CHIP
Studenckie Koło Naukowe
Politechniki Gdańskiej

Warsztaty STM32L073

Materiały pomocnicze do zajęć

SKN CHIP

Gdańsk, 5-12/12/2018

CHIP
Studenckie Koło Naukowe
Politechniki Gdańskiej

Plan warsztatów

- Ogólne informacje o ekosystemie STM32.
- Podstawowe właściwości mikrokontrolera STM32L073.
- Omówienie i podłączenie płytki Nucleo64.
- Omówienie i pierwszy projekt CubeMX.
- Omówienie i pierwszy projekt Atollic True Studio.
- Podgląd zmiennych globalnych przy użyciu STM Studio.
- Debugowanie oprogramowania poprzez wydruki na terminalu.

CHIP
Studenckie Koło Naukowe
Politechniki Gdańskiej

Plan warsztatów c.d.

- Odczyt USART2 w poolingu.
- Bloki TIMER (przyciemnianie LD2, sterowanie serwomechanizmem, pomiar czasu w czujniku odległości).
- Przetwornik ADC w trybie pracy pojedynczej.

Wymagane oprogramowanie (wszystko jest open/free)

- Generator kodu CubeMX
- Biblioteki STM32L0 do CubeMX
- Środowisko programistyczne IDE Atollic True Studio
- Terminal Putty
- ST-Link Utility Software
- STM Studio Software

PKM, LKM – prawy/lewy klawisz myszki

Porównanie rdzeni ARM Cortex M [wg Wiki]

ARM Cortex-M instruction sets[6][7]

ARM Cortex-M	Thumb	Thumb-2	Hardware multiply	Hardware divide	Saturated math	DSP extensions	Floating-point	ARM architecture	Core architecture
Cortex-M0[1]	Most	Subset	1 or 32 cycle	No	No	No	No	ARMv6-M[6]	Von Neumann
Cortex-M0+[2]	Most	Subset	1 or 32 cycle	No	No	No	No	ARMv6-M[6]	Von Neumann
Cortex-M1[3]	Most	Subset	3 or 33 cycle	No	No	No	No	ARMv6-M[6]	Von Neumann
Cortex-M3[4]	Entire	Entire	1 cycle	Yes	Yes	No	No	ARMv7-M[7]	Harvard
Cortex-M4[5]	Entire	Entire	1 cycle	Yes	Yes	Yes	Optional	ARMv7E-M[7]	Harvard

- Note: The Cortex-M0 / M0+ / M1 doesn't include these **Thumb** instructions: CBZ, CBNZ, IT; nor does it include a divide instruction.[6][7]
- Note: The Cortex-M0 / M0+ / M1 only include these **Thumb-2** instructions: DMB, DSB, ISB, MRS, MSR.[6][7]
- Note: If a smaller silicon die size is required, the Cortex-M0 / M0+ / M1 can implement a smaller and slower multiply instruction.

Announced	
Year	**Core**
2004	Cortex-M3
2007	Cortex-M1
2009	Cortex-M0
2010	Cortex-M4
2012	Cortex-M0+
2014	Cortex-M7

ARM Cortex-M optional components[6][7]

ARM Cortex-M	SysTick Timer	Bit-banding	Memory Protection Unit (MPU)	Tightly-Coupled Memory (TCM)	CPU cache	Memory architecture	ARM architecture
Cortex-M0[1]	Optional*	Optional[9]	No	No	No[10]	Von Neumann	ARMv6-M
Cortex-M0+[2]	Optional*	Optional[9]	Optional (8)	No	No	Von Neumann	ARMv6-M
Cortex-M1[3]	Optional	Optional	No	Optional	No	Von Neumann	ARMv6-M
Cortex-M3[4]	Yes	Optional*	Optional (8)	No	No	Harvard	ARMv7-M
Cortex-M4[5]	Yes	Optional*	Optional (8)	No	⊘ **Possible**[11]	Harvard	ARMv7E-M
Cortex-M7	Yes	TBD*	Optional (8 or 16)	Optional	Optional	Harvard	ARMv7E-M

Pierwsze podłączenie płytki Nucleo 64

- Podłączamy płytkę kablem USB do komputera z zainstalowanym uprzednio oprogramowaniem `ST-Link Utility`
- Uruchamiamy `ST-Link Utility` i wybieramy kolejno menu `ST-LINK/Firmware Update/Device Connect` i ewentualnie `YES` jeśli jest dostępna nowsza wersja oprogramowania programatora.

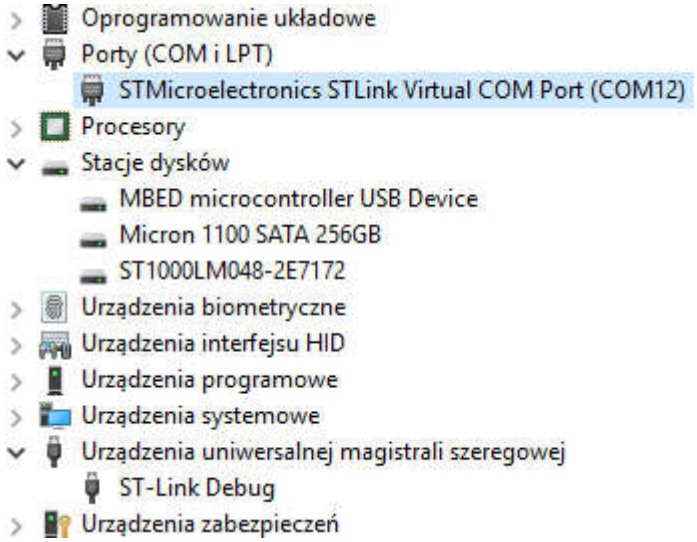
Sprawdzamy czy programator jest widoczny w systemie

Wybieramy menu `PKM` na `Komputer/Menedżer Urządzeń`

Powinny być widoczne 3 następujące urządzenia:

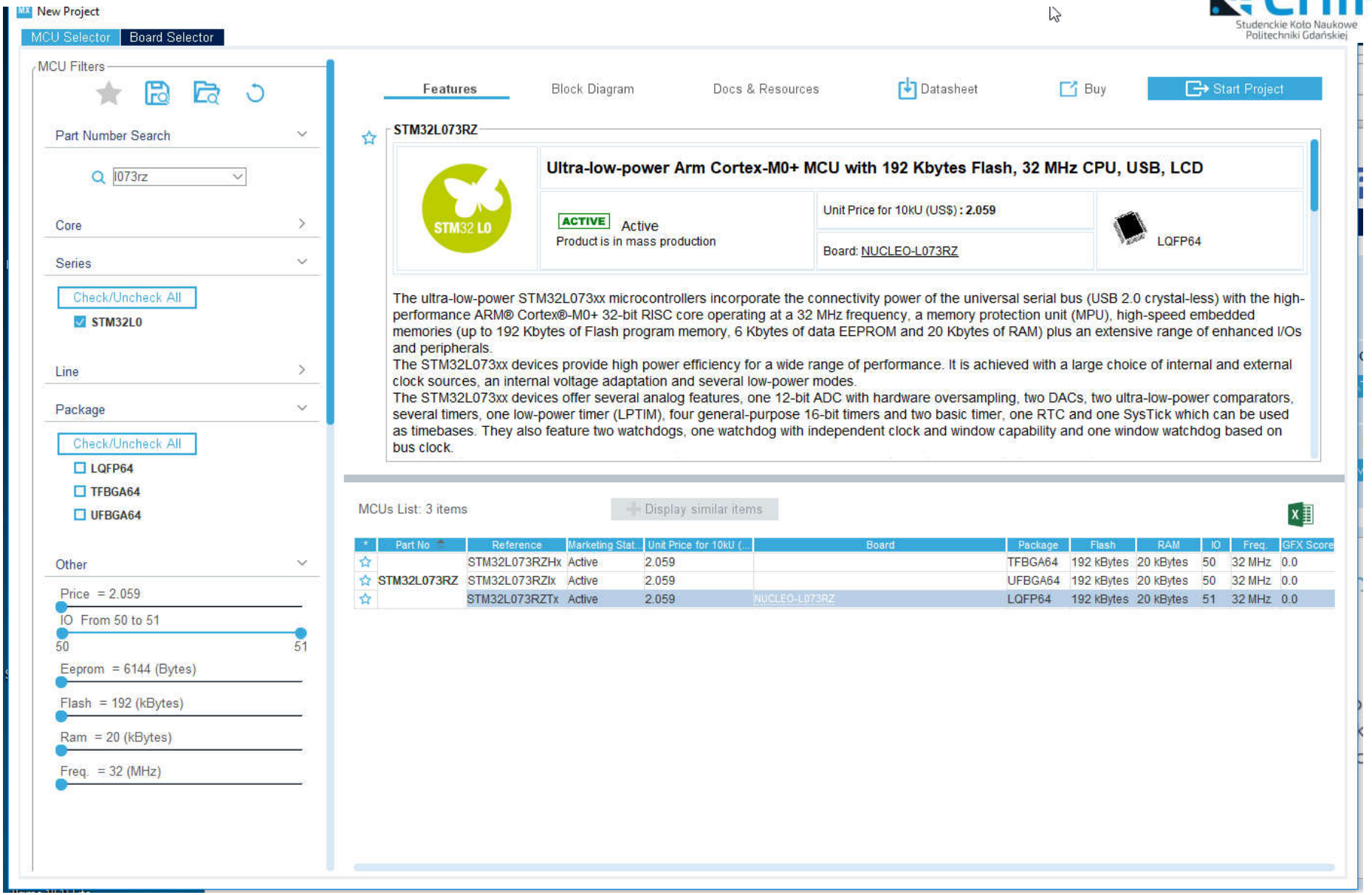
- `STMicroelectronics STLink Virtual COM Port(COMxx),`
- `MBED microcontroller USB Device,`
- `ST-Link Debug.`

Brak tych urządzeń oznacza brak zainstalowanych driverów programatora ST-Link.



Generacja kodu CubeMX (tylko piny I/O)

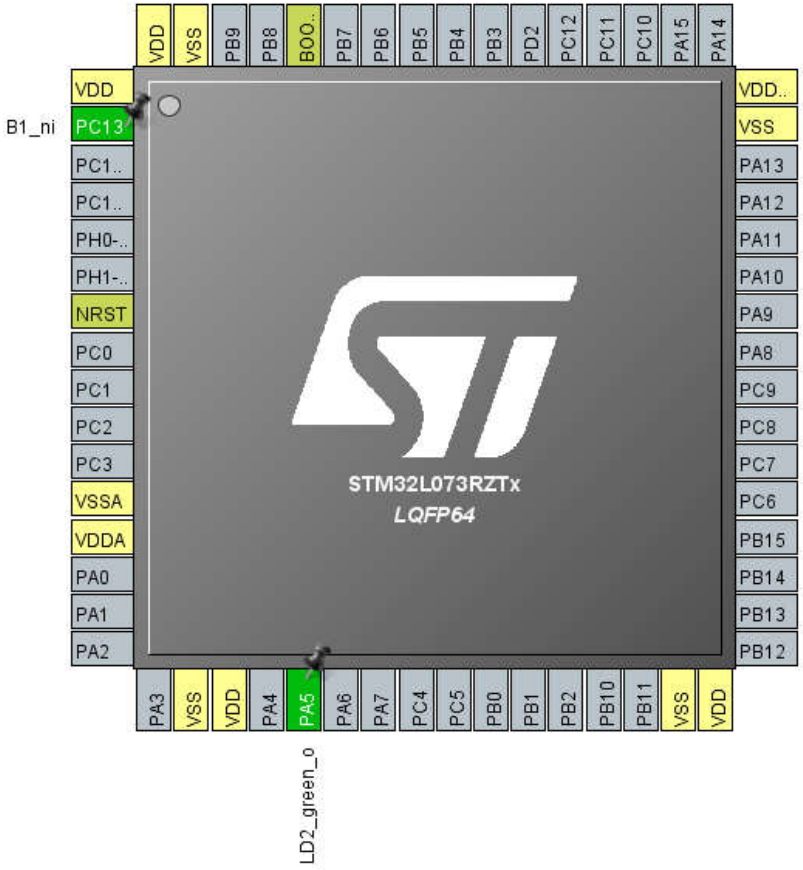
1) Uruchamiamy oprogramowanie `CubeMX`.
2) Klikamy na menu `File/New Project`.
3) Wybieramy układ jak na naszej płytce czyli `STM32L073RZT6.`
4) Klikamy `Start Project`.


New Project
MCU Selector
Board Selector
MCU Filters
Part Number Search
l073rz
Core
Series
Check/Uncheck All
STM32L0
Line
Package
Check/Uncheck All
LQFP64
TFBGA64
UFBGA64
Other
Price = 2.059
IO From 50 to 51
50
51
Eeprom = 6144 (Bytes)
Flash = 192 (kBytes)
Ram = 20 (kBytes)
Freq. = 32 (MHz)
Features
Block Diagram
Docs & Resources
Datasheet
Buy
Start Project
STM32L073RZ
Ultra-low-power Arm Cortex-M0+ MCU with 192 Kbytes Flash, 32 MHz CPU, USB, LCD
ACTIVE
Active
Product is in mass production
Unit Price for 10kU (US$) : 2.059
Board: NUCLEO-L073RZ
LQFP64
The ultra-low-power STM32L073xx microcontrollers incorporate the connectivity power of the universal serial bus (USB 2.0 crystal-less) with the high-performance ARM® Cortex®-M0+ 32-bit RISC core operating at a 32 MHz frequency, a memory protection unit (MPU), high-speed embedded memories (up to 192 Kbytes of Flash program memory, 6 Kbytes of data EEPROM and 20 Kbytes of RAM) plus an extensive range of enhanced I/Os and peripherals.
The STM32L073xx devices provide high power efficiency for a wide range of performance. It is achieved with a large choice of internal and external clock sources, an internal voltage adaptation and several low-power modes.
The STM32L073xx devices offer several analog features, one 12-bit ADC with hardware oversampling, two DACs, two ultra-low-power comparators, several timers, one low-power timer (LPTIM), four general-purpose 16-bit timers and two basic timer, one RTC and one SysTick which can be used as timebases. They also feature two watchdogs, one watchdog with independent clock and window capability and one window watchdog based on bus clock.
MCUs List: 3 items
Display similar items
Part No
Reference
Marketing Stat...
Unit Price for 10kU (...
Board
Package
Flash
RAM
IO
Freq.
GFX Score
STM32L073RZHx Active 2.059 TFBGA64 192 kBytes 20 kBytes 50 32 MHz 0.0
STM32L073RZ STM32L073RZIx Active 2.059 UFBGA64 192 kBytes 20 kBytes 50 32 MHz 0.0
STM32L073RZTx Active 2.059 NUCLEO-L073RZ LQFP64 192 kBytes 20 kBytes 51 32 MHz 0.0

Konfiguracja I/O

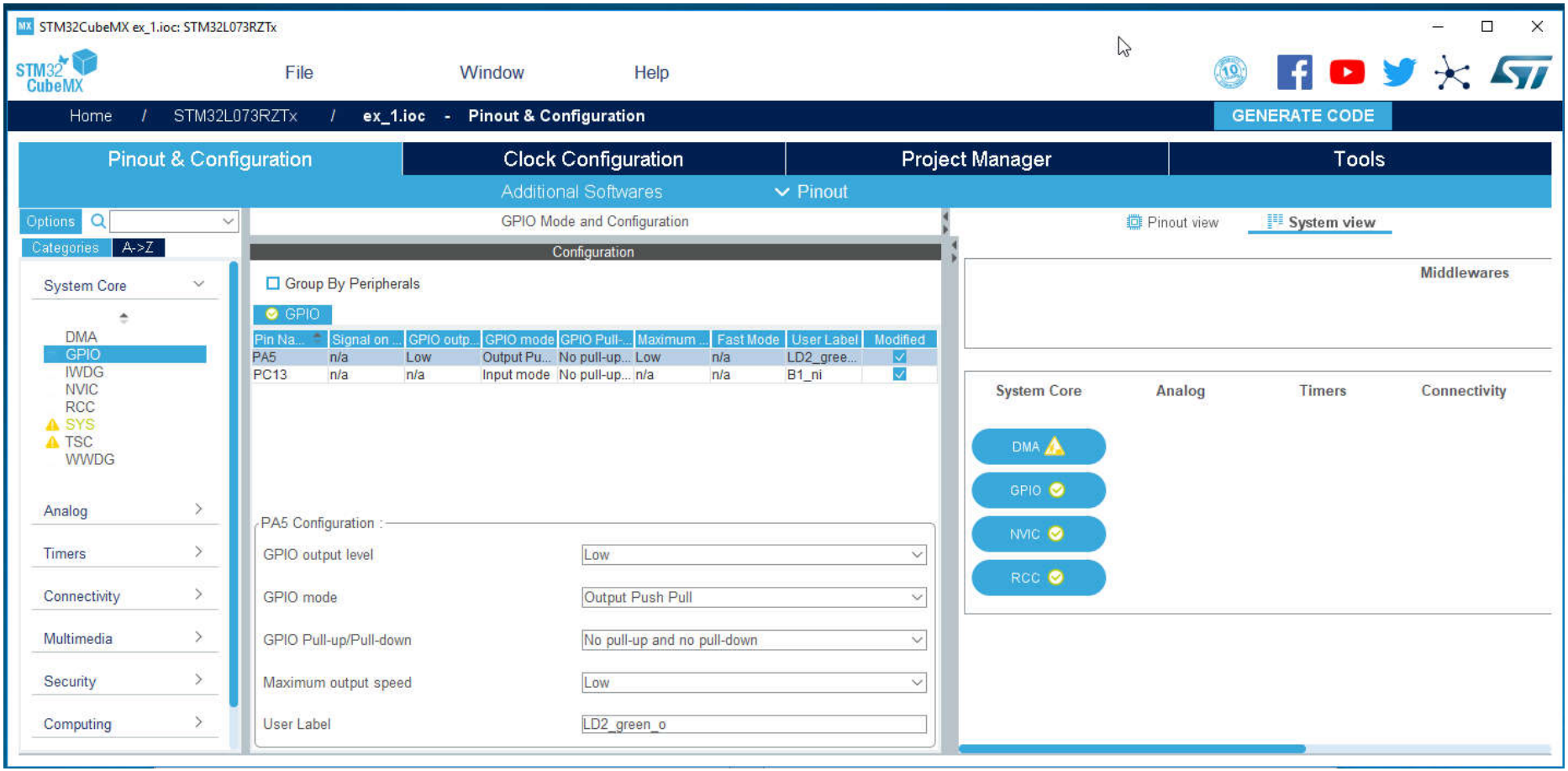
Zakładka `Pinout & Configuration`, podzakładka `Pinout view`:

- Port `PC13` ustawiamy jako wejściowy i nazywamy `B1_ni`
- Port `PA5` ustawiamy jako wyjściowy i nazywamy `LD2_green_o`
- Po lewej stronie pojawiają się możliwe konflikty bloków/pinów



Dodatkowa konfiguracja peryferiów

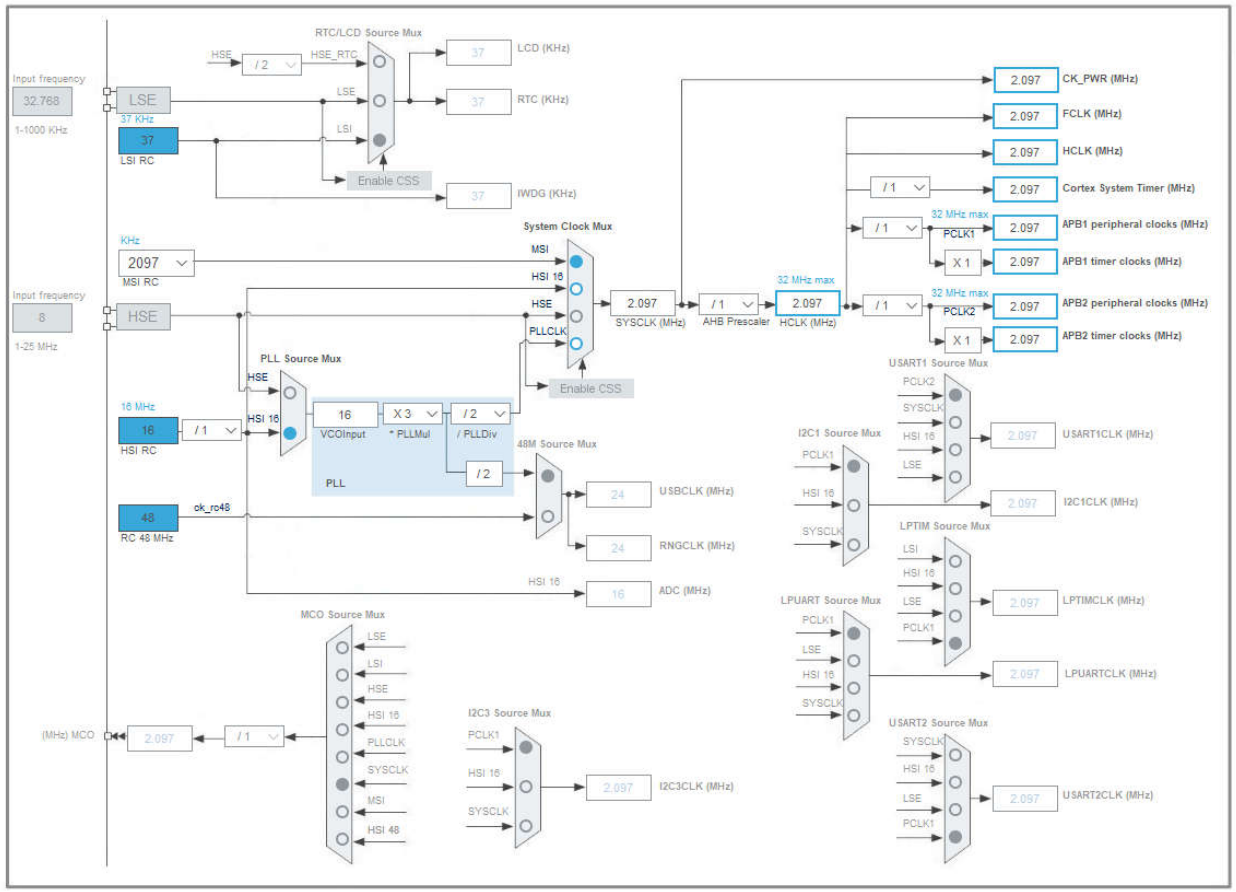
- Podzakładka `System view`
- Możliwa szczegółowa konfiguracja wyprowadzeń I/O jak i pozostałych bloków



Konfiguracja zegarów układu

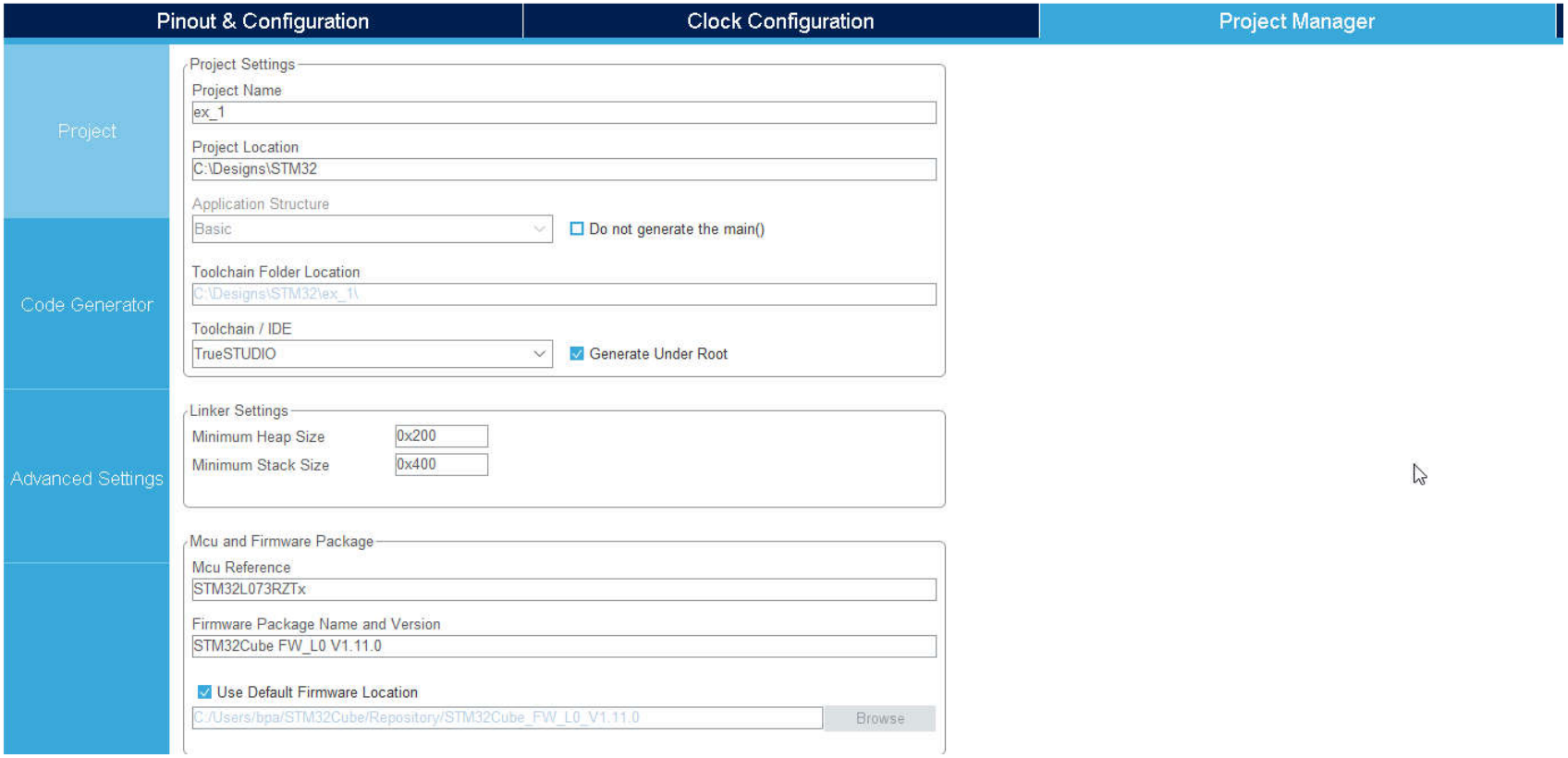
Zakładka `Clock Configuration`

- Mamy możliwość skonfigurowania zegarów układu.
- Pozostawiamy wartości domyślne (jak na rysunku).



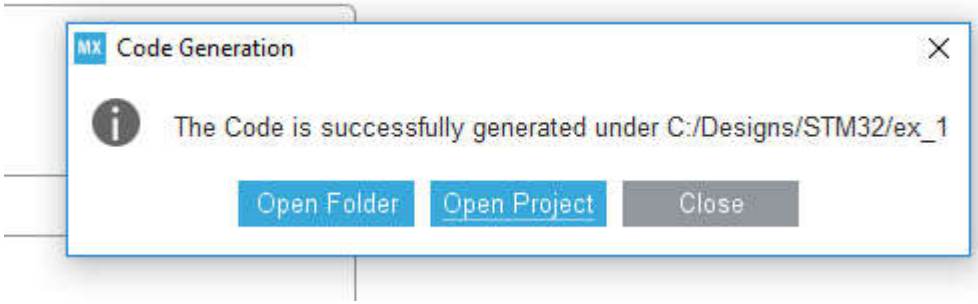
Konfiguracja ogólna projektu

- Zakładka `Project Manager`.
- Ustawienia jak na poniższym rysunku z możliwym wyjątkiem na katalog docelowy.
- Zapisujemy projekt (`Ctrl-S`).
- Generujemy kod – przycisk `GENERATE CODE`.



Otworzenie wygenerowanego kodu

- Należy kliknąć w przycisk `Open Project`.
- Uruchomi się oprogramowanie `Atollic True Studio` (o ile jest zainstalowane).
- Potwierdzamy domyślny `„Workspace”`.
- Po lewej stronie można znaleźć wygenerowany kod i dołączone biblioteki HAL.
- Rozwijamy `ex_1/src` i otwieramy plik `main.c`.
- Uzupełniamy kod aby zaświecić diodą LED.



Modyfikacja `main.c` - przykład

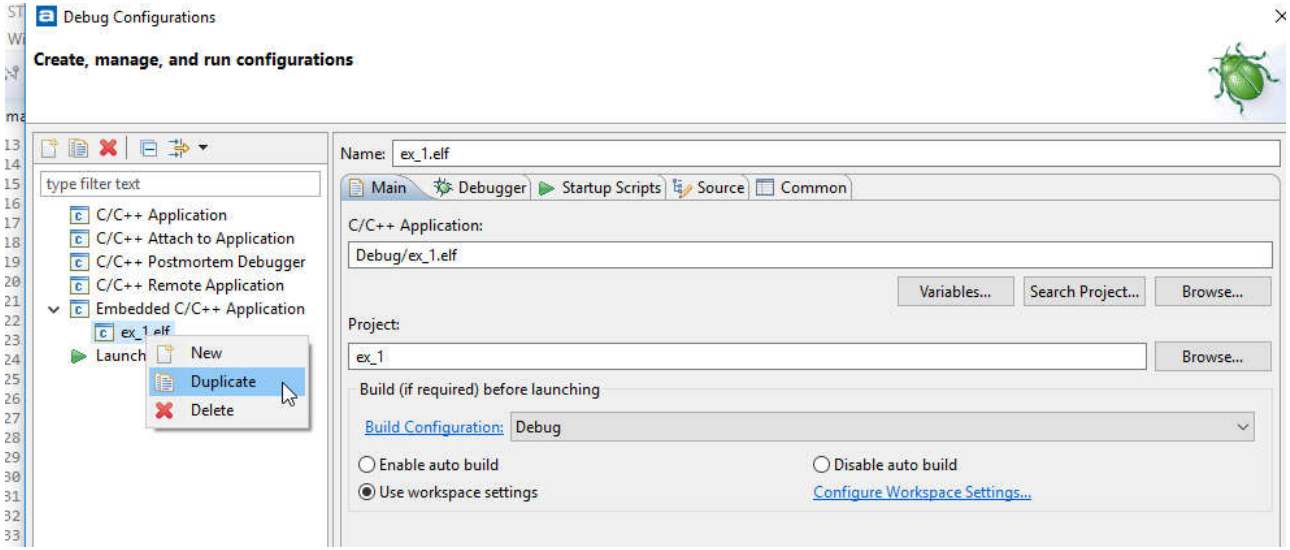
```
/* USER CODE BEGIN 2 */
 uint16_t del=200;
 uint16_t button=0;
 /* USER CODE END 2 */
 /* Infinite loop */
 /* USER CODE BEGIN WHILE */
 while (1)
 {
   /* USER CODE END WHILE */
   /* USER CODE BEGIN 3 */
 button=HAL_GPIO_ReadPin(B1_ni_GPIO_Port,B1_ni_Pin);
 if (button)
 del=500;
 else
 del=100;
 HAL_GPIO_WritePin(LD2_green_o_GPIO_Port,LD2_green_o_Pin,1);
 HAL_Delay(del);
 HAL_GPIO_WritePin(LD2_green_o_GPIO_Port,LD2_green_o_Pin,0);
 HAL_Delay(del);
 }
 /* USER CODE END 3 */
```

Uzupełnianie kodu – przydatne informacje

- Funkcje z biblioteki HAL mają typowo nazwy funkcji w formacie: `HAL_(nazwa_bloku)_(nazwa_funkcjonalności)(zmienne wejściowe)`, np.: `HAL_GPIO_ReadPin(B1_ni_GPIO_Port,B1_ni_Pin);`
- Wciśnięcie `Ctrl-spacja` rozwija możliwe uzupełnienia funkcji.
- Najechanie na zmienną i wciśnięcie `F3` przechodzi do deklaracji zmiennej.
- Własny kod należy wpisywać pomiędzy komentarze typu `/* USER CODE BEGIN ??? */` a `/* USER CODE END ??? */`, w przeciwnym razie ponowna generacja kodu startowego skasuje wpisany kod.
- Jeśli wystąpiła potrzeba zmiany konfiguracji bloków układu STM32 można wykonać kolejną generację kodu. Jeśli chcemy zachować nasz dopisany wcześniej kod - patrz punkt powyżej.
- Najechanie i przytrzymanie na chwilę kursora myszki na funkcji/zmiennej powoduje przedstawienie opisu tego elementu.
- Aby uruchomić kod (w trybie debug) klikamy menu `Run/Debug`.
- Uruchamianie w trybie Run nie jest domyślnie skonfigurowane i albo można uruchamiać kod przez Debug albo dodać konfigurację Run.

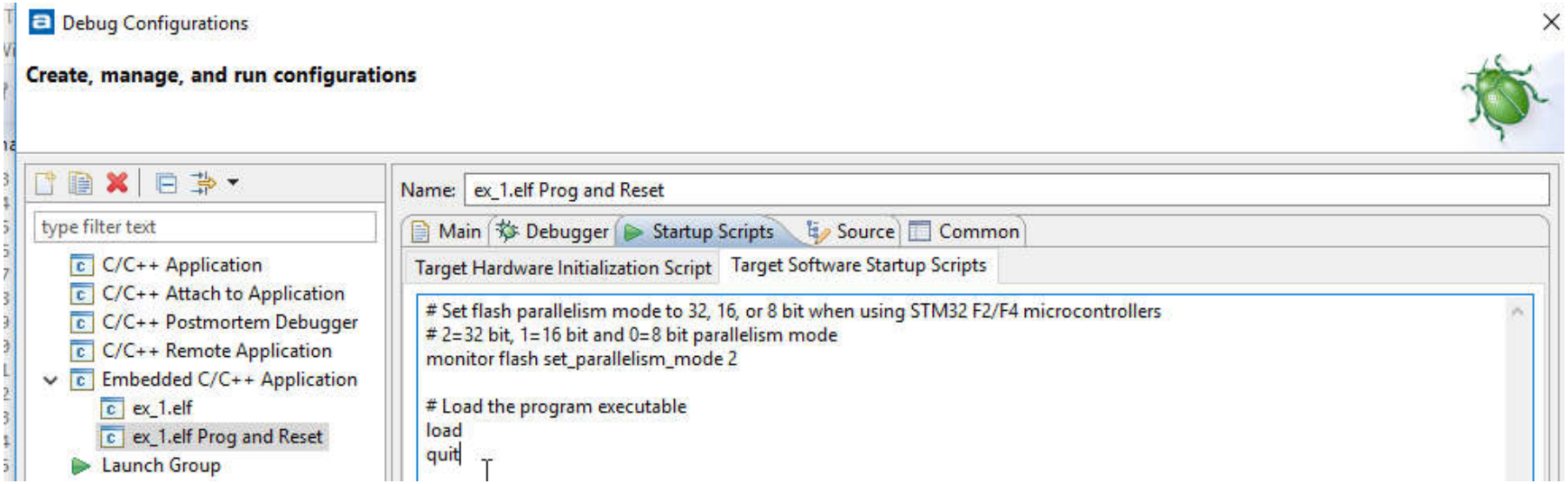
Dodanie konfiguracji tylko do programowania

1) Aby poniższe było możliwe do wykonania chociaż 1 raz powinno być wykonane wcześniej menu `Run/Debug.`
2) Skopiowanie konfiguracji Debug, menu `Run/Debug Configurations` `PKM` na nazwie konfiguracji `Debug` (u nas `ex_1.elf`) i wybranie opcji `Duplicate`.

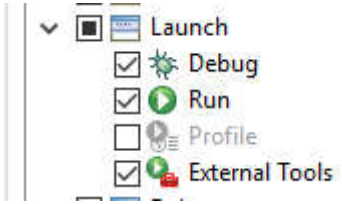


2) Zmieniamy nazwę na `ex_1.elf Prog and Reset` a następnie wybieramy zakładkę `Startup Scripts/Target Soft...`

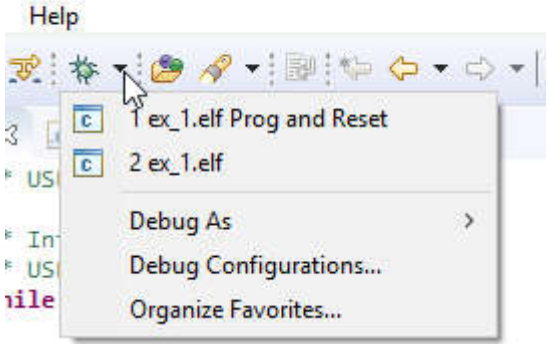
3) Zmieniamy zawartość wpisując `quit` za poleceniem `load` a resztę usuwamy.



4) Aby było dostępne pole wyboru wersji konfiguracji Debugu należy dodatkowo dodać menu: `Window/Perspective>/Customomize Perspective...` i tu zaznaczyć jak na poniższym rysunku

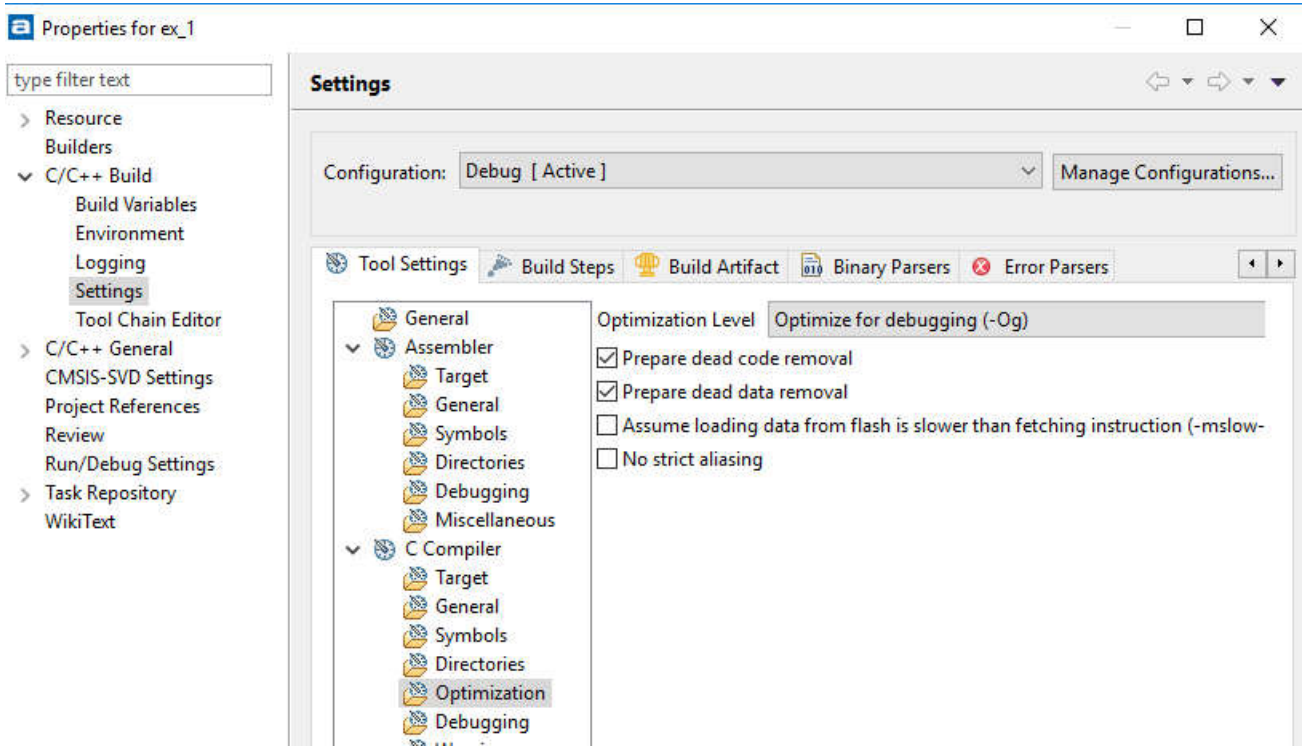


5) Wyboru czy ma nastąpić tylko wgranie pamięci Flash czy też dodatkowo Debugowanie kodu wybieramy przez menu:

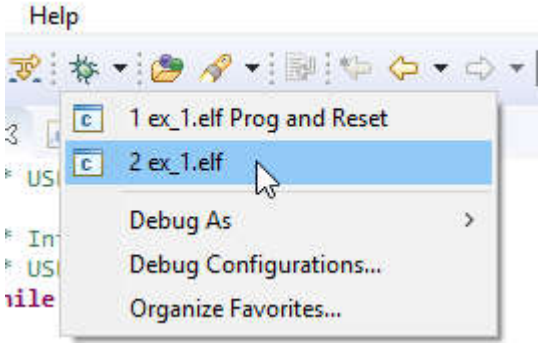


Debugowanie kodu

1) Przed wykonaniem sesji Debugu warto jest skonfigurować ustawienia optymalizacji kompilatora na niskie, menu `Project/Build Settings/Tool Settings` wybieramy konfigurację `Debug` i dla `C/C++ Build/C Compiler` wybieramy `Optimize for debugging(-Og)`.



2) Uruchamiamy sesję Debug poprzez menu:



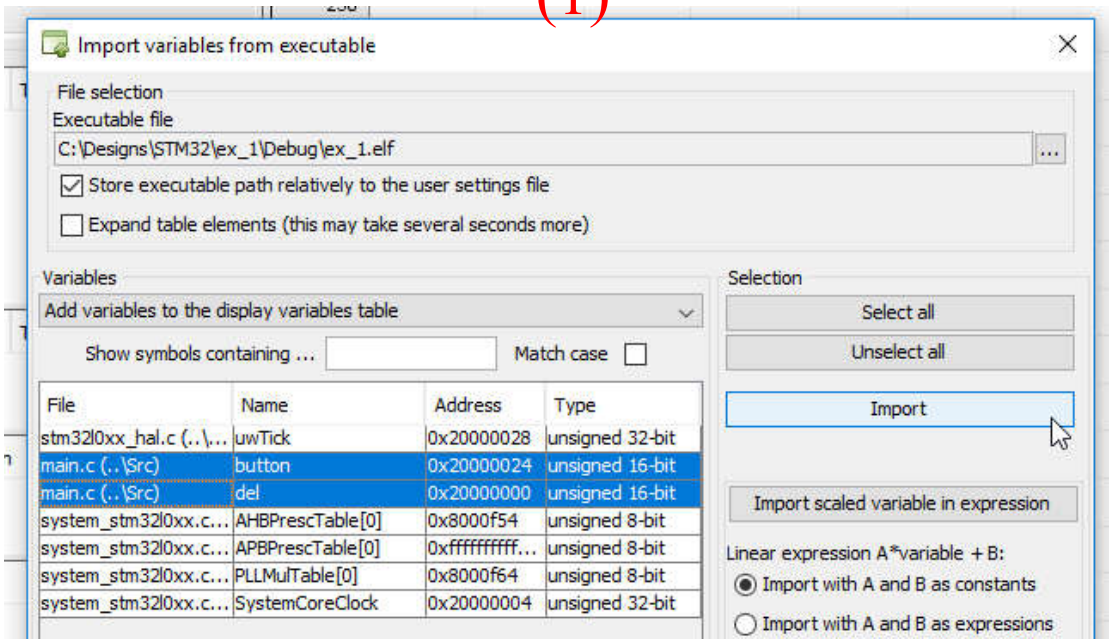
3) Zostanie wgrana zawartość pamięci Flash do mikrokontrolera a widok IDE zmieni perspektywę na Debug. Klawiszami

można uruchamiać program, przechodzić pomiędzy poszczególnymi liniami kodu. W pozostałych oknach możliwe jest wyświetlanie zawartości rejestrów procesora, zmiennych i pamięci. Możliwe jest również założenie breakpoint'u w kodzie programu (PKM na danej linii kodu i wybranie odpowiedniej opcji).

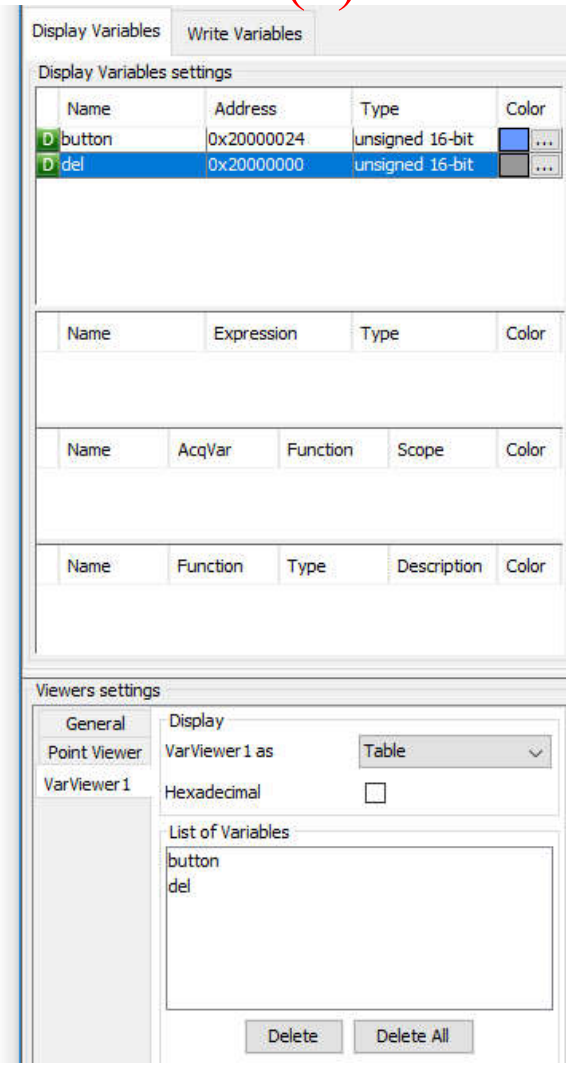
Podgląd zmiennych w STM Studio

- Możliwy jest wyłącznie podgląd zmiennych globalnych.
- Wgrywamy zawartość do mikrokontrolera (Debug).
- Uruchamiamy oprogramowanie `STM Studio`.
- Wybieramy opcję `File/Import variables` (1).
- Wybieramy zmienne do obserwacji i klikamy `Import` oraz `Close`.
- Przeciągamy zmienne przeznaczone do obserwacji do pola `List of Variables` (2).
- Zmieniamy rodzaj prezentacji na `Table`.
- W zależności od stanu przycisku B1 aktualizują się na bieżąco wartości w kolumnie `Read Value` (3).

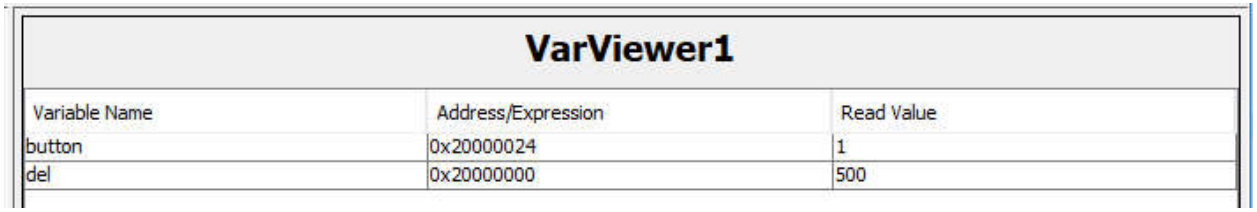
(1)


Import variables from executable
File selection
Executable file
C:\Designs\STM32\ex_1\Debug\ex_1.elf
Store executable path relatively to the user settings file
Expand table elements (this may take several seconds more)
Variables
Add variables to the display variables table
Show symbols containing ...
Match case
File
Name
Address
Type
stm32l0xx_hal.c (..\...
uwTick
0x20000028
unsigned 32-bit
main.c (..\Src)
button
0x20000024
unsigned 16-bit
main.c (..\Src)
del
0x20000000
unsigned 16-bit
system_stm32l0xx.c...
AHBPrescTable[0]
0x8000f54
unsigned 8-bit
system_stm32l0xx.c...
APBPrescTable[0]
0xffffffffff...
unsigned 8-bit
system_stm32l0xx.c...
PLLMulTable[0]
0x8000f64
unsigned 8-bit
system_stm32l0xx.c...
SystemCoreClock
0x20000004
unsigned 32-bit
Selection
Select all
Unselect all
Import
Import scaled variable in expression
Linear expression A*variable + B:
Import with A and B as constants
Import with A and B as expressions

(2)

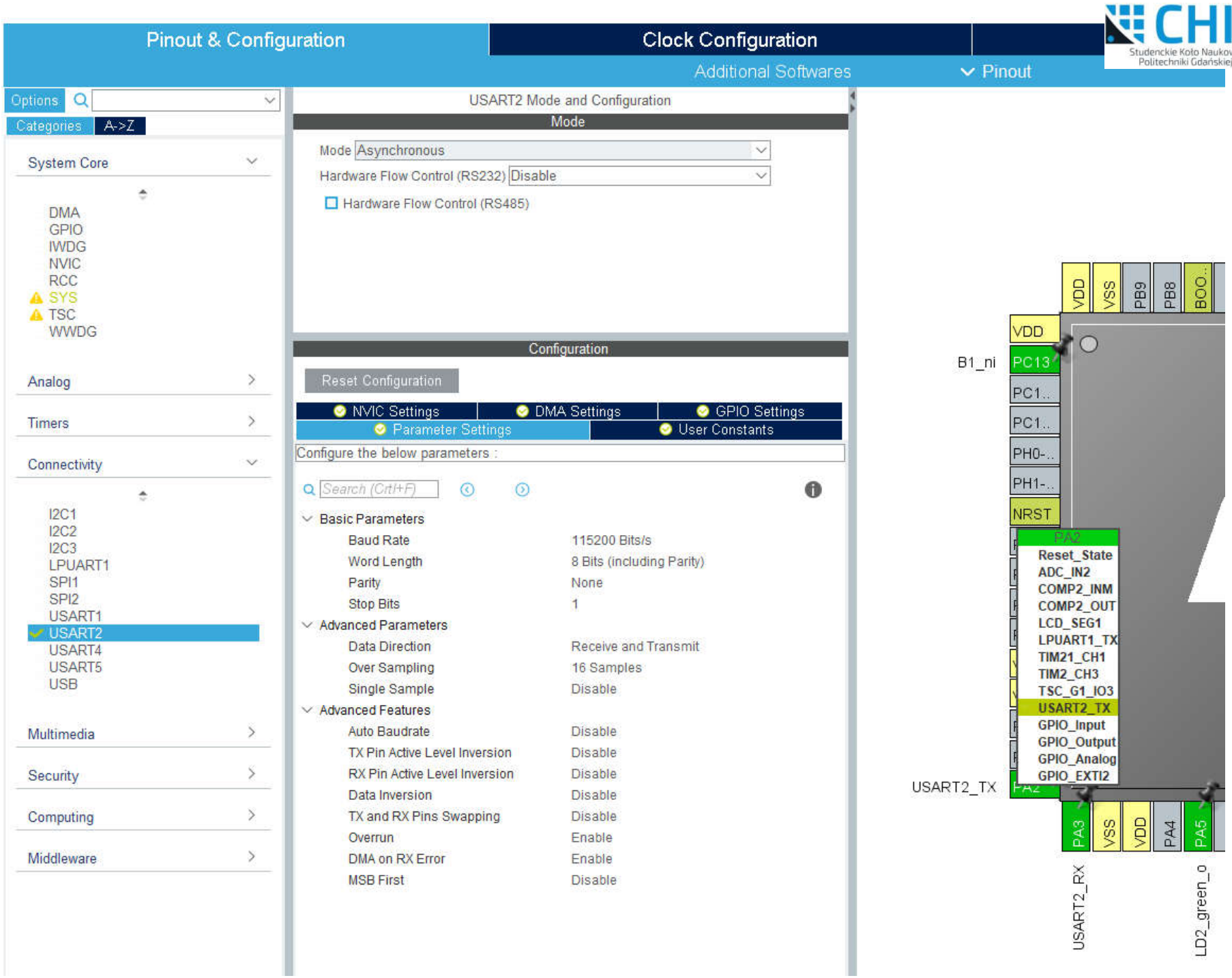

Display Variables
Write Variables
Display Variables settings
Name
Address
Type
Color
button
0x20000024
unsigned 16-bit
del
0x20000000
unsigned 16-bit
Name
Expression
Type
Color
Name
AcqVar
Function
Scope
Color
Name
Function
Type
Description
Color
Viewers settings
General
Point Viewer
VarViewer1
Display
VarViewer1 as
Table
Hexadecimal
List of Variables
button
del
Delete
Delete All

(3)


VarViewer1
Variable Name
Address/Expression
Read Value
button
0x20000024
1
del
0x20000000
500

Debugowanie kodu poprzez funkcję `printf()`

- Programator/debuger ST-Link v2-1 ma wbudowany konwerter USB-rs232. Domyślnie wyprowadzenia tego portu połączone są z PA2 i PA3.
- Wracamy do konfiguracji projektu w `CubeMX`.
- Uaktywniamy wyprowadzenia `PA2/PA3` jako `USART2_???`.
- Uaktywniamy `USART2` i ustawiamy parametry jak na kolejnym rysunku.
- Generujemy nowy kod i ponownie wczytujemy projekt do `Atollic True Studio`.


Pinout & Configuration
Clock Configuration
Additional Softwares
Pinout
CHIP
Studenckie Koło Naukowe Politechniki Gdańskiej
Options
Categories
A->Z
System Core
DMA
GPIO
IWDG
NVIC
RCC
SYS
TSC
WWDG
Analog
Timers
Connectivity
I2C1
I2C2
I2C3
LPUART1
SPI1
SPI2
USART1
USART2
USART4
USART5
USB
Multimedia
Security
Computing
Middleware
USART2 Mode and Configuration
Mode
Mode Asynchronous
Hardware Flow Control (RS232) Disable
Hardware Flow Control (RS485)
Configuration
Reset Configuration
NVIC Settings
DMA Settings
GPIO Settings
Parameter Settings
User Constants
Configure the below parameters :
Search (Crtl+F)
Basic Parameters
Baud Rate 115200 Bits/s
Word Length 8 Bits (including Parity)
Parity None
Stop Bits 1
Advanced Parameters
Data Direction Receive and Transmit
Over Sampling 16 Samples
Single Sample Disable
Advanced Features
Auto Baudrate Disable
TX Pin Active Level Inversion Disable
RX Pin Active Level Inversion Disable
Data Inversion Disable
TX and RX Pins Swapping Disable
Overrun Enable
DMA on RX Error Enable
MSB First Disable
VDD
VSS
PB9
PB8
B1_ni
PC13
PC1..
PC1..
PH0-..
PH1-..
NRST
PA2
Reset_State
ADC_IN2
COMP2_INM
COMP2_OUT
LCD_SEG1
LPUART1_TX
TIM21_CH1
TIM2_CH3
TSC_G1_IO3
USART2_TX
GPIO_Input
GPIO_Output
GPIO_Analog
GPIO_EXTI2
USART2_TX
PA3
VSS
VDD
PA4
PA5
USART2_RX
LD2_green_o

Niezbędne modyfikacje kodu `main.c`

Dodanie nagłówka:

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

Deklaracja funkcji obsługi wydruku:

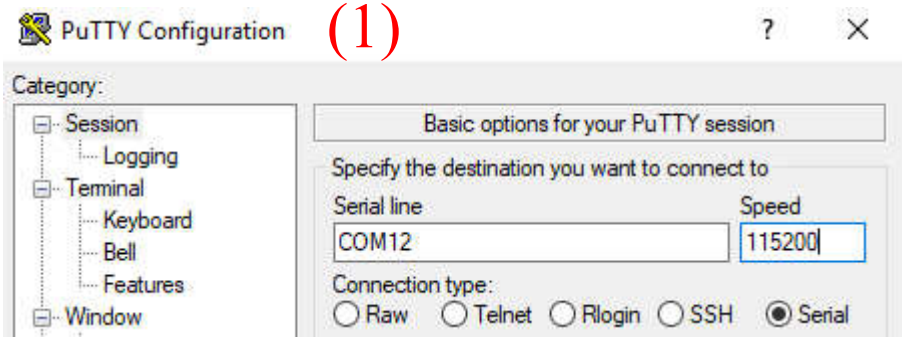
```
/* USER CODE BEGIN 4 */
int _write(int file, char *ptr, int len)
{
HAL_UART_Transmit(&huart2,ptr,len,10);
return len;
}
/* USER CODE END 4 */
```

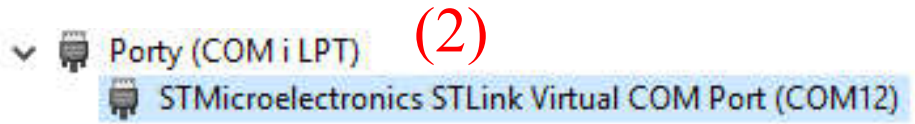
Przykłady użycia funkcji `printf()`:

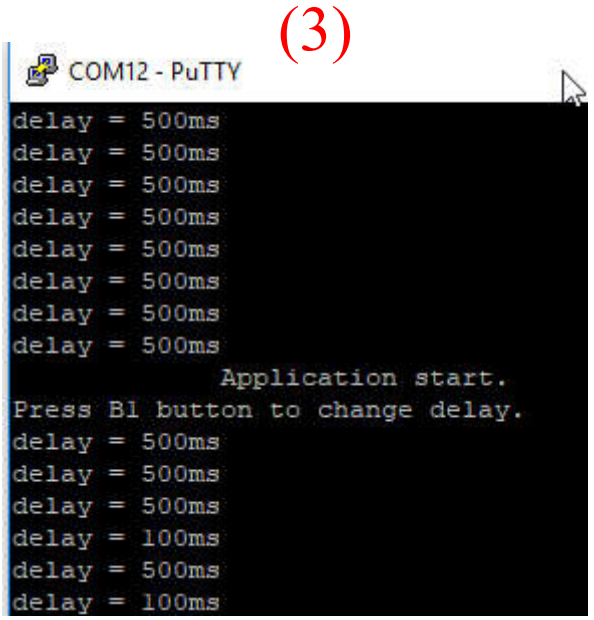
```
 /* USER CODE BEGIN 2 */
  printf("Application start.\n\r");
  printf("Press B1 button to change delay.\n\r");
  /* USER CODE END 2 */


 HAL_Delay(del);
  printf("delay = %dms\n\r",del);
  }
  /* USER CODE END 3 */
```

`printf()` - sprawdzenie działania

- Uruchamiamy `putty` (1) konfigurując go uprzednio na nr portu COM znaleziony w managerze urządzeń (2).
- W oknie terminala (3) powinny pojawić się komunikaty jak na rysunku poniżej.







Dodanie odczytu UART poprzez funkcję `getchar()`

(1) Dodanie kodu definiującego funkcję `_read()`

```
int _read(int file, char *result, size_t len)
{
    HAL_StatusTypeDef status;
    int retcode = 0;
    if (len != 0) {
        status = HAL_UART_Receive( &huart2, (uint8_t *) result, len, HAL_MAX_DELAY);
        if (status == HAL_OK) {
            retcode = len;
        } else {
            retcode = -1;
        }
    }
    return( retcode);
}
/* USER CODE END 4 */
```

(2) Modyfikacja kodu `main.c` (zmniejszenie standardowego rozmiaru bufora wejściowego)

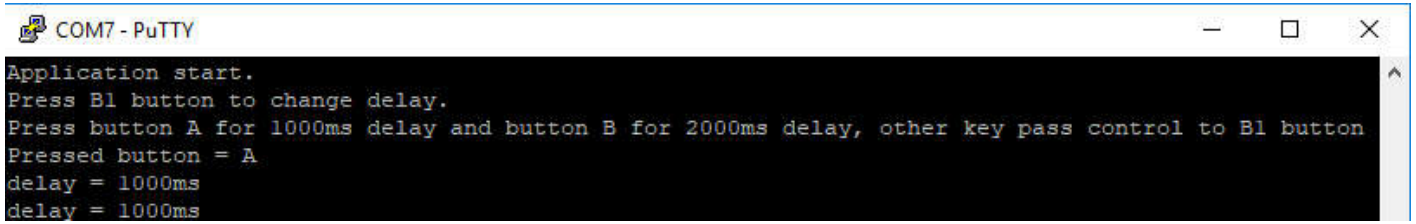
```
 //(for getchar funcion) setting 0 buffer size instead of standard 1k, it should be inserted before
 // first getchar() function usage
  setvbuf(stdin, NULL, _IONBF, 0);
```

Dodanie odczytu UART poprzez funkcję `getchar()` c.d.

(3) Modyfikacja kodu `main.c` (wykorzystanie funkcji `getchar()` – poniżej prosty przykład)

```
printf("Press button A for 1000ms delay and button B for 2000ms delay, other key pass control to B1
    button\n\r");
char ch="1";
ch=getchar();
printf("Pressed button = %c\n\r",ch);
while (1)
{
   /* USER CODE END WHILE */
   /* USER CODE BEGIN 3 */
   button = HAL_GPIO_ReadPin(B1_i_GPIO_Port,B1_i_Pin);
   if (button)
      delay=100;
   else
      delay=500;
   if (ch=='A')
      delay=1000;
   if (ch=='B')
      delay=2000;
```

...

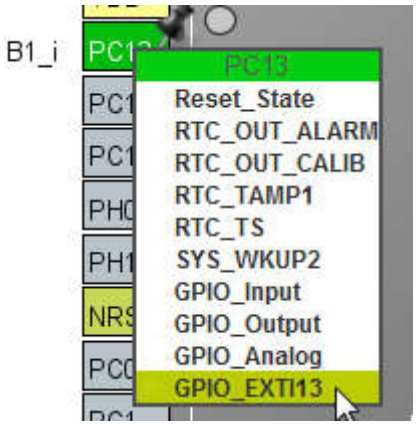


Użycie przerwania do odczytu stanu przycisku

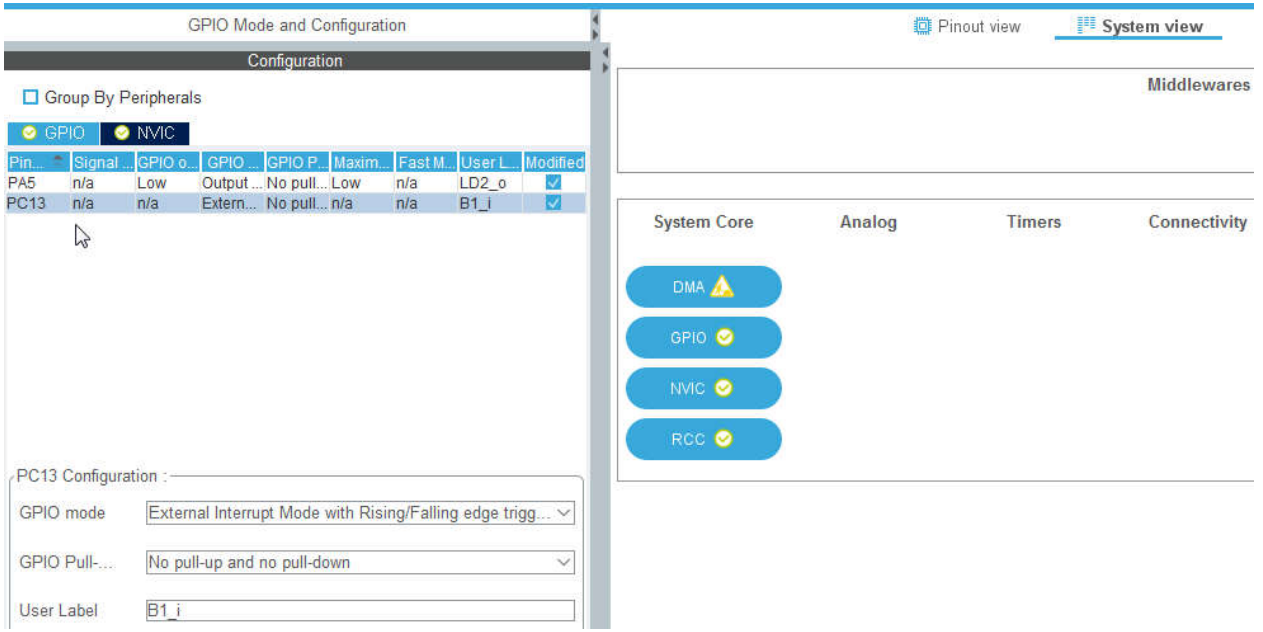
(1) Aktywacja przerwania na I/O wejściowym, trzeba pamiętać o nadaniu własnej, łatwej do interpretacji nazwy.

(2) Konfiguracja przerwania aktywnego na oba zbocza (`System view/GPIO`).

(1)



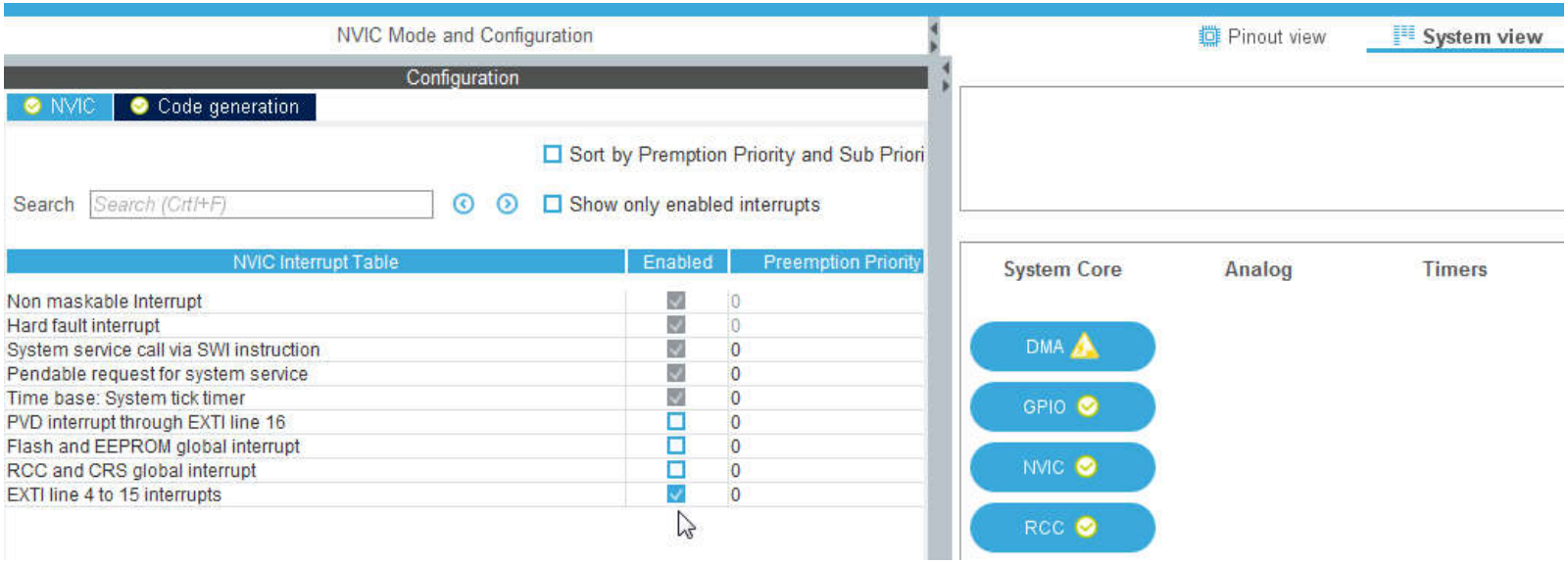
(2)



Użycie przerwania do odczytu stanu przycisku c.d.

(3) Aktywacja przerwania na I/O wejściowym, trzeba pamiętać o nadaniu własnej, łatwej do interpretacji nazwy (menu `System view/NVIC/EXTIline...`

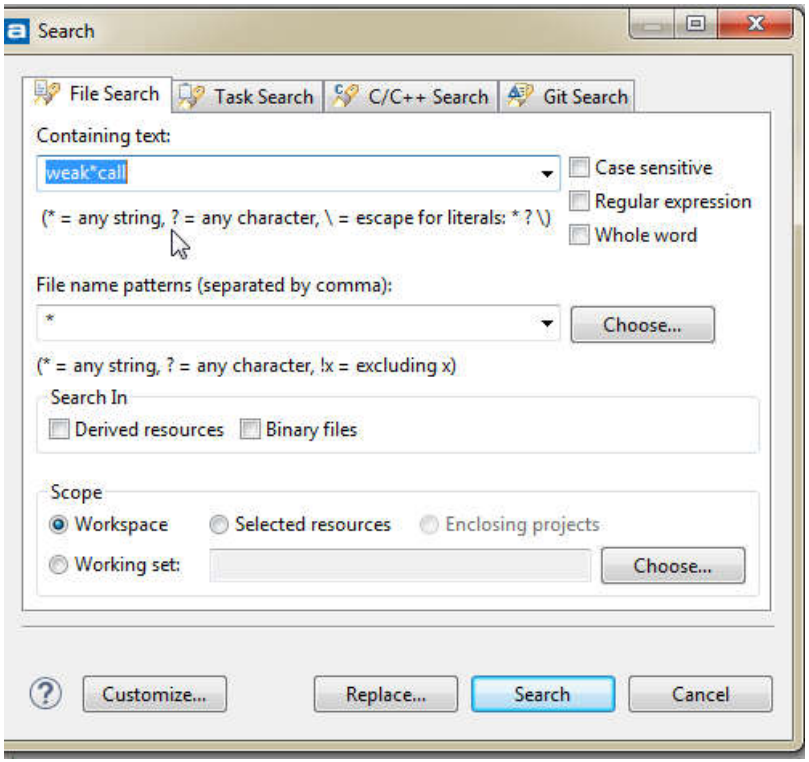
(4) Generacja kodu.

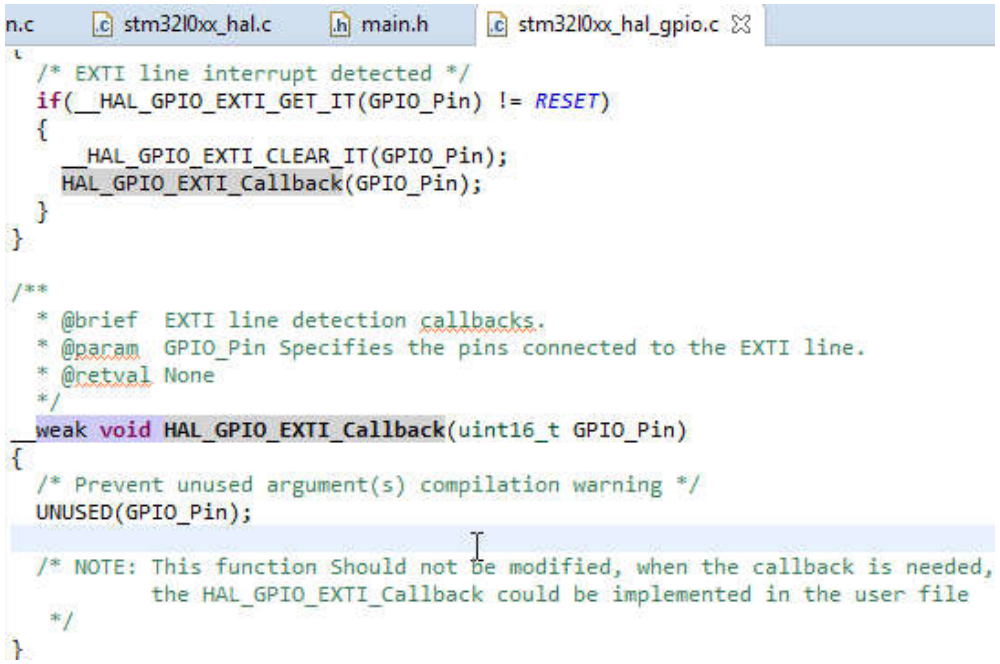


Użycie przerwania do odczytu stanu przycisku c.d.

(5) Wracamy do Atollic True Studio, nowo wygenerowany kod powinien automatycznie się wczytać, ewentualnie wciskamy F5 (Refresh).

(6) Wyszukujemy funkcję obsługi przerwania. Menu `Search/File` i wyszukujemy ciągu „`weak*call`” w pliku obsługi GPIO





Użycie przerwania do odczytu stanu przycisku c.d.

(7) Kopiujemy tą funkcję w odpowiednie miejsce `main.c` i uzupełniamy kodem obsługi, np. jak poniżej:

```
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
/* USER CODE BEGIN PFP */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if (GPIO_Pin==B1_i_Pin)
        if (HAL_GPIO_ReadPin(B1_i_GPIO_Port,B1_i_Pin))
                delay = 500;
        else
                delay = 100;
}
/* USER CODE END PFP */
```

(8) Przenosimy zmienną `delay` do globalnych i modyfikujemy zawartość głównej pętli:

```
  /* USER CODE BEGIN WHILE */
  while (1)
  {
/* USER CODE BEGIN 3 */
        HAL_GPIO_TogglePin(LD2_o_GPIO_Port,LD2_o_Pin);
        HAL_Delay(delay);
  }
  /* USER CODE END 3 */
```

Zadanie do domu

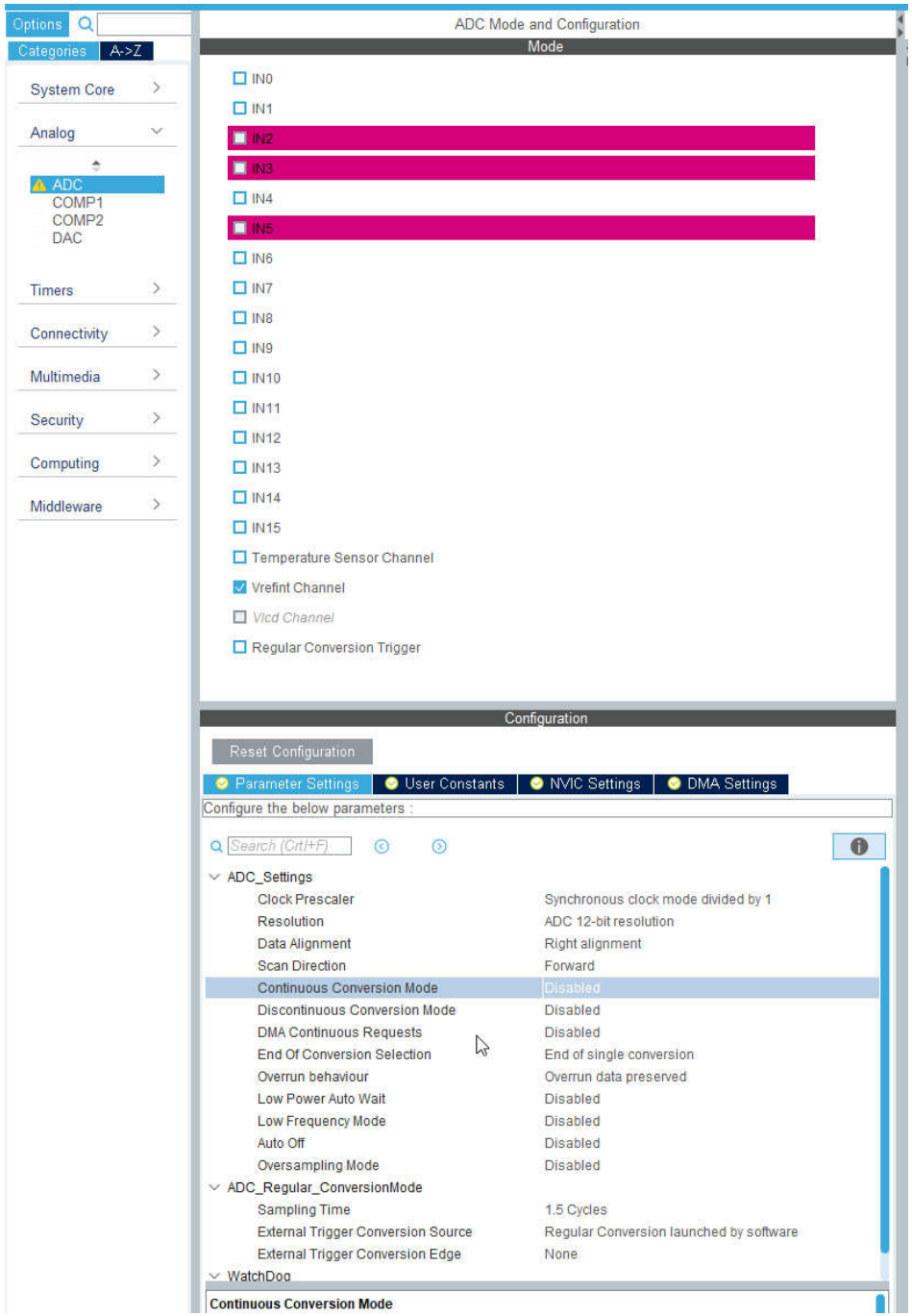
- Zmodyfikuj projekt tak aby realizował funkcję pomiaru czasu wciśnięcia przycisku.
- Wyniki powinny być prezentowane w oknie terminala.
- Prezentowany czas powinien być w [ms].

CHIP
Studenckie Koło Naukowe Politechniki Gdańskiej

Obsługa ADC w poolingu

Cele do osiągnięcia:

1) Skonfigurowanie ADC do pracy pojedynczej z 1 kanałem: `ADC_no_0` lub `Vrefint_channel`.
2) Odczyt wartości ze skonfigurowanego kanału.
3) Interpretacja wyników i prezentacja ich w formie rzeczywistych napięć.



1) Konfiguracja w CubeMX

Obsługa ADC w poolingu c.d.

2) Obsługa w kodzie `main.c`, deklaracja zmiennych globalnych:

```
/* USER CODE BEGIN PV */
uint32_t delay=1000;
uint8_t button=0;
uint16_t ADC_readout=0;
const float Vref=3.3;
float ADC_readoutV = 0;
/* USER CODE END PV */
```

Table 29. Embedded internal reference voltage(1)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$V_{REFINT\ out}$(2)	Internal reference voltage	– 40 °C < T_J < +125 °C	1.202	1.224	1.242	V

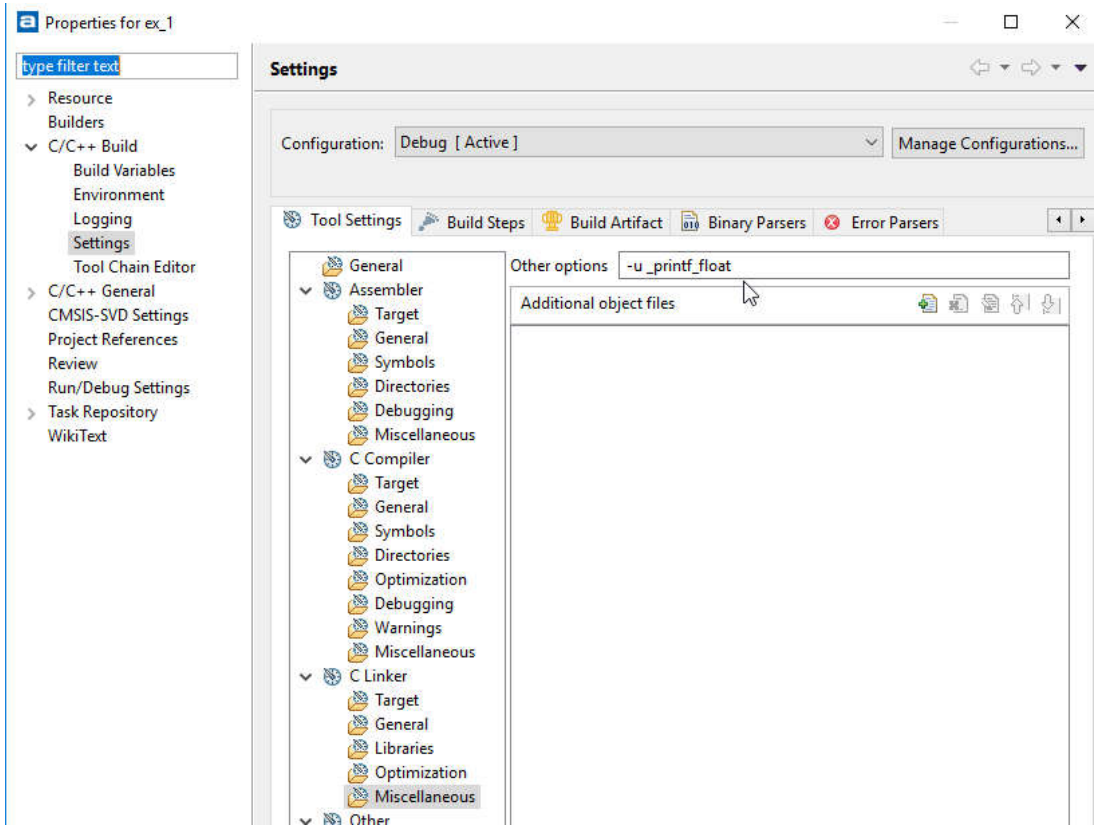
3) Obsługa w kodzie `main.c`, modyfikacja pętli głównej:

```
HAL_ADC_Start(&hadc);
res1 = HAL_ADC_PollForConversion(&hadc,1);
ADC_readout = HAL_ADC_GetValue(&hadc);
res2 = HAL_ADC_PollForConversion(&hadc,1);
ADC_readoutV = Vref*((float)ADC_readout)/4095.0;
printf("ADC_readout = %d, ADC_ReadoutV = %.3f, res1 = %d, res2 =
%d\n\r",ADC_readout,ADC_readoutV,res1,res2);
```

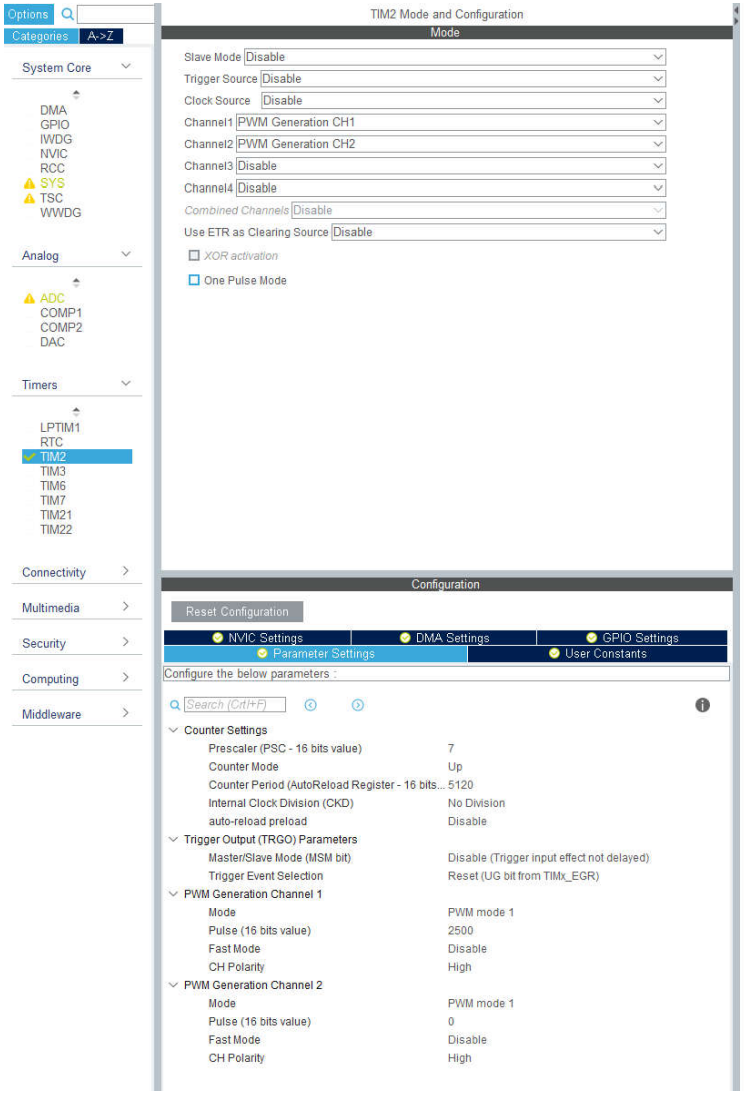
```
Pressed button = 1
delay = 1000ms
ADC_readout = 1557, ADC_ReadoutV = 1.255, res1 = 0, res2 = 3
```

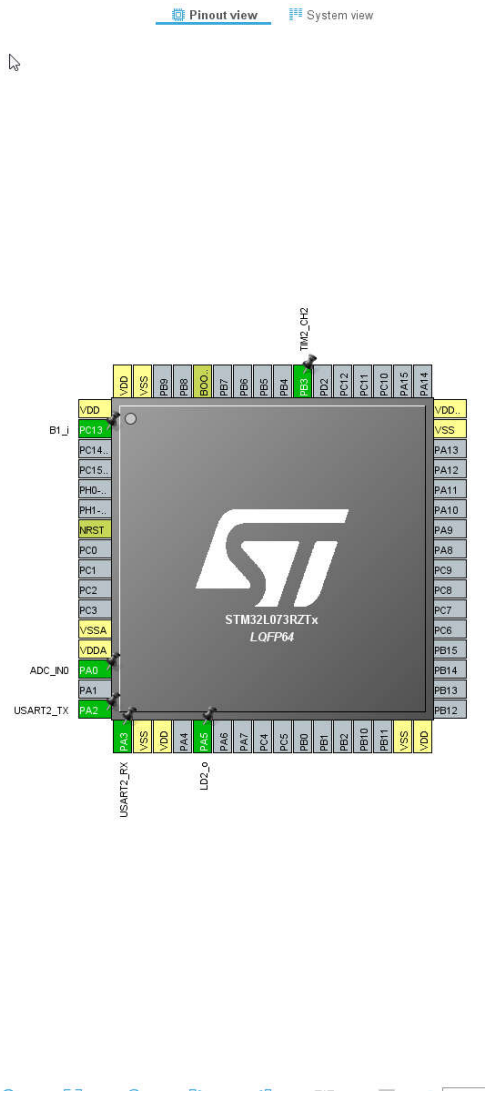
Atollic Studio - konfiguracja wydruku zmiennych `float` przez `printf`, `sprintf`

- Wybieramy menu `Project/Build settings`,
- Wybieramy `C/C++/Settings/C Linker/Miscellaneus` i dopisujemy ciąg: `-u _printf_float`



PWM – użycie licznika do przyciemniania diody LED





1) Konfiguracja w CubeMX

PWM – użycie licznika do przyciemniania diody LED c.d.

2) Obsługa w kodzie `main.c`, deklaracja zmiennych lokalnych i start PWM:

```
uint8_t PWM=10;
uint8_t SERVO=50;
printf("Main loop delay, press: A->100ms, B->200ms, C->500ms, D->1s, E->2s.\n\r");
ch=getchar();
printf("Pressed button = %c\n\r",ch);
if (ch=='A')  delay=100;
if (ch=='B')  delay=200;
HAL_TIM_PWM_Start(&htim2,TIM_CHANNEL_1);
HAL_TIM_PWM_Start(&htim2,TIM_CHANNEL_2);
```

3) Obsługa w kodzie `main.c`, modyfikacja pętli głównej:

```
// SERVO = ADC_readout*100/4095;
TIM2->CCR1=PWM*512/10;
printf("SERVO = %d\n\r",SERVO);
TIM2->CCR2=262+(SERVO*26)/10;
PWM=PWM+10;
if (PWM>=100)  PWM=0;
SERVO=SERVO+1;
if (SERVO>=100)  SERVO=0;
```