Sterowniki robotów

Laboratorium – VS Code

Przygotowanie i praca w środowisku VS Code

Wojciech Domski

Dokument `sr_lab_vscode.pdf`, Wersja 1.0.0

Spis treści

1 Mikrokontrolery STM32

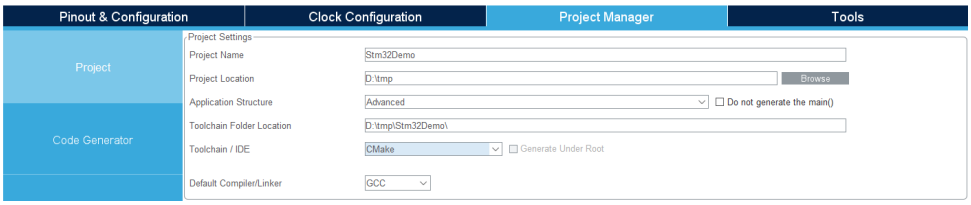
W celu pracy z mikrokontrolerami STM32 w środowisku VS Code wymagane jest następujące dodatkowe oprogramowanie:

- `STM32CubeCLT` – zestaw narzędzi od ST służących do budowy projektów, zawiera on ARM Toolchain oraz inne niezbędne aplikacje jak ST-Link,
- `STM32CubeMX` – graficzny konfigurator mikrokontrolera (jest on częścią STM32CubeIDE, ale również dostępny osobno),
- `stmicroelectronics.stm32-vscode-extension` – wtyczka pozwalająca na konfigurację projektu dla mikrokontrolerów STM32,
- `marus25.cortex-debug` – wtyczka pozwalająca na debugowanie projektów,
- `mcu-debug.peripheral-viewer` – wtyczka pozwalająca na wygodną wizualizację stanu peryferiów mikrokontrolera (opcjonalne),
- `ms-vscode.cmake-tools` – wtyczka zapewniająca rozszerzone wsparcie dla systemu CMake (opcjonalne, ale zalecane).

1.1 Generowanie projektu w STM32CubeMX

Wygeneruj nowy projekt w STM32CubeMX. Pamiętaj o tym, aby wybrać płytkę, a nie sam mikrokontroler i zainicjalizować peryferia. Ułatwi to późniejszą pracę.

Podczas tworzenia projektu wybierz `CMake` jako system budowania (Rysunek 1).

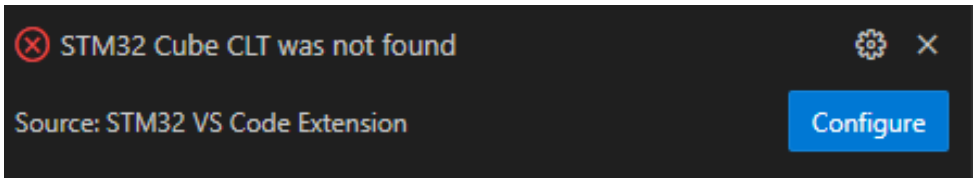


Rysunek 1: Wybór CMake jako systemu budowania w STM32CubeMX

1.2 Importowanie projektu w VS Code

Przy wykorzystaniu wtyczki STM32Cube w programie VS Code możliwe jest uruchomienie aplikacji STM32CubeMX i wygenerowanie projektu. Kroki związane z generowaniem projektu są takie jak w rozdziale 1.1. Po wygenerowaniu projektu można przystąpić do dalszych czynności jakimi są importowanie projektu oraz jego budowa.

Jeżeli STM32CubeCLT nie został znaleziony wcześniej należy wskazać jego lokalizację (Rys. 2).



Rysunek 2: Wskazanie lokalizacji STM32CubeCLT

Aby zaimportować projekt do środowiska VS Code wystarczy otworzyć katalog zawierający wygenerowany uprzednio projekt.

Po zaimportowaniu projektu można przystąpić do jego budowy. Możliwe jest to z poziomu aplikacji VS Code lub z poziomu terminala.

W przypadku VS Code budowanie projektu odbywa się przez skrót klawiaturowy F7. W tym momencie możliwe jest, że środowisko poprosi o wybranie konfiguracji budowania. W tym przypadku zalecane jest wybranie konfiguracji `Debug`.

Innym sposobem na zbudowanie projektu jest wykorzystanie terminala. W tym celu należy otworzyć terminal w katalogu głównym projektu i wykonać następujące polecenia:

```
$ cmake --preset Debug
Preset CMake variables:

  CMAKE_BUILD_TYPE="Debug"
  CMAKE_TOOLCHAIN_FILE:FILEPATH="D:/tmp/Stm32Demo/cmake/gcc-arm-none-eabi.cmake"

-- The C compiler identification is GNU 13.3.1
-- The CXX compiler identification is GNU 13.3.1
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/ST/STM32CubeCLT_1.19.0/GNU-tools-for-STM32/bin/arm-none-eabi-gcc.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/ST/STM32CubeCLT_1.19.0/GNU-tools-for-STM32/bin/arm-none-eabi-g++.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
Build type: Debug
-- The ASM compiler identification is GNU
-- Found assembler: C:/ST/STM32CubeCLT_1.19.0/GNU-tools-for-STM32/bin/arm-none-eabi-gcc.exe
-- Configuring done (3.5s)
-- Generating done (0.0s)
-- Build files have been written to: D:/tmp/Stm32Demo/build/Debug
```

Następnie należy zbudować projekt poleceniem:

```
$ cmake --build --preset Debug
[27/27] Linking C executable Stm32Demo.elf
Memory region         Used Size  Region Size  %age Used
             RAM:        1720 B        96 KB      1.75%
            RAM2:           0 GB        32 KB      0.00%
           FLASH:       12336 B         1 MB      1.18%
```

Podczas budowania projektu mogą pojawić się pewne problemy, szczególnie wtedy, gdy na komputerze zainstalowanych jest wiele wersji ARM Toolchain, czy generatorów takich jak Ninja.

Przykładowo, aby wymusić użycie konkretnego generatora Ninja można wykonać polecenie:

```
cmake --preset Debug -DCMAKE_MAKE_PROGRAM=/c/ST/STM32CubeCLT_1.15.0/Ninja/bin/ninja.exe
```

Jeżeli pojawią się problemy z wykorzystaniem niewłaściwego kompilatora z toolchaina ARM, możliwe jest wymuszenie priorytetowej ścieżki, gdzie będzie on poszukiwany. Wszak pakiet STM32CubeCLT jest dystrybuowany jako samodzielny zestaw narzędzi, który między innymi zawiera kompilator ARM. W tym celu należy wykonać polecenie:

```
PATH=/C/ST/STM32CubeCLT_1.15.0/GNU-tools-for-STM32/bin/:$PATH cmake --preset Debug
```

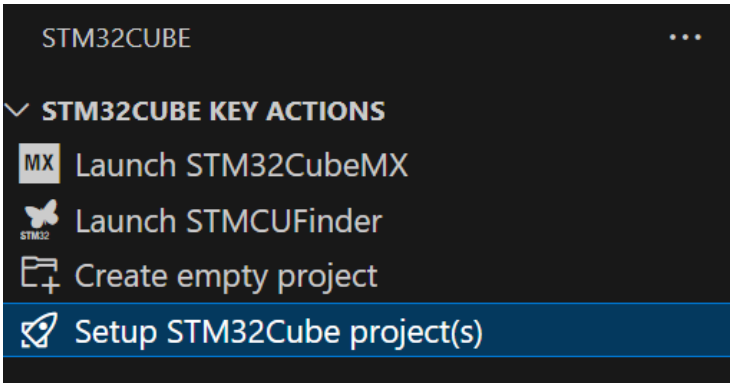
1.3 Debugowanie

Aby przystąpić do debugowania projektu należy utworzyć plik `launch.json` jeżeli jeszcze nie istnieje. Wewnątrz tego pliku należy umieścić następującą konfigurację:

```
{
  "cwd": "${workspaceFolder}",
  "executable": "./build/Debug/Stm32Demo.elf",
  "name": "Debug with ST-Link",
  "request": "launch",
  "type": "cortex-debug",
  "runToEntryPoint": "main",
  "showDevDebugOutput": "none",
  "servertype": "stlink"
}
```

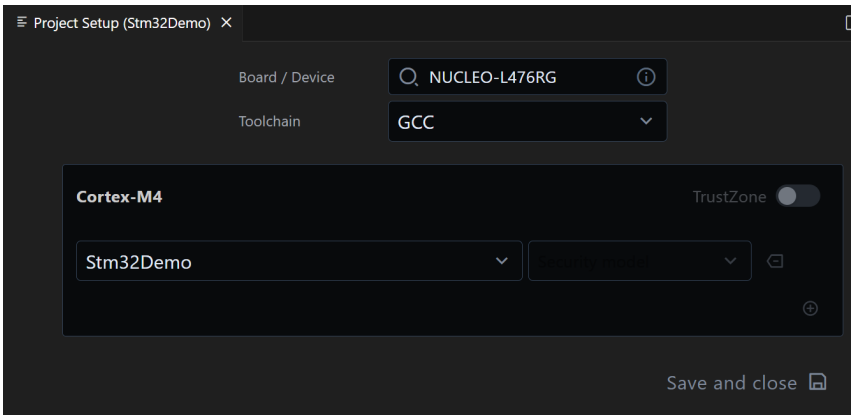
Szczególną uwagę należy zwrócić na pole `executable`, które powinno wskazywać na plik wykonywalny projektu. jeżeli jest to konieczne wskazana jest modyfikacja ścieżki.

Dodatkowo, jeżeli chcemy skorzystać z generatora konfiguracji dla mikrokontrolerów STM32, który jest dostarczony wraz z wtyczką STM32Cube należy uprzednio skonfigurować projekt (Rys. 3).



Rysunek 3: Konfiguracja projektu STM32 w VS Code

Wówczas pojawi się konfigurator (Rys. 4), w którym należy wybrać odpowiedni mikrokontroler/płytkę i toolchain (GCC). Wykonanie takiej konfiguracji zmodyfikuje działanie projektu, jednak ręczna jego kompilacja będzie nadal możliwa.



Rysunek 4: Ustawienia konfiguracji projektu STM32 w VS Code

Wówczas wygenerowane konfiguracje debugowania jak np. *STM32Cube: STM32 Launch ST-Link GDB Server* będą działać bez zarzutu. Przykład takiej konfiguracji znajduje się poniżej:

```
{
  "type": "stlinkgdbtarget",
  "request": "launch",
  "name": "STM32Cube: STM32 Launch ST-Link GDB Server",
  "origin": "snippet",
  "cwd": "${workspaceFolder}",
  "preBuild": "${command:st-stm32-ide-debug-launch.build}",
  "runEntry": "main",
  "imagesAndSymbols": [
    {
      "imageFileName": "${command:st-stm32-ide-debug-launch.get-projects
          -binary-from-context1}"
    }
  ]
}
```

1.4 Zdalne debugowanie

Zdalne debugowanie mikrokontrolera STM32 jest bardzo podobne do „normalnego” debugowania. Różnicap olega na tym, że serwer debugowania musi być dostępny pod wskazanym portem. W tym celu należy wykorzystać infrastrykturę RemoteLab, wybrać odpowiednią płytkę i numer portu do niej

przypisany. Po zidentyfikowaniu płytki należy przekierować port zdalny na port lokalny wykorzystując tunelowanie SSH.

Kolejnym krokiem jest dodanie konfiguracji debugowania w pliku launch.json.

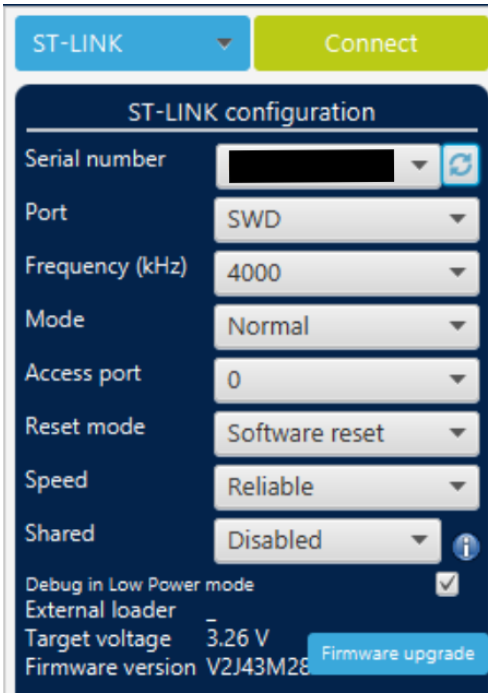
Przykładowa konfiguracja zdalnego debugowania może wyglądać następująco:

```
{
  "name": "Launch Program remote 3010",
  "cwd": "${workspaceRoot}",
  "executable": "./build/Debug/Stm32Demo.elf",
  "request": "launch",
  "type": "cortex-debug",
  "servertype": "external",
  "gdbPath" : "arm-none-eabi-gdb",
  "gdbTarget": "127.0.0.1:3010",
  "device": "STM32L476RGTx",
  "svdFile": "c:/ST/STM32CubeCLT_1.18.0/STMicroelectronics_CMSIS_SVD/
     STM32L476.svd",
  "runToEntryPoint": "main",
  "showDevDebugOutput": "raw"
}
```

Jak można zauważyć występują tutaj dodatkowe pola klucz-wartość, jak np. `svdFile`. To pole służy do określenia pliku SVD, który jest wykorzystywany przez wtyczkę *Peripheral Viewer*. Pozwala ona na podgląd rejestrów peryferiów mikrokontrolera.

1.5 Uwagi końcowe

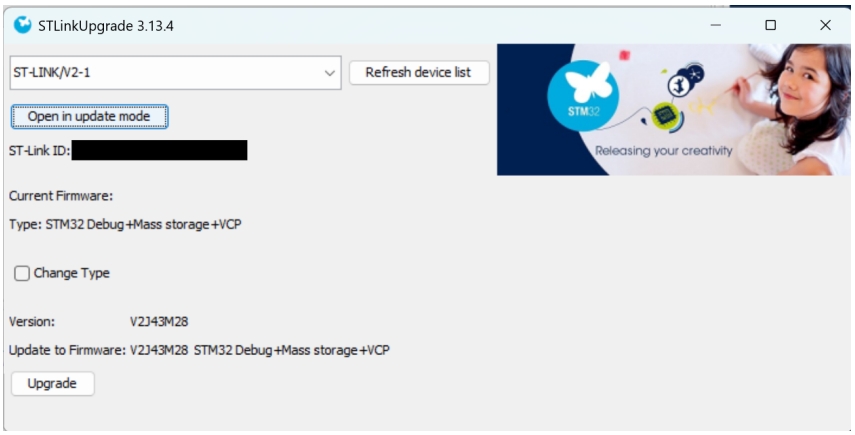
Może się zdarzyć, że uruchomienie debugowania zakończy się niepowodzeniem. Wówczas należy zapoznać się z wyjściem wygenerowanym w terminalu i rozwiązać problem. Jednakże, jedną z najczęstszych przyczyn problemów z debugowaniem mikrokontrolera STM32 w normalnej konfiguracji jest przestarzała wersja oprogramowania debugera ST-Link. Aby zaktualizować oprogramowanie dla debugera można wykorzystać aplikację STM32CubeProgrammer, która pozwala na aktualizację oprogramowania układowego (Rys. 5).



Rysunek 5: Aktualizacja oprogramowania układowego ST-Link w STM32CubeProgrammer

Po wybraniu opcji *Firmware Upgrade* otworzy się aplikacja umożliwiająca aktualizację oprogramo-

wania układowego debugera ST-Link. Na rysunku 6 przedstawiono proces aktualizacji oprogramowania układowego ST-Link.



Rysunek 6: Proces aktualizacji oprogramowania układowego ST-Link

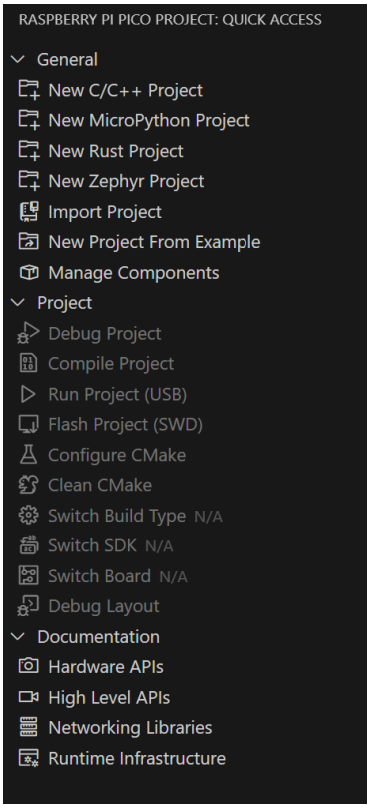
2 Mikrokontrolery Raspberry Pi Pico

W celu pracy z mikrokontrolerami STM32 w środowisku VS Code wymagane jest następujące dodatkowe oprogramowanie:

- `raspberry-pi.raspberry-pi-pico` – wtyczka pozwalająca na konfigurację projektu dla mikrokontrolerów Raspberry Pi.
- `marus25.cortex-debug` – wtyczka pozwalająca na debugowanie projektów,
- `mcu-debug.peripheral-viewer` – wtyczka pozwalająca na wygodną wizualizację stanu peryferiów mikrokontrolera (opcjonalne),
- `ms-vscode.cmake-tools` – wtyczka zapewniająca rozszerzone wsparcie dla systemu CMake.

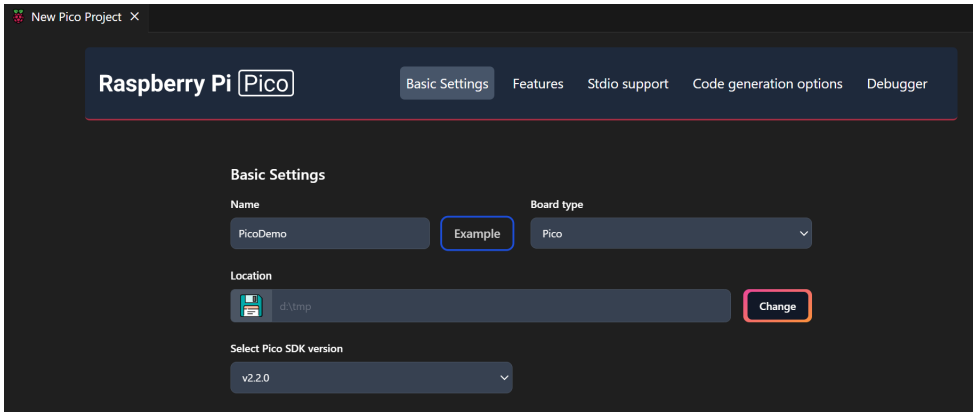
2.1 Generowanie projektu

Nowy projekt dla mikrokontrolera Raspberry Pi Pico można wygenerować za pomocą wtyczki Raspberry Pi Pico w VS Code (Rys. 7).



Rysunek 7: Panel wtyczki Raspberry Pi Pico w VS Code

Z panelu należy wybrać akcję *Create C/C++ Project*, a następnie wskazać katalog, w którym projekt ma zostać utworzony oraz nazwę projektu (Rys. 8).

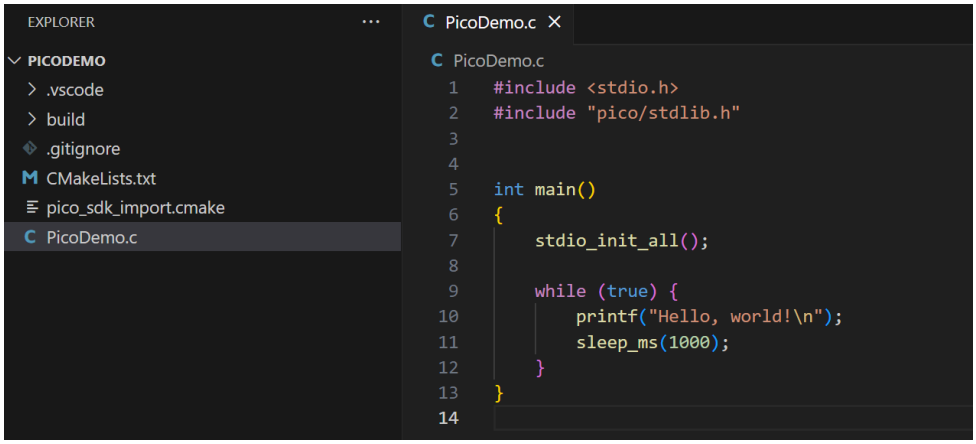


Rysunek 8: Tworzenie nowego projektu Raspberry Pi Pico w VS Code

Jest to moment, w którym nalezy wybrać rodzaj płytki Pico/Pico 2 lub inna oraz niezbędne dodatkowe biblioteki, które umożliwą korzystanie z peryferiów. Po konfiguracji, na samym dole panelu należy wybrać przycisk *Create*.

Procedura to powoduje stworzenie nowego projektu, ale nie tylko. Wraz z tworzeniem nowego projektu zostaną popbrane niezbędne narzedzia, a w tym Pico SDK.

Po utworzeniu projektu zostanie on automatycznie otworzony w VS Code i będzie gotowy do dalszej pracy. Jeśli to nie nastąpi, należy ręcznie otworzyć projekt w programie VS Code. Wówczas drzewo projektu powinno przypominać to pokazane na rysunku 9.



Rysunek 9: Struktura katalogów nowo utworzonego projektu Raspberry Pi Pico w VS Code

2.2 Budowa projektu

Projekt można zbudować na dwa sposoby: z poziomu aplikacji VS Code lub z poziomu terminala.

W przypadku budowy z poziomu VS Code należy skorzystać ze skrótu klawiaturowego F7. Wówczas może pojawić się okno z wyborem konfiguracji budowania – w tym przypadku należy wskazać *Pico*.

W przypadku budowania z poziomu terminala należy wykonać polecenie:

```
$ cmake -S . -B build -G "Ninja"
```

Czasem koniecznym jest wybranie generatora Ninja ręcznie, dlatego został on tutaj podany explicite. Mogą wystąpić również pdoobne problemy jak przy kompilacji projektu dla STM32. Jeśli takie wystąpią należy rozwiązać je w podobny sposób jak opisano to wcześniej.

Następnie należy zbudować projekt poleceniem:

```
$ cmake --build build
```

2.3 Debugowanie

Aby przystąpić do debugowania projektu należy utworzyć plik launch.json jeżeli jeszcze nie istnieje. Wewnątrz tego pliku należy umieścić następującą konfigurację:

```
{
  "name": "Pico Debug (Cortex-Debug)",
  "cwd": "${userHome}/.pico-sdk/openocd/0.12.0+dev/scripts",
  "executable": "${command:raspberry-pi-pico.launchTargetPath}",
  "request": "launch",
  "type": "cortex-debug",
  "servertype": "openocd",
  "serverpath": "${userHome}/.pico-sdk/openocd/0.12.0+dev/openocd.exe",
  "gdbPath": "${command:raspberry-pi-pico.getGDBPath}",
  "device": "${command:raspberry-pi-pico.getChipUppercase}",
  "configFiles": [
    "interface/cmsis-dap.cfg",
    "target/${command:raspberry-pi-pico.getTarget}.cfg"
  ],
  "svdFile": "${userHome}/.pico-sdk/sdk/2.2.0/src/${command:raspberry-pi
     -pico.getChip}/hardware_regs/${command:raspberry-pi-pico.
     getChipUppercase}.svd",
  "runToEntryPoint": "main",
  // Fix for no_flash binaries, where monitor reset halt doesn't do what
      is expected
  // Also works fine for flash binaries
  "overrideLaunchCommands": [
    "monitor reset init",
```

```
    "load \"${command:raspberry-pi-pico.launchTargetPath}\""
  ],
  "openOCDLaunchCommands": [
    "adapter speed 5000"
  ]
}
```

2.4 Zdalne debugowanie

Zdalne debugowanie mikrokontrolera Raspberry Pi Pico jest bardzo podobne do „normalnego" debugowania. Różnica polega na tym, że serwer debugowania musi być dostępny pod wskazanym portem. Czyli podobnie jak dla STM32 należy wykorzystać infrastrykturę RemoteLab, wybrać odpowiednią płytkę i numer portu do niej przypisany. Po zidentyfikowaniu płytki należy przekierować port zdalny na port lokalny wykorzystując tunelowanie SSH.

Kolejnym krokiem jest dodanie konfiguracji debugowania w pliku launch.json. Przykładowa konfiguracja zdalnego debugowania może wyglądać następująco:

```
{
  "name": "Pico Debug (Cortex-Debug with external OpenOCD)",
  "cwd": "${workspaceRoot}",
  "executable": "${command:raspberry-pi-pico.launchTargetPath}",
  "request": "launch",
  "type": "cortex-debug",
  "servertype": "external",
  "gdbTarget": "localhost:3117",
  "gdbPath": "${command:raspberry-pi-pico.getGDBPath}",
  "device": "${command:raspberry-pi-pico.getChipUppercase}",
  "svdFile": "${userHome}/.pico-sdk/sdk/2.2.0/src/${command:raspberry-pi
     -pico.getChip}/hardware_regs/${command:raspberry-pi-pico.
     getChipUppercase}.svd",
  "runToEntryPoint": "main",
  // Fix for no_flash binaries, where monitor reset halt doesn't do what
      is expected
  // Also works fine for flash binaries
  "overrideLaunchCommands": [
    "monitor reset init",
    "load \"${command:raspberry-pi-pico.launchTargetPath}\""
  ]
}
```

2.5 Uwagi końcowe

W przypadku projektów wykorzystujących mikrokontrolery Raspberry Pi Pico bardzo istotne jest odpowiednie skonfigurowanie pliku CMakeLists.txt.

W przypadku przekierowania standardowego wyjścia (funkcja `printf`) wymagane jest wybranie odpowiedniego interfejsu. Aby to uczynić należy odnaleźć linie:

```
# Modify the below lines to enable/disable output over UART/USB
pico_enable_stdio_uart(PicoDemo 0)
pico_enable_stdio_usb(PicoDemo 0)
```

i zmienić wartości 0 na 1 w zależności od wybranego interfejsu, dla infrastruktury RemoteLab zalecane jest wykorzystanie UARTa. zatem modyfikacja powinna wyglądać następująco:

```
# Modify the below lines to enable/disable output over UART/USB
pico_enable_stdio_uart(PicoDemo 1)
pico_enable_stdio_usb(PicoDemo 0)
```

W przypadku wykorzystania peryferiów mikrokontrolera należy pamiętać o dodaniu odpowiednich bibliotek do projektu w pliku CMakeLists.txt. Przykładowo chcąc skorzystać z ADC i PWM należy odnaleźć linie:

```
# Add the standard library to the build
target_link_libraries(PicoDemo
)
```

i je zmodyfikować w następujący sposób:

```
  # Add the standard library to the build
target_link_libraries(PicoDemo
        pico_stdlib hardware_adc hardware_pwm)
```

Każda modyfikacja pliku CMakeLists.txt wymaga ponownego zbudowania projektu.

