Exercise 4 Prep

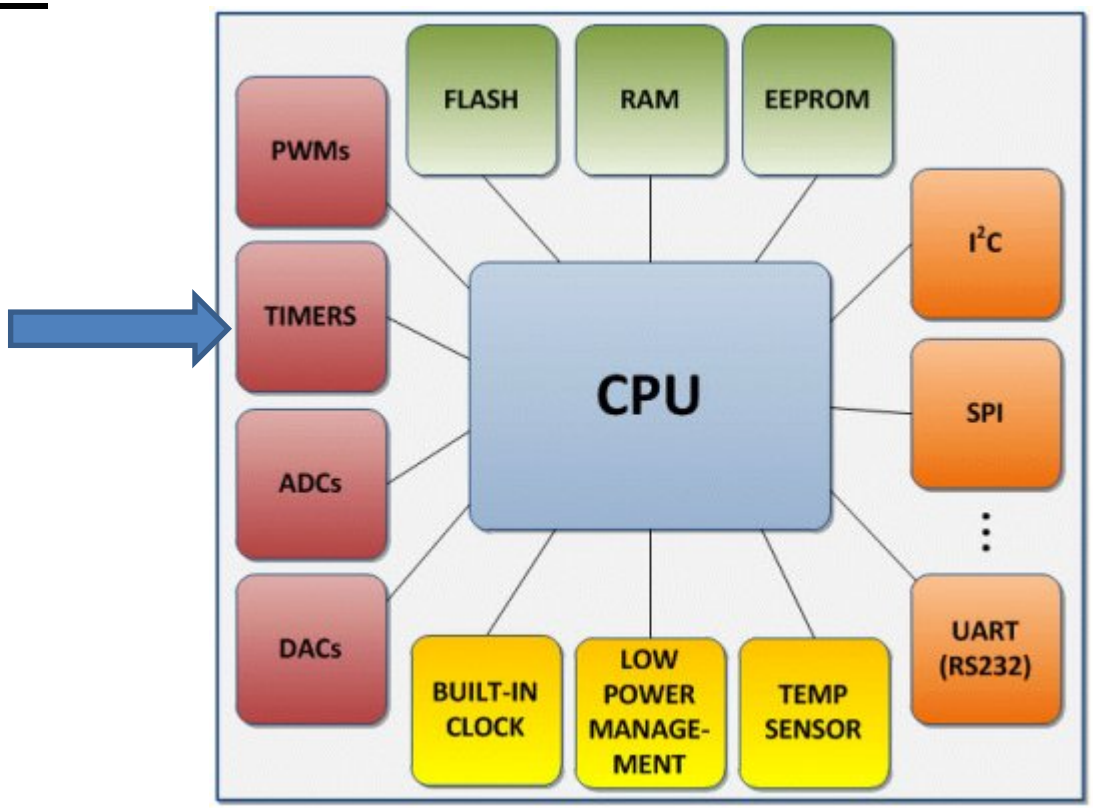
Timers and Interrupts

Exercise Overview

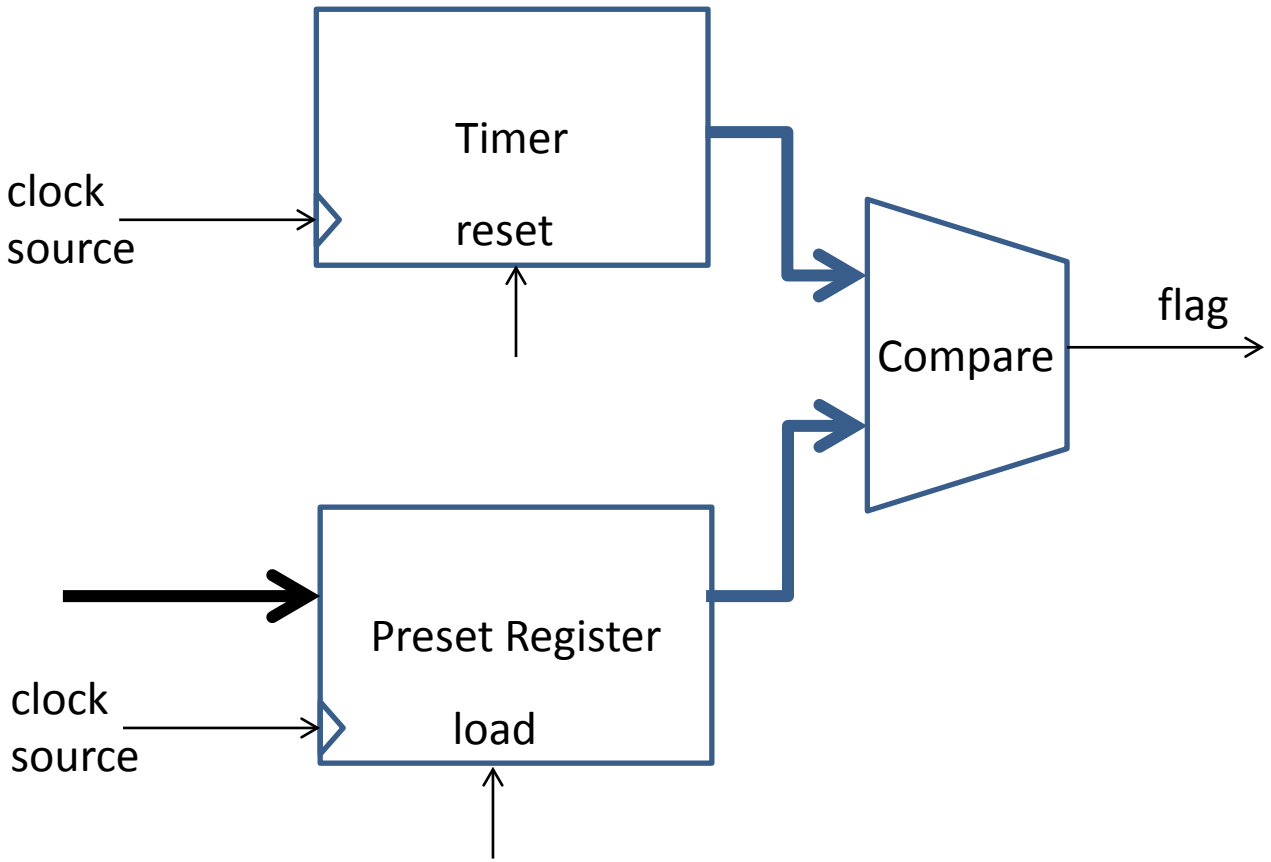
- Hardware
  - Microstick II kit – we will use LED on board.
  - Breadboard, switch, resistor.
- Software – blink the LED
  - Polling Timer Flag – use timer without interrupts to produce a delay between blinks.
  - Interrupt on Timer Flag – blink light in the interrupt service routine.
- Software – control LED with switch
  - Blink light manually using switch input using timer and interrupt service routine

Timers

- Timers are a common microcontroller peripheral.

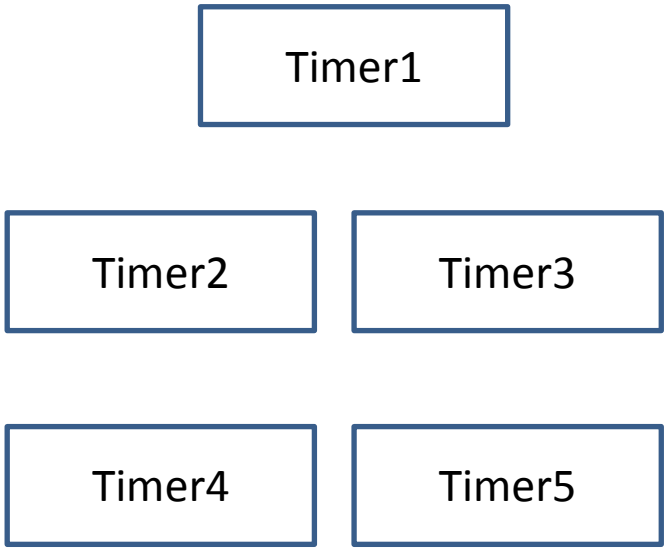


Timer Basics



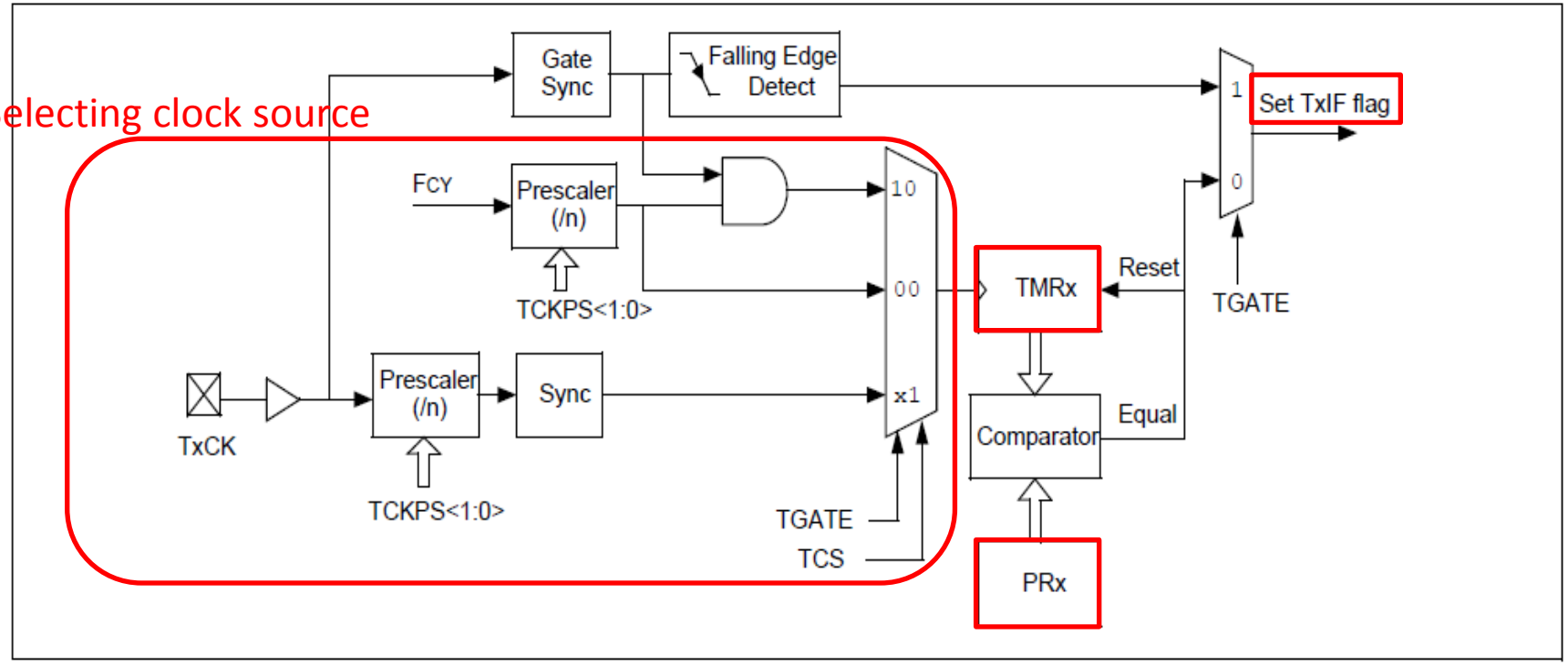
PIC24 Timers

- Five 16-bit timers
- Timer 1
  - Special features to support real-time clock generation
- Timers 2/3 and Timers 3/4 can be paired to make 32-bit timers.



Timer 2 Details (let x=2)

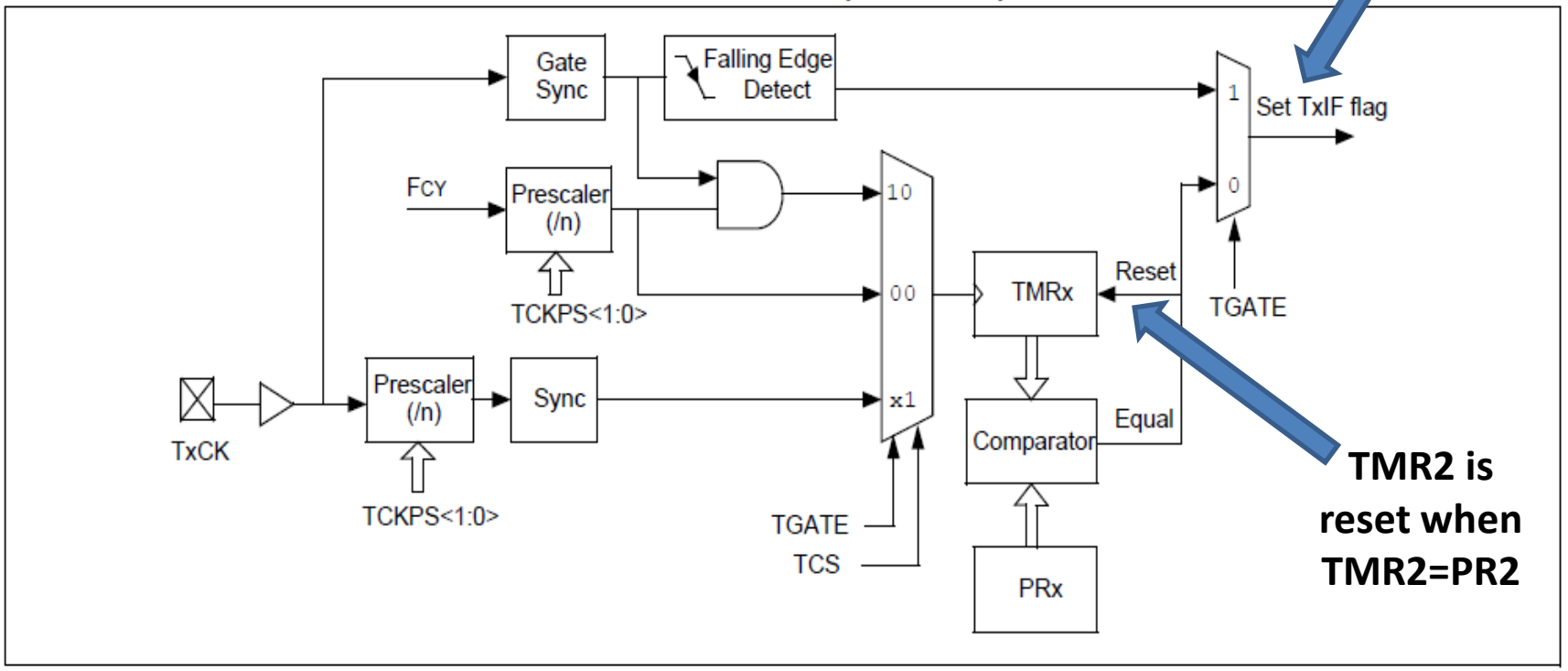
FIGURE 13-1: TYPE B TIMER BLOCK DIAGRAM (x = 2 or 4)



Notice

T2IF is set when TMR2=PR2

FIGURE 13-1: TYPE B TIMER BLOCK DIAGRAM (x = 2 or 4)



TMR2 is reset when TMR2=PR2

How to Use Timers?

Access via Special Function Registers!

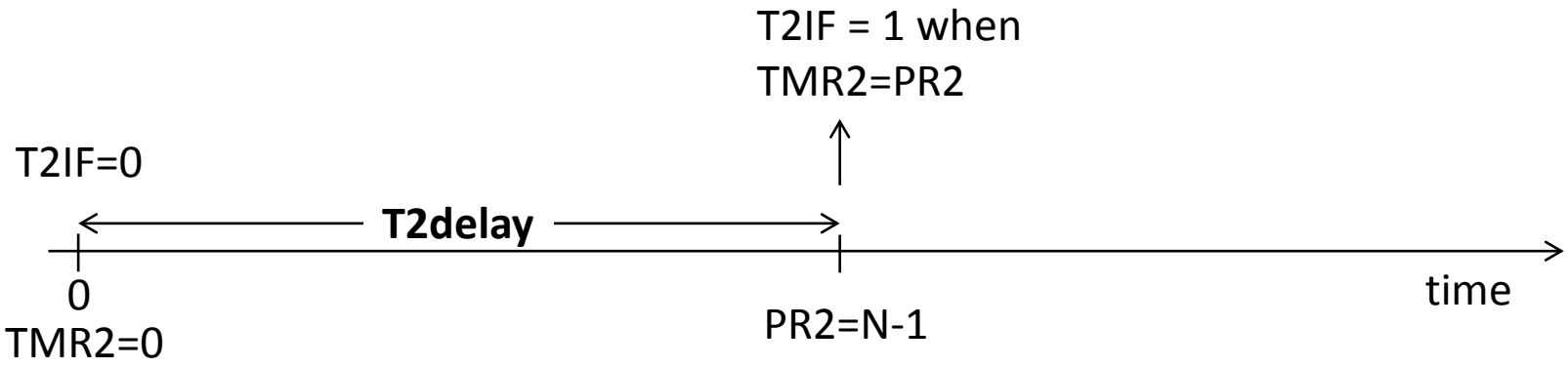
- TMR2 – the timer itself, can be loaded with initial value
- PR2 – preset register, set to target “time”
- T2CON – Timer2 CONtrol register with control bits for
  - Turning timer on/off (TON)
  - Selecting clock source (TCS, TCKPS, TGATE)
  - Set whether timer is enabled in idle mode (TSIDL)

  (T2CON detailed in THE DATASHEET)
- _T2IF - Timer 2 Interrupt Flag – bit in IFS0 Interrupt Flag Status Register 0

Using Timer as Delay

- Like an alarm clock (sort of)
  - Set PR register to the desired delay (future time)
  - Start timer
  - Timer "Flag" is set when Timer = PR+1

Timer 2 as Delay Details



T2delay = N* Tick where Tick is the selected timer clock period

In program:

1. Start timer at “0” and reset timer flag:
   - Reset TMR2
   - Reset T2IF
2. Set the delay time:
   - Set PR2 = N-1
3. Turn timer on
4. Wait for T2IF =1 (the Timer 2 “flag”)

Options for Responding to T2IF

1. Test for it with software using “condition”:

   (_T2IF) //true if T2IF = 1
   (!_ T2IF) //true if T2IF = 0
   (_T2IF == 1) //true if T2IF = 1
   (_T2IF == 0) //true if T2IF = 0

   Do NOT use a single “=“ in condition!
   (_T2IF = 0) //this assigns _T2IF to 0, and is always true!
   (_T2IF = 1) //this assigns _T2IF to 1, and is always true!

2. Use Interrupts (requires “enabling” the interrupt)

Testing Examples

```
while (! _T2IF);   //while _T2IF is 0, do nothing
do something    //when _T2IF is 1

while (_T2IF == 1) {    //while _T2IF is 1, do something
do something
}

if (_T2IF) {      //check if T2IF = 1
do something           // and if so, do something
}

if (_T2IF == 0)  {   //check if T2IF = 0
do something      // and if so, do something
}
```

Testing Inputs ➔"Polling"

The following are main programs that implement the "polling" method of responding to input events:

```
void main (void) {
config tasks
initialize tasks
while (1) {  //main infinite loop
      task 1
      wait for input;
         input tasks
      task 2
      task 3
      …
      }
}
```

"polling"

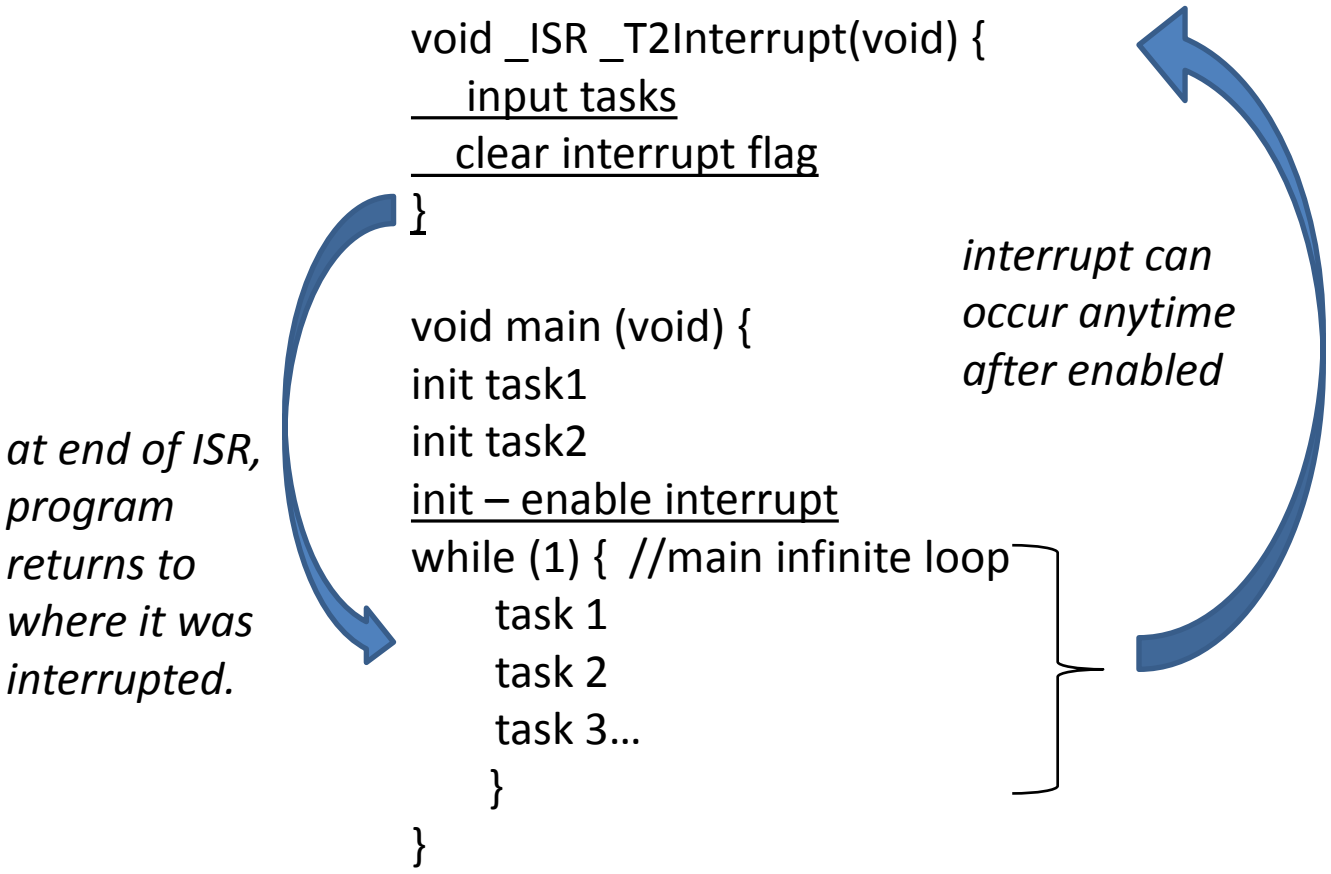
```
void main (void) {
config tasks
initialize tasks
while (1) {  //main infinite loop
      task 1
      if input {
           input tasks }
      task 2
      task 3
      …
      }
}
```

"polling"

What is the difference? How often will tasks 1, 2, 3 be executed in each case?

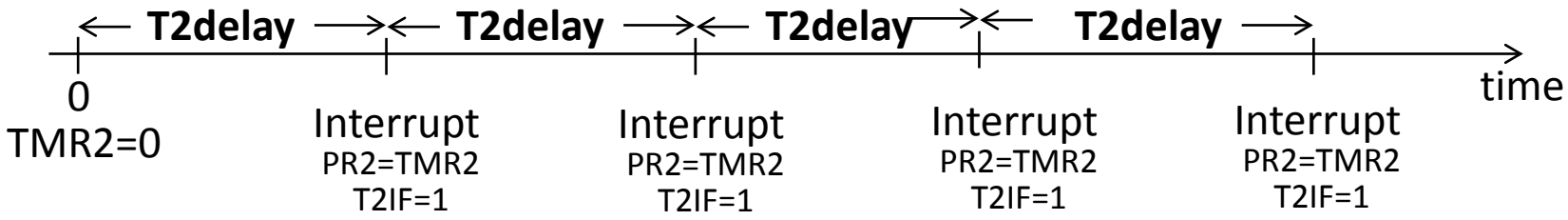
Using Interrupts to Respond to Inputs

Interrupts events "interrupt" the main program and cause a separate "interrupt service routine" (ISR) to execute. Control returns to main program when ISR completes.



Timer Interrupts

- Interrupt when TMRx=PRx
- Results in “periodic” interrupts



Reminder – Clock Timing

- FCY = Instruction clock frequency
- TCY = Instruction clock period
- The textbook configClock() library function sets

  FCY = 40 MHz

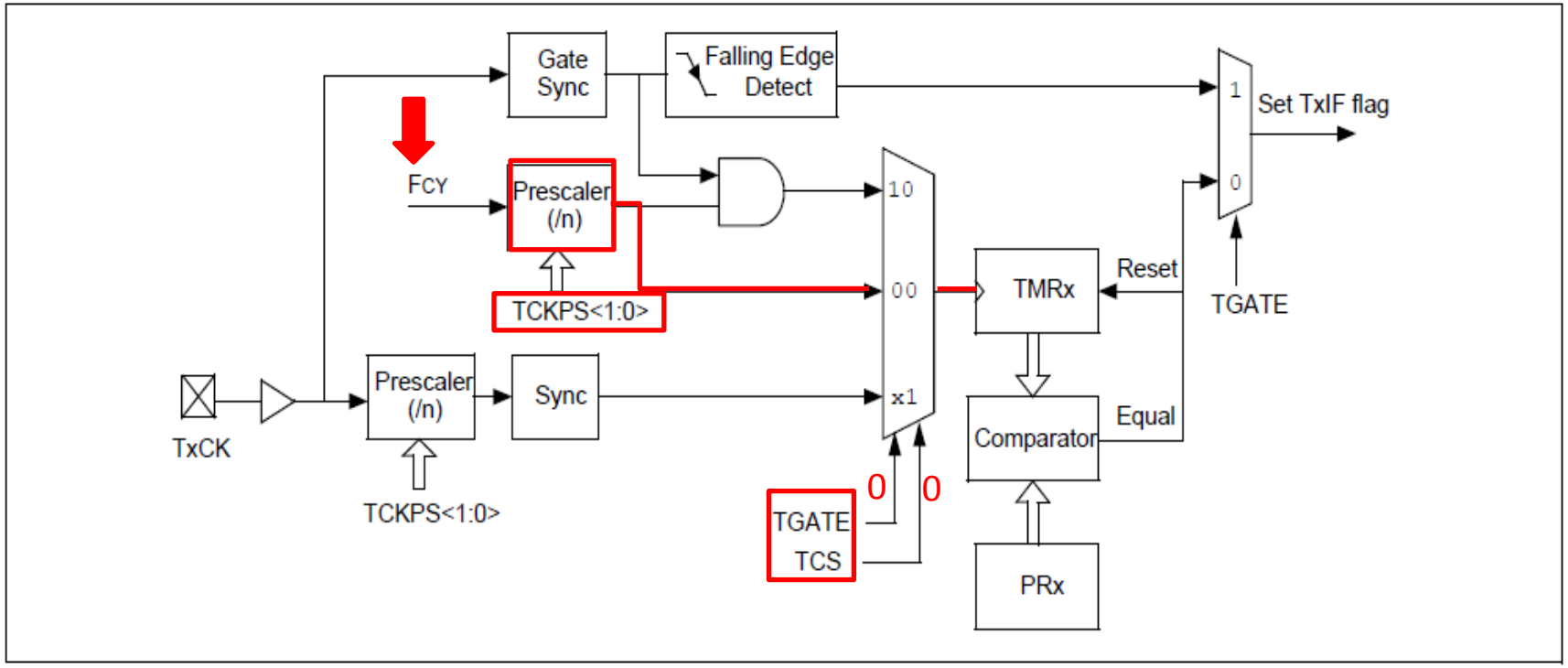
  TCY = 0.000000025s = 25 ns

If we use FCY as Timer 2 clock, how much time goes by while the timer counts from 0x0000 to 0xFFFF?

0xFFFF X 25ns = 0.00163s = 1.63ms

Timer 2 Clock Selection

FIGURE 13-1: TYPE B TIMER BLOCK DIAGRAM (x = 2 or 4)



T2CON – Timer 2 Control Register

Clock Prescale

TCKPS<1:0>: Timer1 Input Clock Prescaler Select bits

11 = 1:256
10 = 1:64
01 = 1:8
00 = 1:1

- For TCKPS<1:0> = 11
- Timer 2 Clock Frequency = 40 MHz/256
  = 156.25 KHz
- Timer 2 Clock Period: T_{tmr2} = 25ns * 256 =6.4 µs

T2CON – Timer 2 Control Register

REGISTER 13-1: TXCON: TIMER CONTROL REGISTER (X = 2 OR 4, Y = 3 OR 5)

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

U-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	U-0
—	TGATE	TCKPS<1:0>		T32	—	TCS	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

TON turns timer on and off
TSIDL stop in idle mode – we can leave this at default of 0
TGATE turns "gated accumulation" on or off. Leave it off.
TCKPS<1:0> selects the prescale value
T32 1 = 32-bit timer, 0 = 16-bit timer (default)
TCS Timer 2 clock source 0 = internal clock (default)

Setting Timer2 Bits

- All at once:

  T2CON = 0x0030 // TMR2 off, default idle mode, gate off, prescale 1:256, 32-bit mode off, internal clock

- One at a time:

  T2CONbits.TON = 0; //TMR2 off

  T2CONbits.TCKPS = 0b11; //prescale 1:256

- Using text library macros:

  T2CON = T2_OFF | T2_IDLE_CON |T2_GATE_OFF | T2_32BIT_MODE_OFF | T2_SOURCE_INT | T2_PS_1_256; //Results in 0x0030

Timer 2 Interrupt Flag and Enable

- _T2IF - Timer2 Interrupt Flag

  Bit that goes high when TMR2 = PR2

  ```
  while (! _T2IF);   //while _T2IF is 0, do nothing
  ```

- _T2IE -Timer2 Interrupt Enable

  Bit that must be set high to enable interrupts

  ```
  _T2IE = 1;  //enable interrupt
  ```

References for Timer 2 and Interrupts

P24HJ128GP502 Datasheet

- Available on Nexus
- Datasheet section 13 describes Timers 2 and 4 (identical)