

Using Timers

Why Timers?

- The Timer is a hardware counter to expedite counting and delay operation that would be time consuming and inefficient if done in software.
- The Timer once set up performs its function with little to no oversight by the CPU, and only alerts the CPU once an overflow event has occurred.
- Timers in the PIC can be set up to count external signals or source with internal clock. Rates and initial count settings are all configurable

Timers with in PIC16F887

- Three total (Timer0, Timer1, Timer2) that allow for hardware to compensate for software operations
- Two accepts external inputs (Timer0 and Timer1)
- All accept internal inputs
- All generate a Flag when timer counter overflows , that is, moving FF-> 00
- All have a preset – initial timer setting
- All have a Prescaler—scale down to internal clock
- Some have a post-scaler-Timer2

Simple Block Diagram-Timer0

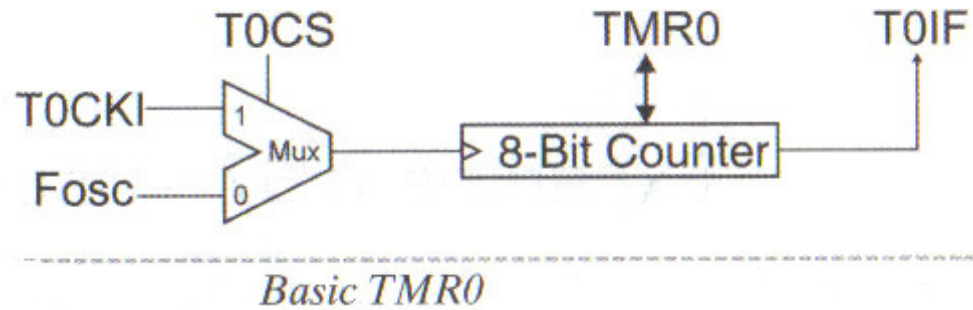
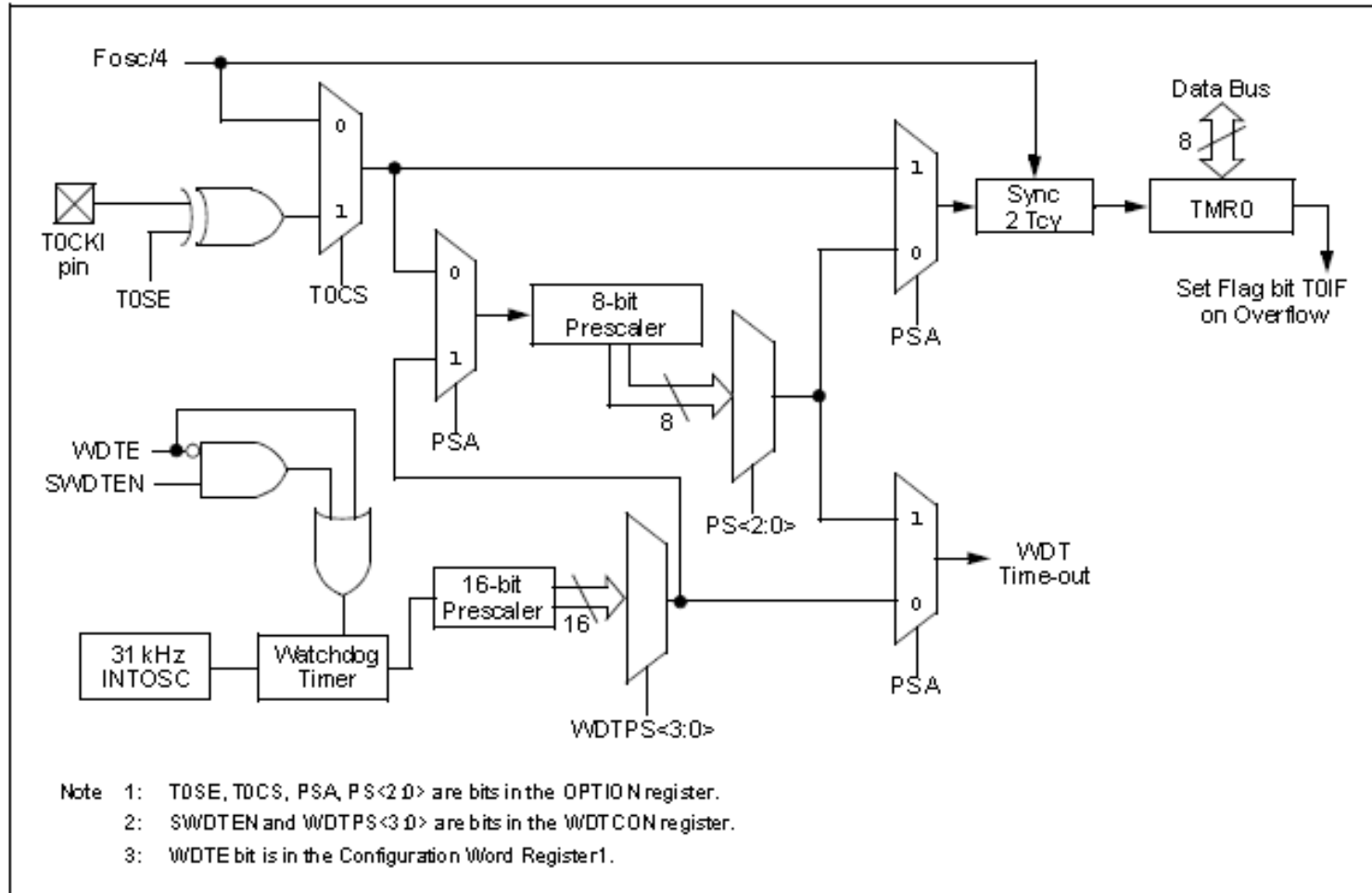


FIGURE 5-1: TIMER0/WDT PRESCALER BLOCK DIAGRAM



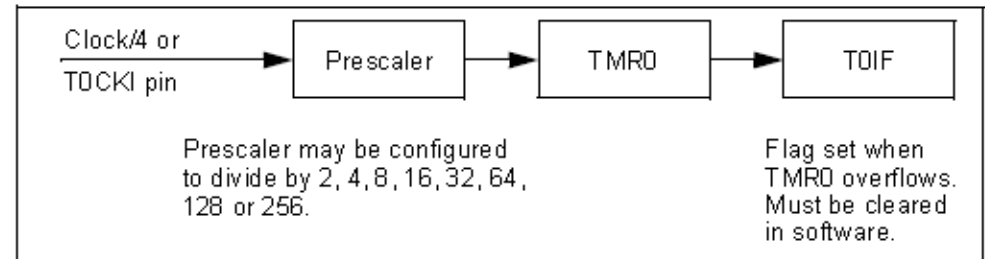
Timer0

- Timer0 is a counter implemented in the processor.
- It may be used to count processor clock cycles or external events. This course's Lesson configures it to count instruction cycles and set a flag when it rolls over.
- This frees up the processor to do meaningful work rather than just counting cycles for a delay.
- Timer0 is an 8-bit counter with an optional prescaler, which is configured to divide by 256 before reaching the Timer0 counter.

TIMER0 Description-as per data sheet

- TMR0 is a Special Function Register (SFR) and may be read or modified by the program.
- The prescaler is not a SFR and thus cannot be read or modified by the program.
- However, writing to TMR0 clears the prescaler.
- The timer may be fed either by the same clock that drives the processor or by an external event. Driven by the processor clock, it increments once for every instruction cycle.
- This is a convenient method of marking time and is better than delay loops, as it allows the processor to work on the problem rather than waste cycles in delay loops.
- The prescaler is configured through the OPTION

FIGURE 3-7: TIMER0 SIMPLIFIED



Timer0 Prescaler

- Prescaler will divide the processor clock by 256
- Timer0 Flag will be set every $65536 \mu\text{s}$ ($.0000001 \text{ second} * 256 *$), or about 15 times a second.
- The main program sits in a loop waiting for the rollover and when it does, it increments the display and then loops back.

FIGURE 3-8: PRESCALER CONFIGURATION THROUGH OPTION_REG

X	X	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

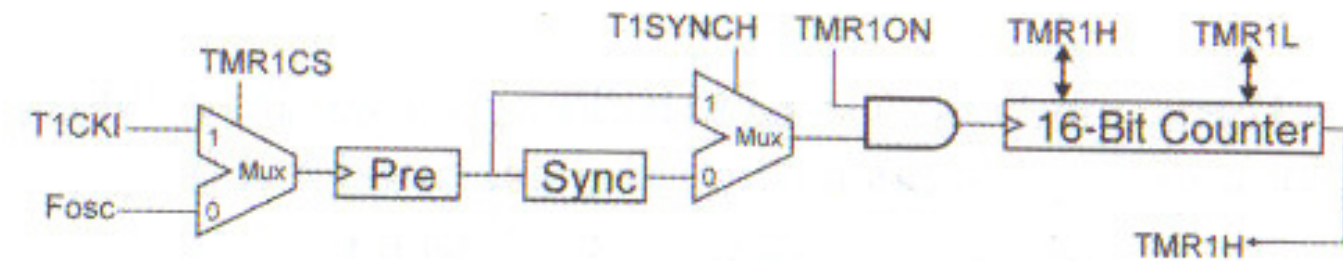
Legend:

- X: Don't cares – not Timer0 related.
- T0CS: Timer0 Clock Source 0 for Instruction Clock.
- T0SE: Timer0 Source Edge – Don't care when connected to instruction clock.
- PSA: Prescaler Assignment 0 assigns to Timer0.
- PS: Prescaler rate select '111' – full prescaler, divide by 256.

Timer0 Exercise

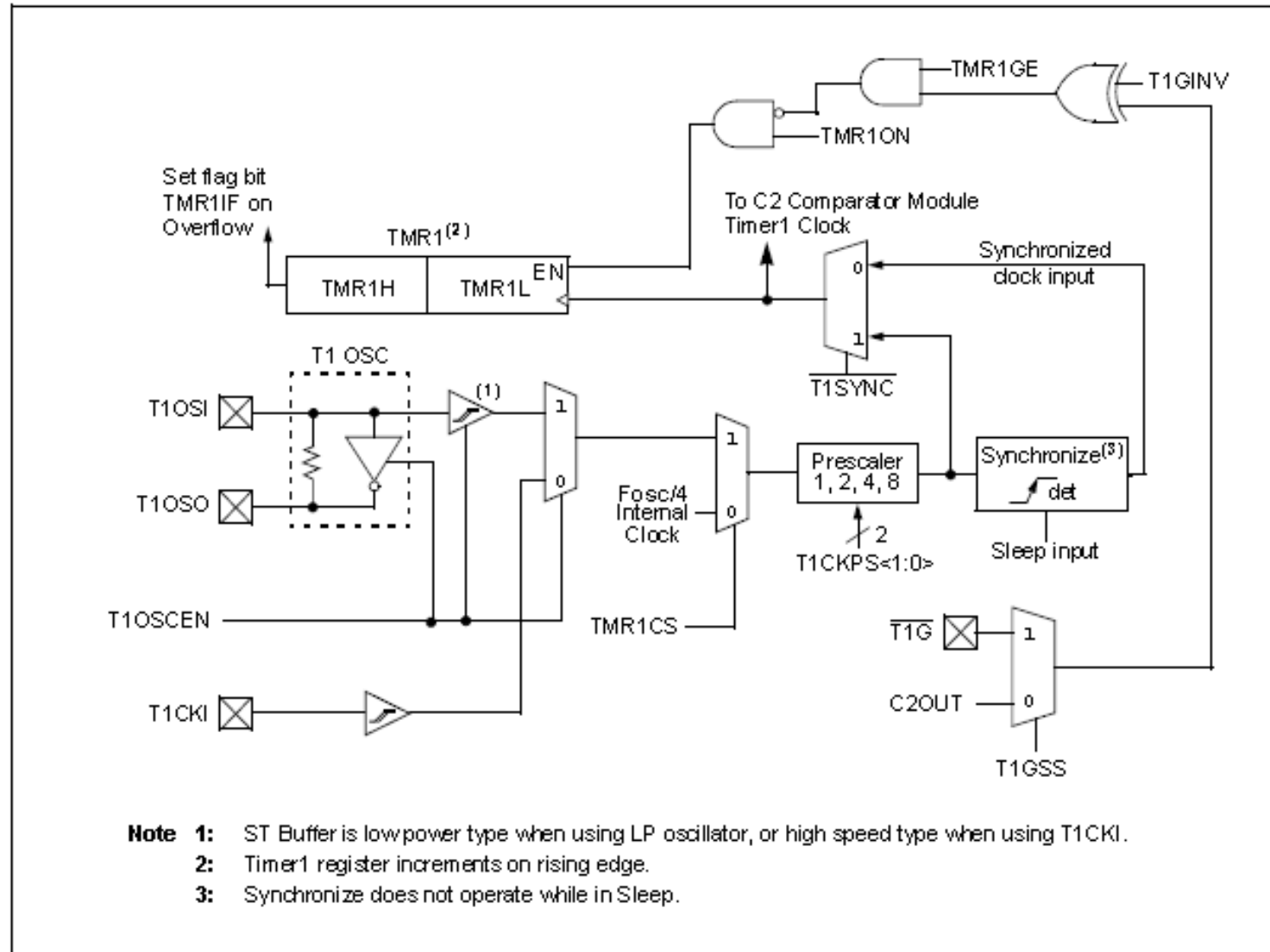
1. Navigate to: C:\EET250\CS2117 course\Lesson 9 Timers with Demo Board\TMR0
2. Open TMRpre.mcp
3. Select debugger-> PICKIT2
4. Make sure output window says PICKIT2 ready
5. This code uses Timer0 to cause delay for RD7 blink
6. Examine code to understand TIMER0 settings
 - Clock source, prescaler and count
 - What is the calculated delay?
7. Build/download/program
8. Validate blink operation
9. Select debugger -> simulator
10. Make sure that simulation is running at 4Mhz clock rate
11. Set break point at NOP(); Use stopwatch. Reset and run simulation.
12. At break clear stopwatch and then resume simulation
13. At next break the stopwatch will show measured time delay—is it the same as your calculations?

Timer1 Simple Block Diagram



Basic TMR1

FIGURE 6-1: TIMER1 BLOCK DIAGRAM

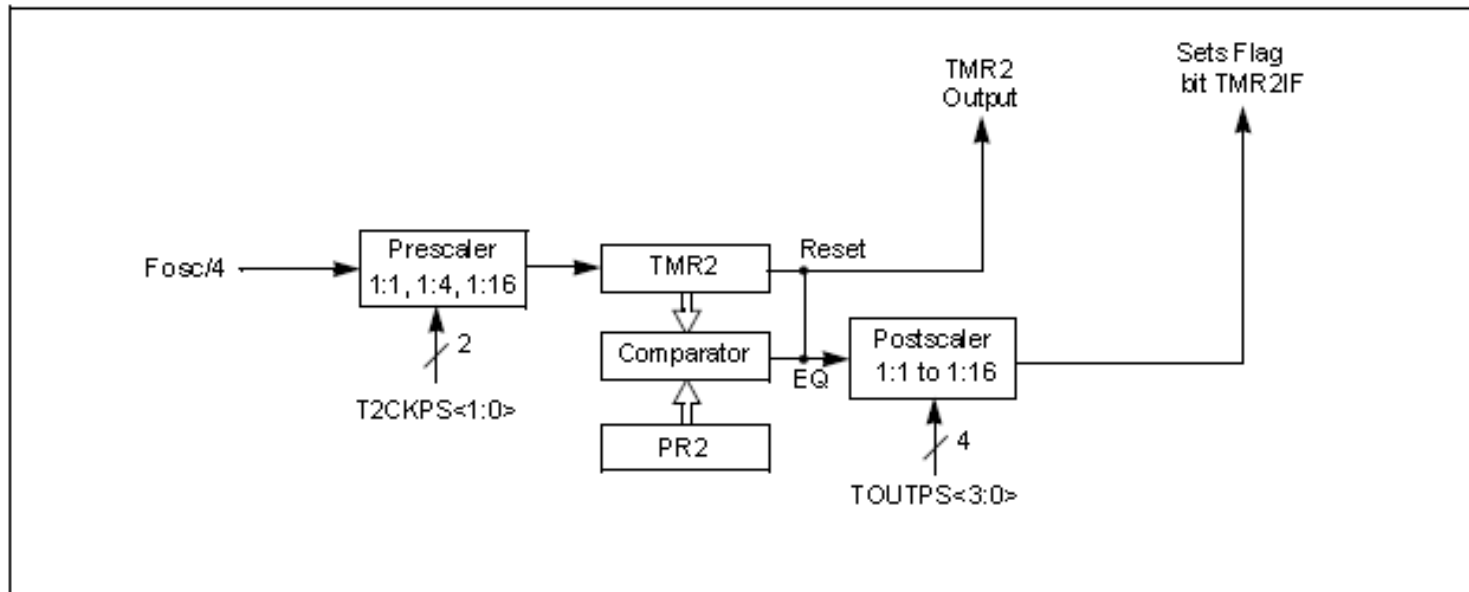


Timer1 Exercise

1. Navigate to: C:\EET250\CS2117 course\Lesson 9 Timers with Demo Board\TMR1
2. Open TMR1.mcp
3. Select debugger-> PICKIT2
4. Make sure output window says PICKIT2 ready
5. This code uses Timer1 to cause delay for RD0 blink
6. Examine code to understand TIMER1 settings
7. Clock source, prescaler and count-note Timer1 is 16 bits versus 8 bit Timer0
 - What is the calculated delay?
8. Build/download/program
9. Validate blink operation
10. Select debugger -> simulator
11. Make sure that simulation is running at 4Mhz clock rate
12. Set break point at NOP(); Use stopwatch. Reset and run simulation.
13. At break clear stopwatch and then resume simulation
14. At next break the stopwatch will show measured time delay—is it the same as your calculations?

Timer2

FIGURE 7-1: TIMER2 BLOCK DIAGRAM



Timer2

- PWM usage
- Reserved for CCP peripheral