4.10 Timers

Using timers for accurate timing

EE302 – Real time and embedded systems

Overview

- Aims
  - To understand and use timer peripherals to aid real time applications
- Learning outcomes – you should be able to…
  - Explain the operation of a timer device (based on the PIC16 family integrated timers)
  - Configure a timer device (prescaler, postscaler, etc) for a particular timeout or period
  - Calculate the maximum, minimum timeouts possible with a particular timeout
  - Write pseudocode that uses timer based timing

Programmable timer peripheral

- A hardware device which can be used to accurately mark time to schedule events in the future or time the duration of events
- Available as standalone devices or commonly integrated with microcontrollers
- A timer marks time by incrementing or decrementing a counter register once every tick
  - Generally, there is one tick per clock period derived from a clock signal (such as the main system oscillator) – the tick is synchronized to one of the clock edges
  - If a prescaler is used, the tick does not occur on each clock period but on some multiple of the clock period.
- Timer events
  - A **timer event** occurs when the **timer counter reaches a configured value** or the **timer counter register rolls over** (e.g. 8 bit register rollover is from 0xFF to 0x00)
  - When a timer event occurs, the timer may be configured to …
    - generate an interrupt – allows a programme to take action immediately (important for real time)
    - automatically reset itself – allows the timer to generate events with a regular period
    - Generate a digital signal that may drive cause an internal peripheral (on a microcontroller) or external peripheral to initiate some action

Contd.

- Progammable features
  - Configure the basic timer interval and any prescale/postscale multipliers to use
  - Sometimes: configure periodic expiry
  - Set/reset the time of next timer event
  - Get the current timer value

Timer/counter general operation

- Assume
  - Timer/counter increments by one each “tick”
  - For a B-bit timer, incrementing from 2^B-1 rolls over to 0
    - e.g. For an 8 bit timer/counter, incrementing 0xFF results in 0
- To have timer rollover event occur N ticks from now
  - First, ensure N <= 2^B
  - Then set the timer to 2^B – N
  - E.g. to have 8 bit timer rollover event occur in 5 ticks time, set timer = 256 – 5 = 251. On next 5 ticks the timer will be 252, 253, 254, 255, 0 (rollover = timer event!)

Periodic timing

- Assume we want a periodic timer event, every N ticks
  - Useful for PWM, ADC sampling, communications line sampling, etc.
- With timer reset in software (when no hardware reset available)
  - Set timer to rollover in N ticks
  - When timer rolls over and an interrupt occurs, reset the timer to rollover in N ticks again inside the interrupt handler code
  - Disadvantage: code must be written for this and the value set in the timer must be adjusted to allow for the time consumed by servicing the interrupt and resetting the timer
- With timer reset support in hardware (if available)
  - Set period to N, set timer to 0, configure periodic reset
  - Timer increments on each and is compared to period value on each tick. When it equals the period (N) the timer is reset to 0 and an interrupt generated
  - Advantage: no reset code required, and no time correction required

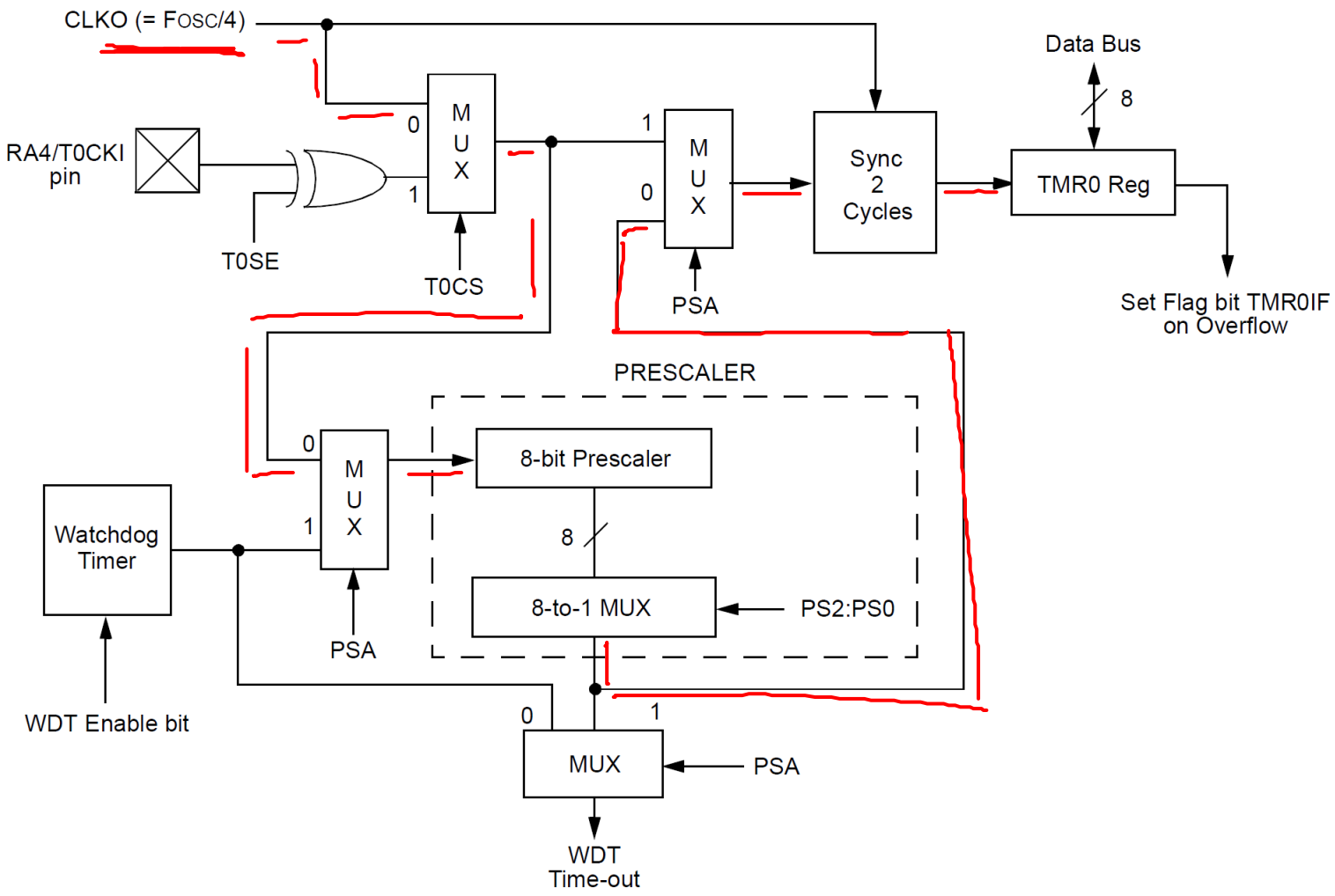
PIC16xxx Timers

- **Timer0** is an 8 bit timer/counter with a **prescaler** that supports multipliers of 1, 2, 4, 8, 16, 32, 64, 128, 256x.
  - A Timer0 interrupt (TMR0IF) may be generated on rollover from 0xFF to 0x00.
- **Timer1** is a 16 bit (2 register) timer/counter with a prescaler that supports 1, 2, 4, 8x.
  - A Timer1 interrupt (TMR1IF) may be generated on rollover from 0xFFFF to 0x0000
  - Alternatively, to support periodic events, a period value may be specified via the two Capture/Compare/PWM (CCP) devices. When Timer1 reaches the period value, the CCP1 module can reset the timer and generate an interrupt (CCP1IF), whereas the CCP2 module can reset the timer and generate an interrupt (CCP2IF) and automatically start an A/D conversion!
- **Timer2** is an 8 bit timer designed for periodic timing. It has a prescaler with 1, 4, 16x and **postscaler** with 1,2,3,…,16x.
  - An 8 bit period register (PR) dictates the (periodic) value at which Timer2 will reset back to zero. If the postscaler value is M, then after M Timer2 resets, the Timer2 interrupt (TMR2IF) will be generated.

ASIDE: timer terms

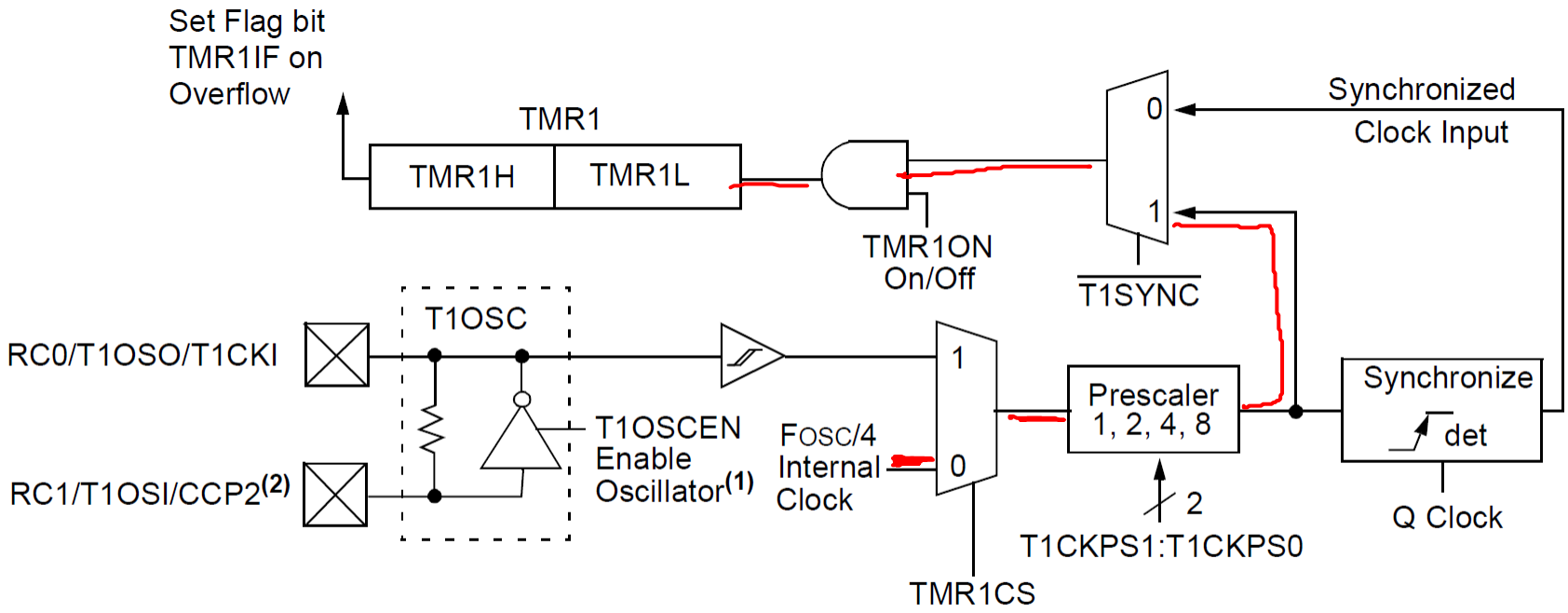
- Clock
  - A timer can usually be configured to use one of several different clock sources for its underlying timing input
  - E.g. PIC allows instruction clock, external clock, etc.
- Tick
  - The timer/counter value increments by one on each tick.
  - With no prescaling, each timer tick corresponds to exactly 1 cycle of the clock source. But this is not the case if it is pre-scaled…
- Pre-scaler
  - Sits between the source clock signal and the timer
  - A prescaler value of N means a timer tick takes place every N clock cycles instead of every clock cycle
- Post-scaler
  - A post-scaler sits between the timer reset and the timer event signalling functionality
  - A post-scaler value of M, means a timer event will be signalled only after the timer resets M times instead of each time it resets

Timer0



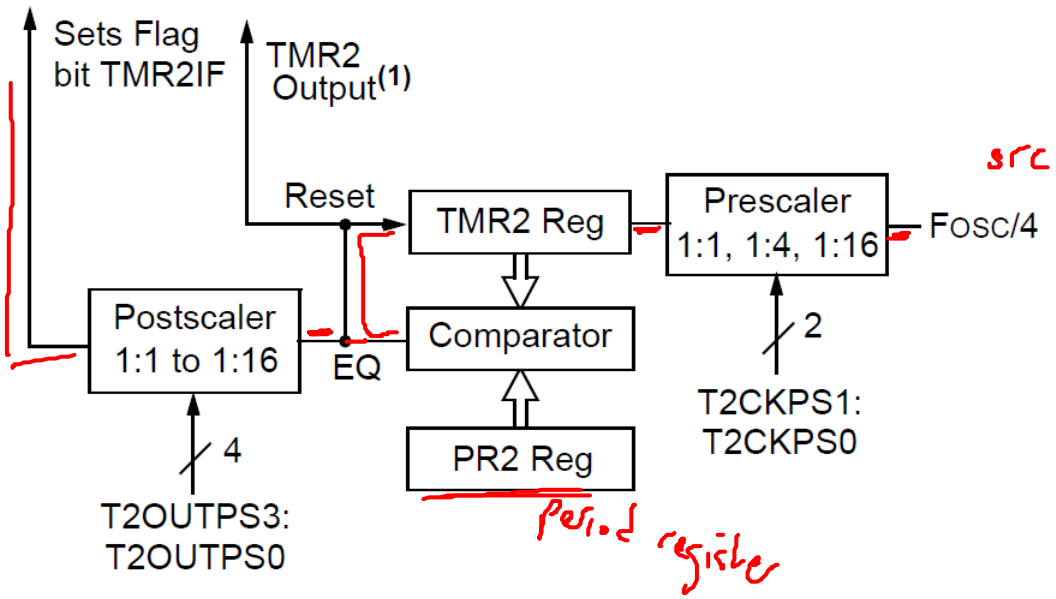
Note: T0CS, T0SE, PSA, PS2:PS0 are (OPTION_REG<5:0>).

Timer1



Note 1: When the T1OSCEN bit is cleared, the inverter is turned off. This eliminates power drain.

Timer2



Note 1: TMR2 register output can be software selected by the SSP module as a baud clock.

What does all this mean?

- Assume F_{OSC} = 4MHz — $F_{CYC} = F_{OSC}/4$
  - $T_{OSC}=1/F_{OSC}$
  - $T_{CYC} = 4 * T_{OSC}$ $= \frac{4}{4 \times 10^6} = 1\mu s$
- With Timer0 (8 bit) using T_{CYC}
  - What is the maximum duration that can be timed using 1x, 8x, and 256x prescaling?
  - How would you time a duration of 1 ms?
  - How would you time a duration of 1001 microsecs?

1x : 256 µs

8x : 8 × 256 = 2048 µs

256x : 256 × 256 = 65536 µs

use 4x prescaler ⇒ tick is 4 µs

⇒ 250 ticks × 4 µs/tick = 1000 µs

N = 250 ⇒ timer0 = 256 − N = 6

\ can't really do it

Contd.

- With Timer1 (16 bit) using T_{CYC} = 1µs, $2^{16} = 65536$
  - What is the maximum duration that can be timed using 1x, or 8x prescaling?
  - How would you time a duration of 1001 microsecs?

1µs x 1 x 65536 = 65536µs @ 1µs ticks

1µs x 8 x 65536 ≈ 512ms @ 8µs ticks

1x prescaler, N = 1001

⇒ TMR1 = $2^{16} - N$

- With Timer2 (8 bit) using T_{CYC}
  - What is the max duration that can be timed using 1x, 16x prescaling and 1x, 16x postscaling (all combinations)?
  - Is there any practical difference between using a prescaler and using a postscaler?

src	pre	maxN	post	
1µs	1	256	1	= 256µs
1µs	16	"	1	= 4096µs
1µs	1	"	16	= 4096µs
1µs	16	"	16	= 65536µs

Before looking at code/pseudocode to handle timers in more detail we need to quickly introduce interrupts….

Interrupt driven I/O

I/O devices “interrupt” CPU via an “interrupt request” (IRQ) line/pin as follows…

1. When an I/O device wants service it causes a transition on the CPU’s IRQ pin (or internal IRQ line when it is an internal peripheral)
2. CPU immediately(*) pauses main program execution
3. CPU executes Interrupt Service Routine (ISR) and ISR code services the device (e.g. by reading or writing data)
4. When ISR completes, CPU resumes program execution

Interrupt Service Routine - ISR

- ISR is the code that is executed to service the device which caused the interrupt
- CPU saves PC (and possibly SP) register
  - ISR must explicitly save and later restore any other registers it needs to use (called context switching) – C compiler handles this for us
- Guidelines
  - ISRs must be kept as short as possible, to minimise delays to main program and other interrupts
  - Care needs to be taken when ISR and main program share some variables (e.g. when main program and ISR both access the same data buffer)
    - Main part of program may need to disable (or mask) interrupts when accessing variables shared with ISR
  - Usually disable interrupts while executing an ISR (but it is possible to nest interrupts on some processors)

Interrupt driven I/O - analogy

- You need a colleague to do something for you (e.g. get some information)
- Go ask them to do the job (this is the I/O request)
- Unlike polling (where you wait for job to be done or else check back regularly), instead you ask your colleague to contact you when job done. This allows you to immediately continue with other work.
- Later when your colleague is done, they give you a call (this is the interrupt)
- You save your current work, perhaps marking the page in a book, or whatever. (This is pausing the main program)
- You go back to your colleagues office to pick up the job output (this is the work of the ISR and servicing the interrupt)
- When you get back to your office, you pick up your previous work where you left off (this is resuming the main program)

Example – interrupt driven timing for superloop


```
// NOTE: Use a timer to guarantee periodic superloop timing
main()…
setup()…
loop()
  while timerExpired is FALSE, do nothing // i.e. wait
  set timerExpired = FALSE // ready to wait again
  doTimeConsumingTask()
  // no extra delay required - handled by waiting for timer

doTimeConsumingTask()…

isr() // polled interrupt service routine
  if (timer interrupt)
    // assume timer not setup for periodic reset so we must
    // explicitly reset it here
    set timer = SUPERLOOP_TICKS
    set timerExpired = TRUE
```

Exam question: Q4b, 2011-2012

On the PIC 16F877 microcontroller, Timer0 is an incrementing 8-bit timer that generates an interrupt on rollover (from 0xFF to 0x00). Timer0 also incorporates a prescaler which can be configured such that the timer increments on every 1, 2, 4, 8, 16, 32, 64, 128, or 256 instruction cycles.

$T_{src} = \frac{1}{F_{OSC}/4}$

Consider a system using the PIC 16F877 with an oscillator frequency of 4 MHz. The system must use its internal ADC to sample analogue input at a rate of 2000 samples/second.

1. For this system, what is the maximum timeout period supported by Timer0 with the minimum and maximum prescaling factors applied?
2. Choose an appropriate prescaling factor and Timer0 start value to achieve a timeout equal to the required sampling period of this system.

VARIATION: Same question as above but use Timer2 with prescaler AND postscaler

Q1

$T_{src} = 1\mu s$

$1\mu s \times 1 \times 256 = 256\mu s$, min is $1\mu s$

$1\mu s \times 256 \times 256 = 65536\mu s$, min is $256\mu s$

Q2

$T_{sample} = \frac{1}{200} = 500\mu s$

desired $= \frac{500\mu s}{256} = 1.??$ $\Rightarrow$ prescale $= 2\times$

$N = \text{round}\left(\frac{500\mu s}{1\mu s \times 2}\right) = 250$

$\Rightarrow TMR0 = 2^8 - N = 6$

Exam question: Q4b, 2011-2012 – contd.

System outline code is as shown.

The polled interrupt routine must use the timer interrupt (TMR0IF) to determine when to start an ADC conversion and use the end of ADC conversion interrupt (ADIF) to determine when processSignal should do its processing.

Write the pseudocode implementation of this polled interrupt routine, setting variables needed by the superloop functions as necessary. Ensure the timing is as specified previously. Assume that the Timer0 register is TMR0, that ADC conversion is started by setting the ADGO bit, and that we are only interested in 8-bit ADC samples from register ADRESH.

```
global gSampleToProcess = FALSE
global gSample = 0

loop():
  processSignal()
  serviceUSB()

processSignal():
  if gSampleToProcess is TRUE:
    // ready for next sample
    set gSampleToProcess = FALSE
    … // do the processing here

serviceUSB():
  … // details unimportant
```

come back after next notes

Timer summary

- B-bit timer => max num increments before expiry = 2^B

- Timer expiry/period match
  - Timer with rollover: To expire in N increments, start at 2^B-N
  - Timer with period register and period reset: To expire in N increments, set timer to 0 and set period register (PR) to N-1.

- Prescaler/postscaler
  - Prescaler – defines ratio of timer increments to source clock cycles

    e.g. 1:4 or 4x means 4 source clock cycles required for every 1 timer increment, i.e. $T_{INC} = 4*T_{SRC}$
  - Post-scaler – defines ratio of timer interrupts to timer expiry or period match events

    e.g. 1:2 or 2x means 2 timer expiry/period match events required for every 1 timeout interrupt, i.e. $T_{INTR} = 2*T_{EXP}$

Time summary contd.

- Timeout duration (T_{INTR}) given N, prescale, and postscale

  $T_{EXP} = T_{SRC} * prescale * N$

  $T_{INTR} = T_{EXP} * postscale$

  $T_{INTR} = \frac{T_{src} \times pre \times N \times postscale}{T_{TICK}}$

- N, prescale, and postscale given desired timeout duration (T_{INTR})
  - First determine if any pre/post-scale required
    - desiredScale = $T_{INTR} / (T_{SRC} * 2^B)$
    - If desiredScale < 1, then **N = round (T_{INTR} / T_{SRC}), prescale = postscale = 1**
    - If desiredScale >= 1, then
      - To make the timer resolution as good as possible, choose from available prescale and postscale values so that **totalScale = prescale * postscale** is the smallest integer which is **>= desiredScale**
      - Then calculate **N = round (T_{INTR} / [T_{SRC} * prescale * postscale])**
  - PercentageTimeoutError = 100 * $T_{INTR\text{-}calculated}$ / $T_{INTR\text{-}desired}$