Experiment

3

Generating Timing Sequences

Objective

The aim of this lab experiment is to generate timing sequences using software delays and programming 8253 Programmable Interval Timer (PIT) chip.

Equipment

Flight 8086 training board, the Application Board, PC with Flight86 software, download cable

Tasks to be Performed

- Generate time delays using software delays and 8253 PIT chip.
- Use generated delays to turn ON/OFF LEDs for specific amounts of time
- Generate waveforms of different frequencies, and observe them on an oscilloscope/logic analyzer.
- Interface a simple relay driver circuit to 8255 port and switch ON/OFF a device for a specific amount of time.

3.1 Background

It is often necessary to control how long certain actions last. This can be achieved using software delays, or more accurately by the use of a timer (i.e. 8253 PIT chip). In this experiment, you will learn how to generate time delays using both software delays and 8253 PIT chip. Also, you will learn how to use time delays to control the operation of some devices (e.g. LEDs and Relays), and to generate periodical waveforms of different frequencies.

3.2 Software Delays

The easiest way to generate delays is a software delay. If we create a program which loops around itself, and does this for a fixed number of times, for a given processor, running at a given clock rate, this process will almost take the same time each time the program is executed. All we have to do is to write this loop such that it takes 1 second. Then, by calling this loop *n* times, we generate a delay of *n* seconds. Notice that the time taken by the loop does not need to be 1 second. For example, a loop that takes 0.25 seconds to execute, can be called 4×*n* times (i.e. *n*/0.25) to generate a delay of *n* seconds. The question now is **how to determine the time taken by a loop**. This can be answered by the following example.

Example 3.1: Calculate the total time taken by the following loop.

```
    MOV CX, 8000h; load CX with a fixed value 8000h (32768)
L1: DEC CX       ; decrement CX, loop if not zero
    JNZ L1
```

From the 8086 data sheets, we find that **DEC CX** requires 2 clock cycles and **JNZ** requires 16 clock cycles. Thus, the total number of clock cycles required by these two instructions is **18** clock cycles.

Since the FLIGHT-86 board is running at 14.7456/3 MHz, 1 clock cycle will take 3/14.7456 microseconds, and 18 clock cycles will take **54/14.7456** microseconds. Thus, the total time taken by the loop is **32768 × (54/14.7456 × 10^{-6}) = 0.12 seconds**

The previous loop requires 0.12 seconds. Thus, this loop needs to be executed almost 8 times to generate a delay of 1 second. The following example shows how to use this loop inside a program to turn ON/OFF an LED for specific amounts of time.

Example 3.2: Write a program to turn ON an LED for 3 seconds, then turn it OFF for another 3 seconds, and repeat this cycle.

```
COMSEG SEGMENT BYTE PUBLIC 'CODE'
ASSUME CS:COMSEG, DS:COMSEG, ES:COMSEG, SS:COMSEG
ORG 0100h
Start:
         MOV   AL, 99h ; initialize 8255 ports:
         OUT   06h, AL ; A and C in, B out
         MOV   AL, 01h ; set bit 0 in AL to 1
ON_OFF:  OUT   02h, AL ; turn on/off LED 0
         MOV   DL,  25 ; delay of 25*0.12 = 3 sec
         CALL  Delay
         XOR   AL, 01h ; complement bit 0 in AL
         JMP   ON_OFF

Delay    PROC
  L1:    MOV   CX, 8000h
  L2:    DEC   CX
         JNZ   L2
         DEC   DL
         JNZ   L1
         RET
Delay    ENDP
COMSEG   ENDS
END Start
```

Run the above program on the FLIGHT-86 board and estimate the ON/OFF time of LED 0. What you conclude about the accuracy of the software delays?

3.3 Time Delays Using the 8253 PIT Chip

Software delays are the easiest way to generate time-delays, as they do not require any additional hardware. However, software delays are not accurate especially for long delays. Therefore, timers like the 8253 PIT chip are used to generate accurate delays. Figure 3.1 shows the circuit diagram of the 8253 PIT chip. It consists mainly of three identical counters (Counter0 to Counter2) and one Control Word Register.

Counting and Control Registers of the 8253 PIC Chip

The three counters are 16-bit *count down* registers, which decrement on the falling edge of their input clocks (CLK0 to CLK2). In the case of the FLIGHT-86, CLK0 to CLK2 are connected to the PCLK clock that is running at **14.7456/6** MHz. Thus, the counters will be decremented every **6/14.7456** microseconds. The three counters are loaded through the low byte of the data bus (D0-D7). Hence, two write cycles are required to load any one of the 16-bit registers. The Control Word register is used to determine the mode, size and type of count for each counter to be used.

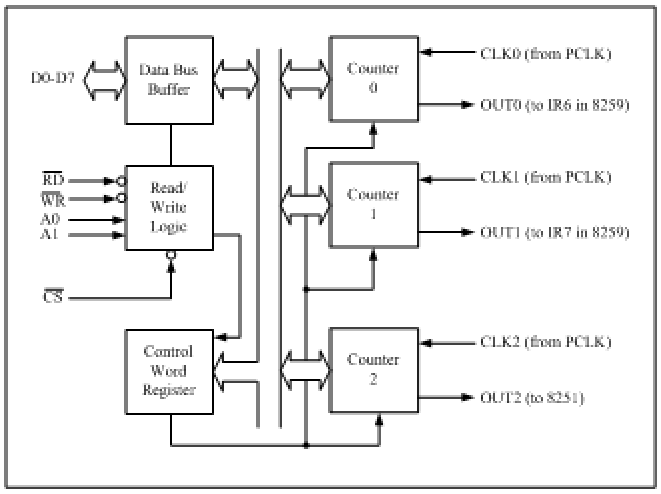

Figure 3.1: The 8253 PIT circuit diagram

Each one of the previous registers has a unique address, and can be accessed using I/O operations (i.e. IN and OUT). Table 3.1, shows the addresses assigned to four registers in the FLIGHT-86 board.

Table 3.1: The 8253 PIT chip register addresses

Register	Activity Allowed	Actual Port Address
Counter 0	Read/Write	08h
Counter 1	Read/Write	0Ah
Counter 2	Read/Write	0Ch
Control Word	Write Only	0Eh

Programming the 8253 PIT Chip

Each counter of the 8253 PIT chip can be programmed independent from the other counter by sending a control word to the Control Word Register. Figure 3.2 shows the format of the control word. Bit D0 specifies counting type (i.e. binary or BDC). Bits D3, D2, and D1 specify the counting mode. Bits D5 and D4 specify how the counter is read and loaded. Bits D7 and D6 specify the counter to be programmed (i.e. Counter 0 to Counter 2).

There are four options for reading/loading the counter:

1. **Latch Counter:** allows you to latch the current register count, and then read the counter value 'on the fly'

2. **Read/Load Least Significant Byte (LSB):** only the low byte of the counter can be read or loaded

3. **Read/Load Most Significant Byte (MSB):** only the high byte of the counter can be read or loaded

4. **Read/Load Least LSB then MSB:** allows two bytes to be read from or loaded into the counter such that the LSB comes first.

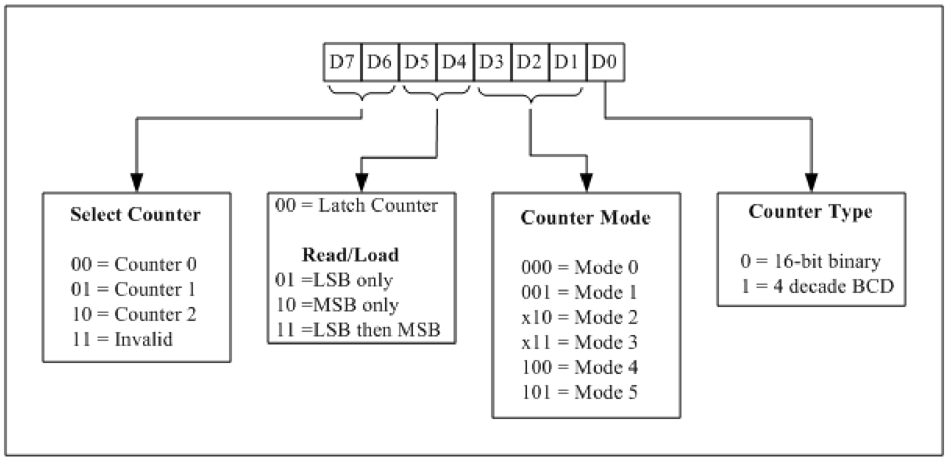

Figure 3.2: Control Word Format of the 8253 PIT Chip

As indicated in Figure 3.2, there are six counting modes:

Mode 0 - Interrupt on Terminal Count: The output goes low after the mode set operation, and remains low while counting down. When the count decrements to zero, the output goes high and remains high until then next mode is set or a new count is loaded. See Figure 3.3 (a)**.**

Mode 1 - Programmable One-shot: not available on FLIGHT-86

Mode 2 - Rate Generator: A divide by N counter. The output is low for one input clock period and then high for N clock periods. This cycle repeats until a new mode is selected. See Figure 3.3 (b).

Mode 3 - Square Wave Rate Generator: Similar to Mode 2, except that the output is high for the first half of the count and goes low for the other half. See Figure 3.3 (c).

Mode 4 - Software Triggered Strobe: The output goes high once the mode is set, and remains high while the counter is decremented. When the counter decrements to zero, the output goes low for one clock cycle and then goes high again. The output will remain high until a new mode or count is loaded. See Figure 3.3 (d).

Mode 5 -Hardware Triggered Strobe: not available on FLIGHT-86.

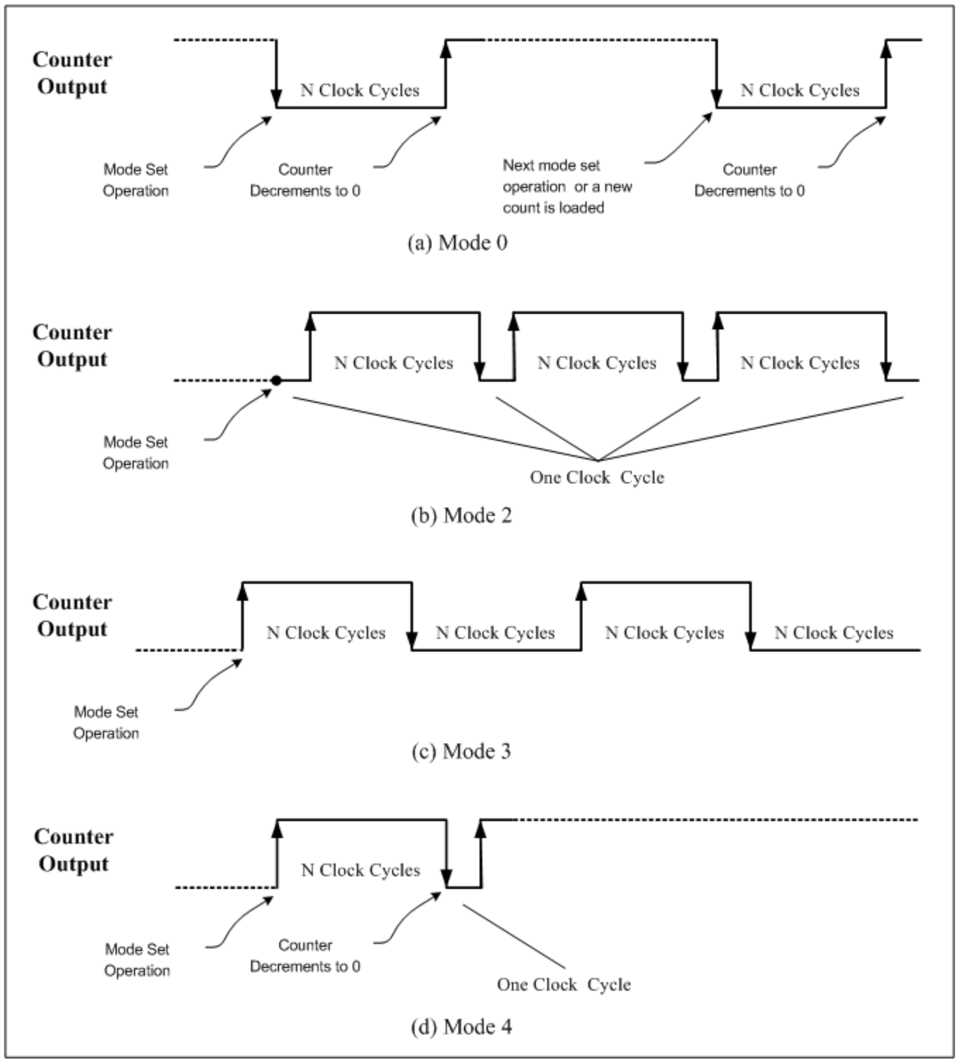

Figure 3.3: Counting modes of the 8253 PIT chip

In order to program any one of the three counters in a certain mode, you need to do two things. First, send a control word to the Control Word Register. Then, load a proper value into the counter register. These two steps are illustrated in the following example.

Example 3.3: Write an assembly code to do the following:
(1) Set Counter0 as a 16-bit binary counter operating in Mode0
(2) Load Counter0 with the proper value, such that OUT0 goes high after 0.025 seconds.

(1) The required control word is shown below:

```
 0 0 1 1 0 0 0 0 = 30h
 --- --- ----- -
  |   |    |   |16 bit binary counter
  |   |    |Mode 0
  |   |Read/Load LSB then MSB
  |Counter 0
```

(2) Since the counter clock input is connected to PCLK (**14.7456/6 MHz**), it will be decremented every **6/14.7456** microseconds. Hence, we need to load the counter with the value **$(0.025 \times 14.7456 \times 10^{-6})/6 = 61440 = \text{F000h}$**.

The following code will load the required control word (i.e. 30h) into the Control Word Register, and will load Counter0 with F000h.

```
MOV AL,  30h ; load the control word into AL
OUT 0Eh, AL  ; and send it to the Control Register

; since the 8253 PIT chip is connected to the low byte of
; the data bus, two write cycles are required to load
; F000h into counter0

MOV AL,  00h ; load the low byte of F000h
OUT 08h, AL  ; into low byte of Counter0

MOV AL,  F0h ; load the high byte of F000h
OUT 08h, AL  ; into high byte of Counter0
```

Handling the Outputs of Counter0 and Counter1

You may noticed that the outputs of Counter0 and Counter1 (i.e. OUT0 and OUT1) in Figure 3.1 are connected to inputs IR6 and IR7 of the 8259 chip respectively. This allows these two counters to operate in an interrupt driven manner. The 8259 Programmable Interrupt Control (PIC) chip accepts requests from the peripheral chips through inputs IR0 to IR7, and determines which of the incoming requests has the highest priority. Then, it issues the interrupt to the 8086 processor together with the *interrupt pointer* to enable the correct Interrupt Service Routine (ISR) to be executed. The 8259 PIC chip can be programmed to perform a number of modes of operation, which may be changed dynamically at any time in the program. Programming the 8253 PIC chip is not covered in this experiment. Instead, you will be given the necessary code to set the chip in a proper mode of operation.

When the output of Counter0/Counter1 goes high, it generates a request on IR6/IR7. The 8253 PIC chip handles this request as follows. If maskable interrupts are enabled by the instruction **STI**, the 8259 will send an interrupt to the 8086 via the **INT** line. The 8086 acknowledges the interrupt request with an $\overline{\text{INTA}}$ pulse. Upon receipt of the $\overline{\text{INTA}}$ from the 8086, the 8259 freezes the pending interrupts and waits for another $\overline{\text{INTA}}$ pulse. When the 8086 sends the second $\overline{\text{INTA}}$, the 8259 treats this as a Read pulse and places an 8-bit pointer onto the data bus. The pointers corresponding to requests on IR6 and IR7 are 38 and 39 respectively.

The 8086 processor uses the 8-bit pointer to fetch the address (i.e. *offset* and *segment*) of the corresponding ISR from the **Interrupt Vector Table (IVT)**. This is done as follows. Suppose that the 8-bit pointer is *n*, then the 8086 will fetch **four bytes** starting from the address **0000:*n**4**. The first two bytes contain the offset of the ISR, while the next two bytes contain the segment of the ISR.

Illustrative Example

The following example illustrates how to program the 8253 PIT and 8259 PIC chips to generate time delays.

Example 3.4: Write a program to turn ON an LED for 3 seconds, then turn it OFF for another 3 seconds, and repeats this cycle. Do not use software delays.

```
COMSEG SEGMENT BYTE PUBLIC 'CODE'
ASSUME CS:COMSEG, DS:COMSEG, ES:COMSEG, SS:COMSEG
ORG 0100h
Start: ; set the extra segment to point to the
       ; base of the Interrupt Vector Table (IVR)
        XOR AX,AX
        MOV ES,AX

       ;store the offset of ISR in the IVT
       MOV WORD PTR ES:[38*4],OFFSET IR6_ROUTINE

       ;store the segment of ISR in the IVT
        MOV WORD PTR ES:[38*4+2],CS

       ; initialize the 8255 PPI chip:
       ; A and C input ports, B output port
        MOV AL, 99h
        OUT 06h, AL
       ; initialize the 8259 PIC chip
        MOV AL, 17h
        OUT 10h, AL
        MOV AL, 20h
        OUT 12h, AL
        MOV AL, 03h
        OUT 12h, AL
        MOV AL, 3Fh
        OUT 12h, AL

       ; initialize 8253 PIT chip (00110110 = 36h)
       ;Counter0, load MSB then LSB, mode 3, binary
        MOV AL, 36h
        OUT 0Eh, AL

       ; counter loaded with F000h for 25 ms delay
        MOV AL, 00h
        OUT 08h, AL  ; first load low byte
        MOV AL, 0F0h
        OUT 08h, AL  ; now load high byte

       STI          ; enable 8086 maskable interrupts
       MOV DL, 120  ; count for 120 interrupts (3s)

       ; start of main program

       ; switch off all LEDs
        MOV DH, 00h
        MOV AL,  DH
        OUT 02h, AL
Again: JMP Again   ; wait for interrupt on IR6
                   ; (Counter0 decrements to 0)

; Interrupt Service Routine (ISR) for IR6
; this routine toggles ON/OFF LED 0 every 3 seconds
IR6_ROUTINE:

        DEC DL       ; decrement interrupts counter
        CMP DL,0     ; if counter < 120
        JNZ Return   ; then exit ISR
        XOR DH, 01h  ; else toggle LED0
        MOV AL,  DH
        OUT 02h, AL

        MOV DL, 120  ; count for 120 interrupts (3s)

Return:IRET

COMSEG ENDS
END    start
```

In the previous program, lines 6 and 7 set the ES segment to 0000h, which is the base address of the IVT. Lines 9 and 12 load the starting address of the ISR (IR6_ROUTINE) into the IVT. This routine will handle any request on IR6. Lines 16 and 17 initialize the 8255 PPI chip. Lines 19 to 26 initialize the 8259 PIC chip. Lines 29 and 20 initialize the 8253 PIT chip. Lines 32 to 35 load the Counter0 with the value F000h. This will generate an interrupt every 25 ms (120 interrupts every 3 seconds). The main routine starts by setting all LEDs off by sending 00h to port B (Lines 40 to 42), and waits for an interrupt on IR6 (Line 43). Upon receipt of the interrupt, the control is transferred to IR6_ROUTINE (Line 47). This routine toggles LED0 every 120 interrupts (i.e. every 3 seconds).

Exercises

3.1. Consider the following loop:

```
         MOV CX, Y
    L1:  DEC CX
         JNZ L1
```

What value of Y makes the loop executes in 0.225 seconds?

3.2. Modify the program in Example 3.2 such that Counter0 is set in Mode 0

3.3. Generate square waveforms with the following frequencies:

a. 100 KHz

b. 10 KHz

c. 1 KHz

3.4. Interface a simple relay driver circuit to 8255 port, and write a program switch ON/OFF a lamp every 10 seconds.

3.5. Write a program to simulate a traffic light controller (home assignment)

3.6. Write a program to simulate a lift controller (home assignment)