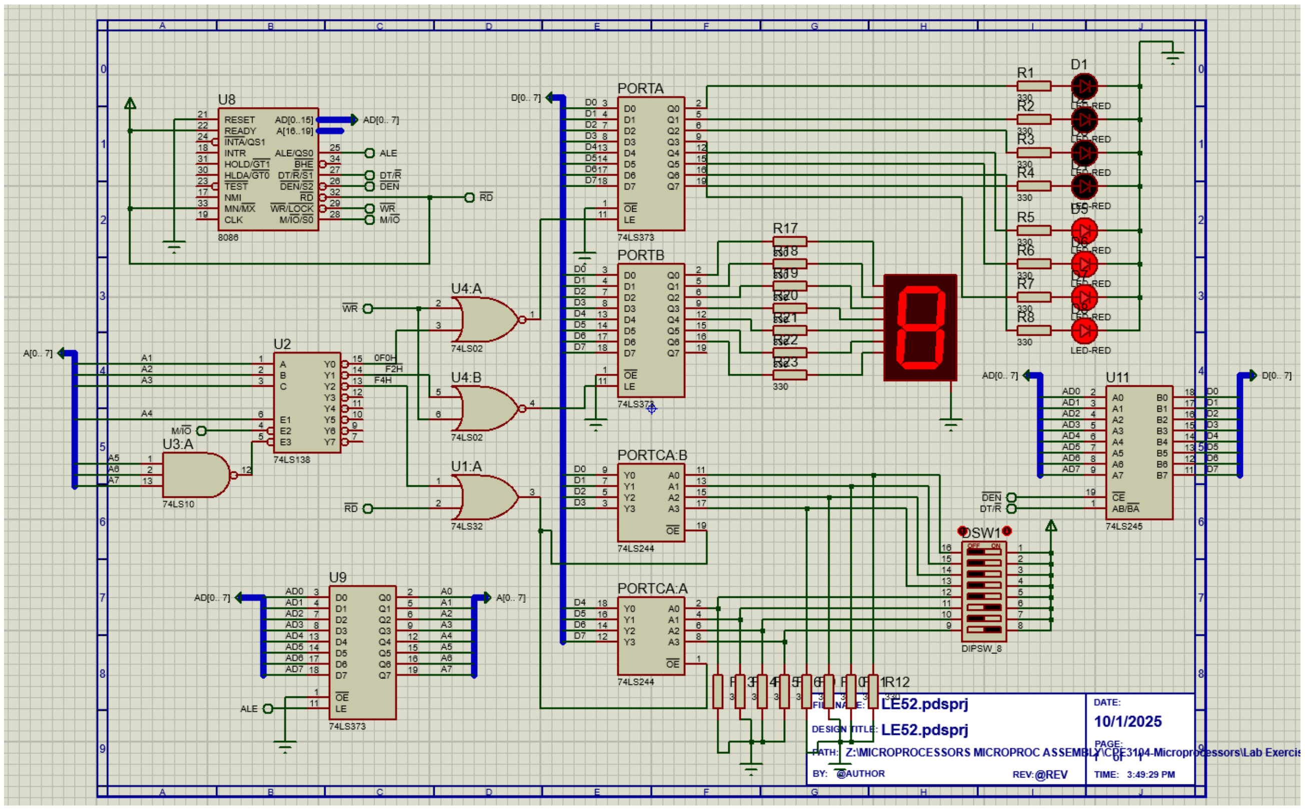
Address ($A_0 – A_7$)	WR’	RD’	M/IO’	I/O Port Enabled
F0H	0	1	0	PORTA(F0H)
F1H	1	0	0	None
F4H	0	1	0	None
F4H	0	1	1	None

F5H	1	0	0	PORTC (F5H)
F3H	0	1	0	PORTB (F3H)
F2H	0	1	1	None
02H	1	0	0	None
65H	0	1	1	None
F6H	1	0	0	None

1. **Observe the data in Table 1. What is the role of the control lines , and in I/O address decoding?**
   The M/IO’, WR’, and RD’ lines are used in I/O address decoding. M/IO’ determines whether the operation is to memory or I/O, M/IO’ = 1 is memory and M/IO’ = 0 is I/O. WR’ is active low and allows the processor to write to the selected device or memory location. RD’ is also active low and allows the processor to read from the device or memory. Together these lines ensure the CPU does the right operation at the right location.
2. **What do you think is the purpose of the latches and buffers?**
   Latches and buffers are also used in I/O interfacing. Latches are used to hold or store output data from the CPU, keeping the data stable until the I/O device can use it. Buffers are used on the input side to drive signals and control the flow of data, so the CPU can safely get information from external devices without being overwhelmed. In short latches are for holding data outputs and buffers are for data inputs.

3. **Based on the decoder circuit and I/O address range, is the I/O system “memory mapped” or “isolated”? Why?**
   Based on the decoder circuit and I/O address range the I/O system is isolated I/O not memory mapped I/O. This is because specific I/O addresses like F0H, F3H and F5H are assigned to ports and these addresses are only accessed when M/IO’ is set to 0 which specifically selects the I/O space. In memory mapped I/O devices share the same address space as memory and no separate M/IO’ control is needed. Since the given system uses M/IO’ to distinguish I/O operations from memory operations it is clearly an isolated I/O system.

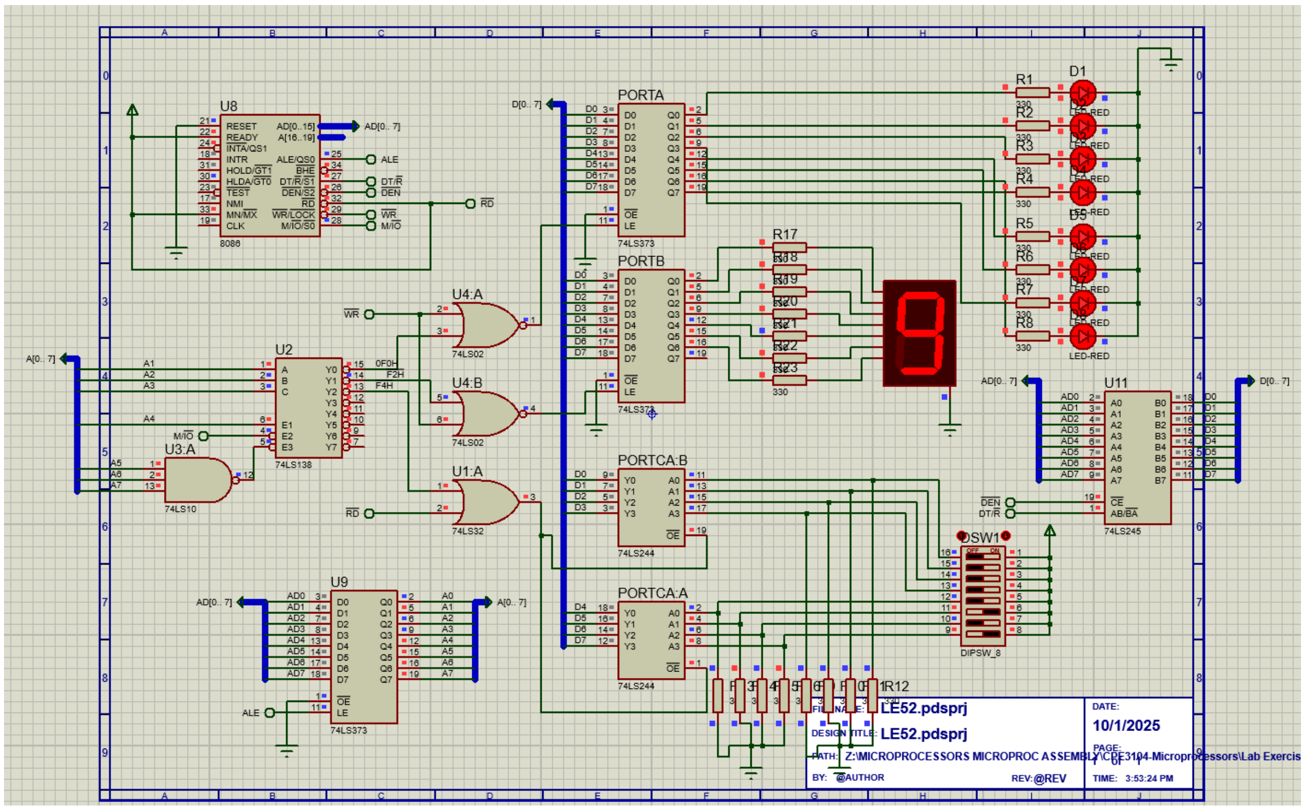
Activity #2



The program below will send an 8-bit data to the I/O port PORTA with address F0H.

```
Shell
DATA SEGMENT
    PORTA EQU 0F0H ; PORTA address
    PORTB EQU 0F2H ; PORTB address
    PORTC EQU 0F4H ; PORTC address
DATA ENDS

CODE SEGMENT
    MOV AX, DATA
    MOV DS, AX ; set the Data Segment address
    ORG 0000H ; write code below starting at address 0000H

START:
    MOV DX, PORTA ; set port address of PORTA
    MOV AL,11110000B       ; turn PORTA on
    OUT DX, AL ; send 1111000B to PORTA

HERE:
    NOP ; do nothing
    JMP HERE

CODE ENDS
END
```

Modify the program in #3 to send data to PORTB. Send an 8-bit data to display number ‘9’ on the 7-segment display. Compile and run the simulation.



The program below will send an 8-bit data to the I/O port PORTB with address F2H.

```
Shell
DATA SEGMENT
    PORTA EQU 0F0H ; PORTA address
    PORTB EQU 0F2H ; PORTB address
    PORTC EQU 0F4H ; PORTC address
DATA ENDS

CODE SEGMENT
    MOV AX, DATA
    MOV DS, AX          ; set the Data Segment address
    ORG 0000H           ; write code below starting at address 0000H

START:
    MOV DX, PORTB       ; set port address of PORTB
    MOV AL, 01101111B   ; 7-segment code for '9'
    OUT DX, AL          ; send data to PORTB

HERE:
    NOP                 ; do nothing
    JMP HERE            ; infinite loop

CODE ENDS
END
```

Modify the program to read the data from PORTC and display it in PORTA. For example, if the data in PORTC (through the DIP switch) is 25H, then the LEDs would turn on or off correspondingly in PORTA. Compile and run the simulation. Verify the output in PORTA. Change the switch status in PORTC and verify the data output in PORTA.

```Shell
DATA SEGMENT
    PORTA EQU 0F0H ; PORTA address
    PORTB EQU 0F2H ; PORTB address
    PORTC EQU 0F4H ; PORTC address
DATA ENDS

CODE SEGMENT
    MOV AX, DATA
    MOV DS, AX             ; set the Data Segment address
    ORG 0000H              ; code starts at address 0000H

START:
MAIN_LOOP:
    MOV DX, PORTC          ; select PORTC (input from DIP switch)
    IN  AL, DX             ; read data from PORTC into AL

    MOV DX, PORTA          ; select PORTA (LEDs)
    OUT DX, AL             ; send the same data to PORTA

    JMP MAIN_LOOP          ; repeat forever to continuously update

CODE ENDS
END
```

Write an assembly program that will create a running LED pattern (single cycle) on PORTA when the data in PORTC is 01H. When the data in PORTC is 02H, the 7-segment display in PORTB will count from 0-9. Nothing will happen if the data in PORTC is neither 01H or 02H. Compile and run the simulation. See appendix for details.

```Shell
DATA SEGMENT
    PORTA EQU 0F0H
    PORTB EQU 0F2H
    PORTC EQU 0F4H
    TABLE DB 0C0H,0F9H,0A4H,0B0H,099H,092H,082H,0F8H,080H,090H
DATA ENDS

CODE SEGMENT
    ASSUME CS:CODE, DS:DATA

    ORG 0000H

START:
    MOV AX, DATA
    MOV DS, AX

MAIN_LOOP:
    ; --- Read PORTC input ---
    MOV DX, PORTC
    IN  AL, DX

    CMP AL, 01H
    JE  RUNNING_LED      ; if PORTC = 01H

    CMP AL, 02H
    JE  COUNT_DISPLAY    ; if PORTC = 02H

    JMP MAIN_LOOP        ; otherwise do nothing

RUNNING_LED:
    MOV AL, 80H          ; start with 1000 0000b
LED_LOOP:
    MOV DX, PORTA
    OUT DX, AL           ; output LED pattern

    CALL DELAY           ; short delay

    SHR AL, 1            ; shift right
    JNZ LED_LOOP         ; repeat until AL = 00H

    JMP MAIN_LOOP

COUNT_DISPLAY:
    MOV CX, 0AH          ; 10 digits
    MOV SI, OFFSET TABLE
SEG_LOOP:
    MOV AL, [SI]         ; get digit code
    MOV DX, PORTB
    OUT DX, AL           ; output to 7-seg

    CALL DELAY           ; short delay

    INC SI
    LOOP SEG_LOOP

    JMP MAIN_LOOP

DELAY PROC
    MOV BX, 0FFFFH
WAIT1:
    NOP
    DEC BX
    JNZ WAIT1
    RET
DELAY ENDP

CODE ENDS
```

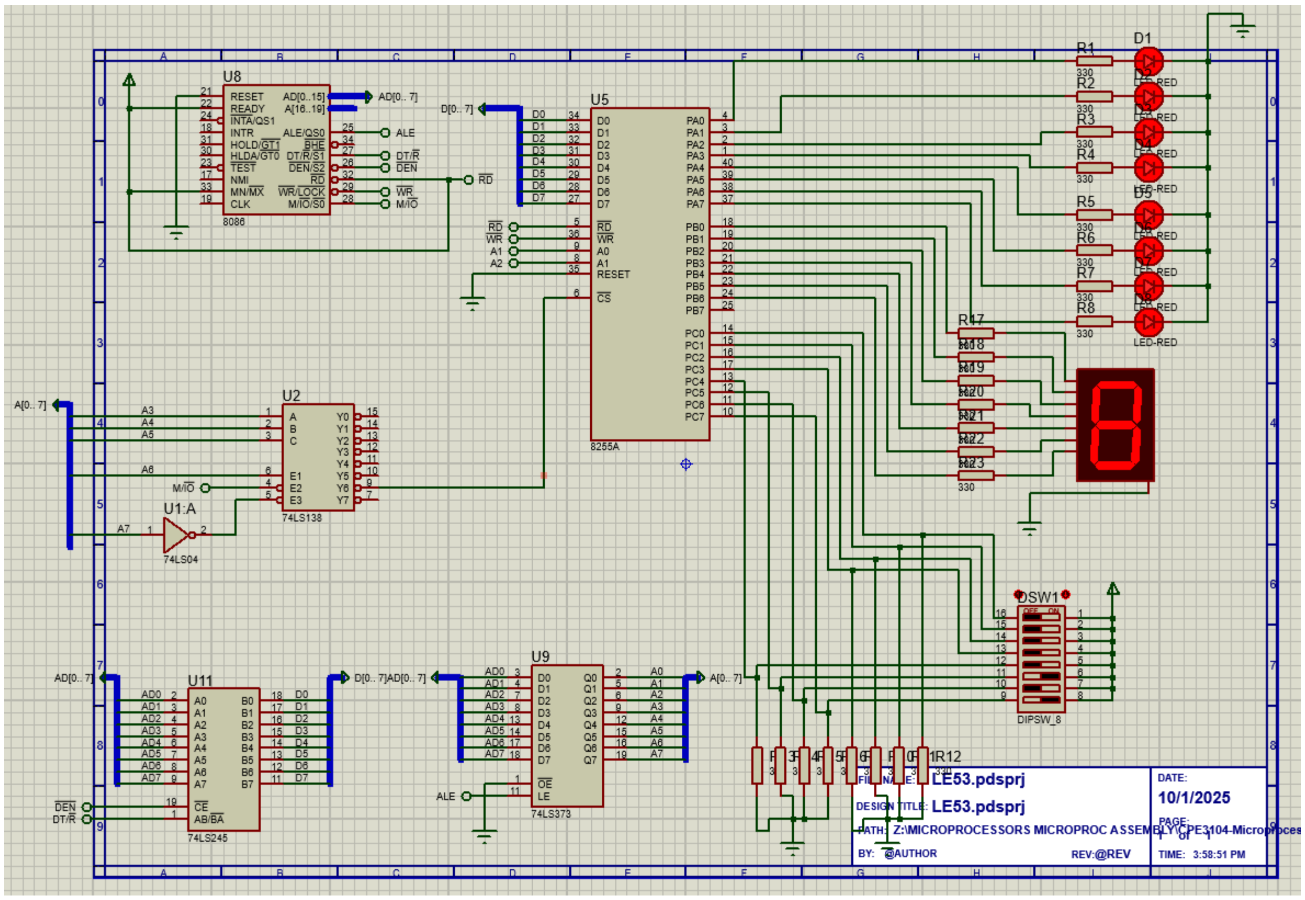
```
END START
```

Activity #3

Address: **PORTA:** F0H **PORTB:** F2H

PORTC: F4H **COM_REG:** F6H

Command Byte: 10001001B (89H)



Write an assembly program to increment the two-digit counter (PORTB & PORTA) when the switch in PORTC is pressed. Upon reset, the value of the counter is "00". When the counter reaches "99", it reverts back to "00" on the next increment. Apply a simple (10 ms delay) software switch debouncing to make it function properly.

```
Shell
DATA SEGMENT
    PORTA   EQU 0F0H        ; PORTA address (LSD)
    PORTB   EQU 0F2H        ; PORTB address (MSD)
    PORTC   EQU 0F4H        ; PORTC address (Switch input)
    COM_REG EQU 0F6H        ; Command Register address

    TABLE DB 0C0H,0F9H,0A4H,0B0H,099H,092H,082H,0F8H,080H,090H
    LSD    DB 0             ; Least significant digit (0-9)
    MSD    DB 0             ; Most significant digit (0-9)
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE, DS:DATA

START:
    MOV AX, DATA
    MOV DS, AX
    MOV DX, COM_REG
    MOV AL, 10001000B       ; PORTA=out, PORTB=out, PC upper=out, PC lower=in
    OUT DX, AL

    MOV BYTE PTR LSD, 0
    MOV BYTE PTR MSD, 0
    CALL DISPLAY

MAIN_LOOP:
    MOV DX, PORTC
    IN  AL, DX
    TEST AL, 01H            ; check PC0 bit
    JZ MAIN_LOOP            ; if 0, not pressed → loop

    CALL DELAY              ; 10 ms debounce delay
    IN  AL, DX              ; read again
    TEST AL, 01H
    JZ MAIN_LOOP            ; if released after debounce, ignore

    INC BYTE PTR LSD
    CMP BYTE PTR LSD, 10
    JB NO_ROLLOVER
    MOV BYTE PTR LSD, 0
    INC BYTE PTR MSD
    CMP BYTE PTR MSD, 10
    JB NO_ROLLOVER
    MOV BYTE PTR MSD, 0     ; reset to 00 after 99

NO_ROLLOVER:
    CALL DISPLAY

WAIT_RELEASE:
    IN  AL, DX
    TEST AL, 01H
    JNZ WAIT_RELEASE        ; stay here until released

    JMP MAIN_LOOP

DISPLAY PROC
    ; Display LSD
    MOV BL, LSD
    MOV SI, OFFSET TABLE
    MOV AL, [SI+BX]
    MOV DX, PORTA
    OUT DX, AL

    ; Display MSD
    MOV BL, MSD
    MOV AL, [SI+BX]
    MOV DX, PORTB
    OUT DX, AL
    RET
DISPLAY ENDP

; ------------------------------------------------
; DELAY: Simple ~10ms software delay
; ------------------------------------------------
DELAY PROC
    MOV CX, 0FFFFH
DELAY_LOOP:
    NOP
    LOOP DELAY_LOOP
    RET
DELAY ENDP

CODE ENDS
END START
```

1. Based on Activity #3, what can you say about the 8255 Programmable Peripheral Interface (PPI)?

   The 8255 Programmable Peripheral Interface (PPI) is a flexible device that allows the microprocessor to communicate with peripheral devices through its three programmable ports. Unlike simple latches and buffers that provide only fixed input or output, the 8255 can be configured by software to operate in different modes, so it's more versatile for applications like controlling LEDs, driving seven segment displays or reading data from switches. This activity shows how the 8255 simplifies interfacing with multiple devices and allows dynamic control of inputs and outputs through programming rather than hardware changes.

2. What do you think are the advantages and disadvantages of using the 8255 from the simple latches and buffers as I/O ports?

   One of the main advantages of the 8255 is its programmability, you can change the role of each port without changing the hardware. It also integrates multiple I/O functions in a single chip, saves space and simplifies the circuit design. It's ideal for systems that require multiple configurable I/O operations or advanced modes like handshaking. But the 8255 also has disadvantages: it's more complex to use because you need to send a control word before operation, it's more expensive and it may be slightly slower than simple latches or buffers. For very basic I/O operations, using simple latches or buffers might be more practical.