001

Hello World - Qemu

Operating Systems

Lab001 Hello World

We need to install the arm-none-eabi-gcc compiler. This compiler is essential for cross-compiling code for ARM Cortex-M and Cortex-A processors in a bare-metal environment (without an operating system). Below are detailed instructions to install arm-none-eabi-gcc on macOS, Linux, and Windows.

For macOS

Method 1: Using Homebrew

Homebrew is a popular package manager for macOS that simplifies the installation of software.

```
# Make sure taps & casks are up to date
brew update
brew upgrade --cask

# Install the current Arm GNU Toolchain (recommended)
brew install --cask gcc-arm-embedded
```

Verify the Installation

Check if the compiler is installed correctly:

```
arm-none-eabi-gcc --version
```

You should see output similar to:

```
arm-none-eabi-gcc (GNU Arm Embedded Toolchain ...) ...
```

Method 2: Download Pre-Built Binaries from ARM

Step 1: Download the Toolchain

Visit the official ARM developer website to download the GNU Arm Embedded Toolchain:

- GNU Arm Embedded Toolchain Downloads

Select the macOS version (.pkg file) appropriate for your development needs.

Step 2: Install the Toolchain

- Run the downloaded .pkg installer.
- Follow the installation prompts.

Step 3: Update Your PATH Environment Variable

Add the installation directory to your PATH. Assuming it installs to /usr/local/bin:

```
export PATH="/usr/local/bin:$PATH"
```

You may want to add this line to your ~/.bash_profile, ~/.zshrc, or ~/.bashrc file, depending on your shell.

Step 4: Verify the Installation

```
arm-none-eabi-gcc --version
```

For Linux

Method 1: Using Package Manager (Ubuntu/Debian)

Step 1: Update Package Lists

```
sudo apt-get update
```

Step 2: Install the ARM Cross-Compiler

```
sudo apt-get install gcc-arm-none-eabi gdb-arm-none-eabi
```

Note: On some systems, the package may be named gcc-arm-none-eabi or binutils-arm-none-eabi. Adjust accordingly.

Step 3: Verify the Installation

```
arm-none-eabi-gcc --version
```

Method 2: Download Pre-Built Binaries from ARM

Step 1: Download the Toolchain

- Visit the GNU Arm Embedded Toolchain Downloads.
- Choose the Linux 64-bit tarball (.tar.bz2 file).

Step 2: Extract the Archive

Navigate to the directory where you downloaded the file and run:

```
tar -xjf gcc-arm-none-eabi-*-x86_64-linux.tar.bz2
```

Step 3: Move to an Appropriate Location

```
sudo mv gcc-arm-none-eabi-*/ /opt/gcc-arm-none-eabi
```

Step 4: Update Your PATH Environment Variable

Add the toolchain to your PATH by editing ~/.bashrc or ~/.bash_profile:

```
export PATH="/opt/gcc-arm-none-eabi/bin:$PATH"
```

Reload your shell configuration:

```
source ~/.bashrc
```

Step 5: Verify the Installation

```
arm-none-eabi-gcc --version
```

For Windows

Method 1: Using the Official Installer

Step 1: Download the Installer

- Go to the GNU Arm Embedded Toolchain Downloads.
- Download the Windows version (.exe installer).

Step 2: Run the Installer

- Double-click the downloaded .exe file.
- Follow the installation wizard steps.
- Accept the license agreement.
- Choose the installation directory (e.g., C:\Program Files (x86)\GNU Arm Embedded Toolchain).

Step 3: Add Toolchain to the System PATH

- Open Control Panel > System and Security > System.
- Click on Advanced system settings.
- Click on Environment Variables.
- Under System variables, select Path and click Edit.
- Click New and add the path to the bin directory of your installation, e.g.:

C:\Program Files (x86)\GNU Arm Embedded Toolchain\bin

- Click OK to close all dialogs.

Step 4: Verify the Installation

Open Command Prompt and run:

```
arm-none-eabi-gcc --version
```

Method 2: Using MSYS2

MSYS2 provides a Unix-like environment on Windows and includes package management.

Step 1: Install MSYS2

Download and install MSYS2 from msys2.org.

Step 2: Update Package Database and Core System Packages

Open the MSYS2 MSYS terminal and run:

```
pacman -Syu
```

Close and reopen the MSYS2 terminal, then run:

```
pacman -Su
```

Step 3: Install the ARM Cross-Compiler

```
pacman -S mingw-w64-x86_64-arm-none-eabi-gcc
```

Step 4: Add to PATH (Optional)

Add the following to your Windows PATH environment variable:

```
C:\msys64\mingw64\bin
```

Adjust the path if you installed MSYS2 in a different location.

Step 5: Verify the Installation

Open MSYS2 MinGW 64-bit terminal and run:

```
arm-none-eabi-gcc --version
```

Additional Notes

• Administrator Privileges: Some steps may require administrator or root privileges, especially when installing software or modifying system-wide settings.
• Version Compatibility: Ensure that all components (compiler, debugger, etc.) are compatible in terms of versions.
• Multiple Versions: If you have multiple versions of arm-none-eabi-gcc installed, make sure the correct one is being used by checking your PATH variable.

Emulating ARM Bare-Metal Code with QEMU

While QEMU doesn't have a specific machine model for the BeagleBone Black, you can emulate a similar ARM platform to test your bare-metal code. We'll use the VersatilePB machine provided by QEMU, which is sufficient for running simple ARM code like the "Hello World" example.

Steps to Test the "Hello World" Example on QEMU

1. Install QEMU

First, install QEMU on your development machine.

- **On Ubuntu/Debian:**

```
sudo apt-get update
sudo apt-get install qemu-system-arm
```

- **On macOS (using Homebrew):**

```
brew install qemu
```

Installing QEMU on Windows

There are several methods to install QEMU on Windows:

1. Using the Official Windows Binaries
2. Using MSYS2
3. Using Windows Subsystem for Linux (WSL)

I'll provide detailed steps for each method so you can choose the one that best fits your requirements.

Method 1: Installing Official QEMU Windows Binaries

The QEMU project provides pre-built Windows binaries that you can download and use directly.

Step 1: Download QEMU for Windows

• Visit a Trusted Source: While the official QEMU website doesn't provide Windows binaries, you can download them from reputable sources like:
qemu.weilnetz.de: Offers 64-bit Windows binaries.

Step 2: Download the Installer or Zip File

• Choose the Latest Version: Download either the installer (.exe) or the zip archive (.zip).
Example: qemu-w64-setup-20240816.exe (version and date may vary).

Step 3: Install QEMU

• Using the Installer:
• Run the downloaded .exe file.
• Follow the installation prompts.
• Select the components you wish to install.
• Choose the installation directory (e.g., C:\Program Files\qemu).
• Using the Zip Archive:
• Extract the contents to a directory of your choice (e.g., C:\qemu).

Step 4: Add QEMU to the System PATH

1. Press Win + X and select System.
2. Click on Advanced system settings.
3. In the System Properties window, click Environment Variables.
4. Under System variables, select Path and click Edit.
5. Click New and add the path to QEMU's bin directory (e.g., C:\Program Files\qemu or C:\qemu).
6. Click OK to close all dialogs.

Step 5: Verify the Installation

- Open Command Prompt:
- Press Win + R, type cmd, and press Enter.
- Run:

```
qemu-system-arm --version
```

- You should see output displaying the QEMU version information.

Method 2: Installing QEMU via MSYS2

MSYS2 provides a Unix-like environment on Windows, including a package manager to install software like QEMU.

Step 1: Install MSYS2

- Download the Installer:
- Visit https://www.msys2.org/.
- Download the installer (msys2-x86_64-*.exe).
- Run the Installer:
- Follow the installation prompts.
- Install MSYS2 to the default location (e.g., C:\msys64).

Step 2: Update the Package Database and Core Packages

- Open the MSYS2 MSYS terminal from the Start menu.
- Run the update commands:

```
pacman -Syu
```

- If prompted, press Y to proceed.
- Close and reopen the MSYS2 terminal if instructed.

- Run the update again:

```
pacman -Su
```

Step 3: Install QEMU

- In the MSYS2 terminal, install QEMU:

```
pacman -S mingw-w64-x86_64-qemu
```

- Press Y when prompted to confirm the installation.

Step 4: Add MSYS2 QEMU to the System PATH (Optional)

- If you want to use QEMU from the regular Command Prompt, add the MSYS2 mingw64\bin directory to your system PATH:
- Path to add: C:\msys64\mingw64\bin
- Follow the same steps as in Method 1 to edit the Environment Variables.

Step 5: Verify the Installation

- Open the MSYS2 MinGW 64-bit terminal.
- Run:

```
qemu-system-arm --version
```

- You should see QEMU version information.

Method 3: Using Windows Subsystem for Linux (WSL)

If you're using Windows 10 (version 2004 and later) or Windows 11, you can install WSL to run a Linux environment directly on Windows.

Step 1: Enable WSL

- Open PowerShell or Command Prompt as Administrator.
- Run:

```
wsl --install
```

- This command installs WSL with the default Linux distribution (Ubuntu).
- Restart your computer when prompted.

Step 2: Install QEMU in WSL

- Open the Ubuntu terminal from the Start menu.
- Update package lists:

```
sudo apt-get update
```

- Install QEMU:

```
sudo apt-get install qemu-system-arm
```

- Press Y to confirm the installation.

Step 3: Verify the Installation

- Run:

```
qemu-system-arm --version
```

- You should see QEMU version information.

Understanding the difference between using an emulator and the real device:

• Memory Addresses:
The memory map of the VersatilePB machine differs from the BeagleBone Black. Adjusting the starting address in the linker script and the stack pointer ensures that your code and data are placed in valid RAM locations.

• UART Registers:
The UART peripheral on the VersatilePB is different from that on the BeagleBone Black. Updating the base address and register offsets allows your code to interact correctly with the emulated UART device.

Limitations and Considerations

• Hardware Differences:
The VersatilePB machine does not emulate the BeagleBone Black's hardware exactly. For simple programs like "Hello World," this is acceptable, but more complex hardware-specific code may not work without further adjustments.

• Instruction Set Compatibility:
The VersatilePB uses an ARM926EJ-S processor (ARMv5 architecture), while the BeagleBone Black uses an ARM Cortex-A8 (ARMv7-A). For basic ARM instructions used in simple programs, this difference is negligible.

• Peripheral Support:
If you plan to emulate more advanced peripherals or need closer hardware matching, consider using other QEMU machine models or simulators.

Advantages of Testing on an Emulator

• No Hardware Dependency:
Develop and test your code without needing the physical BeagleBone Black hardware.

• Faster Development Cycle:
Quickly iterate on your code and test changes immediately.

• Debugging Capabilities:
Utilize powerful debugging tools to diagnose and fix issues.

Limitations

• Hardware Accuracy:
Emulators may not perfectly replicate the behavior of the actual hardware, especially for complex peripherals.

• Performance Differences:
Execution speed and timing may differ from real hardware.

• Learning Curve:
Setting up and configuring emulators requires additional effort and understanding.

A few comments for Mac Users.

If you're on macOS, and pressing Ctrl-a followed by x isn't exiting QEMU as expected. This issue can occur because macOS terminals or shells might handle control characters differently, or they might intercept certain key combinations for their own use.

Here is an alternative method to exit QEMU on macOS:

Method 1: Change the QEMU Escape Character

By default, QEMU uses Ctrl-a as the escape character when running in -nographic mode. You can change this escape character to one that doesn't conflict with macOS key bindings.

Steps:

1. Choose a New Escape Character
Let's use Ctrl-t (which corresponds to ASCII code 0x14).

2. Start QEMU with the -echr Option
```
qemu-system-arm -M versatilepb -nographic -echr 0x14 -kernel hello.elf
```
• -echr 0x14 sets the escape character to Ctrl-t.

3. Use the New Escape Sequence to Exit
Press Ctrl-t, then x to exit QEMU.

Explanation: Ctrl-t is less likely to be intercepted by macOS terminal applications.
The -echr option allows you to specify an escape character that suits your environment.

"Hello World"

Creating a "Hello World" program for the BeagleBone Black at the bare-metal level is an excellent way to learn about low-level programming and hardware interaction. In this guide, we'll create a minimal program that outputs "Hello World" via the serial console using BeagleBone Black's UART0 interface.

Writing the "Hello World" Program

We will write two source files: an assembly startup file (startup.s) and a C program (hello.c). We will also use a linker script (memmap) to control the memory layout.

Compiling the Program

Use the ARM cross-compiler toolchain to compile and link your program.

1. Clean Previous Builds

```
rm -f *.o hello.elf
```

2. Compile the Program

Use the ARM cross-compiler to build your program.

```
arm-none-eabi-gcc -c startup.s -o startup.o
arm-none-eabi-gcc -c hello.c -o hello.o
arm-none-eabi-ld -T memmap.ld startup.o hello.o -o hello.elf
```

- Explanation:
    - startup.s: Assembly startup code.
    - main.c: The main program.
    - memmap.ld: Linker script adjusted for QEMU's VersatilePB machine.

3. Run the Program in QEMU

Execute your program using QEMU with the appropriate machine and options.

```
qemu-system-arm -M versatilepb -nographic -kernel hello.elf
```

- Option Breakdown:
- -M versatilepb: Specifies the VersatilePB machine model.
- -nographic: Disables graphical output; redirects serial I/O to the console.
- -kernel hello_world.elf: Loads your compiled ELF binary as the kernel.

4. View the Output

You should see the following output in your terminal:

```
Hello World
```