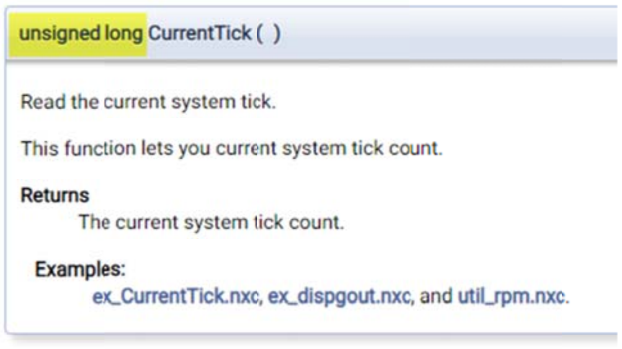
Hands-on Lab

Lego Programming – BricxCC Timers

Timing is important in controller design; data and actions often must be respectively acquired and commanded at prescribed intervals (i.e. sampling time). Introduced, is the NXC `CountTick()` function and serves as the foundation for measuring elapsed time. This is important for things like setting a sampling time.

Preamble

All microprocessors employ a crystal. Manufactures sell crystal that a specified by voltage and frequency. By applying the specified voltage, the crystal will vibrate at the specified frequency. Computer languages often provide functions to poll the number of times the crystal has vibrated since the voltage was applied. In NXC, that statement is called `CountTick()`. From the NXC help, one sees:



unsigned long CurrentTick ( )

Read the current system tick.

This function lets you current system tick count.

Returns
The current system tick count.

Examples:
ex_CurrentTick.nxc, ex_dispgout.nxc, and util_rpm.nxc.

In other words, calling `CurrentTick()` returns a 32-bit number. That number does not represent the time that's elapsed. Rather, it's the crystal's current tick value. One has to call `CurrentTick()` once more, and compute the difference between the second and first call. This difference reflects the number of milliseconds that has elapsed between the two calls. A 32-bit timer can measure quite a long time:

$$2^{32} = 4294967296 \text{ msec} = 49 \text{ Days}, 17 \text{ hours}, 2 \text{ min}, 47 \text{ seconds, and } 296 \text{ msec}$$

It is unlikely one would keep one's NXT Brick on for over 49 days. If one did, `CountTick()` will faithfully poll the crystal. Once the timer surpasses 2^{32}msec, it will restart the count from zero.

Concept 1: A simple stopwatch using `CountTick()`

Step 1: Type **`stopWatch1_0a.nxc`**, save, compile and execute

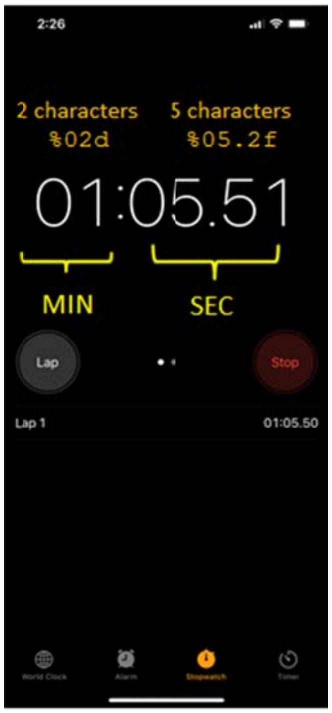
```
// FILE: stopWatch1_0a.nxc - Works!
// DATE: 03/13/23 11:46
// AUTH: P.Oh
// DESC: Display seconds elapsed
// VERS: 1_0a - release version for Spring 2023 ME 425/625
// REFS: mtrSpeed0_2a5.nxc; stopWatch0_1.nxc

task main() {

  // Declare variable ----------------------------------------------------

  // Button related variables
  bool orangeButtonPushed, rightArrowButtonPushed;

  // Timing related variables
  long ticPrev, ticCurr, ticDelta;    // previous, current and delta ticks
  float elapsedSeconds;               // seconds elapsed

  // Initialize variables --------------------------------------------------
  elapsedSeconds = 0.0; // set elapsed time to zero

  // Algorithm starts here ---------------------------------------------------
  // (1) Prompt user to begin stopwatch
  TextOut (0, LCD_LINE1, "-> starts" );
  do { // wait until user hits right button
   rightArrowButtonPushed = ButtonPressed(BTNRIGHT, FALSE);
  } while(!rightArrowButtonPushed);
  ClearScreen();
  TextOut (0, LCD_LINE1, "Orange Btn quits" );
  TextOut (0, LCD_LINE2, "Time = " );

  // (2) User started stop watch
  ticPrev = CurrentTick(); // <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
  do {
    // (2A) Poll timer with second CurrentTick
    ticCurr = CurrentTick(); // read timer value <<<<<<<<<<<<<<<<<<<<<<<<<<<<<
    // (2B) Difference in CurrentTick values is elapsed milliseconds
    // Take sum to of elapsed milliseconds to calculate total time elapsed
    // Format as a string so value can be displayed on Brick
    ticDelta = ticCurr - ticPrev;  // difference in ticks [msec]
    elapsedSeconds = elapsedSeconds + (ticDelta/1000.0); // elapsed time [sec]
    TextOut(0, LCD_LINE6, FormatNum("%5.2f s" , elapsedSeconds));
    // (2C) make previous tick value now equal to last read value i.e. ticCurr
    ticPrev = ticCurr;
    // Check if user wants to quit
    orangeButtonPushed = ButtonPressed(BTNCENTER, FALSE);
  } while( !orangeButtonPushed );

  // Orange button pressed, so quit
       TextOut(0, LCD_LINE2, "Quitting", false);
  PlaySound(SOUND_LOW_BEEP);  // Beep to signal quitting
       Wait(SEC_2);
  StopAllTasks();
} // end main
```

Program: `stopWatch1_0a.nxc`

Code Explanation: A typical stop watch (e.g. on one's smart phone) displays time as a real number e.g. 1.99 means 1/100th of second shy of 2 seconds. Here, the do-while loop computes the difference between `CountTick` values (3rd yellow highlight) and displays the elapsed seconds on the Brick. Here the string specifier `%5.2f` is used to yield 2 digits past the decimal.

Step (2) first polls the crystal (1st yellow highlight), just before entering the `do-while` loop. Step (2A) immediately polls the crystal a second time (2nd yellow highlight). Step (2B) computes the difference between the second and first polls (3rd yellow highlight). That difference `ticDelta`, represents the number of milliseconds that elapsed between the two polls. Before looping back, Step (2C) assigns the 2nd tick value call, now the 1st tick call. The net effect is that number of elapsed seconds is displayed on the Brick.

Concept 2: A stopwatch that looks more like the one on my phone



On the left is an annotated screenshot of an iPhone's stopwatch app. The format is MM:SS.XX where MM is a 2-digit number for minutes. This means it will increment from 0 to 59 minutes. SS.XX is real number that represents the number of seconds. SS will increment from 0 to 59 seconds. XX denotes 10th of a second and increments from 00 to 99.

Mimicking this stopwatch on the Brick requires using string format specifiers. These specify how one wishes the numbers to be displayed.

For example, the figure shows that zeros are used for padding. Here, one sees "01" and not "1" or " 1". A zero is used instead of a white space or a single digit. Likewise, one sees "05" instead of "5" or " 5".

Lastly, the figure shows a colon to separate the minutes from seconds and the seconds are real numbers i.e. up to 2 digits following the decimal point.

Step 1: Write an NXC program that mimics a smart phone stopwatch as seen above. Write **`stopWatch1_0b.nxc`**, save, compile and execute.

```
// FILE: stopWatch1_0b.nxc - Works!
// DATE: 03/13/23 12:39
// AUTH: P.Oh
// DESC: Mimic phone timer.  Display seconds elapsed
// VERS: 1.0a: Displays time elapsed
//       1.0b: Displays time in 00:00.00 format with leading zeros
//       Release version for Spring 2023 ME 425/625
// REFS: mtrSpeed0_2a5.nxc, stopWatch1_0a.nxc, stopWatch0_2a.nxc

task main() {

  // Declare variable ------------------------------------------------------

  // Button related variables
  bool orangeButtonPushed, rightArrowButtonPushed;

  // Timing related variables
  long ticPrev, ticCurr, ticDelta;    // previous, current and delta ticks
  float elapsedSeconds;               // elapsed seconds e.g. 1.23 seconds
  string strElapsedSeconds;           // string form of elapsedSeconds
  int elapsedMinutes;                 // elapsed minuters e.g. 59 minutes
  string strElapsedMinutes;           // string form of elapsedMinutes
  string strDisplayTime;              // string to display 01:23.45 format
```

```
   // Initialize variables -------------------------------------------------
   elapsedSeconds = 0.0;
   elapsedMinutes = 0;

   // Algorithm starts here ---------------------------------------------------
   // (1) Prompt user to begin stopwatch
   TextOut (0, LCD_LINE1, "-> starts" );
   do { // wait until user hits right button
    rightArrowButtonPushed = ButtonPressed(BTNRIGHT, FALSE);
   } while(!rightArrowButtonPushed);
   ClearScreen();
   TextOut (0, LCD_LINE1, "Orange Btn quits" );
   TextOut (0, LCD_LINE2, "Time = " );

   // (2) User started stop watch
   ticPrev = CurrentTick(); // <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
   do {
     ticCurr = CurrentTick(); // read timer value <<<<<<<<<<<<<<<<<<<<<<<<<<<<<
     ticDelta = ticCurr - ticPrev; // computes difference in ticks (i.e. msec)
     elapsedSeconds = elapsedSeconds + (ticDelta/1000.0); // [sec]

     // (3) Calculate elapsed minutes
     if(elapsedSeconds > 60.0) { // seconds > 60 seconds means a minute passed
       elapsedMinutes = elapsedMinutes + 1; // increment number of minutes
       elapsedSeconds = 0.0; // reset elapsed number of seoonds to 0
     }; // end if

     // (4) Brick display e.g. 01:23.45 means 1 minute, 23.45 seconds
     strElapsedMinutes = FormatNum("%02d" , elapsedMinutes); // format to 00
     // (4A) The max number of minutes (before increment an hour) is 59.
     // Hence %02d means to allot at least 2 spaces to be displayed. This
     // results in display 01..09 (for 1 to 9 minute) or 10...59 (for 10 to
     // 59 minutes)

     strElapsedSeconds = FormatNum("%05.2f" , elapsedSeconds); // format to 00.00
     // (4B) 9.99 means 9 seconds and 99/100th of a second.  We want to display
     // as 09.99 i.e. pad with a zero for single digits (1...9).  We also count
     // 09.99 has 5 characters.  So using %05.2f pads with 0 and allots 5
     // white spaces to be populated with seconds and 100ths of second

     strDisplayTime = StrCat(strElapsedMinutes, ":" , strElapsedSeconds);
     // (4C) the ":" inserts the colon in stopwatch format e.g. 01:23.45
     TextOut(0, LCD_LINE4, strDisplayTime);

     // (5) Reset tick count variables before looping back
     ticPrev = ticCurr;
     // Check if user wants to quit
     orangeButtonPushed = ButtonPressed(BTNCENTER, FALSE);
   } while( !orangeButtonPushed );

   // (6) Orange button pressed, so quit
        TextOut(0, LCD_LINE2, "Quitting", false);
   PlaySound(SOUND_LOW_BEEP);  // Beep to signal quitting
        Wait(SEC_5); // give enough time for user to view Brick before program exits
   StopAllTasks();
 } // end main
```

Program: `stopWatch1_0b.nxc`

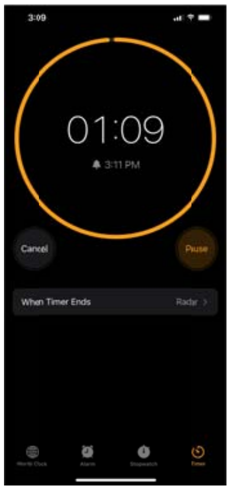
Code Explanation: The program is very similar to Concept 1, using a do-while loop and polling `CountTick()` twice. **`stopWatch1_0b.nxc`** introduces the variable `elapsedMinutes` that increments by one every time `elapsedSeconds` increments past 59 as shown in the `if`-statement in Step (3).

Mimicking the smart phone display, Step (4) provides details. The 1st yellow highlight creates a string that is 2 characters wide by using the "`%02d`" specifier. The `0` in the specifier means to pad the 2-character value with a zero if the value is only 1 character wide.

Likewise, the 2nd yellow highlight creates a string that is 5 characters wide. Recall, SS.XX involves two characters for the seconds, plus the decimal point, plus two characters for the tenths of a second. Thus 2 + 1 + 2 = 5 character spacing is needed. Recall that `elapsedSeconds` is a float. Also recall one uses a `0` in the string format specifier to pad. The net effect is to employ "`%05.2f`" where `0` is for padding with zeros, `5` is for 5 characters spaces, and `2f` is a float with 2-number precision i.e. 2 digits after the decimal point.

The 3rd yellow highlight constructs the final string to be displayed on the Brick. A StrCat is used. One sees that the colon character concatenates in-between the two strings `strElapsedMinutes` and `strElapsedSeconds`.

Exercise 1: In NxC create programs for the following:



On the left is a screenshot of a smartphone timer. One sets a time to count down from e.g. 1 minute and 10 seconds. Once the timer reaches zero, a sound is played.

1-1 A timer that counts down from 1 minute and 10 seconds and plays a sound once it reaches zero. The Brick should display as MM:SS and pads with zeros any single digit values.