**Calculations For Generating 1s Delay Using TMR1**

***Tout =***

**Where:**

*Tout* = desired output time interval

*PS* = PRESCALER value

*Fosc* = frequency of local oscillator (if Timer1 is configured to operate in timer mode)

*TMR1 =* initial value we start counting from (usually 0 but this is not always the case)

*X* = required number of Timer1 overflows

After switching the Timer1 module ON, the timer will start incrementing once per instruction cycle. Which means the time of each Tick is calculated as follows:

*Tick Time****=****1 / (Fosc/4) = 1/(4000000/4) = 1 us*

Assuming that our MCU is running @ 4MHz

Time it takes our Timer1 module to reach the overflow state and wraps-around to start counting again from 0. The formula is as follows

***T****overflow****=****# of Ticks****x****Tick time*

The number of ticks is strongly dependent on the TIMER1 resolution which in this case is 16-bit. Therefore:

*# of Ticks = - 1 = 65535*

Therefore:

***T****overflow****=****# of Ticks****x****Tick time = (65535) \* (1x) = 0.065535 s 65ms approximately.*

Using Timer1 general formula in order to generate 1 s time interval with the following values chosen for parameters:

*Tout* = 1s

*PS* = 1:1 or 1

*Fosc = 4 MHz*

*TMR = 0 start counting value*

*65535 – indicates that our timer1 module has 16 bit resolution*

*X – required no. timer of timer overflows*

Substituting our predefined value and solving for X we get the following result:

X = 15.26 because this a floating point number then it is arbitrarily chopped to

X = 15

Proof: That this 1s time interval is not 100%:

Output Time = Number of Overflows \* Time of single overflow

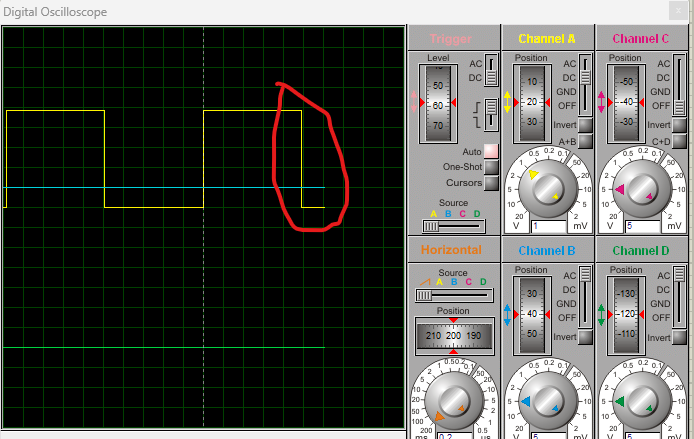
*Tout = X \** ***T****overflow  = (15)(0.06553) = 0.0983 s (this is not accurate)*

Time Error = 1s – 0.0983 = 0.02 sec

Error percentage = 2 %

Although this error is seemingly small, but it’s an accumulating error that propagates from the 1st sec. to the 2nd, 3rd, and so on. If you’ve used the timer1 module in the exact same configurations in order to fire an alarm after 12hours. The alarm will get fired after 11.75h. The small 0.02 sec error has grown to 15min over time.

What if my application is critically time-dependent? What if we need to generate a pure 100% accurate time interval? Well, we’ll do what I call **Timer Pre-Loading**!



If you carefully observe the above figure, there is a tiny error in the output period. It’s just less than 1-second by a tiny portion. Mathematically, this error can be easily calculated by performing the following multiplication

Output Time = Number of Overflows \* Time of single overflow

*Tout = X \** ***T****overflow*

**Timer Preloading Technique**

The source of error prior to timer preloading technique was chopping the floating point number 15.26 to the whole integer 15.

The basic idea of this technique is to avoid getting a floating number as a value for the number of overflows (**X**). How can we achieve this?

By substituting for the number of overflows(**X**) by 1, you’ll get the Toverflow

Note that: For (X=1) ==>  Tout = Toverflow

***Tout =***

The result will be Toverflow = 0.06553 second

if that number is the time it takes our timer1 module to reach overflow state. Then how many overflow interrupts do we need to complete a 1-second time interval?

Since 1 / 0.06553 results in **15.26** overflows

This means that Toverflow needs to be adjusted.

Well, if it takes the timer1 module about (0.06553 seconds) to count from (0 to 65535). What if we could force it to overflow in less time? if we chose a period of (0.05 second) which is less than 0.06553 and more importantly won’t generate a floating number when we divide 1second/0.05second = 20. Which means that we’ll exactly need clean 20 overflows to get our desired 1-second output interval. There won’t be any remainder fractions and so

 The solution for this problem is to force the timer1 module to overflow in just 0.05 second instead of 0.06553?

Well, if counting from (0 to 65535) takes 0.06553 seconds. Then if it started counting from a specific value in between, then it can theoretically overflow nearly at any instance of time we wish!

By using the general equation, setting (Tout = 1, X = 20, Fosc= 4MHz, PS = 1), and solving for TMR1. We’ll get the initial value to be preloaded in timer1 module. This will result in

TMR1 = **15535**

And that is the value we should preload to the TMR1 register in order to make (Toverflow = 0.05sec). Which means that a Tout of 1-second will need clear 20 overflows!

That’s the basic idea behind the timer preloading technique. We’re changing the time till overflow Toverflow by loading a specific value in the TMR1 register at the beginning of counting, instead of starting at zero as usual!