EXAMPLE INPUT CAPTURE TIMER FUNCTION

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 an input capture clock, TIC1, and its event flag, IC1F
- Demonstrated how the selection of signal transition to be captured is made (here rising edge, EDG1A) and that the MCU is only sensitive to that particular transition.
- Show the calculation of actual time in $\mu$s from the timer readings, T1, T0, and overflow count, NOF.

PROGRAM EX_IC
This program is an example of timer input capture. The process is started when a keystroke is received. Then, the time is measured until a low-high transition is captured on line IC1. In this process, the initial time, T0, and the capture time, T1, as well as the number of overflows are recorded. After the input capture, the IC1F flag is reset, the overflow counter is zeroed, and the process is repeated.

FLOWCHART AND CODE
The program flowchart is show to the right of the program instructions. Note the variable definition block in which T0, T1, and NOF are defined. Next, the initialization block contains reg. X initialization, timer IC initialization, and SCI initialization. The program loop starts at label BEGIN with the overflow counter, NOF, being zeroed. First, the RDRF (reception data register full) flag is checked in a loop to verify if a keystroke has been received. When keystroke was received, the time counter is read and stored in T0. Then, the programs loops on LABEL1 until input capture IC1 is recorded. In this loop, TOF is first check to verify if timer overflow takes place. When timer overflow is detected, the overflow counter, NOF, is incremented and TOF is reset. Next, IC1F is check to verify if input capture on IC1 took place. If input capture is not detected, the program returns to LABEL1. When input capture is detected, the program exits the loop, loads the IC1 timer from TIC1 and stores it in the capture time variable, T1. The program loops back to the beginning and wait for a new keystroke to restart the process.

The essential code for the program is shown to the right of the program flowchart. This essential code was incorporated into the standard asm template to generate the file Ex_IC.asm.
Instructions

i) Define variables:
   • Origin of time, $T_0 = 2$ bytes
   • Capture time, $T_1 = 2$ bytes
   • Overflow counter, $NOF_1 = 1$ byte

ii) Initialize
   • Initialize index $X$ to REGBAS
   • Initialize timer IC1 function: set EDG1A in TCTL2
   • Initialize SCI

iii) Zero overflow counter $NOF_1$

iv) Wait for a keystroke reception

v) Store initial time

vi) Wait for the input capture
   • Check TOF
   • Jump if no TOF; else
     o Increment overflow counter
     o Reset TOF
   • Check IC1F
   • Loop back

vii) After input capture
   • Load and store $t_1$
   • Reset IC1F by writing 1 to it

viii) Loop back to iii) and do it again

Flowchart

Code

\begin{verbatim}
ORG DATA
T0 RMB 2
T1 RMB 2
NOF1 RMB 1

ORG PROGRAM
START LDX #REGBAS
LDAA #$00010000
STAA TCTL2,X
LDAA #$00110000
STAA BAUD,X
LDAA #$00000000
STAA SCCR1,X
LDAA #$000001100
STAA SCCR2,X

BEGIN LDAA #$00
STAA NOF1

LABEL0 LDAA SCSR,X
ANDA #$00100000
BEQ LABEL0
LDAA SCDR,X
LDD TCNT,X
STD T0

LABEL1 LDAA TFLG2,X
ANDA #$10000000
BEQ LABEL2
INC NOF1
LDAA #$10000000
STAA TFLG2,X

LABEL2 LDAA TFLG1,X
ANDA #$00000100
BEQ LABEL1
LDD TIC1,X
STD T1
LDAA #$00000100
STAA TFLG1,X
BRA BEGIN
SWI
\end{verbatim}
EXECUTION

Open THRSim11. Close the Commands window. View CPU registers, timer registers, port A pins, memory list, serial transmitter. Open and assemble Ex_IC.asm. Set breakpoints at $c01d, $c029, $c036, and at 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) Press the RUN button. The program should loop on LABEL0. Type letter T into the serial transmitter and send. The program should exit the loop and stop at $c01d. At this point your screen looks like this:
b) Step. You will notice that the contents of SCDR ($54, i.e. ASCII for T) has been loaded into accA

c) Step again. The screen looks like this:

You notice that the contents of TCNT (in this case, $1955) has been loaded into accD. (The double accumulator accD is used, since the time values are in double precision, i.e., require 4 hex digits).
d) Step again.

You notice that the content of accD (for me, $1955, but you may get something else!) has been loaded in memory at location T0. It takes two consecutive memory locations, and is loaded as $19 followed by $55. At this stage, The timer count (TNCT) value was captured and stored into T0 as initial time.

e) Check for TOF. Switch display of accA to binary. Step. The value of TFLG2 (%01000000) is loaded into accA.

f) Step 2 times. The value in accA is AND-ed with the TOF mask (%10000000). Since TOF was not set, the result is zero and the BEQ operation jumps to LABEL2.

g) Check for IC1F. Step once. The value of TFLG1 (%00000000) is loaded into accA. Step 2 times. The value in accA is AND-ed with the IC1F mask (%00000100). Since IC1F was not set, the result is zero and the BEQ operation jumps to LABEL1.

h) You are back at LABEL1. Step through the TOF check and stop when you are at $c030

i) Toggle IC1 pin in Port A pins window from 0 to 1. Notice that, as you do so, bit 2 in TFLG1 gets also set. This bit is the IC1F bit, and it detects that a transition has taken place on line IC1.
j) Step. The value in TFLG1 (%00000100) gets loaded into accA. Step again. AccA is AND-ed with the IC1F mask (%00000100). The result is (%00000100), i.e., non-zero.

k) Step again. Since the result of the previous operation was non-zero, the BEQ operation does not branch, and you proceed to line $c036

l) Change display of accA to hex. Step. The value in TIC1 (for me, $1971, but you may get something else!) is loaded in accD (The double accumulator accD is needed, since the timer values are in double precision, i.e., four hex long)

m) Step again. The value in accD (for me, $1971, but you may have something else!) is stored as the captured time in T1. Note that two memory location are used, and the storage is done as $19 followed by $71

n) You will do this now on automatic. Press the Run button. Then, press the send button in the serial transmitter to start the input-capture process. Your program stops at the breakpoint $c01d.

o) Step. The TCNT value (mine is $cf13, yours may differ!) gets captured in accD.

p) Step two times. The TCNT value ends up in T0 (in my case, $cf13, yours may differ!).

q) Run. Toggle pin IC1 in port A window. As you toggle from 1 to 0, nothing happens, since you only set EDG1A=1 in the initialization section. This means that you programmed the MCU to only recognize raising edges on line IC1, i.e., transitions from 0 to 1.

r) Toggle again, now from 0 to 1. Note that the loop is exited, and the program stops at the breakpoint $c036.

s) Run again several times, and consolidate your understanding of the way the program runs. Try sending different characters, and notice that the program is only sensitive to you sending a character, but insensitive to what character you sent.

t) When you are content with your understanding, remove all breakpoint but SWI. Set a new breakpoint at $c03e. Run, go through the sending and the toggling, and, after it stops at the breakpoint, write below the values of T0, T1, NOF:

<table>
<thead>
<tr>
<th></th>
<th>My results</th>
<th>Your results</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>$49c8</td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td>$6227</td>
<td></td>
</tr>
<tr>
<td>NOF</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Calculate the time taken between your sending of the character (start of the process) and your toggling of IC1 from low to high (stop of the process).

$$\Delta t_{mine} = (T1-T0 + 10000*NOF)*0.5 \mu s = (6227 - 49c8 + 10000*0)*0.5 \mu s = 3119 \mu s = 3.1 \text{ ms}$$

$$\Delta t_{yours} = (T1-T0 + 10000*NOF)*0.5 \mu s = (???? - ????? + 10000*0)*0.5 \mu s = ??? \mu s = ??? \text{ ms}$$
u) Try to wait for the NOF to change to 1 before you toggle the IC1 pin. Write below the values of T0, T1, NOF:

<table>
<thead>
<tr>
<th></th>
<th>My results</th>
<th>Your results</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>$872e</td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td>$366e</td>
<td></td>
</tr>
<tr>
<td>NOF</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

\[(\Delta t)_{\text{mine}} = (T1-T0 + 10000*\text{NOF}) \times 0.5 \, \mu s = (366e - 872e + 10000*1) \times 0.5 \, \mu s = 55,200 \, \mu s = 55.2 \, ms\]

\[(\Delta t)_{\text{yours}} = (T1-T0 + 10000*\text{NOF}) \times 0.5 \, \mu s = (???? - ????? + 10000*1) \times 0.5 \, \mu s = ???? \, \mu s = ????.? \, ms\]

Note: The results from u) and t) do not need to agree. They depend on human factor, like when you decide to toggle the IC1 bit.

WHAT YOU HAVE LEARNED
In this example, you have learned:

- The use of MCU Timer function as an Input Capture (IC) device
- The use of the free running clock, TCNT, and its overflow flag, TOF
- The use of an input capture clock, TIC1, and its event flag, IC1F
- The selection of signal transition to be captured (here rising edge, EDG1A) and the fact that the MCU is only sensitive to that particular transition.
- The calculation of actual time in \(\mu s\) from the timer readings, T1, T0, and overflow count, NOF.