```
#include "myheader.h"
#include <stdio.h>

uint8_t count = 0;
int flag = 0;

extern const char img[1024];
extern const char img1[1024];

int main(void)
{
 SysClockConfig();
 GPIO_Init();
 TIM6Config ();
 GLCD_Animate();
 while(1);
 }
```

۱۲ مبدل آنالوگ به دیجیتال

یک مبدل ADC (آنالوگ به دیجیتال) یک مدار الکترونیکی است که ولتاژ آنالوگ را به عنوان ورودی می‌گیرد و آن را به داده‌های دیجیتال تبدیل می‌کند. ADC طی فرآیند کوانتیزاسیون از ورودی آنالوگ نمونه‌برداری می‌کند تا کد باینری نظیر سطح ولتاژ آنالوگ را ارایه دهد.

ADC یکی از پرهزینه‌ترین اجزای الکترونیکی است، به‌ویژه زمانی که دارای نرخ نمونه‌برداری بالا و وضوح بالا باشد. بنابراین، این یک منبع ارزشمند در میکروکنترلرها است و تولیدکنندگان مختلف امکانات متنوعی را برای استفاده بهتر ارایه میدهند.

در stm32f407 از مبدل آنالوگ به دیجیتال ۱۲ بیتی استفاده می‌شود، این ADC دارای حداکثر ۱۹ کانال چندگانه است که به آن اجازه می‌دهد سیگنال‌ها را از ۱۶ منبع خارجی، دو منبع داخلی و کانال VBAT اندازه‌گیری کند. تبدیل A/D کانال‌ها می‌تواند در حالت‌های تک، پیوسته، اسکن یا ناپیوسته انجام شود. نتایج ADC در یک رجیستر داده ۱۶ بیتی با تراز چپ یا راست ذخیره می‌شود.

ویژگی‌های اصلی ADC

- وضوح قابل تنظیم ۱۲ بیتی، ۱۰ بیتی، ۸ بیتی یا ۶ بیتی
- تولید وقفه در پایان تبدیل، پایان تبدیل تزریقی، و در صورت وقوع رویدادهای نظارت آنالوگ یا سرریز
- حالت‌های تبدیل تک و پیوسته
- حالت اسکن برای تبدیل خودکار از کانال ۰ تا کانال ’n‘
- تراز داده با هم‌خوانی داخلی
- زمان نمونه‌برداری قابل برنامه‌ریزی به‌صورت کانال به کانال
- گزینه تحریک خارجی با قطبیت قابل تنظیم برای هر دو تبدیل معمولی و تزریقی
- حالت ناپیوسته
- حالت دوگانه/سه‌گانه (در دستگاه‌هایی با ۲ ADC یا بیشتر)
- ذخیره‌سازی داده‌های DMA قابل تنظیم در حالت Dual/Triple ADC
- تأخیر قابل تنظیم بین تبدیل‌ها در حالت بینابینی دوگانه/سه‌گانه

- نوع تبدیل ADC (به برگه‌های مشخصات مراجعه کنید)
- الزامات تغذیه ADC: 2.4 ولت تا ۳.۶ ولت در سرعت کامل و تا ۱.۸ ولت در سرعت پایین‌تر
- دامنه ورودی ADC: VREF– ≤ VIN ≤ VREF+
- تولید درخواست DMA در حین تبدیل کانال معمولی"

[redacted]پالس ساعت ADC

"ADC دارای دو طرح ساعت است:

- ساعت برای مدار آنالوگ: ADCCLK، که برای تمامی ADCها مشترک است.

این ساعت از ساعت APB2 تولید می‌شود و با یک تقسیم‌کننده قابل برنامه‌ریزی تقسیم می‌شود که به ADC اجازه می‌دهد با fPCLK2/2، 4/، 6/ یا ۸/ کار کند. برای حداکثر مقدار ADCCLK به برگه‌های مشخصات مراجعه کنید.

- ساعت برای رابط دیجیتال (استفاده شده برای دسترسی به خواندن/نوشتن رجیسترها)

این ساعت برابر با ساعت APB2 است. ساعت رابط دیجیتال می‌تواند به‌صورت جداگانه برای هر ADC از طریق رجیستر فعال‌سازی ساعت محیطی RCC APB2 (RCC_APB2ENR) فعال یا غیرفعال شود."

[redacted]انواع تبدیل در ADC

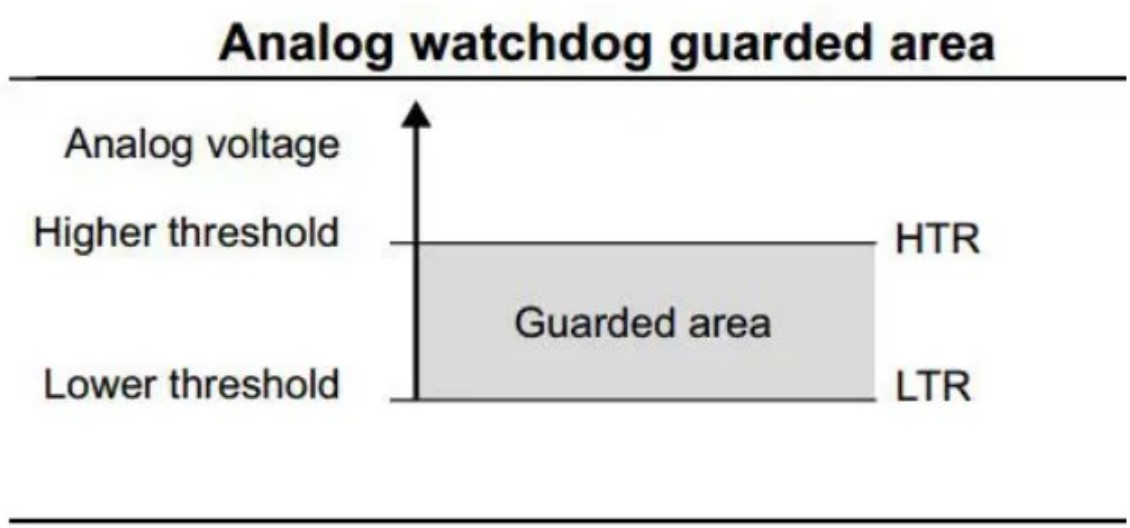
تبدیل ADC به صورت دو گروه تبدیل عادی[5] و تبدیل تزریقی[6]انجام میگیرد. یک گروه شامل یک توالی از تبدیل‌هاست که می‌تواند بر روی هر کانال و به هر ترتیبی انجام شود. به عنوان مثال، می‌توان تبدیل را به شکل زیر انجام داد: Ch3، Ch8، Ch2، Ch2، Ch0، Ch2، Ch2، Ch15.

گروه تبدیل عادی شامل حداکثر ۱۶ تبدیل است و به طور مستقل از یکدیگر کار میکنند واگر تبدیل تزریقی رخ داد متوقف شده و پس از پایان تبدیل تزریقی به روال عادی خود ادامه می‌دهد. نتیجه تمام تبدیل ها به ترتیب در یک رجیستر ذخیره می‌گردد.

گروه تزریقی شامل حداکثر ۴ تبدیل است و از طریق سیگنال خارجی فعال می‌شود و میتواند به صورت ترکیبی با تبدیل عادی نیز کار کند. در این روش نتیجه هر تبدیل در رجیستر جداگانه ای ذخیره می‌شود و در مواقعی که به سرعت بالاتری نیاز هست استفاده می‌شود.

[5] Regular Conversion
[6] Injected Conversion

د ر ADC مانند شکل ۱-۱۲ امکان بررسی سطح ولتاژ کانال آنالوگ وجود دارد و چنانچه از محدوده قابل تنظیم مجاز خارج شود وقفه ای تولید خواهد شد که قابلیت نظارت آنالوگ (Analog Watchdog) نامیده می‌شود.



شکل ۱-۱۲: نظارت آنالوگ در مبدل ADC

با توجه به ۱۲ بیتی بودن ADC و وجود رجیسترهای ۱۶ بیتی، میتوان داده ها را مانند شکل ۲-۱۲ به سمت چپ یا راست منتقل کرد. در کانال‌های گروه تزریقی، کاربر میتواند مقدار افستی (بیت SEXT) را به داده اضافه نماید که از طریق رجیسترهای ADC_JOFRx قابل تنظیم است.

شکل ۲-۱۲: تراز به چپ و راست در تبدیل های ADC

حالت‌های عملکرد ADC STM32

حالت‌های زیادی برای عملکرد ADC STM32 وجود دارد که به طراحان/برنامه‌نویسان سیستم، انعطاف‌پذیری برای پیکربندی آن به هر شکلی که نیازهای برنامه را برآورده کند، می‌دهد. این امر به قیمت داشتن یک قطعه سخت‌افزاری بسیار پیچیده است که می‌تواند به روش‌های مختلف پیکربندی شود.

۱۲/۴/۱ حالت تبدیل تک (Single Conversion Mode)

در حالت تبدیل تک، ADC یک تبدیل انجام می‌دهد. این حالت با تنظیم بیت ADON در رجیستر ADC_CR2 (فقط برای یک کانال عادی) یا با یک تحریک خارجی (برای کانال عادی یا تزریقی) آغاز می‌شود، در حالی که بیت CONT برابر ۰ است. پس از اتمام تبدیل کانال انتخاب‌شده:

اگر یک کانال عادی تبدیل شده باشد:

- داده‌های تبدیل‌شده در رجیستر ۱۶ بیتی ADC_DR ذخیره می‌شود.

- پرچم EOC (پایان تبدیل) تنظیم می‌شود.

- و یک وقفه ایجاد می‌شود اگر بیت EOCIE تنظیم شده باشد.

اگر یک کانال تزریقی تبدیل شده باشد:

- داده‌های تبدیل‌شده در رجیستر ۱۶ بیتی ADC_DRJ1 ذخیره می‌شود.

- پرچم JEOC (پایان تبدیل تزریقی) تنظیم می‌شود.

- و یک وقفه ایجاد می‌شود اگر بیت JEOCIE تنظیم شده باشد.

سپس ADC متوقف می‌شود.

۱۲٫۴٫۲ حالت تبدیل مداوم (Continuous Conversion Mode)

در حالت تبدیل مداوم، ADC به محض اتمام یک تبدیل، تبدیل دیگری را آغاز می‌کند. این حالت با یک تحریک خارجی یا با تنظیم بیت ADON در رجیستر ADC_CR2 آغاز می‌شود، در حالی که بیت CONT برابر ۱ است. پس از هر تبدیل:

- اگر یک کانال عادی تبدیل شده باشد:

- داده‌های تبدیل‌شده در رجیستر ۱۶ بیتی ADC_DR ذخیره می‌شود.

- پرچم EOC (پایان تبدیل) تنظیم می‌شود.

- و یک وقفه ایجاد می‌شود اگر بیت EOCIE تنظیم شده باشد.

- اگر یک کانال تزریقی تبدیل شده باشد:

- داده‌های تبدیل‌شده در رجیستر ۱۶ بیتی ADC_DRJ1 ذخیره می‌شود.

- پرچم JEOC (پایان تبدیل تزریقی) تنظیم می‌شود.

- و یک وقفه ایجاد می‌شود اگر بیت JEOCIE تنظیم شده باشد.

۱۲/۴/۳ حالت اسکن (Scan Mode)

این حالت برای اسکن یک گروه از کانال‌های آنالوگ استفاده می‌شود. تبدیل برای تمام کانالهایی که در گروه انتخاب شده اند انجام میگیرد. اگر بیت CONT تنظیم شده باشد، پس از آخرین تبدیل مجددا از ابتدا و اولین کانال گروه ادامه می‌یابد.

در هنگام استفاده از حالت اسکن، برای کانالهای گروه عادی بایستی بیت DMA تنظیم شود تا داده های رجیستر ADC_DR را به بخش مشخصی از حافظه منتقل نماید. ولی در کانال‌های تزریقی داده ها همیشه در رجیسترهای ADC_JDRx ذخیره می‌شود.

۱۲/۴/۴ حالت ناپیوسته (Discontinuous Mode)

این حالت با تنظیم بیت DISCEN در رجیستر ADC_CR1 فعال می‌شود. می‌توان از آن برای تبدیل یک توالی کوتاه از n تبدیل (n <= 8) که بخشی از توالی تبدیل‌های انتخاب‌شده در رجیسترهای ADC_SQRx است، استفاده کرد. مقدار n با نوشتن در بیت‌های DISCNUM[۲:۰] در رجیستر ADC_CR1 مشخص می‌شود.

هنگامی که یک تریگر خارجی رخ می‌دهد، تبدیل‌های n تایی انتخاب‌شده در رجیسترهای ADC_SQRx آغاز می‌شود و طول کل توالی توسط بیت‌های L[۳:۰] در رجیستر ADC_SQR1 تعریف می‌شود. بدین وسیله کاربر میتواند در مواردی که نیاز فوری به مقدار یک کانال دارد براحتی از این قابلیت استفاده نماید.

ترجمه:

تریگرهای خارجی و داخلی ADC STM32

گزینه پیش‌فرض برای شروع فرآیند تبدیل ADC STM32، منبع تریگر نرم‌افزاری است. بنابراین، هر بار که می‌خواستیم یک تبدیل جدید ADC آغاز کنیم، باید به صورت دستی تابع `HAL_ADC_Start()` را فراخوانی می‌کردیم. با این حال، ADC همچنین می‌تواند به‌طور خودکار توسط منابع تریگر داخلی یا خارجی در خود میکروکنترلر STM32 فعال شود.

لذا میتوان تبدیل ADC به‌طور دوره‌ای با استفاده از یک تریگر تایمر تنظیم نمود تا به نرخ نمونه‌برداری دلخواه ADC دست یابیم. یا برای شروع تبدیل ADC در یک زمان خاص نسبت به یک سیگنال PWM خروجی (که ویژگی بسیار مهمی برای سیستم‌های اندازه‌گیری پیشرفته است) تریگر شود.

کالیبراسیون ADC STM32

ADC دارای یک حالت خودکالیبراسیون داخلی است. کالیبراسیون به‌طور قابل‌توجهی خطاهای دقت را به دلیل تغییرات در بانک خازن‌های داخلی کاهش می‌دهد. در طول کالیبراسیون، یک کد تصحیح خطا (کلمه دیجیتال) برای هر خازن محاسبه می‌شود و در تمام تبدیل‌های بعدی، سهم خطای هر خازن با استفاده از این کد حذف می‌شود.

کالیبراسیون با تنظیم بیت CAL در رجیستر ADC_CR2 آغاز می‌شود. پس از اتمام کالیبراسیون، بیت CAL به‌طور خودکار توسط سخت‌افزار بازنشانی می‌شود و تبدیل‌های عادی می‌توانند انجام شوند. توصیه می‌شود ADC یک‌بار در هر زمان روشن شدن کالیبره شود. کدهای کالیبراسیون به محض پایان مرحله کالیبراسیون در رجیستر ADC_DR ذخیره می‌شوند.

STM32 HALعملکردی را در داخلAPI های ADC ارائه می کند که برای شروع فرآیند کالیبراسیون اختصاص داده شده است و همانطور که قبلاً گفته شد این یک مرحله توصیه شده پس از راه اندازی سخت افزار ADC در هنگام روشن شدن سیستم است.

نمونه‌برداری ADC STM32

ADC STM32 ولتاژ ورودی را برای تعداد مشخصی از دوره‌های `ADC_CLK` نمونه‌برداری می‌کند که می‌توان آن را با استفاده از بیت‌های `SMP[۲:۰]` در رجیسترهای `ADC_SMPR1` و `ADC_SMPR2` تغییر داد. همچنین هر کانال می‌تواند با زمان‌های نمونه‌برداری متفاوتی نمونه‌برداری شود.

این امکان به طراحان اجازه می‌دهد تا زمان‌های نمونه‌برداری را بر اساس نیازهای خاص برنامه یا ویژگی‌های سیگنال ورودی تنظیم کنند، که می‌تواند به بهبود دقت و عملکرد سیستم کمک کند.

Total ADC Conversion Time(Tconv)= Sampling time + 12.5 cycles
SamplingRate = 1 / Tconv

اگر کلاک ADC برابر با 14MHz و زمان نمونه برداری برابر با 1.5 سیکل باشد، زمان کل تبدیل برابر با ۱۴ سیکل خواهد شد که برابر با 1us است.

دقت و ولتاژ مرجع ADC STM32

۱۲/۸/۱ دقت ADC STM32

ADC STM32 دارای دقت ۱۲ بیتی است که منجر به زمان تبدیل کلی معادل `SamplingTime + 12.5` دوره‌ی کلاک می‌شود. با این حال، می‌توان با فدای دقت بالا، نرخ‌های نمونه‌برداری بالاتری به‌دست آورد. بنابراین، دقت می‌تواند به ۱۰ بیت، ۸ بیت یا ۶ بیت کاهش یابد و در نتیجه زمان تبدیل بسیار کوتاه‌تر شده و نرخ نمونه‌برداری افزایش می‌یابد.

این تنظیمات می‌تواند توسط برنامه‌نویس در نرم‌افزار پیکربندی و پیاده‌سازی شود و STM32 HAL APIهایی برای تنظیم تمام پارامترهای ADC از جمله دقت آن ارائه می‌دهد.

۱۲/۸/۲ ولتاژ مرجع ADC

پین‌های ولتاژ مرجع ADC در دیتاشیت تعریف شده‌اند و فرض بر این است که به یک سطح ولتاژ در یک محدوده مشخص متصل شده‌اند. با تغییر حداکثر سطح مجاز ولتاژ مرجع میتوان دقت اندازه گیری را تغییر داد. افزایش ولتاژ مرجع سبب کاهش دقت اندازه گیری و بالعکس میگردد.

Vin = ADC_Res x (Reference Voltage / 4096) v
Reference Voltage = (V_{REF}+) – (V_{REF}-)

وقفه‌های ADC در STM32

در STM32، وقفه‌ها می‌توانند در پایان تبدیل برای گروه‌های عادی و تزریق‌شده تولید شوند و همچنین زمانی که بیت وضعیت نظارت آنالوگ تنظیم می‌شود. برای انعطاف‌پذیری بیشتر، بیت‌های جداگانه‌ای برای فعال‌سازی وقفه‌ها در نظر گرفته شده است.

انواع وقفه‌ها:

۱. وقفه پایان تبدیل (End of Conversion Interrupt) :

- این وقفه زمانی فعال می‌شود که یک تبدیل ADC به پایان می‌رسد و می‌تواند برای پردازش داده‌های جدید استفاده شود. این وقفه برای گروههای عادی و تزریقی جداگانه ایجاد می‌شود

۲. وقفه نظارت آنالوگ (Analog Watchdog Interrupt) :

- این وقفه زمانی فعال می‌شود که ولتاژ ورودی از محدوده مشخص‌شده خارج شود، که می‌تواند برای نظارت بر شرایط خاص یا ایمنی کاربردی مفید باشد.

۳. وقفه (Overrun Interrupt)

این وقفه زمانی فعال می‌شود که داده‌های جدید در رجیستر ADC به دلیل عدم خواندن داده‌های قبلی از دست بروند. این وقفه برای جلوگیری از دست رفتن داده‌ها و اطمینان از خواندن و پردازش درست داده‌ها اهمیت دارد.

۱۲٫۹٫۱ پیکربندی وقفه‌ها:

برنامه‌نویسان می‌توانند با استفاده از APIهای STM32 HAL، این وقفه‌ها را پیکربندی و مدیریت کنند. این امکان به شما اجازه می‌دهد تا برنامه خود را بر اساس نیازهای خاص پروژه تنظیم کنید و از عملکرد بهینه ADC بهره‌مند شوید.

ADC interrupts

Interrupt event	Event flag	Enable control bit
End of conversion of a regular group	EOC	EOCIE
End of conversion of an injected group	JEOC	JEOCIE
Analog watchdog status bit is set	AWD	AWDIE
Overrun	OVR	OVRIE

خواندن ADC در STM32 (پولینگ، وقفه، DMA)

به طور کلی، سه روش مختلف برای خواندن نتیجه تبدیل ADC در STM32 پس از اتمام تبدیل وجود دارد. در این بخش، هر روش را به‌طور مختصر توضیح خواهیم داد .

۱. پولینگ ADC STM32

این ساده‌ترین روش برای انجام تبدیل آنالوگ به دیجیتال با استفاده از ADC در یک کانال ورودی آنالوگ است. با این حال، این روش در همه موارد کارآمد نیست، زیرا به عنوان یک روش مسدودکننده برای استفاده از ADC در نظر گرفته می‌شود. در این روش، ما تبدیل A/D را آغاز می‌کنیم و منتظر می‌مانیم تا ADC تبدیل را کامل کند تا CPU بتواند به پردازش کد اصلی ادامه دهد.

۲. وقفه ADC STM32

روش وقفه یک روش کارآمد برای انجام تبدیل ADC به صورت غیرمسدودکننده است، بنابراین CPU می‌تواند به اجرای روال کد اصلی ادامه دهد تا زمانی که ADC تبدیل را کامل کرده و سیگنال وقفه‌ای را ارسال کند تا CPU بتواند به زیربرنامه ISR سوئیچ کند و نتایج تبدیل را برای پردازش‌های بعدی ذخیره کند.

با این حال، وقتی با چندین کانال در حالت دایره‌ای یا مشابه کار می‌کنید، وقفه‌های دوره‌ای از ADC خواهید داشت که برای CPU بسیار زیاد است. این موضوع می‌تواند باعث ایجاد لرزش، تأخیر در وقفه و انواع مشکلات زمان‌بندی در سیستم شود. این مشکل می‌تواند با استفاده از DMA رفع گردد.

۳. DMA ADC STM32

از آنجا که مقادیر کانال‌های عادی تبدیل‌شده در یک رجیستر داده منحصر به فرد ذخیره می‌شوند، استفاده از DMA برای تبدیل بیش از یک کانال عادی ضروری است بدین ترتیب داده‌های ذخیره‌شده در رجیستر `ADC_DR` دچار اشکار نمی‌شوند. لذا در پایان تبدیل یک کانال عادی، درخواست DMA فعال می‌شود و داده‌های تبدیل‌شده از رجیستر `ADC_DR` به مکان مقصد انتخاب‌شده توسط کاربر منتقل شوند.

در نهایت، روش DMA کارآمدترین روش برای تبدیل چندین کانال ADC با نرخ‌های بسیار بالا است و همچنان نتایج را بدون دخالت CPU به حافظه منتقل می‌کند که یک تکنیک بسیار جالب و صرفه‌جویانه در زمان است.

خطاهای ADC

در مبدلهای ADC دو دسته خطای اصلی وجود دارد که ناشی از خود منبع و ناشی از محیط است که به طور خلاصه دسته بندی شده است.

۱۲,۱۱,۱ خطاهای ناشی از خود ADC

1.1 – خطای جابجایی ADC به نام Offset error

1.2 – خطای بهره ADC به نام Gain error

1.3 – خطای هم‌خطی انتگرالیIntegral Non-Linearity

۱۲,۱۱,۲ خطاهای ناشی از محیط

2.1 – نویز ولتاژ مرجع ADC

2.2 – نویز سیگنال ورودی آنالوگ

2.3 – عدم تطابق دامنه دینامیکی ADC

2.4 – امپدانس منبع سیگنال آنالوگ (مقاومت)

2.5 – ظرفیت و پارازیت‌های منبع سیگنال آنالوگ

2.6 – اثر جریان تزریقی

2.7 – تداخل متقابل پین‌های ورودی/خروجی

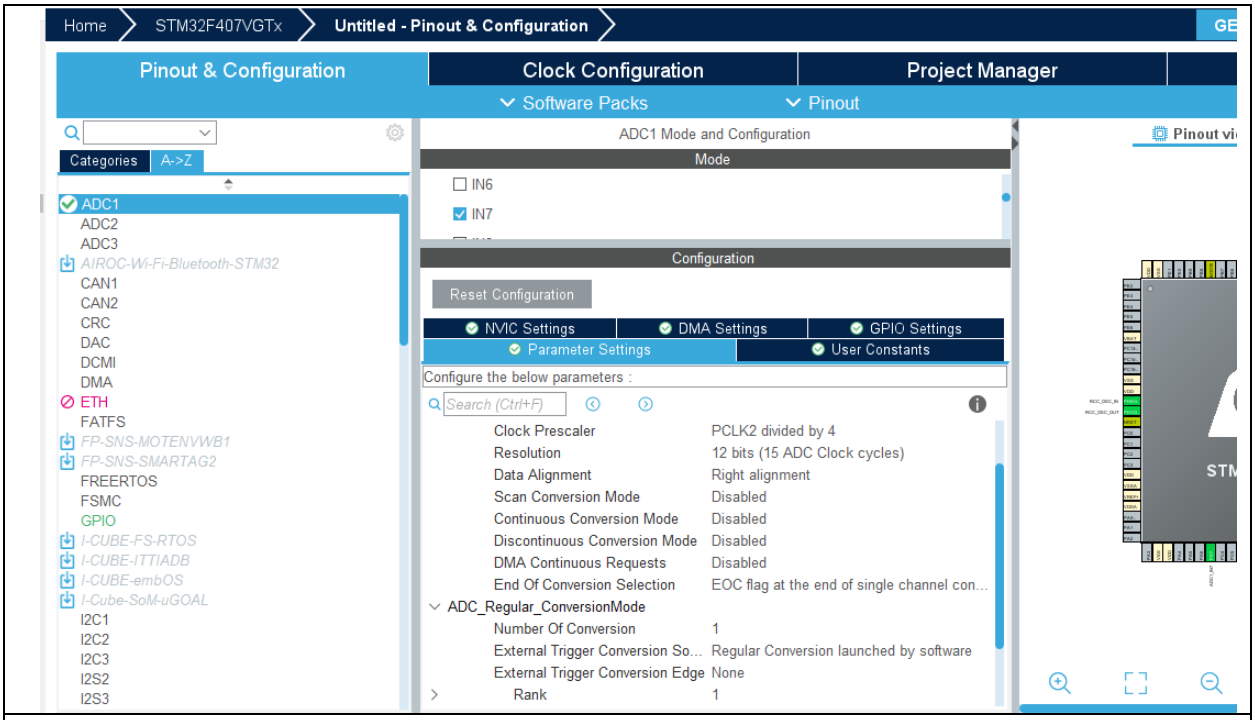
2.8 – نویز ناشی از EMI

[redacted]پروژه خواندن مقدار آنالوگ ورودی

۱۲,۱۲,۱ پروژه خواندن مقدار آنالوگ ورودی با روش polling

در این پروژه محتوای کانال ۷ از ADC خوانده شده و روی LCD نشان داده می‌شود. خواندن با استفاده از روش Polling انجام می‌شود.

در STM32CubeMX بایستی تنظیمات مربوط به کلاک و اتصالات مربوطه به LCD انجام شود و تنظیمات بخش ADC مانند شکل ۳-۱۲ می‌باشد.



شکل ۳-۱۲: نمایی از تنظیمات ADC به روش polling

کد های ایجاد شده در محیط keil را مشابه برنامه ۱-۱۲ تغییر دهید.

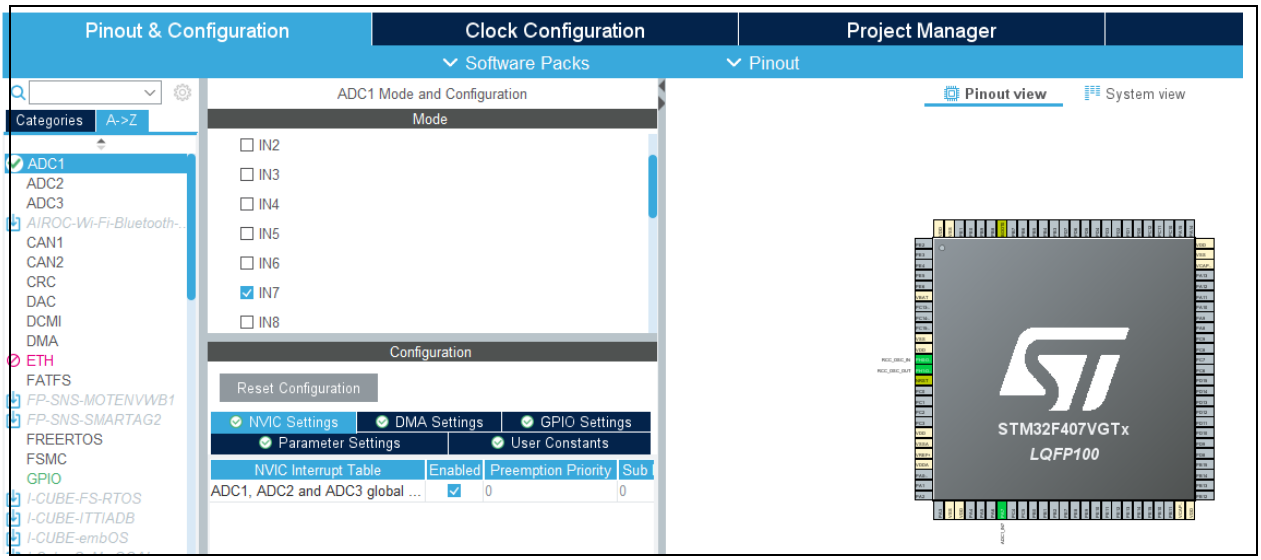
```
#include "main.h"
#include "STM_MY_LCD16X2.h"

ADC_HandleTypeDef hadc1;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);

int main(void)
{
  uint16_t AD_RES = 0;

  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_ADC1_Init();

  LCD1602_Begin4BIT(GPIOE, GPIO_PIN_0, GPIO_PIN_1, GPIOE, GPIO_PIN_4, GPIO_PIN_5,
GPIO_PIN_6, GPIO_PIN_7);
```

```
        HAL_Delay(500);
        LCD1602_clear();
        LCD1602_print("Microlab ");

  while (1)
  {
        HAL_ADC_Start(&hadc1);
        HAL_ADC_PollForConversion(&hadc1, 1);
        AD_RES = HAL_ADC_GetValue(&hadc1);

        LCD1602_clear();
        LCD1602_print("ADC(7)= ");
        LCD1602_PrintInt(AD_RES);
        HAL_Delay(500);
  }

}
```

برنامه ۱-۱۲: نمونه برنامه در محیط keilبرای ADC

۱۲٫۱۲٫۲ پروژه خواندن مقدار آنالوگ ورودی با روش وقفه

تمام تنظیمات ADC همانطور که هستند باقی خواهند ماند، اما باید وقفه را از تب کنترل‌گر NVIC مانند شکل ۴-۱۲ فعال گردد.



شکل ۴-۱۲: نمایی از تنظیمات وقفه ADC

کد های ایجاد شده در محیط keil را مشابه برنامه ۲-۱۲ تغییر دهید.

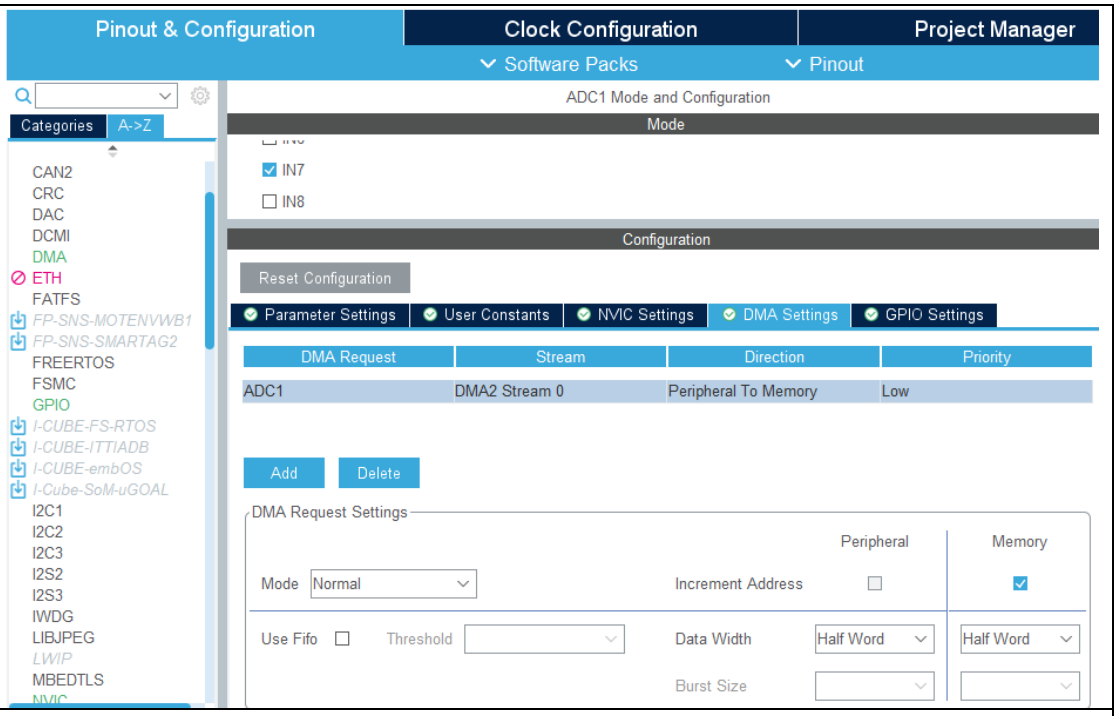
```
#include "main.h"
#include "STM_MY_LCD16X2.h"

 uint16_t AD_RES = 0;
 char adcupdate=0;
 ADC_HandleTypeDef hadc1;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);


void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
   // Read & Update The ADC Result
   AD_RES = HAL_ADC_GetValue(&hadc1);
            adcupdate=1;
}

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_ADC1_Init();

  LCD1602_Begin4BIT(GPIOE, GPIO_PIN_0, GPIO_PIN_1, GPIOE, GPIO_PIN_4, GPIO_PIN_5,
GPIO_PIN_6, GPIO_PIN_7);

  while (1)
  {

    HAL_ADC_Start_IT(&hadc1);
                    if(adcupdate==1){
                              adcupdate=0;
                              LCD1602_clear();
                              LCD1602_print("ADC(9)= ");
     LCD1602_PrintInt(AD_RES);
                    }
       HAL_Delay(500);
          }
```

```
}
```

برنامه ۲-۱۲: کدهای ADC با استفاده از وقفه

herez

تمام تنظیمات ADC در حالت عادی به صورت پیش‌فرض خواهند بود. با این حال، این بار وقفه‌های ADC در STM32 فعال نیستند و به جای آن، DMA پیکربندی شده است و وقفه DMA به طور پیش‌فرض در تب کنترل‌گر NVIC فعال است. پیکربندی‌های DMA STM32 برای ADC به صورت زیر خواهد بود. فقط یک کانال DMA اضافه کنید و همین!




```
#include "main.h"
#include "STM_MY_LCD16X2.h"

uint32_t AD_RES = 0;
int average = 0;
char index1 = 1;

ADC_HandleTypeDef hadc1;
```

```
DMA_HandleTypeDef hdma_adc1;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_ADC1_Init(void);


void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
// Conversion Complete & DMA Transfer Complete As Well
average = (average*(index1-1)+AD_RES)/index1;
}

int main(void)
{

HAL_Init();
SystemClock_Config();
MX_GPIO_Init();
MX_DMA_Init();
MX_ADC1_Init();

LCD1602_Begin4BIT(GPIOE, GPIO_PIN_0, GPIO_PIN_1, GPIOE, GPIO_PIN_4,
GPIO_PIN_5, GPIO_PIN_6, GPIO_PIN_7);


while (1)
{

HAL_ADC_Start_DMA(&hadc1, &AD_RES, 1);
index1++;
if(index1==100)
{
index1=1;

LCD1602_clear();
LCD1602_print("ADC(7)= ");
LCD1602_PrintInt(AD_RES);
}
HAL_Delay(1);
}
```