Electronic Supplementary Material (ESI) for Lab on a Chip.

Supporting Information

Automated Device for Multi-Stage Paper-Based Assays enabled by an Electroosmotic Pumping Valve

Baruch Rofman[1], Rawi Naddaf[2], Maya Bar-Dolev[1], Tal Gefen[2], Nadav Ben-Assa[2], Naama Geva-Zatorsky[2,3,*], Moran Bercovici[1,*]

[1] Faculty of Mechanical Engineering, Technion − Israel Institute of Technology, Haifa, 3200003, Israel
[2] Rappaport Technion Integrated Cancer Center (RTICC), Department of Cell Biology and Cancer Science, Rappaport Faculty of Medicine, Technion – Israel Institute of Technology, Haifa, 3200003, Israel
[3]Canadian Institute for Advanced Research (CIFAR), Toronto, ON, Canada

**Corresponding authors:* mberco@technion.ac.il, naama_gz@technion.ac.il

Table of contents

Section S1 – Pinning effect at the vents

Figure S1 presents the measured minimal electrical potential required to burst the valve with and without vents in the cover, using 5 μm pitch superhydrophobic surfaces (which are always stable, as per Figure 2 in the manuscript). We used equation (4) from the manuscript to translate the measured electrical potential to the pressure at the gap, and from the difference between the two values we estimate the pinning pressure to be $\Delta P_{pin} = 107\ Pa$.

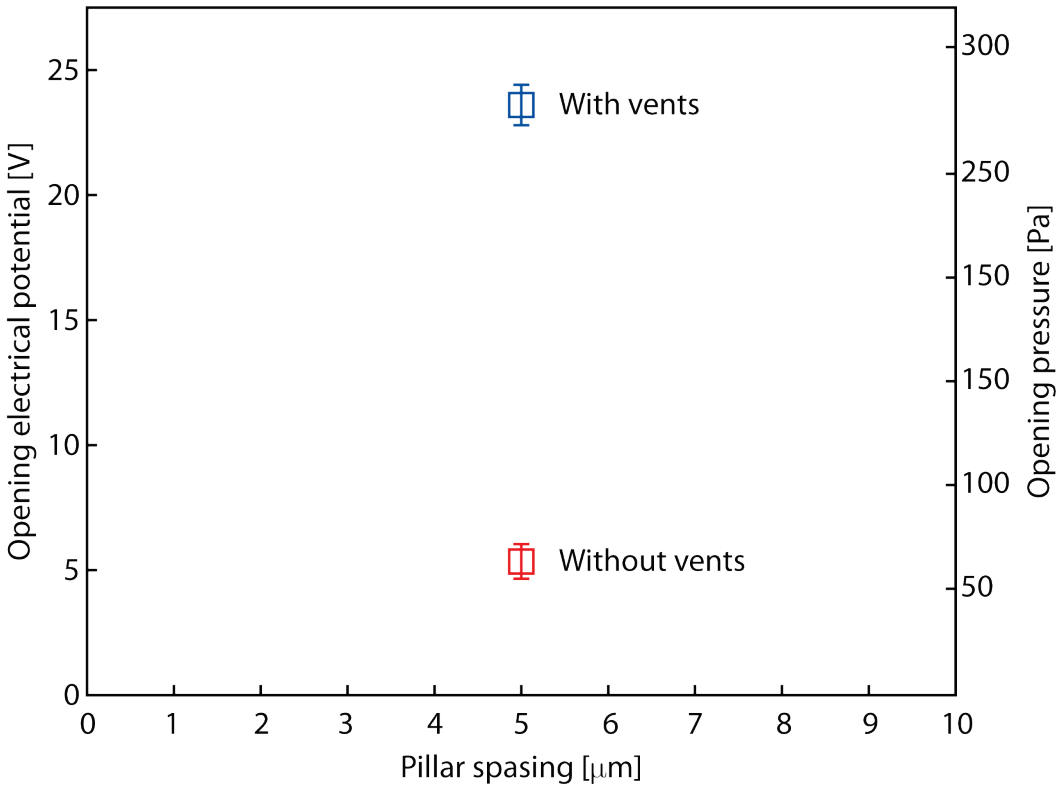

Figure S1. *Experimental results of the minimal electric potential (left y-axis) and the calculated corresponding pressure at the gap (right y-axis) required to burst an EOPV based on a* 5 μm *superhydrophobic surface. Error bars represent 95% confidence on the mean for both axes, based on five repeats.*

Section S2 – 1D forced imbibition in a surface with a varying cross-section

In order to describe the flow in the target pad driven by the EOPV operating in the continuous mode, we expand on the approach by Elizalde et. al.[1] for analysis of natural imbibition in arbitrarily shaped pads, and consider the case of forced imbibition. and develop a generalized model for forced imbibition. Assuming that our flow is incompressible and that the pad can be modeled as an isotropic porous media we can describe the flow using the continuity equation and Darcy's law[2],

$$\boldsymbol{\nabla} \cdot \boldsymbol{v_D} = 0, \qquad v_D = -\frac{k}{\eta}\boldsymbol{\nabla} p, \tag{S1}$$

where we assume k to be the isotropic and constant permeability of the pad and v_D is the Darcy velocity in the porous media. The permeability can be estimated using the bundle of tube model[3] or the Carman-Kozeny model[4,5], or be measured experimentally. For a pad with a varying width, where the liquid is assumed to span over the entire cross-section, and assuming that the flow velocity is uniform at each cross section, we can consider the one-dimensional form of equation (S1), $\frac{d}{dx}(\boldsymbol{v_D} \cdot \boldsymbol{A}) = 0$, $v_D = -\frac{k}{\eta}\frac{dp}{dx}$, where $A(x)$ is the cross-sectional area at each axial coordinate, x (along the flow direction). The flow rate, can then be obtained from integrating the mass conservation equation, as $Q(t) = v_D(x,t)A(x)$. Substituting the velocity into the Darcy relation and integrating yields

$$Q = -\frac{k}{\eta}\frac{(P_C - P_{EO})}{\int_0^{l_T}\frac{dx}{A(x)}}, \tag{S2}$$

where P_{EO} is the electroosmotic pressure generated by the pump and P_C is the capillary pressure at the advancing liquid pad, and l_T is the apparent penetration length.

As we showed in the manuscript, the flow rate generated by the EOPV is given by equation (3)

$$Q_{EOPV} = \frac{1}{\mu}\frac{w_p h_P}{L_p}\frac{\psi_p}{\tau_p}\frac{a_p^2}{8}\left(-\Delta P - \varepsilon\zeta\Delta V\frac{8f}{a_p^2}\right), \tag{S3}$$

Denoting

$$G_p \equiv \frac{w_p h_P}{L_p}\frac{\psi_p}{\tau_p}\frac{a_p^2}{8}[m^3] \quad ; \quad G_e \equiv \frac{8f}{a_p^2}[m^{-2}], \tag{S4}$$

Equation (S3) can be compacted to the form $Q_{EOPV} = \frac{G_p}{\mu}(-\Delta P - \varepsilon\zeta\Delta V G_e)$, where G_p and G_e are constants for a given system.

For a stable liquid bridge, the flow rate into the gap generated by the EOPV is equal to the flow rate exiting the gap into the target pad, i.e. $Q_{EOPV} = Q$, and the pressure the EOPV works against is precisely the pressure that drives the liquid in the target pad (P_{EO}), i.e. $\Delta P = P_{EO} - P_g$. Equating equations $(S2)$ to $(S3)$ yields

$$P_{EO}(l_T) = \frac{G(l_T)\left[P_g - \varepsilon\zeta\Delta V G_e\right] + P_C}{1 + G(l_T)}, \tag{S5}$$

where

$$G(l_T) \equiv \frac{G_p}{k_T}\int_0^{l_T}\frac{dx}{A(x)} \qquad (S6)$$

is a purely geometrical coefficient that combines both the properties of the pumping and target pads and varies with the imbibition time due to variation in the cross-section. The resulting forced flow rate is therefore

$$Q = \frac{G_p}{\eta}\left(\frac{P_g-(P_C+\varepsilon\zeta\Delta V G_e)}{1+G(l_T)}\right). \qquad (S7)$$

By substituting the flow rate $(S7)$ into equation $(S2)$ and using the relation $v_D(x) = dl_T/dt$, we can write an integral equation for the penetration distance as a function of time

$$\int_0^{l_T}\left(A_T(\xi\,) + \frac{G_p}{k_T}A(\xi\,)\int_0^{\xi}\frac{dx}{A(x)}\right)d\xi = \frac{G_p\big(P_g-(P_C+\varepsilon\zeta\Delta V G_e)\big)}{\eta}t. \qquad (S8)$$

The overall supplied liquid volume is then

$$\mathcal{V}(t) = \int_0^t Q_T dt = \int_0^{l_T(t)} A(x)dx\,. \qquad (S9)$$

We obtained the same form of a relation between the penetration distance and time as was described before in Elizalde et. al.[1], with the forced component acting only as a proportionality factor. For a constant area (rectangular strip) and after a sufficiently long time, the supplied liquid volume scales as $t^{1/2}$. Similar analysis can be done in a polar coordinate system. Under the assumption of axi-symmetry, the governing equation (S1) will reduce to the same 1D form, $\frac{d}{dx}(\boldsymbol{v_D}\cdot\boldsymbol{A}) = 0$, $v_D = -\frac{k}{\eta}\frac{dp}{dx}$, where x is now the radial coordinate. Thus, the case of a pad that that the form of a circular sector (i.e. has a constant angle between its edges) can be analyzed using the same resulting equations (S9), where $A(x)$ represents a cross-section at a certain radial coordinate. For this case, the penetration length is a linear function of time.[1]

Section S3 – Effect of zeta potential and pH on the relation between pressure and electric potential

In the manuscript, we use equation (4) to translate measured voltage to pressure at the gap. In figure 2c of the manuscript we use a specific set of conditions ($\zeta = -20$ mV, $\lambda = 100$ nm) and deduce that the burst mechanism is based on breaking the stability of the superhydrophobic surface rather than overcoming the Laplace pressure at the gap. Since the pumped liquid is DI, its exact pH and thus its λ_d value, are very sensitive and typically ranges from 5 to 7. The zeta potential of the pad can also vary substantially, typically in the range of -10 mV to -60 mV.[6] Figure S2 replicates Figure 2c of the main manuscript, using different zeta potentials and different pH values. While the values of the pressure change substantially with the change in these parameters, the conclusion remains unchanged across the entire parameter space – the measured burst pressure never reaches the Laplace pressure in the gap.

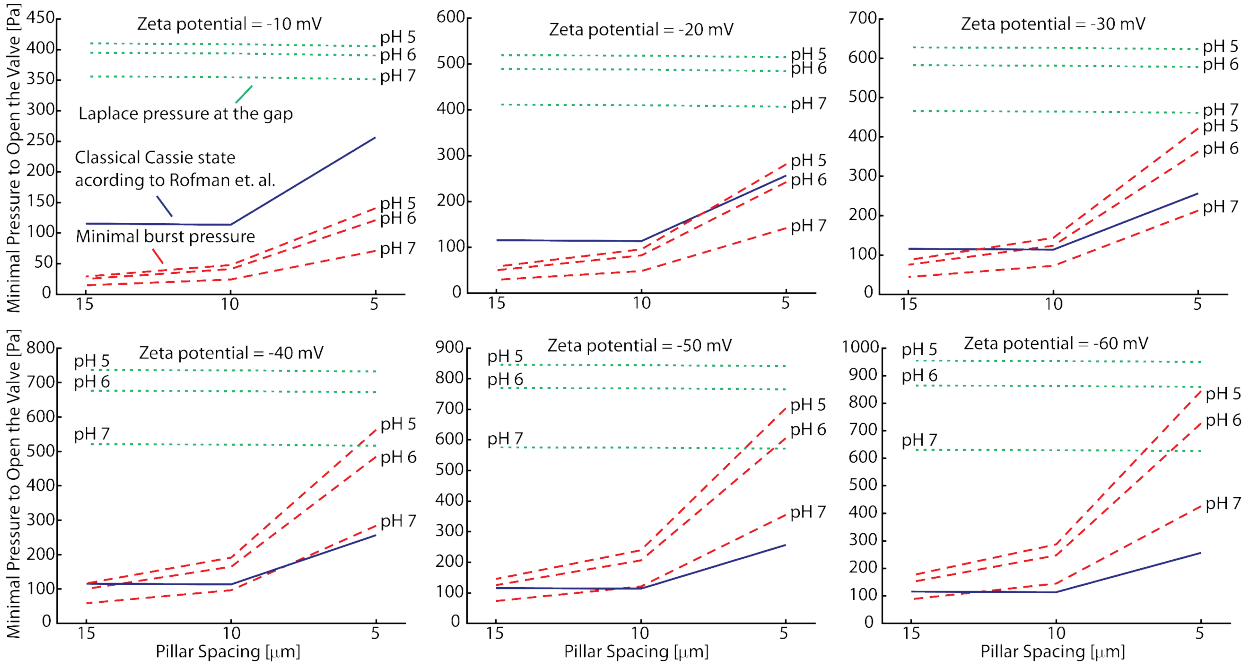

Figure S2. *The minimal burst pressure and the Laplace pressure at the gap, as a function of the pillar spacing, for various values of ζ and pH. In all cases the measured burst pressure is significantly lower than the Laplace pressure, indicating that the valve bursts due to failure of the superhydrophobic surface.*

Section S4 – Electrical and mechanical PCB Design and consideration

Figure S3 presents the component layout of the functional, logical and power PCBs. The functional PCB, and the microfluid chamber on top of it, are separate and replaceable units from the power and logic stack, connected by push-in connectors. Our design guideline was to keep manufacturing components costs at mass production to a minimum while supporting partial disposability of the functional element if need be, e.g. to prevent cross-contamination between tests. Table S1 provides a break-down of the components costs for a single unit in a 100 M units batch, based on available pricing from DigiKey.com, showing that the total cost will be about $6.

The logic PCB is mainly dedicated to the controller chip, while the power PCB is mainly devoted to the two DC-DC converters used to transform the 5V-2A input into 100V-0.1A at the pumping electrodes. We outsourced the manufacturing of the PCBs to a standard supplier (PCBWay, China) and hand soldered the components to the boards ourselves. The power and logic PCBs are two-layered PCBs while the functional PCB is composed of four layers.

The driving electrodes of the EOPV were two exposed conductors on the PCBs that were produced by the standard method of electrolytic copper deposition followed by electroless nickel plating. However, over multiple actuations at a high voltage (100V) the electrode reaction degrades both the nickel and the copper, eventually resulting in a disconnect. Therefore, the devices used for the detection assay were also finished with gold immersion. With gold plating, the same set of electrodes could be used more than 20 times without any visible degradation, and with only nickel plating — at least five times. For a disposable device, this is clearly sufficient. For a device which is to be used repeatedly, it may be beneficial to consider the use of platinum electrodes, though production costs would increase significantly.

The heating elements were insulated resistors formed by conductor lines printed on the two internal layers of the functional PCB. Due to the high sensitivity of such resistors to their length and the length of the conductors leading to them, it is difficult to ensure the same current supply to all resistors by design. We therefore placed a temperature sensor (thermistor) on the backside of each heating element and controlled it individually. For every new functional PCB, we calibrated the temperature sensors to correctly represent the top side temperature.

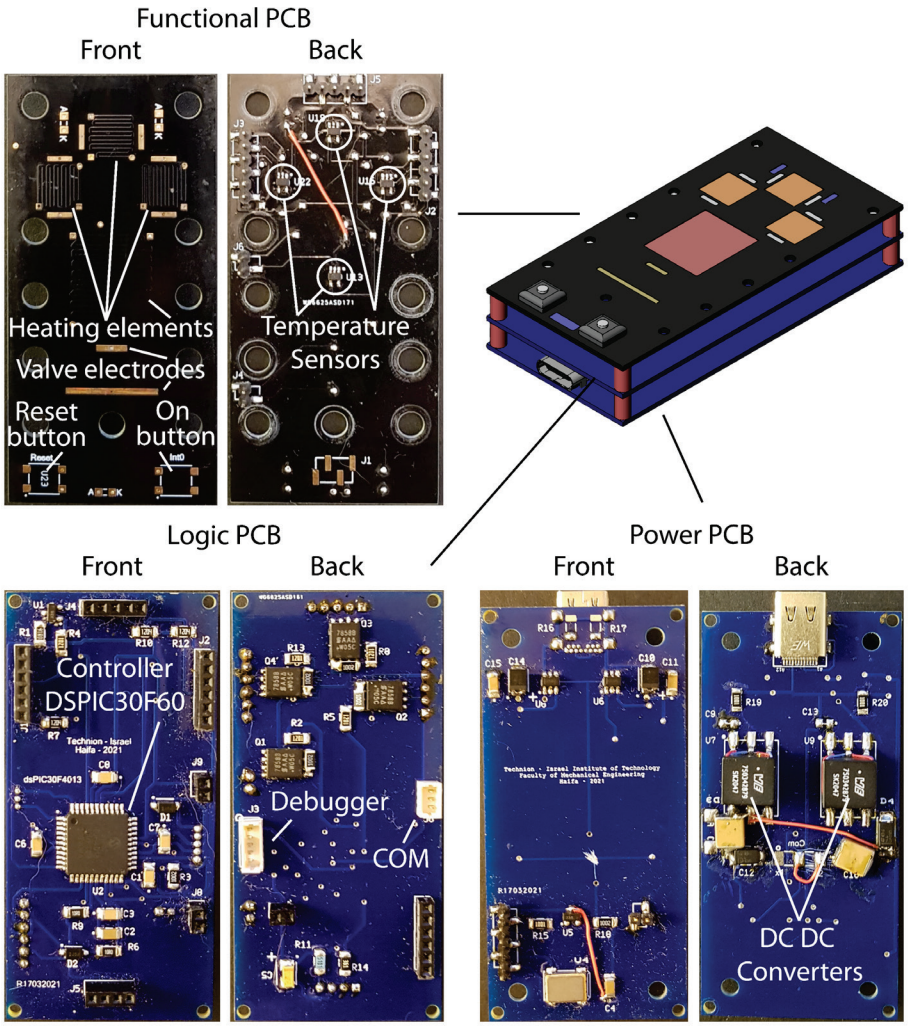

Figure S3. *The layout of the components on the PCBs used in the device. The functional PCB on the top left has the heating elements, driving electrodes and user interface on the front, and temperature sensors on the back. The logic PCB contains a Microchip controller and connectors for communication for the debugger used for burning new code. The power PCB has two DC-DC converters used to raise the 5V/2A input to 100V/0.1A at the pumping electrodes.*

Table S1*. Cost estimation for the components in a single device under mass production*

Electrical Components - for 100 million units						
Index	**Quantity**	**Part Number**	**Manufacturer Part Number**	**Description**	**Unit Price [$]**	**Extended Price [$]**
1	150000000	RNCP1206FTD1K00TR-ND	RNCP1206FTD1K00	RES 1K OHM 1% 1/2W 1206	0.00993	1,489,200
2	100000000	RNCP1206FTD15K0TR-ND	RNCP1206FTD15K0	RES 15K OHM 1% 1/2W 1206	0.00993	992,800
3	100000000	RHM200KAITR-ND	KTR18EZPF2003	RES SMD 200K OHM 1% 1/4W 1206	0.02415	2,415,000
4	200000000	1276-1017-2-ND	CL31B104KBCNNNC	CAP CER 0.1UF 50V X7R 1206	0.02072	4,144,000
5	50000000	478-1542-2-ND	12065C103KAT2A	CAP CER 10000PF 50V X7R 1206	0.05249	2,624,250
6	100000000	A129786TR-ND	CRGCQ1206F10R	CRGCQ 1206 10R 1%	0.01024	1,024,400

7	10000000	DSPIC30F601530IPT-ND	DSPIC30F6015-30I/PT	IC MCU 16BIT 144KB FLASH 64TQFP	12.8552	128,552,000
8	10000000	732-8220-2-ND	632723300021	CONN RCP USB3.1 TYPEC 24P SMD RA	3.64503	36,450,250
9	100000000	311-1.20MFRTR-ND	RC1206FR-071M2L	RES 1.2M OHM 1% 1/4W 1206	0.0097	970,200
11	10000000	SAM1172-07-ND	TMM-107-01-T-S-SM	CONN HEADER SMD 7POS 2MM	0.49896	4,989,600
12	10000000	2563S-05-ND	25630501RP2	CONN RCPT 5POS 0.079 GOLD PCB	1.176	11,760,000
13	10000000	SAM1168-05-ND	TMM-105-01-G-S-SM	CONN HEADER SMD 5POS 2MM	0.73008	7,300,800
14	10000000	2563S-04-ND	25630401RP2	CONN RCPT 4POS 0.079 GOLD PCB	1.036	10,360,000
15	10000000	2057-2PH1-04-UA-SMT-A-ND	2PH1-04-UA-SMT-A	CONN HEADER SMD 4POS 2MM	0.05751	575,140
16	10000000	2563S-03-ND	25630301RP2	CONN RCPT 3POS 0.079 GOLD PCB	0.952	9,520,000
17	10000000	SAM13276TR-ND	TMM-103-01-L-S-SM-P-TR	CONN HEADER SMD 3POS 2MM	0.5866	5,866,000
18	20000000	2563S-02-ND	25630201RP2	CONN RCPT 2POS 0.079 GOLD PCB	0.882	17,640,000
19	20000000	2057-2PH1-02-UA-SMT-A-ND	2PH1-02-UA-SMT-A	CONN HEADER SMD 2POS 2MM	0.02903	580,640
20	100000000	2057-2PH1-10-UA-SMT-B-ND	2PH1-10-UA-SMT-B	CONN HEADER SMD 10POS 2MM	0.1151	11,510,100
21	10000000	2563S-07-ND	25630701RP2	CONN RCPT 7POS 0.079 GOLD PCB	1.358	13,580,000
22	15000000	1497-1479-2-ND	XZVS54S-9A	2.0X1.25MM 365NM UV SMD 0805 LED	1.638	24,570,000
23	60000000	FC4TR100DERTR-ND	FC4TR100DER	RES 0.1 OHM 0.5% 1/2W 1206	0.2584	15,504,000
24	25000000	MCP6004-I/ST-ND	MCP6004-I/ST	IC OPAMP GP 4 CIRCUIT 14TSSOP	0.405	10,125,000
25	12000000	732-3336-2-ND	74437324100	FIXED IND 10UH 1.5A 243 MOHM SMD	1.576	18,912,040
26	60000000	478-7967-2-ND	12062C104KAT2A	CAP CER 0.1UF 200V X7R 1206	0.25875	15,525,000
27	40000000	445-9281-2-ND	CGJ5L3X7T2D224K160AA	CAP CER 0.22UF 200V X7T 1206	0.48429	19,371,600
28	50000000	478-5899-2-ND	12063C224KAT2A	CAP CER 0.22UF 25V X7R 1206	0.06383	3,191,250
29	12000000	161-LT8365HMSE#PBF-ND	LT8365HMSE#PBF	60VIN BOOST/SEPIC/INV CONV 1.5A/	4.1085	49,302,000
30	100000000	541-107KFTR-ND	CRCW1206107KFKEA	RES SMD 107K OHM 1% 1/4W 1206	0.01075	1,075,200
31	100000000	P84.5KFTR-ND	ERJ-8ENF8452V	RES SMD 84.5K OHM 1% 1/4W 1206	0.01212	1,212,000
32	100000000	541-3.24KFTR-ND	CRCW12063K24FKEA	RES SMD 3.24K OHM 1% 1/4W 1206	0.01075	1,075,200
33	100000000	A106072TR-ND	CRG1206F1M0	RES SMD 1M OHM 1% 1/4W 1206	0.01088	1,088,200
34	100000000	311-1.54MFRTR-ND	RC1206FR-071M54L	RES 1.54M OHM 1% 1/4W 1206	0.0097	970,200
35	30000000	311-1466-2-ND	CC1206KKX5R8BB106	CAP CER 10UF 25V X5R 1206	0.17673	5,301,900
36	15000000	SRN6045-100MTR-ND	SRN6045-100M	FIXED IND 10UH 2.5A 58.6MOHM SMD	0.25522	3,828,300
37	15000000	B120AF-13DITR-ND	B120AF-13	DIODE SCHOTTKY 20V 1A SMAF	0.05077	761,520
38	50000000	732-1593-2-ND	742792651	FERRITE BEAD 200 MOHM 0603 1LN	0.095	4,750,000
39	15000000	296-TLV61048DBVRTR-ND	TLV61048DBVR	IC REG BOOST ADJ SOT23-6	0.62125	9,318,750
40	100000000	A106067TR-ND	CRG1206F100K	RES SMD 100K OHM 1% 1/4W 1206	0.01088	1,088,200
41	60000000	SI7858BDP-T1-GE3TR-ND	SI7858BDP-T1-GE3	MOSFET N-CH 12V 40A PPAK SO-8	0.792	47,520,000
42	100000000	RNCP1206FTD20R0TR-ND	RNCP1206FTD20R0	RES 20 OHM 1% 1/2W 1206	0.00993	992,800
43	60000000	1276-2876-2-ND	CL31A106KBHNNNE	CAP CER 10UF 50V X5R 1206	0.09437	5,662,200
Printed Circuit Boards (PCBs) Fabrication - for 100 million units						
cost per unit				0.59 [$]		
Superhydrophobic surfaces (SHS) Fabrication - for 100 million units						
cost per unit				0.026 [$]		
Overall cost per unit				**5.8 [$]**		

Section S5 – Primer selection for SARS-CoV-2 detection assay

The primer mix we used for the amplification of the N-gene from the SARS-CoV-2 genome by a LAMP assay as suggested by Zhang *et al.* [7] is specified in table 2S. In addition, we used the F3 and R3 primers in the PCR assay for the production of the dsDNA that we mixed with the raw saliva sample.

Table S2. *Primer mix for LAMP amplification of the SARS-CoV-2 N-gene.*

Primer	**Sequence**	**Concentration**
F3	TGG CTA CTA CCG AAG AGC T	0.2 µM
R3	TGC AGC ATT GTT AGC AGG AT	0.2 µM
LF	GGA CTG AGA TCT TTC ATT TTA CCG T	0.4 µM
LB	ACT GAG GGA GCC TTG AAT ACA	0.4 µM
FIP	TCT GGC CCA GTT CCT AGG TAG TCC AGA CGA ATT CGT GGT GG	1.6 µM
BIP	AGA CGG CAT CAT ATG GGT TGC ACG GGT GCC AAT GTG ATC T	1.6 µM

Section S6 – Control code

We used a dsPIC30F6015 controller, and therefore wrote the code in C for MPLab X.

```
// Definitions of the microchip controller and the pinout of the PCB
#define FP 8000000L                     //8 clock speed
#define BAUDRATE 115200                 //Desired Baud Rate = 115200
#define PLL_fac 1
#define BRGVAL (((FP*PLL_fac/4)/BAUDRATE)/16) - 1      //Baud Rate Generator

#define Heater_Lysis 1
#define Heater_LAMP_Pos 2
#define Heater_LAMP_Test 3
#define Heater_LAMP_Neg 4
#define Detection_LED 5
#define Indicator_LED 6

#define Temp_Sense_Lysis 0
#define Temp_Sense_Pos 1
#define Temp_Sense_Test 2
#define Temp_Sense_Neg 3
#define Res_Sense_Pos 4
#define Res_Sense_Test 5
#define Res_Sense_Neg 8

#define lysis 1
#define inactivation 2
#define amplification 3
#define measure 4
```

```
#define detect 5
#define amplification_mid 6

#define true 1
#define false 0
#define one_sec 500                    // 100*10ms = 1s

#define Pump_on LATBbits.LATB10 = 1;
#define Pump_off LATBbits.LATB10 = 0;
#define LED_on LATBbits.LATB9 = 1;
#define LED_off LATBbits.LATB9 = 0;
#define Detection_LED_on LATBbits.LATB11 = 1;
#define Detection_LED_off LATBbits.LATB11 = 0;

/*=============================================================================*/
// Included libraries
#include <stdio.h>
#include <stdlib.h>
#include <p30f4013.h>
#include <xc.h>
#include <limits.h>

/*=============================================================================*/
// Definitions of the parameters of the implemented assay
static unsigned int ms_time_count = ((FP/4)*PLL_fac)/(1*1000);          //1ms
static int PWM_Period = (FP/4)*PLL_fac/20000;          //Jumps of 1% in the PWM duty cycle
static double D2A = 0.001221;                    // 5[V]/4095 - analog read resolution

static int G_analog_read_delay = 0.1*one_sec;
```

```
static int G_samp_rate = one_sec*1;                    //

//static double Temp_Sens_Tol = 0.5; //Measurement tolerance on degrees to prevent unnecessary fluctuations.

static long Lyse_Time = 10 * 60L * one_sec;

static double G_Lyse_Temp = 30;

static int DC_Lyse_max = 8;

static int DC_Lyse_min = 4;

static double Temp_Sens_Tol_Lysis = 0.5;

static long Inact_Time = 5 * 60L * one_sec;

static double G_Inact_Temp = 95;

//static int DC_Inact_max = 9;

static int DC_Inact_max = 18;

static int DC_Inact_min = 3;

static double Temp_Sens_Tol_Inact = 0.5;

static long Valve_time_total = 5 * 60L * one_sec;

static int Pulse_Num = 300;

static int Valve_stop_time = 2 * 60L * one_sec;

static long LAMP_Time = 60 * 60L * one_sec;                    //*

static double G_LAMP_Temp = 60;                              //*

static double G_LAMP_Temp_mid = 85;

static int DC_LAMP_max = 6;

static int DC_LAMP_mid_max = 100;

static int DC_LAMP_min = 1;

static double Temp_Sens_Tol_LAMP = 0.5;
```

```
static long Detect_Time = 100 * 60L * one_sec;                    //*8
static double G_detect_Temp = 60;                                 //*
static int DC_detect_max = 10;
static int DC_detect_min = 1;
static double Temp_Sens_Tol_detect = 0.5;

volatile char G_stage = '0';
volatile int Off_flag = true;
volatile int Skip_flag = true;

/*=====================================================*/
// Definitions of used functions
void initialization(void);
char ScanC (void);
void PrintC (char);
void PrintS (char*);
char Echo (void);
void msDelay (long);
int Analog_write (int, int);
double Analog_read (int);
void menu (void);
double avgAnalogRead(int);
void Temp_control(double, int, int);
void Valve_Open(int, int);
void Heating(int);
void Light(void);
void close_shop(void);
void prog (void);
```

```
void Blink (int, int);

/*=====================================================*/
// Initialization of the different controller modules
void initialization(void)
{
   INTCON1bits.NSTDIS = 0;       // no nested interrupts

   // Initiate the Oscillator (inner clock)//
   OSCCON = 0x0000;              //General initiation
   OSCCONbits.COSC = 0b001;       //FRC internal fast RC (8MHz)-Read only?
   OSCCONbits.NOSC = 0b001;       //FRC internal fast RC (8MHz)
   OSCCONbits.POST = 0b00;        //Oscillator postscaler divides clock by 1

   //Button//

   INTCON2bits.INT0EP = 1;
   IEC0bits.INT0IE = 1;
   IPC0bits.INT0IP = 6;
   IFS0bits.INT0IF = 0;

   //UART//
   U1MODE = 0x0000;            //General initialization
   U1BRG = BRGVAL;             //UART BAUD RATE GENERATOR - culculated for 115200 in the .h file
   U1STA = 0x0000;             //General initialization
   U1STAbits.UTXISEL = 0b00;      //the UxTXIF bit is set when a character is transferred from the transmit buffer to the UARTx Transmit Shift Register (UxTSR) or the transmit buffer is empty. This implies at least one location is empty in the transmit buffer.
   U1MODEbits.UARTEN = 1;        //UARTx is enabled
```

```
    U1STAbits.UTXEN = 1;         //UARTx transmitter is enabled; UxTX pin is controlled by UARTx (if UARTEN
= 1)
    U1STAbits.URXDA = 0;          //Flag for a received word

    // UART - receiver Interrupt //
    IEC0bits.U1RXIE = 1;          //enable interrupt
    IFS0bits.U1RXIF = 0;          //flag
    IPC2bits.U1RXIP = 6;       //priority

    // Initiate Timer 1 //
    T1CONbits.TON = 0;            //Stops the timer
    T1CONbits.TGATE = 0;          //Gated time accumulation disabled
    T1CONbits.TCS = 0;            //Internal clock
    T1CONbits.TCKPS = 0b00;       //1:1 prescale
    PR1 = ms_time_count;                  // milisecond count in timer 1

    //Timer 1 interapt //
    IEC0bits.T1IE = 1;            //enable interrupt
    IFS0bits.T1IF = 0;            //flag
    IPC0bits.T1IP = 4;          //priority

    // Initiate Timer 2
    T2CONbits.TON = 0;            //Stops the timer
    T2CONbits.TGATE = 0;          //Gated time accumulation disabled
    T2CONbits.TCS = 0;            //Internal clock
    T2CONbits.TCKPS = 0b00;              //1:1 prescale value on a F_OSC/4 working clock this results in
7.37MHz/(1*4) = 1842500Hz
    //Timer 2 interapt //
    IEC0bits.T2IE = 0;            //enable interapt
```

```
IFS0bits.T2IF = 0;            //flag
IPC1bits.T2IP = 4;           //priority
PR2 = PWM_Period;
T2CONbits.TON = 1;            //Timer enable

 // Initiate Pins
// lysis Temp sens //
TRISBbits.TRISB0 = 1;          //Input pin
ADPCFGbits.PCFG0 = 0;          //Analog input
// LAMP-Pos(#1) Temp sens //
TRISBbits.TRISB1 = 1;
ADPCFGbits.PCFG1 = 0;
// LAMP-Pos(#1) Res sens //
TRISBbits.TRISB4 = 1;
ADPCFGbits.PCFG4 = 0;
// LAMP-Test(#2) Temp sens //
TRISBbits.TRISB2 = 1;
ADPCFGbits.PCFG2 = 0;
// LAMP-Test (#2) Res sens //
TRISBbits.TRISB5 = 1;
ADPCFGbits.PCFG5 = 0;
// LAMP-Neg Temp sens //
TRISBbits.TRISB3 = 1;
ADPCFGbits.PCFG3 = 0;
 // LAMP-Neg Res sens //
TRISBbits.TRISB8 = 1;
ADPCFGbits.PCFG8 = 0;
```

```
// Initiate the ADC module
ADCON1 = 0x0000;          //General initiation
ADCON1bits.ADSIDL = 0;     //Continue module operation in Idle mode
ADCON1bits.FORM = 0b00;      //Integer (DOUT = 0000 00dd dddd dddd)
ADCON1bits.SSRC = 0b111;     //Clearing SAMP bit ends sampling and starts conversion
ADCON1bits.ASAM = 0;       //Sampling begins when SAMP bit set
ADCON1bits.SAMP = 0;       //A/D sample/hold amplifiers are holding
ADCON2 = 0x0000;          //General initiation
ADCON2bits.VCFG = 0b000;     //Voltage Reference Configuration bits VREFH-AVDD : VREFL-AVSS
ADCON2bits.BUFM = 0;       //Buffer configured as one 16-word buffer ADCBUF(15...0)
ADCON2bits.ALTS = 0;       //Always use MUX A input multiplexer settings
ADCON2bits.CSCNA = 0;      //Do not scan inputs
ADCON2bits.SMPI = 0b0000;     //Interrupts at the completion of conversion for each sample/convert
sequence
ADCON3 = 0x0000;          //General initiation
ADCON3bits.ADRC = 0;       //Clock derived from system clock
ADCON3bits.SAMC = 0b11110;    //Auto-Sample Time bits 30*TAD
ADCON3bits.ADCS = 0b11110;    //A/D Conversion Clock Select bits: TCY/2 ? (ADCS<5:0> + 1) = 31*TCY
ADCSSL = 0x0000;          //General initiation
ADCHS = 0x0000;
ADCON1bits.ADON = 1;       //A/D converter module is operating


// Initiate Output compare
OC1CON = 0x0000;          //Lysis heater              //General initialize (Chooses Timer 2 as well)
OC2CON = 0x0000;          //LAMP Positive heater(#1)     //General initialize (Chooses Timer 2 as well)
OC3CON = 0x0000;          //LAMP Test heater(#2)         //General initialize (Chooses Timer 2 as well)
OC4CON = 0x0000;          //LAMP Negative heater(#3)     //General initialize (Chooses Timer 2 as well)
```

```
    TRISDbits.TRISD8 = 1;          //Input - switch


    TRISDbits.TRISD0 = 0;          //Output - Lysis heater
    TRISDbits.TRISD1 = 0;          //Output - LAMP Positive heater(#1)
    TRISDbits.TRISD2 = 0;          //Output - LAMP Test heater(#2)
    TRISDbits.TRISD3 = 0;          //Output - LAMP Negative heater(#3)
    TRISBbits.TRISB11 = 0;         //Output - Detection LED
    TRISBbits.TRISB9 = 0;          //Output - Indicator LED


    TRISBbits.TRISB10 = 0;         //Output - enable HV
    LATBbits.LATB10 = 0;           //( low-Off, high-On)


}


//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// Interrupts for user input during operation stage
void _ISR  __attribute__((auto_psv))  _U1RXInterrupt(void) //receiver interrupt
{
    char in;
    PrintS("\n\r");
    in  = Echo();
    PrintS("\n\r");


    int innerflag = false;


    if (in == 'N' || in == 'n') {
        Skip_flag = true;
        innerflag = true;
```

```
        G_stage ++;
    }

    if (in == 'e' || in == 'E') {
        Off_flag = true;
        innerflag = true;
    }

    if (in >= '1' && in <= '5'){
        innerflag = true;
        G_stage = in;
    }

    if (innerflag == false) {
        PrintS("\n\rwrong, try again\n\r");
        PrintS("If you want to skip a stage press n,\n\r");
        PrintS("and if you want it all to end press e\n\r\n\r");
    }
    IFS0bits.U1RXIF = 0;
}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

void _ISR  __attribute__((auto_psv))  _T1Interrupt( void) //resiver interapt
{
    IFS0bits.T1IF = 0;
}
```

```
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

void _ISR  __attribute__((auto_psv))  _T2Interrupt( void) //resiver interapt
{
   IFS0bits.T2IF = 0;
}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

void _ISR  __attribute__((auto_psv))  _INT0Interrupt( void)
{

//    Off_flag = true;
   G_stage = '1';
   IFS0bits.INT0IF          = 0;

}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//Read an input char
char ScanC (void)
{
   while(!U1STAbits.URXDA);
   char c = U1RXREG;
   U1RXREG;
   return (c);
}
```

```
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

//Print (output) a char

void PrintC (char c)

{

   U1TXREG = c;

   while (!IFS0bits.U1TXIF);

   IFS0bits.U1TXIF = 0;

}


//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

//Print (output) a string of chars

void PrintS (char* s)

{

   while (*s != '\0')

   {

      PrintC(*s++);

   }

}


//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


char Echo (void)

{

   char cc;

   cc = ScanC();

   PrintC(cc);
```

```
   return(cc);
}


//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//Delay of XXX milliseconds
void msDelay (long dt)
{
   long t = 0;                   //Delay in ms counter
   T1CONbits.TON = 1;
   while (t < dt)               //loop until the required delay is reached
   {
      while(!IFS0bits.T1IF);     //false as long as the timer is counting
         t++;
   }
   T1CONbits.TON = 0;
}


//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//enables PWM in "pin" with a duty cycle of "per"
int Analog_write (int pin, int Duty_cycle)
{

   if (Duty_cycle > 100 || Duty_cycle < 0 || pin > 4 || pin < 0){       //wrong input
      return(0);
   }

   switch (pin)
```

```
  {
    case Heater_Lysis:
      OC1R = Duty_cycle*PWM_Period/100;
      OC1RS = Duty_cycle*PWM_Period/100;
      OC1CONbits.OCM = 0b110;   //initialize PWM mode
      break;
    case Heater_LAMP_Pos:
      OC2R = Duty_cycle*PWM_Period/100;
      OC2RS = Duty_cycle*PWM_Period/100;
      OC2CONbits.OCM = 0b110;   //initialize PWM mode
      break;
    case Heater_LAMP_Test:
      OC3R = Duty_cycle*PWM_Period/100;
      OC3RS = Duty_cycle*PWM_Period/100;
      OC3CONbits.OCM = 0b110;   //initialize PWM mode
      break;
    case Heater_LAMP_Neg:
      OC4R = Duty_cycle*PWM_Period/100;
      OC4RS = Duty_cycle*PWM_Period/100;
      OC4CONbits.OCM = 0b110;   //initialize PWM mode
      break;
  }
   return(1);
}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

double Analog_read (int Pin)
```

```
{
   double r,res;
   ADCHSbits.CH0SA = Pin;         //Channel 9 Positive Input Select for MUX A Multiplexer Setting bits
   ADCON1bits.SAMP = 1;
   while(!ADCON1bits.DONE);
   r = ADCBUF0;
   res = r*D2A;
   return(res);
}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

void menu (void)
{
   PrintS("At which stage number (1-5) would you like to begin?\n\r");
   PrintS("Stage 1: lysis\n\r");
   PrintS("Stage 2: Inactivation\n\r");
   PrintS("Stage 3: Valve push\n\r");
   PrintS("Stage 4: Amplification\n\r");
   PrintS("Stage 5: Detection\n\r");
}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

double avgAnalogRead(int p) {
   int i;
   double sum = 0;
```

```
   for (i = 0; i <= 10; i++) {
      sum = sum + Analog_read(p);
   }
   double avg = sum / (i-1);
   return (avg);
}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

void Temp_control(double Temp_Now, int operation, int heater)
{
   double dn;
   double dp;
//    char buff[80];
//    int max = 7;
//    int min = 0;

   switch (operation)
   {
      case lysis:
      {
         dn = G_Lyse_Temp - Temp_Sens_Tol_Lysis;
         dp = G_Lyse_Temp + Temp_Sens_Tol_Lysis;
         if (Temp_Now < dn)
         {
            Analog_write(heater,DC_Lyse_max);
         }
         if (Temp_Now >= dp)
```

```
    {
      Analog_write(heater,DC_Lyse_min);
    }
    break;
  }
  case inactivation:
  {
    dn = G_Inact_Temp - Temp_Sens_Tol_Inact;
    dp = G_Inact_Temp + Temp_Sens_Tol_Inact;
    if (Temp_Now < dn)
    {
      Analog_write(heater,DC_Inact_max);
    }
    if (Temp_Now >= dp)
    {
      Analog_write(heater,DC_Inact_min);
    }
    break;
  }
  case amplification:
  {
    dn = G_LAMP_Temp - Temp_Sens_Tol_LAMP;
    dp = G_LAMP_Temp + Temp_Sens_Tol_LAMP;
    if (Temp_Now < dn)
    {
      Analog_write(heater,DC_LAMP_max);
//        Detection_LED_on
    }
    if (Temp_Now >= dp)
```

```
        {
          Analog_write(heater,DC_LAMP_min);
//          Detection_LED_off
        }
        break;
      }
      case measure:
      {
        int Duty_cycle = 0;
        Analog_write (Heater_Lysis,Duty_cycle);
        Analog_write (Heater_LAMP_Pos,Duty_cycle);
        Analog_write (Heater_LAMP_Test,Duty_cycle);
        Analog_write (Heater_LAMP_Neg,Duty_cycle);
        break;
      }
      case detect:
      {
        dn = G_detect_Temp - Temp_Sens_Tol_detect;
        dp = G_detect_Temp + Temp_Sens_Tol_detect;
        if (Temp_Now < dn)
        {
          Analog_write(heater,DC_detect_max);
//          Detection_LED_on
        }
        if (Temp_Now >= dp)
        {
          Analog_write(heater,DC_detect_min);
//          Detection_LED_off
        }
```

```
        break;
      }
      case amplification_mid:
      {
        dn = G_LAMP_Temp_mid - Temp_Sens_Tol_LAMP;
        dp = G_LAMP_Temp_mid + Temp_Sens_Tol_LAMP;
        LED_on
        if (Temp_Now < dn)
        {
          Analog_write(heater,DC_LAMP_mid_max);
//          Detection_LED_on
        }
        if (Temp_Now >= dp)
        {
          Analog_write(heater,DC_LAMP_min);
//          Detection_LED_off
        }
        break;
      }
    }
  return;

}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

void Valve_Open(int dt, int N) {
```

```
  char buffer[80];

  sprintf(buffer,"We are at stage %c, and we are opening the valve \n\r",G_stage);
  PrintS(buffer);

  int N2;
  N2= N/2;

  long mmm = 2*60L * one_sec;

  if (N == 1) {

    Pump_on
//    Detection_LED_on
    msDelay(dt);
    Pump_off
//    LED_off
//    msDelay(dt);
//    msDelay(dt);
//    msDelay(dt);
//    msDelay(dt);
    return;
  }
  int i = 0;
  while (i < N && Skip_flag == false) {

//    if (i<(N2-1)) {
////      PrintC('A');
//      Pump_on
```

```
////          LED_on
////          msDelay(dt/5);
//          msDelay(dt);
//          Pump_off
////          LED_off
//          msDelay(dt/5);
//          msDelay(dt/5);
//          msDelay(dt/5);
//          msDelay(dt/5);
//      }
//
//      if (i<(N2+1) && i>(N2-1))
//      {
//          Pump_off
//          LED_on
//          msDelay(5*one_sec);
//          LED_off
//          PrintC('A');
//          msDelay(mmm);
//          PrintC('B');
//          LED_on
//          msDelay(5*one_sec);
//          LED_off
//      }
//
//      if (i>(N2+1))
//      {
////          PrintC('C');
//          Pump_on
```

```
////         LED_on
//          msDelay(dt);
//       }
//
       Pump_on
////       LED_on
       msDelay(dt);
//       Pump_off
////       LED_off
//       msDelay(dt/2);
//       msDelay(dt/2);
//       msDelay(dt/2);
//       msDelay(dt/2);
       i++;
   }
   Pump_off
   LED_off
   return;
}


//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

// The heating function including the sensor calibration

```
void Heating(int operation) {


   char buffer[80];
   double Temp_Now;
   double Temp_Now_pos;
   double Temp_Now_test;
```

```
  double Temp_Now_neg;
  double Temp_sens;
  double Res_sens_Pos;
  double Res_sens_Test;
  double Res_sens_Neg;
  long Time_count = 0;
  int pulse_number = 1;

  while (Skip_flag == false && Off_flag == false)
  {
    switch (operation)
    {
      case lysis:
      {
        if (Time_count <= (Lyse_Time/one_sec))
        {
          Temp_control(Temp_Now,measure,Heater_LAMP_Pos);
          msDelay(10);
          Temp_sens = avgAnalogRead(Temp_Sense_Lysis);
          Temp_Now = Temp_sens * 70 - (0.5 * 105);
          Temp_control(Temp_Now,operation,Heater_Lysis);
//                        sprintf(buffer,"We are at stage %c, performing lysis, and the Temp is %f C\n\r",G_stage,Temp_Now);
//            PrintS(buffer);
//                              sprintf(buffer,"This stage is already running for %ld s, out of %ld s\n\r",Time_count,Lyse_Time/one_sec);
//            PrintS(buffer);
          sprintf(buffer,"%c %f %ld %ld\n\r",G_stage,Temp_Now,Time_count,Lyse_Time/one_sec);
          PrintS(buffer);
```

```
            msDelay(G_samp_rate);
            Time_count = Time_count + G_samp_rate/one_sec;
          }
          else
          {
            return;
          }
          break;
        }
        case inactivation:
        {
          if (Time_count <= (Inact_Time/one_sec))
          {
            Temp_control(Temp_Now,measure,Heater_LAMP_Pos);
            msDelay(10);
            Temp_sens = avgAnalogRead(Temp_Sense_Lysis);
            Temp_Now = Temp_sens * 73 - (0.5 * 105);
            Temp_control(Temp_Now,operation,Heater_Lysis);
//                  sprintf(buffer,"We are at stage %c , performing inactivation, and the Temp is %f C\n\r", G_stage, Temp_Now);
//              PrintS(buffer);
//                sprintf(buffer,"This stage is already running for %ld sec, out of %ld sec\n\r", Time_count, Inact_Time/one_sec);
//              PrintS(buffer);
            sprintf(buffer,"%c %f %f %ld %ld\n\r",G_stage, Temp_Now,G_Inact_Temp,Time_count, Inact_Time/one_sec);
            PrintS(buffer);
            msDelay(G_samp_rate);
            Time_count = Time_count + G_samp_rate/one_sec;
          }
```

```
            else
            {
               return;
            }
            break;
         }
         case amplification:
         {
            int a = 73;
            int b = -50;
            if (Time_count <= (LAMP_Time/one_sec))
            {
//                msDelay(100);
               Temp_control(Temp_Now,measure,0);
               msDelay(10);

               Temp_sens = avgAnalogRead(Temp_Sense_Pos);  // left
               //Temp_Now_pos = Temp_sens * a - 53;
//                Temp_Now_pos = Temp_sens * a - 39;
               //Temp_Now_pos = Temp_sens * a - 41;
               Temp_Now_pos = Temp_sens * 72 - 33.5;  //gold
//                Temp_Now_pos = Temp_sens * 73 - 34.5;  //naked
               Temp_sens = avgAnalogRead(Temp_Sense_Test);
               //Temp_Now_test = Temp_sens * a + b;
//                Temp_Now_test = Temp_sens * 73 - 40;
               //Temp_Now_test = Temp_sens * 73 - 42;
               Temp_Now_test = Temp_sens * 73 - 36.5;  //gold
//                Temp_Now_test = Temp_sens * 73 - 34;  //naked
               Temp_sens = avgAnalogRead(Temp_Sense_Neg); // right
```

```
            //Temp_Now_neg = Temp_sens * 73 -45;
//              Temp_Now_neg = Temp_sens * 73 -38;
            //Temp_Now_neg = Temp_sens * 73 - 40;
            Temp_Now_neg = Temp_sens * 73 - 35.9;   //gold
//              Temp_Now_neg = Temp_sens * 73 - 34;   //naked

            Res_sens_Pos = 1000*avgAnalogRead(Res_Sense_Pos);
            Res_sens_Test = 1000*avgAnalogRead(Res_Sense_Test);
            Res_sens_Neg = 1000*avgAnalogRead(Res_Sense_Neg);
////            sprintf(buffer,"The resistance of the negative control pad is %f\n\r",Res_sens);
////            PrintS(buffer);

            Temp_control(Temp_Now_pos,operation,Heater_LAMP_Pos);
            //Temp_control(Temp_Now_test,amplification_mid,Heater_LAMP_Test);
            Temp_control(Temp_Now_test,operation,Heater_LAMP_Test);
            Temp_control(Temp_Now_neg,operation,Heater_LAMP_Neg);

            sprintf(buffer,"\n\rWe are at stage %c.\n\rThe temp of the positive control pad is %f C\n\r",G_stage,Temp_Now_pos);
//              PrintS(buffer);
//              sprintf(buffer,"The temp of the test pad is %f C\n\r",Temp_Now_test);
//              PrintS(buffer);
//              sprintf(buffer,"The temp of the negative control pad is %f C\n\r",Temp_Now_neg);
//              PrintS(buffer);
//              sprintf(buffer,"\n\r The resistance of the positive control pad is %f\n\r",Res_sens_Pos);
//              PrintS(buffer);
//              sprintf(buffer,"The resistance of the Test pad is %f\n\r",Res_sens_Test);
//              PrintS(buffer);
//              sprintf(buffer,"The resistance of the negative control pad is %f\n\r",Res_sens_Neg);
```

```
//              PrintS(buffer);
//
//              Valve_Open(G_samp_rate,pulse_number);
              msDelay(G_samp_rate);
              Time_count = Time_count + G_samp_rate/one_sec;
//                            sprintf(buffer,"\nThis stage is already running for %ld s, out of %ld s\n\r",Time_count,LAMP_Time/one_sec);
//              PrintS(buffer);
              sprintf(buffer,"%c %.2f %.2f %.2f %.0f %.4f %.4f %.4f %ld %ld\n\r",G_stage,Temp_Now_pos,Temp_Now_test,Temp_Now_neg,G_LAMP_Temp,Res_sens_Pos,Res_sens_Test,Res_sens_Neg,Time_count,LAMP_Time/one_sec);
              PrintS(buffer);
            }
            else
            {
              return;
            }
            break;
          }
            case detect:
          {
            int a = 73;
            int b = -50;
            if (Time_count <= (Detect_Time/one_sec))
            {
//              msDelay(100);
              Temp_control(Temp_Now,measure,0);
              msDelay(10);

              Temp_sens = avgAnalogRead(Temp_Sense_Pos);  // left
```

```
            Temp_Now_pos = Temp_sens * a - 53;
            Temp_sens = avgAnalogRead(Temp_Sense_Test);
            Temp_Now_test = Temp_sens * a + b;
            Temp_sens = avgAnalogRead(Temp_Sense_Neg); // right
            Temp_Now_neg = Temp_sens * 73 -45;

            Temp_control(Temp_Now_pos,operation,Heater_LAMP_Pos);
            Temp_control(Temp_Now_test,operation,Heater_LAMP_Test);
            Temp_control(Temp_Now_neg,operation,Heater_LAMP_Neg);

            sprintf(buffer,"\n\rWe are at stage %c.\n\rThe temp of the positive control pad is %f C\n\r",G_stage,Temp_Now_pos);
            PrintS(buffer);
            sprintf(buffer,"The temp of the test pad is %f C\n\r",Temp_Now_test);
            PrintS(buffer);
            sprintf(buffer,"The temp of the negative control pad is %f C\n\r",Temp_Now_neg);
            PrintS(buffer);

            msDelay(G_samp_rate);
            Time_count = Time_count + G_samp_rate/one_sec;
            sprintf(buffer,"This stage is already running for %ld s, out of %ld s\n\r",Time_count,Detect_Time/one_sec);
            PrintS(buffer);
        }
        else
        {
            return;
        }
        break;
```

```
        }
      }
    }
    return;
}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

void Light(void)
{
    char buffer[80];
    long Time_count = 0;
    int delay = Detect_Time/10;

    sprintf(buffer,"We are at stage %c, and turning on the lights\n\r",G_stage);
    PrintS(buffer);

    while (Skip_flag == false && Off_flag == false && Time_count <= Detect_Time)
    {
        Detection_LED_on
        LED_on

        msDelay(delay);
        Time_count = Time_count + delay/one_sec;
        sprintf(buffer,"This stage is already running for %ld s, out of %ld s\n\r",Time_count,Detect_Time/one_sec);
        PrintS(buffer);
    }
```

```
  Detection_LED_off
  LED_off
 return;
}
```

//%%

// Turning off the device

```
void close_shop(void) {
  int Duty_cycle = 0;
  LED_off
  Detection_LED_off
  Pump_off              //Disable pump
  Analog_write (Heater_Lysis,Duty_cycle);
  Analog_write (Heater_LAMP_Pos,Duty_cycle);
  Analog_write (Heater_LAMP_Test,Duty_cycle);
  Analog_write (Heater_LAMP_Neg,Duty_cycle);
  Off_flag = false;
  Skip_flag = false;
  PrintS("\n\rall is buried and done\n\r");
  return;
}
```

//%%

// The main program performing the stages of the assay

```
void prog (void)
{
```

```
   double Valve_Pulse_Time = Valve_time_total/Pulse_Num;
   do
   {
      close_shop();
//      LED_on
//   Detection_LED_on
      mnue();
      G_stage = '0';
      while(G_stage == '0');

      while (G_stage >= '1' && G_stage <= '5' && Off_flag == false)
      {
         PrintS("\n\rIf you want to skip a stage press n\n\r");
         PrintS("and if you want it all to end press e\n\r\n\r");
         switch (G_stage)
         {
             case '1':
               close_shop();
               Heating(lysis); //Lysis
               G_stage = '2';
               Skip_flag = false;
               break;
             case '2':
               close_shop();
               Heating(inactivation); //Inactivation
               G_stage = '3';
         //      PrintS("\n\r");
         //      PrintS("Stage 2 is done. Please add water (not coffee)...\n\r");
         //      PrintS("\n\r");
```

```
//      Off_flag  = true;
        Skip_flag = false;
        break;
      case '3':
        close_shop();
        Valve_Open(Valve_Pulse_Time, Pulse_Num);
//        Pump_on
//        LED_on
//        msDelay(Valve_Pulse_Time);
        G_stage = '4';
        Skip_flag = false;
        break;
      case '4':
        close_shop();
        msDelay(one_sec);
        Heating(amplification);
        G_stage = '0';
        Skip_flag = false;
        break;
      case '5':
        close_shop();
        Detection_LED_on
        LED_on
        Heating(detect);
        Detection_LED_off
        LED_off
//        Light();
        Off_flag  = true;
        PrintS("good job\n\r");
```

```
                G_stage = '0';
                break;
            }
        }
    }
    while(1);
}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

//Testing blink function

```
void Blink (int X_times_to_blink, int dt)
{
    int j;

    for (j=0;j<X_times_to_blink;j++)
    {
        LED_on
        Detection_LED_on
        msDelay (dt);
        LED_off
        Detection_LED_off
        msDelay (dt);
    }
}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
int main (void)
{
//   LED_on
//   Detection_LED_on
   initialization();
   while(1)
   {
//      LED_on
//   Detection_LED_on
      prog();
   }
          return (1);
}

/*=====================================================*/
```

Section S7 - Video description

SI_EOPV_Vid: Demonstration of the EOPV operated in both dosing mode (by applying 20V) and continuous mode (by applying 100V).

References

[1] E. Elizalde, R. Urteaga, C. L. A. Berli, *Lab. Chip* **2015**, *15*, 2173.

[2] H. Darcy, *Les Fontaines Publiques de La Ville de Dijon: Exposition et Application...*, Victor Dalmont, **1856**.

[3] W. R. Purcell, *J. Pet. Technol.* **1949**, *1*, 39.

[4] J. Kozeny, *R. Acad. Sci. Vienna Proc Cl. I* **1927**, *136*, 271.

[5] P. C. Carman, *Trans Inst Chem Eng* **1937**, *15*, 150.

[6] N. Franck, F. Schaumburg, P. A. Kler, R. Urteaga, *ELECTROPHORESIS* **2021**, *42*, 975.

[7] Y. Zhang, N. Odiwuor, J. Xiong, L. Sun, R. O. Nyaruaba, H. Wei, N. A. Tanner, Infectious Diseases (Except HIV/AIDS), **2020**.