Your pc is a programmable event timer capable of precision gating up to 55ms. Alan Bradley describes how to configure it.

Event timing via the pc

Timer channel 2 is present in all pcs. It is normally used as a variable frequency square wave generator for the pc's speaker. But this timer can also be used as an interval timer that is independent of the processor type and speed. This article describes how.

Overview

Timer 2 is within the pc's programmable universal counter/timer. This IC contains three counter/timers each with an associated control register. The original IBM pc used an Intel *8253*, the IBM AT an Intel *8254*. Modern clones may use custom ICs, but all have the same programming model. This universal counter/timer operates at 1.193MHz irrespective of the processor's type or clock speed. All three counters are 16 bits wide.

The three counter/timer channels, namely 0, 1 and 2, are accessed through ports 40_{16}, 41_{16} and 42_{16} respectively within the pc's i/o port address space. The command register is located at port 43_{16}. It selects the mode for reading and writing values to the chosen channel, selects the type of use for that channel, and selects the channel to which the previous selections apply. Examples of uses for the channel are square wave generation, one shot pulse production and terminal down count.

Applying the counter/timer

Timer channel 0 is used to calculate the time of day. This channel is set up by the bios to give 18.2 pulses per second. Each pulse causes the timer interrupt, IRQ 0, after which the counter is reset. A four-byte counter for these timer pulses is stored in the bios data area at $0040{:}006C_{16}$. This counter also synchronises disk operations. Reprogramming it might therefore damage disk reads and writes.

Timer channel 1 is used by ram refresh and also by disk operations. Reprogramming this channel may also cause loss of disk data.

Timer channel 2 is connected to the pc's internal speaker, generating the variable frequency square waves necessary to make simple sounds. The speaker can be turned on and off via the pc's parallel-peripheral interface chip. As this channel controls no vital hardware, and the speaker can be turned off, it can be set up as a timer. A possible use is determining the waiting period for an analogue to digital conversion.

The 8255 programmable peripheral

Timer 2 is also controlled by the pc's *8255* peripheral interface chip, or PPI. This device

Applying the timer

This pseudo-code program illustrates how the pc can form a monostable multivibrator by using timer-channel 2 in conjunction with the parallel printer port. The printer port is provides the digital i/o lines for the trigger and o/p.

This pc monostable is not retriggerable, although it could easily be made so. It has a timed period of 40ms. The parallel-port input line that normally signals printer error, abbreviated PE, is used as the monostable trigger input. If it goes low, the monostable is triggered. Parallel port output line D_0 is used as the monostable output. It goes high when triggered, remains high until chosen time period has passed.

Pseudo code for a 40ms monostable multivibrator

```
Find location of printer port registers.
Reset monostable: set its output low: ie set printer port
   output data line D0 low (pin 2 on D connector).
Set up PPI B register to allow timer-channel 2 to be used
   as a down counter.
Set up Timer channel 2 to select down count mode, a binary
   count, and READ/WRITE msb and lsb consecutively mode.
For a downcount from FFFF16,
   Calculate FinalTimerDownCount for a 40ms delay.
Print title message.
WHILE (forever).
        IF printer port input pin PE (pin 12 on D
   connector) is low then:
   Trigger monostable output: ie set printer port data
   output line D0 high.
   Load counter with maximum count value (FFFF16) and
   start down count.
   Print "triggered: output (D0) goes high"
        Wait for 40ms.
        Reset monostable: ie set printer port D0 (pin 2
   on D connector) low.
        Print "Reset: output (D0) goes low"
              END_IF
END_WHILE
END
```

The pc parallel port

A pc can have up to three parallel printer ports LPT1, LPT2, and LPT3. Each port interface has three 8bit registers, the data latch, the status register and the control register.

Data latch: writing to this register causes the byte sent to be latched and appear on the parallel port's 25-way D connector on pins 2 to 9. Normally reading this register returns the contents of the latch.

Status register: this register represents input lines from the printer with functions as follows: b- BUSY, b_6 ACK, b_5 PE b_4 SLCT and b_3 ERR. Bits $b_{2\text{-}0}$ are unused. This is a read only register. The BUSY input is inverted between the D connector and the register.

Control register. Bit functions of this register are, b_4 IRQ DISABLE, b_3 SLCTINP, b_2 INIT, b_1 AUTOFEED, b_0 STROBE. Bits $b_{7\text{-}5}$ are not used.

This is a latch holding printer control signals. Interrupt is disabled on a falling ACK input when b_4 is low. I always disable this interrupt as it is rarely used by printer software so the associated IRQ channels 5 and 7 are considered free, and available for other expansion cards.

STROBE, AUTOFEED and SLCT INPUT are inverted between the register and D connector output pins although this inversion is corrected again when the control register is read.

controls the keyboard and is used to obtain information about the pc's configuration. It also controls the pc speaker and the speaker's associated timer on channel 2.

Port A of the PPI is a read/write port associated with the keyboard. Port B controls the reading mode for ports A and C. It also controls the speaker and timer channel 2. Port B is located at port 61_{16} in the pc's i/o address space.

Using timer 2 as an interval timer

I wrote the interval timing code in a mixture of C and assembly language. This is because the C compiler generated a much slower shift-by-8 loop. It did not make use of the *80x86* processor's ability to treat 16-bit data registers as 8-bit pairs, ie AX=AH+AL, BX=BH+BL, CX=CH+CL and DX=DH+DL.

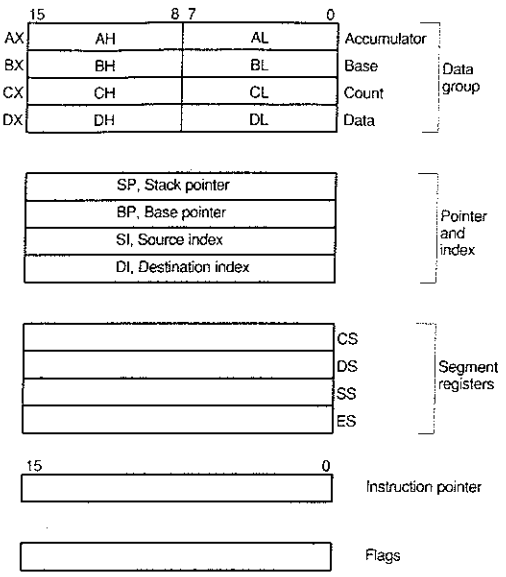
The *80x86* has four more registers, SP, BP, SI and DI. These are normally used as pointer and index registers. In my program I used the DI register simply to store 16-bit data. Turbo C uses SP, BP and SI itself. The DX data register is also used in some IN/OUT instructions results and so cannot be used by my program.

The *8086* has four segment registers, namely CS, DS, SS, ES. These allow addressing over 64K. The C compiler sets these to appropriate values automatically. The *8086* has an instruction pointer, similar to a program counter. It also has a FLAGS register, recording the result of instructions such as NON-ZERO and OVERFLOW.

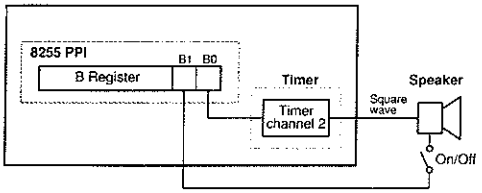
Counting is interrupted if the GATE2 input is switched to a low level and restarted when the GATE2 input is switched back to a high level. Hence GATE2 should be high for a down counting interval timer.

Therefore my program sets bit 0 of the PPI B register to logic 1. I also disable the pc speaker by setting bit 1 of the PPI B register to 0. **Inset 1** shows the Timer control register bit pattern required to select down count mode for Timer channel 2 and the timer control register bit pattern required to perform a latch operation on Timer channel 2.

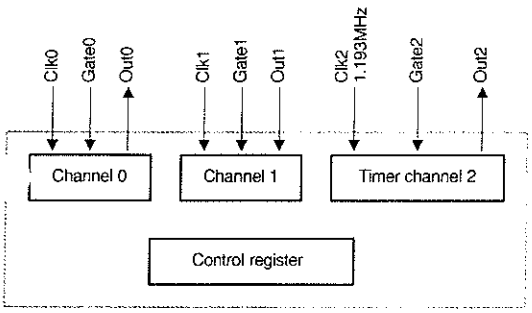
After the latching command has been writ-



80x86 registers, common to all IBM pcs. My C compiler could not treat 16-bit registers as 8-bit pairs, and a shift-by-eight loop is much slower than it needs to be. I used assembly language instead.



Standard on all pc compatibles, Timer channel 2 of the 8253 universal counter/timer ic forms the speaker driver which is controlled via Port B of the 8255 programmable peripheral interface ic but, as can be seen, the speaker can be switched off and the timer put to better use.



Block diagram of the pc's 8253 programmable universal counter/timer. Timer channel 2 counting is enabled when GATE2 is taken high.

Programmable peripheral interface

B register bit usage
PPI B register PC i/o address is 061_{16}.

Bit	Purpose
0	Set Timer channel 2 gate i/p high or low
1	Link/unlink timer 2 o/p from speaker: 0=off if unlinked
2	Must be 0
3	Read high or low dip switches
4	0=enable ram parity check, 1=disable
5	0=enable i/o channel check
6	0=hold keyboard clock low
7	0=enable keyboard, 1=disable

Control register bit usage for the pc's 8253 timer ic

PC I/O address of timer control register is 043 hex.

Location / Function	Bits 7,6	Bits 5,4	Bits 3,2,1	Bit 0
	Select counter channel	Select latch operation or type of read/write	Mode	Select binary or BCD count
Code	Number of the channel to be programmed (0, 1 or 2)	00 Latch current value of counter (done before a read operation) 01=Read/Load lsb 10=Read/Write msb 11=Read/Write lsb followed by msb	Select counter/ timer mode 000=Terminal down count 001=Programmable one shot x10=Rate Generator x11=Square Wave Generator 100=Software triggered strobe 101=Hardware triggered strobe	1=BCD 0=Binary

C pseudo code for pc timing

Calculate final count value for a 10μs delay. Calculate number of clock ticks in a 40,000μs delay.
PRINT"Preparing Count Down"
FINAL COUNT VALUE FOR 40,000μs DELAY:=FFFF–(number of clock ticks in a 40,000μs delay)
Now set up Port B of the PPI so that Timer2 can be used as a terminal down counter and disable speaker: ie set Timer gate 2 high (via PPI Port B, bit 0) (allow counting in Timer channel 2) and disable speaker (via PPI Port B, bit 1)
Set Timer Gate 2 HIGH (allow counting)
Now set up channel 2 of the timer to count in binary, perform a terminal down count and choose option: read/write lsb, msb one after the other, by writing appropriate value to the timer control register.
PRINT"Starting Count Down"
write FF_{16} (lsb) to timer channel 2
write FF_{16} (msb) to timer channel 2
REM: down count from $FFFF_{16}$ has now begun.
Loop until 10μs has passed
Print"Ten microseconds have now passed"
Loop until whole 40,000μs delay has elapsed.
PRINT"Entire chosen timed interval of 40,000μs=40ms has now passed"
END

ten to the timer control register, the latched value can be read from timer channel 2 (042_{16}), least-significant bit first. According to the *8253* data sheet, a counter value of 0000 can not be read.

Programming the chosen delay

First calculate the number of counter clock ticks that equal the chosen delay interval. This is the number which the counter must count down past before the chosen interval has passed. The number of timer clock ticks before interval has passed is equal to the chosen delay interval in microseconds, multiplied by 1.193MHz.

I always start the down count from $FFFF_{16}$. In this way, when the current counter value is less than or equal to $FFFF_{16}$ minus the number of clock ticks before interval has passed, then the chosen delay interval will have expired. For example, for a 40,000µs delay:

No. of clock ticks=
$40{,}000 \times 1.193 = 47720 = BA68_{16}$

Final timer downcount=
$FFFF_{16} - BA68_{16} = 4597_{16}$.

The maximum delay is 55ms.

Example timing program in C

This timing program example is derived from my printer port sound sampler program, where I needed to wait 10µs for the a-to-d converter to complete a conversion, process this value, then wait until the end of the sample period before repeating the loop.

Outlined in the timer software panel **Inset 2**, this routine waits until the first 10µs of a 40,000µs delay has elapsed, then waits until the whole 40,000µs delay has passed.

C and Assembler details of Timer.C

The program compiles under Borland Turbo C++ and Borland C++. This allows the *80x86* registers to be used by name within a C program's asm{ } assembly blocks. Registers can also be accessed from C by preceding the register name with an underscore, eg _AX, _AH, _AL...

The program uses # defines to give PPI Port B, and timer control register and timer channel 2 port addresses meaningful names.

Timer channel 2 is read by sending a latch command via the Control register, then simply reading the lsb, then the msb from Timer channel 2, port 042_{16}. ■

Software on disk

The Timer.c routine and the full monostable example in c can be obtained by sending a cheque of postal order for £7.50 to *Electronics World*'s editorial offices. Please mark the envelope Timer software and make your cheque payable to Reed Business Publishing Group.

Timer control register usage for interval timing.

PC i/o address 043_{16}.
Using terminal down count mode, this is timer control register bit usage to set up a down count.

Bit	7	6	5	4	3	2	1	0
Setting	1	0	1	1	0	0	0	0
Function	Select timer channel 2		Select type II read/write operation: LSB is first written to timer 2 (042_{16}) then MSB is written to timer 2 (042_{16}) counting begins as soon as MSB is written		Select mode 0: terminal down count			Select binary counting

Timer control register bit usage to perform a latch operation, current counter value prior to a read operation.

Bit	7	6	5	4	3	2	1	0
Setting	1	0	0	0	0	0	0	0
Function	Select timer channel 2		Select latch current value of counter operation		Select mode 0: terminal down count			Select binary counting

Accuracy of the pc as a timer

Instructions for reading and checking the current value of timer channel 2 take a finite amount of time, adding inaccuracy to the timing. I have calculated the worst-case delay between timer-channel 2 reaching its chosen final count value. With the program Timer.c, described later, the worst case inaccuracy for an 8MHz ISA i/o bus pc should be 7.144µs:

Calculation of Timer.c program timing limits:

	80x86 instr	8MHz i/o cycle	80386 cycle
START TIMER channel 2 count:	mov reg, immed		2
	OUT immed,al	10	
	OUT immed,al	10	
Send Counter latch command:	mov reg,reg		2
	OUT immed,al	10	
Read and store timer lsb:	in al, immed	12	
	mov reg, reg		2
Read and store timer msb:	in al, immed	12	
	mov reg, reg		2
Compare msb:lsb of current timer value with FinalTimerDownCount:	cmp reg, reg		2
'WHILE': (no jump occurs)	ja 8bitdisplacement		3

Total clock cycles=54 i/o clock+13 processor clock cycles
For a 33MHz 386 pc this would give a worst-case error of:
inaccuracy = i/o clock delay+processor clock delay
= 6.75µs+0.4µs =7.144µs

According to my assembly language book, IN/OUT 386/486 instructions vary in the number of clock cycles they need. I have used the slowest timing, which is similar to that of the 8086/8. The 286 IN/OUT instructions are about twice as fast as those of an 8086/8.

Greater accuracy in small delays would need interrupt programming, which is more difficult to write. This would involve reprogramming timer-channel 0, which needs care to avoid affecting disk operations; timer-channel 2 has no associated interrupt.

Small delays are often needed. When reading an a-to-d converter value for example, the 7µs inaccuracy can be important. In longer delays the 7µs may be insignificant. The inaccuracy should reduce on local-bus machines and on fast ISA buses that unofficially run at 11MHz.

C code for controlling the pc's timer

This program, Timer.c, demonstrates using timer 2 as a down counter to measure time intervals. Written in Turbo c with in-line 8086 assembler. Small memory model: 64K code 64K data and stack. Register keyword enabled.

```
#include <stdio.h>
#include<dos.h>
#include<conio.h>
#include<stdlib.h>
#include<io.h>

/* Prog PPI port B, TimerControl, & Timer2 regs */
#define PPIportB      0x061
#define TimerCtlReg   0x043
#define Timer2        0x042

/* 'register' vars: #defs for direct access to  */
/* DI,CX,CH,CL via meaningful names */
 #define FINALTIMERDWNCOUNT DI
 #define MSB CH
 #define LSB CL
 #define COUNT CX

/* if counter starts at FFFF16 then : */
/* (val of cntr after 10µs)≥FFFF-(10µs*1.193MHz) */
/* thus tenMicroSecDwnCount=(FFFF-C)hex=FFF3hex */
#define tenMicroSecDwnCount  0xfff3

/* clock frequency is 1.193MHz */
/* #def to set No of 1.193MHz clock tick decrements)*/
/* to pass before chosen timed interval has passed: */
/* noofclockticks=chosen interval in µs*1.193MHz */
/*              =  40,000*1.193=47720dec=BA68hex */
#define NoOfClockTicks_inFortyMilliSecDelay  0xba68

/* temporary store for B register of PPI IC */
 unsigned char breg;
int main() {
   puts("\nPreparing count down\n");

/* set FINALTIMERDWNCOUNT: /*
/* = 0xffff - NoOfClockTicks_in... delay */
   asm {
      mov ax, 0xffff;
      sub ax, NoOfClockTicks_inFortyMilliSecDelay;
      mov FINALTIMERDWNCOUNT, ax;
   }

/*Set portB of PPI for timer2 as down counter instead
of driving speaker */
/* 1st get a copy of PPI port's B register */
   breg = inp( PPIportB  );

/* logic OR with 0000 0001 to set bit 0 (timer gate)
high */
   breg = breg | 0x01;

/* AND with 1111 1101 to set bit 1 (spkr data) off */
   breg = breg & 0xfd;

/* set up 8255 PPI port B for speaker off & timer gate
high */
   outp( PPIportB , breg );

/* Now set up channel 2 (Timer2) of Timer chip:—  */
/* send 1011 0000 to Timer control reg to select: */
/* Channel2,oper 11 (r/w both h&l
bytes),terminalcount,binary data */
   outp( TimerCtlReg, 0xb0 );
   puts("\nStarting count down\n");

/* set value (FFFF) from which timer counts down */
   asm{
/* first o/p count low byte (FF) */
      mov al, 0xff;
      out Timer2, al
/* o/p high byte (FF); timer starts on writing high
byte */
      out Timer2, al
   }
/* down count from ffff has now started */

/* now wait until ten microseconds has passed */
/* 10       00            000           0 is 8 hex*/
/* 10:ch 2, 00:ctr latch, 000:term cnt, 0:bin data*/
/*store above in ah for speed*/
   asm{ mov ah , 0x80 }

/*loop until 10µs or more has passed */
   label1DO: asm
   {
/* read ctr latch command from ah into al for I/O
instr */
         mov al, ah
/*now send latch command to timer*/
         out TimerCtlReg, al
/* now read lsb from timer2 */
         in al, Timer2
/* now store lsb */
         mov LSB, al
/* now read msb from timer2*/
         in al, Timer2
/*now move msb into highbyte of COUNT */
         mov MSB, al
/* while COUNT holds value > tenMicroSecDwnCount */
/* ie while timer2 value  > tenMicroSecDwnCount*/
         cmp COUNT, tenMicroSecDwnCount
         ja label1DO
   } /* WHILE COUNT > tenMicroSecDwnCount */
   puts("\nTen microseconds has now passed\n");

/* now wait until end of chosen interval */
/* (wait for count down past FINALTIMERDOWNCOUNT)*/
/* 10:chann 2, 00:counter latch command, */
/* 000:terminal down cnt, 0:bin data */
/* store above 10000000b byte in ah for speed*/
   asm{ mov ah , 0x80 }
   label2DO: asm
   {

/* read ctr latch command from ah into al for i/o
instr */
         mov al, ah
/* send latch (0x80) command to Timer*/
         out TimerCtlReg, al
/* read lsb from timer2 */
         in al, Timer2
/* store lsb in low byte of count*/
         mov LSB, al
/* read msb from timer2*/
         in al, Timer2
/*move msb into highbyte of count */
         mov MSB, al
/* while timer2 value > FINALTIMERDWNCOUNT*/
         cmp COUNT, FINALTIMERDWNCOUNT
         ja label2DO
   }
/* NOTE ! timer count value of 0000 cannot be read */

   printf(
   "\nEntire chosen timed interval of 40,000µs =
     40 msecs has now passed\n\n");

   return 0;
}
```