Петко Маринов

Ладоре ООД

Примерен код за кардиография чрез esp32

таймер, ацп(adc), филтри, i2c, websocket, canvas, etc.

Сигналът

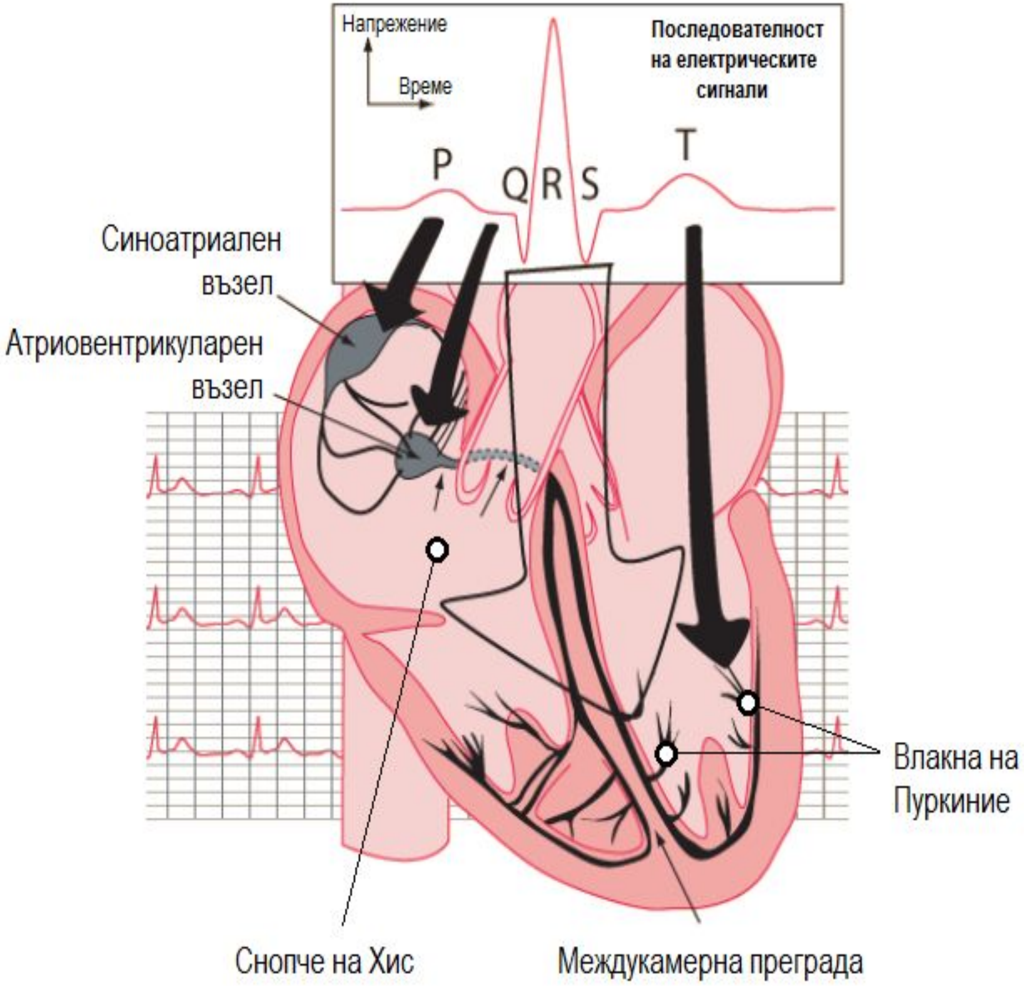
P - възбуждане на ляво и дясно предсърдие (деполяризация) (+) 0.07-0.11s

Q - възбуждане на интервентрикуларна преграда (-) 0.03s

R - камерна деполяризация (+)

S - отрицателна вълна (-) 0.06-0.10s

T - реполяризация (отпускане) на двете вентрикули (+) 0.12-0.28s



Усилвателя

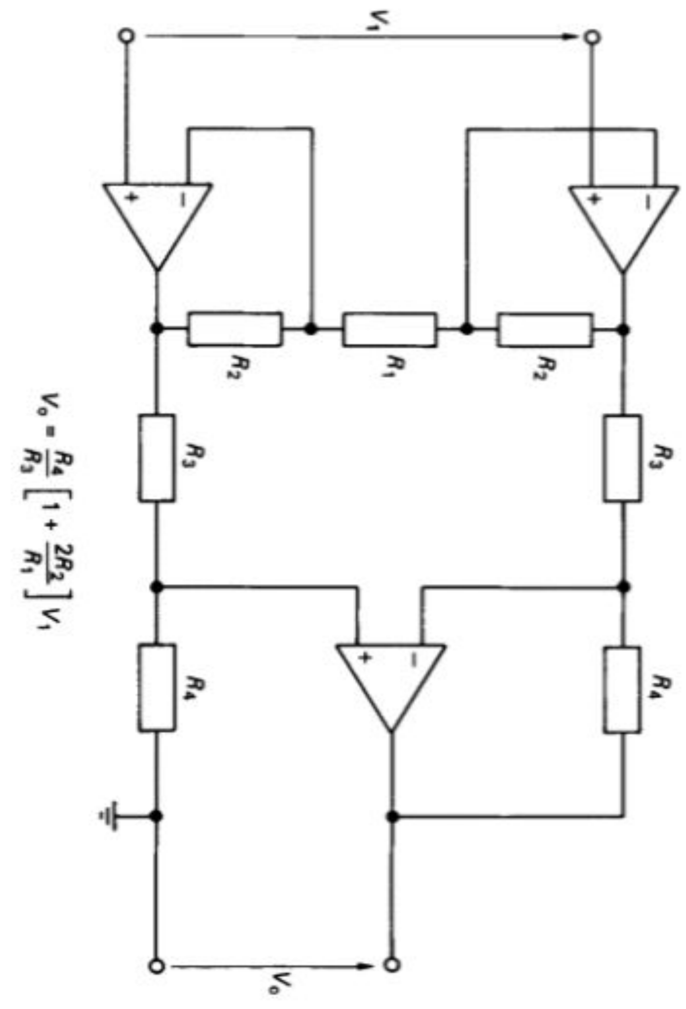
Два огледално разположени неинвертиращи операционни усилватели, чиято верига на обратната връзка се формира от резисторите R2 и R1.

- **Операционен**

  A1=1+2R2/R1

- **Инструментален (Диференциален)**

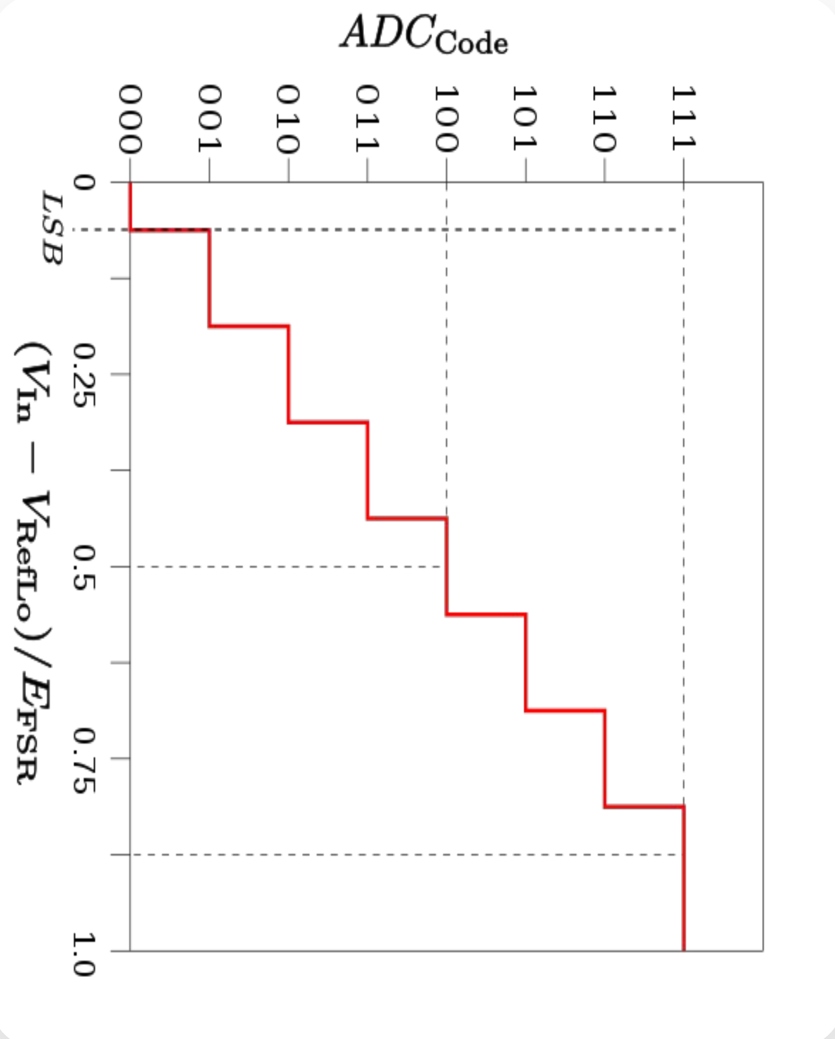
  A2=R4/R3



АЦП/ADC

- Честота на семплиране
- Теорема на Найкуист/Шанън:

“за да представим пълноценно сигнал с честотна лента в диапазона 0 - X [Hz], ни трябват поне 2X проби в секунда”



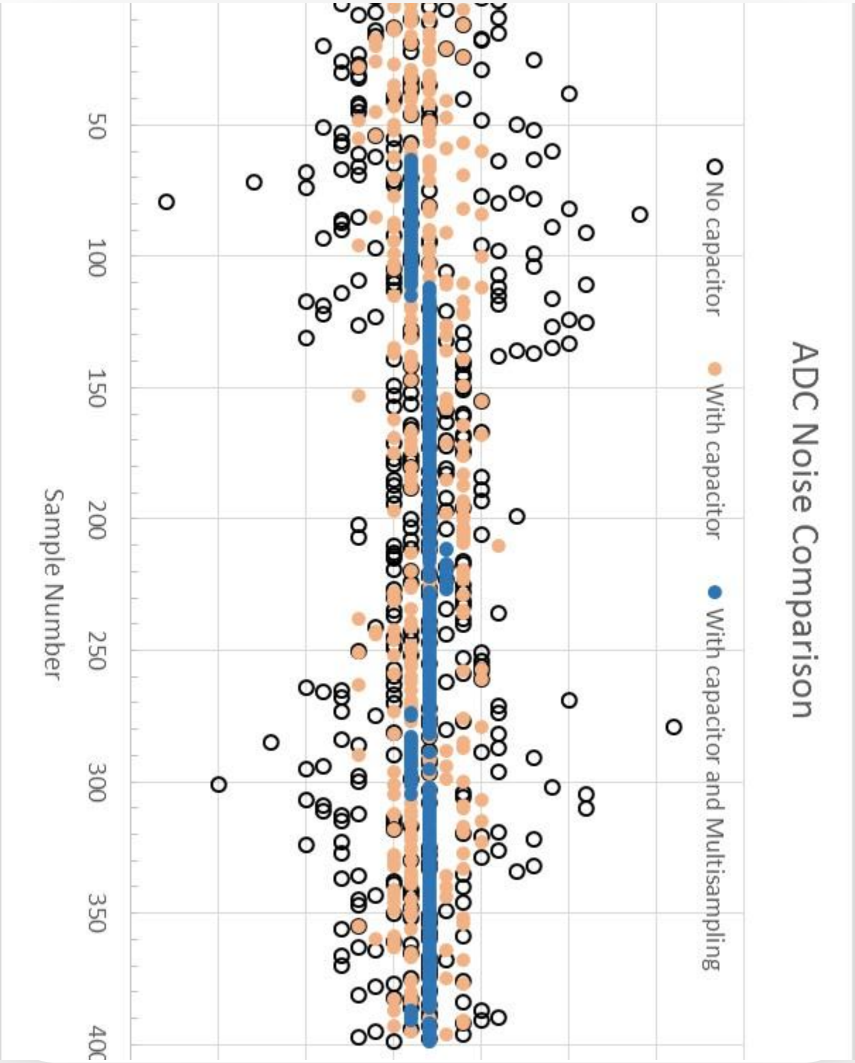
wikipedia:Analog-to-digital_converter

esp32 АЦП/ADC

- ADC1 (12bit) GPIO 32-39
- adc1_config_channel_atten
- adc1_config_width
- adc1_get_raw bugs (see documentation)

```
adc1_config_width(ADC_WIDTH_12Bit);

adc1_config_channel_atten(ADC1_CHANNEL_0,
ADC_ATTEN_DB_11);

int adcVal = adc1_get_raw(ADC1_CHANNEL_0);

uint8_t value = map(adcVal, 0 , 4096, 0, 255);
```

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/adc.html



Timer-esp32

- hw_timer_t * timerBegin(uint8_t timer, uint16_t divider, bool countUp);
- void timerAttachInterrupt(hw_timer_t *timer, void (*fn)(void), bool edge);
- void timerAlarmWrite(hw_timer_t *timer, uint64_t interruptAt, bool autoreload);
- void timerAlarmEnable(hw_timer_t *timer);

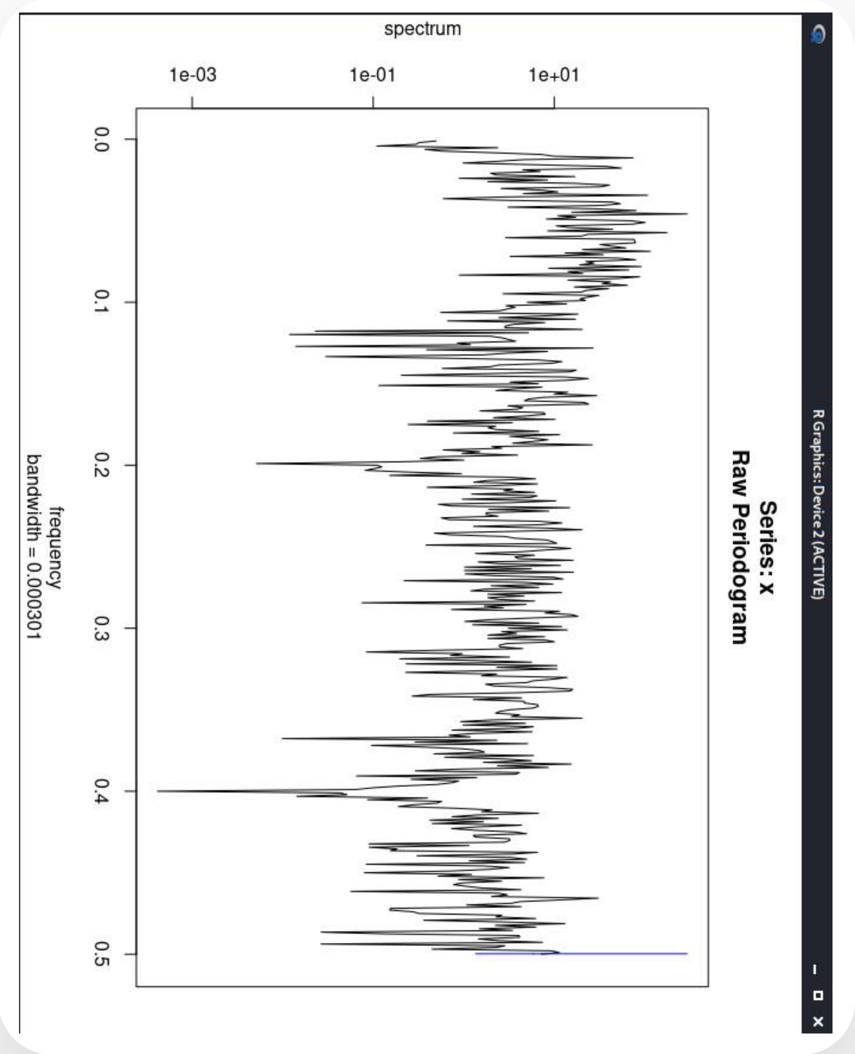
https://github.com/espressif/arduino-esp32/blob/master/cores/esp32/esp32-hal-timer.h

```
hw_timer_t * timer = NULL;
portMUX_TYPE timerMutex = portMUX_INITIALIZER_UNLOCKED;
int main(void) { //snip …
//…


 timer = timerBegin(0, 80, true); // 80 is the divider
 timerAttachInterrupt(timer, &onTimer, true);
 timerAlarmWrite(timer, 4000, true);//4000 == 250/second
 timerAlarmEnable(timer);
}
```

```
#include <driver/adc.h>

void IRAM_ATTR onTimer() {
  //portENTER_CRITICAL_ISR(&timerMutex);

  int adcVal = adc1_get_raw(ADC1_CHANNEL_0);
  uint8_t value = map(adcVal, 0 , 4096, 0, 255);
  ekgBuffer[bufferPointer++] = value;

  if (bufferPointer == EKG_BUFFER_MAX ) {
    bufferPointer = 0;
    transmit = true;
  }
  //portEXIT_CRITICAL_ISR(&timerMutex);
}
```

Шум

- източници
- анализ
- фрактали



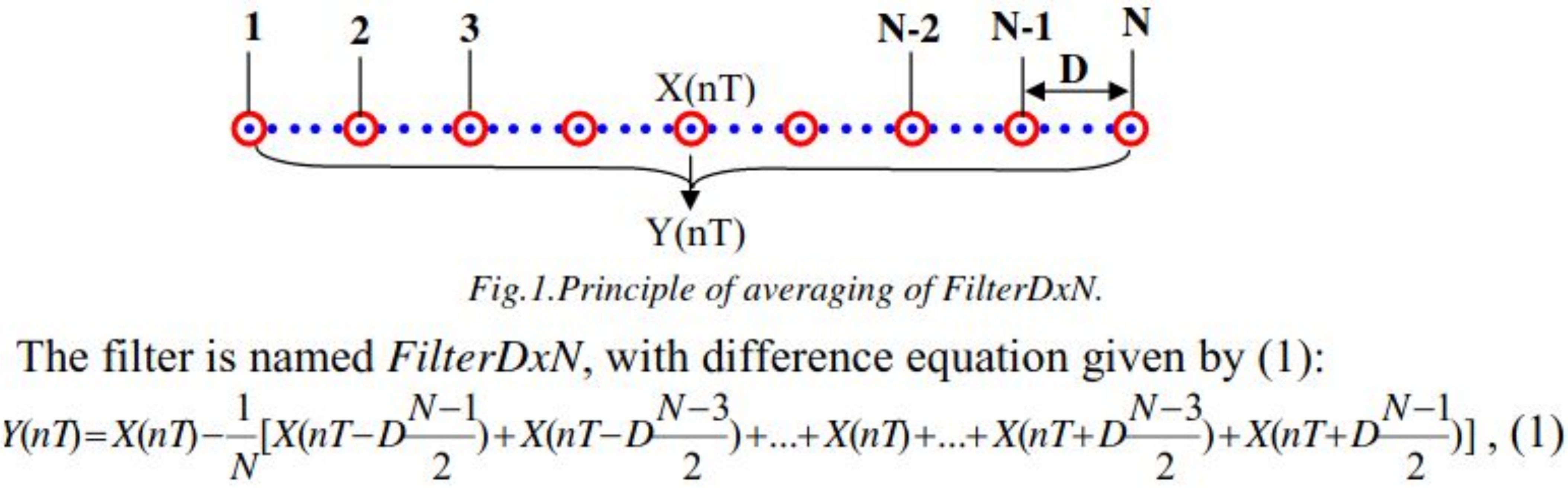
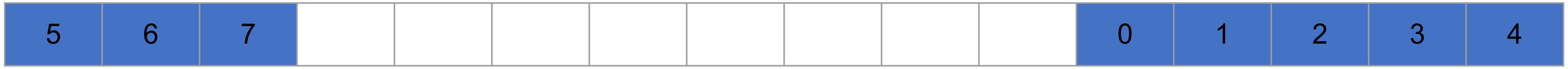


Fig.1.Principle of averaging of FilterDxN.

The filter is named *FilterDxN*, with difference equation given by (1):

$$Y(nT)=X(nT)-\frac{1}{N}[X(nT-D\frac{N-1}{2})+X(nT-D\frac{N-3}{2})+...+X(nT)+...+X(nT+D\frac{N-3}{2})+X(nT+D\frac{N-1}{2})]\ ,\ (1)$$

https://www.academia.edu/20095427/Combined_high-pass_and_power-line_interference_rejecter_filter_for_ecg_signal_processing

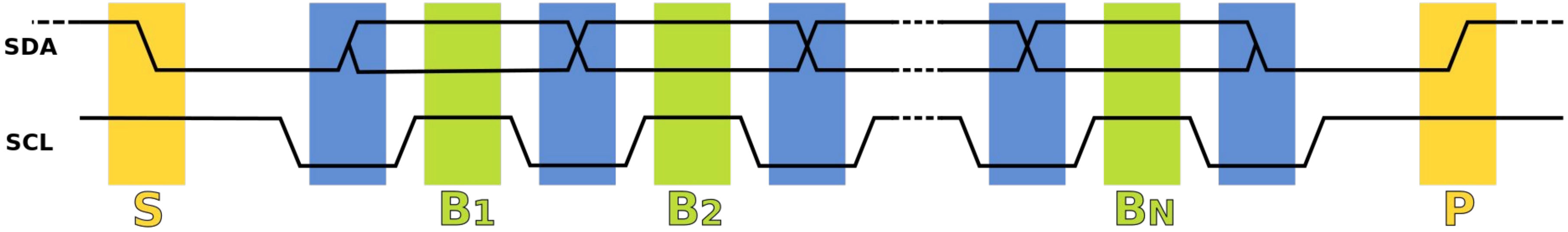


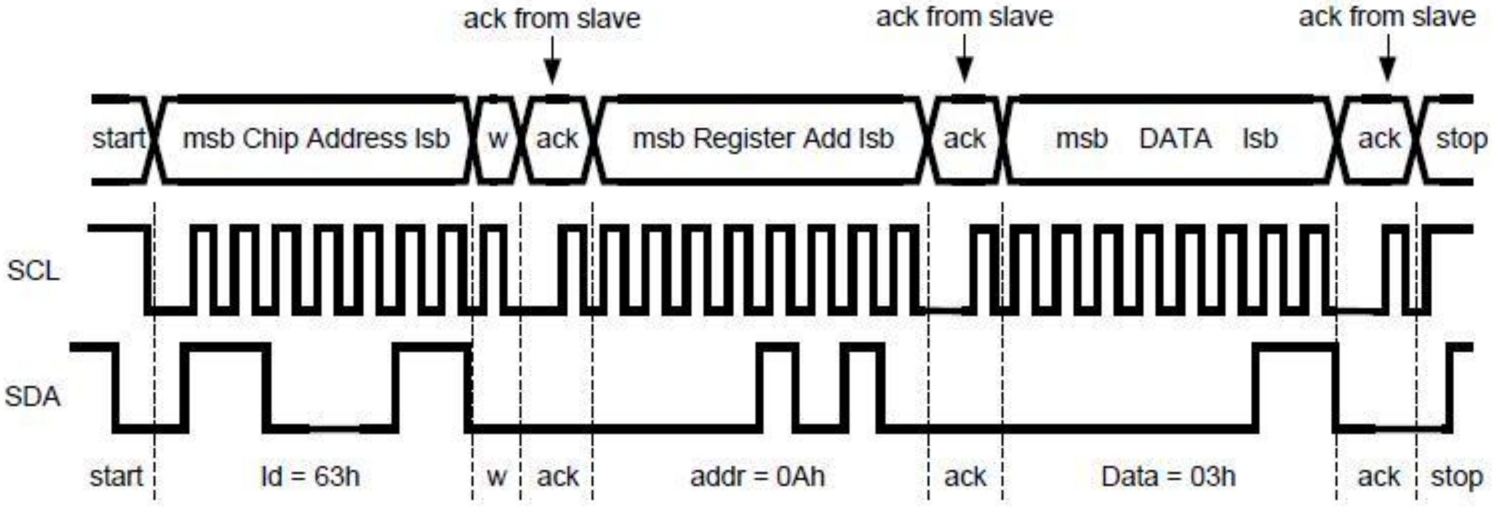
5	6	7									0	1	2	3	4

```
static int idx(int i, int n)
{
 return ((i % n) + n) % n;
}
```

```
for(int i = 0; i < EKG_BUFFER_MAX ; i++)
    {
      filterBuffer[i] = 255 - 128 + (ekgBuffer [i] -
      (  ekgBuffer [idx(i - 40,EKG_BUFFER_MAX )] + ekgBuffer [idx(i - 35,EKG_BUFFER_MAX )]
      + ekgBuffer [idx(i - 30,EKG_BUFFER_MAX )] + ekgBuffer [idx(i - 25,EKG_BUFFER_MAX )]
      + ekgBuffer [idx(i - 20,EKG_BUFFER_MAX )] + ekgBuffer [idx(i - 15,EKG_BUFFER_MAX )]
      + ekgBuffer [idx(i - 10,EKG_BUFFER_MAX )] + ekgBuffer [idx(i -  5,EKG_BUFFER_MAX )]
      + ekgBuffer [i]
      + ekgBuffer [idx(i +  5,EKG_BUFFER_MAX )] + ekgBuffer [idx(i + 10,EKG_BUFFER_MAX )]
      + ekgBuffer [idx(i  - 15,EKG_BUFFER_MAX )] + ekgBuffer [idx(i + 20,EKG_BUFFER_MAX )]
      + ekgBuffer [idx(i + 25,EKG_BUFFER_MAX )] + ekgBuffer [idx(i + 30,EKG_BUFFER_MAX )]
      + ekgBuffer [idx(i + 35,EKG_BUFFER_MAX )] + ekgBuffer [idx(i + 40,EKG_BUFFER_MAX )]
      )
      /17);
    }
```

-40	-35	-30	-25	-20	-15	-10	-5	0	5	10	15	20	25	30	35	40





wikipedia:I²C

```
#include <Wire.h>
#include <max86150.h>

#define I2C_SDA 19
#define I2C_SCL 18
TwoWire I2C6050 = TwoWire(0);
...
MAX86150 max86150Sensor;
uint16_t ppgunsigned16;
...
pinMode (I2C_SDA,OUTPUT);
pinMode (I2C_SCL,OUTPUT);
...
status = I2C6050.begin(I2C_SDA,I2C_SCL, 10000);
...
```

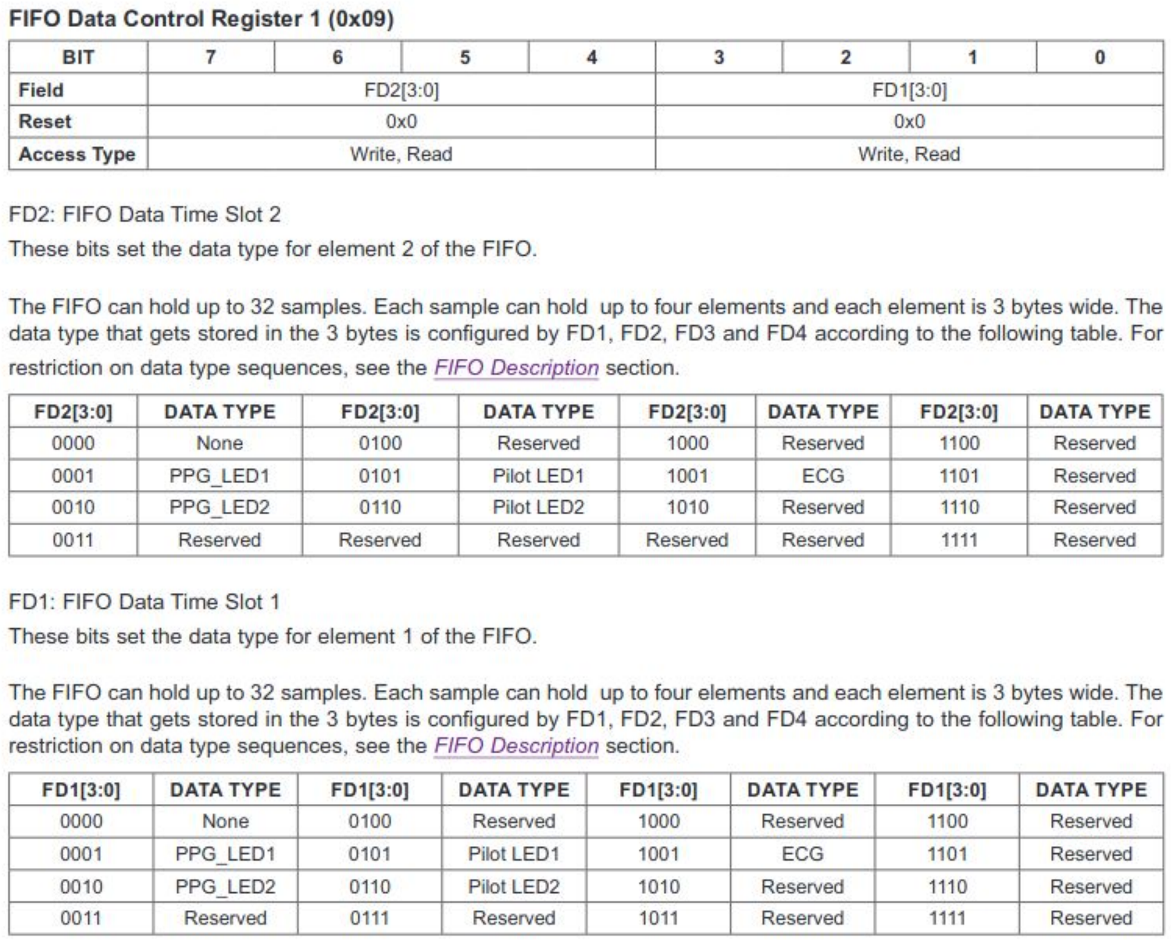
https://datasheets.maximintegrated.com/en/ds/MAX86150.pdf

```
for (byte i = 8; i < 120; i++) {
        I2C6050.beginTransmission (i);
        if (I2C6050.endTransmission () == 0) { //found device
        }
}

if (max86150Sensor.begin(I2C6050, 10000,0x5e) == false)
{
  Serial.println("MAX86150 was not found. Please check wiring/power. ");
}else {
  Serial.println("partid:" + max86150Sensor.readPartID());
  max86150Sensor.setup();
}
```

https://github.com/Protocentral/protocentral_max86150_ecg_ppg

```
writeRegister8(
        _i2caddr,
        MAX86150_FIFOCONTROL1,
        (0b00010010));
```

0001 PPG_LED1
0010 PPG_LED2



FIFO Data Control Register 1 (0x09)

BIT	7	6	5	4	3	2	1	0
Field	FD2[3:0]				FD1[3:0]			
Reset	0x0				0x0			
Access Type	Write, Read				Write, Read			

FD2: FIFO Data Time Slot 2

These bits set the data type for element 2 of the FIFO.

The FIFO can hold up to 32 samples. Each sample can hold up to four elements and each element is 3 bytes wide. The data type that gets stored in the 3 bytes is configured by FD1, FD2, FD3 and FD4 according to the following table. For restriction on data type sequences, see the *FIFO Description* section.

FD2[3:0]	DATA TYPE	FD2[3:0]	DATA TYPE	FD2[3:0]	DATA TYPE	FD2[3:0]	DATA TYPE
0000	None	0100	Reserved	1000	Reserved	1100	Reserved
0001	PPG_LED1	0101	Pilot LED1	1001	ECG	1101	Reserved
0010	PPG_LED2	0110	Pilot LED2	1010	Reserved	1110	Reserved
0011	Reserved	Reserved	Reserved	Reserved	Reserved	1111	Reserved

FD1: FIFO Data Time Slot 1

These bits set the data type for element 1 of the FIFO.

The FIFO can hold up to 32 samples. Each sample can hold up to four elements and each element is 3 bytes wide. The data type that gets stored in the 3 bytes is configured by FD1, FD2, FD3 and FD4 according to the following table. For restriction on data type sequences, see the *FIFO Description* section.

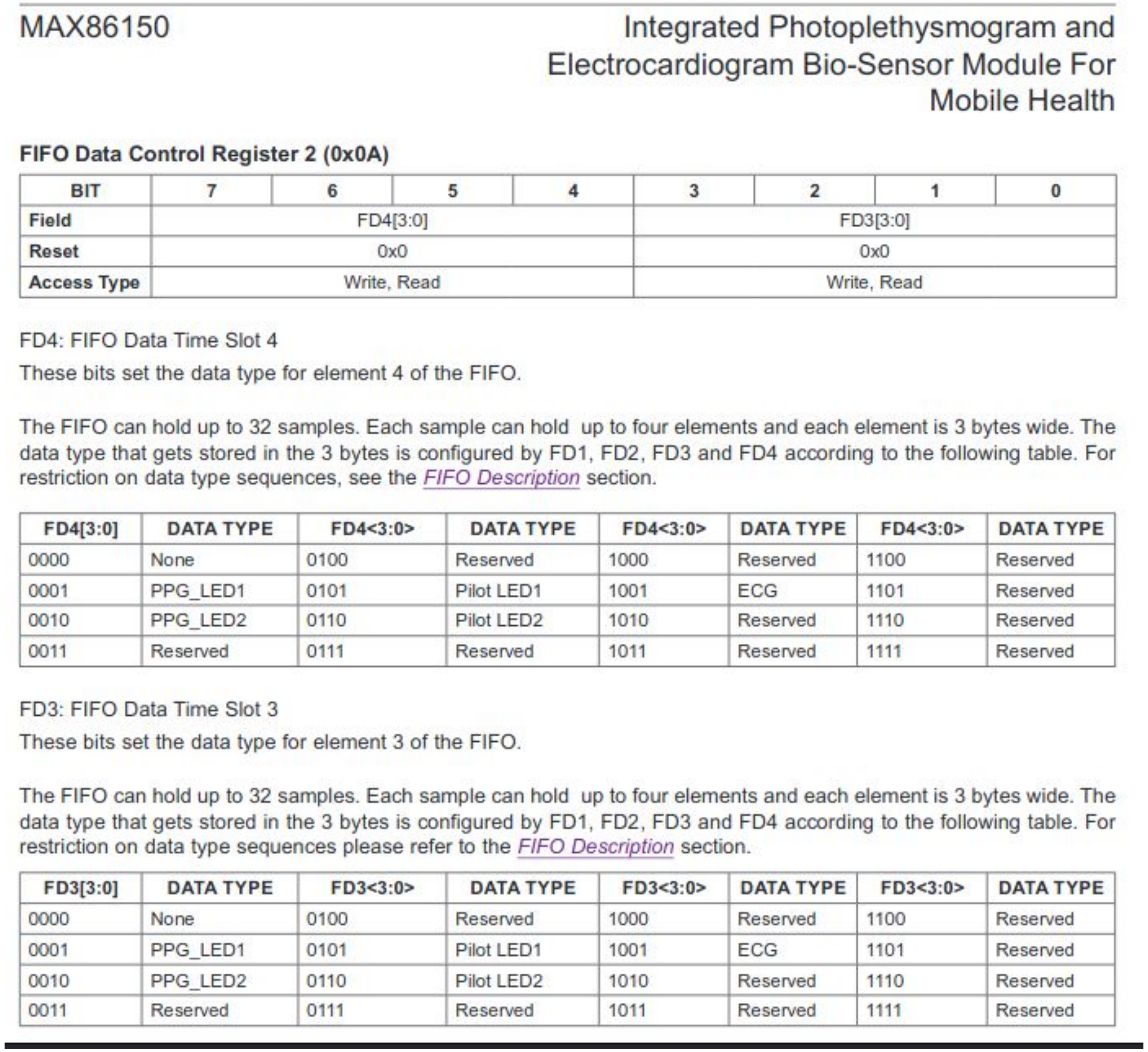
FD1[3:0]	DATA TYPE	FD1[3:0]	DATA TYPE	FD1[3:0]	DATA TYPE	FD1[3:0]	DATA TYPE
0000	None	0100	Reserved	1000	Reserved	1100	Reserved
0001	PPG_LED1	0101	Pilot LED1	1001	ECG	1101	Reserved
0010	PPG_LED2	0110	Pilot LED2	1010	Reserved	1110	Reserved
0011	Reserved	0111	Reserved	1011	Reserved	1111	Reserved

max86150.cpp

```
writeRegister8(
        _i2caddr,
        MAX86150_FIFOCONTROL2,
        (0b00010010));
```

0001 PPG_LED1
0010 PPG_LED2

max86150.cpp



MAX86150 Integrated Photoplethysmogram and Electrocardiogram Bio-Sensor Module For Mobile Health

FIFO Data Control Register 2 (0x0A)

BIT	7	6	5	4	3	2	1	0
Field	FD4[3:0]				FD3[3:0]			
Reset	0x0				0x0			
Access Type	Write, Read				Write, Read			

FD4: FIFO Data Time Slot 4

These bits set the data type for element 4 of the FIFO.

The FIFO can hold up to 32 samples. Each sample can hold up to four elements and each element is 3 bytes wide. The data type that gets stored in the 3 bytes is configured by FD1, FD2, FD3 and FD4 according to the following table. For restriction on data type sequences, see the *FIFO Description* section.

FD4[3:0]	DATA TYPE	FD4<3:0>	DATA TYPE	FD4<3:0>	DATA TYPE	FD4<3:0>	DATA TYPE
0000	None	0100	Reserved	1000	Reserved	1100	Reserved
0001	PPG_LED1	0101	Pilot LED1	1001	ECG	1101	Reserved
0010	PPG_LED2	0110	Pilot LED2	1010	Reserved	1110	Reserved
0011	Reserved	0111	Reserved	1011	Reserved	1111	Reserved

FD3: FIFO Data Time Slot 3

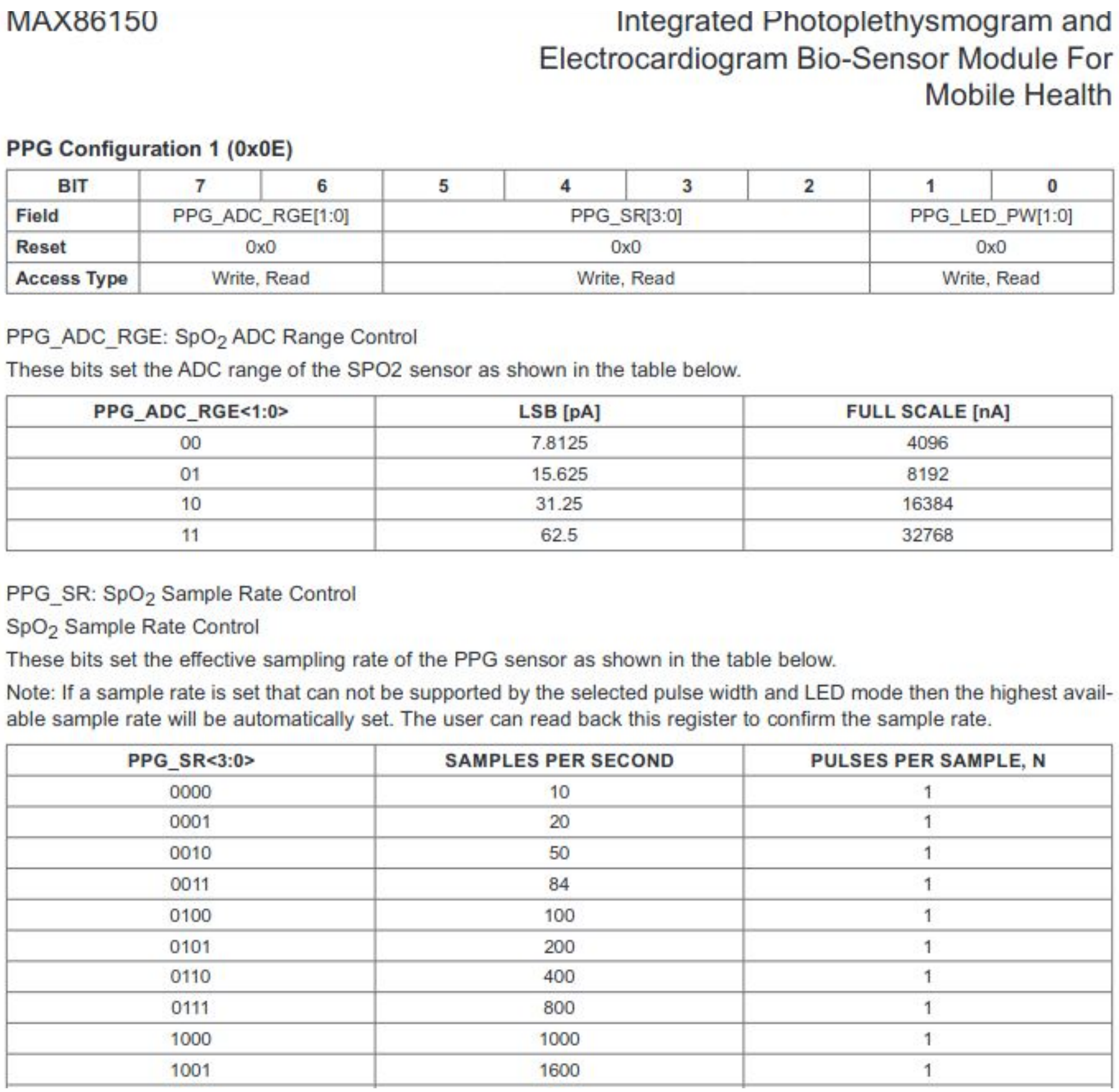
These bits set the data type for element 3 of the FIFO.

The FIFO can hold up to 32 samples. Each sample can hold up to four elements and each element is 3 bytes wide. The data type that gets stored in the 3 bytes is configured by FD1, FD2, FD3 and FD4 according to the following table. For restriction on data type sequences please refer to the *FIFO Description* section.

FD3[3:0]	DATA TYPE	FD3<3:0>	DATA TYPE	FD3<3:0>	DATA TYPE	FD3<3:0>	DATA TYPE
0000	None	0100	Reserved	1000	Reserved	1100	Reserved
0001	PPG_LED1	0101	Pilot LED1	1001	ECG	1101	Reserved
0010	PPG_LED2	0110	Pilot LED2	1010	Reserved	1110	Reserved
0011	Reserved	0111	Reserved	1011	Reserved	1111	Reserved

```
writeRegister8(
        _i2caddr,
        MAX86150_PPGCONFIG1,
        0b11001000);

PPG_ADC 11 //11 62.5 32768
PPG_SR 0010 //50
PPG_LED_PW 00 //50 pulse width
```

max86150.cpp



MAX86150 Integrated Photoplethysmogram and Electrocardiogram Bio-Sensor Module For Mobile Health

PPG Configuration 1 (0x0E)

BIT	7	6	5	4	3	2	1	0
Field	PPG_ADC_RGE[1:0]		PPG_SR[3:0]				PPG_LED_PW[1:0]	
Reset	0x0		0x0				0x0	
Access Type	Write, Read		Write, Read				Write, Read	

PPG_ADC_RGE: SpO_2 ADC Range Control

These bits set the ADC range of the SPO2 sensor as shown in the table below.

PPG_ADC_RGE<1:0>	LSB [pA]	FULL SCALE [nA]
00	7.8125	4096
01	15.625	8192
10	31.25	16384
11	62.5	32768

PPG_SR: SpO_2 Sample Rate Control

SpO_2 Sample Rate Control

These bits set the effective sampling rate of the PPG sensor as shown in the table below.

Note: If a sample rate is set that can not be supported by the selected pulse width and LED mode then the highest available sample rate will be automatically set. The user can read back this register to confirm the sample rate.

PPG_SR<3:0>	SAMPLES PER SECOND	PULSES PER SAMPLE, N
0000	10	1
0001	20	1
0010	50	1
0011	84	1
0100	100	1
0101	200	1
0110	400	1
0111	800	1
1000	1000	1
1001	1600	1

```
if(max86150Sensor.check()>0)
   {
    ppgunsigned16 = (uint16_t) (max86150Sensor.getFIFORed()>>2);
    Serial.print("PPG:");
    Serial.println(ppgunsigned16);
   }
```

https://github.com/Protocentral/protocentral_max86150_ecg_ppg

```
#include <WebServer.h>
#include <WebSocketsServer.h>

WebServer server(80);
WebSocketsServer webSocket = WebSocketsServer(8080);

...

webSocket.begin();
webSocket.onEvent(webSocketEvent);

server.on("/", handle_OnConnect);
server.onNotFound(handle_NotFound);

server.begin();
```

```
void webSocketEvent(uint8_t num, WStype_t type, uint8_t * payload, size_t length) {

  switch(type) {
    case WStype_DISCONNECTED:
      break;
    case WStype_CONNECTED:
      IPAddress ip = webSocket.remoteIP(num);
      webSocket.sendTXT(num, "{\"connected\":true}");
      break;
    case WStype_TEXT:
      Serial.printf("[%u] get Text: %s\n", num, payload);
      break;
     case WStype_ERROR:
      Serial.printf("WS_ERROR");
      break;
    }
}
```

```
void handle_OnConnect() {
  Serial.println("handle on connect routine");
  server.send(200, "text/html", SendHTML());
}

String SendHTML(){
  String ptr = "<!DOCTYPE html> <html>\n";
  ....
  ptr +="<canvas id='eeg' width='1000' height='255'></canvas>";
  ...
  ptr +="</html>\n";
  return ptr;
}
```

```
ptr +="<script>";
ptr +="var connection = new WebSocket('ws://' + location.hostname + ':8080/', ['arduino']);";
ptr +="connection.onopen = function () {";
  ...
ptr +="connection.onerror = function (error) {";
...
ptr +="connection.onmessage = function (e) {";
ptr +="var msg = JSON.parse(e.data);";
ptr +="console.log(msg.type);";
ptr +="if(msg.type == \"ppg\") { document.getElementById(\"ppg\").innerHTML = msg.data +
'%' ;};";
ptr +="if(msg.type == \"ecg\") {";
ptr += "data = msg.data.split(';');";
ptr += "var canvas = document.getElementById('eeg');";
ptr += "var ctx = canvas.getContext('2d');";
ptr += "ctx.beginPath();";
...
```

```
ptr += "for(i=0;i<data.length - 1;i++) {";
ptr += "ctx.lineTo(i,data[i]);";
ptr += "ctx.stroke();";
...
```

Демо

Благодаря!

info@tregatti.tech

Thank you!

Contacts:

- Linkedin profile
- Facebook profile
- Github profile
- Instagram profile

СЛЕДВАЩО СЪБИТИЕ

Лектор

Дата

Език