# به نام خدا

## گزارش پروژه نهایی ریزپردازنده و زبان اسمبلی – تیرماه ۱۴۰۱

## گروه ۱۳


**اعضای گروه:**

امیرحسین ادواری ۹۸۲۴۳۰۰

زهرا حیدری ۹۸۲۴۳۰۲۰


**محتوا:**

# بخش اول : تنظیم ادوات و io

**در این قسمت شرح مختصری در مورد ورودی‌خروجی‌ها و نیز پارامترهای آن‌ها ارائه می‌شود.**

ورودی‌خروجی‌های میکرو :

| PIN | PA0 | PA2 | PB0 | PB1 | PB2 | PB3 | PB4 | PB5 | PB6 |
|---|---|---|---|---|---|---|---|---|---|
| TYPE | Analog Input | USART OUTPUT | OUT | OUT | OUT | OUT | OUT | OUT | OUT |
| SYMBOL | adc | TX | S0 | S1 | S2 | S3 | S4 | S5 | S6 |

در این پروژه از peripheral های زیر استفاده شده است:

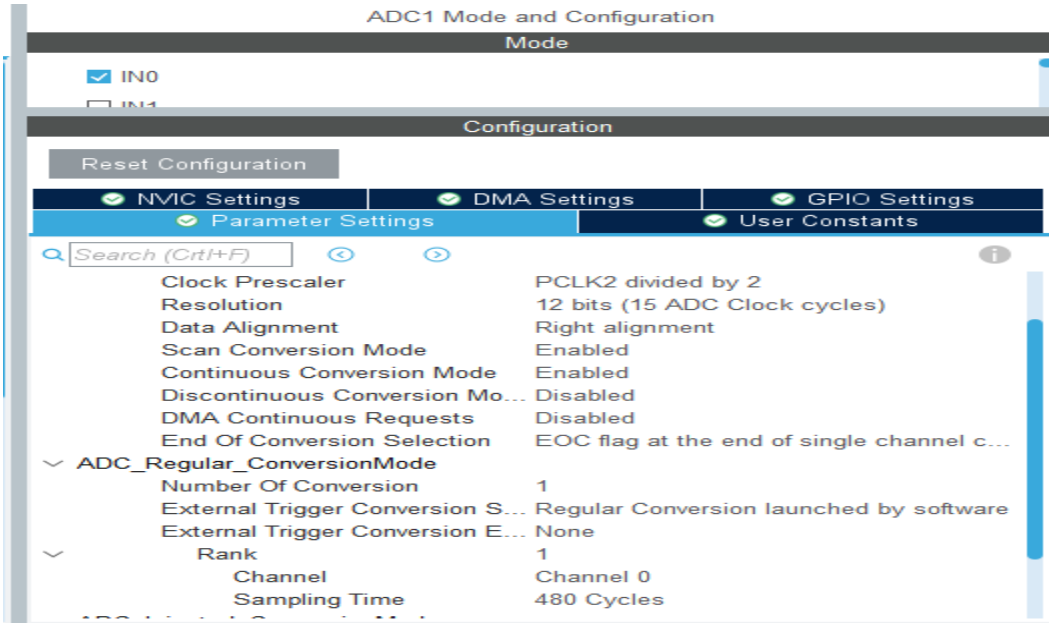
- مبدل آنالوگ به دیجیتال (صرفا یک چنل):

برای خواندن و نمونه برداری از سیگنال ورودی (که عملا یک سیگنال dtmf خواهد بود) استفاده می‌شود.

- یوزارت ٦:

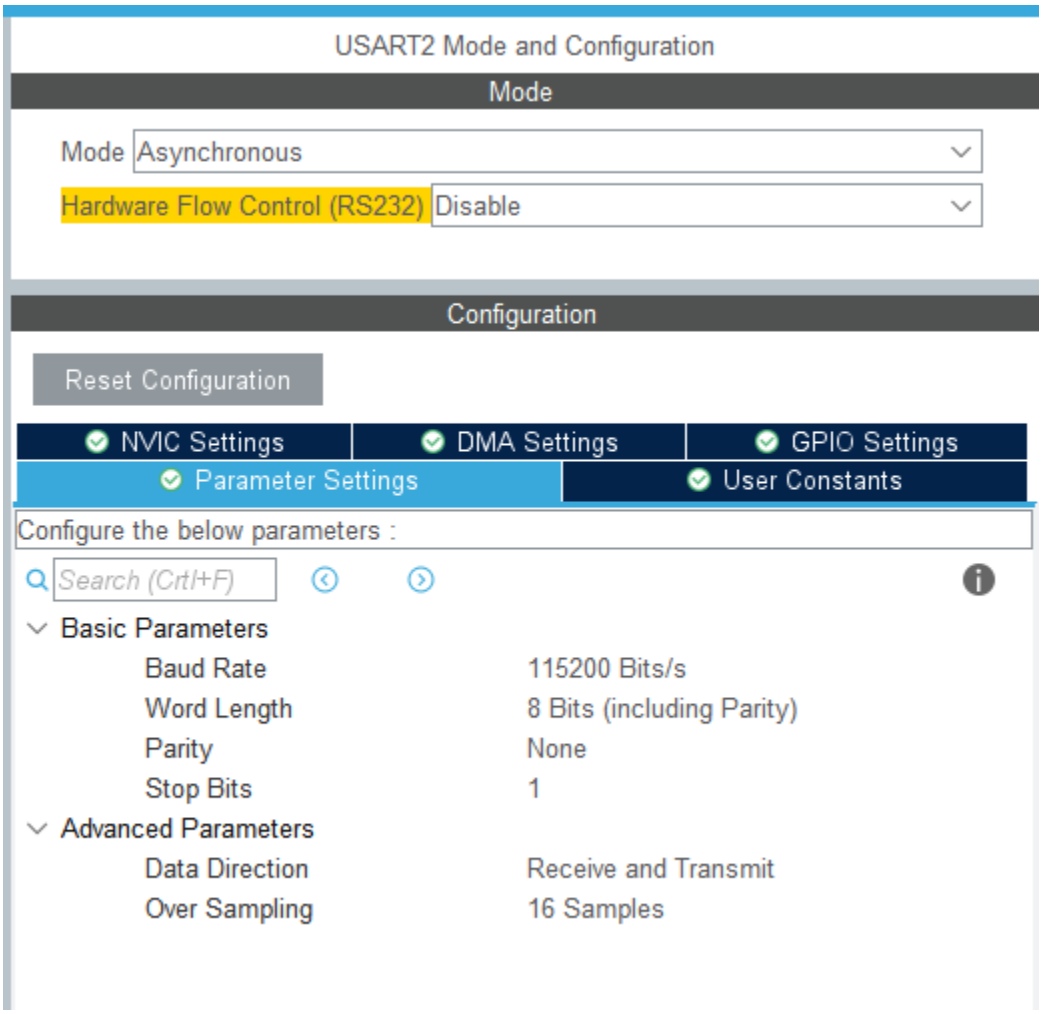
برای لاگ گرفتن و گزارش گردش کار میکرو استفاده می‌شود.( لاگ های خود را به واسطه یوزات ٦ به ترمینال ارسال می‌کنیم )

١) پارامترهای ست شده برای ADC :



از آنجا که Sampling Time برابر 480 کلاک است برای بدست آورد SamplingRate بایستی کلاک روی APB2 را پس از عبور از Prescaler تقسیم بر زمان نمونه‌گیری کرده تا فرکانس نمونه برداری مشخص شود (در این پروژه حدود 8300 هرتز)

۲) پارامترهای ست شده برای USART2




صرفا Baudrate را 115200 قرار می‌دهیم. در ترمینال مجازی نیز BaudRate را با همین مقدار ست می‌کنیم.

# بخش دوم: پیاده‌سازی

**در این بخش قسمت‌های مختلف کد شرح داده می‌شوند.**

برای پیاده‌سازی الگوریتم گوئرتزل داریم :

$$v(n) = 2\cos\left(2\pi {}^{f_0}/_{f_s}\right) v(n-1) + \ v(n-2) + x(n)$$

$$output = v^2(N) + v^2(N-1) - 2\cos\left(2\pi {}^{f_0}/_{f_s}\right) v(N)v(N-1)$$

در روابط بالا N تعداد نمونه‌ها، x(n) آرایه نمونه‌ها (به‌عبارتی سیگنالی که وجود فرکانس f0 در آن بررسی می‌شود) و fs نیز فرکانس نمونه‌گیری میباشد.

به طور کلی روال کار آن است که در بازه‌های خاصی N نمونه از سیگنال ورودی که روی پین PA0 قرار دارد می-خوانیم. سپس الگوریتم گوئرتزل را برای همه فرکانس‌های سطر و ستون اجرا می‌کنیم. سطر و ستونی که بیشترین انطباق را متناسب با خروجی گوئرتزل مربوطه دارند، معِین کاراکتری هستند که سیگنال صوت آن روی پین آنالوگ میکرو قرار گرفته‌است.

روابط بالا پایه و اساس کدهایی است که در ادامه ارائه می‌شوند.

```
// coeffk = 2cos(2PI(freq/sampling_freq))
#define f_697Hz     1.7300996575860355
#define f_770Hz     1.6722788385415637
#define f_852Hz     1.6012981177149554
#define f_941Hz     1.5173419552846932
#define f_1209Hz    1.2249619542608279
#define f_1336Hz    1.0681880511122386
#define f_1477Hz    0.8827313049309178
#define f_1633Hz    0.6660201856861759
#define f_1394Hz    0.993244825179317
#define f_1540Hz    0.7965165138339212
#define f_1704Hz    0.5641556617974589
#define f_1882Hz    0.30232660926717536
#define f_2418Hz    -0.4994682106134935
#define f_2672Hz    -0.8589742874610377
#define f_2954Hz    -1.220785443294959
#define f_3266Hz    -1.5564171122585517

#define VREF 5

#define N 114 // changing N affects all CoeffK factors
float samples[N];

#define A (0)       // output
#define B (1)       // output
#define C (2)       // output
#define D (3)       // output
#define E (4)       // output
#define F (5)       // output
#define G (6)      // output
#define MASK(x) (1UL << (x))
```

برای سهولت در ابتدای برنامه مقادیر ضریب کسینوسی در الگوریتم گوئرتزل را برای همه فرکانس‌های مورد نیازمان (سطرها، ستون‌ها و دوبرابر آن‌ها) محاسبه کرده و define می‌کنیم. مقدار VREF میکرو، و تعداد نمونه‌ها (N) را نیز define می‌کنیم. چند define دیگر هم داریم که برای کنترل سون‌سگمنتند که پیشتر در پروژه‌های قبلی روال‌کار با آن(پین‌ها و نیز توابع مربوط به آن مثل تابع Segment ) تفصیلا شرح داده‌است. لذا از شرح مجدد آن می‌پرهیزیم.

```
void logStr(char* string);
void logNum(uint32_t input);
void endLine();
```

چند تابع ساده برای لاگ گرفتن و کار با ترمینال(خارج از ملزومات پروژه) تعریف می‌کنیم. روال کار آن‌ها در پروژه هفتم و هشتم شرح داده شده‌است.

```
uint32_t Read_ADC_OSC(){
    uint32_t raw;
    HAL_ADC_Start(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
    raw = HAL_ADC_GetValue(&hadc1);
    HAL_ADC_Stop(&hadc1);
    return raw;
}
```

در این تابع صرفا یک نمونه از پین آنالوگ (PA0) را می‌خوانیم. لازم بذکر است ADC همانطور که در بخش اول آورده شده صرفا یک کانال دارد و روی Trigger By Software تنظیم شده. بدین‌ترتیب از طریق نرم‌افزار آنرا Start و سپس Poll کرده و در نهایت مقدار دیجیتال‌شده را می‌خوانیم، آنرا متوقف می‌سازیم و مقدار خوانده‌شده را بازمی‌گردانیم.

```
void sample(){
    for(short i = 0; i < N; i++){
        samples[i] = (((float) Read_ADC_OSC()) - 1401.0)* VREF/317.0;
    }
}
```

این تابع نیز همان تابع نمونه‌گیری میباشد. به تعداد N نمونه از ورودی خوانده، به درستی scale کرده (1401 مقداری است که adc هنگام وصل بودن پین آنالوگ به زمین میخواند، 317 نیز اختلاف بین مقادیر خوانده شده از پین آنالوگ در زمان-های اتصال به vdd و gnd است) و در آرایه نمونه‌ها قرار میدهیم. (عملا این تابع آرایه x(n) ذکر شده در فرمول را پر می‌کند)

```
uint32_t goertzel(float factor)  {
  float  v0, v1, v2;
  float v12, v22, v1v2;

    // base values for recursive equation
  v1  = 0;
  v2  = 0;

    // calculate all Vk values recursively
  for (int i = 0; i < N; i++) {
    v0 = (factor*v1)- v2 + samples[i];
    v2 = v1;
    v1 = v0;
  }
  v12  = v1*v1;
  v22  = v2*v2;
  v1v2 = v1*v2;
  return  (uint32_t)(v12 + v22 - (factor*v1v2)) ;
}
```

این الگوریتم نکته خاصی علاوه بر فرمول‌های ذکر شده ندارد، صرفا بر اساس فاکتور کسینوسی که به آن پاس می‌شود (مجموعه مقادیر فاکتورهای کسینوسی برای فرکانس‌های مختلف در ابتدای برنامه دیفاین شده‌اند) و نیز آرایه نمونه‌ها تابع بازگشتی ارائه شده در ابتدای این بخش را برای همه مقادیر 0 تا N محاسبه کرده و در نهایت از دو مقدار V(N) و V(N-1) (که می‌دانیم با توجه به ساختار بازگشتی تابع V برای محاسبه آنها مقدار تابع V برای همه نمونه‌ها بایستی محاسبه شود) برای تعیین خروجی تابع که یک عدد مثبت است استفاده می‌کنیم (این مقدار مثبت عملا اندازه یک عدد مختلط است)

```
uint32_t dtmf_detect()
```

این تابع ابتدا گوئرتزل را روی فرکانس سطرها محاسبه می‌کند، اگر هیچکدام از آنها از آستانه تعریف شده بیشتر نباشد پس به احتمال زیاد این صدا مربوط به dtmf نبوده و ارور برمی‌گردانیم. در غیراینصورت سطر با بیشترین مقدار گوئرتزل به عنوان سطر کاراکتری که سیگنال صوت آن وارد میکرو شده است، انتخاب می‌شود.

همین روال برای ستون‌ها انجام می‌شود. پس از آنکه سطر و ستون تعیین شدند، تعیین کاراکتر متناظر با سیگنال صوت ورودی سهل است.و در نهایت این مراحل را برای دوبرابر فرکانس سطرها و ستون‌ها اجرا می‌کنیم.(هارمونیک دوم) و در صورتی که نتیجه کاملا مشابه شود ارور برمی‌گردانیم. در غیر اینصورت سطر و ستون را لاگ زده و مجموع چهاربرابر سطرمنهای یک و ستون منهای یک را به‌عنوان خروجی برمی‌گردانیم (این مقدار عملا اندیس کاراکتر متناظر این سطر و ستون در آرایه digits می‌باشد) آرایه digtis به شکل زیر تعریف می‌شود:

```
char digits[16] = {
   '1', '2', '3', 'A',
   '4', '5', '6', 'b',
   '7', '8', '9', 'c',
   '*', '0', '#', 'd',
};
```

این ارایه صرفا کاراکترهای متناظر با اصوات شرح داده شده را داراست (منطبق بر تصویر درج‌شده در صورت پروژه)

```
uint32_t error(){
     logStr("Not a Valid Tone!");
     endLine();
   return 100;
}
```

برای ارور برگرداندن (در تابع dtmf_detect ) از این تابع(error) استفاده می‌کنیم. صرفا یک لاگ را از طریق یوزارت روی ترمینال فرستاده و عدد مربوط به خطا را (100) باز می‌گرداند.

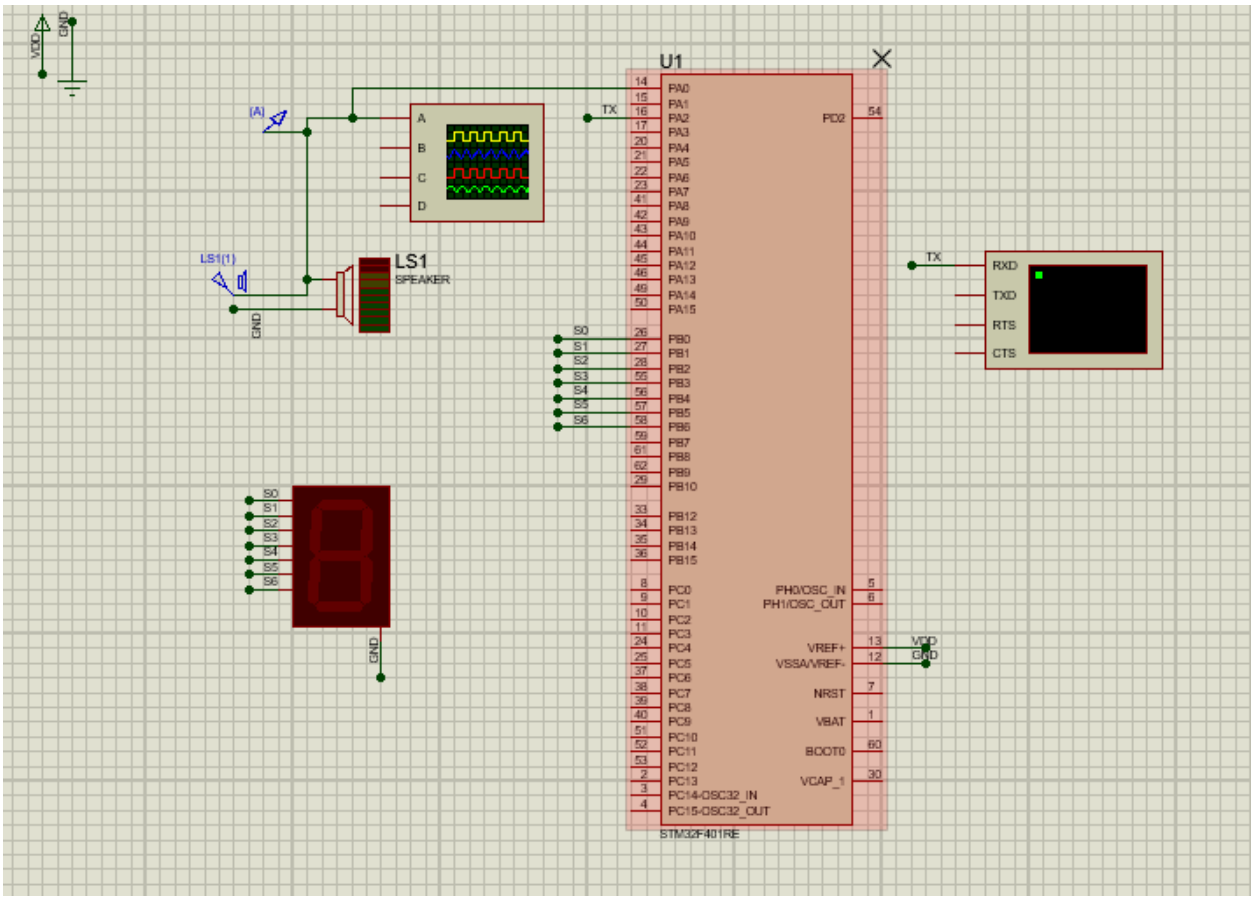
```
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
      sample();
      uint32_t digit = dtmf_detect();
      if( digit != 100){
          logStr("7segment updated"); endLine();
          GPIOB->ODR = 0;
          GPIOB->ODR = segment(digits[digit]);
      }
      HAL_Delay(290);

}
```

در حلقه اصلی برنامه به‌طور متناوب و با یک تاخیر منطقی، از سیگنال ورودی نمونه‌گیری کرده و وجود اصوات dtmf را بررسی می‌کنیم. درصورتی که این تابع مقدار 100 که همان کد ارور است را برنگرداند یعنی صوت جدید شناسایی شده و سون‌سگمنت بایستی آپدیت شود.

## بخش سوم : صحت‌سنجی

برای صحت‌سنجی عملکرد میکرو؛ یک مجموعه صوت مربوط به همه کاراکترهای مجاز dtmf در یک پوشه به‌نام tones قرار داده‌شده که می‌توان سیگنال آن‌ها را یکی پس از دیگری توسط جنریتور پروتئوس وارد پین PA0 کرده و نتیجه را بررسی کرد. به علاوه، دو صوت که شامل دنباله‌ای از اصوات dtmf هستند نیز به همین منظور قرارداده شده‌است. دو ویدیو نیز از تست میکرو در پوشه Testing Videos قرار گرفته‌است.

## بخش چهارم : پروتئوس



# بخش پنجم: منابع

برای تولید اصوات **dtmf**


1. http://dialabc.com/sound/generate/index.html?pnum=AB+CD+*+%23&pad=ext&auFormat=wavpcm44&toneLength=300&mtcontinue=Generate+DTMF+Tones
2. https://forum.arduino.cc/t/goertzel-for-reliable-dtmf-decoding/118501/6
3. https://eng.libretexts.org/Bookshelves/Electrical_Engineering/Signal_Processing_and_Modeling/Fast_Fourier_Transforms_(Burrus)/04%3A_The_DFT_as_Convolution_or_Filtering/4.04%3A_Goertzel's_Algorithm_or_A_Better_DFT_Algorithm
4. https://www.ti.com/lit/an/spra066/spra066.pdf?ts=1656189312971
5. https://medium.com/geekculture/the-great-unknown-the-goertzel-algorithm-492681f30866
6. https://www.quora.com/Whats-the-simplest-DTMF-decoding-algorithm
7. https://www.youtube.com/watch?v=2Aj3HTF9yKQ
8. https://www.youtube.com/watch?v=LS_9Jbpor10
9. https://www.youtube.com/results?search_query=goertzel+algorithm
10. https://en.wikipedia.org/wiki/Goertzel_algorithm#DFT_computations
11. https://github.com/siorpaes/DTMF-decoder
12. https://gist.github.com/sebpiq/4128537
13. https://github.com/ribt/dtmf-decoder