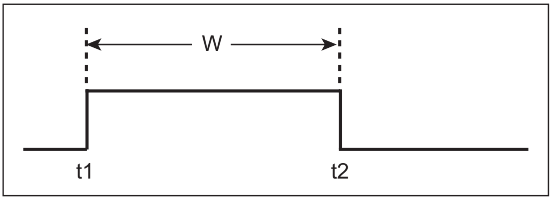
TIP #3 Measuring Pulse Width

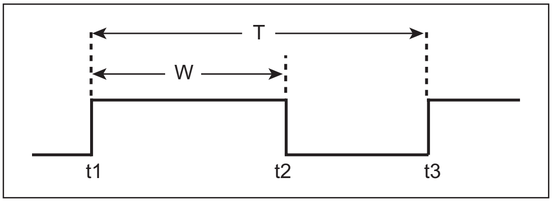
Figure 3-1: Pulse Width



1. Configure control bits CCPxM3:CCPxM0 (CCPxCON<3:0>) to capture every rising edge of the waveform.
2. Configure Timer1 prescaler so that Timer1 will run W_{MAX} without overflowing.
3. Enable the CCP interrupt (CCPxIE bit).
4. When CCP interrupt occurs, save the captured timer value (t1) and reconfigure control bits to capture every falling edge.
5. When CCP interrupt occurs again, subtract saved value (t1) from current captured value (t2) – this result is the pulse width (W).
6. Reconfigure control bits to capture the next rising edge and start process all over again (repeat steps 3 through 6).

TIP #4 Measuring Duty Cycle

Figure 4-1: Duty Cycle



The duty cycle of a waveform is the ratio between the width of a pulse (W) and the period (T). Acceleration sensors, for example, vary the duty cycle of their outputs based on the acceleration acting on a system. The CCP module, configured in Capture mode, can be used to measure the duty cycle of these types of sensors. Here’s how:

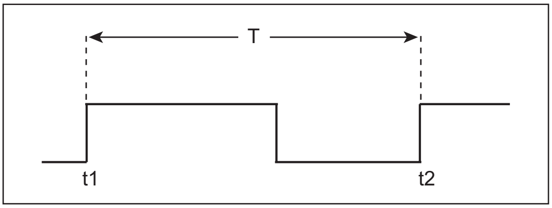
1. Configure control bits CCPxM3:CCPxM0 (CCPxCON<3:0>) to capture every rising edge of the waveform.
2. Configure Timer1 prescaler so that Timer1 will run T_{MAX}(1) without overflowing.
3. Enable the CCP interrupt (CCPxIE bit).
4. When CCP interrupt occurs, save the captured timer value (t1) and reconfigure control bits to capture every falling edge.

> **Note 1:** T_{MAX} is the maximum pulse period that will occur.

5. When the CCP interrupt occurs again, subtract saved value (t1) from current captured value (t2) – this result is the pulse width (W).
6. Reconfigure control bits to capture the next rising edge.
7. When the CCP interrupt occurs, subtract saved value (t1) from the current captured value (t3) – this is the period (T) of the waveform.
8. Divide T by W – this result is the Duty Cycle.
9. Repeat steps 4 through 8.

TIP #1 Measuring the Period of a Square Wave

Figure 1-1: Period



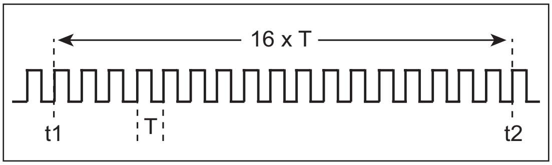
1. Configure control bits CCPxM3:CCPxM0 (CCPxCON<3:0>) to capture every rising edge of the waveform.
2. Configure the Timer1 prescaler so Timer1 with run T_{MAX}(1) without overflowing.
3. Enable the CCP interrupt (CCPxIE bit).
4. When a CCP interrupt occurs:
   a) Subtract saved captured time (t1) from captured time (t2) and store (use Timer1 interrupt flag as overflow indicator).
   b) Save captured time (t2).
   c) Clear Timer1 flag if set.

The result obtained in step 4.a is the period (T).

Note 1: T_{MAX} is the maximum pulse period that will occur.

TIP #2 Measuring the Period of a Square Wave with Averaging

Figure 2-1: Period Measurement



1. Configure control bits CCPxM3:CCPxM0 (CCPxCON<3:0>) to capture every 16th rising edge of the waveform.
2. Configure the Timer1 prescaler so Timer1 will run 16 T_{MAX}(1) without overflowing.
3. Enable the CCP interrupt (CCPxIE bit).
4. When a CCP interrupt occurs:
   a) Subtract saved captured time (t1) from captured time (t2) and store (use Timer1 interrupt flag as overflow indicator).
   b) Save captured time (t2).
   c) Clear Timer1 flag if set.
   d) Shift value obtained in step 4.a right four times to divide by 16 – this result is the period (T).

Note 1: T_{MAX} is the maximum pulse period that will occur.

The following are the advantages of this method as opposed to measuring the periods individually.

- Fewer CCP interrupts to disrupt program flow
- Averaging provides excellent noise immunity

Pulse Width Measurement: Timer0 main

```
 pcrlf();printf("Ready for button mashing!");pcrlf();
 while(1){
   capture_flag = 0;
   // clear timer0, write low byte last
   TMR0H = 0;
   TMR0L = 0;
   INTEDG0 = 0; // falling edge
   INT0IE = 1; //RB0 Interrupt
   while(!capture_flag); // wait for capture
   // compute time in microseconds
   pulse_width_float = TMR0TIC * tmr0_tics * 1.0e6;
   pulse_width = (long)pulse_width_float;
   printf ("Switch pressed, timer ticks: %d, pwidth: %ld (us)",
           tmr0_tics,pulse_width); pcrlf();
 }
}
```

Wait for pulse width to be captured by ISR (arrow points to `INT0IE = 1; //RB0 Interrupt`)

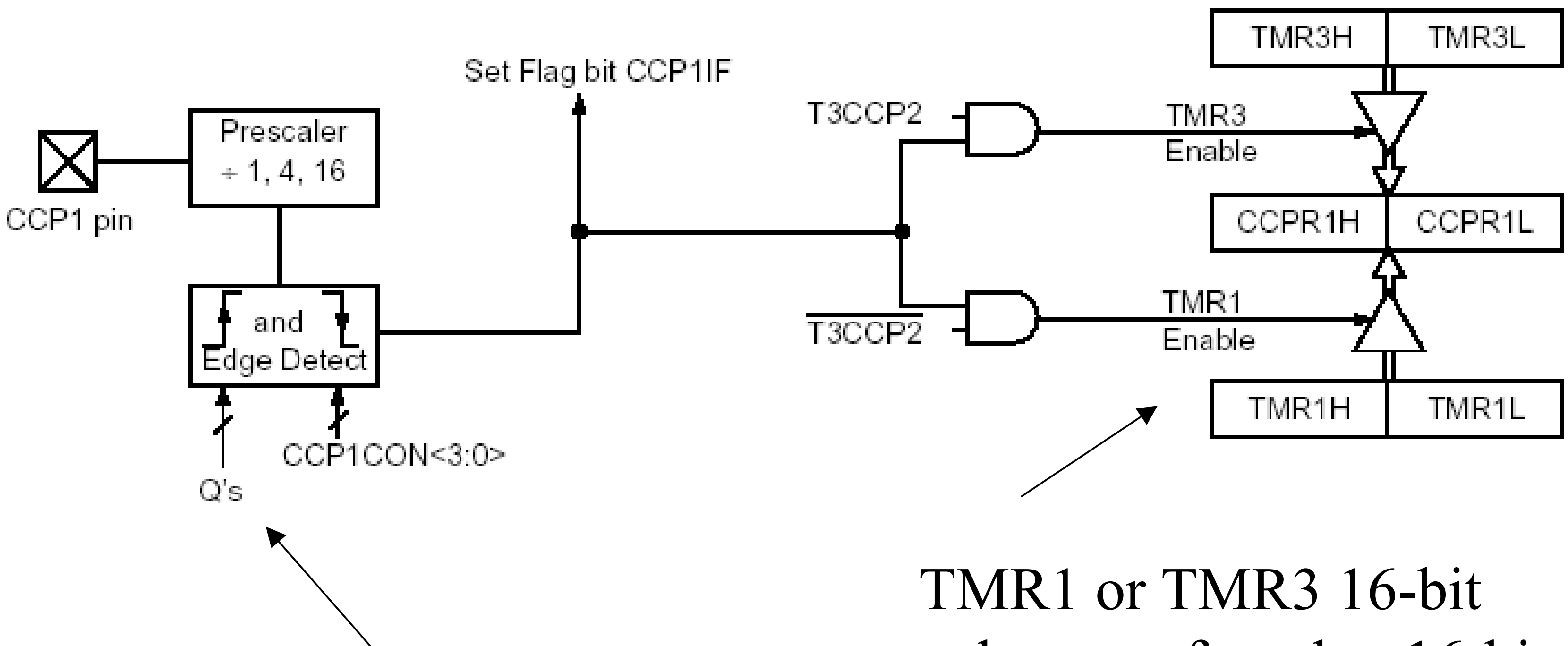
Convert Timer0 tics to microseconds (brace beside the `pulse_width_float` and `pulse_width` lines)

Configuration code before loop is not shown.

This works of for human activated pushbutton time measurement, but if more accurate measurements are needed, then use the **Capture** module.

Capture Module Time Measurement

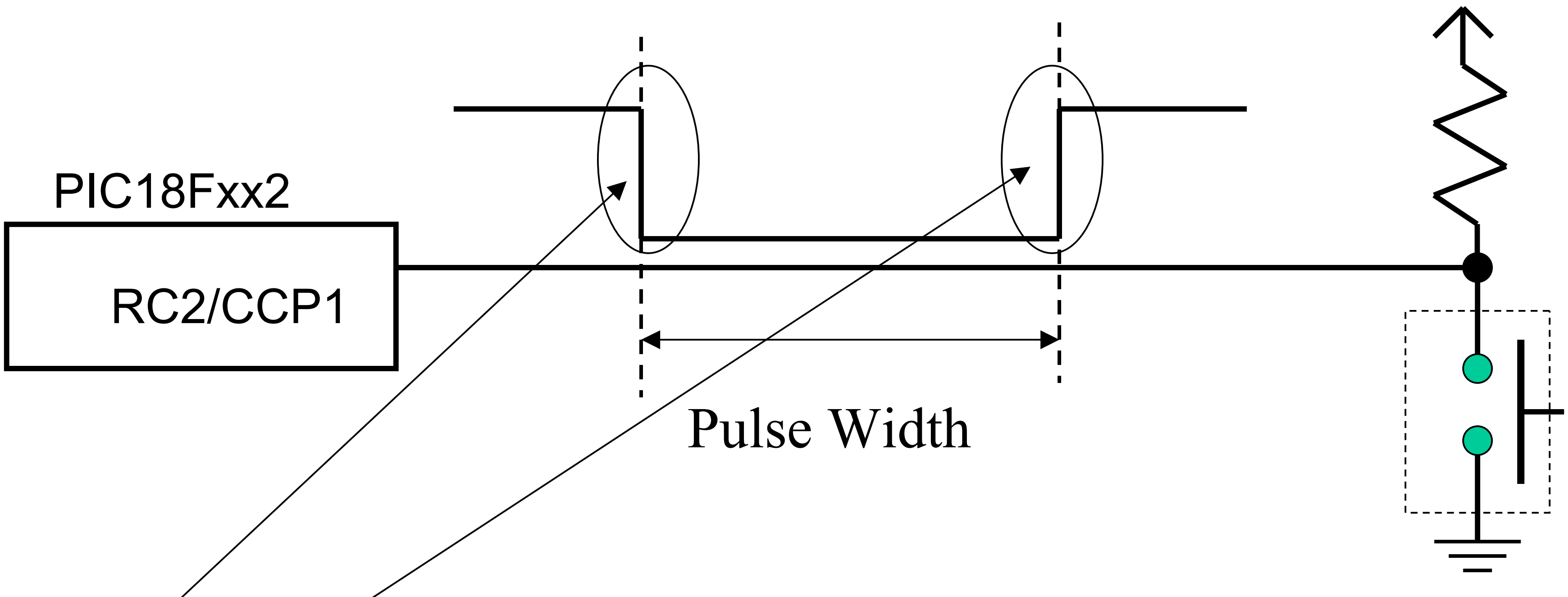
- **Capture Mode** of the Capture/Compare/PWM module is used for time measurement.



Rising or falling edge detect, with interrupt flag set.

TMR1 or TMR3 16-bit value transferred to 16-bit capture register on edge detect.

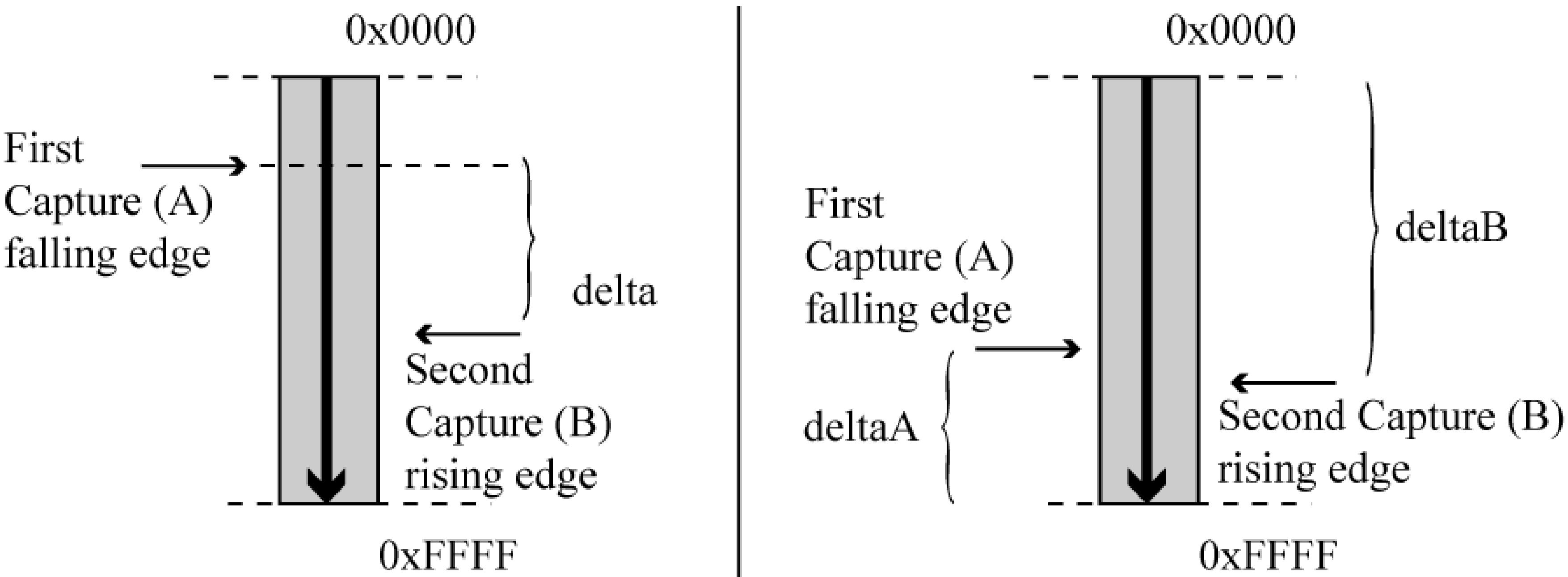
Measuring pushbutton pulse width



1. Capture TMR1 value on falling edge (Tf) in CCPR1
2. Capture TMR1 value on rising edge (Tr) in CCPR1
3. Pulse width = Tr – Tf (Elapsed Timer1 Tics)

Use interrupt to save timer values.

Computing Pulse Width



(a) No overflow case
TimerDelta = B - A

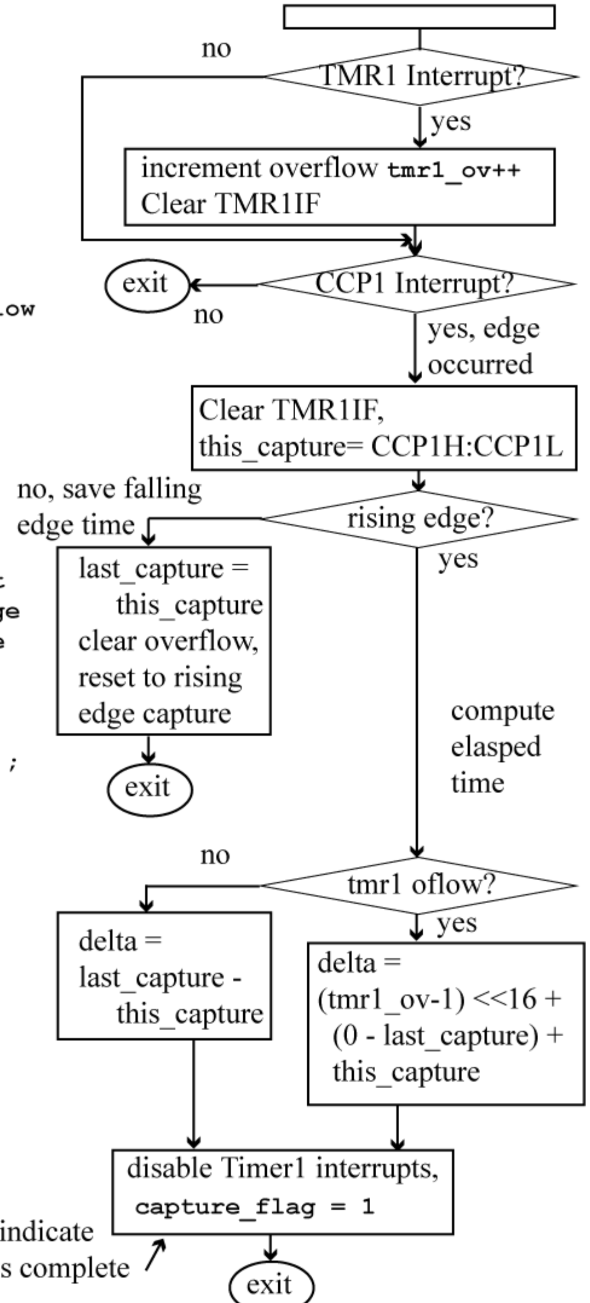
(b) Overflow case
TimerDelta = (#oflows-1)* 65536 + deltaA + deltaB
= (#oflows -1) << 2^{16} + (0 - A) + B

In overflow case, the value can be greater > 16 bits so need to use a LONG type to hold TimerDelta value.

```
volatile unsigned int last_capture;
volatile unsigned int this_capture;
// this must be long
volatile unsigned long delta;
// timer 1 overflow cnt
volatile unsigned char tmr1_ov;
volatile unsigned char capture_flag;

timer_isr(void){
 if (TMR1IF) {
  tmr1_ov++;  // increment timer1 overflow
  TMR1IF = 0;
 }
 if (CCP1IF) {
  // read CCPR1 as a 16-bit value
  this_capture = CCPR1;
  if (!bittst(CCP1CON,0)) {
   //falling edge
   last_capture = this_capture;
   tmr1_ov = 0;  // clear overflow count
   CCP1CON = 0x0; // turn off when change
   CCP1CON = 0x5; // capture rising edge
  } else {
   if (!tmr1_ov) {
    // no overflow at all
    delta = this_capture - last_capture ;
   }
   else {
    // compute delta time
    delta = tmr1_ov-1;
    delta = (delta << 16);
    last_capture = 0 - last_capture;
    delta = delta + last_capture;
    delta = delta + this_capture;
   }
   // disable timer1 interrupt
   TMR1ON = 0; TMR1IE = 0; TMR1IF = 0;
   capture_flag = 1;
  }
  //clear capture interrupt flag
  CCP1IF = 0;
 }
}
```

Semaphore to `main()` to indicate that pulse width capture is complete



ISR for capturing pulse width.

tmr1_ov variable keeps track of timer1 overflows.

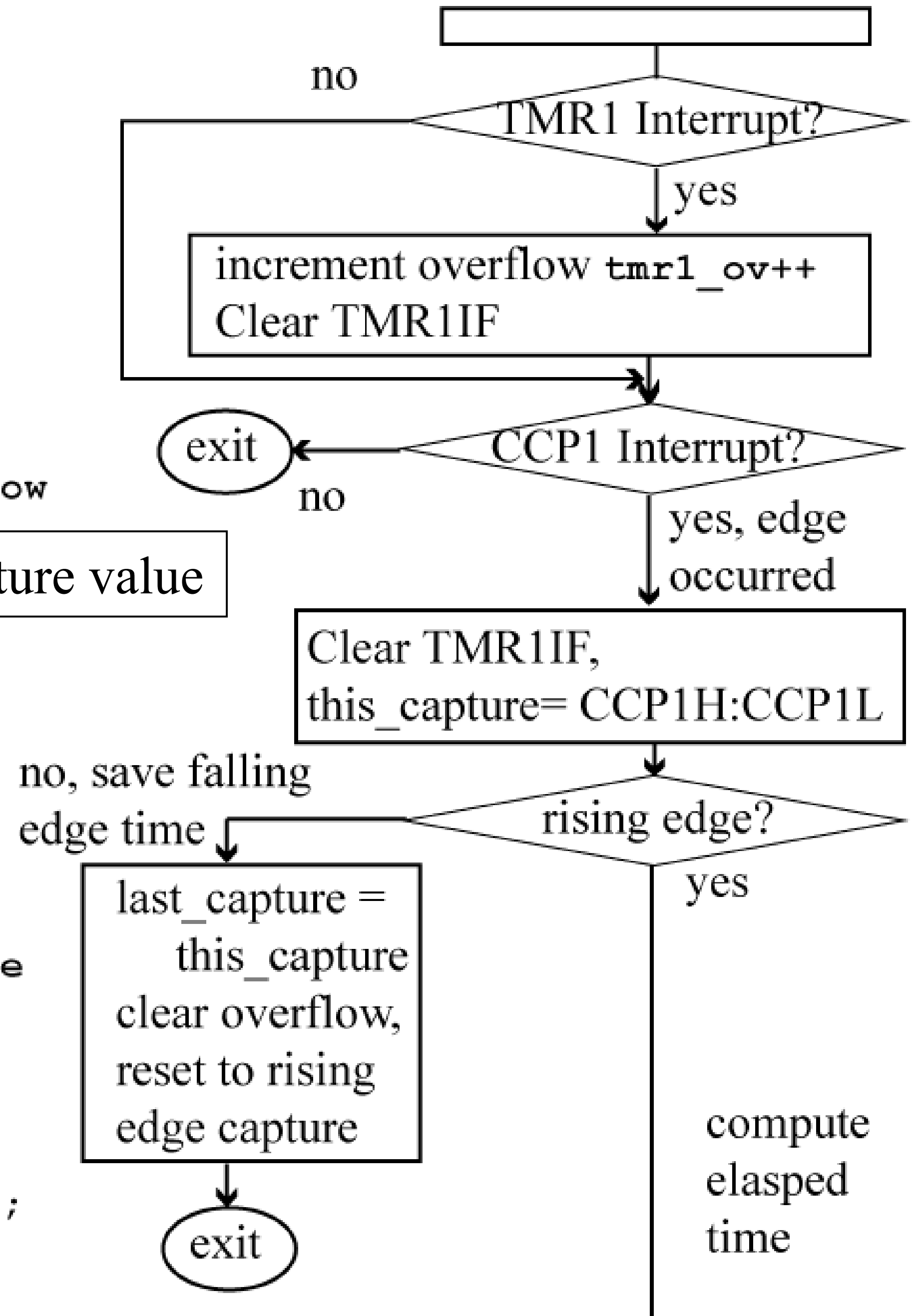
After falling edge, reconfigure for rising edge capture.

After rising edge, compute delta timer tics

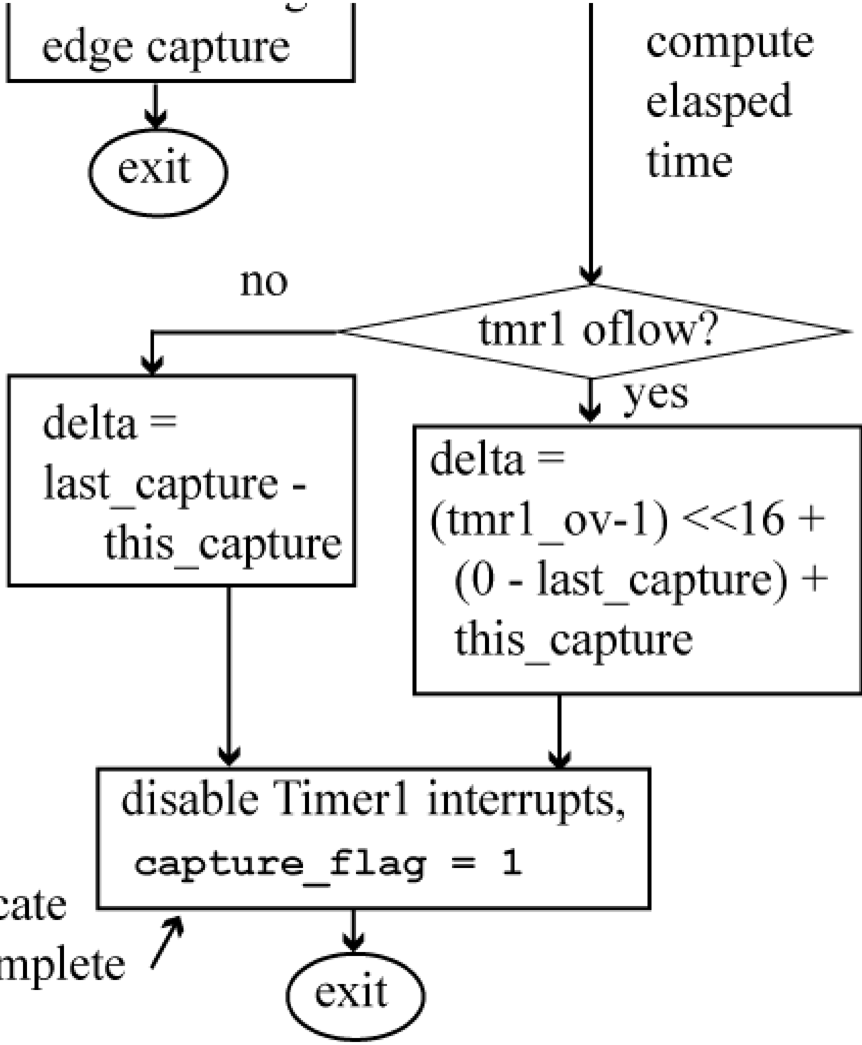
```
volatile unsigned int last_capture;
volatile unsigned int this_capture;
// this must be long
volatile unsigned long delta;
// timer 1 overflow cnt
volatile unsigned char tmr1_ov;
volatile unsigned char capture_flag;

timer_isr(void){
 if (TMR1IF) {
  tmr1_ov++;  // increment timer1 overflow
  TMR1IF = 0;
 }
 if (CCP1IF) {
  // read CCPR1 as a 16-bit value
  this_capture = CCPR1;
  if (!bittst(CCP1CON,0)) {
   //falling edge
   last_capture = this_capture;
   tmr1_ov = 0;  // clear overflow count
   CCP1CON = 0x0; // turn off when change
   CCP1CON = 0x5; // capture rising edge
  } else {
   if (!tmr1_ov) {
    // no overflow at all
    delta = this_capture - last_capture ;
   }
   else {
```

Read 16-bit capture value



```
  if (!tmr1_ov) {
   // no overflow at all
   delta = this_capture - last_capture ;
  }
  else {
   // compute delta time
   delta = tmr1_ov-1;
   delta = (delta << 16);
   last_capture = 0 - last_capture;
   delta = delta + last_capture;
   delta = delta + this_capture;
  }
  // disable timer1 interrupt
  TMR1ON = 0; TMR1IE = 0; TMR1IF = 0;
  capture_flag = 1;
 }
 //clear capture interrupt flag
 CCP1IF = 0;
}
}
```



Semaphore to `main()` to indicate that pulse width capture is complete

After pulse width is captured, the *capture_flag* semaphore is set and the Timer0 interrupt is disabled as the pulse width has been measured.