

# EXPERIMENT 7: PARALLEL INTERFACING USING THE PERIPHERAL INTERFACE ADAPTER (PIA)

Adam Sumner

Illinois Institute of Technology

ECE 441-01

**Lab Date:** April 14th, 2015

**Due Date:** April 28th, 2015

# 1 Introduction

The purpose of this experiment is to introduce the student to the peripheral interface adapter IC (PIA, MC6821), the MC68K's Synchronous Cycle, and the MC68K's Interrupt Generation Mechanism. A circuit will be designed and constructed to allow the user to input a character into the PIA so that the PIA will output the received input back to the user.

## 2 Background

### 2.1 The Peripheral Interface Adapter (PIA)

The Peripheral Interface Adapter or PIA, provides a general purpose means of interfacing peripheral equipment to the MC68000. This integrated circuit is capable of interfacing a microprocessor to peripheral devices through two 8-bit bi-directional peripheral data buses (PA0 to PA7, and PB0 to PB7) and four control lines (CA1, CA2, CB1, and CB2)[1].

The functionality of the PIA is programmable by the CPU during initialization of the system. Each of the data lines can be programmed to perform either as an input or output line, and each of the control lines can be programmed to operate in one of several modes[1].

### 2.2 Internal Architecture

The PIA occupies four consecutive locations in the CPU's memory map. The PIA contains six internal registers, three for Port A, and three for Port B. The Register Select Lines(RS1,RS0) are used to select one of the four registers inside the PIA. Table 1 shows this relationship.

CR2	CRB2	RS1	RS0	PIA Register Selected
0	X	0	0	Data Direction Register, Port A
1	X	0	0	Peripheral Data Register, Port A
X	X	0	1	Control Register, Port A
X	0	1	0	Data Direction Register, Port B
X	1	1	0	Peripheral Data Register, Port B
X	X	1	1	Control Register, Port B

Table 1: Register Selection

### 2.2.1 Registers

1. *Data Direction Register (DDRA or DDRB):*

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

2. *Peripheral Data Register (PDRA or PDRB):*

#### Inputs

The signals from the peripheral data lines are input into this register. The CPU may then read this register to determine the status of the peripheral data lines[1].

#### Outputs

The data written to this register by the CPU will appear on the peripheral data lines that are programmed as outputs. A “1” written into this register by the CPU causes a “high” level signal to appear on the corresponding peripheral data line. Similarly, a “0” written into this register by the CPU causes a “low” level signal to appear on the corresponding peripheral data line[1].

3. *Control Register (CRA or CRB):*

This register allows the CPU to configure the operation of the two peripheral control lines, CA1 and CA2 or CB1 and CB2. This register also allows 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 (RS1, RS0) to determine whether the Data Direction Register or the Peripheral Data Register is to be accessed[1].

## 2.3 MC68000 Synchronous Bus Cycle

In order for the 68000 to interface to 6800 type peripherals, the 68000 modifies its bus cycle to meet the 6800 bus cycle timing requirements whenever a

6800 type device is selected. This feature is possible because both types of processors use memory-mapped I/O. Three signals are used to achieve compatibility. They are:

1. *Enable* (E)

This corresponds to the E signal that exists in 6800 systems. It's the bus clock used by 6800 peripherals to synchronize data transfers. It is a free-running clock that is one tenth the frequency of the 68000's clock signal. It has a 60/40 duty cycle.

2. *Valid Peripheral Address* ( $\overline{VPA}$ )

This input signal informs the 68000 that the address on the bus is the address on the 6800 device and that the bus should conform to the E clock transfer characteristics of the 6800 bus.  $\overline{VPA}$  is derived by decoding the address bus, and qualifying it with *Address Strobe* ( $\overline{AS}$ )

3. *Valid Memory Address* ( $\overline{VMA}$ )

This output signal notifies any 6800 peripherals that the address on the address bus is valid, and that the 68000 is synchronized to the Enable (E) signal. The  $\overline{VMA}$  signal is used as part of the chip select circuitry for the peripheral. This ensures that 6800 type peripherals are selected and unselected at the proper time.

## 3 Equipment/Procedure

### 3.1 Equipment

- SANPER-1 EDUCATIONAL LAB UNIT
- Computer with TUTOR software
- Open Collector Inverter Gate Chip
- NOR Gate Chip
- Decoder Chip
- PIA Chip

## 3.2 Procedure

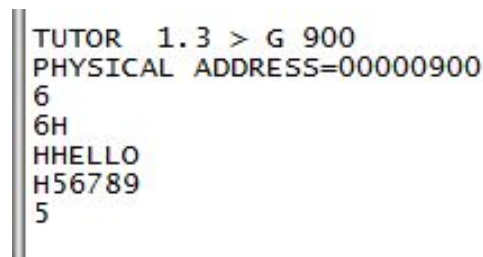
The first step is to draw a detailed schematic diagram of the hardware to be implemented. Once this is done, the designed hardware is to be built on breadboard strips. After this, the software must be designed. An initialization routine to configure the PIA for:

- Port A lines to be inputs
- Port B lines to be outputs
- IRQA interrupt to be enabled and asserted by a high to low transition on CA1
- Port B to use pulse-mode handshaking

must be written, along with an interrupt service routine that reads the peripheral data on PA0  $\rightarrow$  PA7 of Port A and displays the received character on the terminal. Once this is done, a program should be written to continuously read a character from the keyboard and output it to the peripheral data side of Port B of the PIA.

## 4 Results

Below are the results from the experiment. Figure 2 shows the physical implementation of the designed hardware, while Figure 1 shows the I/O in action.

A terminal window showing the output of a program. The text is as follows:

```
TUTOR 1.3 > G 900
PHYSICAL ADDRESS=00000900
6
6H
HHELLO
H56789
5
```

Figure 1: I/O Demonstration

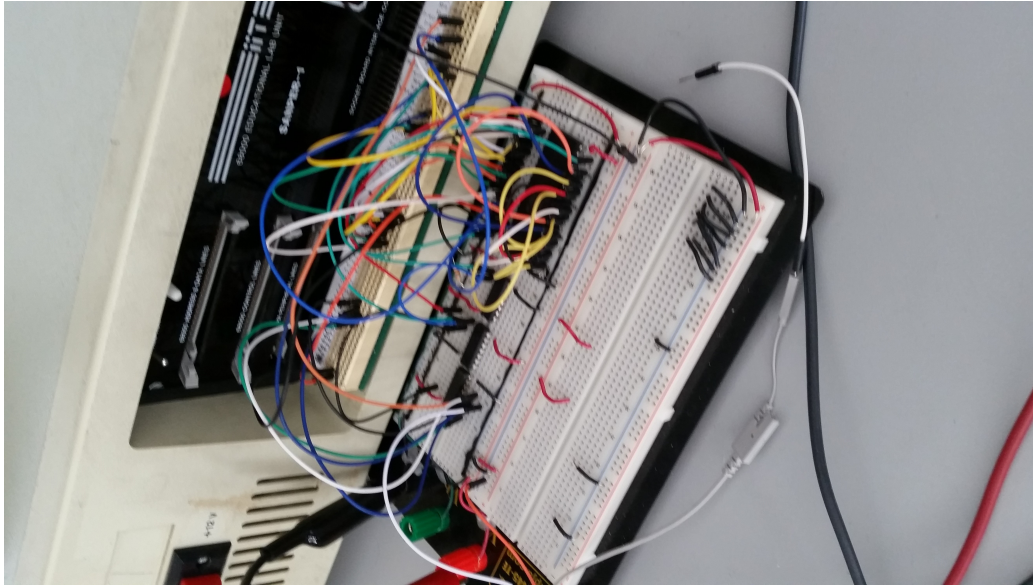


Figure 2: Physical Implementation of Circuit

## 5 Discussion

### 5.1 Answers to Discussion Problems

1. Program is located in Section 7.1
2. The hardware schematic is located in Section 7.2
3. The PIA used an auto vector interrupt because it cannot generate its own.
4. The microprocessor has to poll both sides of the status register to determine which side generated the interrupt request

5.

IRQA1	IRQA2	CA2 Control	DDRA	CA1 Control
0	0	1 1 1	1	11

There are no pending interrupts, CA2 output is always high, PDRA is selected, and rising1 transitions create an unmasked IRQA1 interrupt.

6. The PIA could be used in an old ATARI 800 computer to provide four joystick ports to the machine, it can be used in an old computer to interface the ASCII keyboard and the display, and it could be of use in machines such as an electric pinball machine.
- 7.

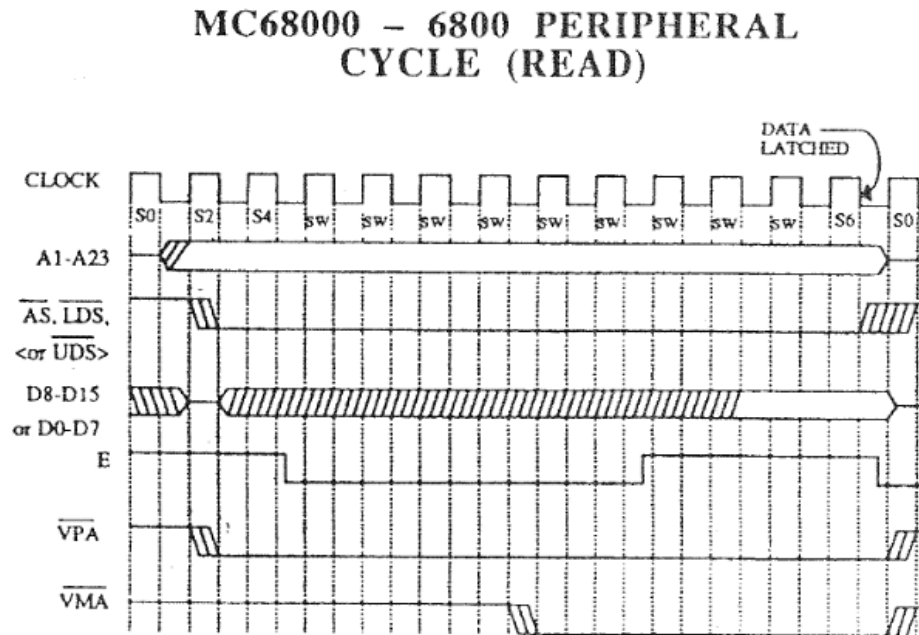


Figure 3: MC68000 Bus Cycle

1. Address lines stabilize
2.  $\overline{VPA}$ ,  $\overline{AS}$ ,  $\overline{LDS}$ , or  $\overline{UDS}$  stabilize
3.  $\overline{VMA}$  stabilizes
4. Data lines stabilize
5. Data latches
6. All lines reset

## 6 Conclusion

Overall the experiment was a success. The student was successfully introduced to the peripheral interface adapter IC (PIA, MC6821), the MC68K's Synchronous Cycle, and the MC68K's Interrupt Generation Mechanism. Through the design work done in this experiment, these concepts were solidified.

## References

- [1] Experiment 7 Lab Manual
- [2] Educational Computer Board Manual
- [3] MC68K User Manual
- [4] SANPER-1 ELU User Manual

## 7 Appendix

### 7.1 Code

```
1 *-----
2 * Title       : Lab 7 Routine
3 * Written by  : Adam Sumner and Ryan Jenkins
4 * Date       : 4/14/2015
5 * Description: This program enables the I/O functionality of
6 *             the PIA
7 *-----
8     ORG      $064
9
10    DC.L ISR      ;pointer to subroutine
11    CRA EQU $52003
12    CRB EQU $52007
13    DDRA EQU $52001
14    DDRB EQU $52005
15    PDRA EQU $52001
16    PDRB EQU $52005
17    BUFFER DC.B 1
18    ORG      $900
19
20
```



```

21      BCLR.B   #2,CRA      ;select DDRA
22      BCLR.B   #2,CRB      ;select DDRB
23
24      MOVE.B   #00,DDRA     ;set PA0 – PA7 to input
25      MOVE.B   #$FF,DDRB    ;set PB0 – PB7 to output
26
27      BSET.B   #0,CRA      ;enable interrupts
28      BSET.B   #3,CRB      ;pulse mode
29      BCLR.B   #4,CRB      ;high to low transition
30      BSET.B   #5,CRB      ;output line
31
32      BSET.B   #2,CRA      ;select PDRA
33      BSET.B   #2,CRB      ;select PDRB
34
35      MOVE.B   #$07,CRA
36      MOVE.B   #$2C,CRB
37      ANDI.W   #$F8FF,SR    ;set interrupt to 0
38
39 LOOP:
40      LEA      BUFFER,A5
41      LEA      BUFFER,A6
42      MOVE.B   #241,D7      ;trap input
43      TRAP     #14
44      MOVE.B   (A5),PDRB
45      BRA      LOOP
46
47
48
49      ORG      $1800
50 ISR    MOVEM.L D0–D7/A0–A6,–(A7) ;store registers
51        MOVE.B PDRA,D0      ;get data from PIA
52        MOVE.B #248,D7      ;output char
53        TRAP     #14
54        MOVEM.L (A7)+,D0–D7/A0–A6 ;restore registers
55        RTE              ;end exception

```

## 7.2 Hardware

