Migrating Mbed Projects to Visual Studio Code (VS Code) and Other Open Source Platforms

Abstract

The impending end-of-life (EOL) for Mbed OS necessitates the migration of existing Mbed projects to alternative development environments. This paper provides a comprehensive guide for migrating Mbed projects to Visual Studio Code (VS Code) and other open-source platforms such as PlatformIO, Eclipse with GNU MCU Eclipse Plugins, and STM32CubeIDE. Each platform's installation, configuration, and common migration issues are discussed in detail.

Introduction

Mbed OS has been a popular choice for embedded development on ARM Cortex-M microcontrollers. However, with its EOL, developers must seek new environments to continue their projects. This paper explores various open-source platforms, including Visual Studio Code (VS Code), PlatformIO, Eclipse with GNU MCU Eclipse Plugins, and STM32CubeIDE, to facilitate the migration process. Each platform offers unique features and benefits, and this paper provides a structured approach to transitioning Mbed projects to these environments.

Benefits of Migrating to Open Source Platforms

Versatility

Open-source platforms support a broad spectrum of programming languages and development environments, making them flexible choices for various projects.

Extensibility

A vast library of extensions and plugins is available to enhance functionality, including support for embedded development, debugging, and code analysis.

Community Support

The robust communities surrounding open-source platforms ensure extensive resources, tutorials, and support, aiding developers in troubleshooting and enhancing their projects.

Prerequisites

Before initiating the migration, ensure the following tools are installed:

- **Visual Studio Code**: Download and install from Visual Studio Code.
- **PlatformIO**: Install from PlatformIO.
- **Eclipse with GNU MCU Eclipse Plugins**: Download and install from GNU MCU Eclipse.
- **STM32CubeIDE**: Download and install from STM32CubeIDE.
- **ARM GCC Toolchain**: Necessary for compiling ARM Cortex-M microcontrollers.
- **CMake**: Required for managing the build process.

Installing ARM GCC Toolchain

Windows

1. **Download the ARM GCC Toolchain**:
   - Visit the ARM Developer website and download the latest version for Windows.
2. **Install the Toolchain**:
   - Run the installer and follow the instructions. By default, it installs to `C:\Program Files (x86)\GNU Tools Arm Embedded\`.
3. **Add to PATH**:
   - Add the bin directory of the installed toolchain to your system's PATH environment variable. For example: `C:\Program Files (x86)\GNU Tools Arm Embedded\8 2019-q3-update\bin`.

macOS

1. **Download the ARM GCC Toolchain**:
   - Visit the ARM Developer website and download the latest version for macOS.
2. **Install the Toolchain**:
   - Extract the downloaded file and move the directory to a desired location, such as `/usr/local/gcc-arm`.
3. **Add to PATH**:

- Add the bin directory to your PATH environment variable by adding the following line to your `.bashrc`, `.zshrc`, or `.profile` file:

  ```
  export PATH=/usr/local/gcc-arm/bin:$PATH
  ```

Linux

1. **Download the ARM GCC Toolchain**:
   - Visit the ARM Developer website and download the latest version for Linux.
2. **Install the Toolchain**:
   - Extract the downloaded file and move the directory to a desired location, such as `/opt/gcc-arm`.
3. **Add to PATH**:
   - Add the bin directory to your PATH environment variable by adding the following line to your `.bashrc`, `.zshrc`, or `.profile` file:

     ```
     export PATH=/opt/gcc-arm/bin:$PATH
     ```

Migration Steps for Visual Studio Code

Step 1: Install Required Extensions

Install the following extensions in VS Code to facilitate embedded development:

- **C/C++**: Microsoft C/C++ extension for IntelliSense, debugging, and code browsing.
- **Cortex-Debug**: Extension for debugging ARM Cortex-M microcontrollers.
- **CMake Tools**: Extension for CMake integration.

Step 2: Set Up the Project Directory

Organize your project directory to align with VS Code's expectations. Move your Mbed source files to a suitable directory and create necessary configuration files.

```
your_project/
├── .vscode
│   └── settings.json
│   └── launch.json
│   └── c_cpp_properties.json
├── include
│   └── some_header.h
├── lib
│   └── some_library
│       └── some_library.cpp
├── src
│   └── main.cpp
├── CMakeLists.txt
└── mbed-os
```

Step 3: Create `CMakeLists.txt`

Create a `CMakeLists.txt` file to configure the build process. Below is an example configuration:

```
cmake_minimum_required(VERSION 3.13)
set(CMAKE_SYSTEM_NAME Generic)
set(CMAKE_SYSTEM_PROCESSOR arm)

# Specify the cross compiler
set(CMAKE_C_COMPILER arm-none-eabi-gcc)
set(CMAKE_CXX_COMPILER arm-none-eabi-g++)

# Set the project name
project(MbedProject)

# Include the Mbed OS directory
include_directories(${CMAKE_SOURCE_DIR}/mbed-os)

# Add source files
file(GLOB_RECURSE SOURCES "src/*.cpp" "lib/*.cpp")

# Create the executable
add_executable(${PROJECT_NAME} ${SOURCES})
```

Step 4: Configure VS Code Settings

Create the `.vscode` directory and add the following configuration files:

`settings.json`

```json
{
    "cmake.generator": "Unix Makefiles"
}
```

`launch.json`

```json
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Cortex Debug",
            "type": "cortex-debug",
            "request": "launch",
            "servertype": "jlink",
            "device": "STM32F401RE",
            "interface": "swd",
            "runToMain": true,
            "cwd": "${workspaceRoot}",
            "executable": "${workspaceRoot}/build/MbedProject.elf"
        }
    ]
}
```

`c_cpp_properties.json`

```json
{
    "configurations": [
        {
            "name": "ARM",
            "includePath": [
                "${workspaceFolder}/**",
                "${workspaceFolder}/mbed-os/**"
            ],
            "defines": [],
            "compilerPath": "/usr/bin/arm-none-eabi-gcc",
            "cStandard": "c11",
            "cppStandard": "c++17",
            "intelliSenseMode": "gcc-arm"
        }
    ],
    "version": 4
}
```

Step 5: Build and Debug

Use the CMake Tools extension to configure and build your project:

1. Open the Command Palette (`Ctrl+Shift+P`) and run `CMake: Configure` .
2. Run `CMake: Build` to compile your project.
3. Use the Cortex-Debug extension to debug your project. Set breakpoints and start a debug session using `F5` .

Migration Steps for PlatformIO

Step 1: Install PlatformIO

Install PlatformIO in your preferred development environment. For example, you can install PlatformIO as an extension in Visual Studio Code.

Step 2: Initialize a PlatformIO Project

Navigate to your project directory and initialize a PlatformIO project. Replace `<your_board>` with the specific board you are using.

```
platformio init --board <your_board>
```

Step 3: Move Project Files

Move your Mbed source files to the `src` directory and any libraries to the `lib` directory. Ensure your project structure aligns with PlatformIO's expectations:

```
your_project/
├── include
│   └── some_header.h
├── lib
│   └── some_library
│       └── some_library.cpp
├── src
│   └── main.cpp
└── platformio.ini
```

Step 4: Configure `platformio.ini`

Create and configure the `platformio.ini` file. Below is an example configuration for an STM32 Nucleo board:

```
[env:nucleo_f401re]
platform = ststm32
board = nucleo_f401re
framework = mbed
build_flags = -DMBED_CONF_PLATFORM_STDIO_BAUD_RATE=115200
monitor_speed = 115200
```

Step 5: Build and Upload

Use PlatformIO to build and upload your project:

```
platformio run
platformio run --target upload
```

Migration Steps for Eclipse with GNU MCU Eclipse Plugins

Step 1: Install Eclipse and GNU MCU Eclipse Plugins

Download and install Eclipse IDE, then install the GNU MCU Eclipse plugins from GNU MCU Eclipse.

Step 2: Create a New Project

Create a new C/C++ project in Eclipse and configure it for your specific microcontroller.

Step 3: Import Mbed Source Files

Import your Mbed source files into the Eclipse project. Place them in appropriate directories such as `src` and `include` .

Step 4: Configure Build Settings

Configure the build settings in Eclipse to use the ARM GCC toolchain. Set include paths and linker settings as necessary.

Step 5: Build and Debug

Build your project using Eclipse's build system. Debug using the GNU MCU Eclipse debugging tools.

Migration Steps for STM32CubeIDE

Step 1: Install STM32CubeIDE

Download and install STM32CubeIDE from STM32CubeIDE.

Step 2: Create a New Project

Create a new STM32 project in STM32CubeIDE and select your specific STM32 microcontroller.

Step 3: Import Mbed Source Files

Import your Mbed source files into the STM32CubeIDE project. Place them in appropriate directories such as `Src` and `Inc` .

Step 4: Configure Build Settings

Configure the build settings in STM32CubeIDE to use the ARM GCC toolchain. Set include paths and linker settings as necessary.

Step 5: Build and Debug

Build your project using STM32CubeIDE's build system. Debug using the integrated debugging tools.

Common Migration Issues

Library Management

Ensure all necessary libraries are included and correctly configured in the new environment.

Build Configuration

Some build settings in Mbed (`mbed_app.json` , `mbed_lib.json`) may need to be translated to the settings of the new environment.

Debugging Setup

Properly configure the debugging settings for your specific microcontroller and debugger interface.

Testing and Validation

Thoroughly test your migrated project to ensure all functionalities work as expected. Implement automated testing to catch any issues early. Validate performance and stability in the new environment.

Additional Resources

- **Visual Studio Code Documentation**: VS Code Docs
- **PlatformIO Documentation**: PlatformIO Docs
- **GNU MCU Eclipse Documentation**: GNU MCU Eclipse Docs
- **STM32CubeIDE Documentation**: STM32CubeIDE Docs
- **ARM GCC Toolchain**: ARM GCC
- **CMake Documentation**: CMake Docs

Conclusion

Migrating from Mbed OS to open-source platforms such as Visual Studio Code, PlatformIO, Eclipse with GNU MCU Eclipse Plugins, and STM32CubeIDE offers several benefits, including versatility, extensibility, and robust development environments. By following the steps outlined in this guide, developers can successfully transition their projects to these platforms and continue development with confidence. The communities and extensive documentation for each platform provide valuable resources for troubleshooting and enhancing projects during and after the migration process.

References

1. Visual Studio Code Documentation. Retrieved from Visual Studio Code Docs.
2. PlatformIO Documentation. Retrieved from PlatformIO Docs.
3. GNU MCU Eclipse Documentation. Retrieved from GNU MCU Eclipse Docs.
4. STM32CubeIDE Documentation. Retrieved from STM32CubeIDE Docs.
5. ARM GCC Toolchain. Retrieved from ARM Developer.
6. CMake Documentation. Retrieved from CMake Docs.