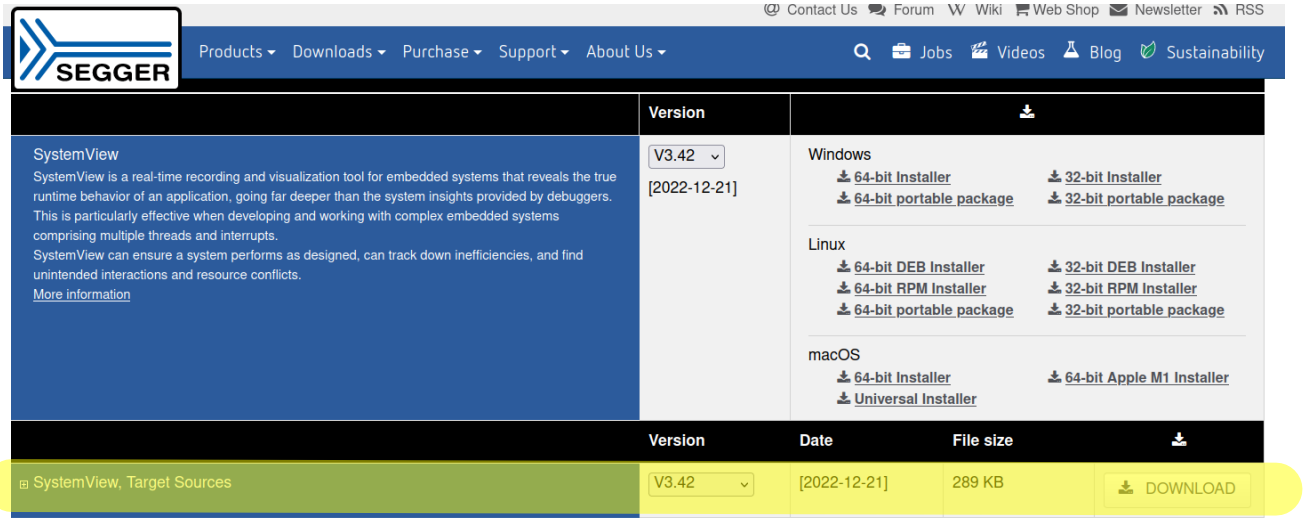
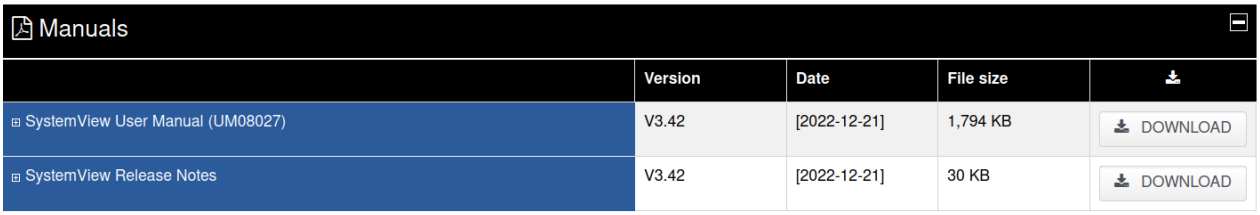
Download the segger system viewer resource files from the official website of segger.

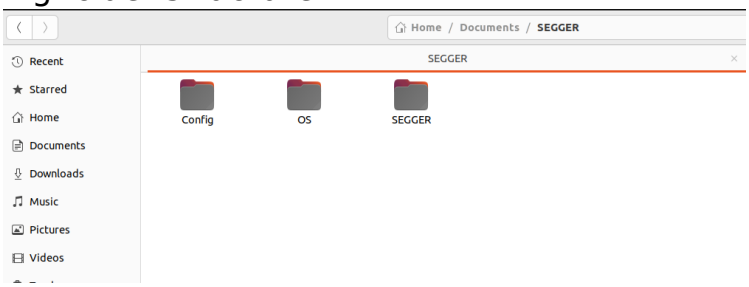
https://www.segger.com/downloads/systemview/





Under "ThirdPartySource" Create the Following folder structure

ThirdPartySource
--> SEGGER
---> Config
---> OS
---> SEGGER



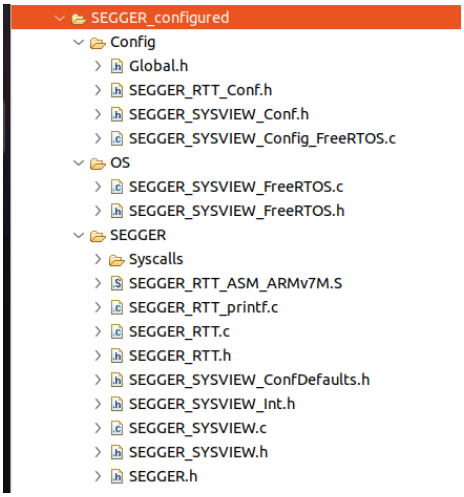
Specific Files are required to be copied into the folde from the downloaded source file folder from segger system viewer webpage.

All the shown files are copied from the donwloaded segger system viewer source file folder.

Once, copied to the workspace, we need to apply one patch to the files (changes).

Patch name is "FreeRTOSvsegger_cm4.patch"

It includes the information on the code changes needed in the system viewer source files and it should pe applied very carefully.



Once done, we need to do further configurations in the workspace which include. -Aman Kanwar

-> Environment configuration
-> Include path settings

1 -> Include the file "SEGGER_SYSVIEW_FreeRTOS.h" at the end of file "FreeRTOSConfig.h"

This defines the macros for creating traceviewer events

```
#define xPortPendSVHandler PendSV_Handler
#define xPortSysTickHandler SysTick_Handler

#include "SEGGER_SYSVIEW_FreeRTOS.h"

#endif /* FREERTOS_CONFIG_H */
```

Problems Tasks Console Properties

2 -> In "FreeRTOSConfig.h" include the following macro switches

#define INCLUDE_xTaskGetIdleTaskHandle
#define INCLUDE_pxTaskGetStackStart 1

```
#define INCLUDE_vTaskSuspend            1
#define INCLUDE_vTaskDelayUntil         1
#define INCLUDE_vTaskDelay              1

#define INCLUDE_xTaskGetIdleTaskHandle  1
#define INCLUDE_pxTaskGetStackStart     1
```

3 -> We need to perform some Microcontroller and project specific settings in the workspace which involves the following chages needed to be done.
-> Mentioning of the MCU core type in "SEGGER_SYSVIEW_ConfDefaults.h"

```
#endif

#ifndef   SEGGER_SYSVIEW_CORE
  #define SEGGER_SYSVIEW_CORE SEGGER_SYSVIEW_CORE_CM3
#endif
#endif
```

-> System viewer buffer size configuration in "SEGGER_SYSVIEW_ConfDefaults.h"

```
/*********************************************************************
*
*       Define: SEGGER_SYSVIEW_RTT_BUFFER_SIZE
*
*  Description
*    Number of bytes that SystemView uses for the RTT buffer.
*  Default
*    1024
*/
#ifndef   SEGGER_SYSVIEW_RTT_BUFFER_SIZE
  #define SEGGER_SYSVIEW_RTT_BUFFER_SIZE        (1024 * 4)
#endif

/*********************************************************************
```

-> Application specific information in the "SEGGER_SYSVIEW_Config_FreeRTOS.c"

```
*
**********************************************************************
*/
// The application name to be displayed in SystemViewer
#define SYSVIEW_APP_NAME        "My FreeRTOS Application"

// The target device name
#define SYSVIEW_DEVICE_NAME     "STM32F411VET Cortex-M4"

// Frequency of the timestamp. Must match SEGGER_SYSVIEW_GET_TIMESTAMP in SE
#define SYSVIEW_TIMESTAMP_FREQ  (configCPU_CLOCK_HZ)

```

Once done, we need to enable the time stamp information to be dumped by our RTOS application the same is needed for monitoring of the events at specific time frames. Which then shows what event was configured at what stage.

In order for our STM32 board to monitor the time stamp information, we need to enable the same at the hardware level using the "Cycle Counter" of ARM M4F
For more information refer to ARM website.

https://developer.arm.com/documentation/ka001406/latest

Summary

Can the Cortex-M3 or Cortex-M4 processor measure the cycle count of its own activity?

Answer

If the Data Watchpoint Trigger (DWT) is implemented, the Cortex-M3 or Cortex-M4 processor can measure elapsed cycles by reading the DWT_CYCCNT register at the start and at the end of the time interval of interest, and calculating the difference in their values.

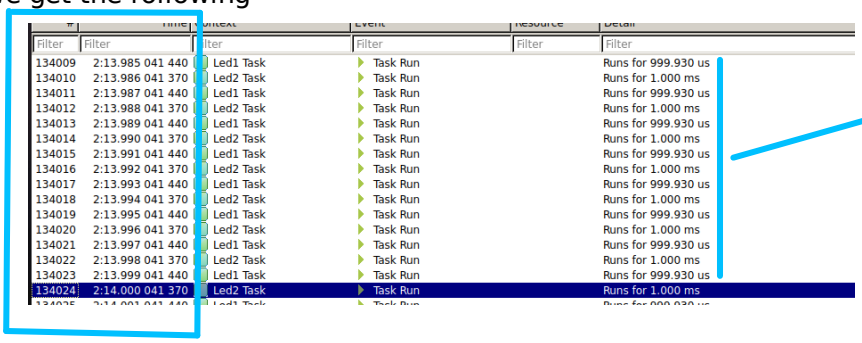
Alternatively, the processor can use its SysTick function to measure elapsed cycle counts.

In short, DWT_CYCCNT register of the ARM Cortex M3/4/4F processor will store the number of clock cycles since the processor was POR or started after reset.

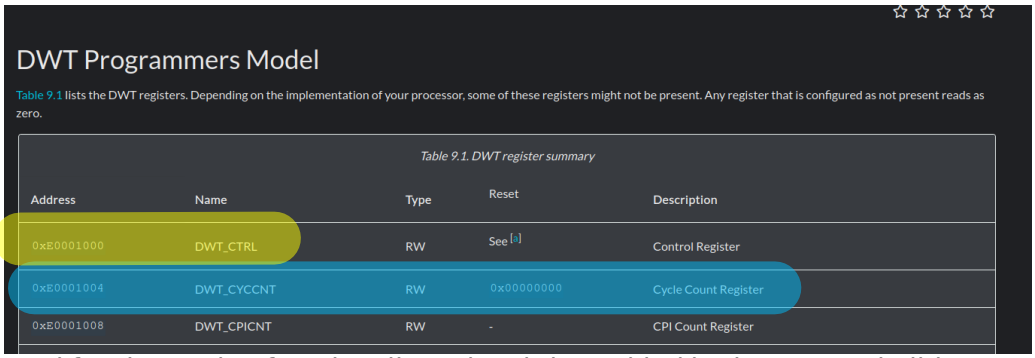
The same can be used to note the number of cycles that have passed for any given logic or instuction execution.

--> Read the value fo Cycle Count R1
--> Execute the instruction
--> Read the values of Cycle Count R2

Cycles consumed in the instruction execution = R2 - R1
using the time interval of each cycle, we can calculate the time needed for that instruction to execute :)
Using the same we get the following

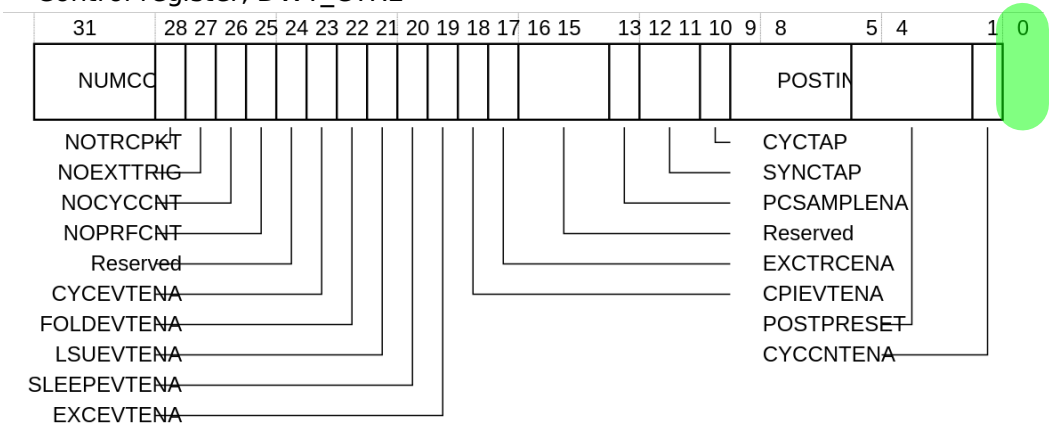


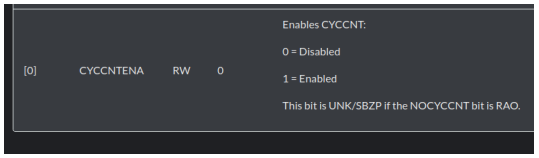
For ARM Cortex M3/4 based controllers, the given register is available at location 0xE0001000. This register addresses are managed by the ARM and the CMSIS is configuring the same.
https://developer.arm.com/documentation/ddi0439/b/Data-Watchpoint-and-Trace-Unit/DWT-Programmers-Model



These are mainly used for the tracing functionality, when it is enabled in the system/build.

Now, we can easlity set the required biy in register "DWT_CYCCNT" by performing a simple pointer operation
Control register, DWT_CTRL





Setting the bit 0 will enable the cycle count.

```
#define DWT_CYCCNT ((volatile uint32_t *) 0xE0001000)

*DWT_CYCCNT = *DWT_CYCCNT | (1 << 0);   or    *DWT_CYCCNT |= (1 << 0);
```

-Aman Kanwar

in main.c, the same can be configured as shown in the main function.

```
/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
/* USER CODE BEGIN 2 */
*DWT_CYCCNT |= (1 << 0);
```

After this, we need to initialize the SEGGER SYSVIEW and need to start the recordings for the same.

-> Single shot
-> Continuous recording over UART/JLINK

Now for single shot recording, we will add the following function calls in the main.c inside the main function.

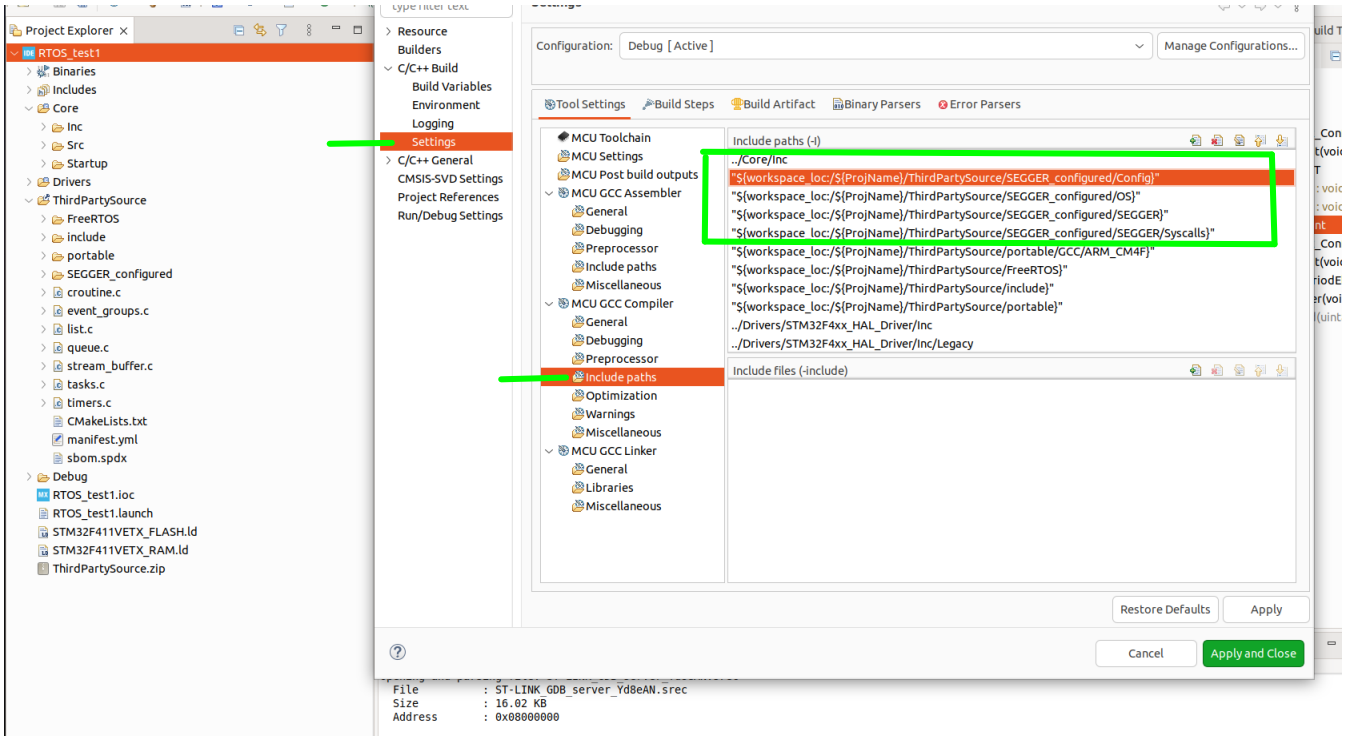
```
/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
/* USER CODE BEGIN 2 */
*DWT_CYCCNT |= (1 << 0);

SEGGER_SYSVIEW_Conf();
SEGGER_SYSVIEW_Start();
```

Now, we need to include the paths of the added SEGGER folder in ThirdPartySource at project level. So that the files and build system should be able to see the added source code and build it as a part of single image.

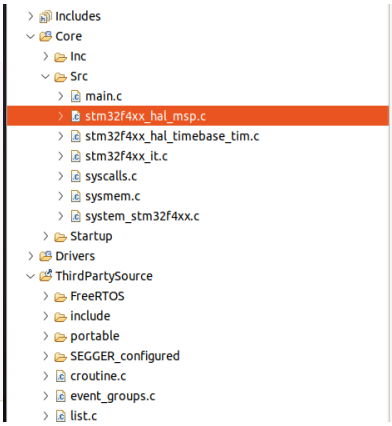


The include path settings needs to be done both at Project level and the directory level whereever the SEGGER SYSVIEW related APIs are being called out. For example in RTOS folder.

After building.

After flashing and debugging, if the execution is stuck at the macro shown below. Then we need to set the priority group value. The same gets set in the main when vTaskStartScheduler() is called in main.c. However, we're calling the SEGGER_SYSVIEW APIs few steps before. Hence, we need to configure the priority group values before these API calls. Hence, add the following function call "vInitPrioGroupValue()" in "stm32f4xx_hal_msp.c"

```
900             * https://www.FreeRTOS.org/RTOS-Cortex-M3-M4.html
901             * https://www.FreeRTOS.org/FAQHelp.html */
902            configASSERT( ucCurrentPriority >= ucMaxSysCallPriority );
903        }
904
905        /* Priority grouping:  The interrupt controller (NVIC) allows the bits
906         * that define each interrupt's priority to be split between bits that
907         * define the interrupt's pre-emption priority bits and bits that define
908         * the interrupt's sub-priority.  For simplicity all bits must be defined
909         * to be pre-emption priority bits.  The following assertion will fail if
910         * this is not the case (if some bits represent a sub-priority).
911         *
912         * If the application only uses CMSIS libraries for interrupt
913         * configuration then the correct setting can be achieved on all Cortex-M
914         * devices by calling NVIC_SetPriorityGrouping( 0 ); before starting the
915         * scheduler.  Note however that some vendor specific peripheral libraries
916         * assume a non-zero priority group setting, in which cases using a value
917         * of zero will result in unpredictable behaviour. */
918        configASSERT( ( portAIRCR_REG & portPRIORITY_GROUP_MASK ) <= ulMaxPRIGROUPValue );
919    }
920
921 #endif /* configASSERT_DEFINED */
922
```




```
54
55 /* USER CODE END ExternalFunctions */
56
57 /* USER CODE BEGIN 0 */
58
59 /* USER CODE END 0 */
60 /**
61   * Initializes the Global MSP.
62   */
63 void HAL_MspInit(void)
64 {
65   /* USER CODE BEGIN MspInit 0 */
66
67   /* USER CODE END MspInit 0 */
68
69   __HAL_RCC_SYSCFG_CLK_ENABLE();
70   __HAL_RCC_PWR_CLK_ENABLE();
71
72   /* System interrupt init*/
73
74   /* USER CODE BEGIN MspInit 1 */
75   vInitPrioGroupValue();
76   /* USER CODE END MspInit 1 */
77 }
78
79 /* USER CODE BEGIN 1 */
80
81 /* USER CODE END 1 */
```

Once done, we can create the RTOS tasks in main.c as shown

Task creation APIs

```
105   /* Initialize all configured peripherals */
106   MX_GPIO_Init();
107   /* USER CODE BEGIN 2 */
108   *DWT_CYCCNT |= (1 << 0);
109
110   SEGGER_SYSVIEW_Conf();
111   SEGGER_SYSVIEW_Start();
112
113   xTaskCreate(&Task2, "Task2", 200, NULL, 2, NULL);
114   xTaskCreate(&Task1, "Task1", 200, NULL, 2, NULL);
115
116   vTaskStartScheduler();
117   /* USER CODE END 2 */
118
119   /* Infinite loop */
120   /* USER CODE BEGIN WHILE */
121   while (1)
122   {
```

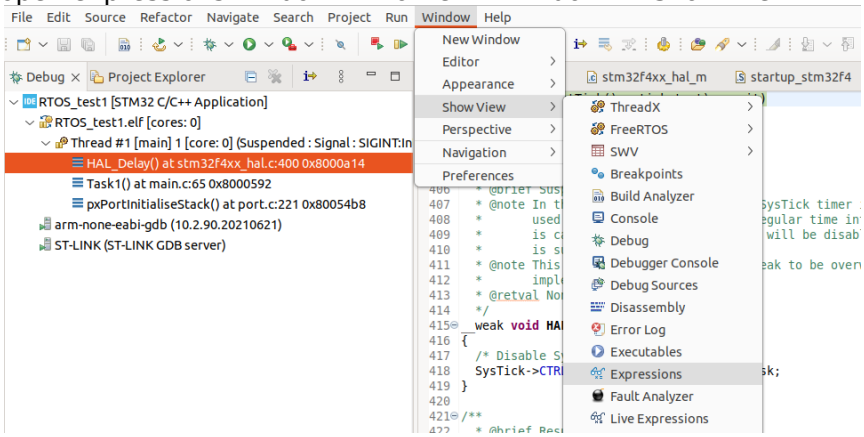
Task functions

```
58 #define DWT_CYCCNT ((volatile uint32_t *) 0xE0001000)
59
60 void Task1(void *ptr)
61 {
62     while(1)
63     {
64         HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_14);
65         HAL_Delay(1000);
66     }
67 }
68
69 void Task2(void *ptr)
70 {
71     while(1)
72     {
73         HAL_Delay(200);
74         HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12);
75     }
76 }
```

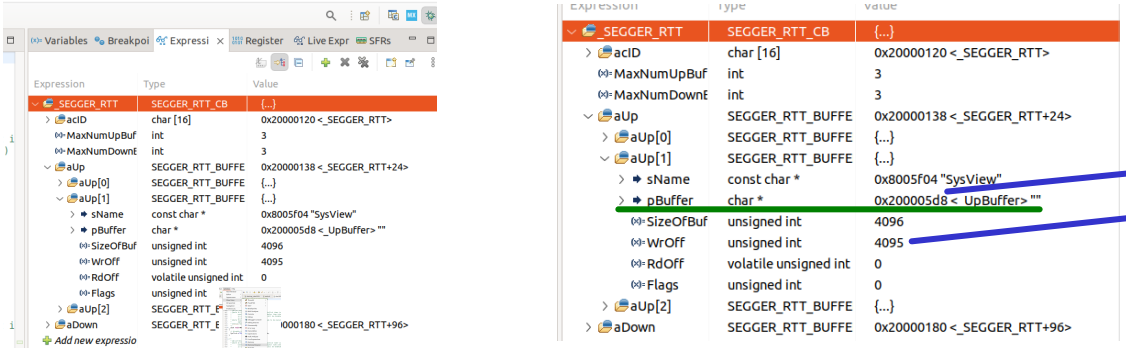
Getting single shot recording for Segger Studio

In CubeMx IDE. -> Start the debugging and Hit Run for a while and then pause the exection.

After that open expressions window in Home -> Window -> Show View -> Expressions

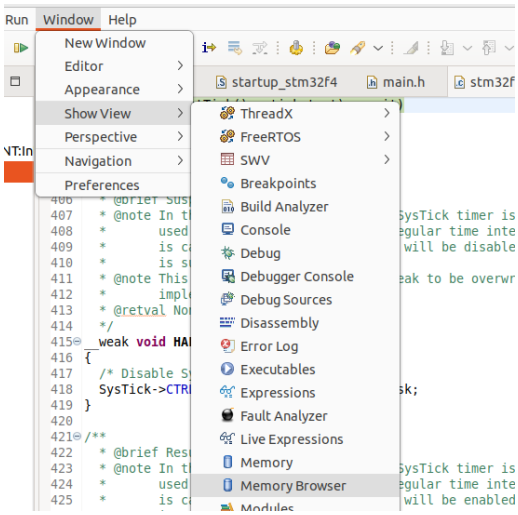


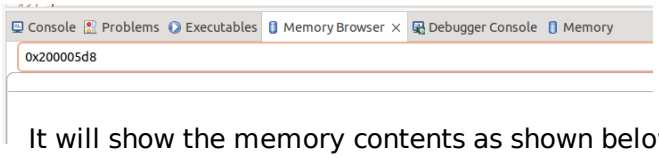
Click on Add new expression. Once we go through the Segger System View user guide section 3.13.2 it is mentioned that for single-shot recording the buffer RTT buffer is named as "_SEGGER_RTT"



-Aman Kanwar

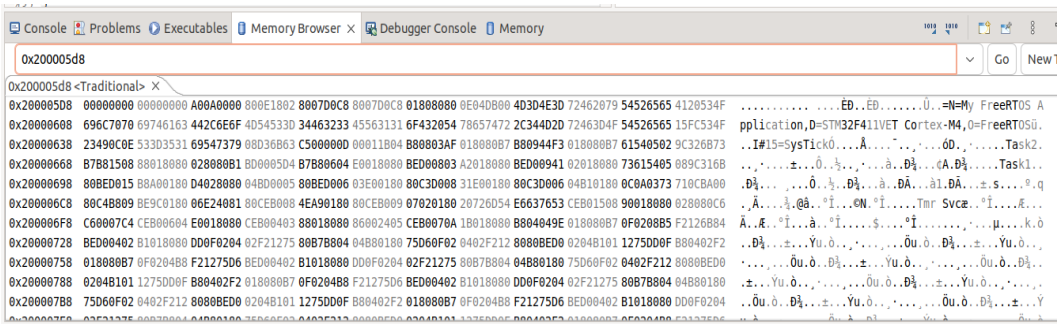
Now, copy the address of recorded buffer (pbuffer) in my case it is "0x200005d8" and open the memory browser window.



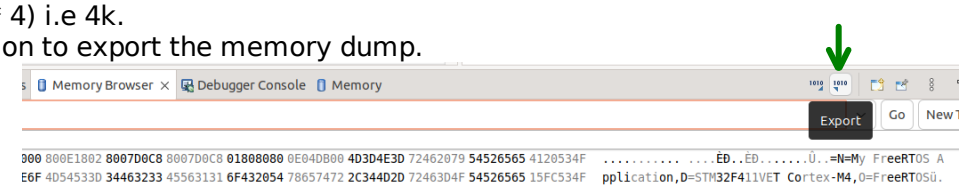


Paste and hit enter

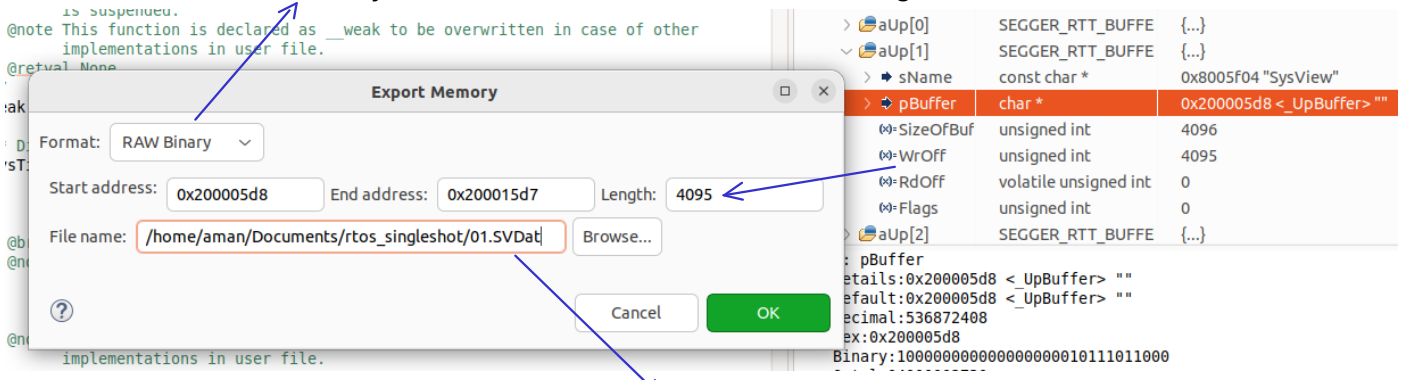
It will show the memory contents as shown below



Now we need to export the memory dump with a size equal to "WrOff" in my case it is 4095. We had set this buffer size before as (1024 * 4) i.e 4k.
Use the following option to export the memory dump.

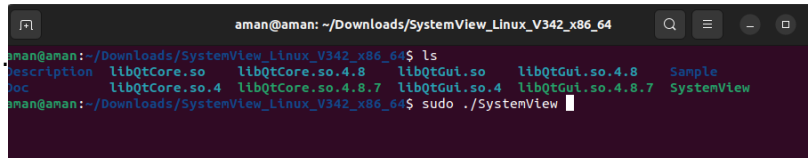


Make sure to select "RAW Binary" in Format section and Provide the length same as WrOff buffer

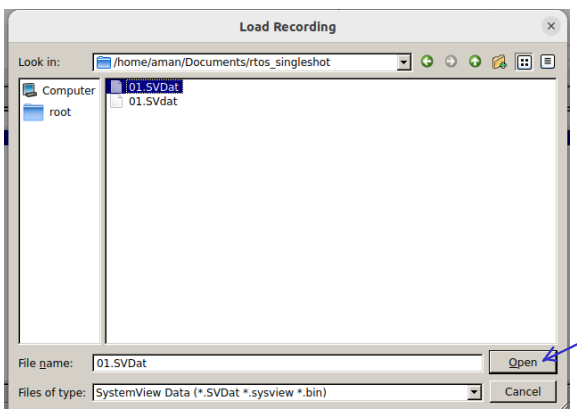


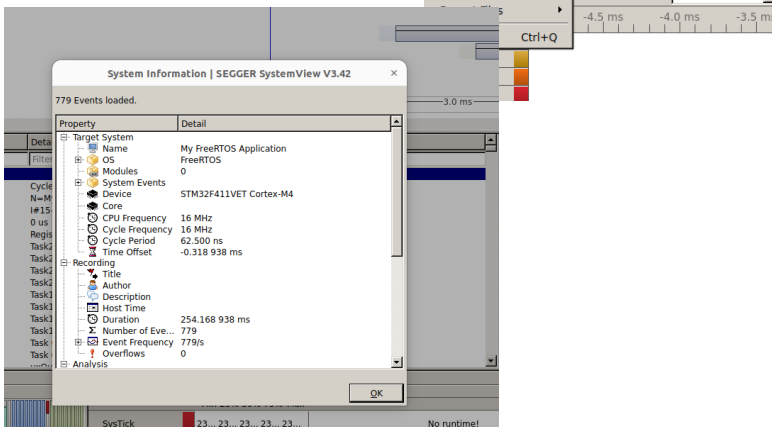
Make sure to save the file with extension <file>.SVDat and then Click on "OK"

Start "Segger System Viewer" via Terminal.

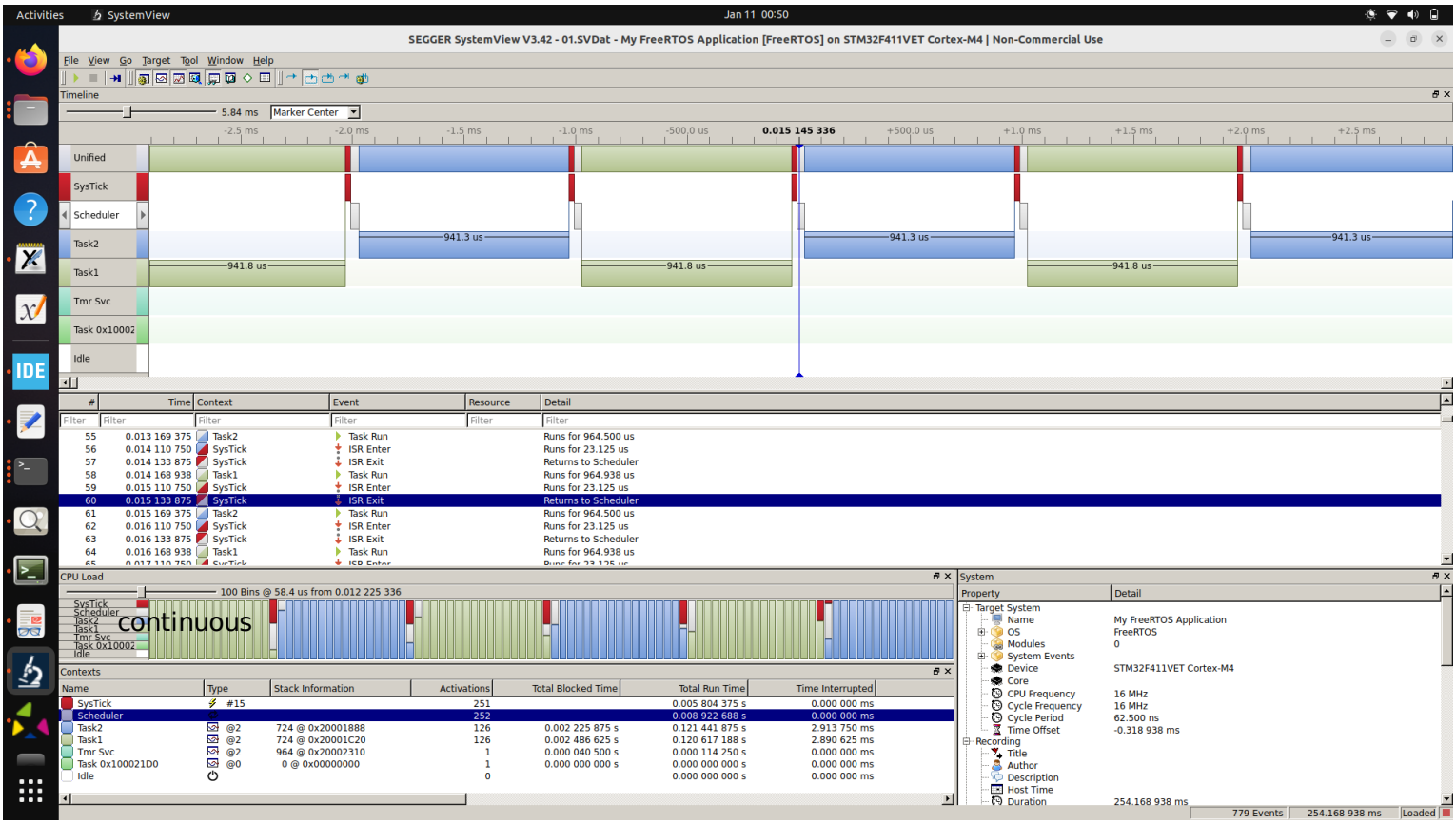


Once opened, open the single shot recoding data file in the system viewer software.

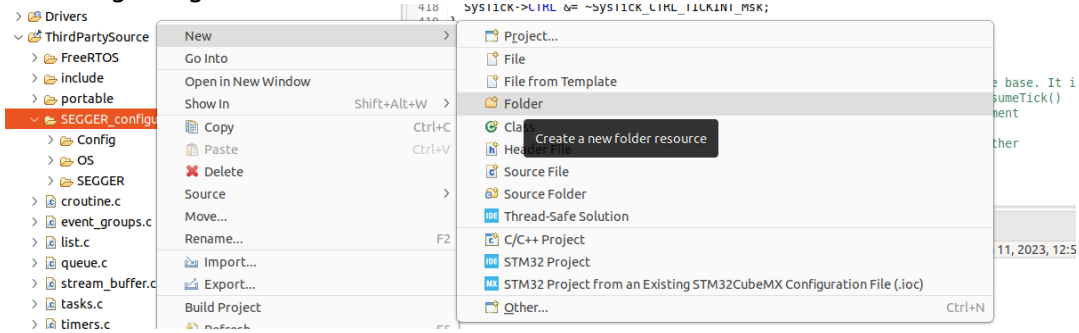




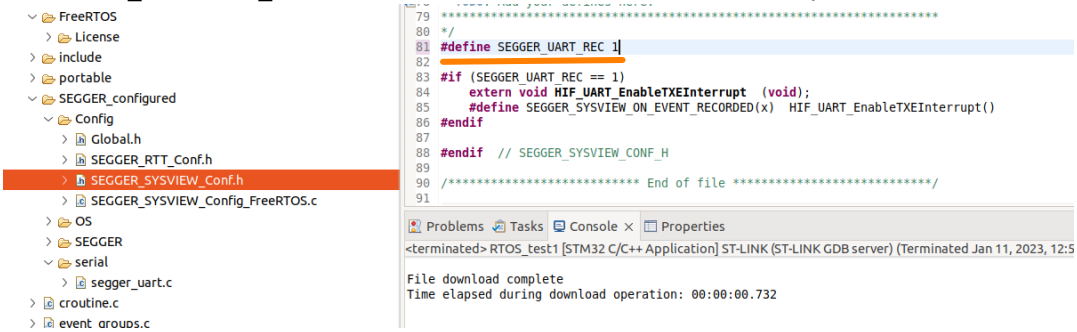
Following runtime for our application will be shown on the screen. -Aman Kanwar



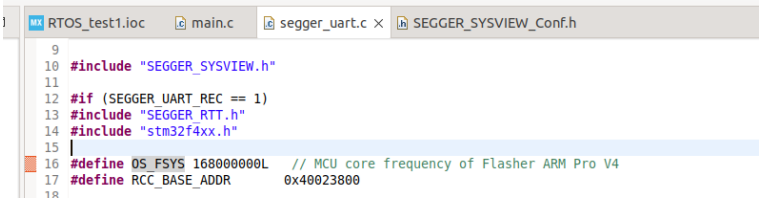
For continuous recording using UART. Create a new "serial" folder under SEGGER folder



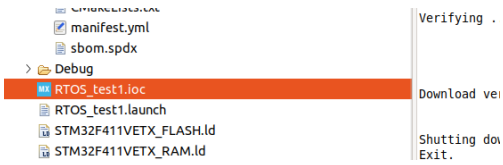
Once created, add the file "segger_uart.c" in the folder and set the "SEGGER_UART_REC" macro as 1 in file SEGGER_SYSVIEW_Conf.h. Also we need to include the build path for the same.

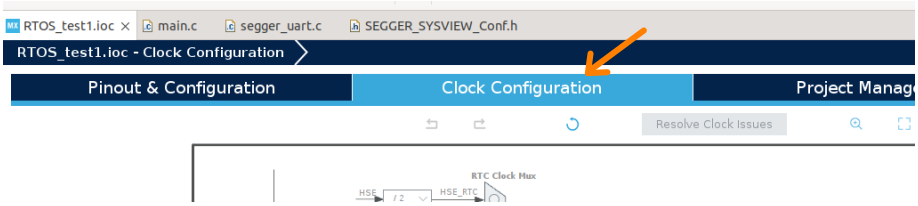


Important!!!
Now, we need to change a few configurations in the "segger_uart.c" file depending on the MCU we're using. For my STM32F411VET controller. Some NVIC and clock related settings we need to set in the CubeMX configuration tool.

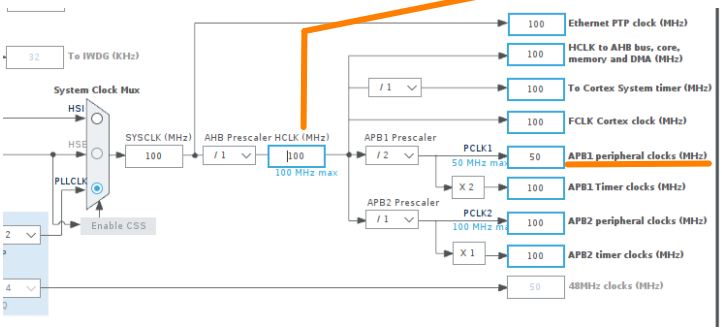


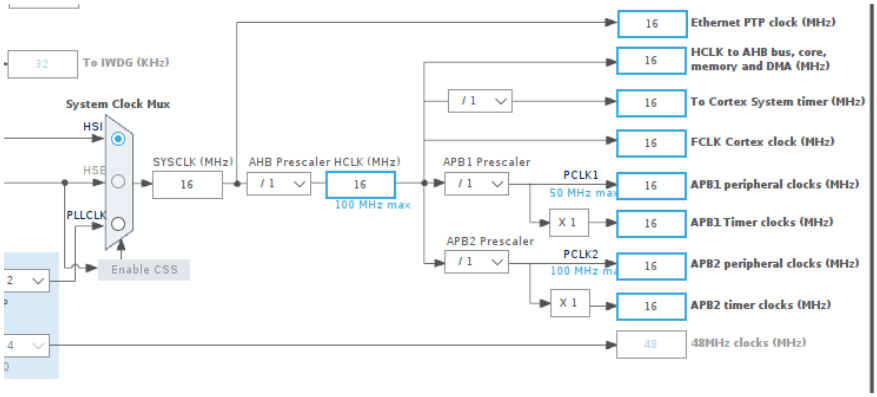
Double click on the CubeMx configuration file.
to open the configuration tool of CubeMx.
Once opened, open the clock configuration tab.





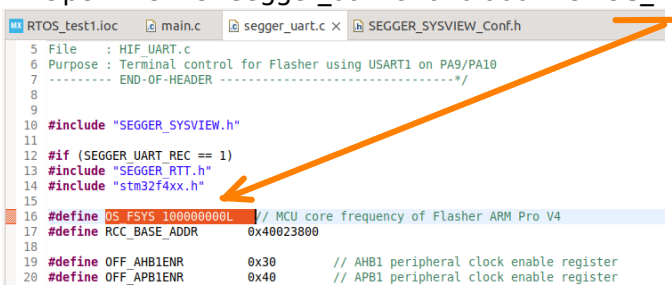
Check for the Max AHB HCLK available.
For my STM32F411VET it is mentioned as
100 MHZ.
So, setting set the value of 100 MHz for AHB clock.

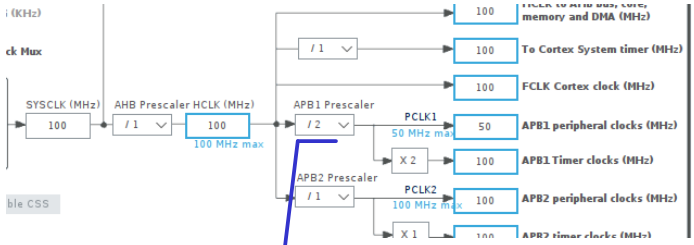




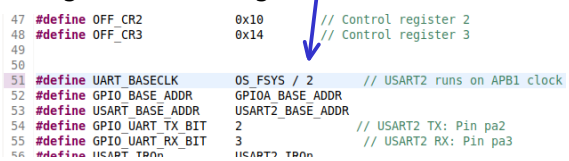
One thing to note here is the APB1 clock value which is 50 MHz in my case.

-> Open the file "segger_uart.c" and add the "OS_FSYS" value based on Max available "HCLK" value 100 MHz in my case





-> Next set the APB1 prescaler value as per the clock configuration settings for our MCU



Once done, add the following initialization in main.c inside the main function in order to configure the baud rate for the uart based continuous recording and comment the SEGGER_SYSVIEW_Start() call.