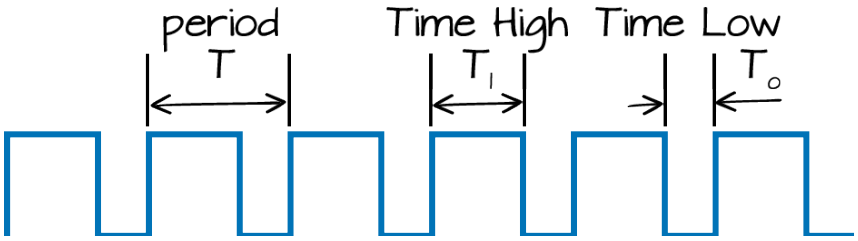
Zen & the Art of External Watchdogs

A Watchdog is there to make sure that an unexpected program execution error is handled; be that error an infinite loop, a panic to console or processor seizure.

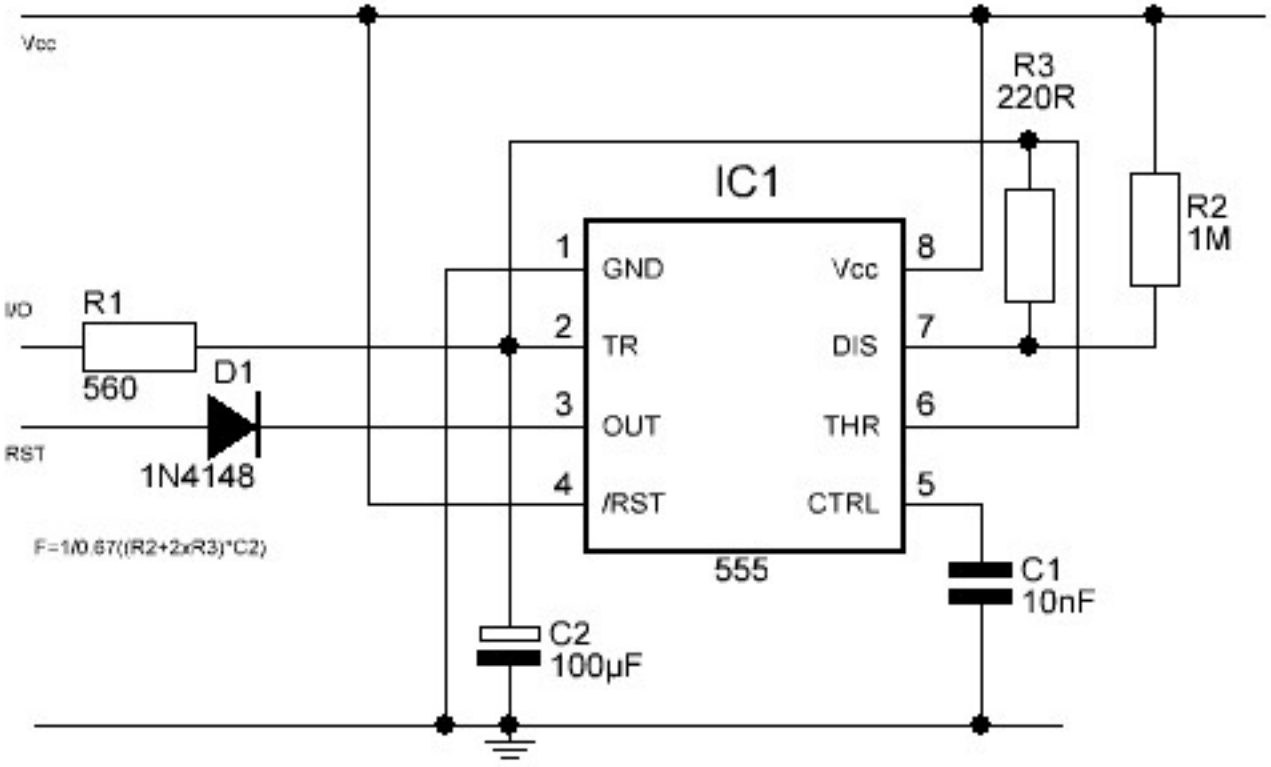
Caveat Emptor - A Watchdog will not recover from an equipment failure, either in the Watchdog itself or in the equipment it is watching or any errors you make. Also be aware that with an external watchdog, escaping to the console will remove the Pat the Dog and reset the Pico after the watchdog times out. You have been warned – disable the watchdog if you're working on the console and remember to enable it again when you're finished. More easily said than done ☺.

In order to placate the Watchdog one needs to Pat the Dog. In an embedded controller setting this would be through a regular GPIO being driven temporarily to a state which Pats the Dog to prevent it barking. The barking will reset the embedded processor in some way resulting in a recovery from its non-responsive condition.

What one needs is an astable multivibrator, which is to say having a waveform where the reset pulse is of short duration when compared to the balance of the waveform. On the basis of the reset pulse being a low level pulse, the high pulse duration will be >99% while the low level reset pulse will be very much shorter. Something like this [from https://ohmslawcalculator.com/555-astable-calculator]

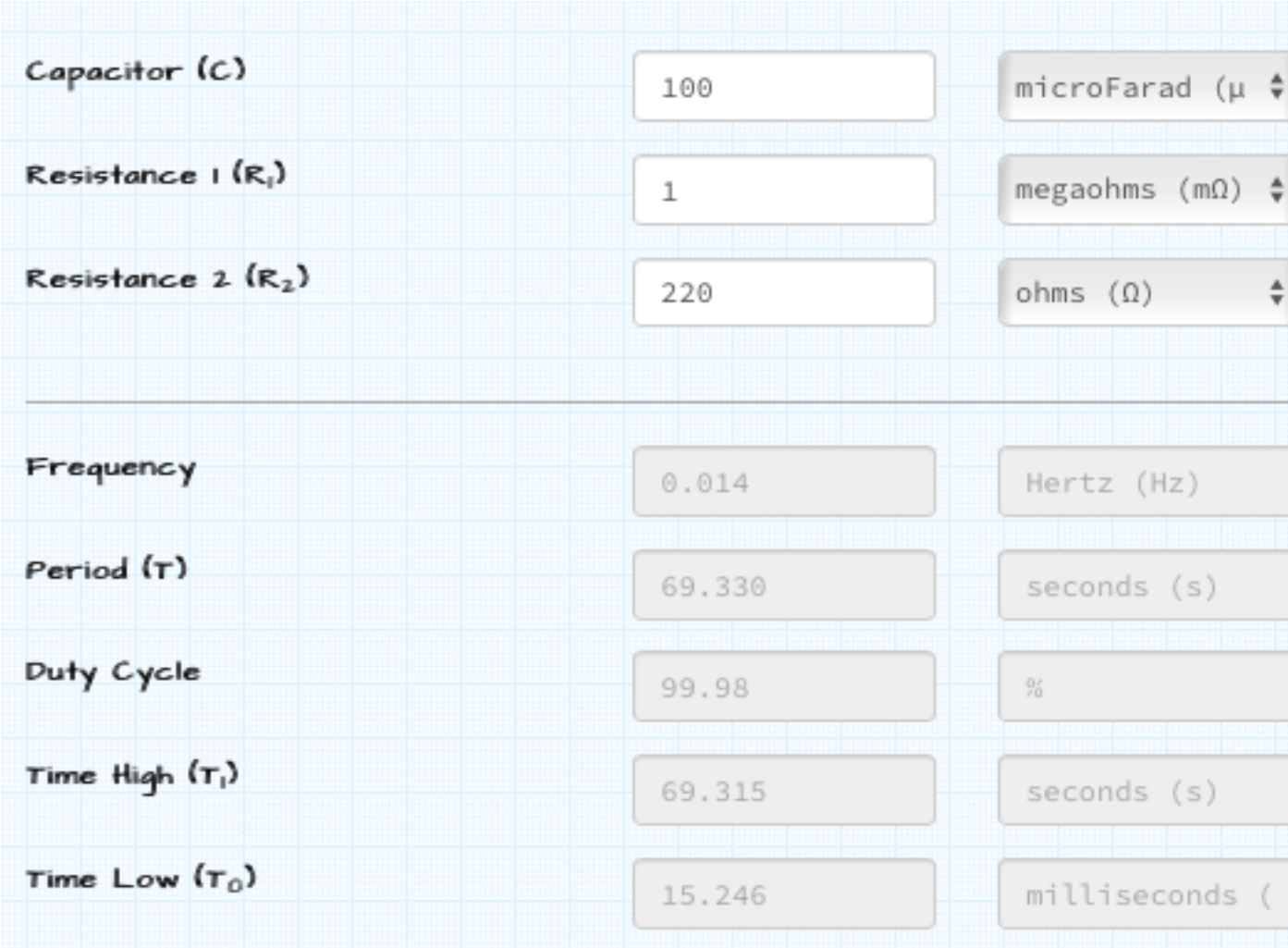


In the example circuit below [from https://arduinodiy.wordpress.com/2017/09/08/arduino-watchdog-timer/], given that the Dog has not been Patted, the high level period is of the order of 69s [https://ohmslawcalculator.com/555-astable-calculator] and the low level reset pulse of the order of 38ms. Bearing in mind the examples given so far relate to the Arduino running on a 5V rail.



The [https://ohmslawcalculator.com/555-astable-calculator] 555 astable calculator provides a convenient place to fiddle with the values.

For the Pico I'm using R1=330Ω, R2=1MΩ, R3=220Ω and C2=100µF. In the calculator this yields the following (noting that for the calculator C=C2=100µF, R1=R2=1MΩ and R2=R3=220Ω)



So from the calculator we have a time to reset of 69.315s and a reset pulse of 15.246ms.

In terms of the operation of the Pico the current sinking capability of ALL the GPIO combined appears to be 50mA [https://forums.raspberrypi.com/viewtopic.php?t=300735&sid=576b934db443f1ebb5905ca008dd2e29&start=25]. In the Pico example of the circuit above R1=330Ω, which according to Ohms Law will result in the GPIO pin having to carry the C2 discharge current of 3.3V/330Ω=11mA (instantaneous values) which is well within the 50mA across ALL the GPIO. However your GPIO usage may be different.

The Pico clocked at 250MHz gives a 20-30µs low pulse [Thanks TassyJim https://www.thebackshed.com/forum/ViewTopic.php?PID=209128#209128#209132] with the following code snippet below;

```
PIN(GP13) = 0
DO
   SETPIN GP13,DOUT
   SETPIN GP13,DIN
   PAUSE 1
```

LOOP

Consider the workings of the 555 timer in as far as the Trigger pin needs to rise above 2/3^{rds} Vcc in order for the astable output (pin 3) to toggle from its high state to a low state to provide the low reset pulse, which is connected to the Pico RUN pin.

So the concept now is that the Pico GPIO is Patting the Dog as fast as it can, which prevents the voltage from exceeding 2/3^{rds} Vcc…but does it??? To discharge/keep C2 below 2/3^{rds} Vcc (2.2V assuming 3.3V VSYS) the Pat has to discharge C2 to keep it below 2/3^{rds} Vcc.

The most convenient approach is to look at the current flows through the charge and discharge loops. These can be determined from the following formulae, assuming a static approach when Ohm's Law is applied;

Icharge = Vcc/(R2+R3); and [1]
Idischarge = Vtrigger/R1 [2]

Where Vcc=3.3V, Vtrigger=2.2V, R1=330Ω, R2=1MΩ and R3=220Ω we get

Icharge=3.3V/(1MΩ+220Ω)=3.2993μA; and [3]
Idiscarge=2.2V/330Ω=0.0067A [4]

So from first principles we need a small portion of Idischarge to offset the Icharge portion. Assuming that Icharge occurs in 1 unit of time, then we need 3.2993μA/0.0067A=0.4949 milli units of discharge time.

Thus for 1s of Icharge we need to Idischarge for 0.4949ms. Let's round this to 0.5ms to facilitate the thinking below; bearing in mind that this is to achieve equilibrium and not to discharge C2 towards its lowest value.

Clearly the more we Pat the Dog the more headroom we will have in terms of the voltage rise across C2 approaching the 2/3^{rds} Vcc trigger voltage.

In the example circuit above lets change R2=1MΩ to a variable 1MΩ resistor and increase the Pause value to 500 in the code snippet, to simulate a scenario where the program is delayed in returning to Pat the Dog.

Now measure the voltage between the ground rail and the positive side of C2. We are aiming to increase the value of R2 such that the voltage on C2 does not exceed the 2.2V trigger voltage, through Patting the Dog in a loop for long enough (very quickly for a short duration Pat or less often with a longer duration Pat); understanding that it won't be zero but a value below 1volt measured with an ordinary multimeter.

Clearly the best strategy is the judicious application of Patting the Dog sprinkled throughout the code. By adjusting R2 we make the Icharge value less and give the shorter but higher Idischarge the opportunity to discharge C2 faster than Icharge can

charge C2.

My coding practice is to make everything sub routines and wherever relevant interrupt driven and reduce the Do Loop to an empty shell. With an external watchdog insert the Pat the Dog sub routine call in the Do Loop as a minimum. Consider Patting the Dog just before any long duration pieces of code, especially anything related to the WEB Commands, just to be safe. Run the program and monitor the voltage across C2. Increase R2 until you achieve the lowest possible voltage across C2 if necessary.

Alternatively, using the circuit provided above, the watchdog timeout can be just over 60s and then live with the longer delay to reset; which will provide the greatest tolerance to strange code behavior.

Best of luck
Carl 2020907 [rev 0]