STM32CubeIDE ACTIVE

Integrated Development Environment for STM32

Speed up the development and debug of your application

STM32 CubeIDE

STM32 SWD(Serial Wire Debug): debug

SWV(*Serial Wire Viewer: SWD+SWO*) + ***ITM***(*Instrumentation Trace Macrocell*) + *printf()*

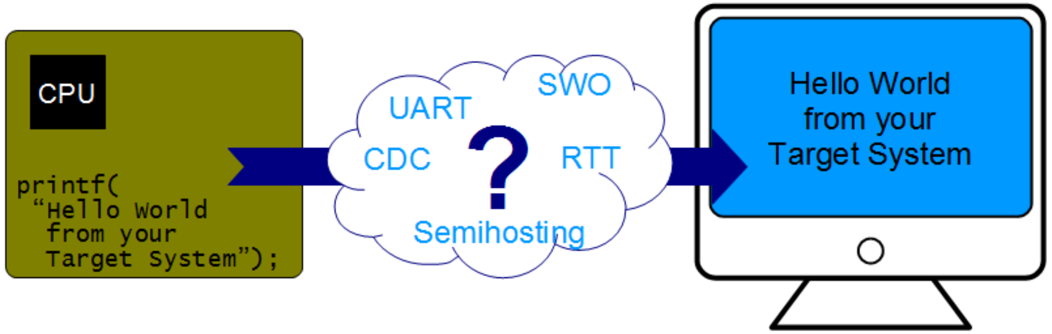
CMSIS

STM32 CMSIS DSP Library

* Debugging

There are some debug techniques used to inspect the firmware running on ARM-based MCUs:

- Semihosting: built-in to every ARM chips, need adding additional library and running in debug mode.
- Console log: forward to a native UART port, a Virtual COM port through a USB port.
- Serial Wire View (SWV): fast output over dedicated Single Wire Output (SWO) pin, but it's only available on Cortex-M3+, and this is uni-direction communication.
- Real Time Transfer (RTT): extremely fast but only work with SEGGER Debugger, can have a real-time bi-direction communication.



* Serial Wire Viewer

- Cortex-M based microcontrollers integrate some debugging and tracing technologies, including ***JTAG*** and ***SWD***.
- Tracing allows exporting in real-time internal activities performed by the CPU.
- The ***Instrumentation Trace MacroCell (ITM)*** allows sending software-generated debug messages through a specific signal I/O named Serial Wire Output (SWO).

The ITM support is available in Cortex-M3/M4/M7 microcontrollers.

- The protocol used on the SWO pin to exchange data with the debugger probe is called ***Serial Wire Viewer (SWV).***
- Compared to other "debugging-alike" peripherals like UART/VCOM redirection or the ARM Semihosting, Serial Wire Viewer is really fast.
- Its communication speed is proportional to the MCU speed.
- To properly decode the bytes sent over the SWO port, the host debugger needs to know the frequencies of the CPU and SWO port.
- SWV protocol defines 32 different stimulus ports: a port is a "tag" on the SWV message used to enable/disable messages selectively.
- These channels allow for separating the diagnostic data into different categories. For instance, ARM recommends channel 0 for text data (e.g., from printf) and channel 31 for RTOS events, while the other channels can be used for any other purposes.

SWV-Supported Debugger

- Any original ST's board has an integrated ST-LINK/V2 debugger which supports SWO to trace ITM outputs on Cortex-M3+.
- That debugger can be used to program and debug an external MCU on other board, or turn into an J-Link debugger.
- Many ST-LINK clones do not have SWO pin exposed.
- When open the clone board, the STM32F103 chip is found, which is the same as the chip used in the original ST-LINK. So, the problem of missing SWO can be solved by exporting the SWO pin.

* ITM Functions

- The ITM stimulus registers are standardized by ARM and found on address 0xE0000000 (port 0) through 0xE000007C (port 31).
- To write data, enable ITM tracing and write data to the corresponding register.
- The CMSIS-Core package for Cortex-M3/M4/M7 cores provides necessary glue to handle SWV protocol.
- For example, the *ITM_SendChar()* routines allows to send a character using the SWO pin.

core_cm4.h

```
__STATIC_INLINE int32_t ITM_ReceiveChar (void) {
  int32_t ch = -1; /* no character available */
  if (ITM_RxBuffer != ITM_RXBUFFER_EMPTY) {
    ch = ITM_RxBuffer;
    ITM_RxBuffer = ITM_RXBUFFER_EMPTY; /* ready for next character */
  }
  return (ch);
}

__STATIC_INLINE uint32_t ITM_SendChar (uint32_t ch) {
  if (((ITM->TCR & ITM_TCR_ITMENA_Msk) != 0UL) && /* ITM enabled */
      ((ITM->TER & 1UL ) != 0UL)) { /* ITM Port #0 enabled */
    while (ITM->PORT[0U].u32 == 0UL) { __NOP(); }
    ITM->PORT[0U].u8 = (uint8_t)ch;
  }
  return (ch);
}
```

* Statistical Profiling

- When enable PC Sampling, the IDE can show the amount of execution time spent within various functions. This is useful when optimizing code.
- When pause the execution, the SWV Statistical Profiling view will display a table with calculated information. Clear the collected data to start a new profiling session.

Console | Problems | Executables | SWV Statistical Profiling

Function	% in use	Samples	Start address	Size
HAL_GetTick()	55.87%	31468	0x8000b15	0x18
HAL_Delay()	44.06%	24815	0x8000b2d	0x48
SysTick_Handler()	0.03%	18	0x80008f1	0xc
HAL_IncTick()	0.03%	16	0x8000aed	0x28
ITM_SendChar()	0.00%	2	0x8000579	0x4e
_write()	0.00%	1	0x800063b	0x38

Overflow packets: 153 PC Samples: 56320

* Exception Trace

- Every event sent when CPU handles an exception will be recorded and these data is used to calculate some information about exceptions.
- There are two tabs, but the useful information is in the statistic tab.

Console | Problems | Executables | SWV Exception Trace Log

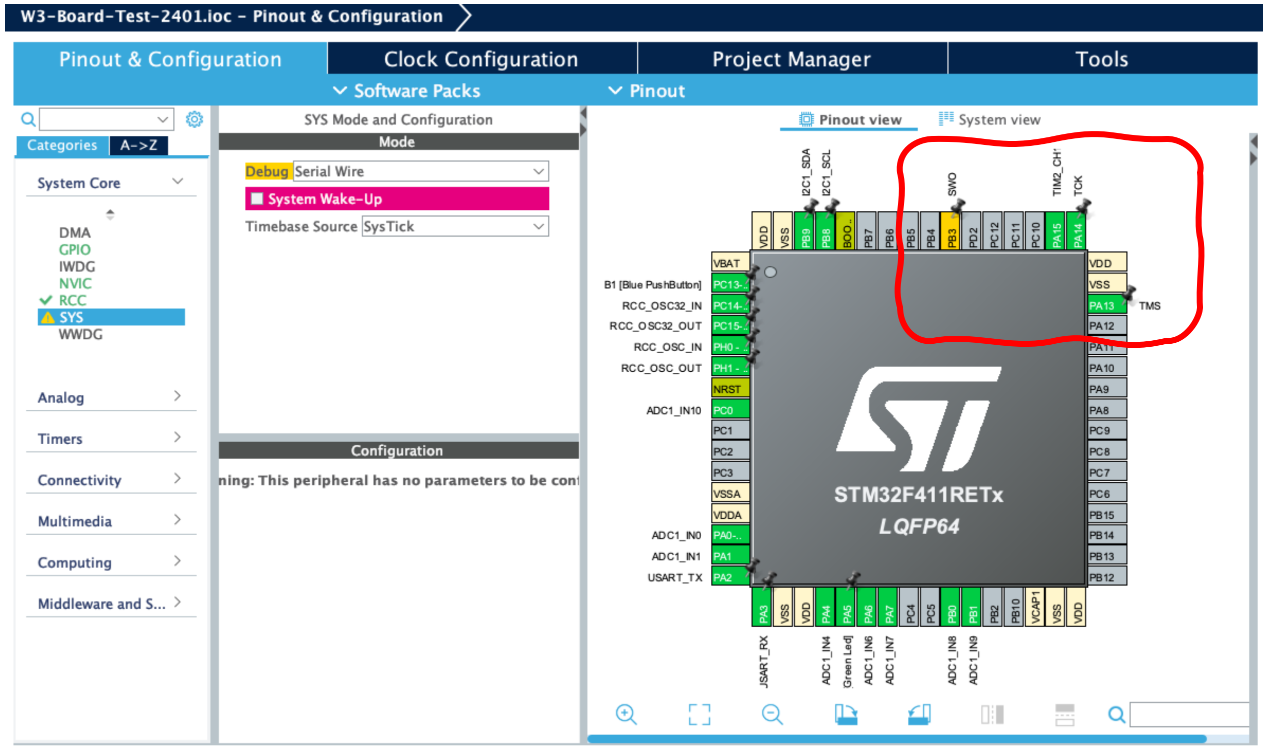
Data | Statistics

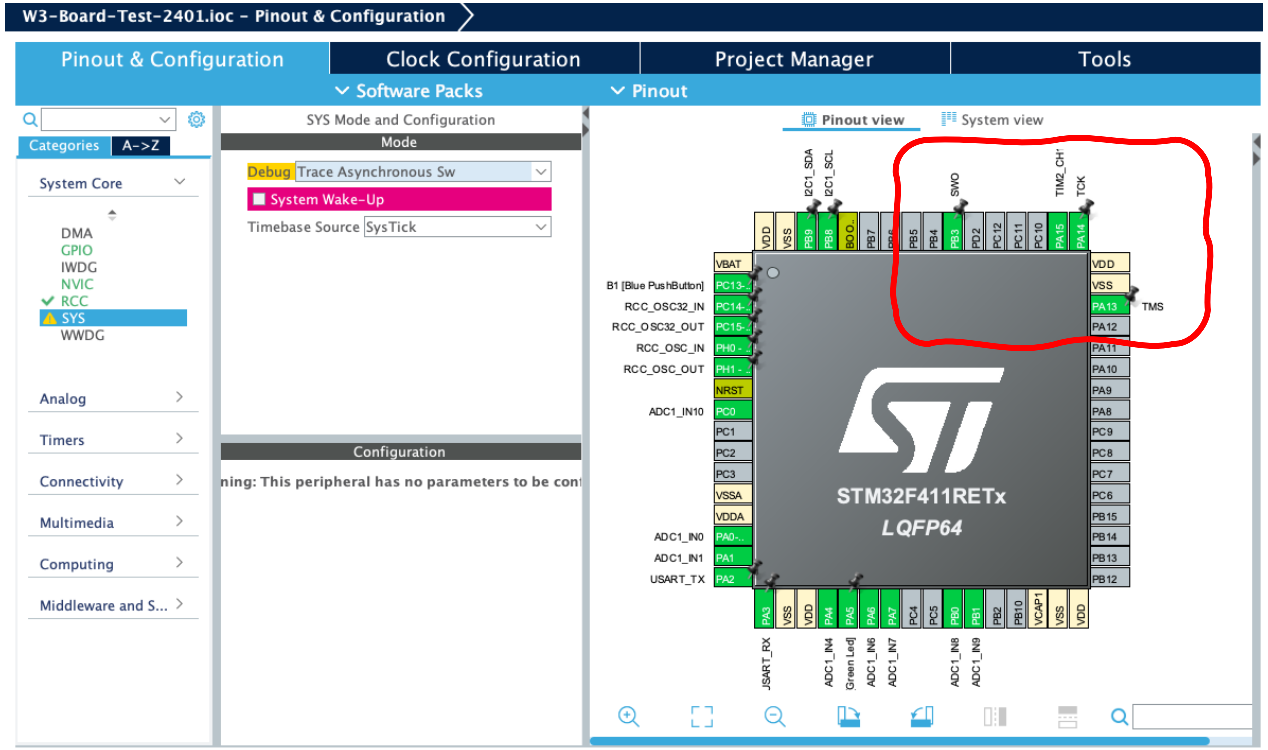
Exception	Handler	% of	Number of	% of exception ti...	% of debug time	Total runtime	Avg runtime	Fastest
SYSTICK (EXC 15)	SysTick_Handler()	100.0000%	12830	100.0000%	0.1198%	1534350	119	34
Total for all			12830		0.1198%	1534350	119	

Overflow packets: 82

❑ ST-LINK의 SWO 핀을 이용한 printf 함수 사용

Step 1: Debug : Trace Asynchronous Sw





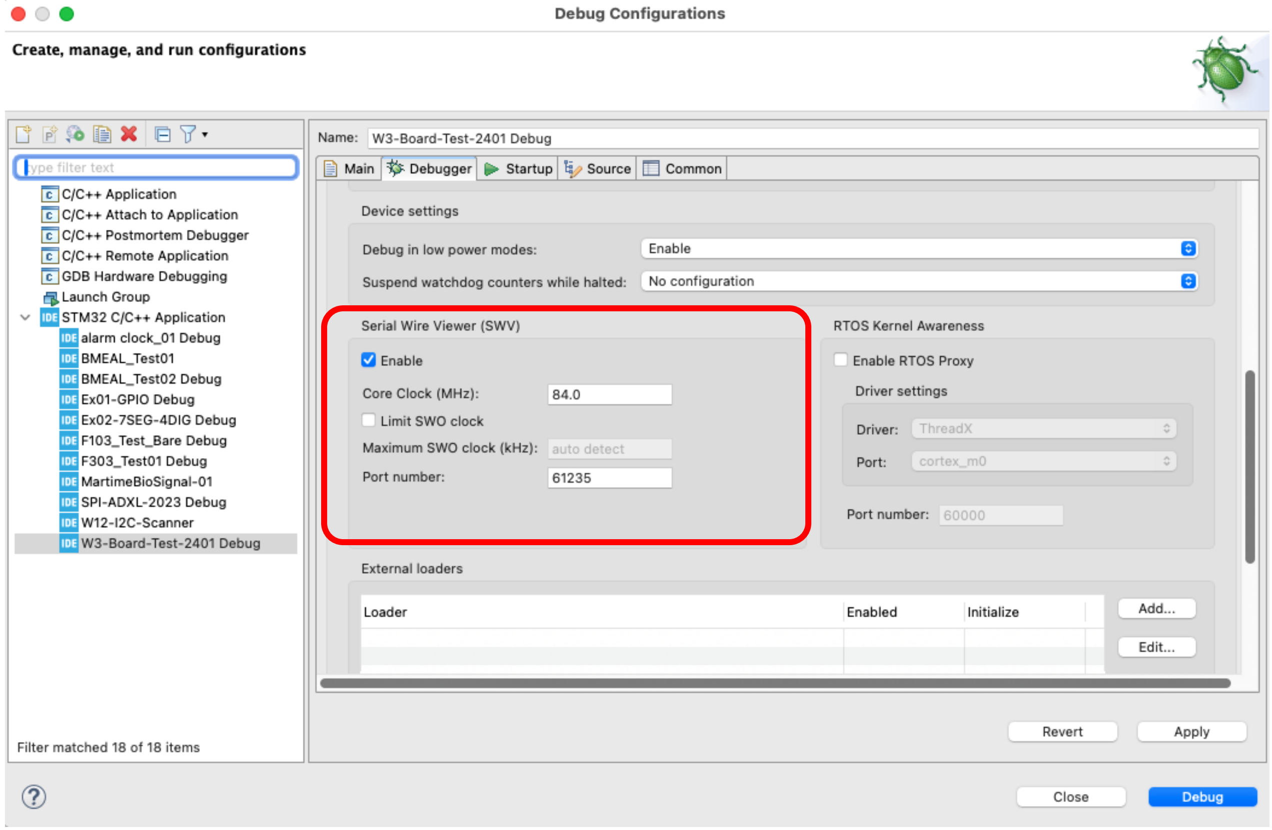
Step 2: Project → Generate Code.......

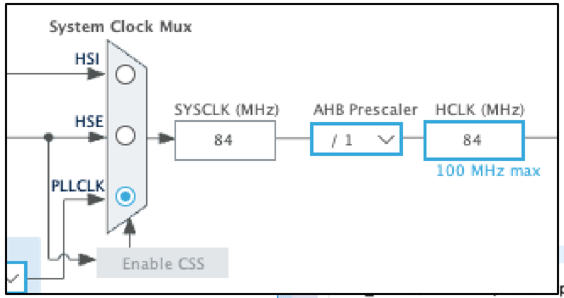
Start a new project through an empty project.

- Take note of the *System Clock Frequency*, such as 72 MHz, as we need to use it later
- Debug mode is set to *Trace Asynchronous SW* at reset, no need to configure this interface

Step 3: Run → Debug Configuration → STM32 Cortex-M C/C++ Application → Debugger

Step 4: Serial Wire Viewer (SWV) enable. 사용 중인 mcu의 Core Clock 으로 자동 변경됨 (그렇지 않은 경우, 사용자가 입력해야 함).






```
{
  (void)file;
  int DataIdx;

  for (DataIdx = 0; DataIdx < len; DataIdx++)
  {
     ITM_SendChar(*ptr++);
```

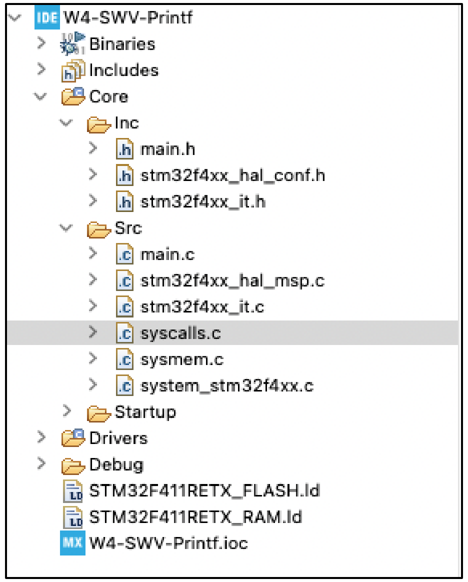
Step 5: #include <stdio.h> 추가

```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdio.h>

/* USER CODE END Includes */
```

Step 6:main.c 에 _write 함수 작성(syscall.c 의 코드를 복사하여 다음과 같이 수정)

```
/* USER CODE BEGIN 4 */

int _write(int file, char *ptr, int len)
{
   (void)file;
   int DataIdx;

   for (DataIdx = 0; DataIdx < len; DataIdx++)
   {
//      __io_putchar(*ptr++);
        ITM_SendChar(*ptr++);
   }
   return len;
}

/* USER CODE END 4 */
```

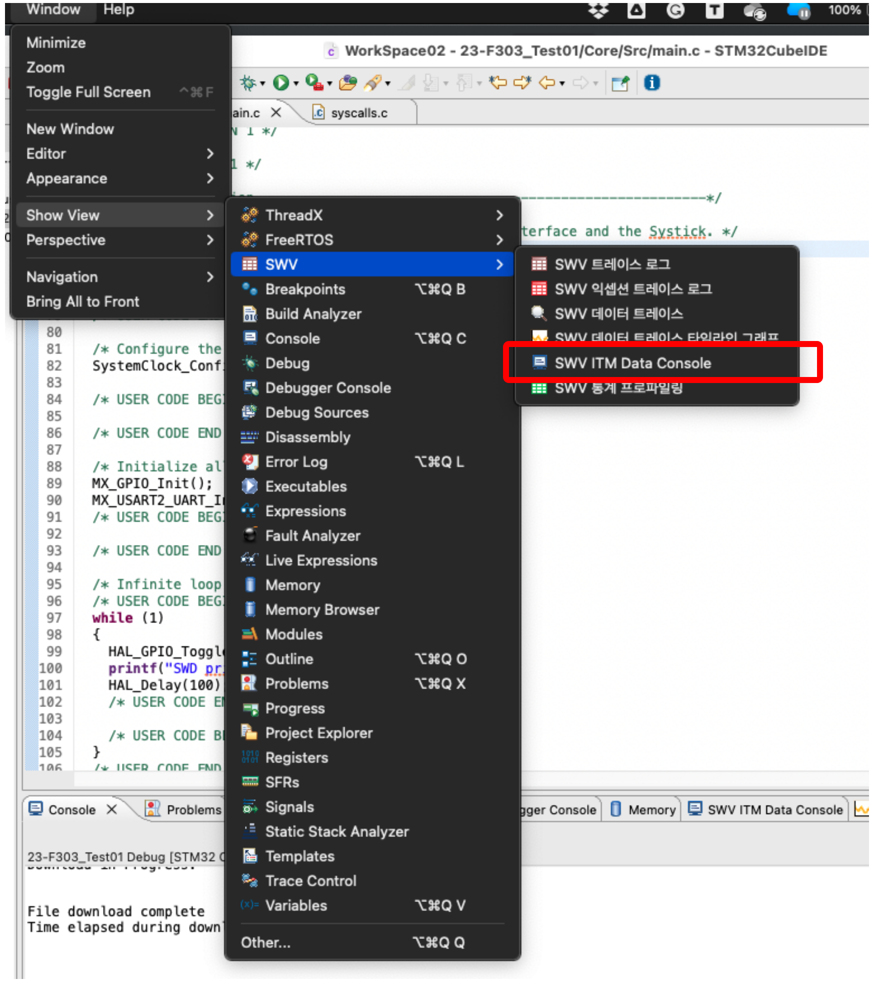
- syscalls.c -

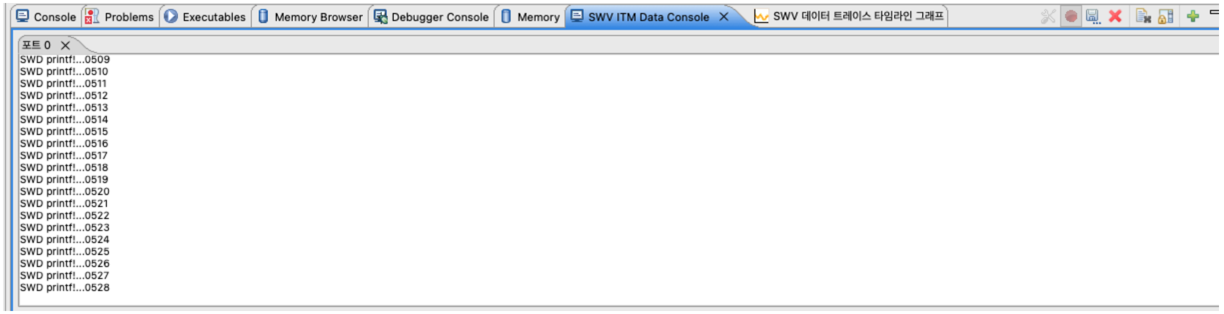


Step 7: while 문에 printf 함수 넣음

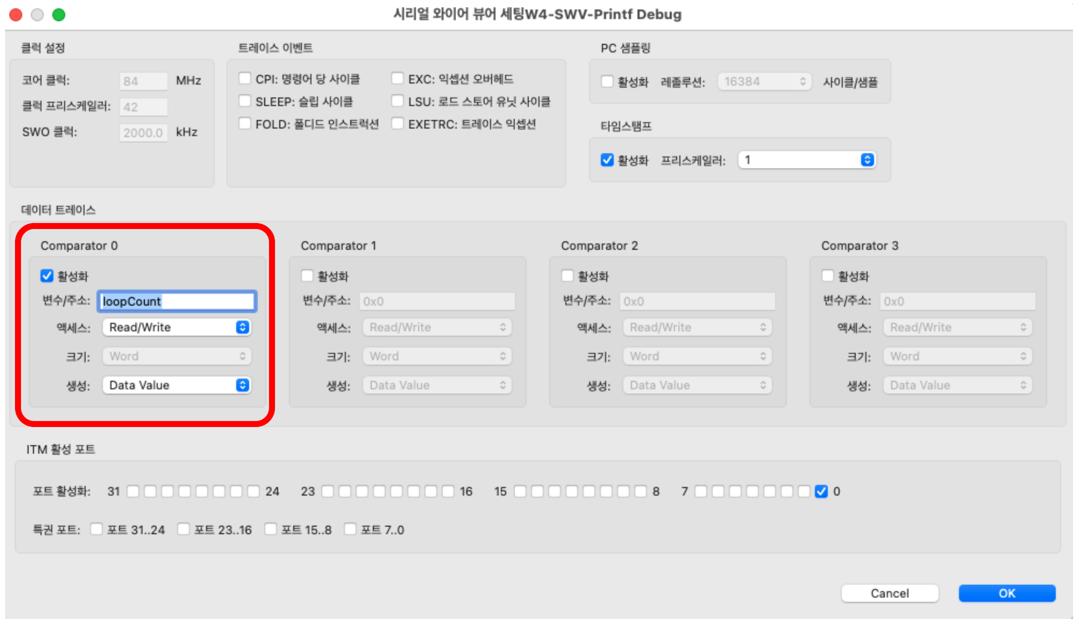
```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
    printf("SWD printf!...\n");
    HAL_Delay(100);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
```

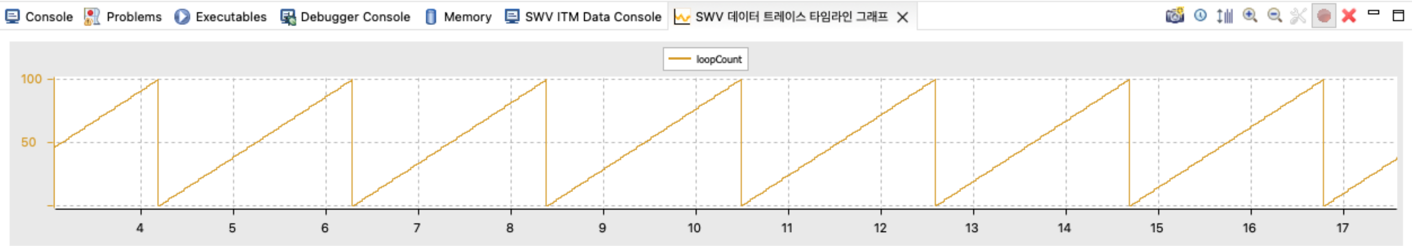
Step 8: Debug(F11) -> SWV ITM Data Console 창 띄움





❑ SWV+ITM을 이용한 데이터 Trace Timeline Graph





Screen capture로 저장한 결과

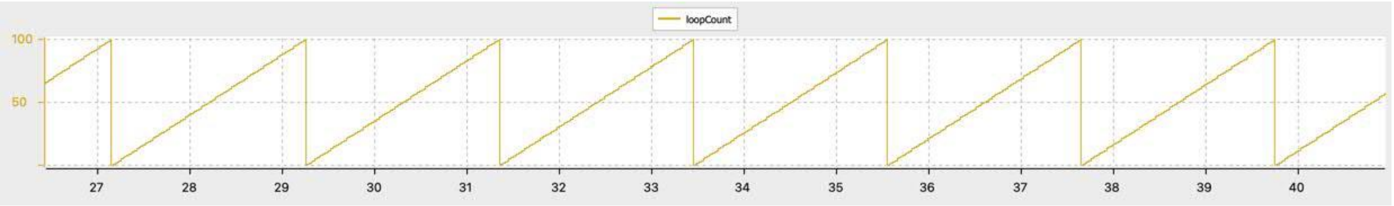
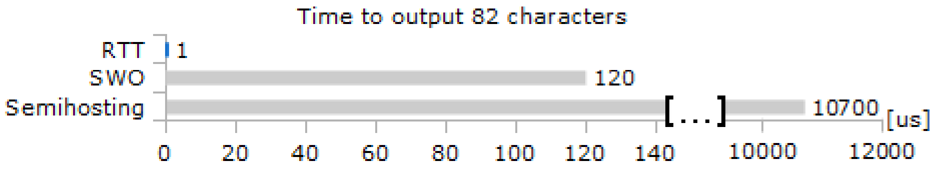


image file로 저장한 결과

Inspect variables

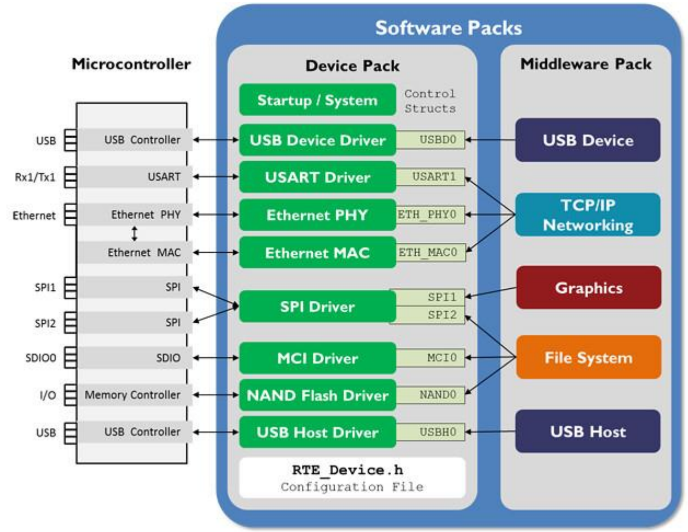
- The Debug IDE can inspect a variable in real-time using the Live Expression feature.
- SWV also has a useful graph mode to monitor variables. Open the SWV Data Trace Timeline Graph and open its configuration to enable Comparator 0 to trace a variable.



RTT Performance in comparison with Semihosting and SWO

❑ *STM32 CMSIS DSP Software Library*

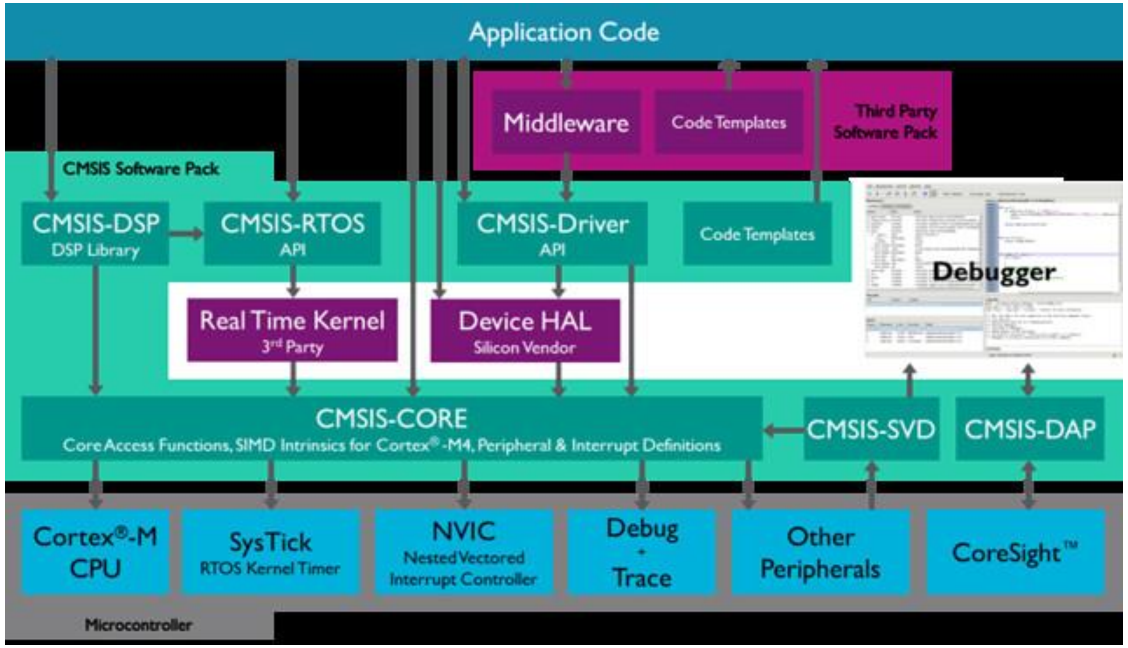
- CMSIS란 Cortex Microcontroller Software Interface Standard의 약자로, 소프트웨어 제품들 간 호환성을 고도화 시키고 소프트웨어 이식성을 증대 시키기 위해 ARM 사에서 개발한 소프트웨어 프레임워크



- **CMSIS-CORE** : Cortex-M Processor core와 주변 장치(peripheral)들을 위한 API
- **CMSIS-Driver** : 미들웨어를 위한 general peripheral driver interface 들을 정의

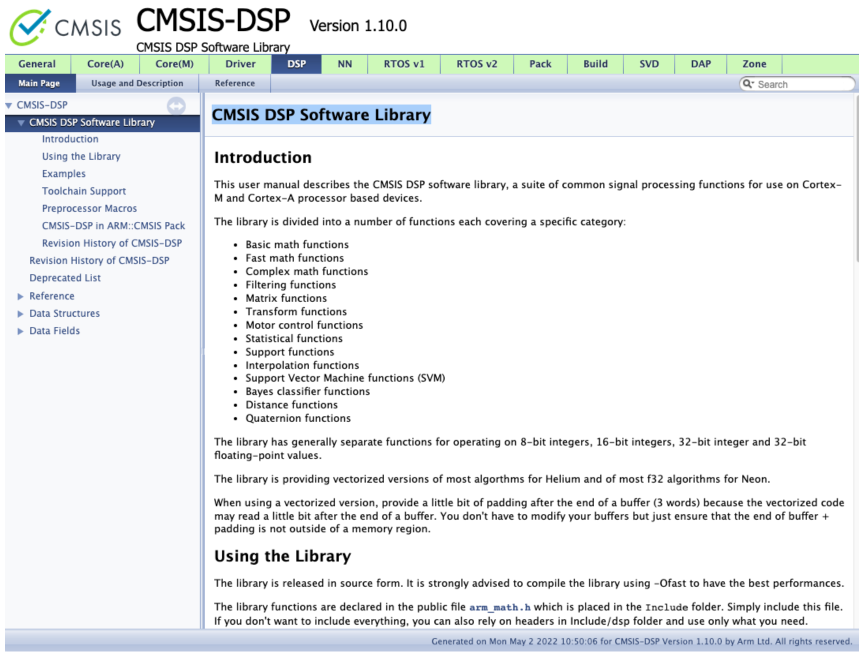
  USART: Universal Synchronous and Asynchronous Receiver/Transmitter interface driver.
  SPI: Serial Peripheral Interface Bus driver.
  I2C: Multi-master Serial Single-Ended Bus interface driver.
  Flash: Flash Memory interface driver.
  NAND: NAND Flash Memory interface driver.
  MCI: Memory Card Interface for SD/MMC memory.
  Ethernet: Interface to Ethernet MAC and PHY peripheral.
  USB: Interface driver for USB Host and USB Device communication.

- **CMSIS-DSP** : DSP Library Collection
  - 고정 소수점(fix-point)와 단정밀도 부동 소수점(single precision floating-point) data type을 사용하기 위한 라이브러리
- **CMSIS-RTOS API** : Real-Time Operating System을 위한 Common API.
- **CMSIS-Pack** : user와 device와 관련된 정보들을 기술한 XML 기반의 package description(PDSC) file.
  - source, header, library file, documentation, Flash programming algorithms, source code templates, example project 등을 포함
- **CMSIS-SVD** : 주변 장치들을 위한 System View Description
  - Device의 주변 장치들이 XML file로 기술 되어 있으며, Debugger에서 주변 장치들을 인식하기 위해 사용.
- **CMSIS-DAP** : Debug Access Port.



❑ *STM32 CMSIS DSP Software Library*

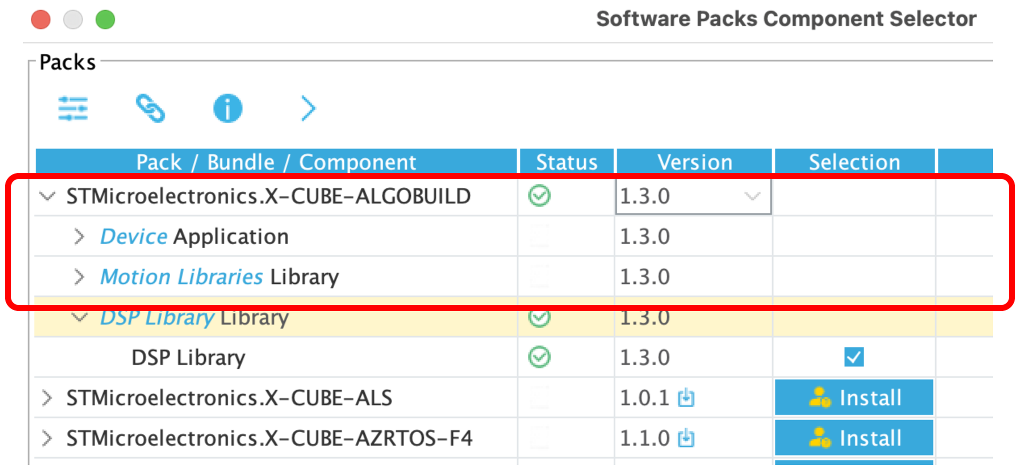
- CMSIS-DSP is an optimized compute library for embedded systems (DSP is in the name for legacy reasons).
- It provides optimized compute kernels for Cortex-M and for Cortex-A.
- Different variants are available according to the core and most of the functions are using a vectorized version when the Helium or Neon extension is available.



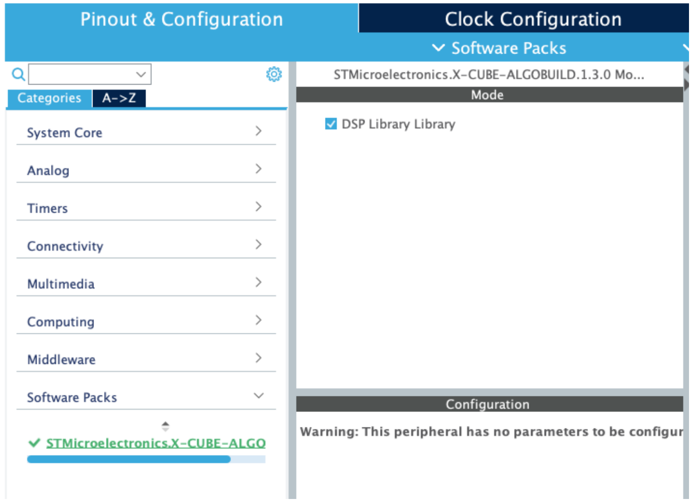
CMSIS-DSP Kernels

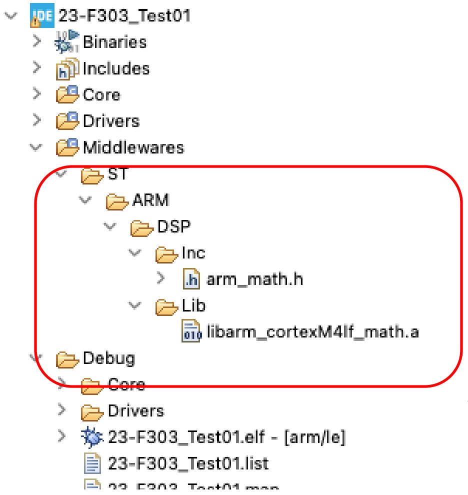
- Kernels provided by CMSIS-DSP (list not exhaustive):
- Basic mathematics (real, complex, quaternion, linear algebra, fast math functions)
- DSP (filtering)
- Transforms (FFT, MFCC, DCT)
- Statistics
- Classical ML (Support Vector Machine, Distance functions for clustering …)
- Kernels are provided with several datatypes : f64, f32, f16, q31, q15, q7.

Step 1: Select CMSIS-DSP Library (github source? o ST Software Packs?)



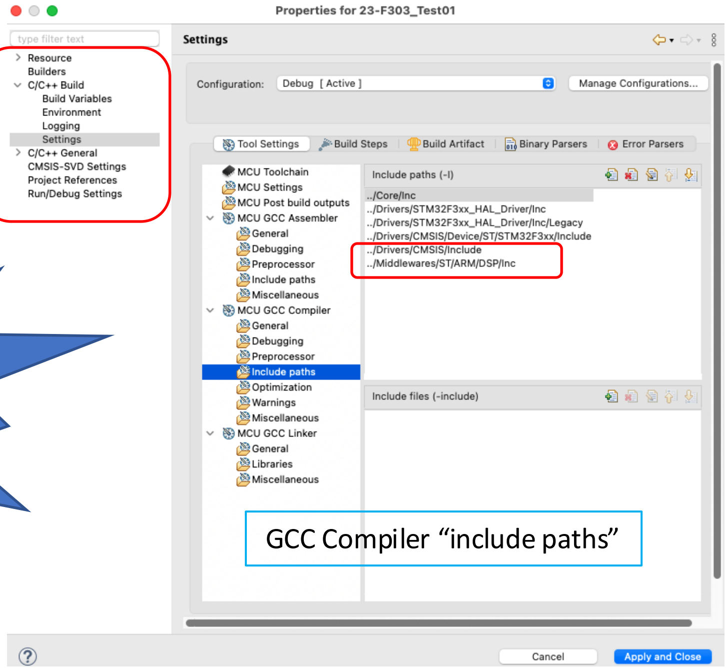
Step 2: Generate Code and Check installed library.





3-2

Build해서 오류가
안나면 일단
성공!...



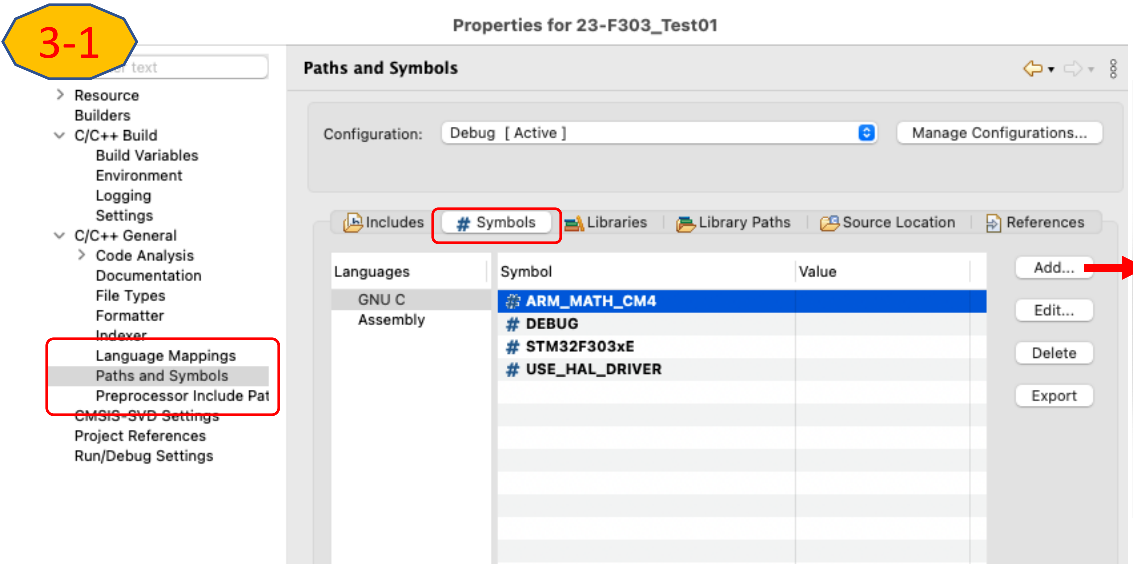
GCC Compiler "include paths"

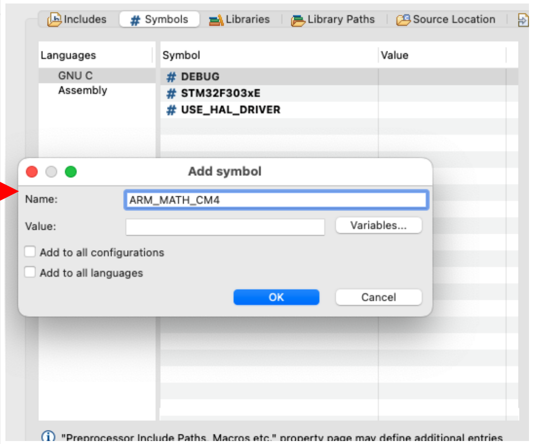
필요한 DSP기능이 설정 되었는 지와 "Middleware"라는 폴더에 화일들이 추가 되었는지 확인한다.

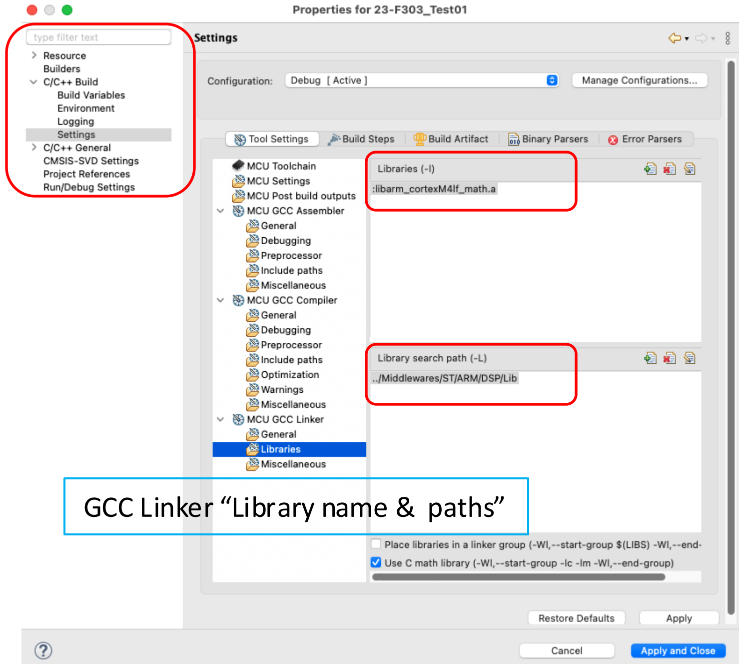
Step 3: 추가적인 환경(PATH, Compile flag 등을 수정한다.

3-3

3-1







GCC Linker "Library name & paths"

필요한 DSP기능이 설정 되었는 지와 "Middleware"라는 폴더에 화일들이 추가 되었는지 확인한다.

* CMSIS-DSP : SineCosine Example

SineCosine Example

Examples

Description:

Demonstrates the Pythagorean trignometric identity with the use of Cosine, Sine, Vector Multiplication, and Vector Addition functions.

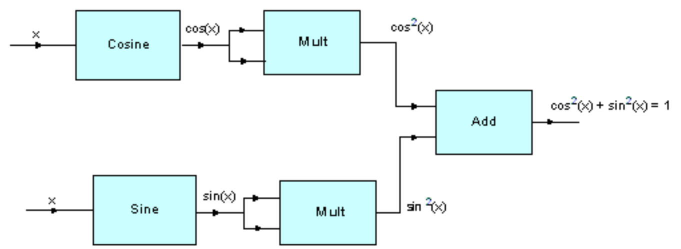
Algorithm:

Mathematically, the Pythagorean trignometric identity is defined by the following equation:

`sin(x) * sin(x) + cos(x) * cos(x) = 1`

where `x` is the angle in radians.

Block Diagram:



Variables Description:

- `testInput_f32` array of input angle in radians
- `testOutput` stores sum of the squares of sine and cosine values of input angle

CMSIS DSP Software Library Functions Used:

- arm_cos_f32()
- arm_sin_f32()
- arm_mult_f32()
- arm_add_f32()

Refer arm_sin_cos_example_f32.c

```c
#include <math.h>
#include "arm_math.h"
#if defined(SEMIHOSTING)
#include <stdio.h>
#endif
/* ----------------------------------------------------------------------
* Defines each of the tests performed
* ------------------------------------------------------------------- */
#define MAX_BLOCKSIZE 32
#define DELTA (0.0001f)
/* ----------------------------------------------------------------------
* Test input data for Floating point sin_cos example for 32-blockSize
* Generated by the MATLAB randn() function
* ------------------------------------------------------------------- */
const float32_t testInput_f32[MAX_BLOCKSIZE] =
{
-1.244916875853235400, -4.793533929171324800, 0.360705030233248850, 0.827929644170887320, -3.299532218312426900,
3.427441903227623800, 3.422401784294607700, -0.108308165334010680,
0.941943896490312180, 0.502609575000365850, -0.537345278736373500, 2.088817392965764500, -1.693168684143455700,
6.283185307179590700, -0.392545884746175080, 0.327893095115825040,
3.070147440456292300, 0.170611405884662230, -0.275275082396073010, -2.395492805446796300, 0.847311163536506600,
-3.845517018083148800, 2.055818378415868300, 4.672594161978930800,
-1.990923030266425800, 2.469305197656249500, 3.609002606064021000, -4.586736582331667500, -4.147080139136136300,
1.643756718868359500, -1.150866392366494800, 1.985805026477433800
};
const float32_t testRefOutput_f32 = 1.000000000;
/* ----------------------------------------------------------------------
* Declare Global variables
* ------------------------------------------------------------------- */
uint32_t blockSize = 32;
float32_t testOutput;
float32_t cosOutput;
float32_t sinOutput;
float32_t cosSquareOutput;
float32_t sinSquareOutput;

arm_status status;
```

```c
float32_t diff;
uint32_t i;
  for(i=0; i< blockSize; i++){
    cosOutput = arm_cos_f32(testInput_f32[i]);
    sinOutput = arm_sin_f32(testInput_f32[i]);
    arm_mult_f32(&cosOutput, &cosOutput, &cosSquareOutput, 1);
    arm_mult_f32(&sinOutput, &sinOutput, &sinSquareOutput, 1);
    arm_add_f32(&cosSquareOutput, &sinSquareOutput, &testOutput, 1);
    /* absolute value of difference between ref and test */
    diff = fabsf(testRefOutput_f32 - testOutput);
    /* Comparison of sin_cos value with reference */
    status = (diff > DELTA) ? ARM_MATH_TEST_FAILURE : ARM_MATH_SUCCESS;
    if ( status == ARM_MATH_TEST_FAILURE){
      printf("CMSIS-DSP Comparison Failure!...\n");
    }
```

https://arm-software.github.io/CMSIS_5/DSP/html/group__SinCosExample.html
https://arm-software.github.io/CMSIS_5/DSP/html/arm_sin_cos_example_f32_8c-example.html

*다음의 예제 코드를 stm32cube project에 적정한 위치에 복사해 넣고, 다양한 변수들의 변화를 debugging해 보자!.

*참고: UART기반 printf를 이용한 terminal debugging을 위한 코드.

```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
#ifdef __GNUC__
/* With GCC, small printf (option LD Linker->Libraries->Small printf
   set to 'Yes') calls __io_putchar() */
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */

/**
  * @brief  Retargets the C library printf function to the USART.
  * @param  None
  * @retval None
  */
PUTCHAR_PROTOTYPE
{
  /* Place your implementation of fputc here */
  /* e.g. write a character to the USART1 and Loop until the end of transmission */
  if (ch == '\n')
    HAL_UART_Transmit (&huart2, (uint8_t*) "\r", 1, 0xFFFF);
  HAL_UART_Transmit (&huart2, (uint8_t*) &ch, 1, 0xFFFF);

  return ch;
}
/* USER CODE END 0 */
```

*참고: SEO기반 printf를 이용한 terminal debugging을 위한 코드.

```
int _write(int file, char *ptr, int len) {
int DataIdx;
      for (DataIdx = 0; DataIdx < len; DataIdx++){
            ITM_SendChar( *ptr++ );
      }
      return len;
}
```