Cross-compiling External C/C++ Libraries using `vcpkg`

A Guide for CMPT433 — Embedded Systems[*]

Group **Server Rack** — *Alexander Li, Puneet Mullhi, Bruce Wang, Chase Shen*

April 14, 2024

1 Introduction

This guide provides step-by-step instructions on how to cross-compile external C/C++ libraries using `vcpkg`. It is intended for developers who are working on embedded systems and need to build libraries for a different target architecture. By following this guide, you will be able to install and configure `vcpkg` for cross-compilation, and build external libraries that are compatible with your target system.

2 Prerequisites

Before you begin, make sure you have the following prerequisites:

- A linux host development system (Our host system is Debian 11.8)
- Basic knowledge for linux system structure and commands
- A target system with a different architecture than your development machine (Our target system is the BeagleBone Green)
- Basic cross-compilation libraries installed on your host system (e.g., `gcc-arm-linux-gnueabihf`)
- Visual Studio Code with CMake Tools installed (Optional. You may use any other IDE or build system, but this guide will only cover Visual Studio Code)

3 Installing vcpkg

To install `vcpkg` on your host system, follow these steps:

1. Install the necessary tools for building `vcpkg`:

```
(host) $ sudo apt-get install build-essential git tar curl
            zip unzip ninja-build
```

2. `cd` to the directory where you want to install `vcpkg` (preferably your home directory)
3. Clone the `vcpkg` repository from GitHub:

```
(host) $ git clone https://github.com/microsoft/vcpkg
```

[*]Copyright permission grated to CMPT 433 instructors.

4. Run the bootstrap script to build `vcpkg`:

   ```
   (host) $ sudo ./vcpkg/bootstrap-vcpkg.sh
   ```

5. Note: If you encounter any errors during the build process, refer to troubleshooting section for possible solutions.
6. If there are no errors present after the bootstrapping process, the installation is successful. Once the build is complete, you can use `vcpkg` to install and build external libraries for your target system.

4 Configuring vcpkg

Once the bootstrapping process is complete, we will have to configure a vcpkg triplet for `arm-linux-gnueabihf` (or any other target system you may have.) A triplet is a cmake configuration file that specifies the target architecture, operating system, and C++ standard library. To configure a triplet for `arm-linux-gnueabihf`, follow these steps:

1. `cd` to the `vcpkg` directory
2. Run the following command to create a new triplet file for `arm-linux-gnueabihf`:

   ```
   (host) $ touch ./triplets/arm-linux-gnueabihf.cmake
   ```

3. This will create a new file named `arm-linux-gnueabihf.cmake` in the `vcpkg/triplets` directory. This file contains the configuration settings for cross-compiling libraries for the `arm-linux-gnueabihf` target system.
4. Put the following content in the `arm-linux-gnueabihf.cmake` file:

   ```
   # vcpkg/triplets/arm-linux-gnueabihf.cmake
   set(VCPKG_TARGET_ARCHITECTURE arm) # Target architecture
   set(VCPKG_CMAKE_SYSTEM_PROCESSOR arm) # Target processor
   set(VCPKG_CRT_LINKAGE dynamic) # Linkage type
   set(VCPKG_LIBRARY_LINKAGE static) # Library linkage type
   set(VCPKG_CMAKE_SYSTEM_NAME Linux) # Target operating system
   ```

5. You can now use this triplet file to build external libraries for your target system using `vcpkg`.

4.1 Integrate vcpkg with Visual Studio Code's CMake Tools Extension

If you are using Visual Studio Code with the CMake Tools extension, you can integrate `vcpkg` with your project to automatically build external libraries for your target system. To do this, follow these steps:

1. Open your project in Visual Studio Code
2. Create a new file named `settings.json` in the `.vscode` directory of your project
3. Add the following content to the `settings.json` file:

```
# .vscode/settings.json
{
    "cmake.configureSettings": {
        "CMAKE_TOOLCHAIN_FILE":
            "<YOUR_VCPKG_ROOT>/scripts/buildsystems/vcpkg.cmake",
        "VCPKG_TARGET_TRIPLET": "arm-linux-gnueabihf",
        "VCPKG_HOST_TRIPLET":
            "arm64-linux", # This would depend on your host system
    }
}
```

4. Note: to find available host triplets, you can list them using the following command:

```
(host) $ ls <YOUR_VCPKG_ROOT>/triplets
```

5. Replace `<YOUR_VCPKG_ROOT>` with the path to your `vcpkg` directory
6. Save the `settings.json` file
7. Note: In theory, you can configure them directly in the `CMakeLists.txt` file, but our team has only tested the `settings.json` method.
8. After configuring the settings, you can use the CMake Tools extension in Visual Studio Code to build your project with external libraries that are cross-compiled for your target system.

5 Installing External Libraries

To install external libraries for cross-compilation using `vcpkg`, follow these steps:

1. `cd` to the directory containing the `vcpkg` directory
2. Run the following command to install a library for the `arm-linux-gnueabihf` target system:

```
(host) $ ./vcpkg/vcpkg install <lib_name>:arm-linux-gnueabihf
```

3. Replace `<lib_name>` with the name of the library you want to install (e.g., `curl`)
4. Note: The installation process may take a long time depending on your system spec and internet connection. If you encounter any errors during the installation process, refer to the troubleshooting section for possible solutions.
5. After you completed all the previous steps, you should have successfully installed the external library for cross-compilation. You can now use the library in your project after properly configuring the CMakeLists.txt file for linkage.

6 Troubleshooting

6.1 CMake Version is Too Old

If you encounter an error message saying that the CMake version is too old, and you cannot upgrade CMake using your package manager, you can download the latest version of CMake from the official website and install it manually. To do this, follow these steps:

1. First, remove the existing CMake installation from your system using the package manager
2. Download the latest version of CMake from the official website: `https://cmake.org/download/`
3. You can choose to download the compressed archive (.tar.gz) or the installer (.sh) for your system, this guide will cover the installer method
4. Move the downloaded installer to the `/usr/local` directory
5. Execute the installer script with root privileges
6. Make sure to install to the default directory (`/usr/local`), do not install to `/usr/local/cmake-<version>` as the installation may not be picked up by the system
7. After the installation is complete, you should have the latest version of CMake installed on your system
8. To verify the installation, run the following command:

   ```
   (host) $ cmake --version
   ```

6.2 Environment variable VCPKG_FORCE_SYSTEM_BINARIES must be set on ...

If you encounter an error message saying that the environment variable `VCPKG_FORCE_SYSTEM_BINARIES` must be set, you can set this variable in your terminal before running `vcpkg`. To do this, follow these steps:

1. Run the following command to set the `VCPKG_FORCE_SYSTEM_BINARIES` environment variable:

   ```
   (host) $ export VCPKG_FORCE_SYSTEM_BINARIES=1
   ```

2. After setting the environment variable, you should be able to run `vcpkg` without encountering the error message. If you want to make this change permanent, you can add the export command to your `.bashrc` or `.bash_profile` file.

6.3 Other Issues

If you encounter any other issues during the installation or configuration process, you can refer to the official `vcpkg` documentation (https://learn.microsoft.com/en-us/vcpkg/) for troubleshooting tips and solutions. The `vcpkg` GitHub repository (https://github.com/microsoft/vcpkg) also contains a list of reported issues and their solutions, which may help you resolve any problems you encounter.

7 Closing Note

Our team originally did not intend to use `vcpkg`. However, during our development process, we had an issue cross-compiling the `curl` library for our target system. After researching and experimenting with different solutions, we found that `vcpkg` was the most straightforward and reliable method for cross-compiling external libraries. We hope this guide will help you successfully build and integrate external libraries for your embedded system project.