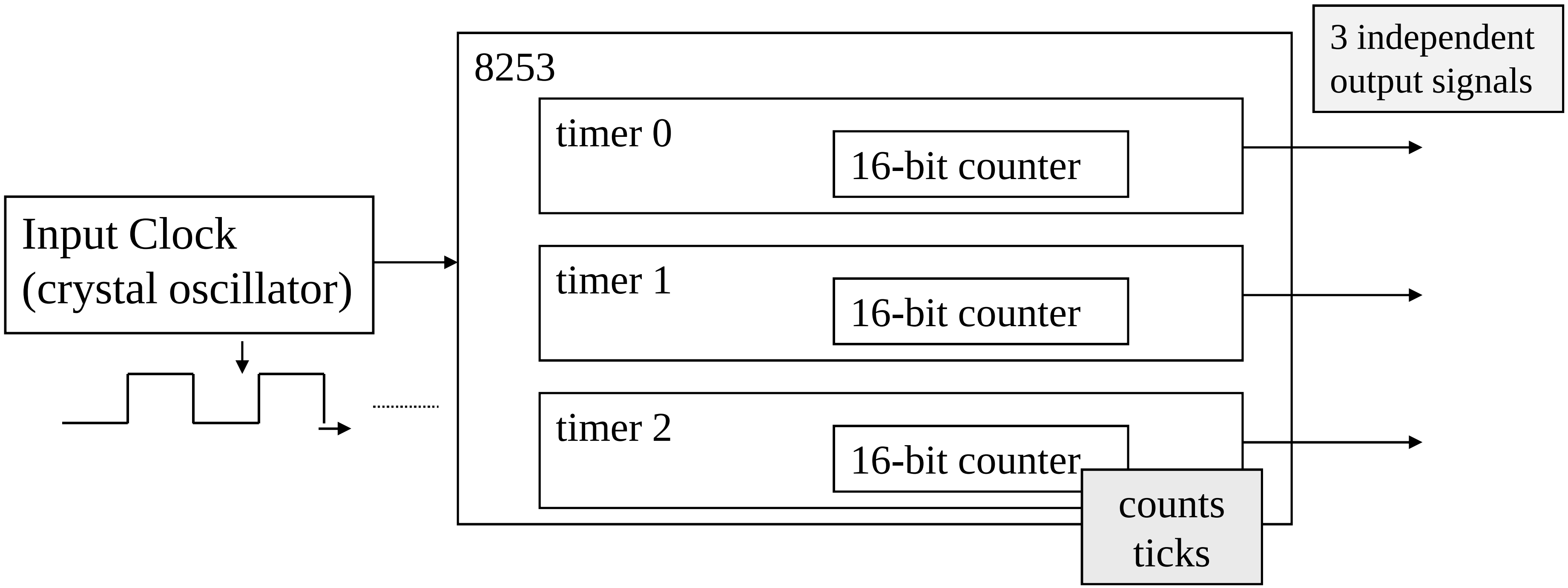
The Timer

SYSC-3006

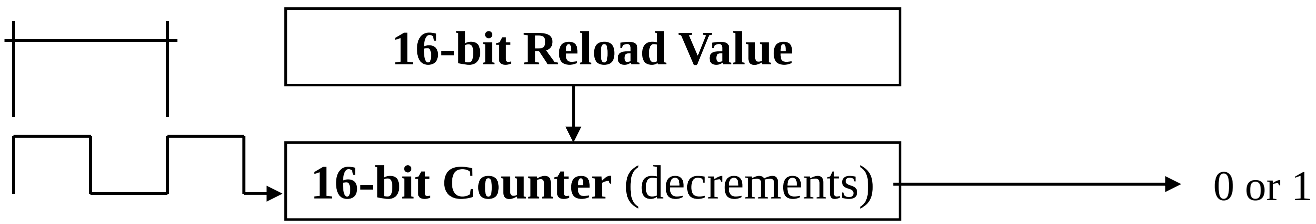
Intel 8253 Programmable Interval Timer (PIT)

- 8253 component has
  - Each timer counts ticks on the
  - Each timer generates its own output signal
    - Can be programmed for one of 6 modes that determine the shape of the output signal.



Basic Timing Function of 8253

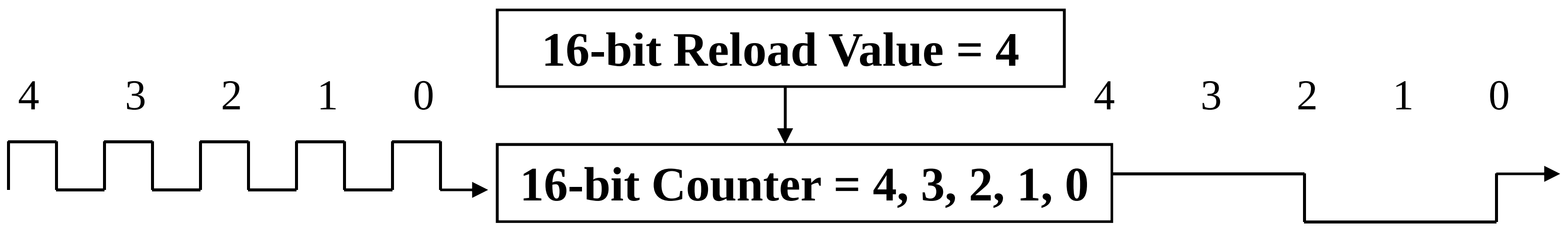
- Each timer's counter is a **16-bit unsigned value**
- Timer **decrements counter every tick** on input clock
  (typically from crystal oscillator)
- Counter reaches zero: output signal changes (if 1, now 0; if 0, now 1)
- "Modes" automatically reload counter and start again,
  => digital patterns.



- Counter value related to the **period of the output signal**

Square Wave Generator Mode (Mode 3)

- In this mode, 8253 generates a **square wave** on output signal.
  - Square wave: approx. 50% duty cycle
  - Master clock input signal “ticks”: timer counter decremented
  - Counter reaches half of original value: output signals toggle
  - Counter reaches 0: output signal toggled and **counter reloaded**.
  - **Last value written** to counter used as **reload** (original) value



Square Wave Generator Mode (Mode 3)

- Output: **scaled down version** of input clock
  - One output cycle every *n* input cycles (*n* called the scaling factor)

$$\textbf{output freq.} = \textbf{input freq.} \div \boldsymbol{n}$$

- Usually, we need to find scaling factor to *program* timer component
  - Determine initial counter value to generate desired output frequency

$$\mathbf{n} = \textbf{input freq.} \div \textbf{output freq.}$$

8253 PIT Programmer's Model

- PIT: I/O device with four 8-bit ports (registers)
  - On the PC: port addresses are 40H → 43H

- Three **8-**bit **Data Registers**

Timer 0 Counter Register	40H
Timer 1 Counter Register	41H
Timer 2 Counter Register	42H

- **Question**: How can 8-bit data registers be used to initialise 16-bit internal counters ?

8253 PIT Programmer’s Model

Control Register (43H) write-only

7 0

SC1	SC0	RL1	RL0	M2	M1	M0	BCD

timer select — read/load sequence — mode

SC1	**SC0**	**timer select**
0	0	select timer **0**
0	1	select timer **1**
1	0	select timer **2**

8253 PIT Programmer's Model

RL1	RL0	read/load sequence
0	1	read/load LSB only
1	0	read/load MSB only
1	1	read/load LSB first, then MSB

M2	M1	M0	mode (6 modes)
x	1	1	mode 3: square wave generator (x = don't care) (other modes in ELEC-3601)

BCD

0	16 bit binary count
1	BCD count (4 decimal digits)

8253 Hardware Configuration and Limits on the PC

Generate timer interrupts!

- On a PC, 8253 wired such that
  - master input signal = 1.19318 MHz
  - output from timer 0 connected to IR0 on PIC

- Scaling factor: unsigned number from 0 … FFFFh (65,535)
  but …0000H = 65,536 (i.e. 10000H, with implied MS bit)

- **Question** : What is the maximum frequency ?

- **Question** : What is the minimum frequency ?

  min output = input ÷ max scaling factor

  = **1.19318 MHz ÷ 65,535**

  = 18.2 Hz

Hardware Timing Example : DOS Time-of-Day

- The DOS maintains the time-of-day feature HH:MM:SS
  - > date
  - Provide file timestamps.

- DOS uses Timer 0: real-time clock interrupting at a frequency of 18.2 Hz.
  - DOS boots: Timer 0 programmed for Mode 3 (square wave) and "*initial time*" is loaded
  - The Timer 0 ISR counts "*ticks*" (at 18.2 Hz)
  - "*initial time* + *ticks*" used to calculate "current" time-of-day

Let’s Review : Hardware Timing Example

1. How was the 8253 programmed ? (Which timer, what mode?)

   Write to the timer’s control register: (at 43H)

   - SC1 SC0 = 00 (select timer 0)
   - RL1 RL0 = 11 (LS byte then MS byte)
   - M2 M1 M0 = 011 (square wave)
     - could also use M2 M1 M0 = 111
   - BCD = 0 (16 bit binary count)

select load mode BCD

Control Register : 00 11 011 0 → 36H

Let’s Review : Hardware Timing Example

2. What frequency was programmed ?

   Determine initial (reload) counter value:

   scaling factor = input freq. ÷ output freq.
   = 1.19318 MHz ÷ 20 Hz
   = 59,659 (decimal) = 0E90BH

- After writing control register, write timer 0 counter value (at 40H)
  - First write LSB: 0BH
  - Then write MSB: E9H

Let's Review : Use of STI/CLI *in Main Program*

```
; Main program
  ; count = 0;
  ; Save original vectors
  CLI
  ; Install new ISR at INT 8
  ; Enable interrupts

  . . .

         CALL   get_ticks       ; Returns tick count
         ; count_high->dx count_low->ax
                                ; Use value as needed
   . . .
```

Why ?

Let’s Review : Use of STI/CLI *in Main Program*

```
get_ticks    PROC    NEAR
        CLI     ; Lock out ints while accessing shared data
                ; (ensure mutual exclusion)

        MOV     DX , count_high     ; Return 32 bit count
        MOV     AX , count_low

        STI     ; Re-enable ints
        RET
get_ticks     ENDP
```

event-driven thinking !

Event-Driven Thinking : An Interference Scenario

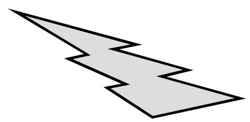
- Suppose CLI / STI protection not there , and:

```
count_high =  0010H
count_low   = FFFFH
```

What's special about this value ?

- Suppose main program was executing `getTicks()` with the following:

```
MOV    DX , count_high
MOV    AX , count_low
```



- Suppose Timer interrupt occurs during /after `MOV DX, count_high`
  - At this point, `DX = 0010H`
  - For count to be correct, `AX = FFFFH`
- BUT . . .

Event-Driven Thinking : An Interference Scenario

- In response to the interrupt, timerISR increments count, to become:

```
count_high =  0011H
count_low  =  0000H
```

- When main program resumes, it executes `MOV AX,count_low`

  `DX = 0010h` (from before interrupt)

  `AX = 0000H` (from after interrupt)

- The value returned from `get_ticks()` is:
  - count_high **before** Timer interrupt `DX = 0010H`
  - count_low **after** Timer interrupt `AX = 0000H`