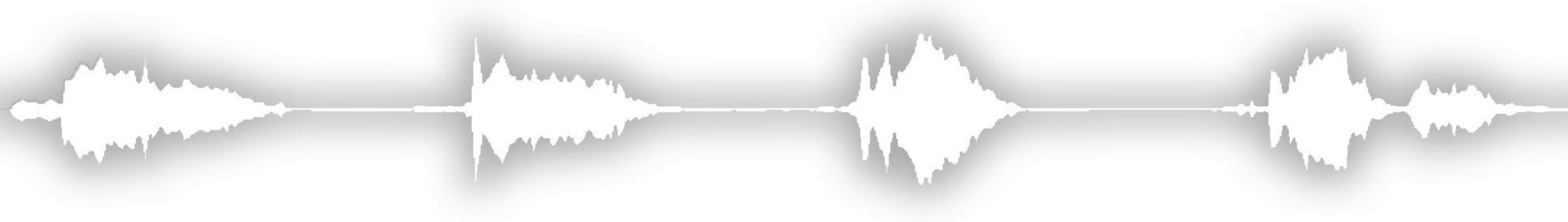
Building a real-time audio sampling application with MicroPython

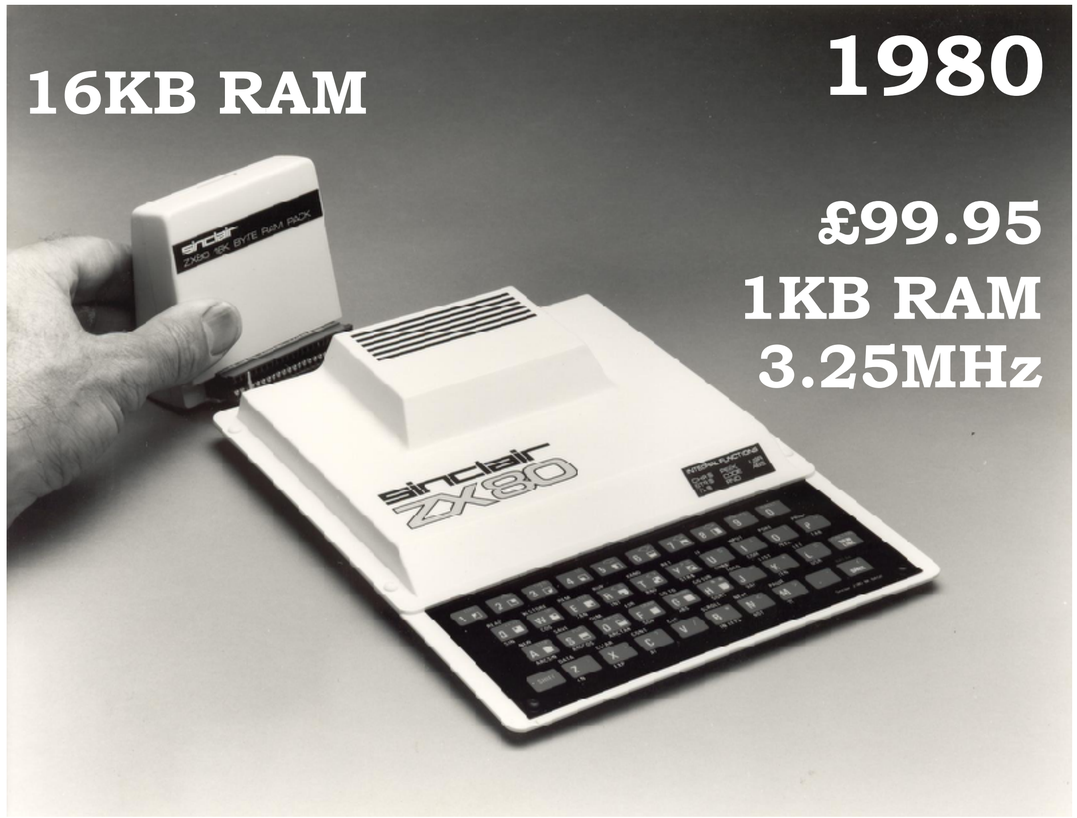


Alan.Christie at **MatildaPeak.com**

AlanBChristie

Me...

- Started writing software a long time ago...



This session...

- ...we'll explore a continuous "listen, repeat" audio application
- ...is for beginners
- ...will introduce...
    - MicroPython (the parts we need anyway)
    - The PyBoard
    - The PyBoard audio skin

MicroPython

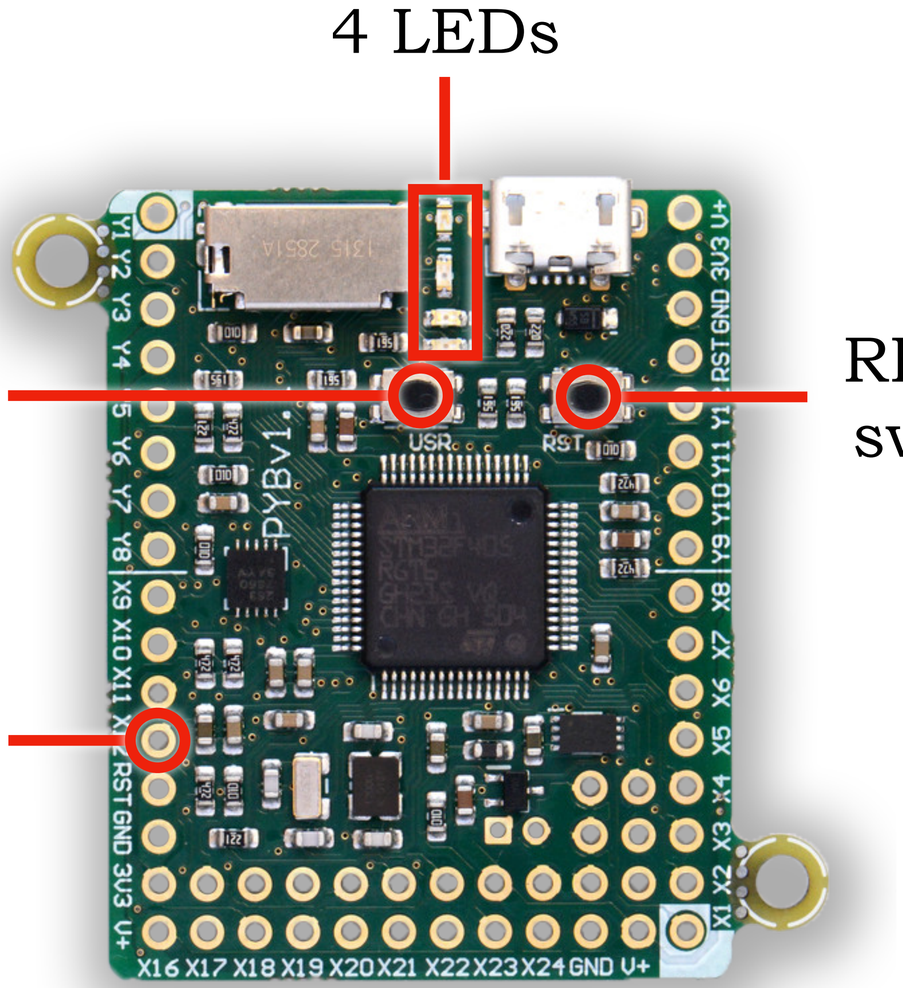
- Lean implementation of Python 3 (256kB)
- Optimised for micro-controllers
- Numerous modules for hardware control

- Originally created by Australian programmer and physicist Damien George
- Backed by Kickstarter campaign in 2013

The PyBoard (v1.1)

- Just one of a number of hardware architectures
- 192K RAM (100K heap)
- 48-168MHz ARM
- 12-bit digital resolution (**DAC** & **ADC**)

```
import pyb
```



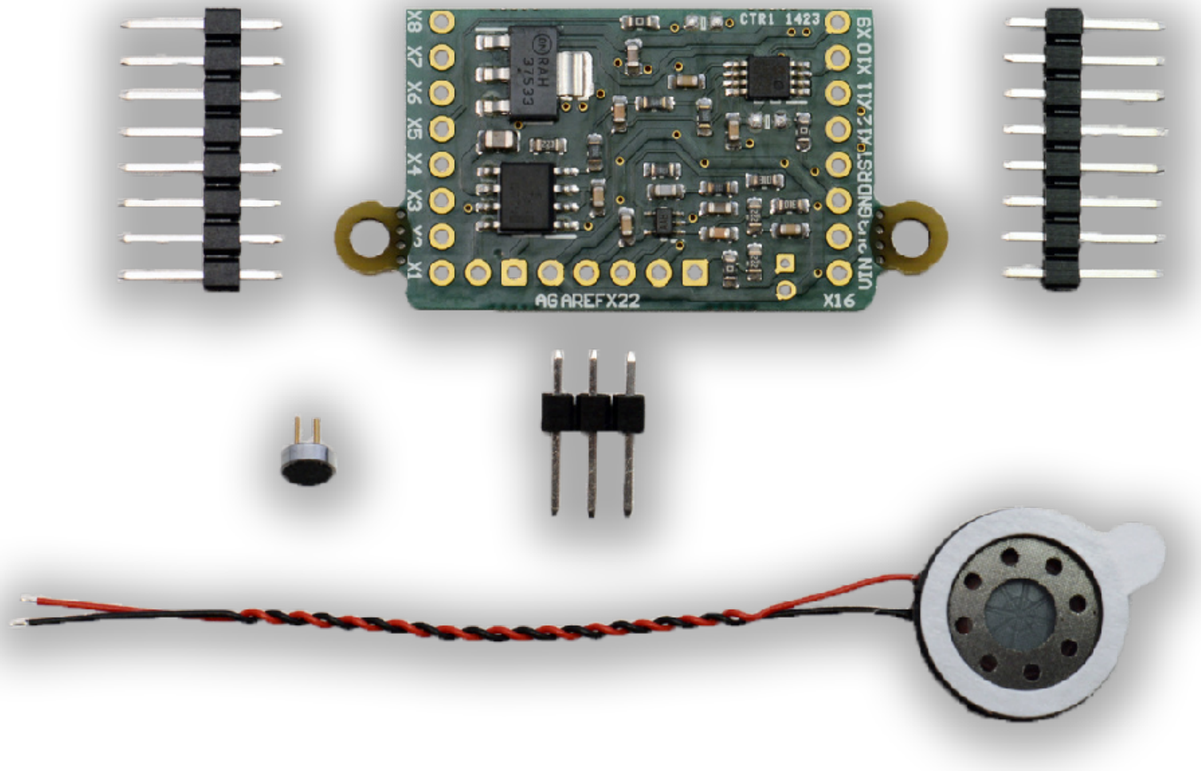
https://store.micropython.org/#/features

The audio skin

- Plugin for PyBoard
- Loudspeaker with small power amp
- Built-in microphone with pre-amp
- Option of external mic

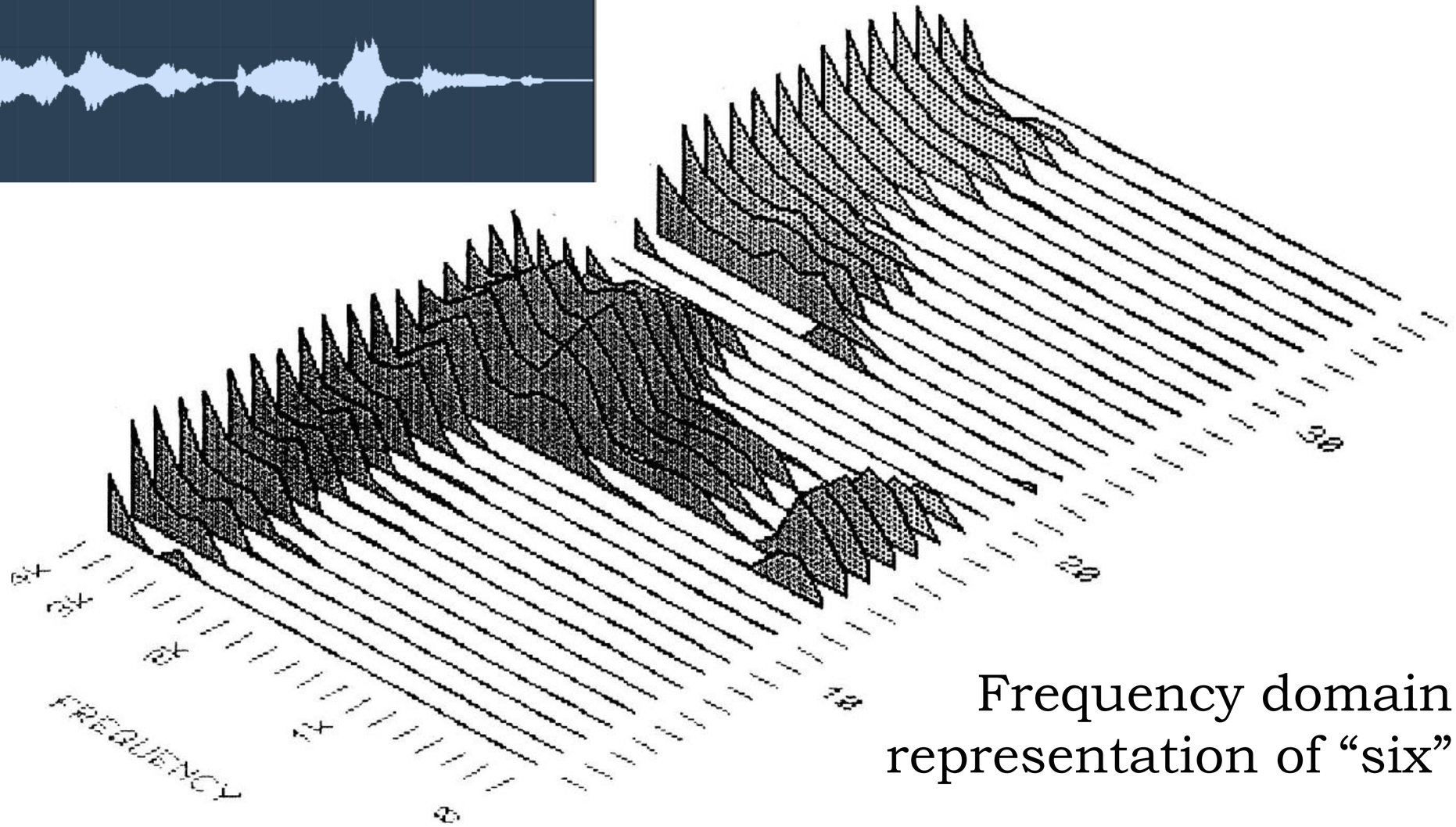
Be prepared !

- The audio skin does require some assembly…



Speech

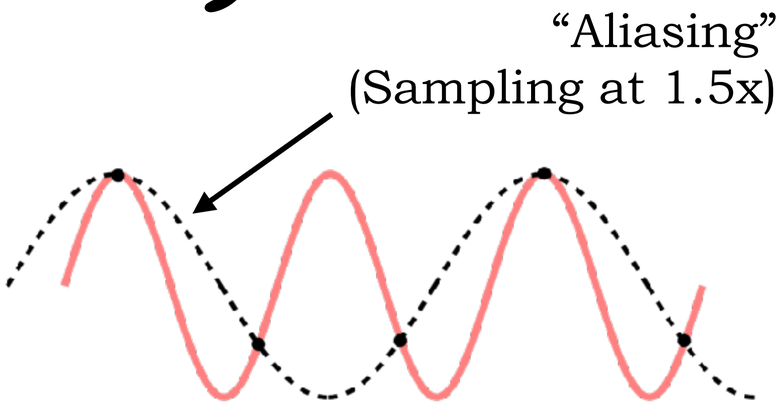
- Hi-fi
  14kHz
- Telephone
  3.4kHz

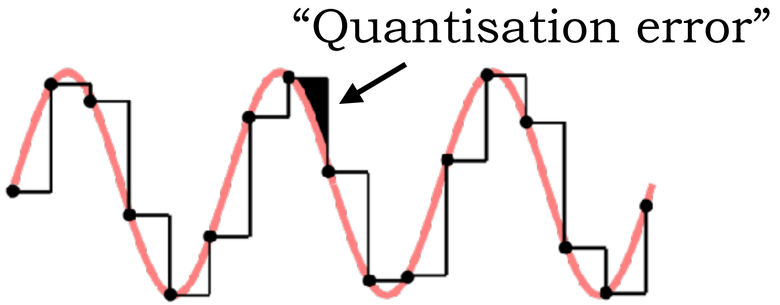


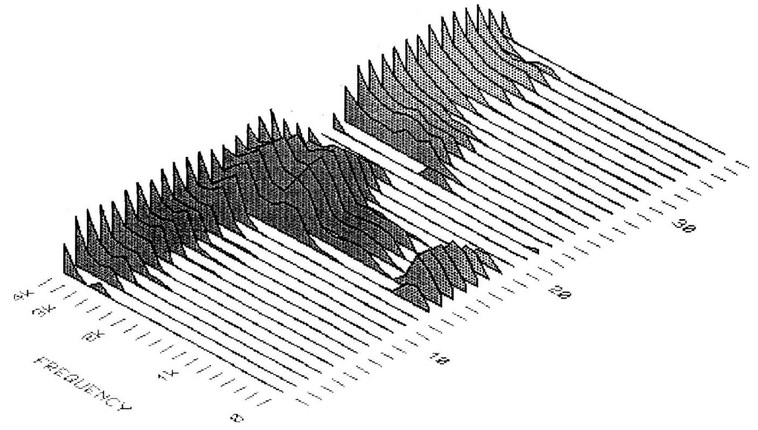
Frequency domain representation of “six”

From analogue to digital (the ADC)

- **Sampling frequency**
  - To avoid “aliasing” sample at > 2x
- **Resolution**
  - Choice of 8 or 12 bits
  - CD audio is 16-bits at 44.1kHz
- **Duration**
  - 300-500mS per word







How do we record? (the ADC)

- Create an ADC object

```
adc = pyb.ADC(pyb.Pin.board.X22)
```

- Then… “read_timed()”

```
timer = pyb.Timer(6, freq=6000)
buf = bytearray(100)
adc.read_timed(buf, timer)
```

- …or (in real-time) “read()”

```
new_sample = adc.read()
```

How do we play? (the DAC)

- Set the volume (using the digital potentiometer on the I2C bus)

```
pyb.I2C(1, pyb.I2C.MASTER).mem_write(volume, 46, 0)
```

- Create a DAC object and tell it where to find the audio

```
dac = pyb.DAC(1, bits=12)
dac.write_timed(buf,
                pyb.Timer(7, freq=SAMPLE_FREQUENCY_HZ),
                mode=pyb.DAC.NORMAL)
```

It has to be real-time - how do we know?

- Use a pin and an oscilloscope
- Clear the pin, “do work”, set the pin

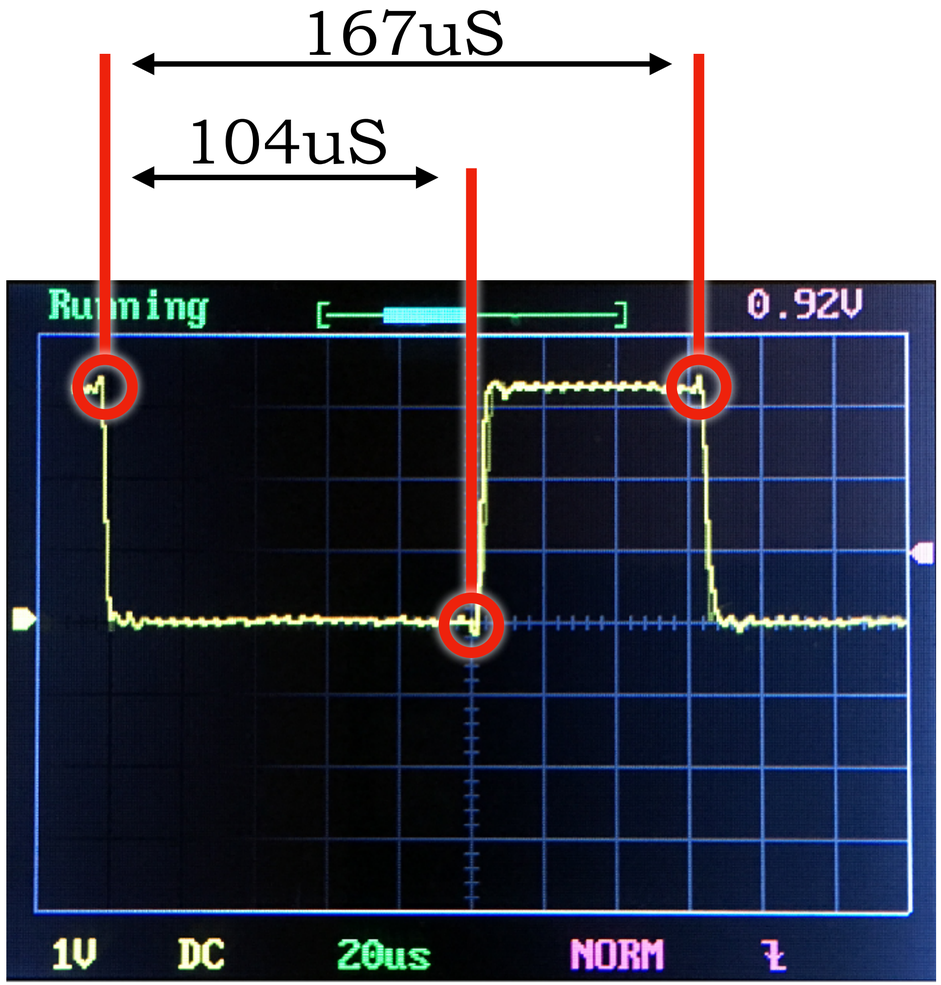
```
timing_pin = pyb.Pin('Y1', pyb.Pin.OUT_PP)
```

```
timing_pin.high()
```

- … or … you can use a MicroPython Timer object

Timing verification

- Confirmation of 6kHz Timer (period of 167uS)
- Capture takes 104uS (9.6kHz max)
- Interestingly…
  - Clearing and reading a PyBoard "Timer" takes **20uS**
  - Calling "adc.read()" takes **50uS** (20kHz upper-bound)

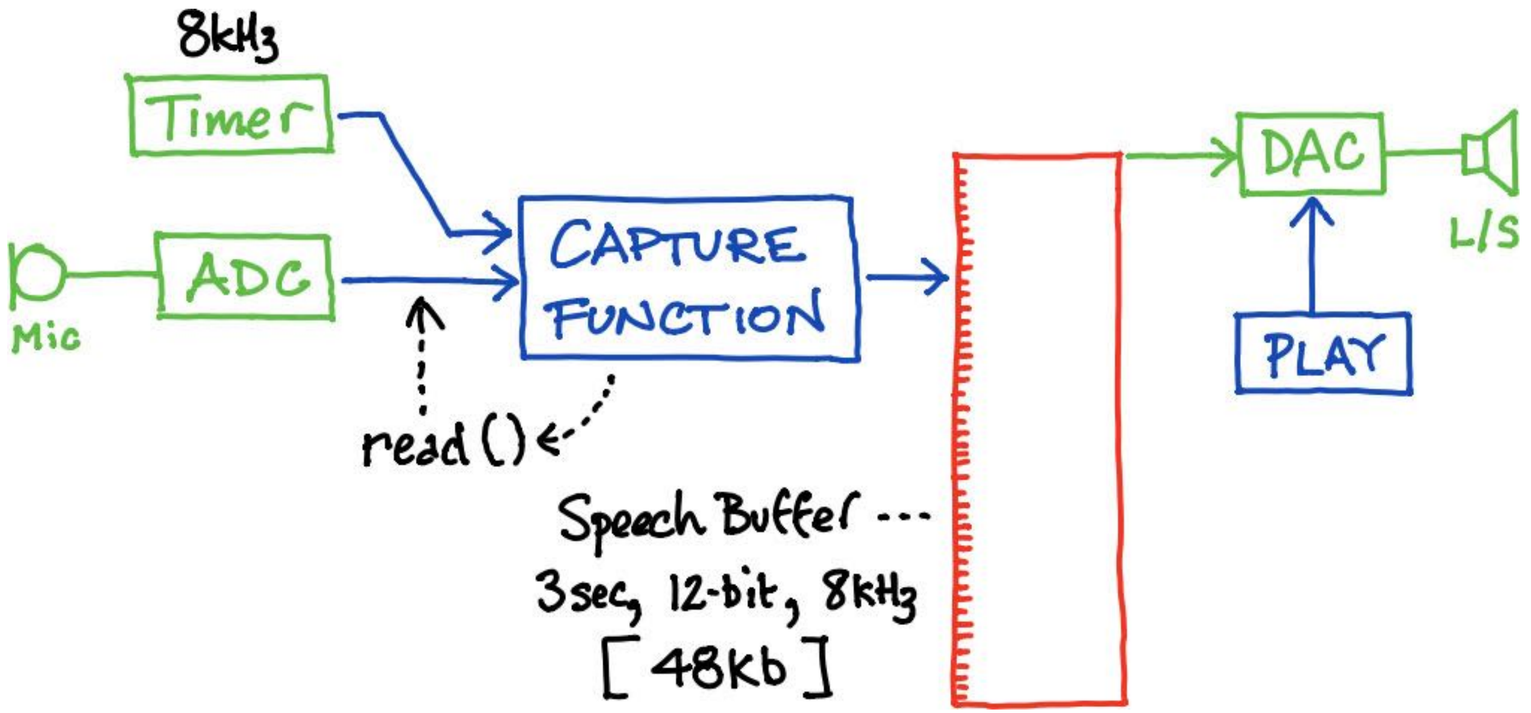


Initial development setup

- Somewhere to store the captured audio (a “**sample buffer**”)
- A time-dependent “**capture function**”

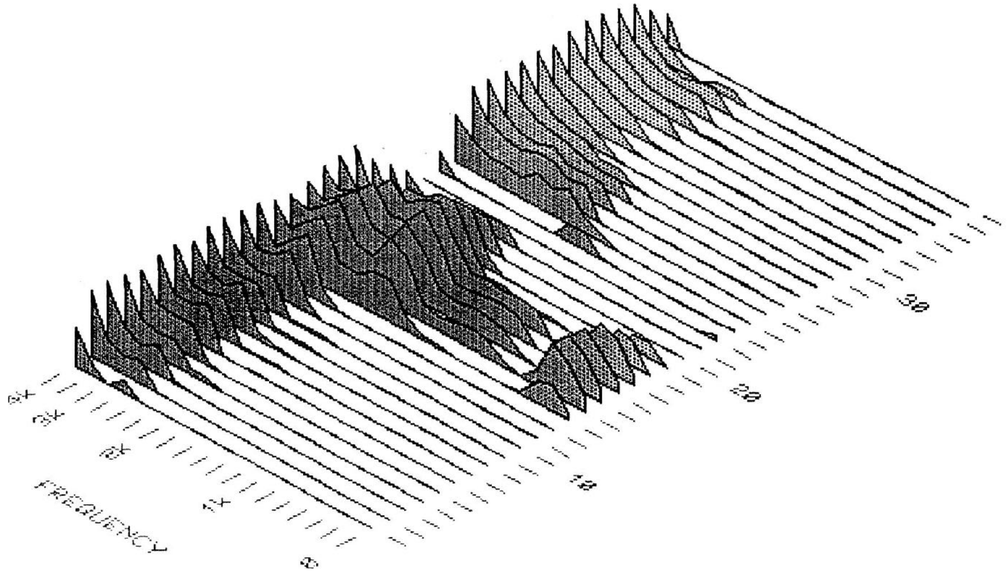
  No object creation
  No Python floats
- Simple “**play**” function

```
capture_timer = pyb.Timer(14)
capture_timer.init(freq=SAMPLE_FREQUENCY_HZ)
capture_timer.callback(_capture_function)
```



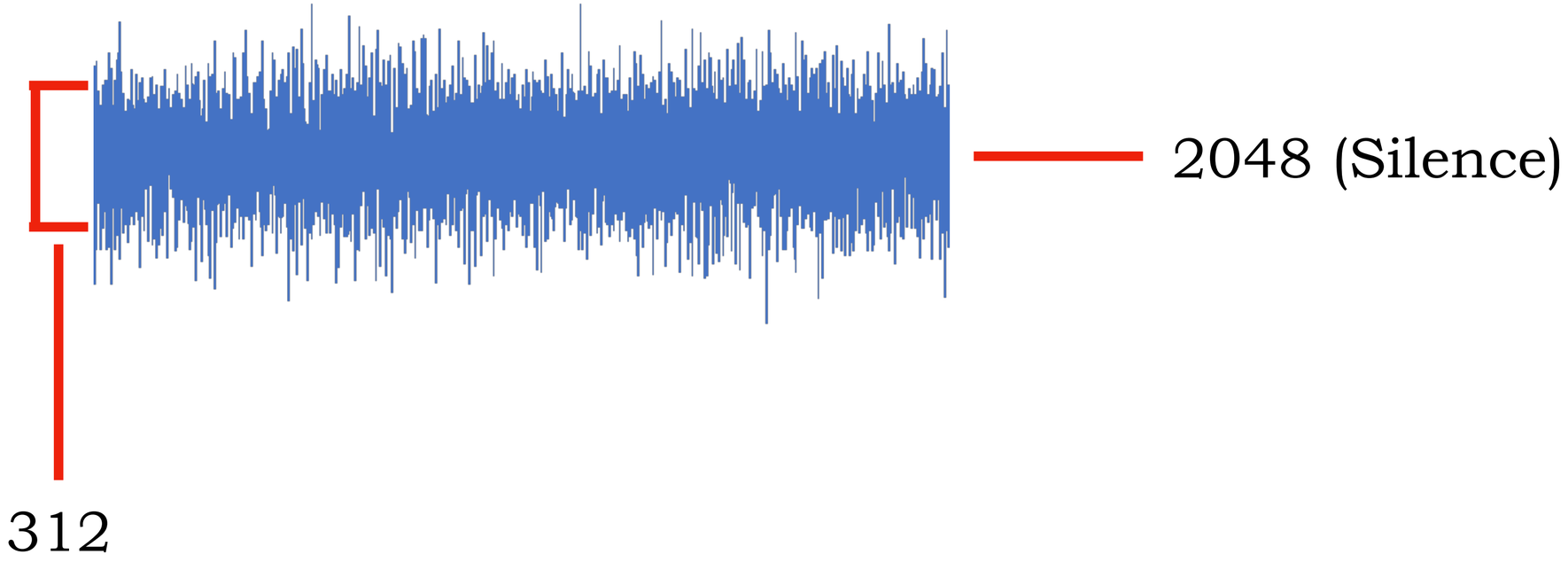
First recordings

- Speaking “**A, B, C**”
  - 8kHz at maximum resolution (12-bits)



- Best sampling frequency, best resolution

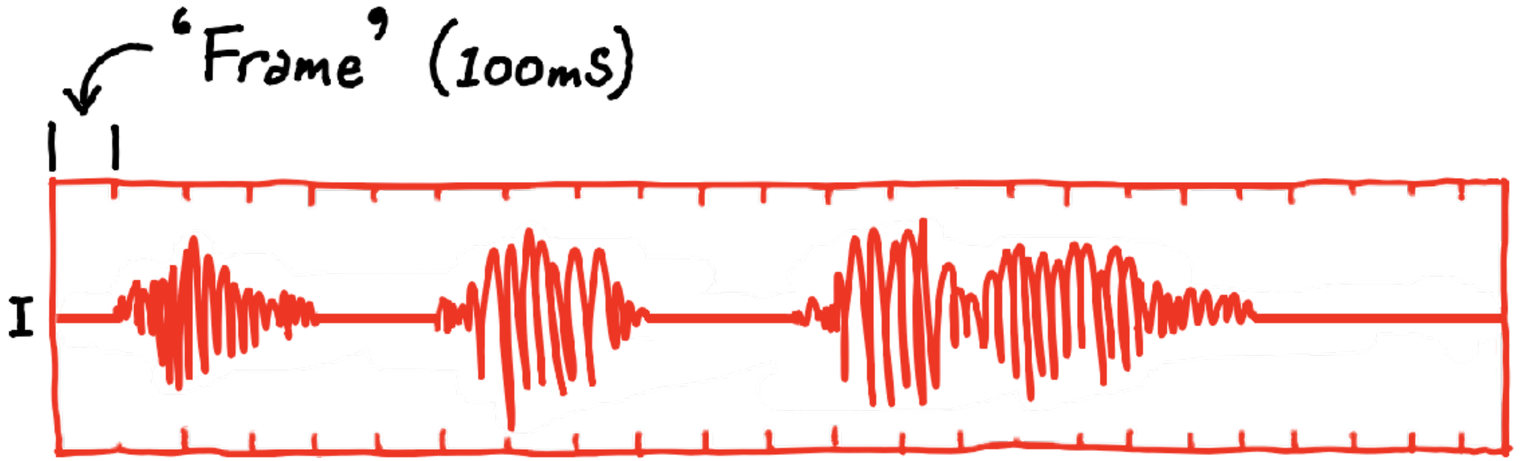
Noise

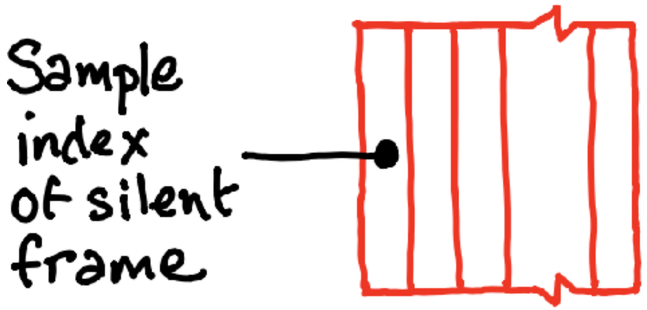


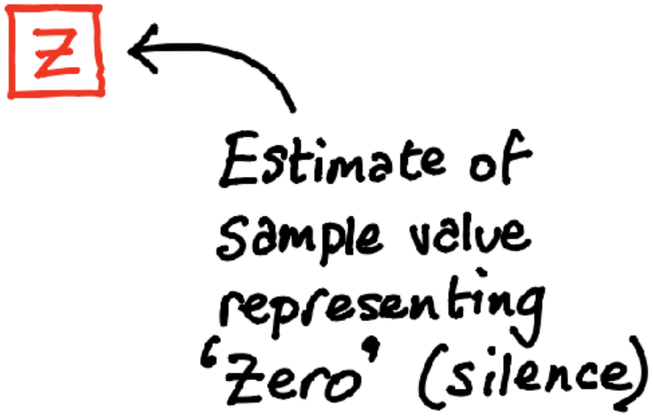
- Recording at highest resolution of 12-bits (0..4095)
- 95% of noise samples occupy 312 values around “zero”
- 8-9 bits consumed

Reducing some noise (a ‘quick-win’)

- **Estimate** the silence level
- **Remember** silent sections
- **Clear** silent sections at the end

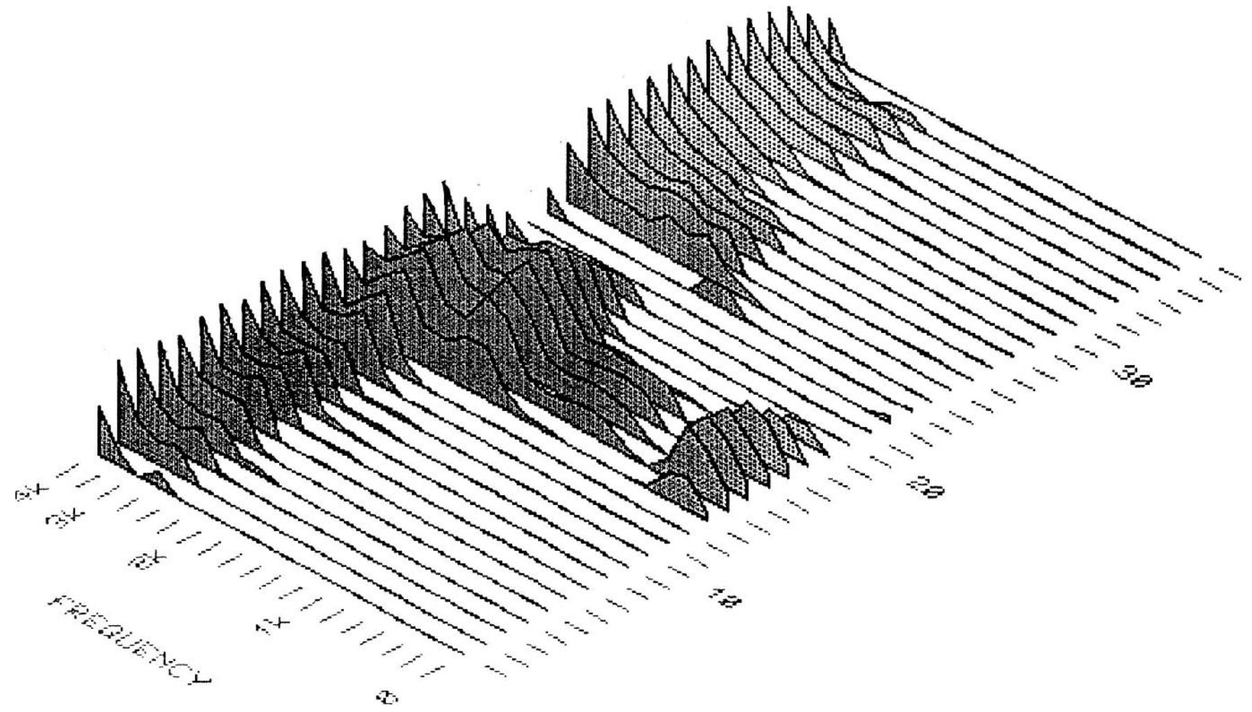






Noise reduced recordings

- Speaking “**A, B, C**”
  - 8kHz at 12-bits
  - 8kHz at 8-bits



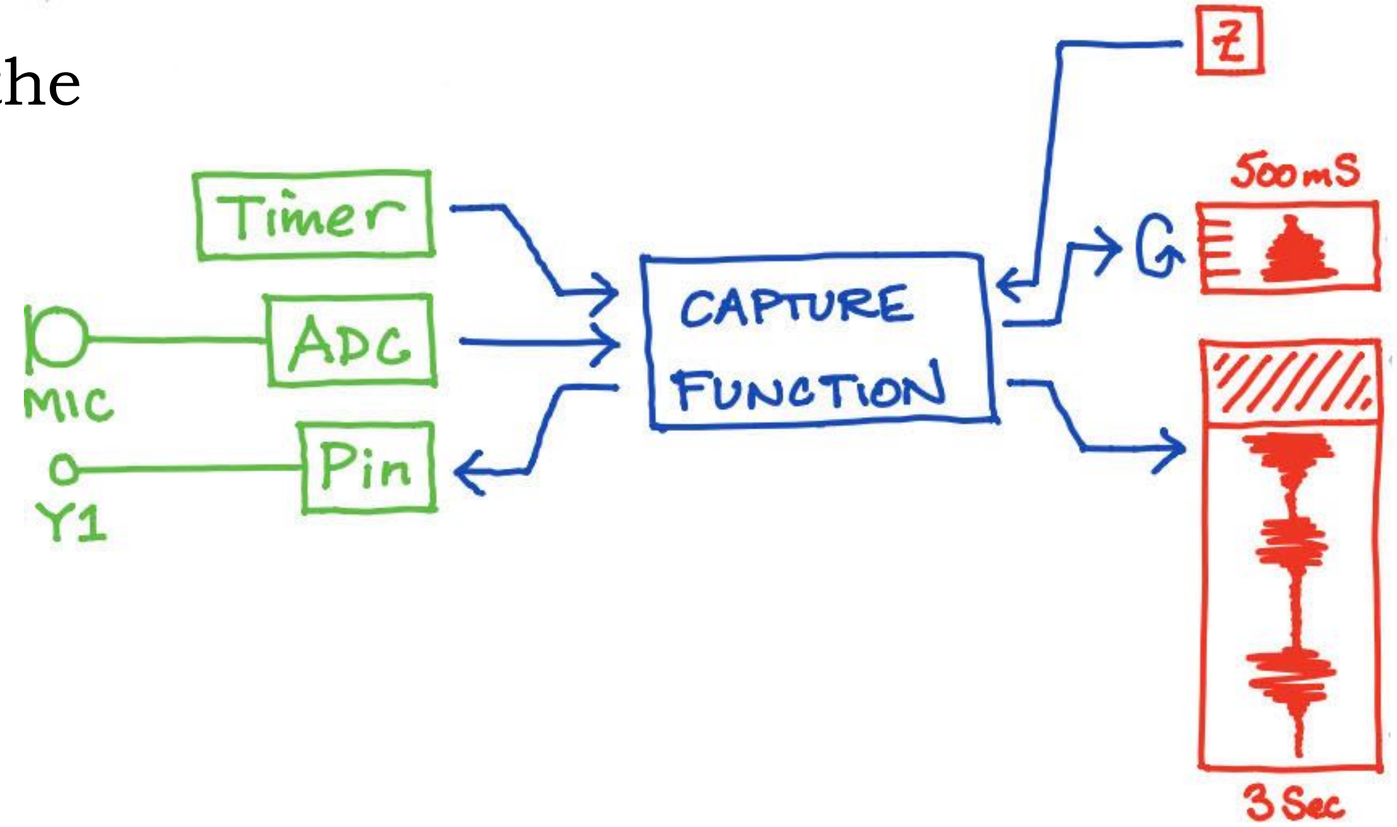
- 8-bits sounds OK and it means we can make longer recordings!

From experiment to application

- **Speech** detection
- **End-of-speech** detection
- **USER switch** to “enable/disable”
- **LEDs** to indicate states
  - “Listening”, “Recording”, “Playing” and “Saving”
- **Save** recordings to an SD card

(Simple) speech detection

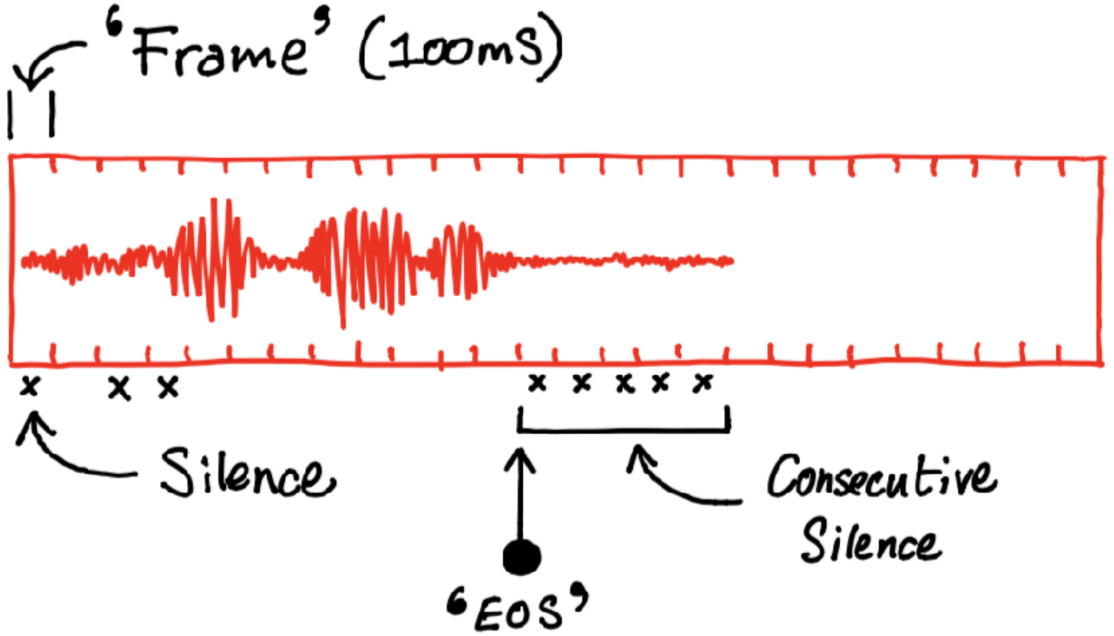
- **Capture** to a small “circular” buffer prior to “recording”
- **Analyse** samples within the “capture function”
- **Switch** to “record” on speech detection



(Simple) end-of-speech detection

- I want to record for as long as possible but also want a more **responsive** (faster) playback

- **Accumulate** consecutive silence frames
- **Stop** after sufficient consecutive frames of silence



Responding to the USER switch

- Provide a “handler” function

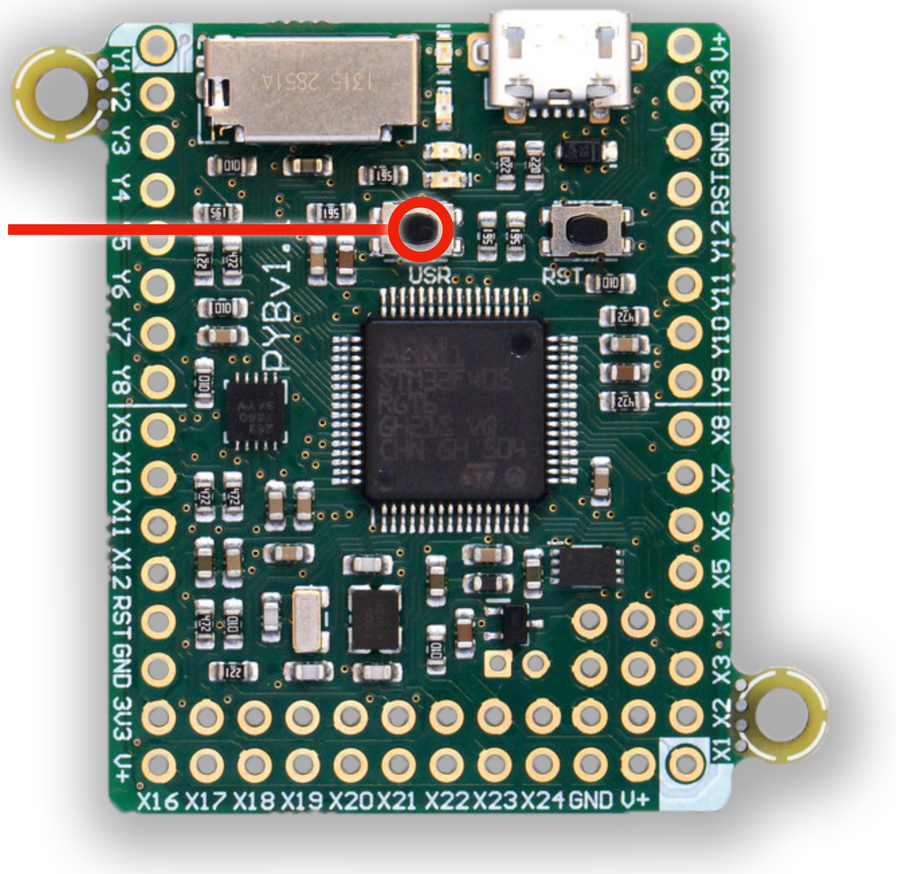
```
def _user_switch_callback():
    if on_hold:
        on_hold = False
    else:
        on_hold = True
```

- Create a “Switch” object

```
sw = pyb.Switch()
```

- Attach the “handler” as a callback

```
sw.callback(_user_switch_callback)
```



Driving the LEDs

- Create an LED object (1..4)

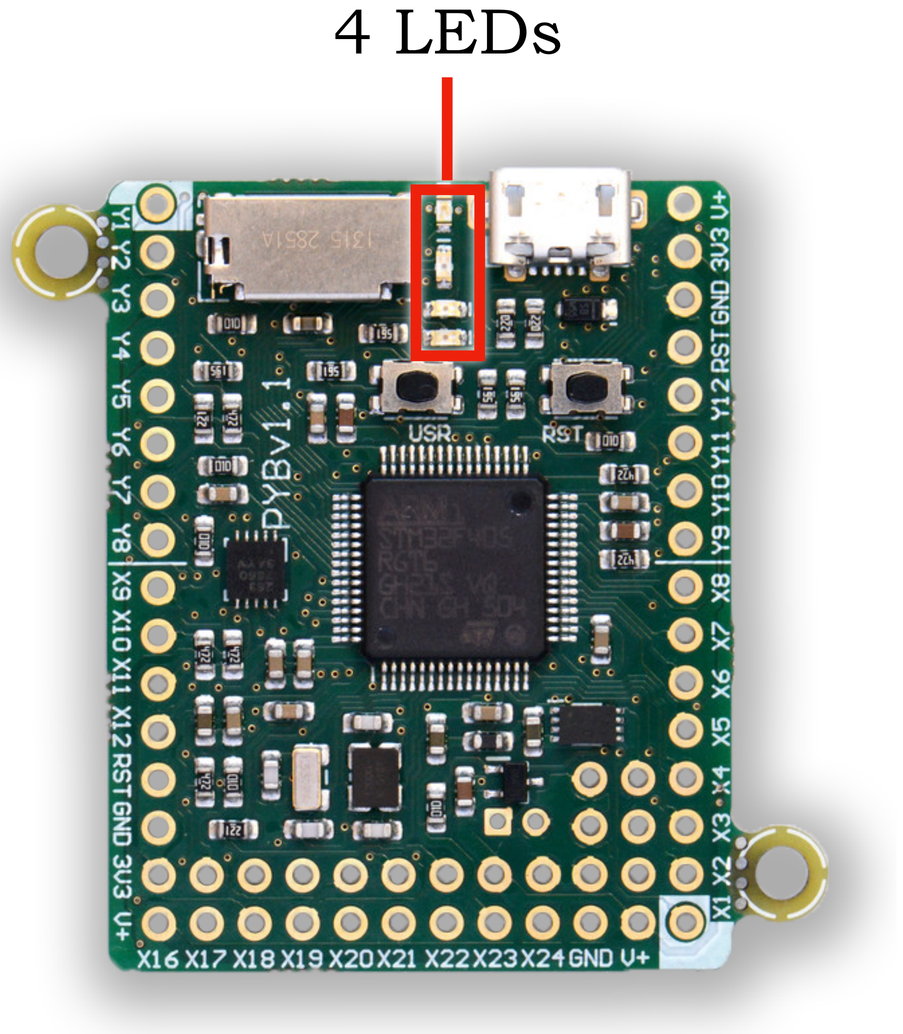
```
green_led = pyb.LED(2)
```

- Call “on()” or “toggle()”

```
green_led.on()
```

- The blue LED (4) supports “intensity(n)”

```
blue_led.intensity(200)
```

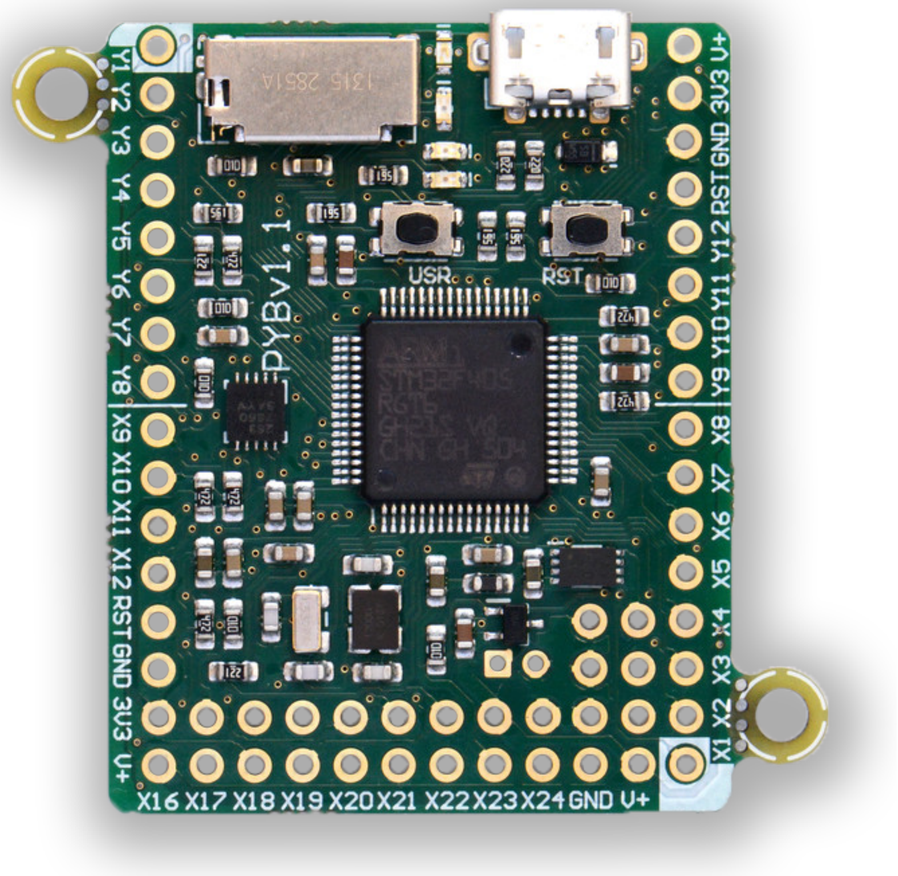


Writing to an SD card

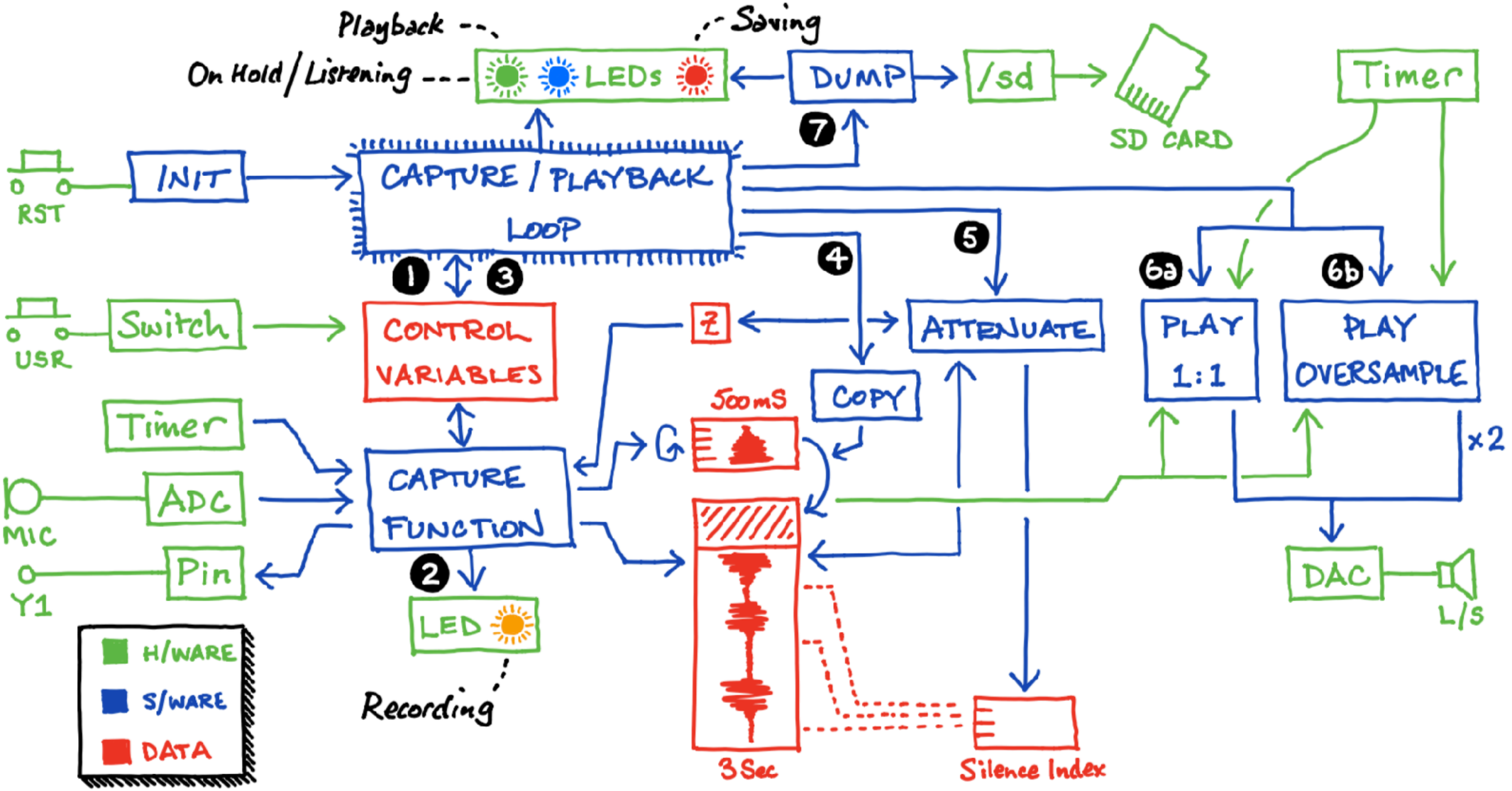
- Check if an SD card is present

```
if '/sd' not in sys.path:
    return
```

- Open, write, close

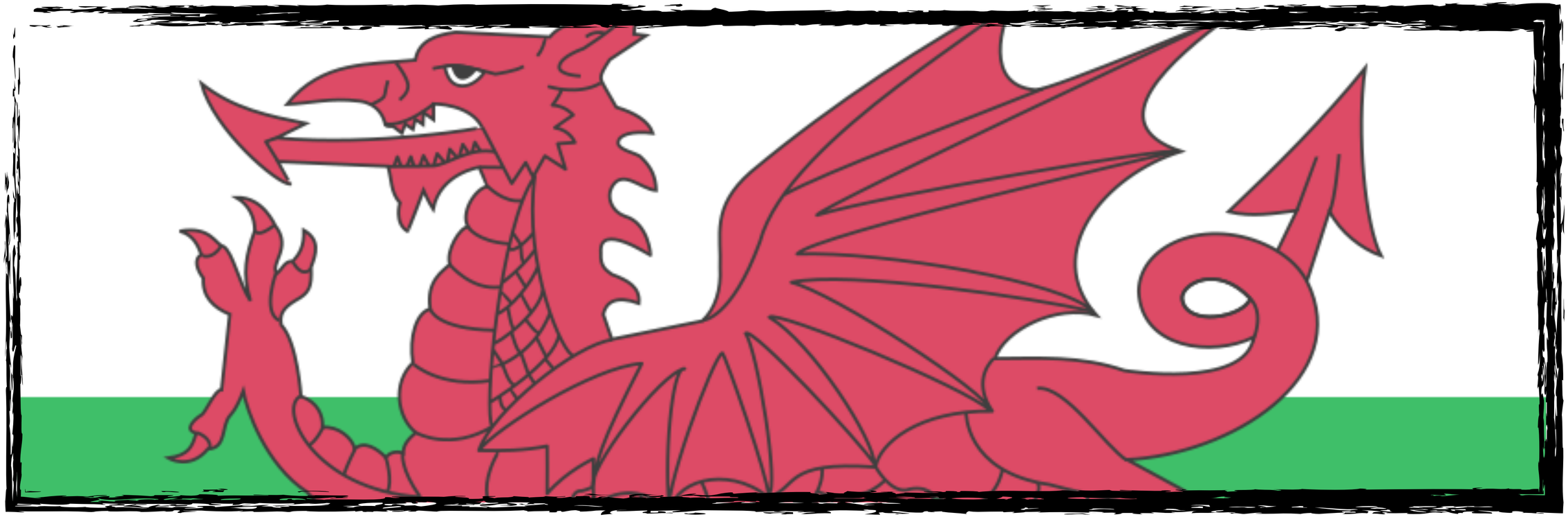


Putting it all together...



Diolch am wrando

Thank you for “listening”



Alan.Christie at **MatildaPeak.com**

@AlanBChristie

GitHub https://github.com/**alanbchristie**/**PyBdEcho**