

ECEN 5803
Mastering Embedded System Architecture
(Fall-2023)

PROJECT-1 : MODULE 4
AUTO-CONFIGURE WITH STMCUBE

Date

10/12/2023

Guided By

Prof. Timothy Scherr

Submitted By

Ajay Kandagal

Kshitija Dhondage

Below are the steps to create a project with above mentioned configuration using STM32CueMX.

-
- The diagram illustrates the clock tree for the STM32MP157C. It starts with three input frequencies: 32.768 KHz, 1-50 MHz, and 12.288 MHz. The 32.768 KHz input is connected to the LSE (Low Speed External) oscillator, which provides the RTC clock (32 KHz) and the LSI (Low Speed Internal) clock (32 KHz). The 1-50 MHz input is connected to the HSE (High Speed External) oscillator, which provides the HSI (High Speed Internal) clock (16 MHz) and the PLL Source Mux input. The 12.288 MHz input is connected to the PLLI2S (Phase-Locked Loop for I2S) oscillator, which provides the PLLI2SCLK (192 MHz) and the PLLI2S output (96 MHz). The HSE input is also connected to the Main PLL (Phase-Locked Loop), which provides the HCLK (84 MHz) and the APB1 and APB2 peripheral and timer clocks (42 MHz, 84 MHz, 48 MHz). The HCLK is also connected to the APB1 and APB2 peripheral and timer clocks (42 MHz, 84 MHz, 48 MHz). The APB1 and APB2 peripheral and timer clocks are also connected to the APB1 and APB2 peripheral and timer clocks (42 MHz, 84 MHz, 48 MHz). The APB1 and APB2 peripheral and timer clocks are also connected to the APB1 and APB2 peripheral and timer clocks (42 MHz, 84 MHz, 48 MHz).

3. To set the ADC sampling rate at 100kHz, select the “Pin & Configuration” under this select “Timers” and then TIM2. The clock source is set to Internal Clock. And to generate timer OV events at 100kHz to trigger ADC following settings are used.

Parameter Settings

User Constants

Configure the below parameters :

Search (Ctrl+F)

Counter Settings

Prescaler (PSC - 16 b.. 84-1

Counter Mode Up

Counter Period (Auto... 100 - 1

Internal Clock Division No Division

auto-reload preload Enable

Trigger Output (TRGO) Par...

Master/Slave Mode (M. Disable (Trigger input effect not...

Trigger Event Selectio.. Update Event

2

$$100\text{kHz} = 84\text{MHz} / (\text{Pre-scaler} * \text{Counter Period})$$

4. Next set the PA0 to ADC1_IN0 from “Pinout View” GUI
5. To configure ADC, select “Analog” and then ADC1. Check IN0 and set the following parameters to set the ADC resolution to 12-bit and trigger source as TIM2.

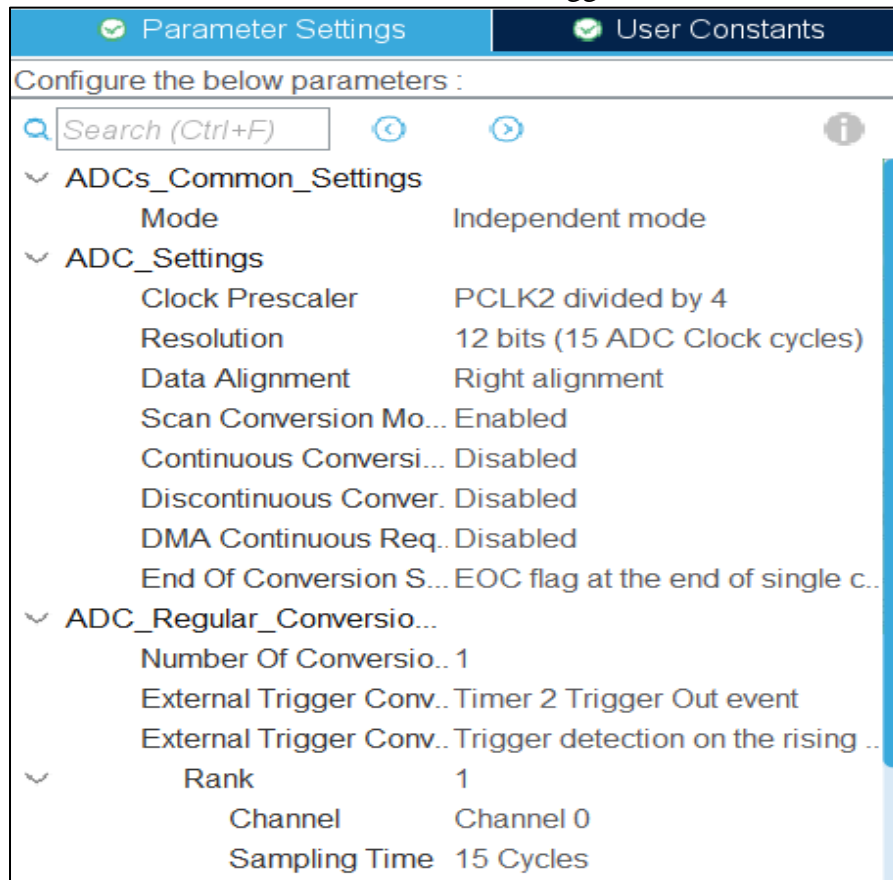


Figure 3: ADC resolution set to 12-bit and external trigger set to TIM2 events.

Also enable the global NVIC for ADC to sample the ADC at 100kHz.

6. In “Project Manager”, project name is assigned, and Tool Chain is set to MDK-ARM. Generate code and then project can be opened in Keil.

Compare this to the startup code you used in the other modules so far. How is it different? Provide a memory map for each.

On pressing the “reset” button in the IDE, the code jumps to address 0x08000234 which is the start address of Reset_Handler. The instruction contains call to the system_init and __main.

The screenshot shows the STM32CubeMX IDE with two windows open. The top window, titled 'Disassembly', shows the assembly code for the Reset_Handler function. The bottom window, titled 'main.c', shows the C code for the same function. The assembly code starts at address 0x0800022C and ends at 0x08000260. The C code starts at line 173 and ends at line 183. The assembly code is as follows:

```

__rt_exit_exit:
0x0800022C BC03      POP        {r0-r1}
0x0800022E F000F8F5  BL.W      0x0800041C __sys_exit
0x08000232 0000      MOVS      r0,r0
176:          LDR        R0, =SystemInit
0x08000234 4809      LDR        r0,[pc,#36] ; @0x0800025C
177:          BLX        R0
0x08000236 4780      BLX        r0
178:          LDR        R0, =__main
0x08000238 4809      LDR        r0,[pc,#36] ; @0x08000260
179:          BX         R0
180:          ENDP
181:
182: ; Dummy Exception Handlers (infinite loops which can be modified)
183:
184: NMI_Handler      PROC

```

The C code is as follows:

```

173      IMPORT      SystemInit
174      IMPORT      __main
175
176      LDR        R0, =SystemInit
177      BLX        R0
178      LDR        R0, =__main
179      BX         R0
180      ENDP
181
182      ; Dummy Exception Handlers (infinite loops which can be modified)
183

```

Figure 4: Code generated using STM32CubeMx, code after reset.

The sequence in Module 2 is the same but the Reset_Handler location is different and is located at address 0x08000318.

The screenshot shows the STM32CubeMX IDE with the assembly code for the Reset_Handler function. The code starts at address 0x08000318 and ends at 0x0800032A. The assembly code is as follows:

```

Reset_Handler:
0x08000318 4806      LDR        r0,[pc,#24] ; @0x08000334
0x0800031A 4780      BLX        r0
0x0800031C 4806      LDR        r0,[pc,#24] ; @0x08000338
0x0800031E 4700      BX         r0
NMI_Handler:
0x08000320 E7FE      B         0x08000320 NMI_Handler
HardFault_Handler:
0x08000322 E7FE      B         0x08000322 HardFault_Handler
MemManage_Handler:
0x08000324 E7FE      B         0x08000324 MemManage_Handler
BusFault_Handler:
0x08000326 E7FE      B         0x08000326 BusFault_Handler
UsageFault_Handler:
0x08000328 E7FE      B         0x08000328 UsageFault_Handler
SVC_Handler:
0x0800032A E7FE      B         0x0800032A SVC_Handler

```

Figure 5: Audio Code from Module 2 Audio Example, code after reset

.constdata	0x08001a0a	Section	16	system_stm32f4xx.o(.constdata)
.constdata	0x08001a1a	Section	8	system_stm32f4xx.o(.constdata)
.data	0x20000000	Section	4	main.o(.data)
.data	0x20000004	Section	12	stm32f4xx_hal.o(.data)
.data	0x20000010	Section	4	system_stm32f4xx.o(.data)
.bss	0x20000014	Section	212	main.o(.bss)
.bss	0x200000e8	Section	96	libspace.o(.bss)
HEAP	0x20000148	Section	512	startup_stm32f401xe.o(HEAP)
Heap_Mem	0x20000148	Data	512	startup_stm32f401xe.o(HEAP)
STACK	0x20000348	Section	1024	startup_stm32f401xe.o(STACK)
Stack_Mem	0x20000348	Data	1024	startup_stm32f401xe.o(STACK)
__initial_sp	0x20000748	Data	0	startup_stm32f401xe.o(STACK)

Figure 6: Code generated using STM32CubeMx, memory initialization.

__heap_base	0x20000428	Data	0	startup_stm32f401xe.o(HEAP)
Image\$\$RW_IRAM1\$\$ZI\$Limit	0x20000828	Number	0	anon\$\$obj.o ABSOLUTE
__heap_limit	0x20000828	Data	0	startup_stm32f401xe.o(HEAP)
__initial_sp	0x20018000	Number	0	startup_stm32f401xe.o ABSOLUTE

Figure 7: Code from Module2 Audio example

In the STM32CubeMx generated code, the heap base is at 0x20000148 and in the audio code of Module 2, it is at 0x20000428. In CubeMx code, the heap size is mentioned as 512 bytes, and if we do the calculation for the module 2 code by subtracting the __heap_base with __heap_limit we get 0x400 which is 1024 bytes.

=====						
	Code (inc. data)	RO Data	RW Data	ZI Data	Debug	
	6262	420	462	20	1844	517986
	6262	420	462	20	1844	517986
	6262	420	462	20	0	0
						Grand Totals
						ELF Image Totals
						ROM Totals
=====						
Total RO	Size (Code + RO Data)			6724 (6.57kB)	
Total RW	Size (RW Data + ZI Data)			1864 (1.82kB)	
Total ROM	Size (Code + RO Data + RW Data)			6744 (6.59kB)	
=====						

Figure 8: Map file of code generated using STM32CubeMx.

=====						
	Code (inc. data)	RO Data	RW Data	ZI Data	Debug	
15620	1260	1900	72	1612	163982	Grand Totals
15620	1260	1900	72	1612	163982	ELF Image Totals
15620	1260	1900	72	0	0	ROM Totals
=====						
Total RO	Size (Code + RO Data)			17520	(17.11kB)
Total RW	Size (RW Data + ZI Data)			1684	(1.64kB)
Total ROM	Size (Code + RO Data + RW Data)			17592	(17.18kB)

Figure 9: Map file of code from Module2 Audio example

Comparing the map files generated for the two codes, it is evident that the Module 2 code uses more code memory (around 15.6 kB), whereas the code generated by CubeMx takes only 6.2 kB. This might be because the Audio code example uses more Hal modules for PWM, serial and Analog because of which the library takes additional space. The library in Module 2 example takes up 10.3kB and in CubeMx, it is 880 bytes. However, the RAM usage in the CubeMx code is higher even with very few user variables used.