01 TWO WAYS TO SEE THE SIGNAL

Option	Board	Link to PC	Viewer	Best for
USB sketch	Any Arduino-compatible board	USB serial, 115200 baud	Arduino Serial Plotter	First test, cleanest signal
Wi-Fi sketch	UNO R4 WiFi, ESP32, ESP8266	Wi-Fi, UDP port 5005	EasyEXG Studio (Python)	No cable to the person, filters, CSV

Source: github.com/pieeg-club/EasyEXG · folders Arduino and SDK

02 USB SKETCH · ARDUINO/EASYEXG_ARDUINO.INO

EasyEXG_Arduino.ino (core of loop)

```
const int  PIN  = A0;
const long BAUD = 115200;
const int  FS   = 500;        // samples per second

void loop() {
  int raw = analogRead(PIN);
  baseline += (raw - baseline) * 0.002f;  // drift, ~1 s
  float signal = raw - baseline;         // centred at 0
  envelope += (fabs(signal) - envelope) * 0.02f;
  Serial.print(signal);   Serial.print(' ');
  Serial.println(envelope);
}
```

- Reads A0 at 500 Hz and prints two numbers per line.
- Open Tools → Serial Plotter at 115200 baud.
- Blue trace: the signal. Yellow trace: the envelope (activity level).
- Other boards: change PIN (e.g. PA0 on STM32; A0 is GP26 on the Pico).
- Laptop on battery while electrodes are on a person.

03 WI-FI SKETCH · SDK/ARDUINO/1_ARDUINO.INO

Samples A0 at 500 Hz, removes drift and 50 Hz hum (notch), calculates the envelope and sends 25 samples per packet, 20 packets per second over UDP. Boards: Arduino UNO R4 WiFi, ESP32, ESP8266. The USB cable is only for uploading; then power the board from a power bank.

Choose the Wi-Fi mode (top of the sketch)

```
#define USE_ACCESS_POINT 0          // 0 = join your Wi-Fi
const char* STA_SSID = "YOUR_WIFI_NAME";   // 2.4 GHz
const char* STA_PASS = "YOUR_WIFI_PASSWORD";
```

Mode	Network
USE_ACCESS_POINT 0	Your router, 2.4 GHz (recommended)
USE_ACCESS_POINT 1	Board makes "EasyEXG", password easyexg123

Setting	Default	Meaning
FS	500	Samples per second. 1000 is better for EMG
ADC_BITS	10	ADC resolution. Use 14 on the UNO R4 for EEG
USE_NOTCH	true	Remove mains hum
NOTCH_HZ	50.0	50 Hz in UK / Europe / Asia, 60 Hz in the Americas
UDP_PORT	5005	Must match the Python viewer

Serial Monitor at 115200 baud after upload

```
Connecting to YOUR_WIFI_NAME....
Board IP: 192.168.1.57  UDP port: 5005
Waiting for PC on Wi-Fi 'YOUR_WIFI_NAME', board IP 192.168.1.57
```

04 EASYEXG STUDIO · SDK/PYTHON/EASYEXG_STUDIO.PY

Windows, macOS, Linux · Python 3.9+

```
pip install matplotlib numpy scipy
cd SDK/Python
python easyexg_studio.py
```

- PC on the same Wi-Fi as the board (or on "EasyEXG" in mode 1).
- Click EEG, EOG, EMG or ECG: the window shows the JP1 setting and electrode positions.
- Source Wi-Fi → Connect: the board is found automatically.
- Allow Python on Private and Public networks when Windows asks.



Real horizontal EOG in EasyEXG Studio: raw (top), 0.1–10 Hz band-pass with 50 Hz notch (bottom)

Mode	Band-pass	JP1 / gain	Window	Extra reading
EEG	1–40 Hz	Bridged, 1 048×	5 s	Alpha (8–12 Hz) share of power
EOG	0.1–10 Hz	Open, 221×	10 s	–
EMG	20–240 Hz	Open, 221×	3 s	Activity line (moving RMS)
ECG	0.5–40 Hz	Bridged, 1 048×	5 s	Heart rate (bpm)

Studio setting	Default	Meaning
BOARD_IP	"auto"	Finds the board; put the Serial Monitor IP here if that fails
UDP_PORT	5005	Must match the sketch
VREF	5.0	ADC reference: 5.0 for UNO R4, 3.3 for ESP32
MAINS_HZ	50	50 or 60

In the window: change band-pass, notch (50 / 60 Hz), gain (must match JP1) and window length; start ● Record CSV; or pick Source → Demo to try it without a board. The status bar shows samples per second (≈ 500) and samples lost.

05 COMMAND-LINE VIEWER AND CSV FILES

easyexg_wifi_plot.py: signal and envelope

```
python easyexg_wifi_plot.py                          # finds the board
python easyexg_wifi_plot.py --host 192.168.1.57      # board IP from the Serial Monitor
python easyexg_wifi_plot.py --units uV --gain 1048 --csv ecg.csv
python easyexg_wifi_plot.py --vref 3.3               # ESP32
python easyexg_wifi_plot.py --demo                   # no board needed
```

CSV column	Meaning
time_s	Time from the start of the recording, in seconds
raw_uV	Raw signal, µV at the electrodes
filtered_uV	Signal after the band-pass and notch filters
mode	EEG, EOG, EMG or ECG
band_Hz	Band-pass used, e.g. 0.5-40
gain	Gain setting (221 or 1048)

06 CONVERT ADC COUNTS TO MV

```
uV = counts * VREF / (2**ADC_BITS) / GAIN * 1e6
# UNO R4, 10-bit, JP1 open    -> 1 count = 5.0 / 1024 / 221 = about 22 uV
# UNO R4, 14-bit, JP1 bridged -> 1 count = about 0.29 uV
# subtract the resting level (about half the ADC range) first
```

07 WI-FI PACKET FORMAT (LITTLE-ENDIAN)

Bytes	Type	Content
0–3	char[4]	"EXG1"
4–7	uint32	First sample index
8–9	uint16	Samples (normally 25)
10–11	uint16	Sample rate (Hz)
12	uint8	ADC bits
13–15	—	Padding
16...	float × 2	Signal, envelope per sample

Minimal Python receiver

```
BOARD = ("192.168.1.57", 5005)
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.settimeout(1.0)
HEADER = struct.Struct("<4sIHHB3x")
while True:
    sock.sendto(b"HELLO", BOARD)   # keep the stream alive
    try: data, _ = sock.recvfrom(2048)
    except socket.timeout: continue
    magic, first, count, fs, bits = HEADER.unpack_from(data)
    if magic != b"EXG1": continue
    s = struct.unpack_from(f"<{count*2}f", data, HEADER.size)
    signal = s[0::2]              # ADC counts, centred at 0
```

HELLO KEEP-ALIVE

The PC sends the text HELLO to the board's UDP port 5005 about once a second. The board streams to the last PC that said hello and stops 5 s after the hellos stop.