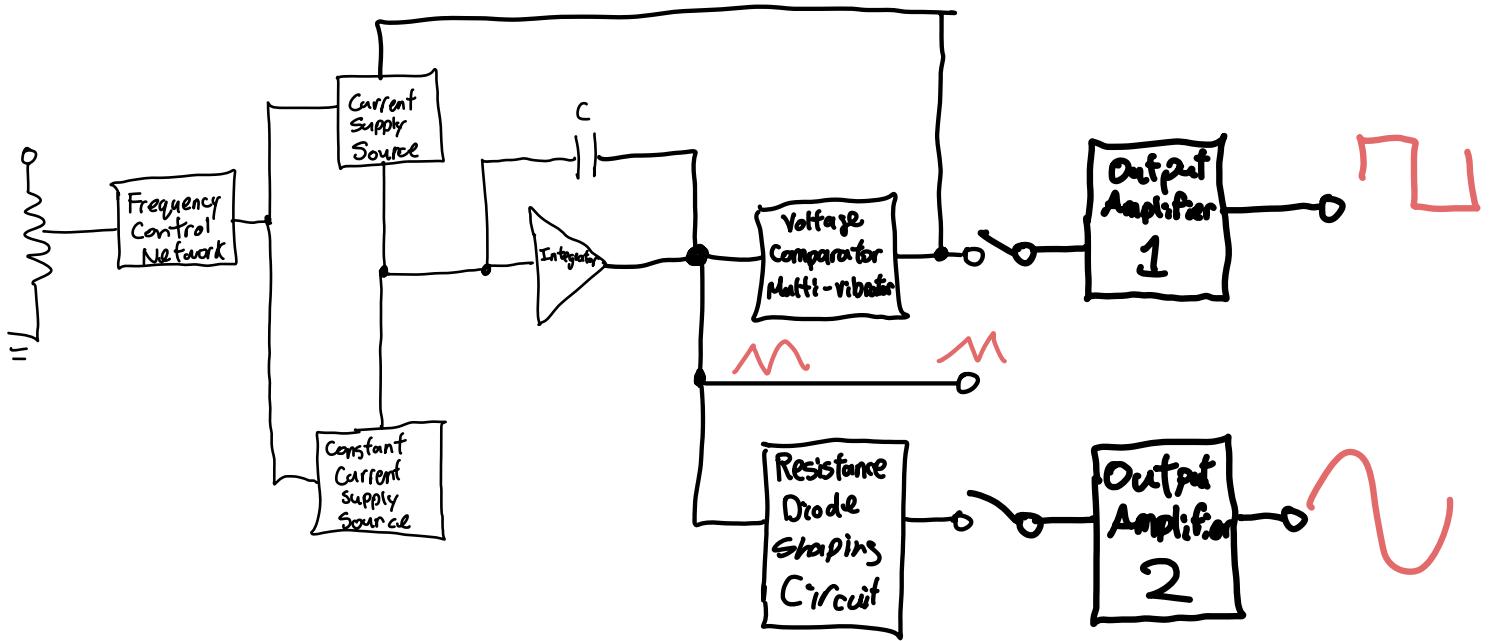


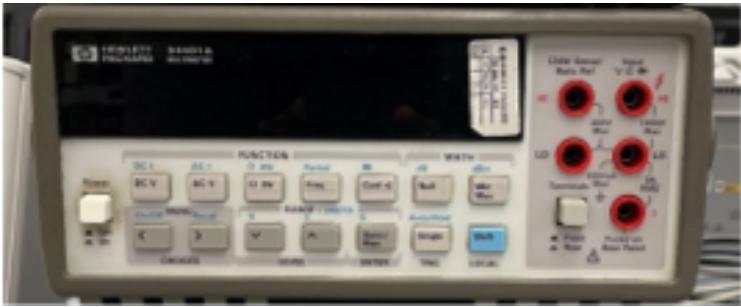
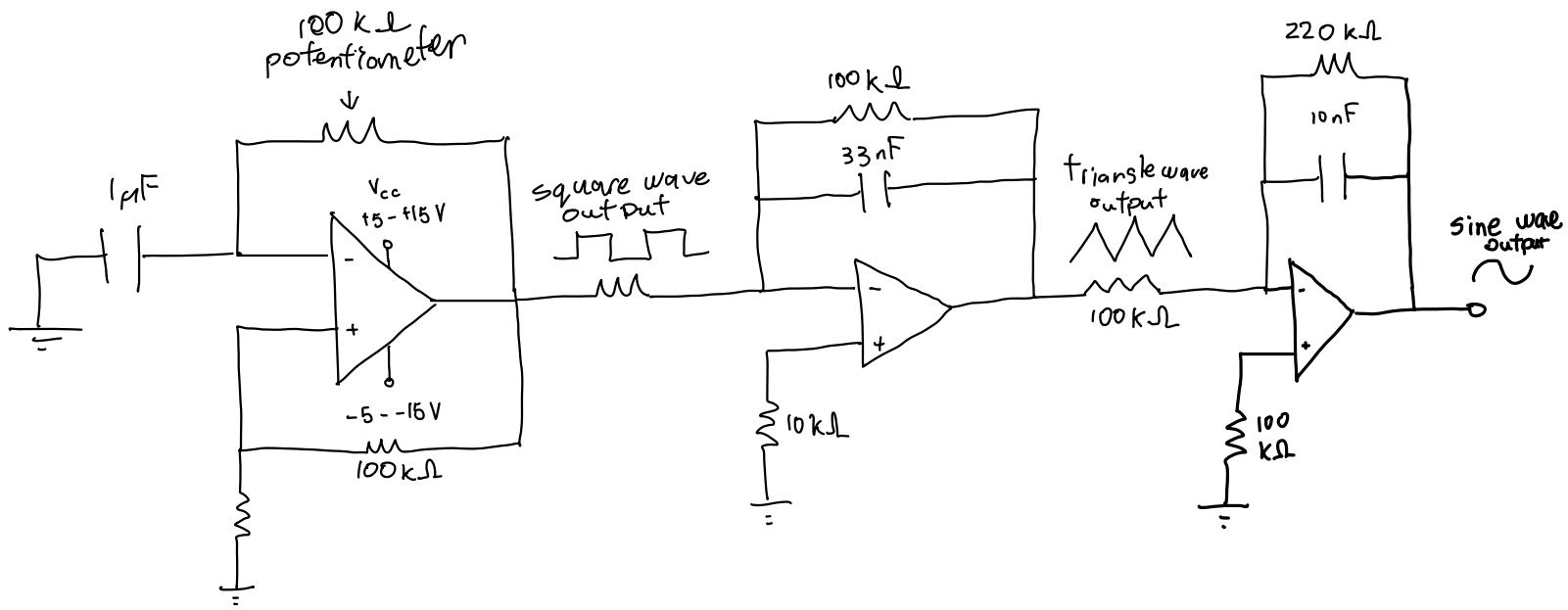
Raspberry Pi replaces  
functional generator

# Functional Generator

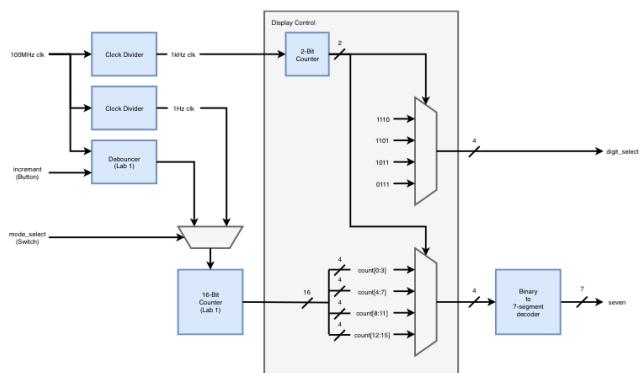


> Agilent 33120A

> TTI TGF 4000

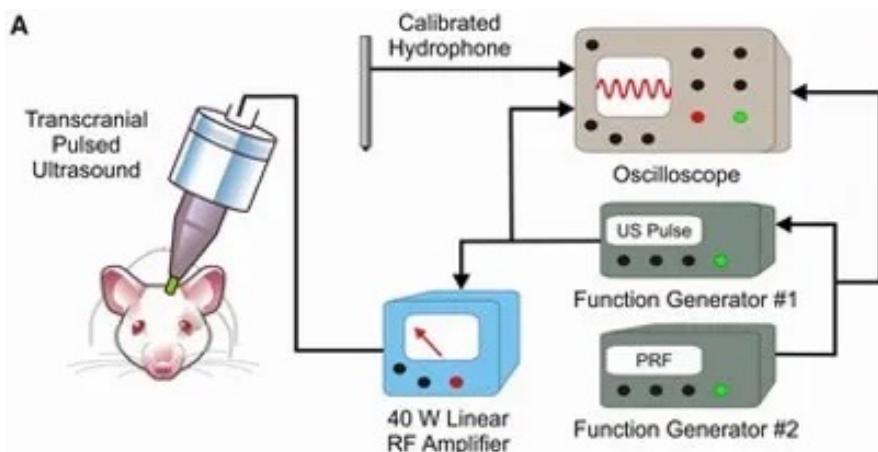


## Function Generator

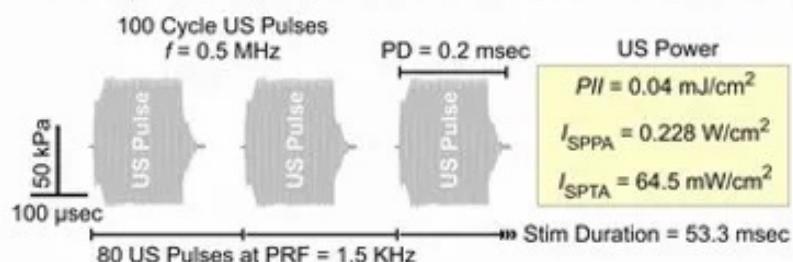


**Figure 3:** The overview of top level module for part 2. Note that reset is not shown on the diagram but it must be included in your design.

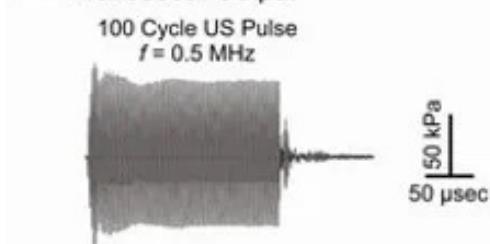
The piezoelectric micromachined ultrasonic transducers (pMUTs) with small membranes (sub-mm membranes) generate enough power to stimulate neurons and enable precise modulation of neural circuits. We designed the ultrasound transducer as an array structure to enable localized modulation in the target region



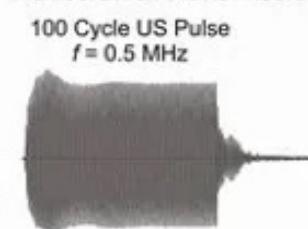
**B Strategy for Designing Pulsed Ultrasound Stimulus Waveforms**

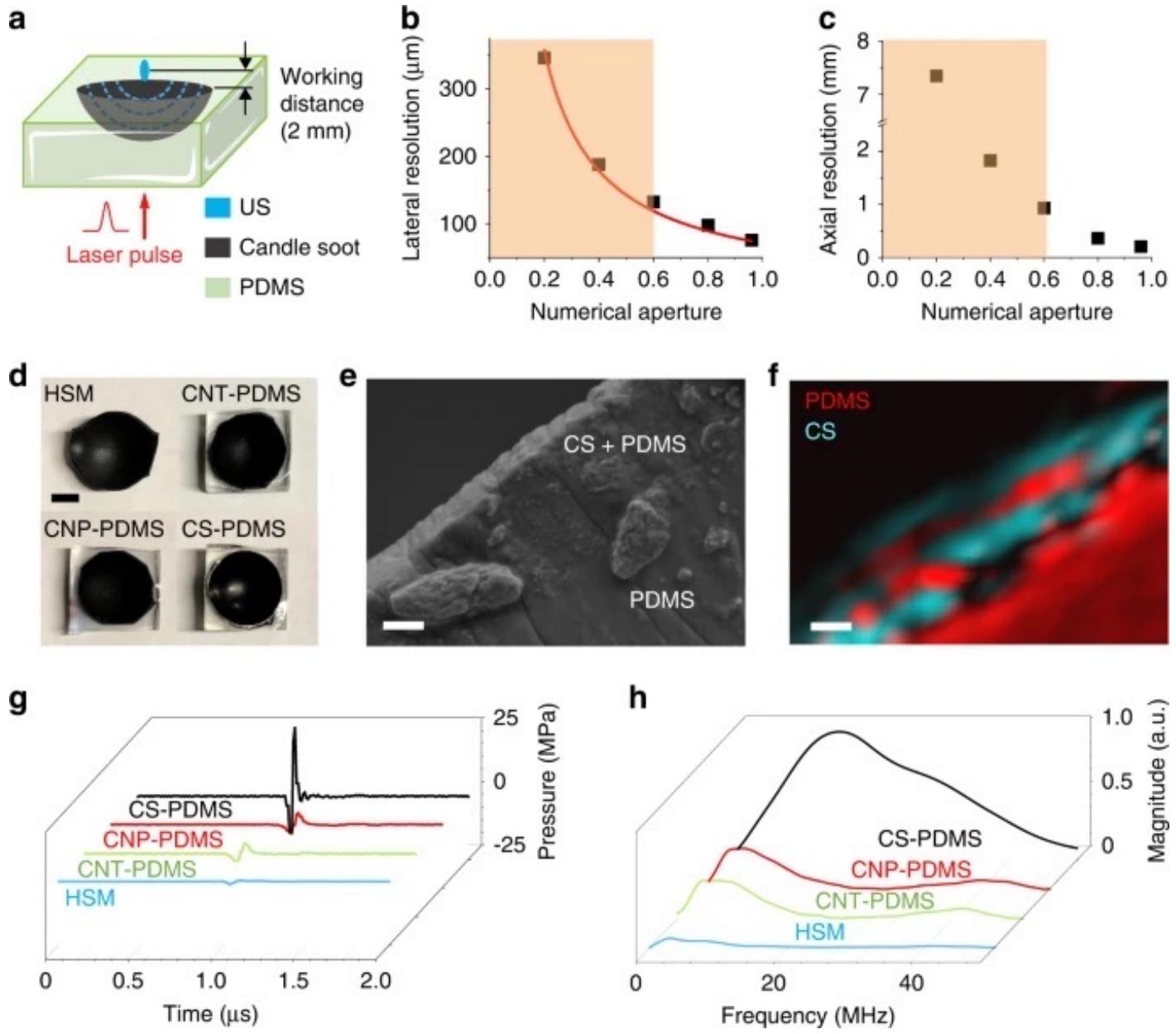


**C Transducer Output**

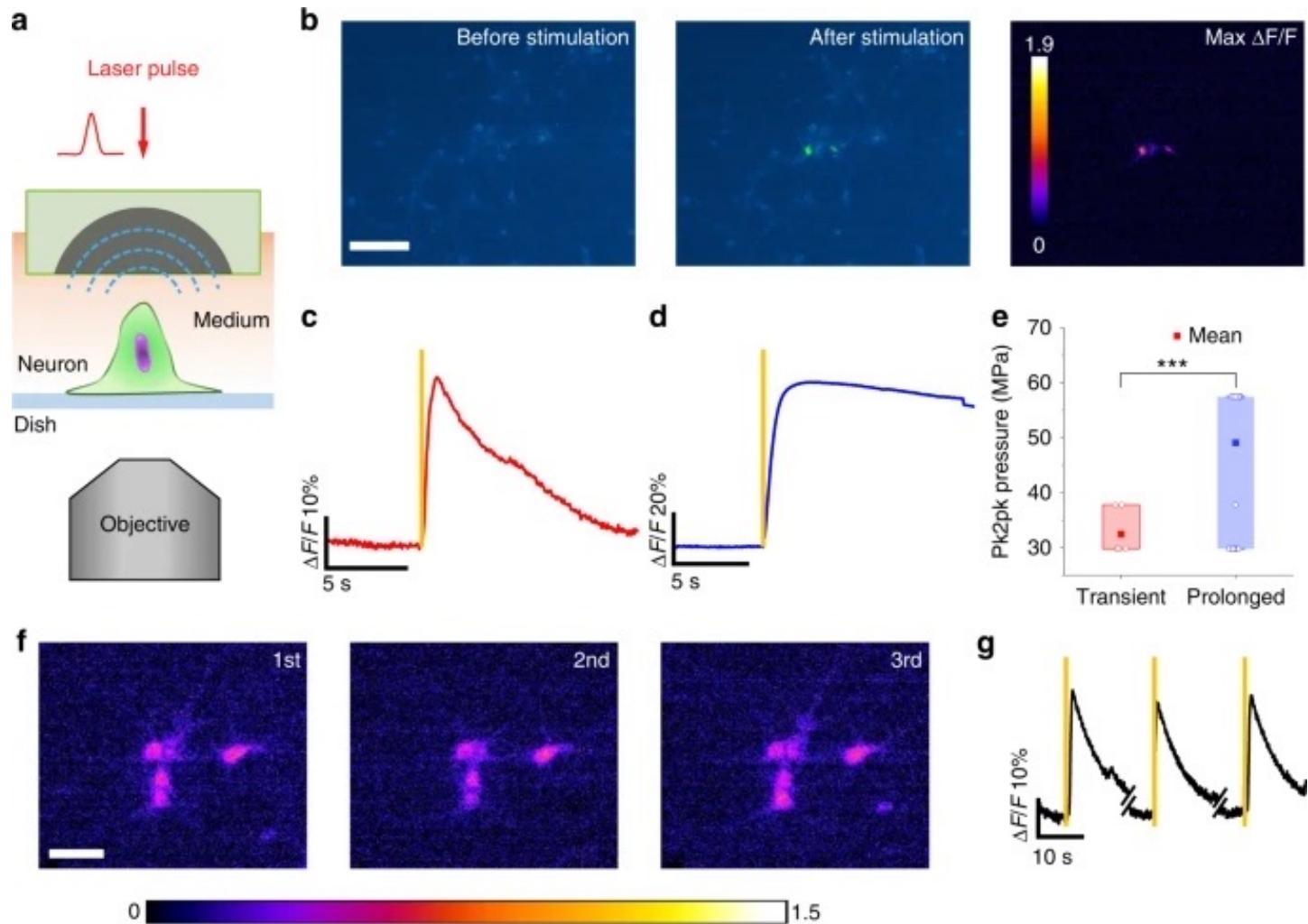


**Transcranial Transmission**





a The schematic of SOAP design. b Numerical aperture and lateral resolutions. Red line: fitting curve. Orange area: the NA range of conventional ultrasound transducers. c Numerical aperture and axial resolutions. d Photos of 4 kinds of SOAPS with the same geometric design. From left to right, top to bottom: heat shrink membrane (HSM), carbon nanotube-PDMS (CNT-PDMS), carbon nanoparticles-PDMS (CNP-PDMS), candle soot-PDMS (CSPDMS). Scale bar: 5 mm. e SEM image of the CS-PDMS SOAP cross-section. Scale bar: 1 μm. f The spatial distribution of PDMS and CS in SOAP cross-section by SRS and photothermal imaging. Red: PDMS. Cyan: CS. Scale bar: 1 μm. g The ultrasound waveforms from 4 kinds of SOAPS with the same laser energy input. h The frequency spectra of ultrasound from 4 kinds of SOAPS



1 ns reading counters that can count at 1 GHz

34 Channel Lab1034 Logic port Logic analysis

Access the Raspberry Pi System Timer registers directly.

```
//  
#include <stdio.h>  
#include <stdlib.h>  
#include <fcntl.h>  
#include <sys/mman.h>  
#include <unistd.h>  
#include <stdint.h>  
  
// #define PERIPHERAL_BASE 0x20000000 // For Pi 1 and 2  
#define PERIPHERAL_BASE 0x3F000000 // For Pi 3  
#define SYSTEM_TIMER_OFFSET 0x3000  
#define ST_BASE (PERIPHERAL_BASE + SYSTEM_TIMER_OFFSET)  
  
// System Timer Registers layout  
typedef struct {  
    uint32_t control_and_status;  
    uint32_t counter_low;  
    uint32_t counter_high;  
    uint32_t compare_0;  
    uint32_t compare_1;  
    uint32_t compare_2;  
    uint32_t compare_3;  
} system_timer_t;  
  
// Get access to the System Timer registers in user memory space.  
system_timer_t * get_system_timer() {  
    void *system_timer;  
    int fd;  
  
    if ((fd = open("/dev/mem", O_RDWR | O_SYNC) ) < 0) {  
        printf("can't open /dev/mem \n");  
        exit(-1);  
    }  
  
    system_timer = mmap(  
        NULL,  
        4096,  
        PROT_READ | PROT_WRITE,  
        MAP_SHARED,  
        fd,  
        ST_BASE  
    );  
  
    close(fd);  
  
    if (system_timer == MAP_FAILED) {  
        printf("mmap error %d\n", (int)system_timer); // errno also set!  
        exit(-1);  
    }  
}
```

```

}
return (system_timer_t*)system_timer;
}

int main(int argc, char **argv) {
    volatile system_timer_t* system_timer = get_system_timer();
    int32_t t0, t1;

    while (1) {
        t0 = system_timer->counter_low;
        sleep(1);
        t1 = system_timer->counter_low;
        printf ("Elapsed = %d\n", t1 - t0);
        printf ("Counter high = %d\n", system_timer->counter_high);
        t0 = t1;
    }
    return 0;
}

```

Raspberry Pi

timer  
counter

1 GHz clock

Pico maxes out at 133 MHz

The MicroPython utime page explains how there is a difference between absolute time from `time_ns()` and relative time from `ticks_us()`. They are best used for different purposes, and probably use different resources. From the examples there, you could try something like

```
start = time.ticks_us()
```

```
...
```

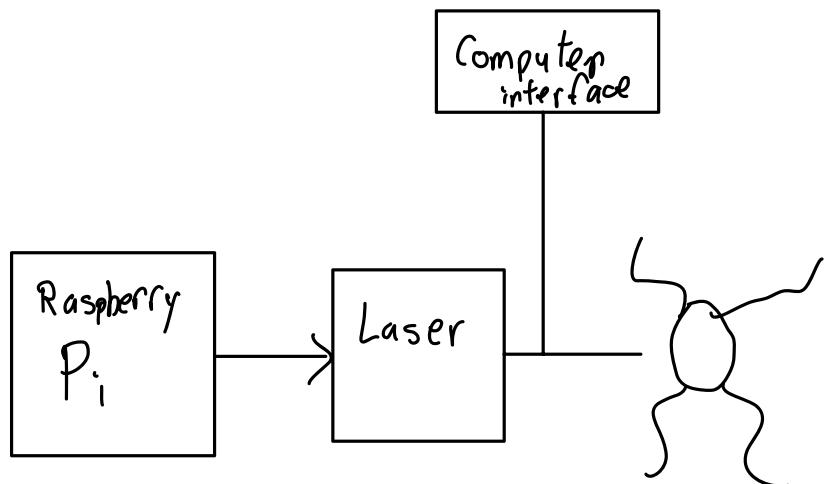
```
end = time.ticks_us()
```

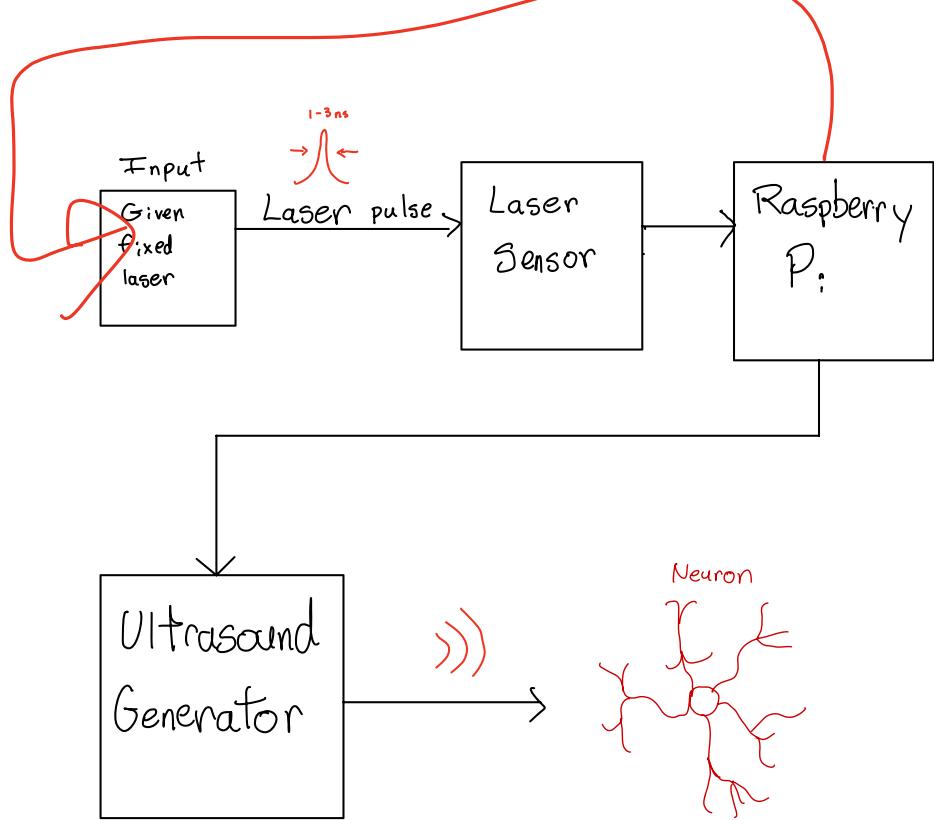
```
usecs = time.ticks_diff(end, start)
```

As the page explains, this cannot be used to measure long times, such as more than 1 or 2 seconds, depending on implementation, and the resolution will not be nanoseconds, but at best microseconds

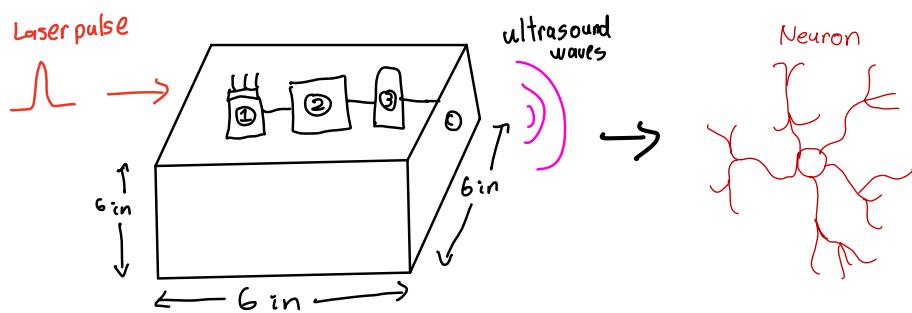
Arduino

↳ video



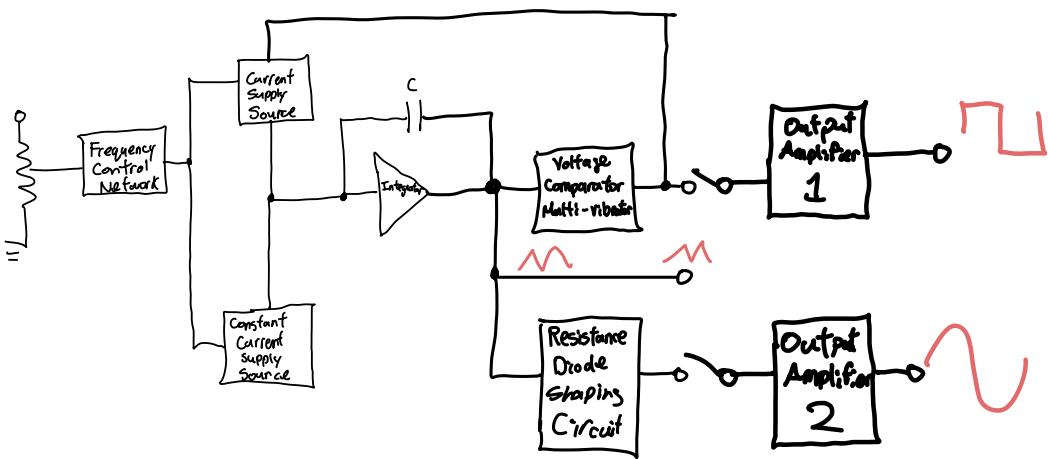


## Visualization



- ① Laser Sensor
- ② Raspberry Pi
- ③ Ultrasound Generator

# Functional Generator



Proposed enclosure for  
device

Idea of what final problem  
will look like

Arduino + Voltage converter

25V

\$500 - 1000