EXAMPLE RPM_1

OBJECTIVE

This example has the following objectives:

- Review the use of MCU Timer function as an Input Capture (IC) device
- Review the use of the free running clock, TCNT, and its overflow flag, TOF
- Review the use of a input capture clock, TIC1, and its event flag, IC1F
- Demonstrated how the selection of signal transition to be captured is made (here rising edge, EDG1B) and that the MCU is only sensitive to that particular transition.
- Introduce the use of soft masks (here, IC1_MSK, TOF_MSK).
- Introduce the concept of layered flowcharting of a program: big-picture with generic descriptions; detailed flowcharts for particular sections of the program.

PROGRAM EX_RPM_1

This program EX_RPM_1 captures the times T1 and T2 required in the tachometer RPM experiment (Figure 1). In this experiment, a disk with an aperture (hole) drilled into it is attached to the shaft of a DC motor. The aperture in the disk spins through an infrared emitter-detector sensor. The emitter-detector sensor sends a High (5V) signal when the aperture in the disk allows the beam of light to pass through. The output of the emitter-detector sensor is attached to the input capture pin IC1 on Port A. The speed of the motor is controlled by a potentiometer electronic circuit. The potentiometer is a variable resistor. By turning the knob of the potentiometer, the speed of the DC motor is varied. T1 is the first time when a falling edge transition is encountered on pin IC1, while T2 is the second time when a falling edge transition is encountered on the same pin IC1. This corresponds to the time between two consecutive passages trough the emitter-detector sensor of the hole in the rotating disk. The difference between T2 and T1, plus the time taken by the overflows, will represent the period of rotation of the disk. Hence, using EX_RPM_2, one can calculate the rotation speed.

Figure 1  DC Motor tachometer experiment to which the present software example is relevant.
FLOWCHART AND CODE

The program flowchart is show below. Two flowchart levels are presented: the big-picture and the details. The big-picture is used to understand the overall architecture of the program. The details are used to explain some of the blocks. (Details are given only for those blocks which are somehow new, and have not been used in previous programs.) This flowchart was use to generate the code file Ex_RPM_1.asm using the standard asm template.

---

**Big-picture Flowchart**

- Define variables
- Initialize Y= REGBAS
- Initialize IC1 for falling: set EDG1B in TCTL2
- Reset IC1F
- Check first IC1 capture
  - Reset IC1F
  - Load and store TIC1 in T1
  - Clear TOF
  - Zero overflow counter NOF
- Check TOF
  - If TOF is set, increment NOF and reset TOF
  - Check second IC1 capture
  - Load and store TIC1 in T2
- SWI

**Detailed Flowcharts**

- Define variables:
  - Overflow counter, NOF1 (1 byte)
  - 1st capture, T1 (2 bytes)
  - 2nd capture, T2 (2 bytes)
  - IC1_MSK=%00000100
  - TOF_MSK=%10000000
- Check IC1F
  - N
  - Y
  - Check TOF
    - Y
      - Increment NOF
      - Reset TOF
    - N
  - Check IC1F
  - Y
  - N
EXECUTION

Open THRSim11. Close the Commands window. View CPU registers, timer registers, port A pins, memory list. Open and assemble Ex_RPM_1.asm. Set breakpoints at:

- $c016 (after first input capture on IC1)
- $c030 (when an overflow happens)
- $c03f (after second input capture on IC1)
- SWI.

Reset registers and memory. Set standard labels (Label/Set Standard Labels). Set display of TCTL2, TFLG1, TFLG2 to binary. Arrange windows for maximum benefit: Press the RESET button. Your screen should look like this:

a) Toggle pin IC1 to 1. Start the program. The program stays in a loop on LABEL 1 waiting for a falling edge transition on pin IC1.
b) Toggle pin IC1 to 0. The program stops at the first breakpoint. Notice that IC1F is set in TFLG1 (bit 2). Also notice the value captured in TIC1. This value should be very close to the value in TCNT (It is not identical, since the program executed a few operations before stopping at the breakpoint, hence TCNT is slightly more than TIC1.) Your screen looks like this:
c) Step through your program until the program counter is on LABEL2. Note that IC1F is cleared. Then, TIC1 is loaded into D and then stored in T1. Further, TOF is cleared. Finally, NOF is zeroed. Your screen looks like this:
d) Run. The program enters the second loop on LABEL2. You will notice the program jumping over a section of code to LABEL3 and then back to LABEL2. The section of code that is bypassed refers to the situation when an overflow takes place. Let the program run until the timer counter, TCNT, goes over. At this point, the program should stop at the breakpoint $c030. Your screen looks like this:
e) Step through your program up to LABEL3. Notice the overflow counter, NOF, being incremented, and the overflow flag, TOF, being reset in TFLG2 (bit 7). Your screen looks like this:
f) Let the program run. Toggle pin IC1 back to 1. Nothing should happen since only falling edges should be detected. Toggle IC1 to 0 again. The program should stop at the second breakpoint. The value of TIC1 should be updated to a new value. Your screen should look like this:
g) Step through your program. The new value of TIC1 is loaded and then stored in T2. The program branches back to the beginning (LABEL0). Your screen looks like this:

After returning to the beginning, the program zeros the NOF counter and goes through the process again.

h) Let the program run again and repeat the steps above until you are satisfied that you understand the process well.

i) Run again but, after each breakpoint, simply hit the run button and let the program run until the next breakpoint. Make sure you understand how the generation of a falling signal on pin IC1 (by you toggling from 1 to 0) generates an input capture.

j) Make sure to run the program long enough such that at least one overflow takes place. Observe how the overflow counter is incremented and the overflow flag is reset.

k) Synchronize your toggling of pin IC1, such that T1 will be just after an overflow and T2 is several cycles later, i.e., T1 < T2. In this situation, T1 and T2 are on the same side of
the timer overflow event and the overflow counter, NOF, should be zero. Write down these values in Table 1, Case 1.

l) Synchronize your toggling of pin IC1, such that T1 will be just before an overflow, while T2 will happen just after the overflow. In this case, the value of T2 will be less than the value of T1, i.e., T2 < T1. Since T1 and T2 are on different sides of the timer overflow event, the overflow counter should register NOF = 1. Write down the T1 and T2 values in Table 1, Case 2.

**Table 1**

<table>
<thead>
<tr>
<th>Case</th>
<th>NOF</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1: T1 &lt; T2 and on the same side of the timer overflow event</td>
<td></td>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td>Case 2: T2 &lt; T1, but on opposite sides of the timer overflow event</td>
<td></td>
<td>$</td>
<td>$</td>
</tr>
</tbody>
</table>

**WHAT YOU HAVE LEARNED**

In this example, you have learned:

- Review the use of MCU Timer function as an Input Capture (IC) device
- Review the use of the free running clock, TCNT, and its overflow flag, TOF
- Review the use of a input capture clock, TIC1, and its event flag, IC1F
- Demonstrated how the selection of signal transition to be captured is made (here rising edge, EDG1B) and that the MCU is only sensitive to that particular transition.
- Introduce the use of soft masks (here, IC1_MSK, TOF_MSK).
- Introduce the concept of layered flowcharting of a program: big-picture with generic descriptions; detailed flowcharts for particular sections of the program.