اول با توجه به اینکه برای استفاده از LCD و Keypad به پورت‌های A و B قسمت نمونه برداری نیاز داریم، این دو تا پورت را initialize میکنیم:

6.3.9 RCC AHB1 peripheral clock enable register (RCC_AHB1ENR)

Address offset: 0x30

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									DMA2EN	DMA1EN	Reserved				
									rw	rw					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CRCEN	Reserved				GPIOHEN	Reserved		GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
			rw					rw			rw	rw	rw	rw	rw

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..E and H)

Address offset: 0x00

Reset values:

- 0x0C00 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

6.3.12 RCC APB2 peripheral clock enable register (RCC_APB2ENR)

Address offset: 0x44

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													TIM11EN	TIM10EN	TIM9EN
													rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	SYSCFGEN	SPI4EN	SPI1EN	SDIOEN	Reserved		ADC1EN	Reserved		USART6EN	USART1EN	Reserved			TIM1EN
	rw	rw	rw	rw			rw			rw	rw				rw

```
void ports_init(void) {
        RCC -> AHB1ENR |= RCC_AHB1ENR_GPIOAEN;/* turn on the GPIOA clk */
        RCC -> AHB1ENR |= RCC_AHB1ENR_GPIOBEN; /*turn on the GPIOB clk */
        GPIOA -> MODER |= 0x555500; /* set A4-A11 as output */
        GPIOB -> MODER |= 0x55000015; /* set B0-B2 and B12-B15 as output */
        RCC -> APB2ENR |= RCC_APB2ENR_SYSCFGEN; /* enable system configuration controller clock */
}
```

سپس با برای ارتباط سریال بین دو قسمت از USART2 => پورت A2 و A3 استفاده میکنیم.

8.4.9 GPIO alternate function low register (GPIOx_AFRL) (x = A..E and H)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **AFRLy:** Alternate function selection for port x bit y (y = 0..7)
These bits are written by software to configure alternate function I/Os

AFRLy selection:

0000: AF0	1000: AF8
0001: AF1	1001: AF9
0010: AF2	1010: AF10
0011: AF3	1011: AF11
0100: AF4	1100: AF12
0101: AF5	1101: AF13
0110: AF6	1110: AF14
0111: AF7	1111: AF15

8.4.10 GPIO alternate function high register (GPIOx_AFRH) (x = A..E and H)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRH15[3:0]				AFRH14[3:0]				AFRH13[3:0]				AFRH12[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRH11[3:0]				AFRH10[3:0]				AFRH9[3:0]				AFRH8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **AFRHy:** Alternate function selection for port x bit y (y = 8..15)
These bits are written by software to configure alternate function I/Os

AFRHy selection:

0000: AF0	1000: AF8
0001: AF1	1001: AF9
0010: AF2	1010: AF10
0011: AF3	1011: AF11
0100: AF4	1100: AF12
0101: AF5	1101: AF13
0110: AF6	1110: AF14
0111: AF7	1111: AF15

USART Control Register 1 (USART_CR1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER8	Reserved	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
rw	Res.	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- OVER8: 0 oversampling by 16/; 1 oversampling by 8
- UE: Write 1 to enable
- M: Word length, 0: 8 data bits; 1: 9 data bits
- WAKE: Wakeup method
- PCE: Parity enabled with 1
- PS: Odd parity with 1, even parity with 0
- TE: Transmitter enable
- RE: Receiver enable

```
void UART2_init(void){
	RCC->APB1ENR |= 0x20000;  // Enable UART2 CLOCK
	RCC->AHB1ENR |= 0x01; // Enable GPIOA CLOCK

	GPIOA->MODER |= 0x000000A0;  // bits 7-4 = 1010 = 0xA --> Alternate Function for Pin PA2 & PA3
	GPIOA->OSPEEDR |= 0x000000F0;  // bits 7-4 = 1111 = 0xF --> High Speed for PIN PA2 and PA3
	GPIOA->AFR[0] |= 0x07700;  // bits 15-8=01110111=0x77  --> AF7 Alternate function for USART2 at Pin PA2 & PA3

	USART2->BRR = 0x0683;   // Baud rate = 9600bps, CLK = 16MHz

	USART2->CR1 = 1<<13;  // UE = 1 -> Enable USART
	USART2->CR1 &= ~(1<<12);  // M =0; 8 bit word length

	USART2->CR1 |= (1<<2); // RE=1 -> Enable the Receiver
	USART2->CR1 |= (1<<3);  // TE=1 -> Enable Transmitter
}
```

برای ورودی آنالوگ نیز از C1 استفاده میکنیم:

11.12.3 ADC control register 2 (ADC_CR2)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved	SWSTART	EXTEN		EXTSEL[3:0]				reserved	JSWSTART	JEXTEN		JEXTSEL[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved				ALIGN	EOCS	DDS	DMA	Reserved						CONT	ADON
				rw	rw	rw	rw							rw	rw

```
void ADC_init() {
	RCC->AHB1ENR |= RCC_AHB1ENR_GPIOCEN; //anable GPIOC clk
	GPIOC->MODER |= 0xC; //C1 = analog

	RCC->APB2ENR |= 0x00000100; //enable ADC1 clock
	ADC1->CR2 = 0; //SW trigger
	ADC1->SQR3 = 1; //conversion sequence starts at ch 1
	ADC1->SQR1 = 0; //conversion sequence length 1
	ADC1->CR2 |= 1; //enable ADC1
}
```

نکته Vref :

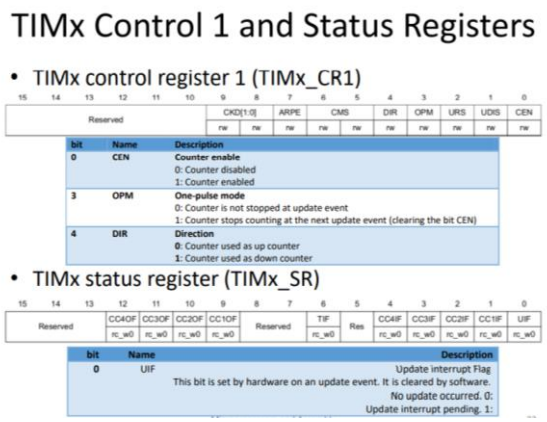
Reference Voltage (V_{ref})

- V_{ref} is an input voltage used for the *reference voltage.*
- V_{ref} and the ADC resolution together dictate the step size.
  - For an *8-bit ADC*, the step size is $V_{ref}/256$ because it is an 8-bit ADC, and $2^8 = 256$ steps.
  - If the analog input range is 0 to 4 volts, then V_{ref} is connected to 4 volts => 4 V/256 = 15.62 step size

V_{ref} (V)	V_{in} Range (V)	Step Size (mV)
5.00	0 to 5	5/256 = 19.53
4.0	0 to 4	4/256 = 15.62
3.0	0 to 3	3/256 = 11.71
2.56	0 to 2.56	2.56/256 = 10
2.0	0 to 2	2/256 = 7.81
1.28	0 to 1.28	1.28/256 = 5
1	0 to 1	1/256 = 3.90

سپس تایمرها را initialize میکنیم (هر 200ms فریم ها فرستاده شوند + هر 10us نمونه برداری شود)

از General purpose timer TIM2 , TIM3 استفاده شده است:



Tout = ((ARR+1)x(PSC+1))/Fclk , Fclk = 16MHz:

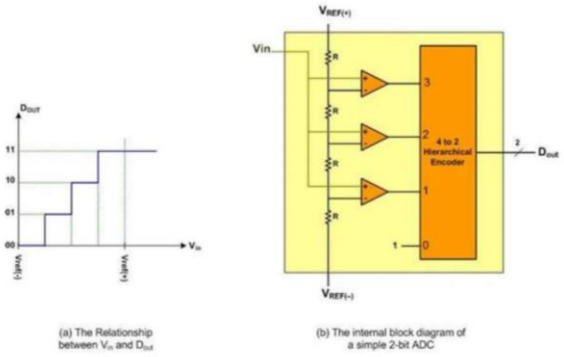
$Tout_2$ = 200ms -> PSC_2 = 16000 – 1, ARR_2 = 200 – 1

$Tout_3$ = 10us -> PSC_3 = 160 – 1, ARR_3 = 1 – 1

```
void TIMER_init() {
	__disable_irq();

	RCC->APB1ENR |= RCC_APB1ENR_TIM2EN; //enable clock of timer 2
	TIM2->PSC = 16000 - 1; //set prescaler -> clk divided by 16000 to get counter frequency
	TIM2->ARR = 200 - 1; //auto reload register
	TIM2->DIER |= 1; //enable the update interrupt
	TIM2->CNT = 0; //clear timer counter
	TIM2->CR1 |= 1; //enable counting
	NVIC_EnableIRQ(TIM2_IRQn); //CMSIS ISR name

	RCC->APB1ENR |= RCC_APB1ENR_TIM3EN; //enable clock of timer 3
	TIM3->PSC = 160 - 1; //set prescaler
	TIM3->ARR = 1 - 1; //auto reload register
	TIM3->DIER |= 1; //enable the update interrupt
	TIM3->CNT = 0; //clear timer counter
	TIM3->CR1 |= 1; //enable counting
	NVIC_EnableIRQ(TIM3_IRQn); //CMSIS ISR name

	__enable_irq();
}
```

هر موقع در تایمر۲ یک update event اتفاق افتاد یعنی 200ms گذشته و باید فریم‌های نمونه گیری شده از طریق UART به بخش display فرستاده شود.

هر موقع در تایمر۳ یک update event اتفاق افتاد یعنی 10us گذشته و باید عمل نمونه گیری انجام شود. به اینگونه که جواب دیجیتالی که از تبدیل ADC را در نظر میگیریم و مانند یک انکودر دیتایمان را بدست میاوریم. با توجه به اینکه Vref = 8v است این کار را انجام میدهیم.



(a) The Relationship between V_{in} and D_{out} (b) The internal block diagram of a simple 2-bit ADC

این داده نمونه گیری شده، همان X است. برای به دست آوردن Y هم از رابطه موجود در صورت پروژه استفاده کرده و Y های نمونه را در یک آرایه ذخیره میکنیم.

وقتی دکمه ای فشرده میشود باید مقدار جدید متغیر را به قسمت display بفرستیم برای این کار: اگر a تغییر کرده بود اول یک ’a‘ میفرستیم و بعد مقدار جدید a را، اگر b تغییر کرده بود اول یک ’b‘ میفرستیم و بعد مقدار جدید b را و اگر time_unit تغییر کرده بود اول یک ’t‘ میفرستیم و بعد مقدار جدید time_unit را.

اگر هیچکدام از این ۳ کاراکتر فرستاده نشده بود یعنی در در حال فرستادن فریم هاست.

در قسمت display نیز USART را مانند قسمت sampling راه اندازی میکنیم.

در قسمت display هم سعی کردم با استفاده از https://github.com/bagherian78/glcd-lib GLCD را درست کنم اما به نتیجه نرسیدم.