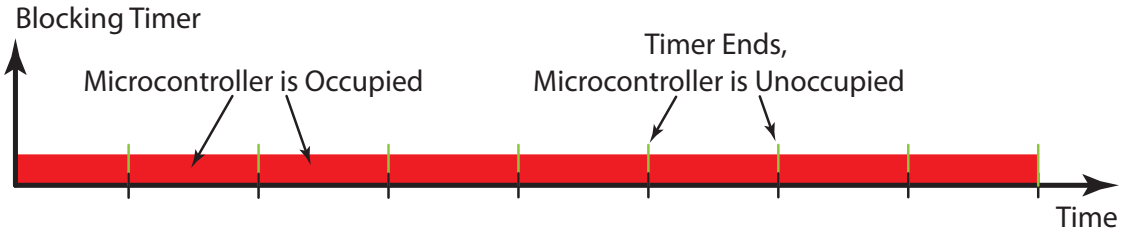
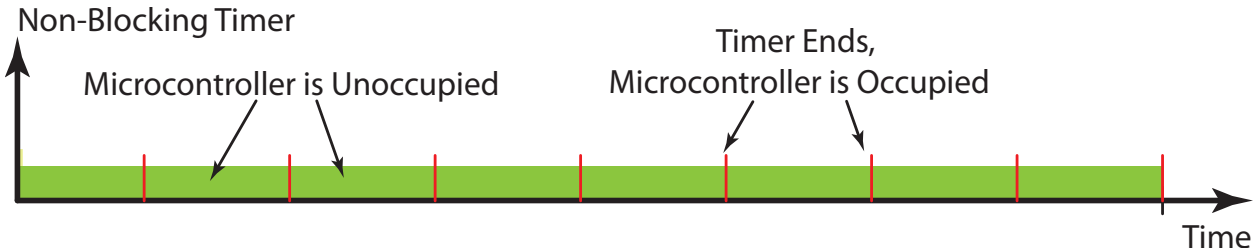
Stopwatch Tutorial
Version 1.0
Malcolm Knapp
Workshop Weekend: Arduino
February 8th and 9th 2014

1. **Introduction**: This tutorial is focused on creating a stopwatch using an Arduino. We will use two buttons as inputs and an LCD as an output. We will start with a simple 1s timer. Then we will and start/pause using one of the buttons. Finally we will add a reset using the other button.
   1.1. For this tutorial we will use two buttons as input and an LCD as an output.
   1.2. This tutorial assumes that you know how to do the following. If you do not, please do those tutorials first before continuing with this one.
      1.2.1. Add a library to Arduino
      1.2.2. Hook up an LCD to the Uno.
      1.2.3. Hook up a button to the Uno
   1.3. All code is called out in **bold,** all lines that you need to change are called out in **red bold** and lines you need to add are in **green bold.**
   1.4. At various points you will have to insert code. The insertion points are marked with a with a number in a comment (e.g //-1-).

2. **Background**
   2.1. This project that looks like it could be done with the delay() function but this will not work because delay is what is known as a **blocking timer**. The reason for this is that a blocking timer occupies the microcontroller while it is running. This means that any button presses that happen when the microcontroller is occupied will not be registered. Only if you are lucky enough to press the button at the same moment when the timer ends will you get a response.



To solve this problem we will use a **non-blocking timer**. This code structure only occupies the microcontroller when it ends. This leaves the microcontroller available to register the button press whenever it happens.



Also every time the timer ends there is an “event” that happens. This event can be anything from turning on an LED, to sending a message, to turning a motor on or off.

3. **Getting Started.**
   3.1. Download the Stopwatch Tutorial 1V0 zip file from the Workshop Weekend: Arduino website.
   3.2. Unzip it. Within that folder you should see
      3.2.1. The Stopwatch Tutorial 1V0.pdf. This is the tutorial document that you are reading right now.
      3.2.2. The Stopwatch Start code. The is the code we will be modifying.
      3.2.3. Stopwatch Wiring Diagram. This is a Fritzing Diagram that shows you how to hook up the circuit.

4. **Hook up the circuit.** See the Stopwatch Wiring Diagram for details
   4.1. Hook up the LCD
   4.2. Hook up a button to the Uno, pin 6
   4.3. Hook up a button to the Uno, pin 7

5. **Add the needed libraries.** This tutorial uses two libraries.
      5.0.1. The first is the Liquid Crystal Library which drives the LCD. It is built into Arduino so you do not need to do anything. Yay!
      5.0.2. The second is the Button Library. It makes reading the button easier. You can download it from the Workshop Weekend:Arduino website.

6. **Testing the circuits.** I always use the example code provided with the library. That way you have known working code testing your circuit. If you get stuck ask for help.
   6.1. To test the button
      6.1.1. Open File --> Examples --> Button --> Examples --> UniquePress
      6.1.2. To match the way we hooked up the button we need to change following line from:

      Button button = Button(12, BUTTON_PULLUP_INTERNAL);

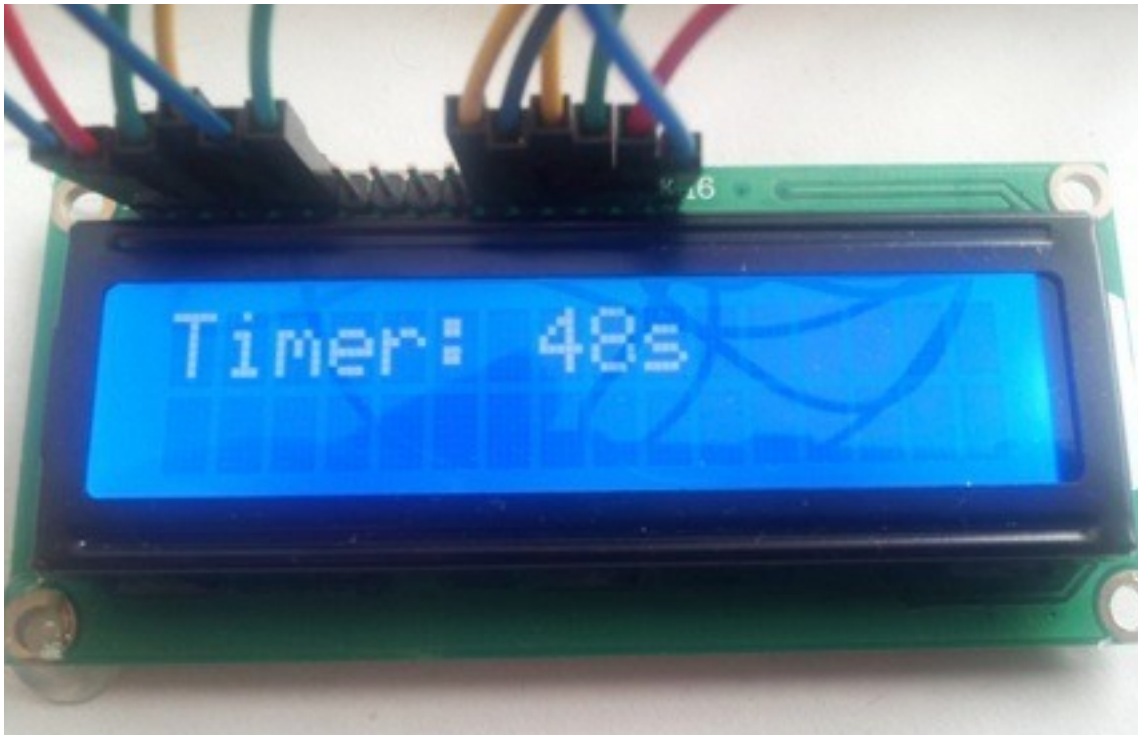
      to:

      Button button = Button(6, BUTTON_PULLUP_INTERNAL);

      6.1.3. Up load the code
      6.1.4. Click on the Serial Monitor in the upper right hand corner of the Arduino IDE.
      6.1.5. In the window that appears, confirm that the baud rate in the lower right hand corner is set to 9600. If it shows something different select 9600 from the menu.
      6.1.6. Confirm that you get a message every time the button is pressed.
   6.2. To test the LCD

6.2.1. Open File --> Examples --> Liquid Crystal --> Hello World
6.2.2. Upload Code
6.2.3. Adjust Contrast potentiometer so you can see the text
6.2.4. Confirm you see “Hello World” in line 1 and a counting second timer in the second line

7. **Our first Timer.** Now that we have confirmed everything is working correctly we get to start timing things!
   7.1. Open Stopwatch_Start.ino
   7.2. Save it with our own name.
   7.3. Load the code and you should see the word Timer on the LCD and it should be counting up the seconds. The LCD screen should look like this:



   7.4. Let’s take a closer look at the parts of a non-blocking timer. There are three parts to every non-blocking timer. We are going to ignore the button for a moment but we will come back to it very soon.
      7.4.1. Duration Variable: This determines how long the timer runs for

unsigned long timerInterval = 1000; // in milliseconds

For each non-blocking timer you create you will need a unique duration variable.
      7.4.2. Last start time Variable. This stores the last time when the timer started

```
unsigned long  timerLastStart = 0;  // in milliseconds
```

For each non-blocking timer you create you will need a unique last start time variable.

7.5. Timer End Check. This checks whether the timer has run out or not.

```
if(millis() - timerLastStart > timerInterval) {
        timerLastStart = millis();
      // Event Code
}
```

For each non-blocking timer you create you will need a unique time elapsed check variable. Any Event code put after timerLastStart reset defines what happens when the timer ends. NOTE: For a detailed explanation of the logic see appendix, section 12

8. Now you get to make your own timer for the stop watch.
   8.1. Add a new duration variable at point -1-

```
//-1-
unsigned long stopwatchInterval = 1000; // in milliseconds
```

   8.2. Copy and paste the line at point -2-

```
//-2-
unsigned long stopwatchLastStart = 0;
```

   8.3. Add a new count variable below the current one at point -3-

```
//-3-
int secondCount2 = 0;
```

   8.4. Copy and paste the Timer End Check code and paste it below the first timer below point -4-

```
//-4-
if(millis() - timerLastStart > timerInterval) {
        timerLastStart = millis();
        // event code
        secondCount1++;
        lcd.setCursor(0, 0);
        lcd.print("Timer: ");
        lcd.print(secondCount1);
        lcd.print ("s");
}
```

8.5. In the code you just inserted, update the duration and last start variables in the code as shown below.

```
if(millis() - stopwatchLastStart > stopwatchInterval) {
    stopwatchLastStart = millis();
```

8.6. Then update the count variable to the new name.

```
secondCount1++;
```

to

```
secondCount2++;
```

8.7. Now we need to make it display correctly so we need to change the line it displays on.

```
lcd.setCursor(0, 0);
```

to

```
lcd.setCursor(0, 1);
```

8.8. And we will change the line

```
lcd.print("Timer:");
```

to

```
lcd.print("Stopwatch:");
```

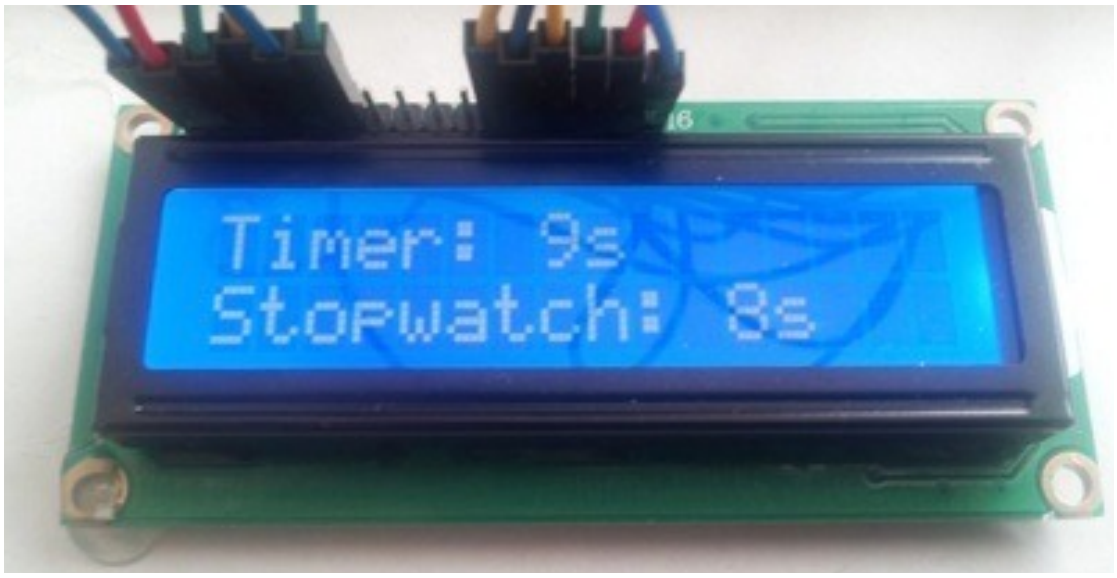
8.9. Finally we need to change the line

```
lcd.print(secondCount1);
```

to

```
lcd.print(secondCount2);
```

8.10. Compile the code to make sure there are no errors
8.11. Upload the code
8.12. The LCD should now look like this.



8.13. If you it does not work this way try and debug it on your own and then if it still does not work ask for help.

9. The next thing to do is to add the start functionality to the code.

9.1. To do this we need to add an enable to the stopwatch timer. Add the following code in the variable initialization section at point 5

```
//-5-
boolean isStopwatchOn = false;
```

9.2. Then update stopwatch End Check check from

```
if(millis() - stopwatchLastStart > stopwatchInterval) {
```

to

```
if(millis() - stopwatchLastStart > stopwatchInterval && isStopwatchOn) {
```

9.3. Then we need to add the logic that starts the timer when the timer is on or pauses it when the timer is on. So add the code below to the button press logic below the following line

```
if (start_button.uniquePress ()) {
        if (isStopwatchOn) {
                isStopwatchOn = false;
        } else {
                isStopwatchOn = true;
        }
}
```

9.4. This logic will start the timer when the button is first pressed and pause it when the button is pressed again.

9.5. Then we need to make sure the LCD is displaying clearly so add the following code at point -6-

```
//-6
lcd.setCursor(0, 1);
lcd.print("Stopwatch Off");
```

9.6. Load the code and you should see

9.7. Now press the button. and you should see

10. The last functionality we are going to add is a timer reset.
    10.1. To begin add a new pin defined at point -7-

```
//-7-
#define RESET_BUTTON_PIN 7
```

    10.2. Then initialize the button

```
//-8-
Button reset_button = Button(RESET_BUTTON_PIN, BUTTON_PULLUP);
```

    10.3. Add the button press check below point -9-

```
//-9-
if (reset_ button.uniquePress ()) {
}
```

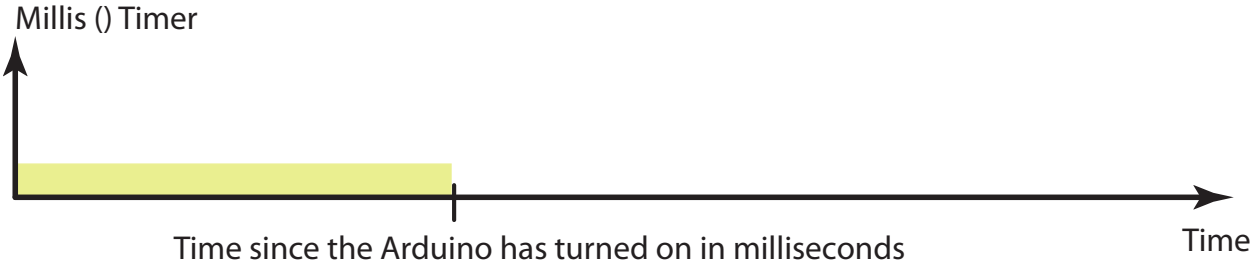
    10.4. Finally we need to add the code that will execute when the button is pressed. In this case we need to reset the stopwatch timer to zero and display that the timer has been reset.

```
if (reset_ button.uniquePress ()) {
        Serial.println ("Timer reset");
        secondCount2 = 0;
        isStopwatchOn = false;
        lcd.setCursor(0,1);
        lcd.print ("Stopwatch Off");

}
```

10.5. Compile the code and make sure there are no errors
10.6. Load the code
10.7. You should see the the stopwatch start when you press the start button and reset when you press the reset button.
10.8. If you it does not work this way try and debug it on your own and then if it still does not work ask for help.
10.9. If everything works then, YAY!!, you are DONE!. You now have an Arduino Stopwatch

11. If you get this far and still want to play some more, here are a few small projects you can try.
    11.1. Make the stopwatch into a countdown timer
    11.2. Make two timers that start each other. This is a great way to make a LED blink with unequal on and off periods.
    11.3. After you create a few timers you will start creating a lot of duplicate code. Can you figure a way to not repeat yourself?. Arrays and function calls will be useful for this.
12. Appendix
    12.1. Non-Blocking Timer Code Logic
        12.1.1. The non-blocking timer is based on the millis() function. This function uses a timer that is built into the circuitry of the microcontroller (also known as a hardware timer) and it keeps track of how many milliseconds have elapsed since the microcontroller was turned on. Since this a hardware timer it can run while the program is doing other things.

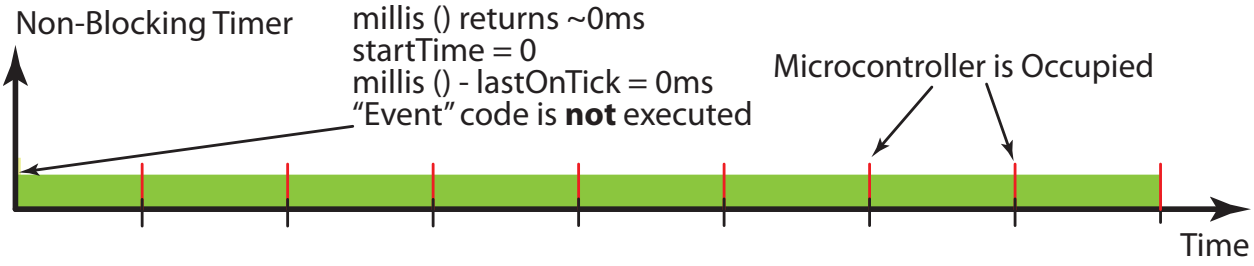


        12.1.2. The logic of the non-blocking timer is centered around this code

```
if(millis() - timerLastStart > timerInterval) {
        timerLastStart = millis();
        //Event code
}
```

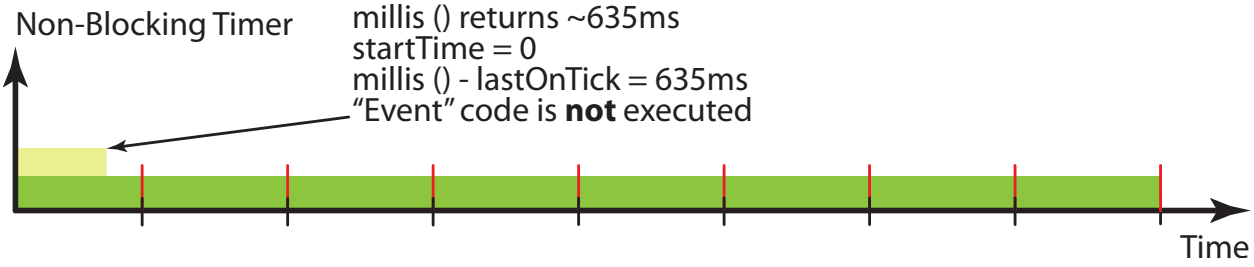
        To see how this code works, let's go through an example. Let us assume that timer Interval is 1000 (or 1s).
        12.1.3. The code starts. At this moment millis () on 0 and timerlastStart is 0 so the IF statement evaluates to false and the Event code within it does not execute.

```
if(0 - 0 > 1000) {
        timerLastStart = millis();
        //Event code
}
```



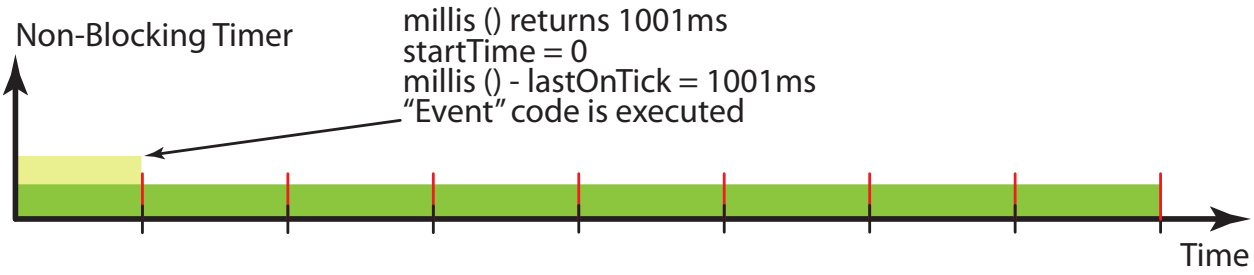
12.1.4. After 635ms the situation is still the same. (635 - 0) is still less the 1000 so that section of the code is still not going to execute.

```
if(635 - 0 > 1000) {
        timerLastStart = millis();
        //Event code does not execute
}
```



12.1.5. But let's now check after 1001ms. Now the situation is different (1001-0) **is** greater than 1000 so the code **is** run. The first thing that happens is that the timerLastStart is set to the current value of millis() or 1001. This step is necessary for the timer to run correctly. Then the event code within the timer will execute.

```
if(1001 - 0 > 1000) {
        timerLastStart = millis();
        //Event code does execute!
}
```



12.1.6. The final thing to look at is some time later. Lets check after 1050ms total elapsed time. We are back to the original situation. (1050-1001) is again less than 1000 and the IF statement evaluates to false again.

```
if(1050 - 1001 > 1000) {
        timerLastStart = millis();
        //Event code does not execute
}
```