

University of Portsmouth
Faculty of Technology
Department of Electronic and Computer Engineering

Module: *Digital Electronics and Microprocessors*
Module Code: *B122*
Module Topic: *Microcontroller Applications*
Lecturer: *Branislav Vuksanovic*

Lecture Notes:

Programming Timers on 8051

The purpose of this handout is to explain how to use the internal 8051 timers to generate time delays.

Content

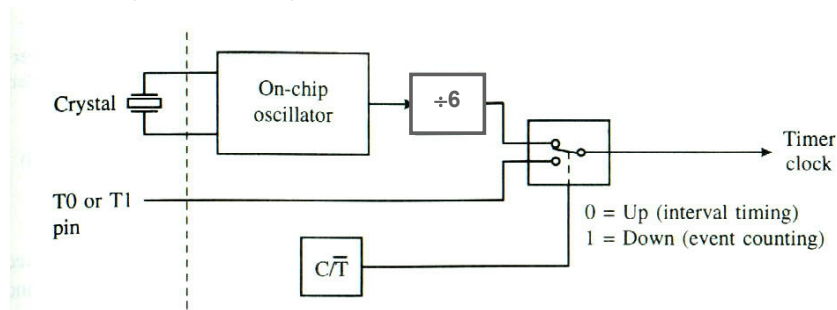
<i>Uses of Timers & Counters</i>	<i>2</i>
<i>8051 Timers</i>	<i>2</i>
<i>Timer Registers.....</i>	<i>2</i>
<i>TCON.....</i>	<i>2</i>
<i>TMOD</i>	<i>2</i>
<i>Timer Modes</i>	<i>4</i>
<i>Mode 0: 13-Bit Timer.....</i>	<i>4</i>
<i>Mode 1: 16-bit</i>	<i>4</i>
<i>Mode 2: 8-bit Auto Reload</i>	<i>4</i>
<i>Mode 3- Split Timer.....</i>	<i>5</i>
<i>Timer Delay and Timer Reload Value</i>	<i>5</i>
<i>Example 1</i>	<i>6</i>
<i>Example 2</i>	<i>6</i>
<i>Example 3</i>	<i>7</i>
<i>Example 4</i>	<i>7</i>
<i>Alternative Technique for Timers Loading</i>	<i>8</i>
<i>Example 5</i>	<i>8</i>
<i>Example 6</i>	<i>8</i>

Uses of Timers & Counters

- Interval Timing
 - Periodic event timing
 - Time base for measurements
- Event Counting
- Baud Rate Generation

8051 Timers

- 2 timers (Timer 0 and Timer 1)
- 16-bit timers (65,535) max
- Flag is set when the timer overflows
- Timers can be based on internal clock (OSC/6) or from external source (counter mode).

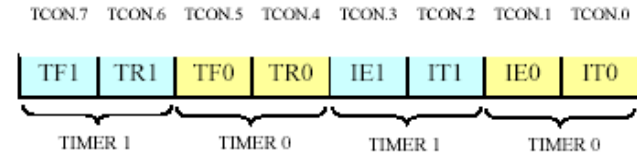


Internal operation of the 8051 timer (same for both timers)

Timer Registers

- TCON Timer Control
- TMOD Timer Mode
- TH0/TL0 Timer 0 16 bit register (byte addressable only)
- TH1/TL1 Timer 1 16 bit register (byte addressable only)

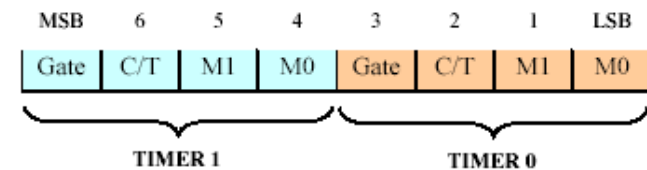
TCON



TCON SFR and its individual bits

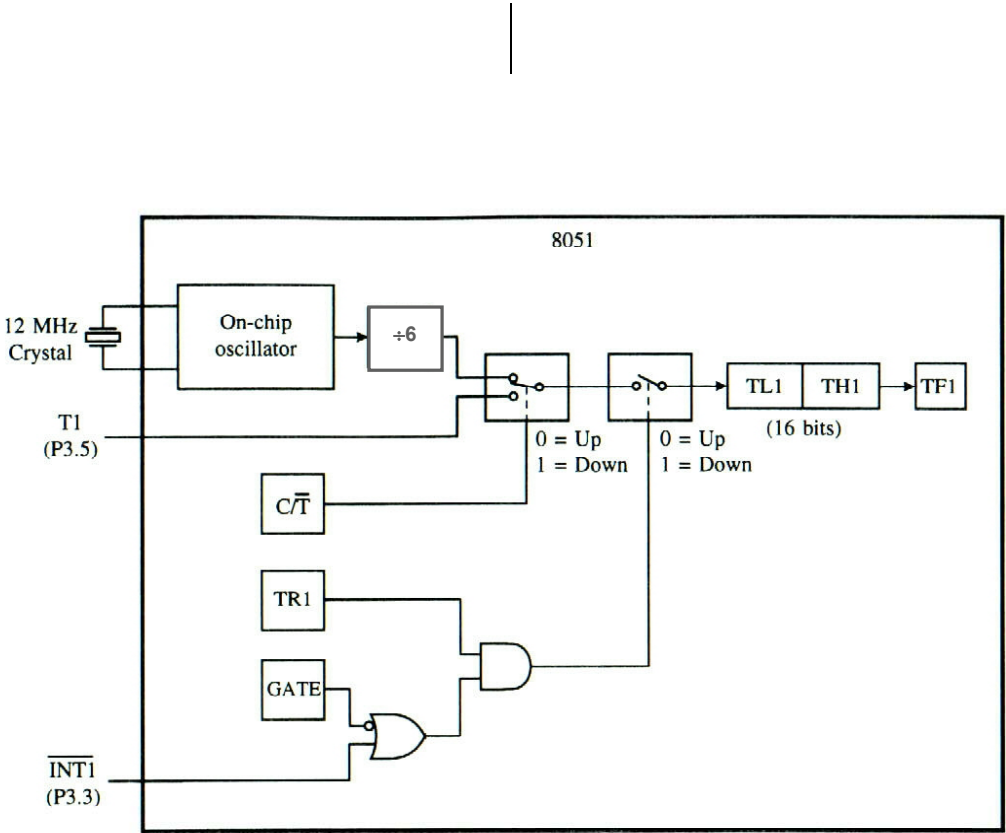
- IT0/IT1: Used for timer Interrupts
- IE0/IE1: Used for external Interrupts
- TR0/TR1: Timer 0/1 run control flag
 - 1 = Run
- TF0/TF1: Timer 0/1 overflow flag
 - 1 = Overflow

TMOD



TMOD SFR and its individual bits

- M0/M1: sets the Mode of the respective timer
- C/T: External Counter/Internal Timer select
 - 1 = Counter, 0 = Timer
- Gate: When set (1), timer runs only when respective INT input is high.



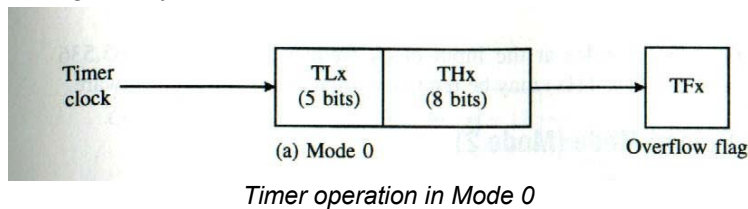
Internal architecture of the 8051 timer. The diagram shows timer 1.
 Timer 0 has an identical architecture

Timer Modes

- 0: 13 bit timer
- 1: 16-bit timer
- 2: 8-Bit auto reload
- 3: Split timer mode

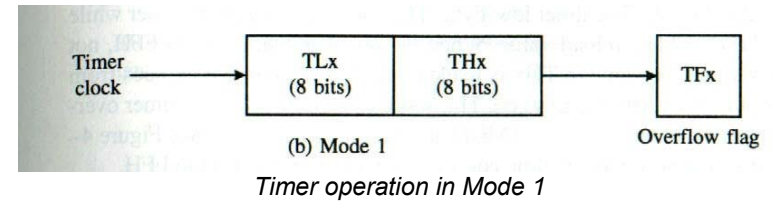
Mode 0: 13-Bit Timer

- Lower byte (TL0/TL1) + 5 bits of upper bytes (TH0/TH1).
- Backward compatible to the 8048
- Not generally used



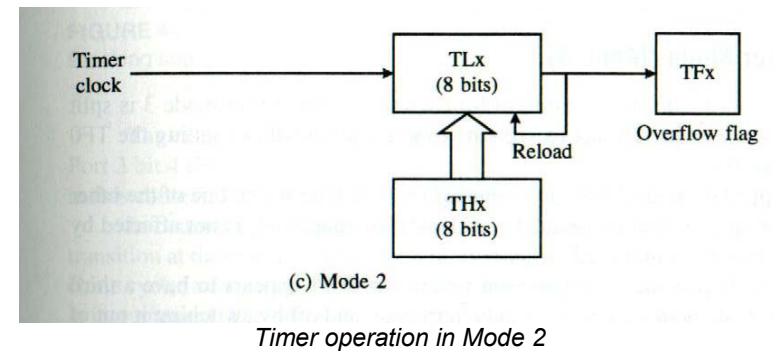
Mode 1: 16-bit

- All 16 bits of the timer (TH0/TL0, TH1, TL1) are used.
- Maximum count is 65,536
- At 12Mhz, maximum interval is 65536 microseconds or 65.536 milliseconds
- TF0 must be reset after each overflow
- THx/TLx must be manually reloaded after each overflow.



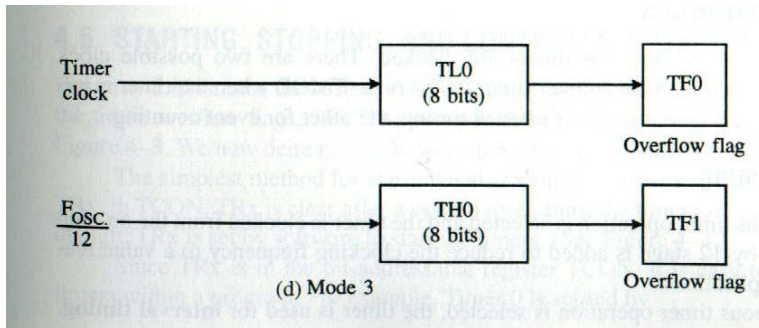
Mode 2: 8-bit Auto Reload

- Only the lower byte (TLx) is used for counting.
- Upper byte (THx) holds the value to reload into TLx after an overflow.
- TFx must be manually cleared.
- Maximum count is 256
- Maximum interval is 256 Microseconds or .256 milliseconds



Mode 3- Split Timer

- Splits Timer 0 into two 8-bit timers
- TL0 sets TF0
- TH0 sets TF1
- Timer 1 is available for other 3 modes, but the TF1 is not available.



Timer operation in Mode 3

Timer Delay and Timer Reload Value

$$\text{Timer Delay} = \text{Delay Value} \times \text{Timer Clock Cycle Duration}$$

Delay Value = how many counts before register(s) roll over

$$\text{Timer Clock Cycle Duration} = 6 / \text{oscillator frequency}$$

$$\text{Delay Value} = \text{Maximum Register Count} - \text{Timer Reload Value}$$

$$\text{Maximum Register Count} = 65535$$

$$\text{Timer Reload Value} = ?$$

Example 1

Calculation of Timer 0 reload value needed to achieve timer delay of 20 ms. Oscillator frequency is 11.0592 MHz.

$$\begin{aligned} \text{Delay Value} &= \text{Timer Delay} / \text{Timer Clock Cycle Duration} \\ &= \frac{20 \times 10^{-3}}{6} \\ &= \frac{20 \times 10^{-3}}{11.0592 \times 10^6} \\ &= 36864 \text{ (must be rounded to the nearest integer)} \end{aligned}$$

$$\begin{aligned} \text{Timer Reload Value} &= \text{Maximum Register Count} - \text{Delay Value} \\ &= 65535 - 36864 \\ &= 28671 \\ &= 0x6FFF \end{aligned}$$

so Timer 0 is loaded with:

$$\begin{aligned} \text{TH0} &= 0x6F; \\ \text{TL0} &= 0xFF; \end{aligned}$$

Example 2

Function to generate 100 μ s delay using timer 0.

Procedure is:

- Initialise TMOD register
- Initialise TL0 and TH0
- Start the Timer
- Monitor TF0 until it is set

```
Delay:      MOV    TMOD,#01H    ; initialise TMOD
            MOV    TL0,#47H    ; initialise TL0
            MOV    TH0,#FFH    ; initialise TH0
            SETB   TR0         ; start timer

Wait:      JNB    TF0,Wait     ; wait for TF0
            CLR    TR0         ; stop timer
            CLR    TF0        ; clear TF0
            RET
```

$$\text{Delay Value} = \frac{100 \times 10^{-3}}{6} = 184$$

$$\text{Timer Reload Value} = 65535 - 184 = 65351 = 0xFF47$$

so Timer 0 is loaded with:

$$\text{TH0} = 0x6F; \quad \text{TL0} = 0xFF;$$

Example 3

C version of the function from Example 2.

```
void Delay(void)
{
    TMOD = 0x01;
    TL0 = 0x47;
    TH0 = 0xFF;
    TR0 = 1;
    while(!TF0)
    TR0 = 0;
    TF0 = 0;
}
```

Example 4

Program to toggle pin 7 on Port 1 with a time delay of 20 ms.

```
#include <reg66x.h>

#define off 0
#define on 1

sbit pin7 = P1^7; // label pin7 is port 1 pin 7
```

```
main()
{
    TMOD = 0x01;
    // timer 0 mode 1,
    // TH0TL0 = 16 bit register

    while(1)
    // keep repeating the following section
    {
        pin7 = on;
        // pin 7 to 5 volts, i.e. logic 1

        // use timer 0 to generate delay
        TH0 = 0x6F; // hex 6F into TH0
        TL0 = 0xFF; // hex FF into TL0
        TR0 = on; // start timer
        while(!TF0);
        // wait here until TF0 = 1
        TR0 = off; // stop timer
        TF0 = off; // clear overflow flag

        pin7 = off;
        // pin 7 to 0 volts, i.e. logic 0

        // repeat timer delay
        TH0 = 0x6F; // hex 6F into TH0
        TL0 = 0xFF; // hex FF into TL0
        TR0 = on; // start timer
        while(!TF0);
        // wait here until TF0 = 1
        TR0 = off; // stop timer
        TF0 = off; // clear overflow flag
    }
}
```

Alternative Technique for Timers Loading

Example 5

Load the timer 0 in order to produce 1 kHz square wave (i.e. cycle time of 1000 μ s and delay time 500 μ s). Oscillator frequency is 11.0592 MHz.

$$\text{Delay Value} = \frac{500 \times 10^{-6}}{\frac{6}{11.0592 \times 10^6}} = 922$$

Timer Reload Value = 65535 – 922 = 64614 = 0xFC66

so Timer 0 is loaded with: TH0 = 0xFC; TL0 = 0x66

Alternatively if we use: TH0 = ~(922/255);

result of integer division 922/255 = 3 will be byte complemented to 0xFC and stored in TH0

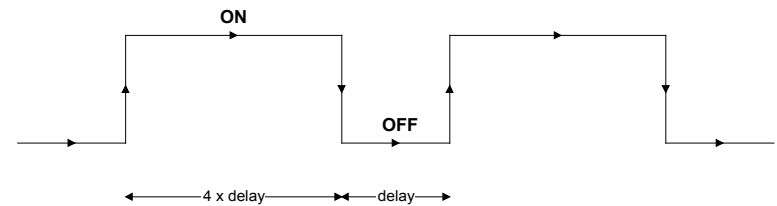
Second line to fill up lower timer 0 register: TL0 = -(922%255)

will negate remainder of division 922/255 and store the result in TL0

i.e. $922\%255 = 154$
 $-(922\%255) = 256-154 = 102 = 0x66$

Example 6

C program to generate 1 kHz square wave from figure below. Square wave should be generated on pin 7 of port 1. Functions are used to generate two delays needed in the program. (delay = 200 μ s)



```
// header file containing SFR addresses
#include<reg66x.h>

// to make program more readable:

// define ON and OFF states
#define on 1
#define off 0

// give a name to output pin
sbit pwm = P1^7;

// long and short delay functions
void delay_on();
void delay_off();
```



```
main()
{
    TMOD = 0x01;
    // initialise TMOD for Timer 0 in mode 1

    while(1)    // repeat this
    {
        pwm = on;    // output pin high
        delay_on(); // 800 us delay
        pwm = off;   // output pin low
        delay_off(); // 200 us delay
    }
}

// 800 us delay function
void delay_on()
{
    // loading Timer 0 for longer delay
    TH0 = ~(1475/256);
    TL0 = -(1475%256);
    TR0 = on;    // turn the Timer 0 ON
    while(!TF0); // wait for timer overflow
    TR0 = off;   // switch the Timer 0 off
    TF0 = off;   // clear the overflow flag
}

// 200 us delay function
void delay_off()
{
    // loading Timer 0 for shorter delay
    TH0 = ~(369/256);
    TL0 = -(369%256);
    TR0 = on;
    while(!TF0);
    TR0 = off;
    TF0 = off;
}
}
```