Experiment No. 06:

Parallel Interfacing Using the Peripheral Interface Adapter (PIA)

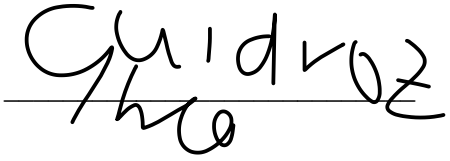
By: Theo Guidroz

Instructor: Dr. Jafar Saniie

TA: Xinrui Yu

ECE 441-001

Acknowledgment: I acknowledge all of the work (including figures and codes) belongs to me and/or persons who are referenced.

Signature : 

I. Introduction

A. Purpose

In this lab, students are introduced to the peripheral adapter IC (PIA, MC6821), the MC68k's synchronous cycle, and the MC68i's interrupt generation mechanism. Students were to design a circuit that allows the user to input a character into the PIA so that the PIA will output the received input back to the user.

B. Background

PIA provides a general-purpose means of interfacing peripheral equipment to the MC68000. The MC6821 features two 8-bit bi-directional peripheral data buses and four control lines. The functionality of the PIA is programmable by the CPU during initialization of the system. Based on what the programmer enters the memory-mapped registers, the data lines can act as an input or output line, etc.

There are six registers on the PIA which fits 3 different categories:

1. Data Direction Register (DDRA or DDRB):

   This register programs each of the eight peripheral data lines (PA0 to PA7 or PB0 to PB7) to act as either input or output. A '1' bit defines its corresponding peripheral data line to be an output, while a '0' bit defines its corresponding peripheral data line to be an input.

2. Peripheral Data Register (PDRA or PDRB):

   The signals form the peripheral data lines are input into this register. The CPU may the read this register to determine the status of the peripheral data lines. On the other hand,

the data written to this register by the CPU will appear on the peripheral data lines that are programmed as output. A ‘1’ written into this register by the CPU causes a ‘high’ level signal to appear on the corresponding peripheral data line. A ‘0’ written into the register causes a ‘low’ level signal to appear on the corresponding peripheral data line.

3. Control Register (CRA or CRB):

   This allows the CPU to configure the operation of the two peripheral control lines, CA1 and CA2 or CB1 and BC2. Control registers also allow the CPU to enable interrupt lines and monitor the status of the interrupt flags. Bit 2 of this register is used along with select lines to determine whether the Data Direction Register or the Peripheral Data Register is to be accesses.

II. Lab Procedure and Equipment List

A. Equipment

The lab will not be done physically

B. Procedure

1. Write the code for the lab.
2. Draw the schematic.
3. Attempt the questions.

III. Results and Analysis

A. Discussion

Programs

```
PORTA EQU $52000
CRA   EQU $52002
PORTB EQU $52004
CRB   EQU $52006

    ORG $980

INBUF DS.B 4
```

```
    ORG $1000

START:
    MOVE.L #$1200,$64    ;Move the Auto Vector 1 routine to $64
    BSR PIAINIT          ;Initialize the PIA
LOOP:
    MOVE.B #247,D7
    TRAP #14            ;Read a character from the keyboard
    MOVE.B D0,PORTB     ;Move the byte to Port B
    BRA LOOP            ;Branch back to the loop
PIAINIT:
    BCLR.B #$2,CRA      ;Select DDRA
    MOVE.B #$00,PORTA   ;Set DDRA to be the input
    BCLR.B #$2,CRB      ;Select DDRB
    MOVE.B #$FF,PORTB   ;Set DDRB to be the output
    MOVE.B #$05,CRA     ;Set bit 0 to get interrupt and bit 2 to point to
PDRA
    MOVE.B #$2C,CRB     ;Set bit 2 to point to PDRB, and set bit 3 and bit 5
to be in Pulse Mode
    ANDI.W #$F8FF,SR    ;Set interrupt level to 0
    RTS

AUTOVEC1:
    OR.W #$0700,SR      ;Set interrupt level to 7
    MOVEM.L A0/A5-A6/D0-D2/D7, -(SP) ; Store registers used
    CLR.L D0            ;Clear D0
    MOVE.B PORTA,D0     ;Read incoming byte
    MOVE.B #248,D7      ;Task number to print data to screen
    TRAP #14            ;Print data
    MOVEM.L (SP)+, A0/A5-A6/D0-D2/D7 ;Restore registers
    ANDI.W #$F8FF, SR   ;Set interrupt level to 0
    RTE

    END    START        ; last line of source
```

Figure 1: PIA program

Questions

1. **A commented listing for the programs of Prelim #3.**

   See above section for program segments and comments.

2. **A schematic diagram of your hardware design.**

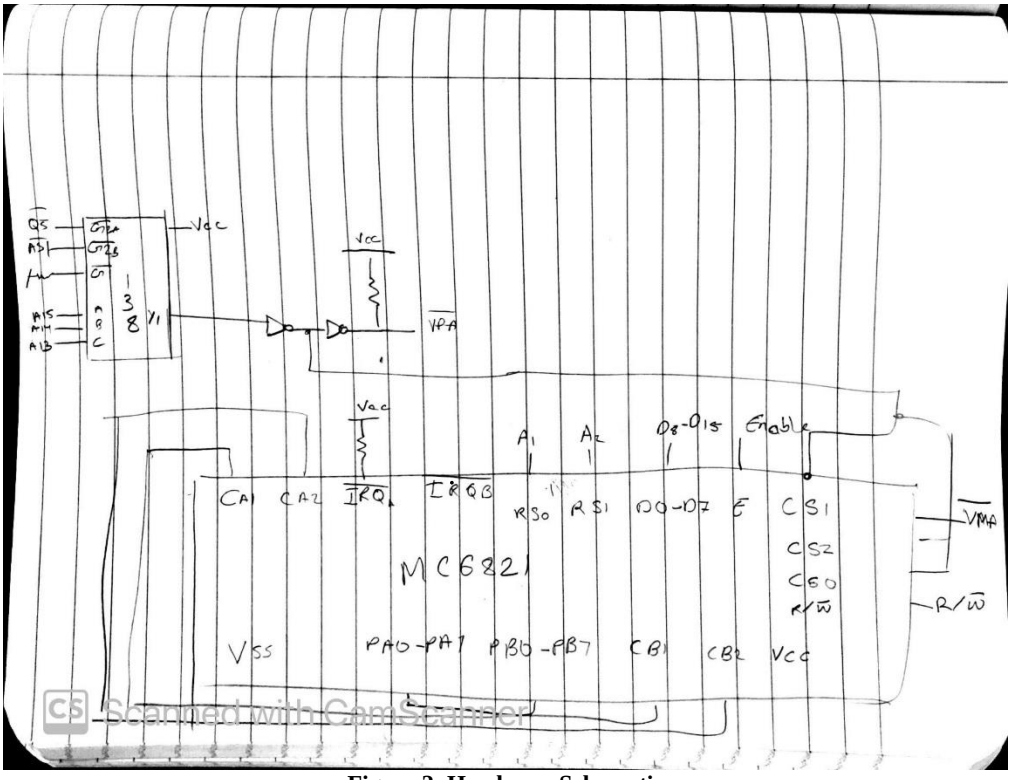


Figure 2: Hardware Schematic

3. **Describe, in detail, the function of each of the programs from the Preliminary Assignment.**

1. The address of the Auto Vector routine was stored at memory address $64.
2. The PIAINIT routine, PORTA and PORTB are set to all 0's and 1's respectively, to make the DDRA the input data direction register and DDRB the output data direction register.
3. Control Register A is configured so that interrupts are enable high-to-low, and Control Register B is configured so pulse mode handshaking is turned on and interrupts are masked.
4. The ISR routine changes the status register to set the interrupt level to 7, so that all interrupts are masked. Additionally, it reads the data inputted into PORTA and prints it to the screen.
5. Sets the interrupt level to 0 and branches back to an infinite loop that reads a user entered character from the terminal into PORTB.

4. **Describe what type of interrupt acknowledge method (user or auto vectored) was used for the PIA in this experiment and why?**

   The PIA used an auto vector interrupt because it cannot generate its own.

5. **Describe how a microprocessor determines which side of the PIA generated an interrupt request?**

   The microprocessor must poll both sides to determine which side generates the interrupt request.

6. **If PIA Control Register A reads $3F, how has the PIA been configured?**

   Control Register A would read (in binary): 0011 1111. This means that there are no pending interrupts, CA2 output is always high, PDRA is selected, and unmasked IRQA1 interrupt.

7. **Discuss three possible applications of the PIA.**
   a. In a gaming system, the PIA can be used to provide 4 joysticks ports to the machine.
   b. It can be used in an old computer to interface the ASCII keyboard and the display.
   c. Facilitate transmission between two independent microcomputer transmissions

8. **Draw and discuss the timing diagram for a read operation using a MC68000 Synchronous Bus Cycle. Indicate the order in which the signals are asserted (and negated) and the timing relationships between them.**

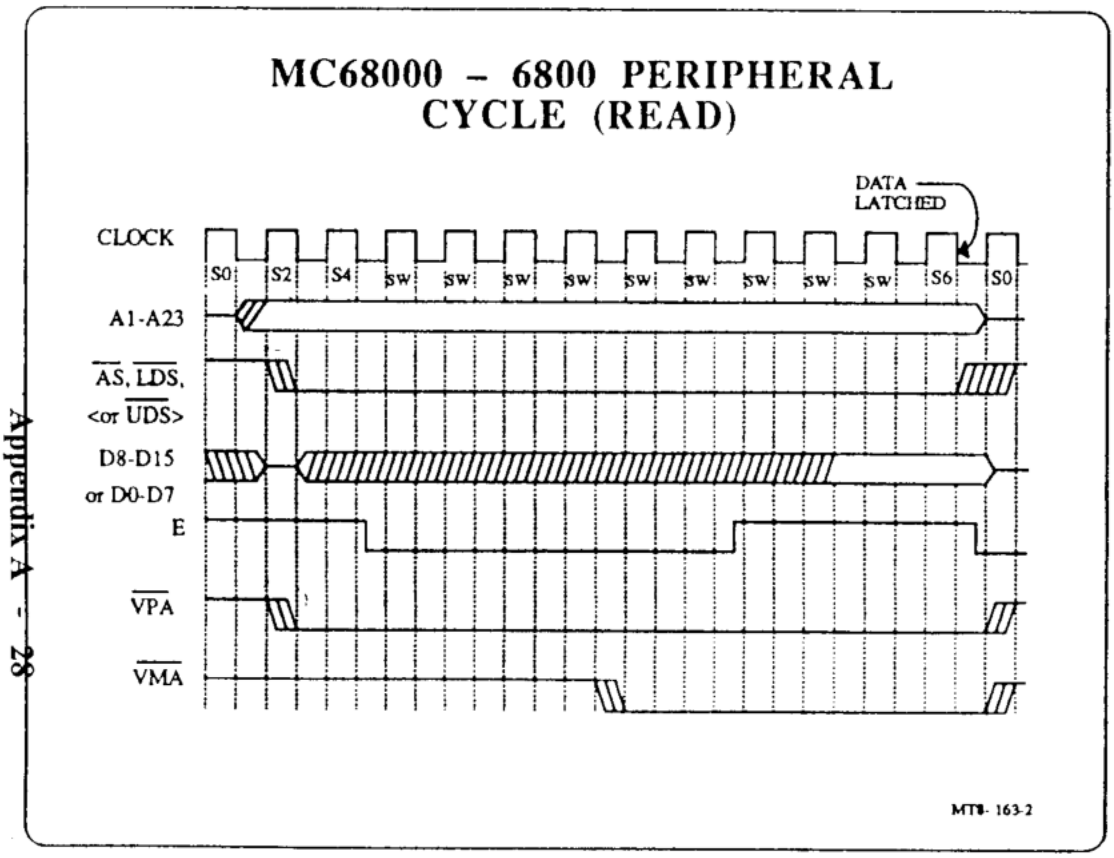


Figure 3: Synchronous Bus Cycle for Read operation

a. Address lines stabilize
b. VPA*, AS*, LDS* or UDS* stabilize
c. VMA* stabilizes
d. Data lines stabilize
e. Data latches
f. All line reset

IV. Conclusions

This lab served as an introduction to the PIA, MC6800's Synchronous Cycle, and the MC68000's Interrupt Generation Mechanism. This concepts were solidified while making the schematic and writing the code.

References

[1] Experiment 6 Lab Manual

[2] ACIA & PIA (Ron Bishop Book)

[3] MC68K User Manual