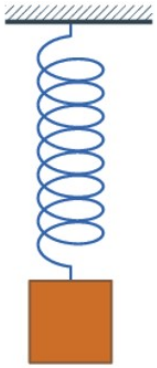
Embedded Systems
3.2 - Oscillator and Timer (Clicker 2)

Enrico Simetti

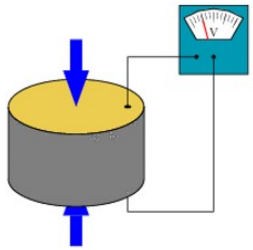
ISME - Interuniversity Research Center on Integrated Systems for Marine Enviroment
DIBRIS - Department of Computer Science, Bioengineering, Robotics and System Engineering
University of Genova, Italy
enrico.simetti@unige.it

The oscillator I

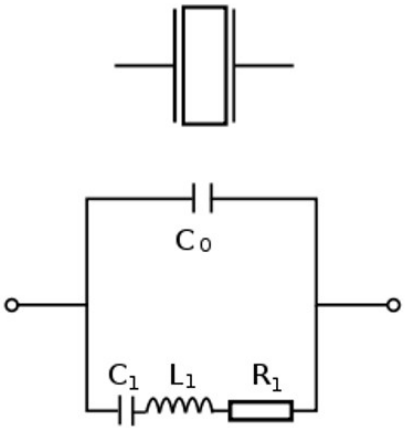


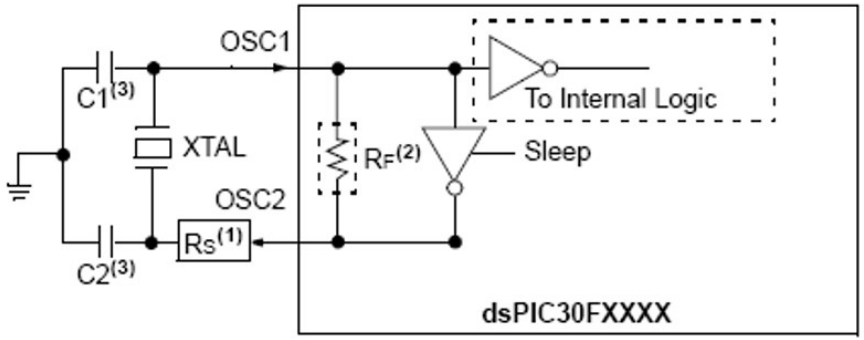
Mechanical oscillator

Crystal oscillator

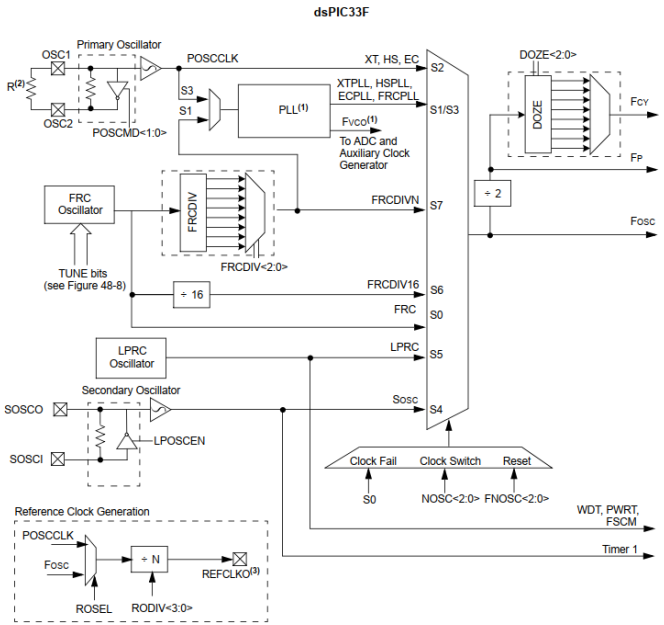


Piezoelectric property





The oscillator II



- 4 clock sources available:
  - external primary oscillator
  - external secondary oscillator
  - internal fast RC oscillator
  - internal low-power RC oscillator
- An on-chip Phase-Locked Loop (PLL) (for increasing clock rate)
- Clock switching between clock sources
- Programmable clock postscaler (for system power savings)
- A Fail-Safe Clock Monitor (FSCM) (detects clock failures)
- Special Function Registers and Configuration bits used for clock-control (FOSC, OSCCON, OSCTUN)

The oscillator III

Table 48-1: Configuration Bit Values for Clock Selection

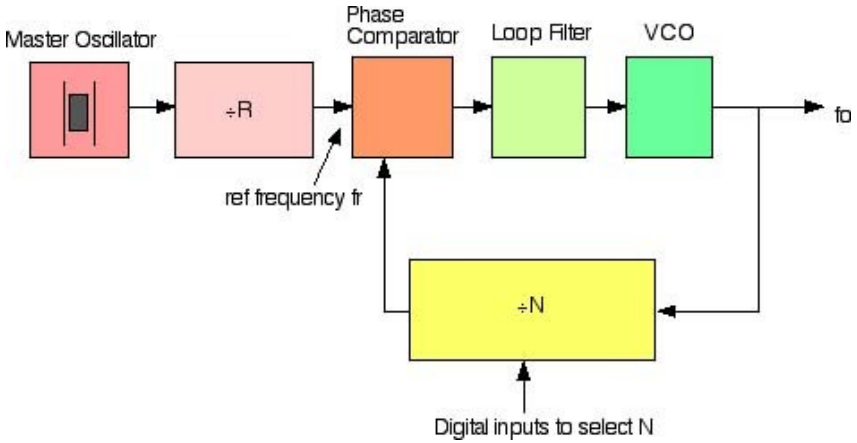
Oscillator Source	Oscillator Mode	FNOSC Value	POSCMD Value	See Note
S0	Fast RC Oscillator (FRC)	000	xx	1
S1	Fast RC Oscillator with PLL (FRCPLL)	001	xx	1
S2	Primary Oscillator (EC)	010	00	1
S2	Primary Oscillator (XT)	010	01	—
S2	Primary Oscillator (HS)	010	10	—
S3	Primary Oscillator with PLL (ECPLL)	011	00	1
S3	Primary Oscillator with PLL (XTPLL)	011	01	—
S3	Primary Oscillator with PLL (HSPLL)	011	10	—
S4	Secondary Oscillator (SOSC)	100	xx	1
S5	Low-Power RC Oscillator	101	xx	1
S6	Fast RC Oscillator with Divide-by-16 divider (FRCDIV16)	110	xx	1
S7	Fast RC Oscillator with Divide-by-N divider (FRCDIVN)	111	xx	1, 2

Note 1: The OSC2 pin function is determined by the OSCIOFNC Configuration bit.

2: The default oscillator mode for an unprogrammed (erased) device.

MikroE Clicker2 board oscillator (XT): 8 MHz

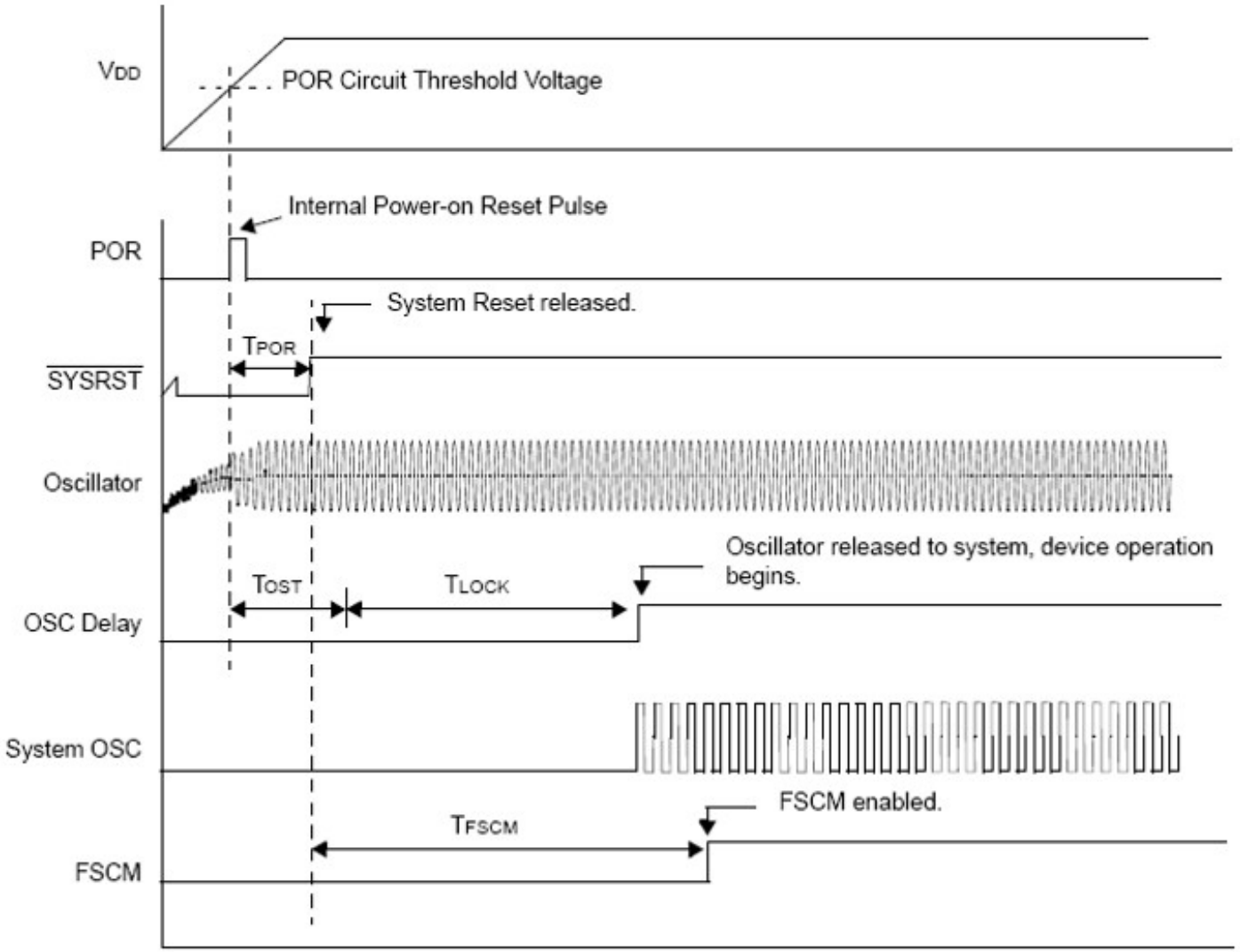
Phase-locked loop



A phase detector compares two input signals and produces an error signal which is proportional to their phase difference. The error signal is then low-pass filtered and used to drive a VCO which creates an output phase. The output is fed through an optional divider back to the input of the system, producing a negative feedback loop. If the output phase drifts, the error signal will increase, driving the VCO phase in the opposite direction so as to reduce the error. Thus the output phase is locked to the phase at the other input. This input is called the reference.

VCO = voltage controlled oscillator

Behaviour at power on



Primary Oscillator Sources

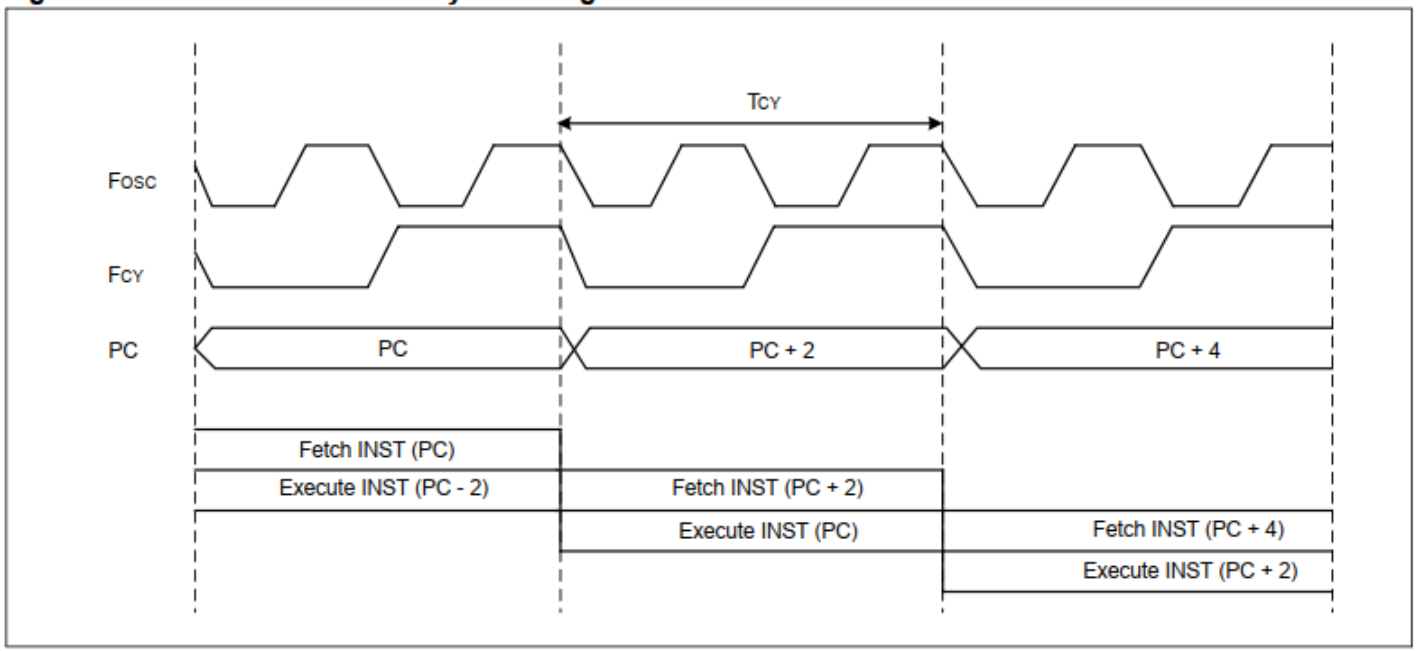
Table 48-2: Primary Oscillator Clock Source Options

FNOSC Value	POSCMD	Primary Oscillator Source/Mode
011	00	Primary Oscillator with PLL: External Clock Mode (ECPLL)
011	01	Primary Oscillator with PLL: Crystal Oscillator with PLL Mode (XTPLL)
011	10	Primary Oscillator with PLL: High-Speed Oscillator with PLL Mode (HSPLL)
010	00	Primary Oscillator: External Clock Mode (EC)
010	01	Primary Oscillator: Crystal Oscillator Mode (XT)
010	10	Primary Oscillator: High-Speed Mode (HS)

We'll be using the 8Mhz external crystal oscillator with PLL

Fcy and Fosc

Figure 48-2: Clock/Instruction Cycle Timing

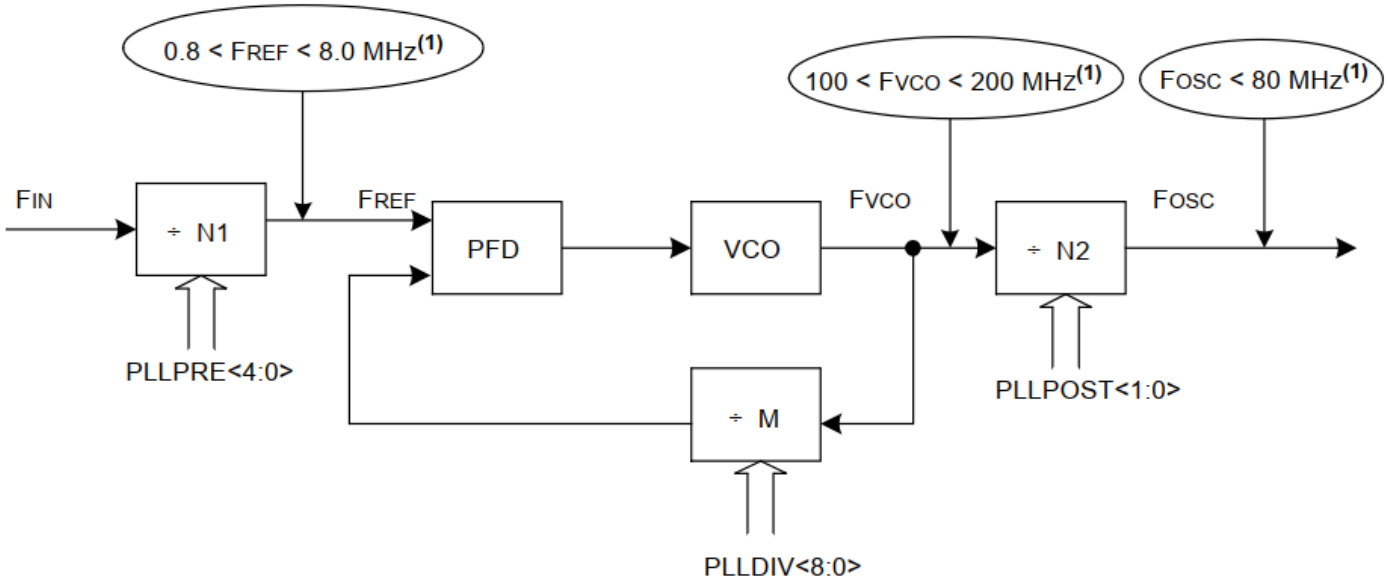


This relation is extremely important: we often need to know the value of Fcy to setup the peripherals!

Note that differently from dsPIC30, for the dsPIC33 family Fcy is half the Fosc!

PLL Configuration

Figure 48-9: dsPIC33F PLL Block Diagram



Note 1: This frequency range must be satisfied at all times.

Equation 48-3: Fosc Calculation

$$FOSC = FIN \times \left(\frac{M}{N1 \times N2}\right) = FIN \times \left(\frac{(PLLDIV + 2)}{(PLLPRE + 2) \times 2(PLLPOST + 1)}\right)$$

where:

$N1 = PLLPRE + 2$

$N2 = 2 \times (PLLPOST + 1)$

$M = PLLDIV + 2$

***FIN** is the frequency of the source before entering the PLL (8 MHz for the XT source)*

Oscillator Configuration

Clock config

copy-paste at the beginning of the main

Example of oscillator configuration:

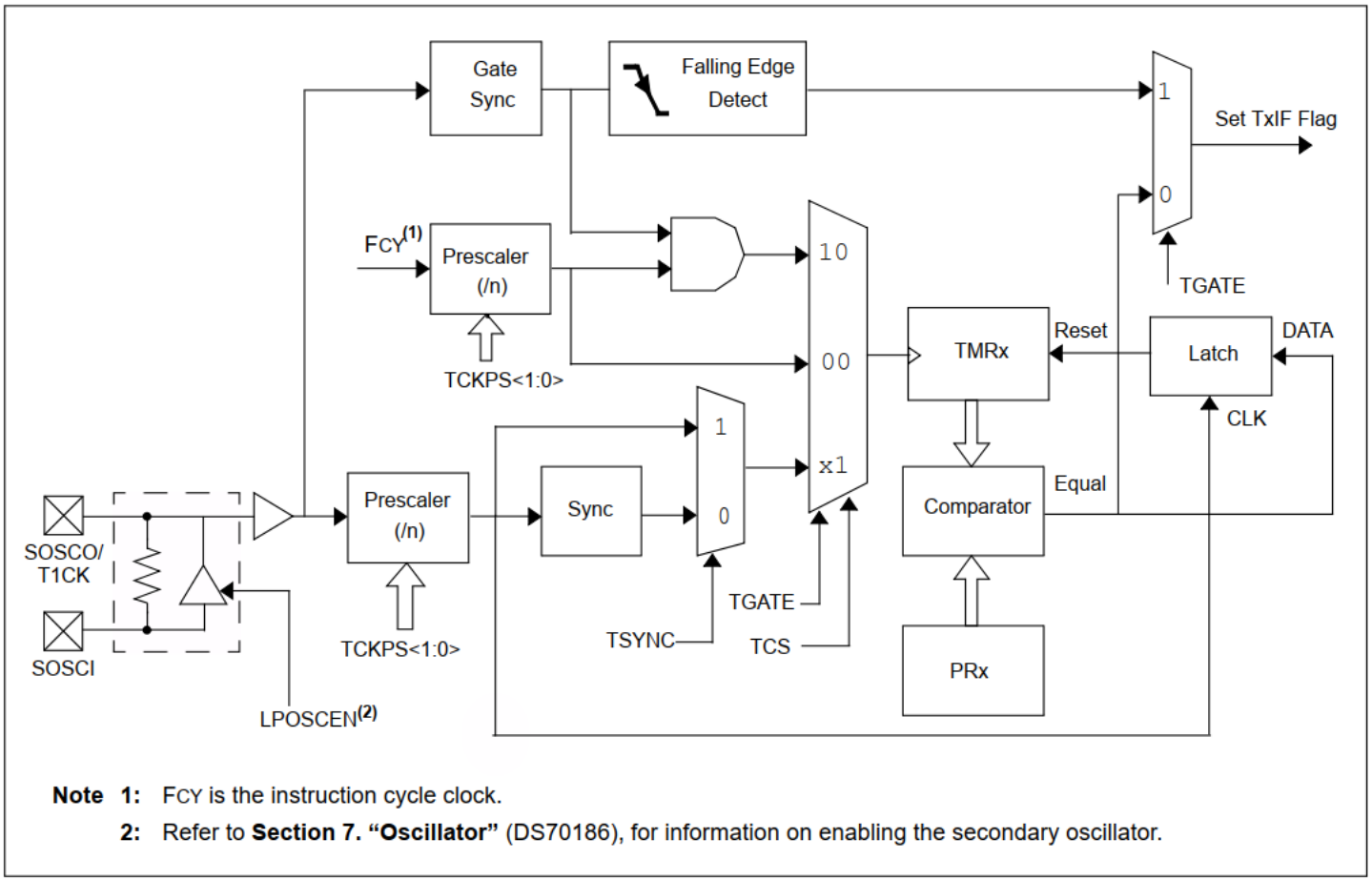
$F_{osc} = 8MHz \cdot (\frac{8}{(2 \cdot 2)}) = 16MHz, F_{cy} = 8MHz$

```
PLLFBD = 6; // M = 8
CLKDIVbits.PLLPOST = 0x00; // N1 = 2
CLKDIVbits.PLLPRE = 0x00; // N2 = 2
RCONbits.SWDTEN = 0; // Disable Watch Dog Timer
while (OSCCONbits.LOCK != 1) {
  }; // Wait for PLL to lock
```

Timers I

- 5 16-bit timers available:
  - T1,T2,T3,T4,T5
- 3 kind of timers:
  - Type A (T1) – can be operated also by the LPRC – can work in asynchronous counter mode
  - Type B (T2 and T4) – can be concatenated with C-type leading to a 32-bit timer
  - Type C (T3 and T5) – can trigger an A/D conversion
- 4 functions performed:
  - synchronous timer – increments at every internal clock cycle (A,B,C)
  - synchronous counter – increments at every external clock cycle (A,B,C)
  - asynchronous counter – increments at every external raising signal (A)
  - gated timer – counts the number of clock cycles an external signal is "high" (A,B,C)
- Special Function Registers used for configuration and access
  - *TxCON, PRx, TMRx*

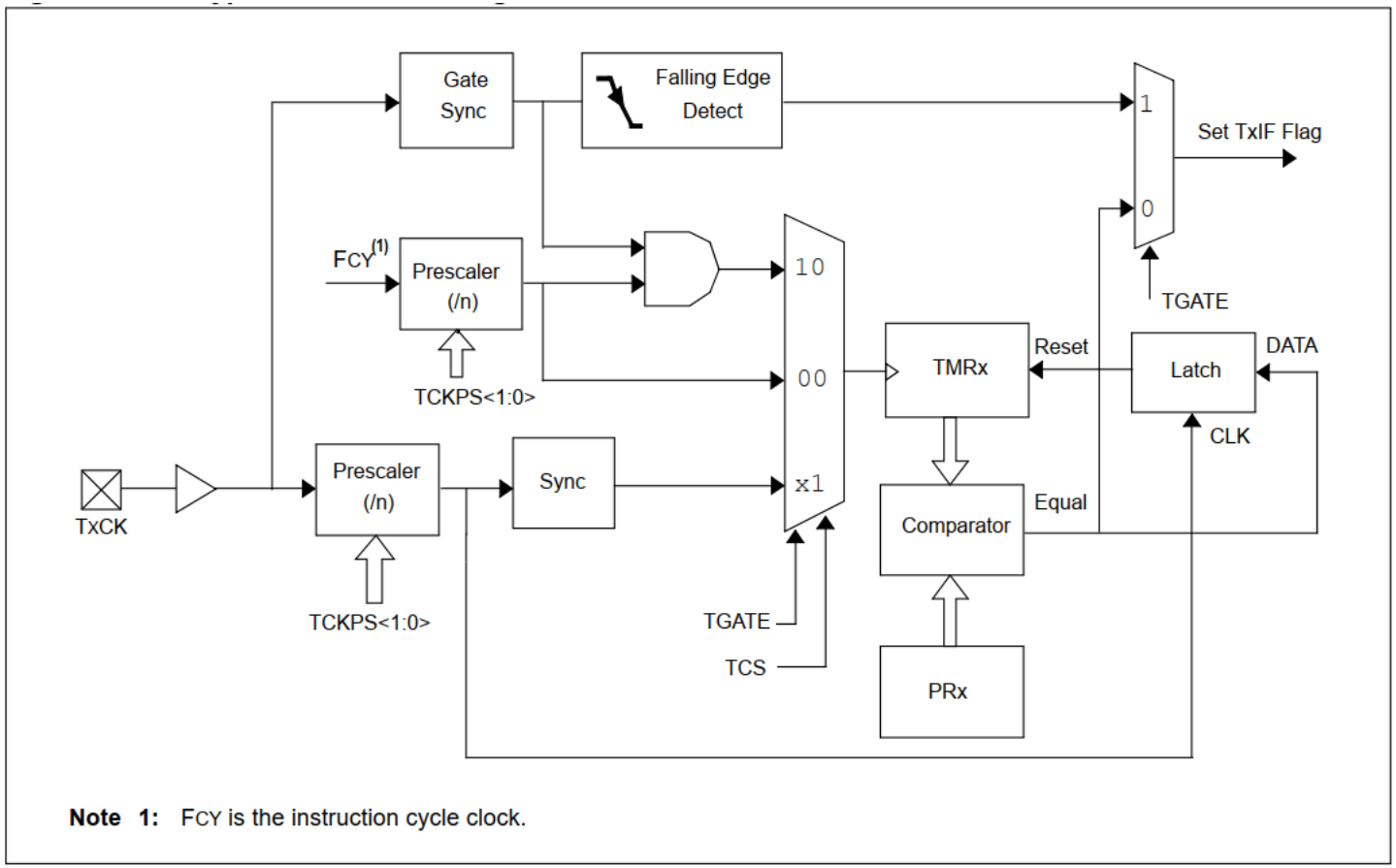
Timer A:



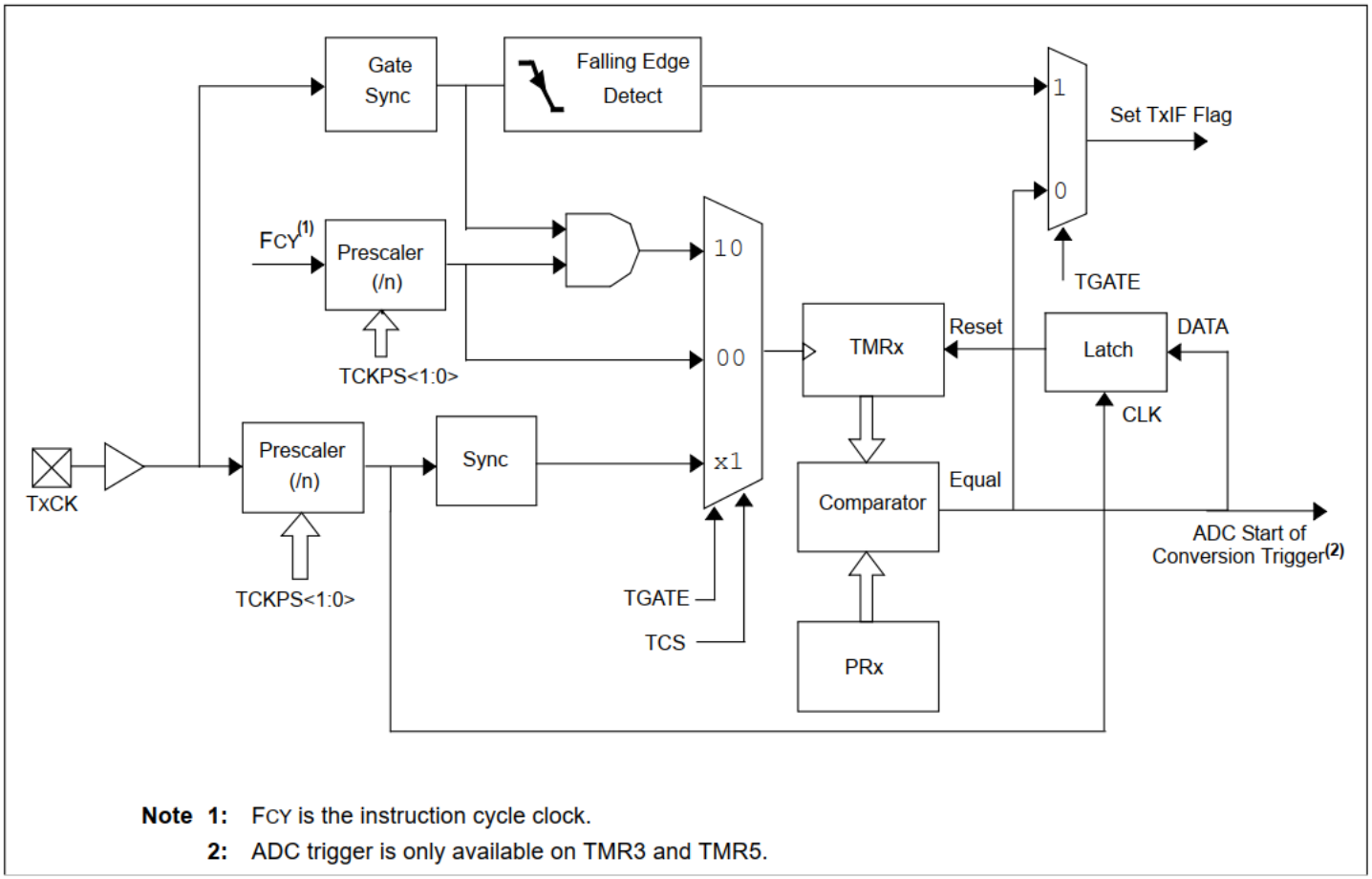
Note 1: FCY is the instruction cycle clock.

2: Refer to **Section 7. "Oscillator"** (DS70186), for information on enabling the secondary oscillator.

Timer B:

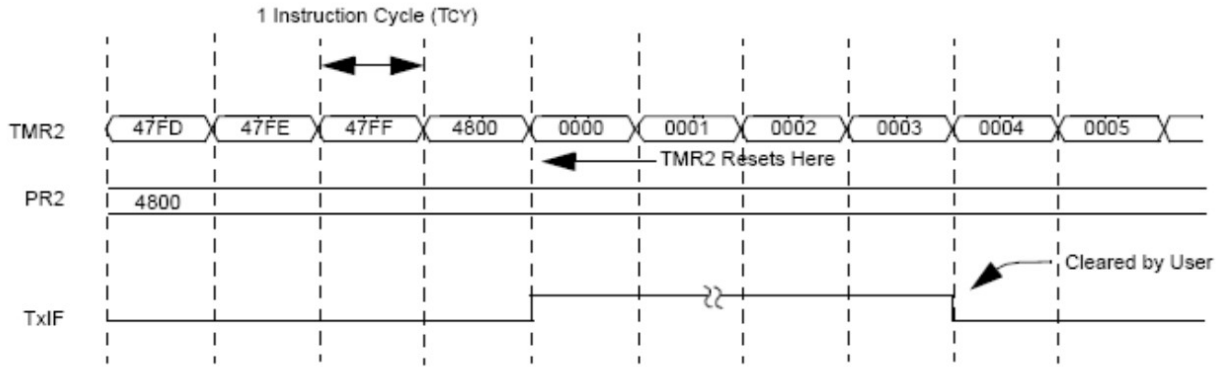

Gate Sync
Falling Edge Detect
1
0
Set TxIF Flag
TGATE
FCY(1)
Prescaler (/n)
TCKPS<1:0>
10
00
x1
TMRx
Reset
Latch
DATA
CLK
TxCK
Prescaler (/n)
TCKPS<1:0>
Sync
TGATE
TCS
Comparator
Equal
PRx
Note 1: FCY is the instruction cycle clock.

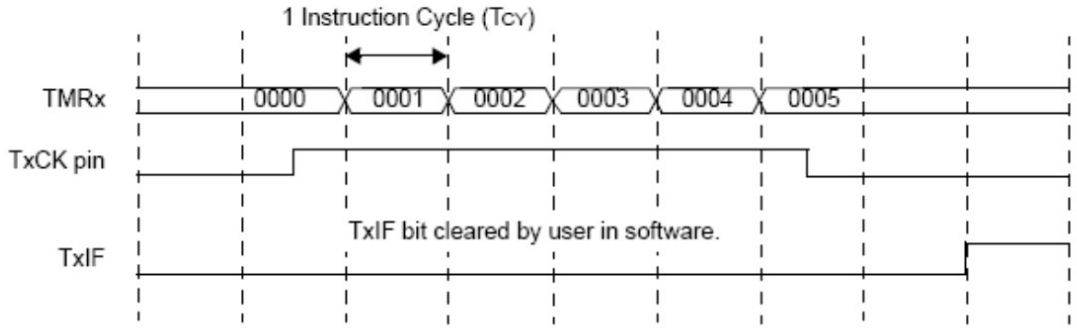
Timer B+C (32 bit timer):



Note 1: FCY is the instruction cycle clock.
2: ADC trigger is only available on TMR3 and TMR5.

Counter and gated modes





- counter: TMRx is incremented every internal clock cycle
- gated: counts the number of clock cycles an external signal is "high"

Timer related registers I

Register 12-1: TxCON: Type A Time Base Register

Upper Byte:							
R/W-0	U-0	R/W-0	U-0	U-0	U-0	U-0	U-0
TON	—	TSIDL	—	—	—	—	—
bit 15							bit 8

Lower Byte:							
U-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	U-0
—	TGATE	TCKPS<1:0>		—	TSYNC	TCS	—
bit 7							bit 0

TxCON: Timer X Control register.

- *TON*: starts the timer set to one AFTER setting up the timer
- *TCKPS*: sets the prescaler extremely important!!
- *TCS*: clock source (0 = internal clock = Fosc/2)
- *TGATE*: if the gated mode should be enabled

Timer related registers II

- *PRx*: set this to the value the timer must count to
- *TMRx*: its the current value of the timer. Set to zero to reset the timer

IFSybits.TxIF: it's the interrupt flag, which signals an event related to the corresponding timer. Must be put to zero manually to be notified of a new event

The interrupt flags are the following ones:

- IFS0bits.T1IF
- IFS0bits.T2IF
- IFS0bits.T3IF
- IFS1bits.T4IF
- IFS1bits.T5IF

Example timer programming

Example: timer expiring every 100 ms, assuming FOSC = 8 MHz

```
TMR1 = 0; // reset timer counter

// Fcy = Fosc / 2 = 8000000 / 2 = 4000000 (number of clocks in one second)
// in 0.1 second there would be 400000 clocks steps
// this is too high to be put in a 16 bit register (max 65535)
// If we set a prescaler of 1:8 we have 400000/8 = 50000 clock steps, OK!
PR1 = 50000; // (8 MHz / 2) / 8 / 10 < 65535!

T1CONbits.TCKPS = 1; // prescaler 1:8
T1CONbits.TON = 1; // starts the timer!
```

Assignment: Timers I

1. Create two functions:

```
void tmr_setup_period(int timer, int ms);
void tmr_wait_period(int timer);
```

   the first function setups the timer *timer* to count for the specified amount of milliseconds. The function should support values up to one second (at least). The second function should use the timer flag to wait until it has expired.

2. Use the two above defined functions to blink D3 led at 1 Hz frequency (500ms time on, 500ms off)

3. Create a function:

```
void tmr_wait_ms(int timer, int ms);
```

   that uses the timer *timer* to wait for a given number of milliseconds, given as input. Use the function to: turn on the led for 1 second, turn off for 5 seconds, turn on for 500ms, turn off.

Assignment: Timers II

```
#define TIMER1 1
#define TIMER2 2
void tmr_setup_period(int timer, int ms);
void tmr_wait_period(int timer);

int main() {
  // initialization code
  // [...]

  tmr_setup_period(TIMER1, 500);

  while (1) {
    // code to blink LED
    tmr_wait_period(TIMER1);
  }
}
```

Assignment: Timers III

```
#define TIMER1 1
#define TIMER2 2
void tmr_wait_ms(int timer, int ms);

int main() {
  // initialization code
  // [...]

  tmr_wait_ms(TIMER2, 1000);
  // [...]
  tmr_wait_ms(TIMER2, 5000);
  // [...]
  tmr_wait_ms(TIMER2, 500);
  // [...]
}
```

Assignment: Timers IV

Test the following code. Observe the behaviour. Try changing the delay value from 250, 500, 2000.

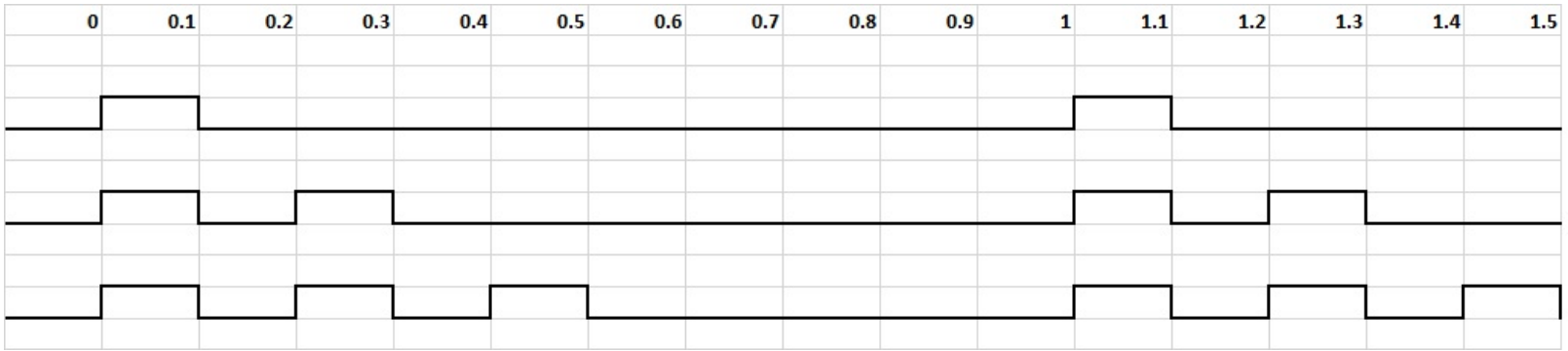
```
// [...]
int main() {
  // initialization code
  // [...]

  tmr_setup_period(TIMER1, 500);
  int delay = 250;
  while (1) {
    tmr_wait_ms(TIMER2, delay);
    // code to blink LED
    tmr_wait_period(TIMER1);
  }
}
```

Assignment: Timers and IO (advanced)

Pulsing LED

- A LED pulse is defined as 100 ms on, then off. Pulses are separated by 100 ms. The period is 1000 ms.
  - With one pulse only, the LED should stay on 100 ms and off 900 ms;
  - With two pulses, LED should stay on 100 ms, off 100 ms, on 100 ms, and finally off 700 ms.



Exercise:

- Initially, D1 is off.
- Whenever E8 is pressed, D1 should increase its pulses, up to a maximum of 3. Once 3 is reached, a further press will reset the count to 1.
- If E8 is kept pressed for at least 3 seconds, D2 should be turned off.