Interfacing a speed control processor to the IBM PC

by

Ting-Woen Woen

A Thesis Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Department: Electrical Engineering and Computer Engineering
Major: Computer Engineering

Iowa State University
Ames, Iowa

1988

TABLE OF CONTENTS

PAGE

I. INTRODUCTION 1

II. OVERVIEW OF THE IBM PC SYSTEM 3
A. System Board 3
B. 8088 Registers 7
C. Addressing Scheme 7
D. System Bus 9
Bus cycle 15
E. ROM Software and Interrupt Vectors 18
F. The 8253 Timer Clock 22

III. MC14460 SPEED PROCESSOR 25

IV. THE IMPLEMENTATION 31
A. The PC System Board and System Bus 31
B. The Address Decoder and Latches 31
1. The latches 33
2. The address decoders 35
3. The additional logic circuits 35
C. The PPI Chip and Feedback Circuits 35
1. The feedback circuits 36
2. The PPI chip 37
D. The Speed Control Processor Circuits 38
E. The Supporting Software 40
1. The Command 40
a. Increase the speed 43
b. Decrease the speed 43
c. Brake 43
d. Resume 43
e. Inhibit 43
f. Terminate the program 44
2. The feedback 44

V. RESULT AND APPLICATION 46
A. Automotive Cruise Control 47
B. Electric Motor Speed Control 50

VI. CONCLUSION 53

VII. BIBLIOGRAPHY 54

IX. ACKNOWLEDGEMENT 55

X. APPENDIX: THE PROGRAM LISTING 56

LIST OF TABLES

PAGE

TABLE 1. Hardware interrupts in the PC 6

TABLE 2. 1-megabyte memory addresses 10

TABLE 3. I/O port addresses 11

TABLE 4. The truth table of the MC14460 outputs 26

LIST OF FIGURES

PAGE

FIGURE 1. System board block diagram 4

FIGURE 2. 8088 registers 8

FIGURE 3. The 62-pin I/O channel bus 12

FIGURE 4. Memory read bus cycle 17

FIGURE 5. Memory write bus cycle 18

FIGURE 6. I/O port read bus cycle 19

FIGURE 7. I/O port write bus cycle 20

FIGURE 8. Extending the timer interrupt 23

FIGURE 9. The MC14460 speed control processor 26

FIGURE 10. System timing of MC14460 27

FIGURE 11. The oscillator of the MC14460 27

FIGURE 12. The MC14460 28

FIGURE 13. The speed control system block diagram 32

FIGURE 14. Latches and address decoder 34

FIGURE 15. The feedback circuits and the PPI 36

FIGURE 16. The 8255 PPI chip block diagram 38

FIGURE 17. Speed control processor circuits 41

FIGURE 18. The PPI control register 42

FIGURE 19. Output timing diagram 47

FIGURE 20. The output decoder 48

FIGURE 21. The speed control system in application . . . 48

FIGURE 22. Typical automotive cruise control application 49

FIGURE 23. DC motor speed control block diagram 52

I. INTRODUCTION

Within recent years, there has been a rapid growth in the microcomputer industry. Microcomputers have become smaller in size, less expensive, easier to use and some even more powerful than earlier mainframe computers. From the many microcomputers available today, the IBM PC and its compatibles (referred simply as PC for the rest of this paper) are the most popular and widely used. The reasons for this are:

- IBM does not make most of the software and hardware used in the PC. IBM selects a set of programs and hardware components from the microcomputer industry. For example:
    - IBM uses a disk operating system, MS-DOS, made by Microsoft, a leading microcomputer software company.
    - IBM uses a well known microprocessor made by Intel, the Intel 8088.
- IBM publishes a complete technical reference manual.

All of these have encouraged companies to build PC compatible computers, develop application software, and additional hardware components for the PC. A PC compatible computer can now be purchased for less than $500.

As a result, there have been more applications using the PC than ever before. These applications involve not only software (spreadsheets, word processors, games) but also hardware (communications, device controllers).

Unfortunately, interfacing the PC to devices to provide real time control has been a challenge for engineers. This application will be the focus of this work. In particular, we will implement the interfacing of a PC to a speed control processor to control the speed of motors. We have chosen the MC14460 speed control processor as the processor chip which is used to help the PC in controlling speed. We will implement the hardware using a prototype card which can be plugged into one of the I/O channels in the system board inside the PC cabinet.

The advantage of using a PC to control the speed is that it provides more accurate, reliable, and automatic control than a conventional circuit. In a conventional circuit, hundreds of electronic components are typically needed. In our design, we only need one speed control processor and some interface circuitry.

II. OVERVIEW OF THE IBM PC SYSTEM

Before we describe our design, we will give an overview of some important parts of the PC that are related to our design. We will start by looking at the system board inside the PC.

A. System Board

When we open the PC cabinet, we will find a system board, several add-on boards, one or more disk-drives, a power supply, and a fan.

Figure 1 shows the block diagram of the system board. In this board, there are five I/O channels (eight on the PC-XT system) in which expansion boards can be inserted. The I/O channels, which have 62 pins, are also called the system bus.

On the system board, up to 256K bytes of RAM and 40K bytes of ROM can be installed. The ROM contains programs for basic interfacing to the keyboard, screen, disk, cassette, timer, serial communication, parallel printer, etc.

The most important chip on the system board is the 8088 microprocessor. The 8088 has eight 16-bit arithmetic registers. Beside the 8088, there are four other important chips: the 8253 counter/timer chip, the 8237 DMA chip, the

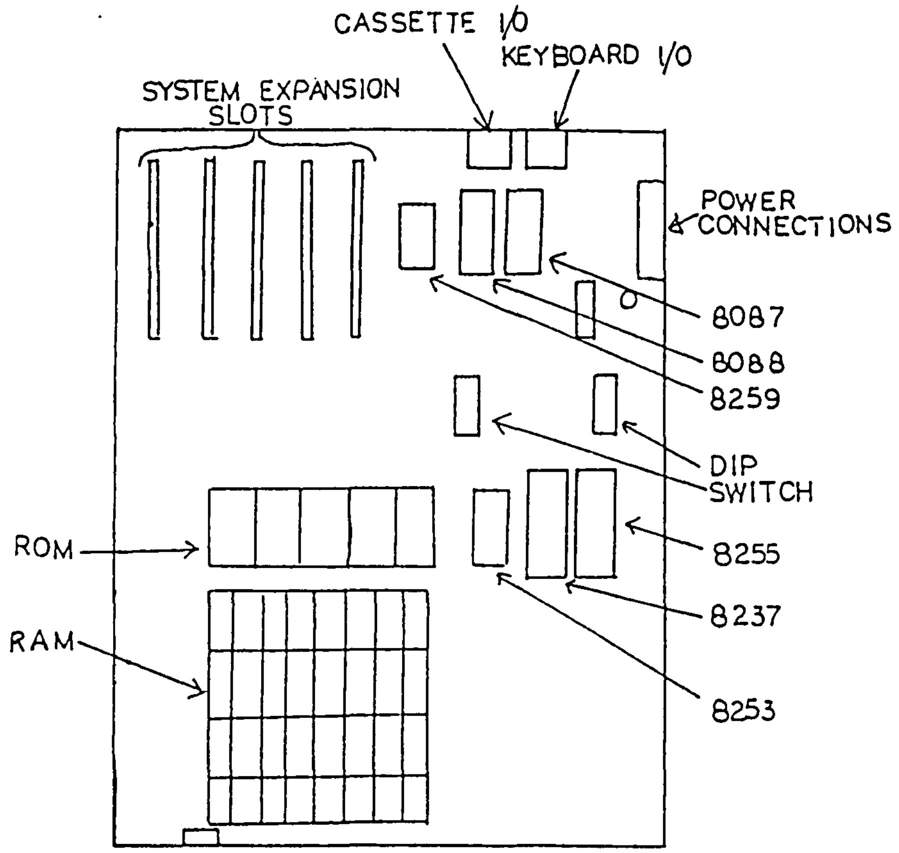


FIGURE 1. System board block diagram

8255 PPI chip, and the 8259 interrupt controller chip. We will discuss them briefly here. In the latter part of this chapter, we will discuss the 8253 counter/timer chip in detailed because we will utilize this chip in our speed control system.

First, we discuss the 8253 counter/timer chip. This chip allows the PC to maintain the time of the day and the date if the power is left on, to keep track of refreshing the RAM memory at a specific time interval, and to create sound through the use of the speaker. Then, we discuss the 8237 direct memory access (DMA) chip. This chip will allow slower devices like disk-drives controller to transfer data directly to and from the main memory without involving the CPU. This chip together with the 8253 are also used to refresh the RAM memory.

Next, we discuss the 8255 parallel I/O chip. This chip is also called the PPI (Programmable Peripheral interface) chip. This chip is used by the PC to read the two system board DIP switches. From these two DIP switches, the PC determines the amount of memory in the PC, the number of disk-drives, and the type of the screen used. The 8255 is also used to read the data coming from the keyboard. The last chip is the 8259 eight-channel interrupt controller chip. This chip is used to handle the hardware interrupts.

The hardware interrupts are signals sent to the CPU by the hardware to request an attention from the CPU. The 8259 will determine the level of each interrupt and send interrupt to the CPU according to the level of priority. There are eight levels of priority. Table 1 shows the interrupt assignments in the PC.

TABLE 1. Hardware interrupts in the PC

IRQ	Used by
0	timer
1	keyboard
2	I/O channel
3	COM1
4	COM2
5	fixed disk
6	diskette controller
7	LPT1

There is an empty socket reserved for the 8087 floating-point arithmetic coprocessor. With this chip installed, the PC's power in calculating is increased in the neighborhood of ten to fifty times faster.

B. 8088 Registers

To program the 8088, we need to know its registers. Figure 2 shows the 8088 registers. These registers are grouped into four groups: data registers, segment registers, pointer and index registers, and flag register. Data registers consist of AX, BX, CX, and DX. Each register is sixteen-bit long but we can use it as an eight-bit long register by specifying 'H' (high) or 'L' (low); for example, AH, AL, etc. These registers are used to temporarily store the intermediate results and operands of arithmetic and logical operations. The four segment registers hold the segment part of addresses (discuss shortly). The pointer and index registers are used to hold the offset part of the addresses. Finally, the flag register consists of nine one-bit flags that record the status of the 8088 operations.

C. Addressing Scheme

The PC can address up to one mega bytes of addresses. Twenty-bit registers are required to hold these addresses but the 8088 registers only have sixteen bits. To overcome this problem, the 8088 uses segmented address scheme. The 8088 divides the memory space into segments; each segment contains 64K bytes. Each segment begins at a location that is evenly divisible by 16 bytes. This segment is called the

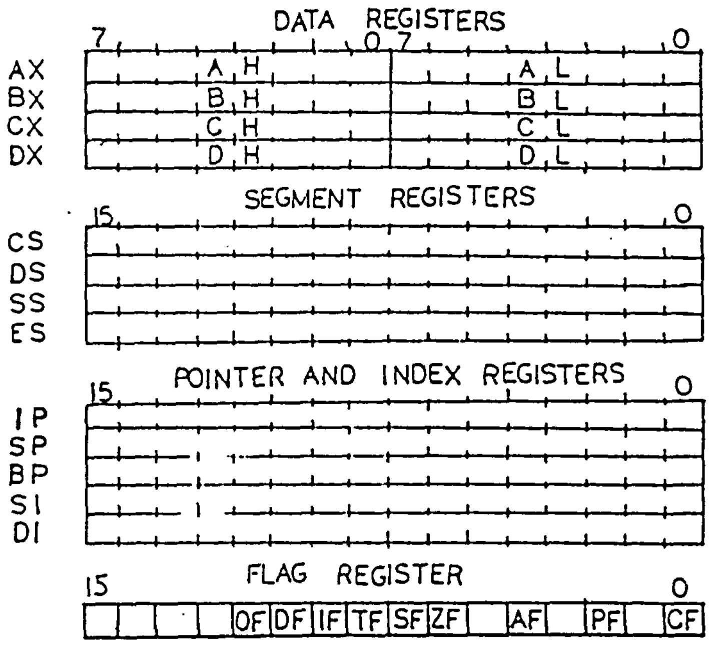


FIGURE 2. 8088 registers

segment part of an address. To access individual byte, the 8088 uses additional address called the offset address, which points to the specific byte within the 64K segmented memory. The address is created by combining the sixteen-bit segment address and the sixteen-bit relative offset address. To do this, the segment address is shifted four bits to the left and added to the offset address. Now the address is 20 bits long. The segment registers in the 8088 registers set are used to hold the segment addresses; the pointer and the index registers are used to hold the offset addresses. Note that since the segment register is shifted 4 bits to the left, it can only point to actual memory spaces that are multiple of 16. A specific address will be written as xxxx:yyyy (in hexadecimal); where x is the segment address and y is the offset address. The actual address is xxxx0+yyyy.

The PC uses separate addresses for the memory and I/O ports. The PC uses only 10 bits to address the I/O ports: A0 through A9 are used and A10 through A19 are ignored. Therefore, there are only 64K spaces available for the I/O ports.

Now we will show how the PC allocates its 1-mega memory addresses and its 64K I/O port addresses. Table 2 shows the 1-mega memory addresses assigned by the PC. Table 3 shows the I/O port addresses assigned by the PC. For our design, we will use a prototype card; therefore, we will use location 300-31F of I/O port addresses.

D. System Bus

For interfacing application, the connection to the PC is through the system bus. The system bus has 62 signal lines which consist of 20 address lines, 8 data lines, 27 control lines, 7 power supplies and ground. There are 5 62-pin I/O channel slots on the system board (7 in the PC-

TABLE 2. 1-megabyte memory addresses

Address	Function
0-3FF	Interrupt vector table
400-47F	BIOS data area
480-5FF	BASIC and special system function RAM
600-9FFFF	Program memory
A0000-AFFFF	EGA graphics mode video RAM
B0000-B7FFF	Monochrome video RAM
B8000-BFFFF	Color/graphics video RAM
C0000-CFFFF	I/O ROM BIOS's, EMS window
D0000-DFFFF	I/O ROM BIOS's, EMS window
E0000-F3FFF	Unused on PC
F4000-F5FFF	PC spare ROM socket
F6000-FDFFF	ROM BASIC
FE000-FFFFF	ROM BIOS

XT) that are available for attaching interfacing cards. Figure 3 shows the 62-pin of the I/O channel bus. Now we discuss the 62 signals that are important for our design.

1. A0 through A19

   These are the address lines. These signals are the output lines from the CPU or the DMA controller to address the memory or the I/O ports. A0 is the least significant bit and A19 is the most significant bit. One mega bytes of memory location can be addressed by these 20 lines. For the I/O ports addressing, only 10 address lines are used.

TABLE 3. I/O port addresses

Address	Function
	System Board
0-1F	8237 DMA controller
20-3F	8259 interrupt controller
40-5F	8253 counter/timer
60-7F	8255 PPI
80-9F	DMA 64K page register
0A0-0BF	NMI reset
0C0-0DF	Unused on PC
0E0-0FF	8087 math coprocessor interface
100-1FF	Unused on PC
	I/O Channel
1F0-1F8	Unused on PC
200-20F	Game I/O adapter
210-217	Expansion unit
220-24F	Reserved
250-277	Not used
278-27F	LPT2
280-2EF	Not used
2F0-2F7	Reserved
2F8-2FF	COM2
300-31F	Prototype card
320-32F	XT hard disk
378-37F	LPT1
380-38C	SDLC
3A0-3AF	Primary binary synchronous
3B0-3BF	Monochrome display and LPT1
3D0-3DF	Color/graphics display adaptor
3F0-3F7	Floppy disk drive controller
3F8-3FF	COM1

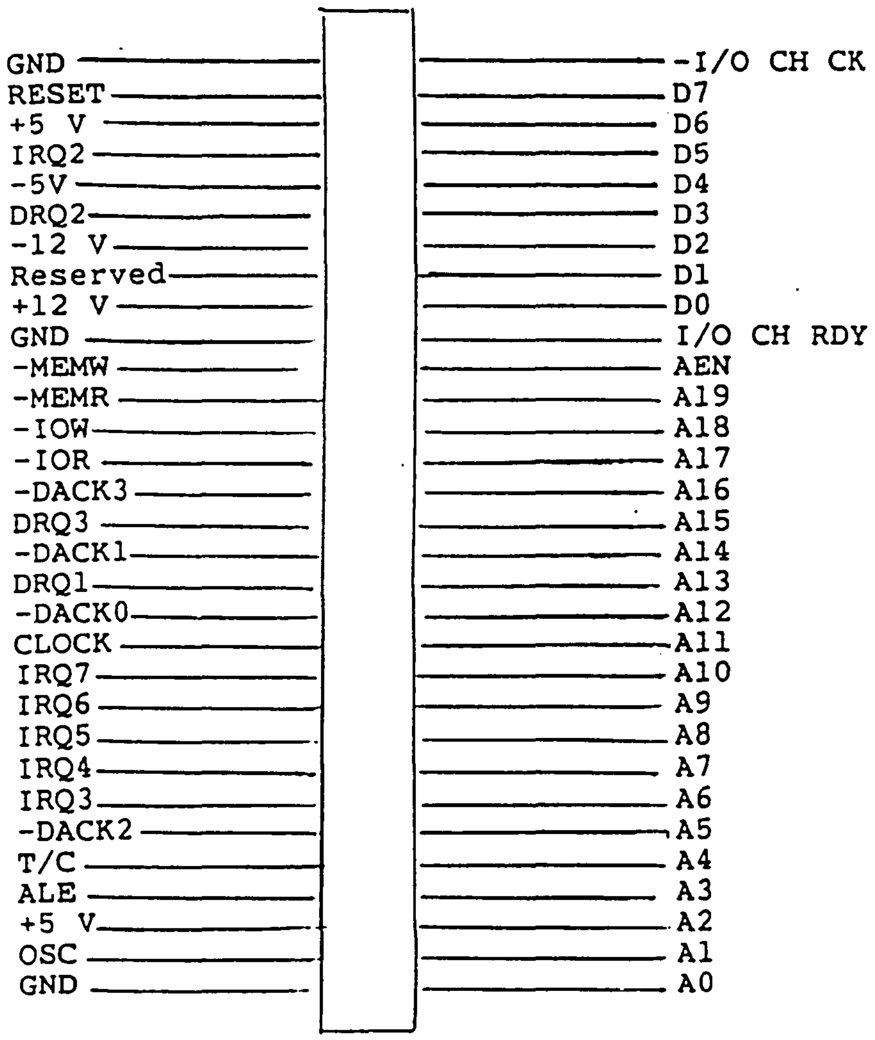


FIGURE 3. The 62-pin I/O channel bus

2. D0 through D7

   These are eight data lines used to carry data from and to the CPU, the memory, and the I/O ports. The data should be placed in D1 through D7 right before IOW, MEMW, IOR, or MEMR line is activated.

3. IOW, MEMW, IOR, MEMR

   These signals are low-level active output-only signals from the CPU. IOW indicates that the CPU wants to write data into the I/O ports. MEMW indicates that the CPU wants to write data into the memory. IOR indicates that the CPU wants to read data from the I/O ports. MEMR indicates that the CPU wants to read data from the memory.

4. IRQ2 through IRQ7

   These are interrupt requests 2 through 7 lines. These lines are used to request the hardware interrupts to the CPU. These signals go to the 8259 interrupt controller. If the interrupt is not masked, a signal will be generated to interrupt the CPU.

5. ALE

   ALE is address latch enable line. This is an output line from the bus controller and is used to indicate that the address lines are valid addresses.

6. AEN

   This is a signal issued by the DMA controller indicating a DMA bus cycle. This signal is used to disable the CPU address, data, and control buses from the system bus.

7. DRQ1 through DRQ3 and DACK0 through DACK3

   DRQ1-DRQ3 are direct-memory access request 1 through 3 lines and are used to request the DMA bus cycle. DACK0-DACK3 are direct-memory access acknowledge 0 through 3 lines. These signals issued by the 8237 DMA controller to indicate that a DRQ has been granted.

8. TC (terminal count)

   This is an output signal from the DMA controller used to indicate that the DMA has reached the preprogrammed number of transfer cycles.

9. I/O Ch Ck (I/O channel check)

   This signal is used by interface cards to indicate error conditions.

10. I/O Ch Rdy (I/O channel ready)

    This is a ready line used to extend the length of the bus cycles so that the memory or the I/O ports that are not fast enough to respond to the CPU bus cycle can still be attached to the system bus.

11. OSC (Oscillator) and CLK (Clock)
    The OSC has a frequency of 14.31818 MHz. It has the highest frequency on the system bus and it is used to derive other timing signal. CLK is derived from OSC signal. This is the system clock and has a frequency of 4.77 MHz. The period is 210 ns with 70 ns high and 140 ns low time.
12. RESET DRV (reset driver)
    During system power-on, this signal is held high until all levels have reached their operating range; then, this signal goes low.
13. Power supplies and ground
    The power supplies are +5 VDC, -5 VDC, +12 VDC, -12 VDC, and GND (ground).

<u>Bus cycle</u>

The communication between the CPU and the memory or the I/O devices is done in a bus cycle.

There are five different bus cycles:

- Memory read bus cycle:

In this bus cycle, instructions and data are fetched from the memory to the CPU. This bus cycle consists of a minimum of 4 processor clocks or approximately 840 nanoseconds. This cycle can be extended by memory device by using the ready line. The memory-read bus cycle is shown in Figure 4.

- Memory write bus cycle:
  In this bus cycle, data are written into the memory from the CPU. The cycle length is the same as the memory read bus cycle. This bus cycle is shown in Figure 5.
- I/O port read bus cycle:
  This cycle is initiated when the CPU executes "IN" instruction. Data are fetched from the I/O port to the CPU. It has a minimum of 5 clocks, or approximately 1.05 ms in length. The ready bus signal can be activated by the I/O port to slow down the bus cycle. This bus cycle is shown in Figure 6.
- I/O port write cycle:
  This bus cycle writes data from the CPU to the I/O port. It is used when the instruction "out" is executed. The bus cycle length is the same as the I/O port read bus cycle. This bus cycle is shown in Figure 7.

- DMA-driven bus cycles:

  DMA driven bus cycles are not driven by the CPU but they are driven and controlled by the DMA controller.

  There are two DMA-driven bus cycles:

  - DMA write I/O:

    This cycle reads data from an interface adaptor and writes them to a memory location.

  - DMA read I/O:

    This cycle reads data from a memory location and writes them to an interface adaptor.

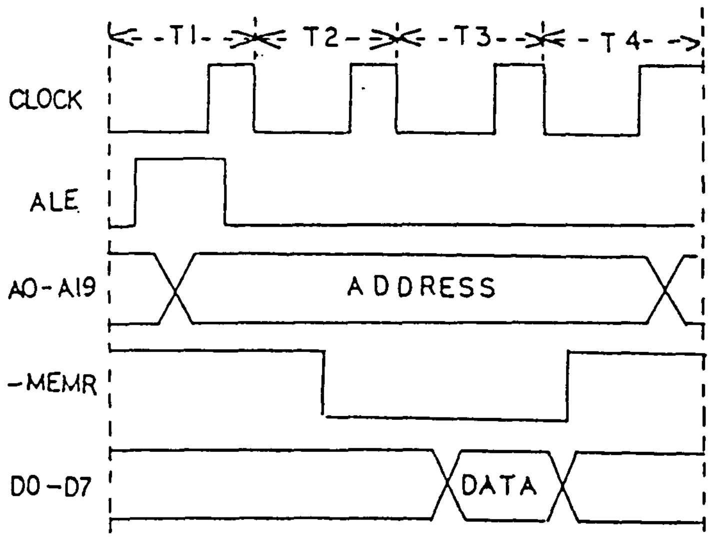


FIGURE 4. Memory read bus cycle

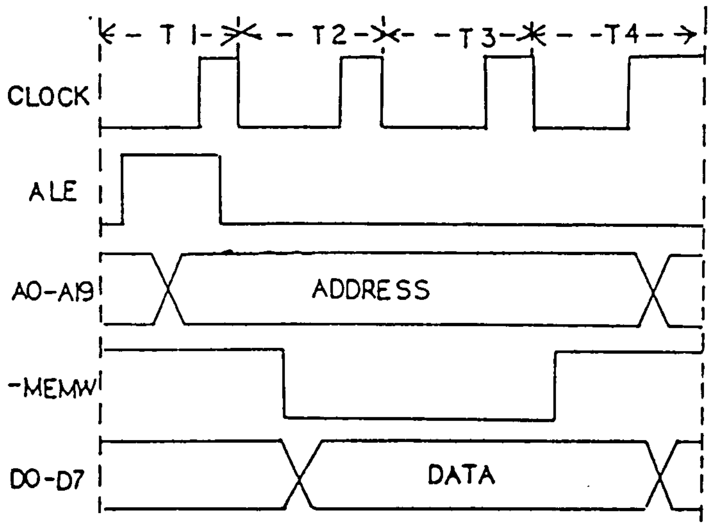


FIGURE 5. Memory write bus cycle

E. ROM Software and Interrupt Vectors

To make the computer runs, the software is needed. Some of this software is permanently built into the computer. This is called the ROM (Read Only Memory) software.

There are 4 parts of ROM in the PC:

1. The start-up program is used to get the computer started.
2. The ROM-BIOS (Basic Input/Output System) is used to provide services needed for continuing

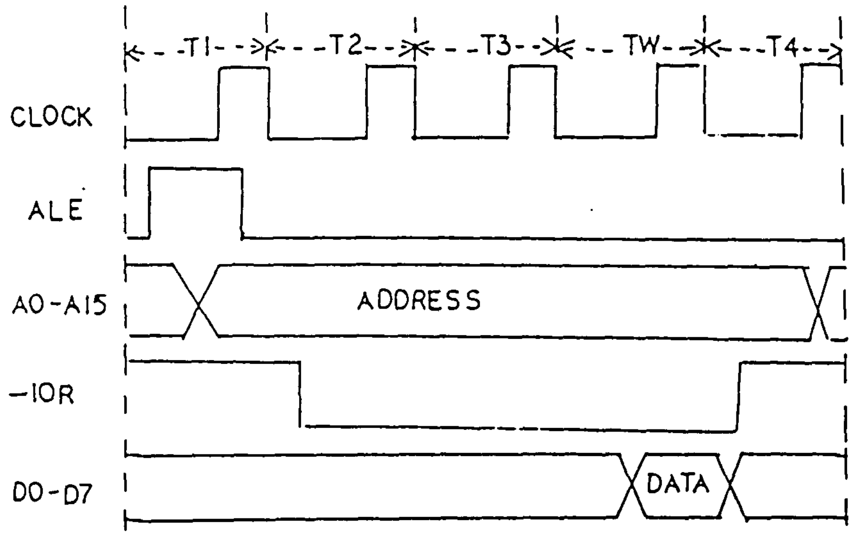


FIGURE 6. I/O port read bus cycle

operation of the computer such as the display screen, keyboard, and disk-drives. We can access these services by using the interrupts. The ROM-BIOS is also used to handle the hardware interrupts.

3. The ROM-BASIC is used for BASIC programming language.

4. The ROM-Extensions are added when optional equipments are installed to the PC.

In the start-up program, several tasks are performed:

- The Power On Self Reset (POST):

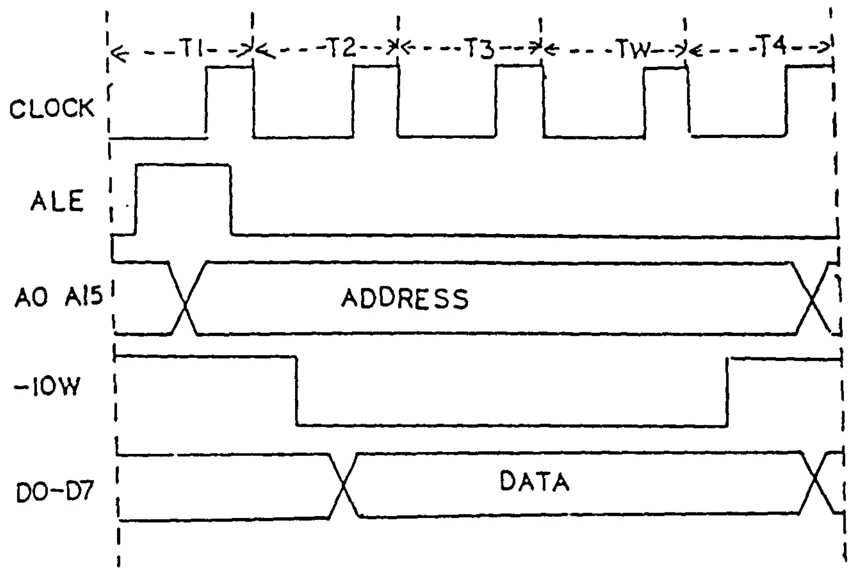


FIGURE 7. I/O port write bus cycle

This is a reliable test to make sure that the computer is ready.

- The Initialization Process:
  This routine will initialize the chips and standard equipments in the PC, fill the default values to the interrupt vectors, and check to see what optional equipments have been installed in the PC.
- The Boot-Strap Loader:
  This routine will load the operating system from the disk and pass the control to this operating system.

The PC uses interrupts intensively to control the operation of the computer. These interrupts can be generated by hardware and software. Each of the ROM-BIOS service routines is assigned an interrupt number that can be called to get its service. When an interrupt occurs, the control of the CPU is passed to the interrupt service routine. The address of this routine can be found in the interrupt vector table. The first 400H bytes of memory are reserved for the interrupt vector table. In writing a program, we can generate a software interrupt by executing the instruction INT nn, where nn is the interrupt number. This interrupt number is used to find an entry in the interrupt vector table. This entry is in the absolute address 0:nn*4. The content of this entry will be the address of the interrupt service routine. We will show how to change the entry of this table to point to our service routine in Chapter IV.

One of the interrupt, interrupt 21, will invoke the DOS functions. There are 98 different functions provided by the DOS. Some of these functions are used to read the input from the keyboard, write data to the screen, change or read interrupt vector table entries, and manage the files. We will discuss some of these functions as we use them in implementing our design.

F. The 8253 Timer Clock

Before we conclude discussing about the PC, we will talk about one chip that will be used in our design, the 8253 timer chip. The PC uses the 8253 timer chip to count the system clock. The outputs of 8253 are used to count the time of the day, or to produce the sound, or to refresh the RAM.

There are three programmable channels in the 8253 chip. Channel 0 is used by BIOS timer routine to keep the time of the day. This BIOS routine keeps a time-of-day count. In every 18.2 times per second, the output of channel 0 causes a hardware interrupt (INT 8) to this BIOS routine to increment the time-of-day count. This time-of-day count is reset to 0 at midnight. The actual time can be calculated by dividing this count with 18.2 for every second.

When this BIOS routine is executed, another interrupt (INT 1CH) is called. Initially, INT 1CH service routine is empty (only has an interrupt return instruction). We can supply a special routine to replace INT 1CH. As a result, this new routine will be executed 18.2 times per second. Figure 8 shows the timer interrupt and the INT 1CH.

Channel 1 is used to refresh the RAM. The output of this channel will cause the DMA chip to refresh the RAM.

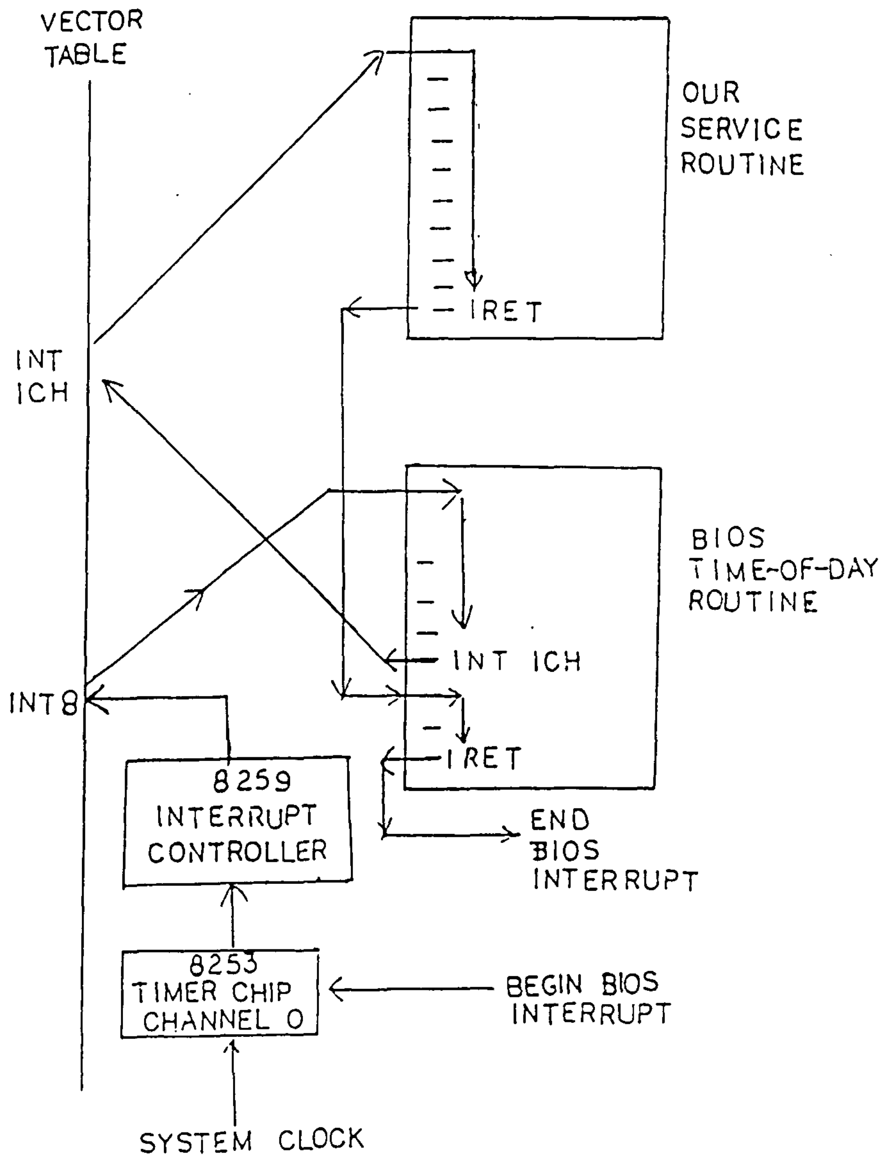


FIGURE 8. Extending the timer interrupt

Channel 2 is connected to the speaker in the system board. This channel's output is a square wave which frequency can be changed by programming the counter of channel 2. In addition, the PPI chip in the system board is used to enable this channel's output to the speaker.

III. MC14460 SPEED PROCESSOR

In this chapter, we will discuss about the MC14460 automotive speed control processor. The MC14460 is a processor dedicated to control speed. It has an internal register to store a reference speed. The MC14460 will measure an input speed and provide pulse-width modulated outputs to increase or decrease the speed to maintain an internally stored reference speed.

The internally stored reference speed can be altered by Decel (deceleration) and Accel (acceleration) commands. The Accel command will increase the speed; the Decel command will decrease the speed. This processor also has two other input commands: brake and resume. The brake command will turn off the outputs and the resume command will increase or decrease the speed to the last stored reference speed.

The output commands are derived from two output lines: VAC and VENT. The truth table of VAC and VENT is shown in Table 4.

Figure 9 shows the block diagram of the MC14460. There are two separate oscillators: the master oscillator for the system time reference and the pulse oscillator for the output pulse width. Figures 10 and 11 show the system timing and oscillators of the MC14460.

TABLE 4. The truth table of the MC14460 outputs

VAC	VENT	FUNCTION
0	0	Decrease speed
0	1	Hold speed
1	0	Invalid output
1	1	Increase speed

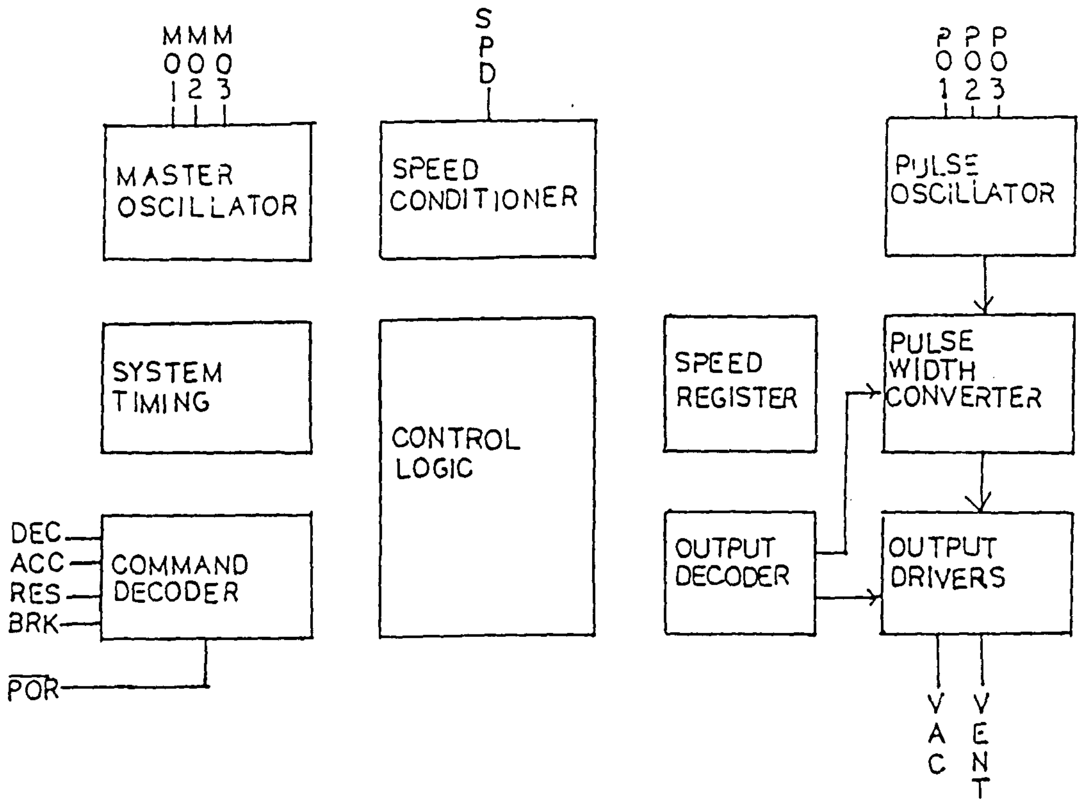
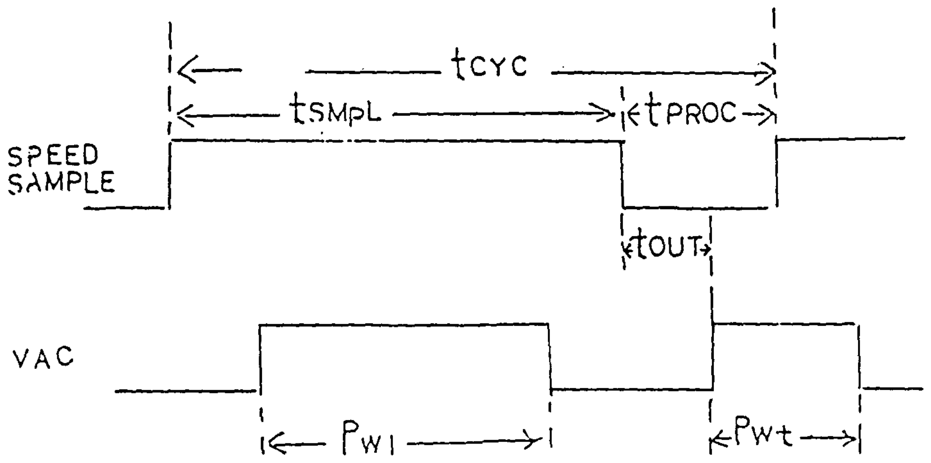


FIGURE 9. The MC14460 speed control processor



tcyc	= system cycle time	= 1024/fM
tsmpl	= speed sample time	=1008/fM
tproc	= speed processing time	=16/fM
tout	= output delay time	=9/fM
Pwl	= initialization output pulse width	= 1/fP
Pwt	= trim output pulse width	= 1/fP
fM	= main oscillator frequency	= 1/2.43RC
fP	= pulse oscillator frequency	= 1/2.43RC

FIGURE 10. System timing of MC14460

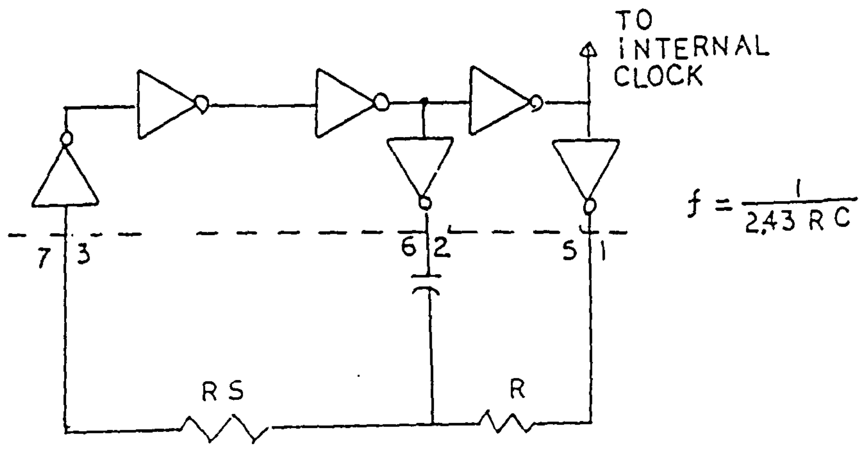


FIGURE 11. The oscillator of the MC14460

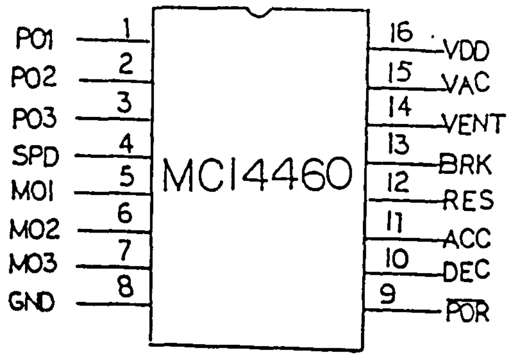


FIGURE 12. The MC14460

Figure 12 shows the pin assignment of the MC14460. The operations of these pins are described as follows:

1. Pulse oscillator (Po1, Po2, Po3; Pins 1,2,3):
   This oscillator sets the relative pulse width of the VAC and VENT outputs. These pins are the outputs of the output pulse oscillator which is a three-terminal RC type.
2. Speed (Spd; Pin 4):
   This is the speed input to be stored and controlled. This input frequency should not exceed 1/3 of the master oscillator frequency (fM).
3. Master oscillator (M01, Mo2, Mo3; Pins 5,6,7):

Like the pulse oscillator, these pins determine the master system timing. The master oscillator is also a three-terminal RC type.

4. Power-on Reset (POR; Pin 9):
When this pin is held low, the internal system is cleared and the outputs are disabled. Internally, a pull up device will source 15-200 μA of current from this pin to allow capacitor charging for automatic power-on reset.

5. Decel (Dec; pin 10):
This is an input line to decrease the speed. When the Decel is activated, the VAC and VENT outputs will be low. When the Decel is inactivated, the current speed is stored in the reference speed register.

6. Accel (Acc; Pin 11):
This is an input line for the acceleration command. When the Accel is activated, the VAC and VENT outputs will be modulated to maintain a fixed rate of acceleration. When the Accel is inactivated, the last sample input input will be stored in the reference speed register.

7. Resume (Res; Pin 12);

This is an input line for the resume command. When the resume is activated, the VAC and VENT outputs will be modulated to maintain a fixed rate acceleration which end when the input speed matches the stored reference speed.

8. Brake (Brk; Pin 13):

   This is an input line for the brake command. When the brake is activated, the system is disabled until the Decel, the Accel, or the Resume command is received.

9. VENT and VAC (pins 14.15):

   These are the outputs used to control the speed. The truth table is shown in Table 4.

IV. THE IMPLEMENTATION

In this chapter, we will discuss the implementation of a speed control system. We will begin the discussion by looking at the block diagram, Figure 13, which consists of four parts:

1. The PC system board and system bus.
2. The address decoders and latches.
3. The PPI and feedback circuits.
4. The speed control processor circuits.

The circuits are implemented on a prototype card and tested in one of the I/O channel buses.

A. The PC System Board and System Bus

We have discussed this topic in Chapter II.

B. The Address Decoder and Latches

To interface the system bus to the rest of our design, some address decoders and latches/buffers are essential. These circuits separate the interface circuits from the system board and drive the interface circuits. By using the latches, the system bus does not have to source as much current as driving the interface circuits directly.

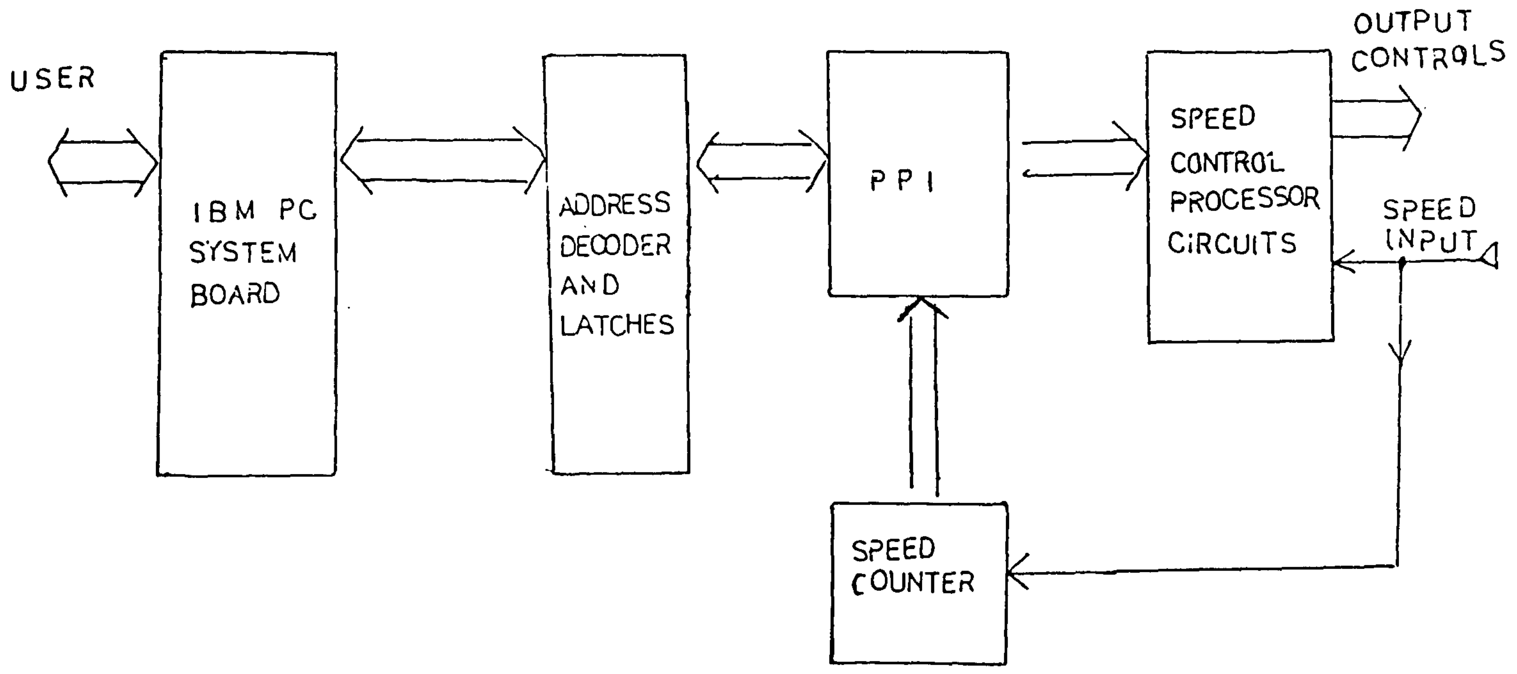


FIGURE 13. The speed control system block diagram

Figure 14 shows the circuit design for this purpose. There are three parts:

- The latches.
- The address decoders.
- The additional logic circuits.

1. The latches

We use three chips: one 74LS245 and two 74LS244 (IC1, IC2, and IC3). IC1 (the 74LS245) is an octal bidirectional tristate buffer with Schmitt-trigger inputs. This chip is used to buffer and drive the eight bidirectional data lines (D0-D7). This chip drives the data in two ways: to the interface circuits or to the CPU. Pin 1 of this chip is the direction line which controls the direction of the data. When this pin is high, the data flow from the system bus to the interface circuits. When this pin is low, the data flow from the interface circuits to the system bus. This line is controlled by IOR (I/O read) line.

IC2 and IC3 are both 74LS244 which are octal tristate buffers with Schmitt-trigger inputs. These two chips are used to latch address and control lines (IOR, IOW, MEMR, MEMW, Reset, ALE, and AEN). These lines are one direction lines, from the system bus to the interface circuits.

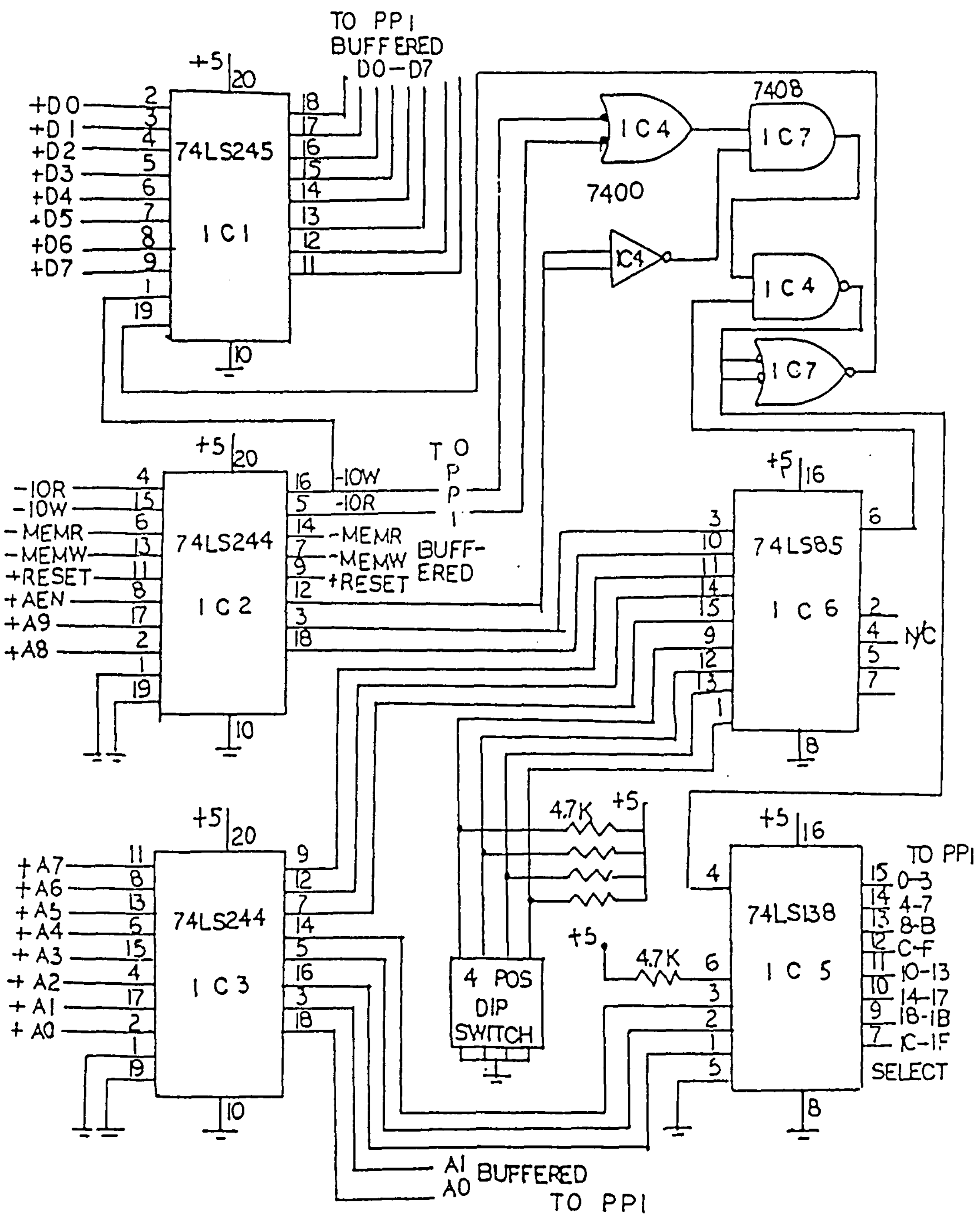


FIGURE 14. Latches and address decoder

2. The address decoders

Two chips (IC5 and IC6) are used for this purpose. IC5 (the 74LS138) is a 3-to-8-line decoder. Three of the address lines, A2 through A4, are decoded to select eight group-of-four addresses. For our system, we will use the first group which will address xxxxx00000 through xxxxx00011 in binary. IC6 (the 74LS85) is a 4-bit magnitude comparator. This chip is used to compare A5 through A8 with the DIP switch setting. We use the DIP switch so that the user can select the desired A5-A8. For prototype card, the address used in the PC ranges between 300 and 31F or A8 through A5 will be 1000 in binary. In the PC design, for standard interfacing purpose, line A9 will always be 1. Now, we have selected our address from 1100000000 through 1100000011 in binary or 300 through 303 in hexadecimal.

3. The additional logic circuits

IC4 and IC7 are used to provide the additional logic circuits to enable IC1, IC5, and IC6. The logics are the 'and' and 'or' gates.

C. The PPI Chip and Feedback Circuits

We will first discuss the feedback circuits and then the PPI chip.

1. The feedback circuits

The purpose of using the feedback circuits is to send the input speed to the CPU by using the PPI chip. Figure 15 shows the feedback circuits diagram. This diagram shows how the speed is channelled to the PPI chip by using two 4-bit binary counters. Two 74191 binary counters are used to count the frequency of the input speed. The PPI chip will read these counters.

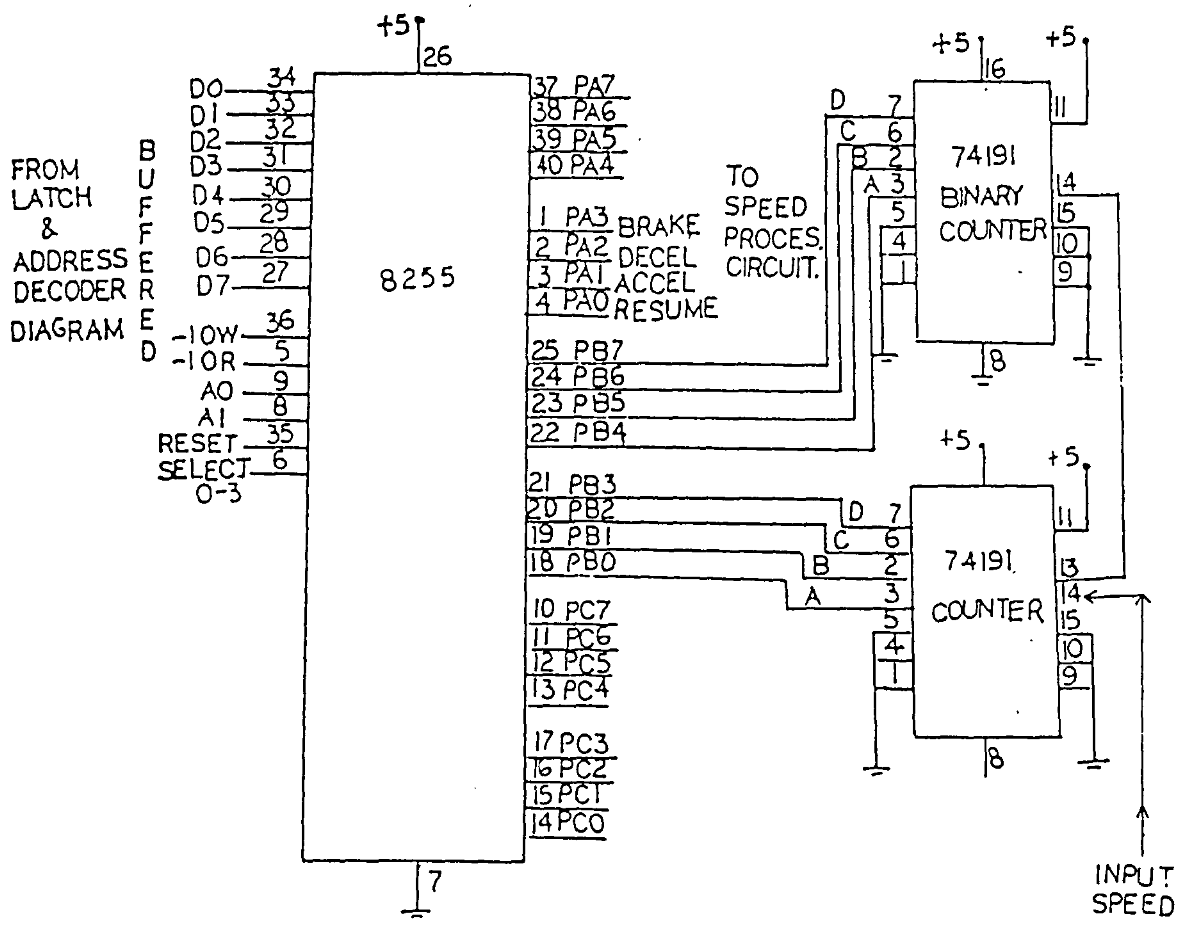


FIGURE 15. The feedback circuits and the PPI

2. The PPI chip

The 8255 PPI (Programmable Peripheral Interface) chip is also called the PIO (Parallel I/O) chip. We first encounter this chip in the system board; here, we will use it for our design. Figure 16 shows the block diagram of the 8255 chip. There are three 8-bit ports: A, B, and C. Each of these ports can be programmed separately as input or output port. Moreover, port A can be used as bidirectional port by using 5 lines of port C for handshaking. The PPI chip has four internal registers which are addressed by A1 and A0:

- A1,A0 = 00; select port A
- A1,A0 = 01; select port B
- A1,A0 = 10; select port C
- A1,A0 = 11; select the control register

The control register is used to select the direction of data flow through ports A, B, and C and to choose the modes of operation. For our design, we will use mode 0 for no handshaking mode.

Figure 15 shows the PPI chip and its connections. We use port A to send the commands to the speed control processor circuits and port B to read the two 4-bit counters for the input speed frequency.

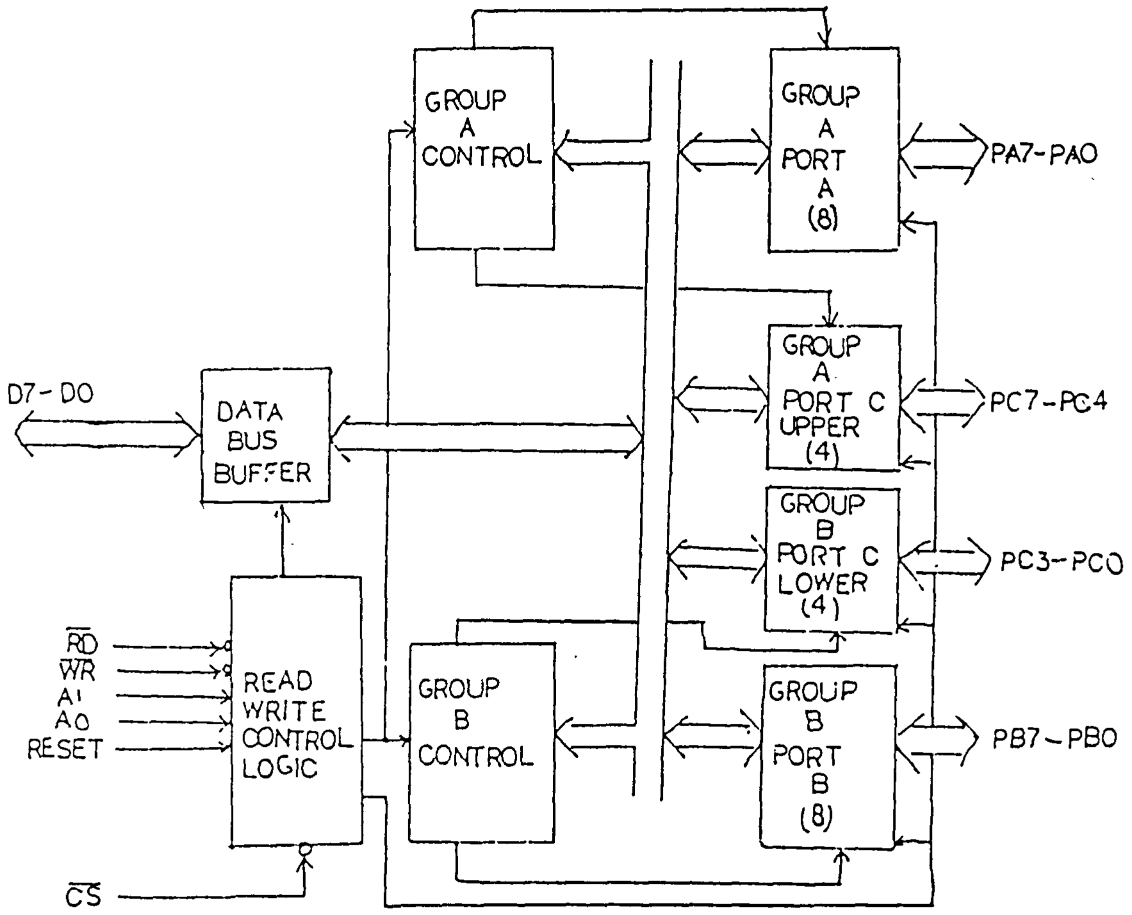


FIGURE 16. The 8255 PPI chip block diagram

D. The Speed Control Processor Circuits

In order for the speed control processor chip (MC14460) to work properly, some additional circuits are needed as shown in Figure 17. The descriptions of the design are as follows:

1. Pin 1, 2, 3 (Pulse oscillator):

   The design parameters chosen are:

   R = 43 K OHM

Rs = 100 K OHM

C = 5100 pF

The pulse oscillator frequency, fP, is:

$$fP = \frac{1}{2.43\ R\ C}$$

$$= 1.877\ KHz$$

2. Pin 5, 6, 7 (Master Oscillator):

   The design parameters chosen are:

   R = 43 K OHM

   Rs = 100 K OHM

   C = 5100 pF

   The master oscillator frequency, fM, is:

$$fM = \frac{1}{2.43\ R\ C}$$

$$= 1.877\ KHz$$

3. '+Sensor' and '-Sensor':

   These are speed input. '+Sensor' is connected to pin 4. '-Sensor' is connected to ground. For our design, the speed input is simulated by using a function generator. The function generator will produce a square wave with frequency between 50 and 300 Hz.

4. Decel, Accel, Resume, and Brake:

These are the input commands from the PPI chip.

5. VAC and VENT:

   We connect 2 transistors to pin 14 and pin 15. The reason is to separate the MC14460 from the outer world and to allow more drives on the VENT and VAC lines.

6. POR:

   POR is connected to a capacitor. At power-up, this capacitor will be able to reset the MC14460 and eventually charged up by the current source from the internal circuit.

E. The Supporting Software

So far, we have discussed the implementation of the hardware. In order for our system to be useful, it needs to communicate with the user. The user should be able to give commands to control the speed and to read the current speed on the screen. We will implement this by using the software. We will divide the software into two parts: the command and the feedback.

1. The Command

The user can issued seven commands:

- Increase the speed
- Decrease the speed
- Brake

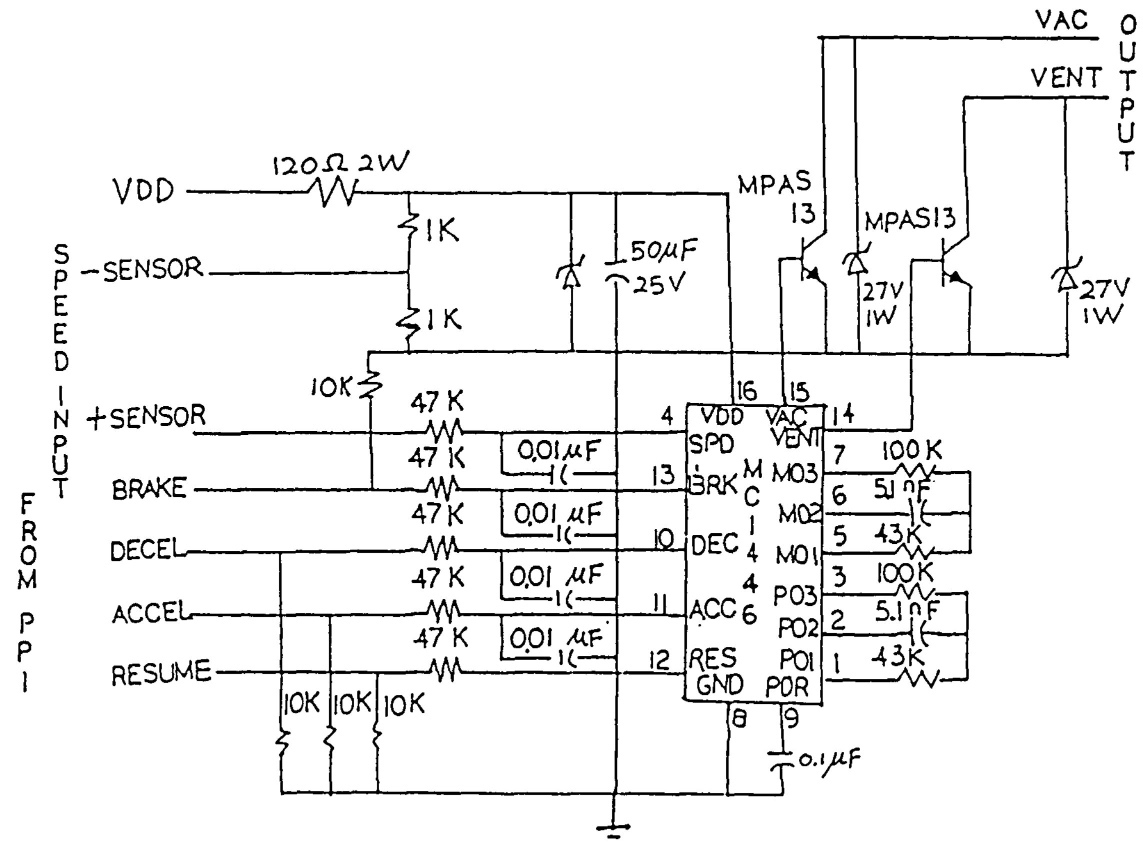


FIGURE 17. Speed control processor circuits

- Resume the speed
- Inhibit
- Terminate the program

We will first discuss the implementation of this commands and we will then discuss the interaction with the user. First, we need to initialize the PPI chip. As discussed earlier, the addresses for our PPI chip will be from 300 to 303 in hexadecimal. Address 300 is used to address port A, address 301 is used to address port B, address 302 is used to address port C, and address 303 is used to address the control register. Figure 18 shows the PPI control register. Since we use port A as the output to send the command to the speed control processor circuits and port B as the input to read the speed counter, we set bit D4 to 0 and bit D1 to 1. The modes setting for both port A and port B are 0. As a result, we write 10000010 in binary to address 303.

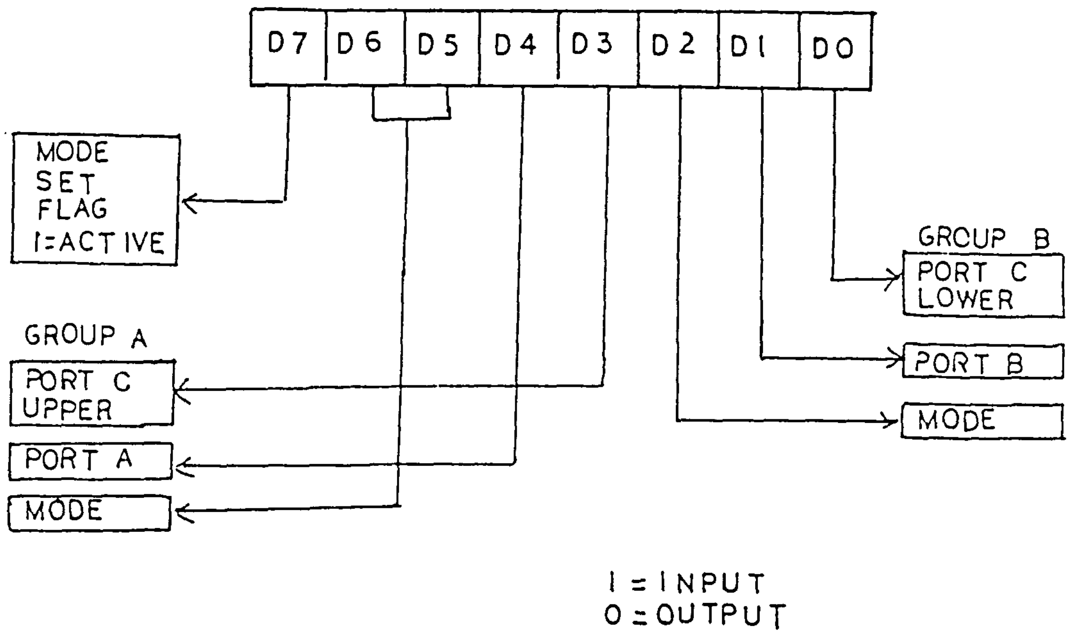


FIGURE 18. The PPI control register

<u>a. Increase the speed</u> If the user chooses to increase the speed, we will send a signal to the PPI to command an increase speed signal to the speed control processor. Since bit 1 of port A (PA1) is connected to the Accel input of the speed control processor, we move 00000010B (binary) to address 300H (hexadecimal).

<u>b. Decrease the speed</u> Bit 2 of port A (PA2) of the PPI is connected to Decel input to the speed control processor. We need to move 00000100B to address 300H.

<u>c. Brake</u> The user can stop the speed without affecting the stored reference speed by using the Brake command. Bit 3 of port A (PA3) of the PPI is connected to Brake input of the speed control processor. We move 00001000B to address 300H.

<u>d. Resume</u> The Brake command can be restored by using the Resume command. The resume will increase the speed to the stored reference speed. Bit 0 of port A (PA0) of the PPI is connected to the Resume input of the speed processor. We move 00000001B to address 300H.

e. Inhibit Whenever the user wants to stop the increase, the decrease or the resume command, he should use inhibit command. To implement this signal, we just send 00000000B to address 300H. This will inactivate all command lines to the speed control processor.

f. Terminate the program   If the user chooses to terminate the program, we restore int 1CH back to its old service routine (this will be discussed in the next section) and terminate the program.

Since the user will select these commands by using the keyboard, we will use DOS function 6 to get these commands. This function can be accessed through INT 21. The command codes will be returned to the register AL.

2. The feedback

The feedback is used to read the speed from the counters by using port B of the PPI and print the speed on the screen. We want to read the PPI periodically at a known interval of time so we can calculate the actual speed. The CPU can continually count the time and read the PPI at a specific interval of time but this will occupy the CPU. The efficient way to do this is to use the timer interrupt service routine as discussed in Chapter II. We write an interrupt service routine (read the speed counters) to replace INT 1CH. Here, we will discuss how to change INT 1CH to point to our routine. Function 35 of INT 21 will get the old interrupt vector (address). We need to get the old interrupt vector because we want to restore the old routine after we have done. To execute function 35 of INT 21, we need to move this function number (35H) into register AH and the interrupt number (1CH) into register AL. We then call INT 21. This function's call will return the old interrupt vector in ES (segment part) and BX (offset part). At the end of our program (when the user chooses to terminate the program), we restore the old interrupt vector. To change the interrupt vector table, function 25 of INT 21 will be used. We place the old vector in DX (offset) and DS (segment), the function number (25H) in AH, the interrupt number (1CH) in AL, and then call INT 21.

After we take care of the old vector, we want to change the interrupt vector table entry to point to our routine. We use function 25 of INT 21. We move the offset of our routine to register DX, the segment of our routine to register DS, the function number (25H) to register AH, the interrupt number (1CH) to register AL, and then call INT 21H.

In this way, the hardware will interrupt INT 8 every 18,2 times per second; INT 8 will call INT 1CH, which will point to execute our routine. In our routine, we will read the speed counters to determine the frequency of the input speed.

Next, we print this speed on the screen by using DOS function 10 of INT 10H. Now, the user can monitor the current speed and send the commands to change it, if necessary, by typing the corresponding code.

V. RESULT AND APPLICATION

We have implemented a speed control system. The timing diagrams of the outputs (VAC and VENT control lines) of our system are shown in Figure 19a. These two lines can be easily decoded to produce three control lines: increase, decrease, and hold speed lines. Figure 20 shows the output decoder and Figure 19b shows the timing diagram.

Figure 21 shows the block diagram of the speed control system. The input to our system will be a variable modulated frequency that need to be controlled. This frequency ranges from of 50 to 300 Hz. The outputs are three control lines: increase speed, decrease speed, and hold speed. The typical device to be controlled is shown in Figure 21. The device's output is a modulated frequency (50-300 Hz). The control is needed for changing or maintaining this frequency. The device will be able to accept input control lines that can be derived from the three control lines. Our system will be able to control the output of the device by using the control lines and the device will return the speed feedback to our system.

In fact, our system can be used to control any devices that can be modeled as in Figure 21. Here, we will discuss two major applications that utilize our system effectively:

1. Automotive cruise control
2. Electric motor speed control

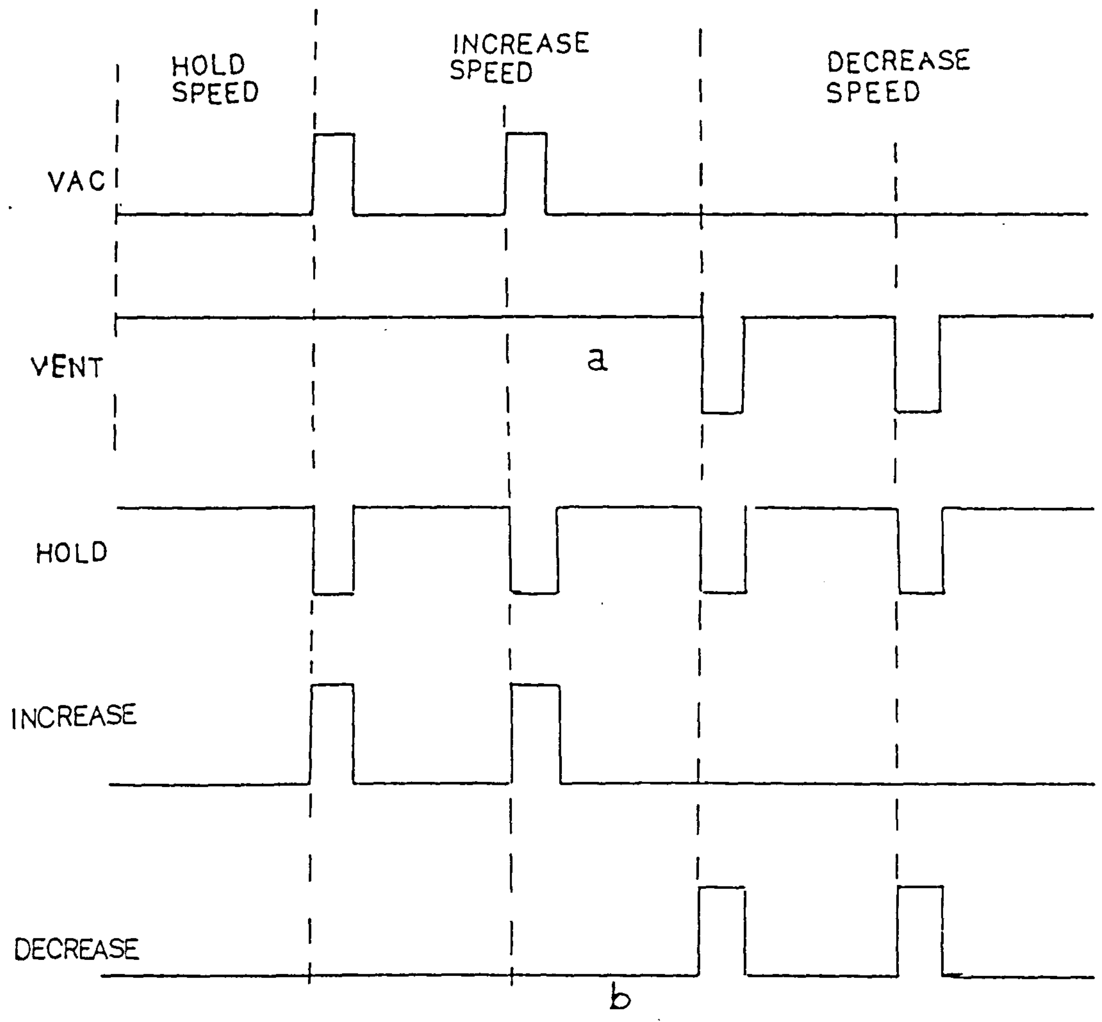


FIGURE 19. Output timing diagram

A. Automotive Cruise Control

A good example to utilize our system is the automotive cruise control. The automotive speed stays constant despite of the road condition. The feedback from the automotive is

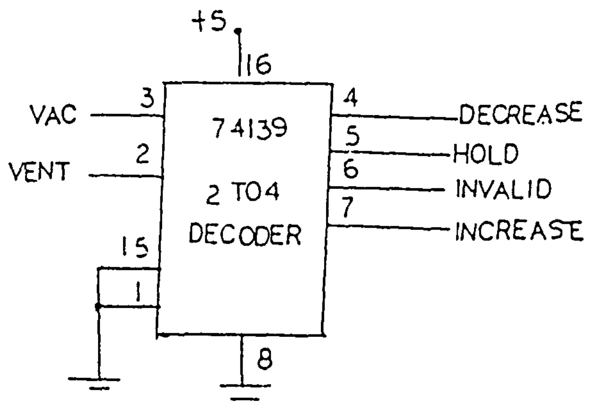


FIGURE 20. The output decoder

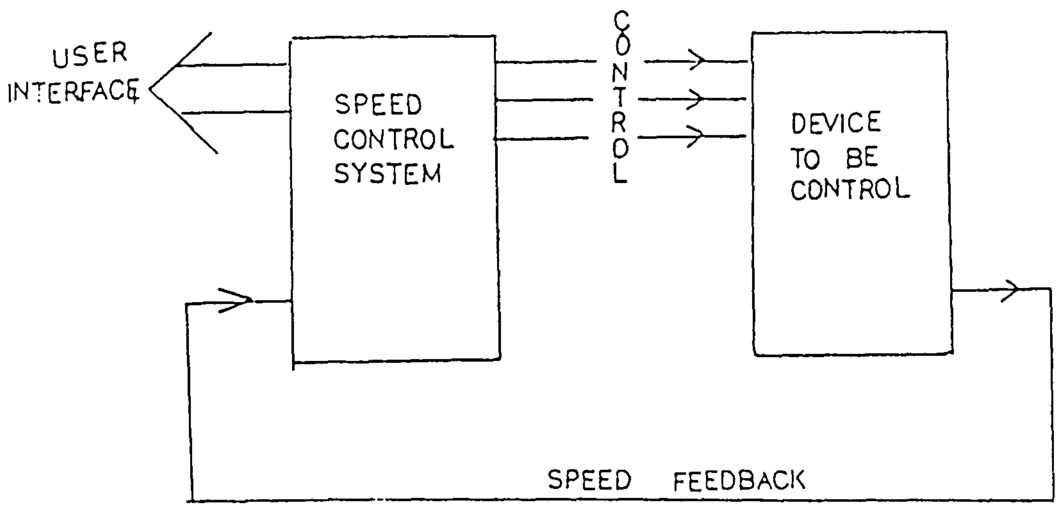


FIGURE 21. The speed control system in application

the speed measured by a speedometer. The controls are the VAC and VENT lines which will drive the gasoline throttle with an output proportional to the difference between the desired speed and the actual speed. Figure 22 shows the typical automotive cruise control application.

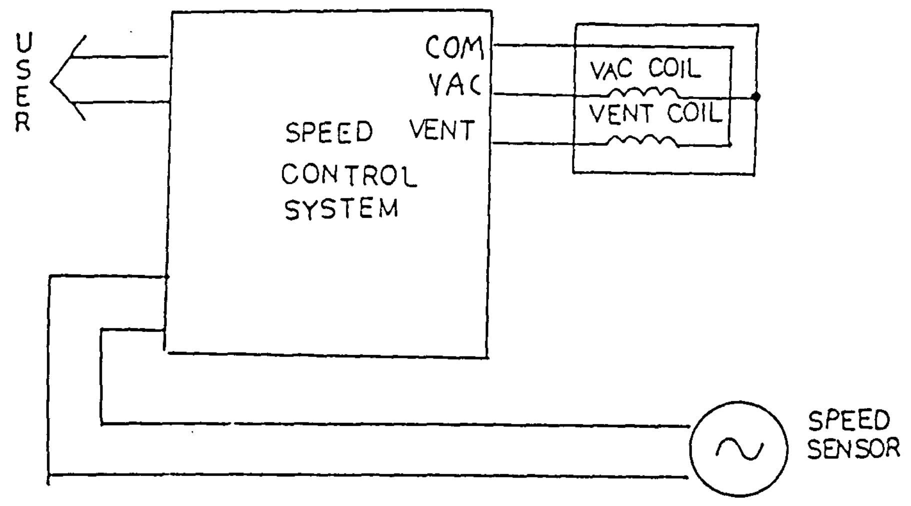


FIGURE 22. Typical automotive cruise control application

To adopt our design to the automotive cruise control, only a microprocessor system is needed instead of the entire PC system. For today's vehicle to have a microprocessor is typical. There are many applications for having a microprocessor in the vehicle beside the speed control, for example, electronic transmission control, electronic cooling system control, on-board diagnostic systems, sleep detector, etc. Without the microprocessor, both active and passive components are used in the automotive system. However, a large number of these components are needed, for example, 60 active and 120 passive components are needed in a cruise control [2].

B. Electric Motor Speed Control

In industry system control, it is necessary to be able to adjust the speed of a motor over a wide range with a good resolution and reliability. To accomplish this, a closed loop system with feedback is essential. The conventional analog control is not accurate, it produces nonlinearity in the analog speed transducer and causes error easily in the control lines due to the temperature, component aging, and noise. By using microcomputer to control the speed of these motors, we eliminate the nonlinearity in the speed transducer (digital rather than analog). The control lines can be transmitted over a long distance with no degradation. These digital control lines are not subject to temperature variation, component aging, or noises.

Most of the speed of a motor is a function of the applied voltage. For example, the speed of any dc motor can be altered by changing the voltage across the armature. On the other hand, the speed of a single phase motor can be altered by changing the applied voltage across the motor line terminals. For these types of motors, our design can be easily adopted for them. As an example, we will show a dc motor speed control design as in Figure 23.

The speed of the motor is measured optically by using light sources and photocells. The pulses from the photocell are fed into our system. The VAC and VENT output lines are decoded into 3 control lines to drive the up/down counter. To increase/resume the speed, the counter is counted up; to decrease/brake the speed, the counter is counted down. The content of the counter is converted into the analog voltage. This analog voltage is applied across the armature of the dc motor. If the counter is increased, the analog voltage also increases, and the speed of the motor will also be increased.

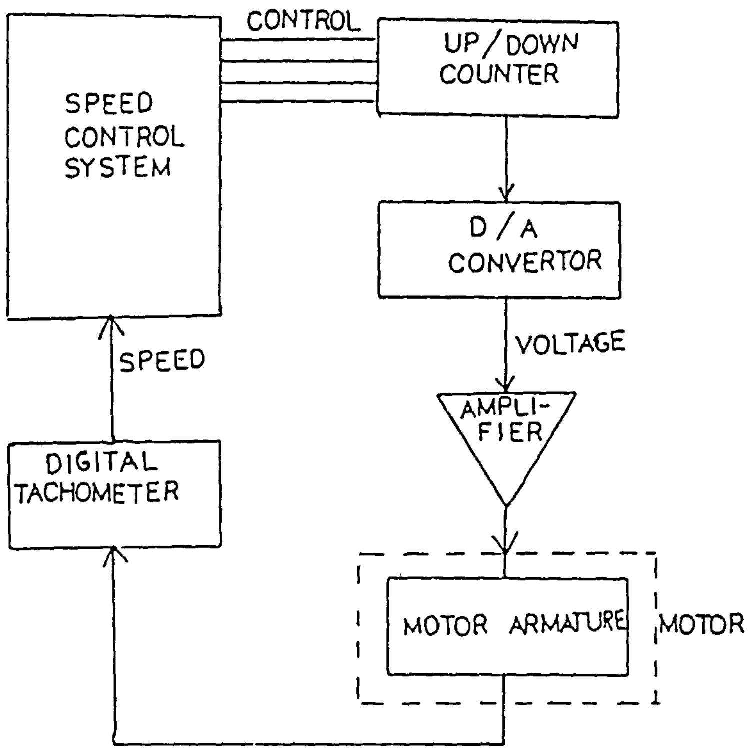


FIGURE 23. DC motor speed control block diagram

VI. CONCLUSION

As technology has advanced, computer control in industrial applications has become increasingly important. In this paper, we have discussed one kind of computer control, a speed control system. To effectively and accurately control a device (motor) with speed, it is essential to have a closed loop system (a system with the feedback) like the speed control system discussed in this paper. The speed control system will continuously sample input from its environment and generate controls to change/maintain the speed of the device.

In this paper, we have emphasized an interfacing methodology involving an IBM personal computer. The circuits needed, especially those interacting with the PPI chip and the speed control processor are discussed in detail. We also discuss the software used to control the hardware. Our experience here suggests a microcomputer control system is often better than a conventional circuit because it is easier to design with and the results are more reliable and accurate.

We intend to continue this work by implementing a dc motor speed controller and to incorporate better real-time response into the system. We also plan to extend this work into more sophisticated speed control applications.

VII. BIBLIOGRAPHY

1. Eggebrecht, L. Interfacing to the IBM Personal Computer. Indiana, IN: Howard W. Sams & CO., Inc., 1983.

2. Hordeski, M. Microprocessors in Industry. New York: Van Nostrand Reinhold Company, 1984.

3. iAPX 86/88, 186/188 User's Manual. Santa Clara, CA: Intel Corporation, 1983.

4. IBM PC Macro Assembler. Boca Raton, FL: IBM Corporation, 1983.

5. IBM PC Technical Reference. Boca Raton, FL: IBM Corporation, 1983.

6. Jourdain, R. Programmer's Problem Solver for the IBM PC, XT & AT. New York, NY: A Brady Book Published by Prentice Hall Press, 1986.

7. Kosow, I. Control of Electric Machines. Englewood Cliffs, N.J.: Prentice-Hall, INC., 1973.

8. Maloney, T. and Alvarado, F. "A Digital Method for DC Motor Speed Control." IEEE Transactions on Industrial Electronics and Control Instrumentation, IECI-23, NO.1 (February 1976): 44-46.

9. Microsystem Components Handbook. Santa Clara, CA: Intel Corporation, 1983.

10. Norton, P. The Peter Norton Programmer's Guide to the IBM PC. Redmond, Washington: Microsoft Press, 1985.

11. Sargent III, M. and Shoemaker, R. The IBM Personal Computer From the Inside Out. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1986.

12. Wadlow, T. Memory Resident Programming on the IBM PC. Reading, Mass.: Addison-Wesley Publishing Company, INC., 1987.

IX. ACKNOWLEDGEMENT

I wish to thank my major professor, Dr. C. Wright, for giving me the opportunity to do my project and helping me in preparing this thesis. I also wish to thank Dr. A. Pohm and Dr. R. Lamb for serving on my committee and Mr. G. Bridges for helping me in setting up the equipment for my project.

X. APPENDIX: THE PROGRAM LISTING

```
TITLE    SPEED CONTROL SYSTEM
; BY TING-WOEN WOEN
;
PROGSTACK           SEGMENT STACK
         DW         80 DUP (?)
PROGSTACK           ENDS
;
PROGDATA            SEGMENT
MSG      DB         'PLEASE CHOOSE :',0AH,0DH
         DB         '(I) INCREASE THE SPEED',0AH,0DH
         DB         '(D) DECREASE THE SPEED',0AH,0DH
         DB         '(R) RESUME THE SPEED',0AH,0DH
         DB         '(B) BRAKE',0AH,0DH
         DB         '(N) NEUTRAL',0AH,0DH
         DB         '(Q) QUIT',0AH,0DH
         DB         'ANY OTHER KEY TO SEE THE CURRENT SPEED',
                     0AH,0DH,'$'
MSG1     DB         'SPEED:   $'
CON1     DB         0
CON2     DB         0
CON3     DB         0
CON4     DB         0
DIF      DW         0
DIFF     DW         0
PROGDATA            ENDS
;
PROGCODE            SEGMENT
         ASSUME     CS:PROGCODE,DS:PROGDATA,ES:PROGDATA
MAIN     PROC       FAR
         PUSH       DS
         SUB        AX,AX
         PUSH       AX
         MOV        AX,PROGDATA
         MOV        DS,AX
;
;----SET THE PPI CONTROL REGISTER----
         MOV        AL,8BH
         MOV        DX,303H
         OUT        DX,AL

;----INHIBIT LOAD TO THE COUNTER----
         MOV        AL,10H
         MOV        DX,300H
         OUT        DX,AL
;
```

```
;----SET INTERRUPT ICH TO POINT TO ROUTINE----
        PUSH    DS
        MOV     DX,OFFSET ROUTINE
        MOV     AX,SEG  ROUTINE
        MOV     DS,AX
        MOV     AH,25H
        MOV     AL,1CH
        INT     21H
        POP     DS

;----CLEAR THE SCREEN----
        MOV     AH,0
        MOV     AL,2
        INT     10H

;----MOVE CURSOR----
START:  MOV     BH,0
        MOV     DH,10
        MOV     DL,0
        MOV     AH,2
        INT     10H

;----PRINT MESSAGE MSG----
        MOV     DX,OFFSET MSG
        MOV     AH,09
        INT     21H

;----RECEIVE INPUT FROM USER----
NEXT1:  MOV     AH,06
AGAIN:  MOV     DL,OFFH
        INT     21H
        JZ      AGAIN

;----QUIT ?----
        CMP     AL,'Q'
        JNE     NEXT8
        JMP     NEXT2

NEXT8:
;----NEUTRAL ?----
        CMP     AL,'N'
        JNE     NX1
        JMP     NOTH

;----INCREASE THE SPEED ?----
NX1:    CMP     AL,'I'
        JNE     NX2
        JMP     INCREASE
```

```
;----DECREASE THE SPEED ?----
NX2:    CMP     AL,'D'
        JNE     NX3
        JMP     DECREASE

;----RESUME ?----
NX3:    CMP     AL,'R'
        JNE     NX4
        JMP     RESUME

;----BRAKE ?----
NX4:    CMP     AL,'B'
        JNE     START
        JMP     BRAKE

PTT:
;----PRINT THE SPEED EVERY NEW SPEED AVAILABLE----
        CMP     CON2,1
        JB      NEXT1
        MOV     CON2,0
;----MOVE CURSOR----
        MOV     BH,0
        MOV     DH,0
        MOV     DL,12
        MOV     AH,2
        INT     10H

;----PRINT MESSAGE MSG1----
        MOV     DX,OFFSET MSG1
        MOV     AH,09
        INT     21H

;----PRINT THE FIRST DIGIT----
NX5:    MOV     BH,0
        MOV     DH,0
        MOV     DL,20
        MOV     AH,2
        INT     10H

        MOV     AX,DIF
        MOV     DIFF,AX
        MOV     CL,4
        SHR     AH,CL
        AND     AH,0FH
        CMP     AH,9
        JLE     NXT1
        ADD     AH,7H
```

```
NXT1:   ADD     AH,30H
        MOV     AL,AH
        MOV     CX,1
        MOV     AH,0AH
        INT     10H

;----PRINT THE SECOND DIGIT----
        MOV     DH,0
        MOV     DL,21
        MOV     AH,2
        INT     10H

        MOV     AX,DIFF
        AND     AH,0FH
        CMP     AH,9
        JLE     NXT2
        ADD     AH,7H
NXT2:   ADD     AH,30H
        MOV     AL,AH
        MOV     CX,1
        MOV     AH,0AH
        INT     10H

;----PRINT THE THIRD DIGIT----
        MOV     DH,0
        MOV     DL,22
        MOV     AH,2
        INT     10H

        MOV     AX,DIFF
        MOV     CL,4
        SHR     AL,CL
        AND     AL,0FH
        CMP     AL,9
        JLE     NXT3
        ADD     AL,7H
NXT3:   ADD     AL,30H
        MOV     CX,1
        MOV     AH,0AH
        INT     10H

;----PRINT THE FOURTH DIGIT----
        MOV     DH,0
        MOV     DL,23
        MOV     AH,2
        INT     10H
```

```
          MOV      AX,DIFF
          AND      AL,0FH
          CMP      AL,9
          JLE      NXT4
          ADD      AL,7H
NXT4:     ADD      AL,30H
          MOV      CX,1
          MOV      AH,0AH
          INT      10H

;----GO BACK----
          JMP      NEXT1


NOTH:     MOV      AL,10H
          MOV      DX,300H
          OUT      DX,AL
          JMP      START

INCREASE: MOV      AL,12H
          MOV      DX,300H
          OUT      DX,AL
          JMP      START

DECREASE: MOV      AL,14H
          MOV      DX,300H
          OUT      DX,AL
          JMP      START

RESUME:   MOV      AL,11H
          MOV      DX,300H
          OUT      DX,AL
          JMP      START

BRAKE:    MOV      AL,18H
          MOV      DX,300H
          OUT      DX,AL
          JMP      START

;
;
```

```
NEXT2:   PUSH     DS


         MOV      DX,OFF53H
         MOV      AX,OF000H
         MOV      DS,AX
         MOV      AL,1CH
         MOV      AH,25H
         INT      21H
         POP      DS

         RET
MAIN     ENDP
;
;
;
;----INTERRUPT SERVICE ROUTINE ICH----
ROUTINE PROC      FAR
         PUSH     AX
         PUSH     BX
         PUSH     CX
         PUSH     DX

         MOV      AX,PROGDATA
         MOV      DS,AX

;----POLL THE SAMPLING CLOCK----
         MOV      DX,302H
         IN       AL,DX
         AND      AL,01H
         CMP      AL,0
         JNE      NEXT6

;----READ THE COUNTER----
         MOV      DX,301H
         IN       AL,DX
         MOV      AH,0
         CMP      AX,0
         JE       NEXT6
         MOV      DIF,AX
         ADD      DIF,AX
```

```
;----RESET THE COUNTER----
          MOV      DX,300H
          IN       AL,DX
          AND      AL,11101111B
          MOV      DX,300H
          OUT      DX,AL
          MOV      CON1,5
LOOP:     DEC      CON1
          CMP      CON1,0
          JNE      LOOP
          OR       AL,00010000B
          MOV      DX,300H
          OUT      DX,AL


NEXT6:
;
          POP      DX
          POP      CX
          POP      BX
          POP      AX
          IRET
ROUTINE ENDP
;
;
;
PROGCODE           ENDS
          END      MAIN
```