



How to program
STM32F7
using open source,
cross-platform tools
only?

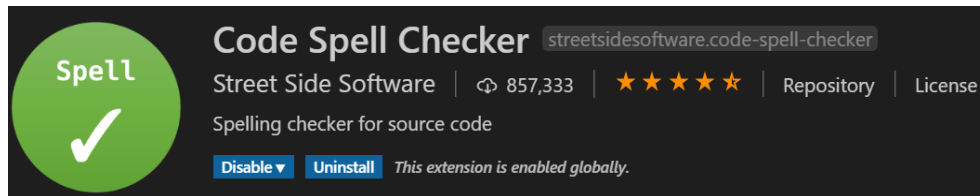
Demo with graphics using STM32F769 Discovery Kit
by Tomasz Jastrzębski, November 2019 (revised March 2020)
source code available at:

https://github.com/tjastrzebski/DISCO-F769NI_LCD_demo

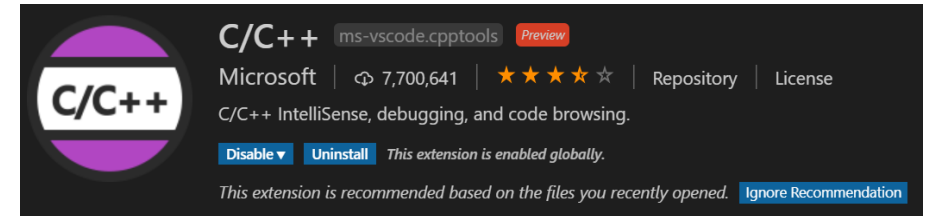
License: MIT

VS Code Installation

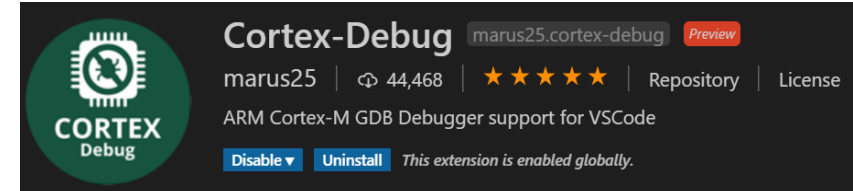
- Install VS Code <https://code.visualstudio.com/>
- Install VS Code Extensions:
 - C/C++ VS
 - Cortex-Debug
 - Intel HEX format
 - LinkerScript
 - Code Spell Checker



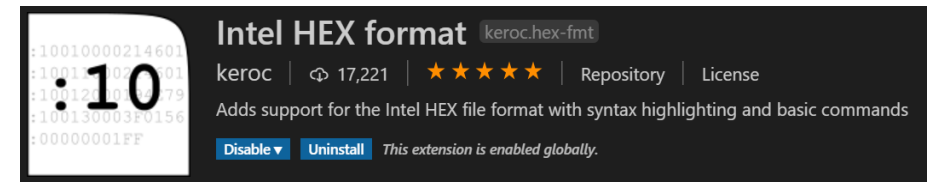
Code Spell Checker `streetsidesoftware.code-spell-checker`
Street Side Software | 857,333 | ★★★★★ | Repository | License
Spelling checker for source code
[Disable](#) [Uninstall](#) This extension is enabled globally.



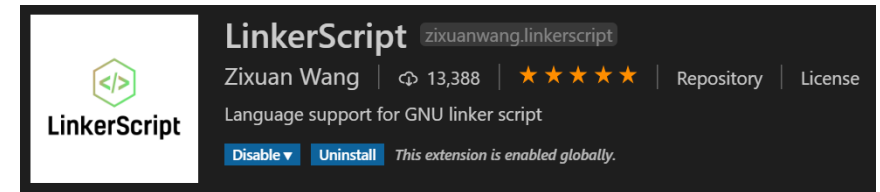
C/C++ `ms-vscode.cpptools` Preview
Microsoft | 7,700,641 | ★★★★★☆ | Repository | License
C/C++ IntelliSense, debugging, and code browsing.
[Disable](#) [Uninstall](#) This extension is enabled globally.
This extension is recommended based on the files you recently opened. [Ignore Recommendation](#)



Cortex-Debug `marus25.cortex-debug` Preview
marus25 | 44,468 | ★★★★★ | Repository | License
ARM Cortex-M GDB Debugger support for VSCode
[Disable](#) [Uninstall](#) This extension is enabled globally.



Intel HEX format `keroc.hex-fmt`
keroc | 17,221 | ★★★★★ | Repository | License
Adds support for the Intel HEX file format with syntax highlighting and basic commands
[Disable](#) [Uninstall](#) This extension is enabled globally.



LinkerScript `zixuanwang.linkerscript`
Zixuan Wang | 13,388 | ★★★★★ | Repository | License
Language support for GNU linker script
[Disable](#) [Uninstall](#) This extension is enabled globally.

GNU Arm Embedded Toolchain Installation

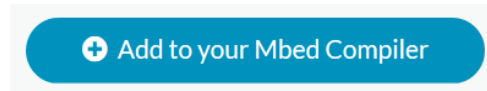
1. Download and install **GNU Arm Embedded Toolchain** installer, version **9 2019-q4-major**
<https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads>
2. During the installation check “Add path to environment variable”

Alternatively:

1. Download zip package, unpack it to selected folder, e.g.
[C:/Program Files \(x86\)/GNU Tools ARM Embedded/9 2019-q4-major](#)
- this is the default location if you use installer, it contains white space but that's OK
2. Add **bin** folder to the **PATH** system environment variable
Example: **C:/Program Files (x86)/GNU Tools ARM Embedded/9 2019-q4-major/bin**

Sample Source Code

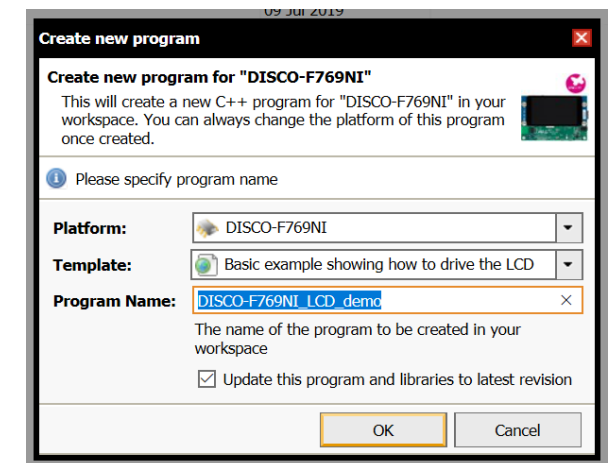
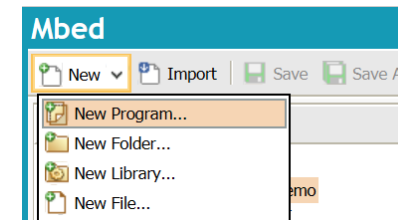
- Login to <https://www.mbed.com/>
- you need to create an account first
- Go to Hardware/Boards website menu,
find **DISCO-F769NI** board (STM32F769I-DISCO),
select "**Add to your Mbed Compiler**" option in the right panel



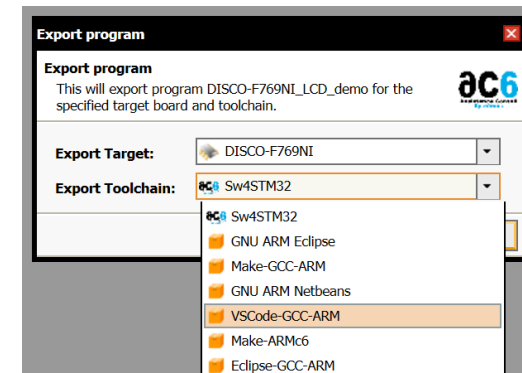
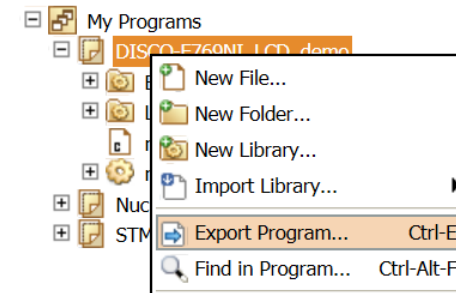
- Go to **Compiler** →



- Select **New/New Program** menu option.
In the config form select your board and
Basic example showing how to drive the LCD template



- Right-click your new program, select **Export Program** option
- Select **VSCode-GCC-ARM** export toolchain
- Save generated ZIP file locally, unzip to preferred location



Git source control

1. Download and install **Git**
<https://git-scm.com/downloads>
2. On Windows we need not only source control but some convenient Unix commands available in Bash shell installed with Git.
Namely: **find** and **rm**

Git

1. Add **.gitignore** file with the following content:

```
/BUILD  
*.state.json
```

1. From the terminal window issue commands:

- `git config --global user.email "your@email.here.com"`
- `git init`
- `git add .`
- `git commit -m "initial version"`

GNU make

Obtain GNU Make build tool

<https://www.gnu.org/software/make/>

There many sources available.

Example:

1. Download **gnumake-4.2.1-x64.exe** or **gnumake-4.2.1.exe** from <https://github.com/mbuilov/gnumake-windows>
2. Rename downloaded file to **make.exe**
3. Place it where it will be easily accessible, e.g. **C:/Windows/System32**

modify c_cpp_properties.json

1. Leave only one configuration, name it **"GCC"**
 - we do not need separate configurations for Mac, Linux and Windows since build is not OS specific but compiler specific
2. "compilerPath": "arm-none-eabi-g++"
3. "includePath": [
 "/usr/src/mbed-sdk/",
 "BSP_DISCO_F769NI/**",
 "LCD_DISCO_F769NI/**",
 "mbed/**"
]

recreate **tasks.json**

Supplied **tasks.json** file uses old syntax version 0.1.0

Create new content:

```
{  
  "version": "2.0.0",  
  "tasks": [  
  ]  
}
```

recreate tasks.json

Add **Build** task

```
{
  "label": "Build",
  "type": "shell",
  "command": "make",
  "args": [
    "-j8",
    "-output-sync=recurse",
    "all",
    "OPT=-O0 -g"
  ],
  "options": {
    "cwd": "${workspaceRoot}"
  },
  "presentation": {
    "clear": true
  },
  "problemMatcher": ["$gcc"],
  "group": {
    "kind": "build",
    "isDefault": true
  }
}
```

recreate tasks.json

Add **Clean** task to **tasks.json** file:

```
{
  "label": "Clean",
  "type": "shell",
  "command": "make",
  "args": [
    "clean"
  ],
  "options": {
    "cwd": "${workspaceRoot}"
  },
  "group": {
    "kind": "build",
    "isDefault": true
  },
  "problemMatcher": [],
  "presentation": {
    "clear": true
  }
}
```

For debug build modify Makefile

- In line 35 before **VPATH** variable add **OPT** variable definition:

```
# optimization params for RELEASE, change to [-O0 -g] for DEBUG version  
OPT = -Os -g1
```

- In **C_FLAGS**, **CXX_FLAGS** and **ASM_FLAGS** replace **-Os** and **-g** flags with **OPT** variable, e.g. change:

```
C_FLAGS += -Os  
C_FLAGS += -g  
to:  
C_FLAGS += $(OPT)
```

- Fix **mbed_config.h** path in **ASM_FLAGS**

Run build

At this stage you should be able to build your project.
Press Ctrl+Shift+B, select **Build** task.

After the build is finished, BUILD bolder should contain **DISCO-F769NI_LCD_demo.bin** output file.

Run build

Note: on some systems you may get the following “unexpected EOF” error in Makefile around line **530**

```
/usr/bin/sh: -c: line 0: unexpected EOF while looking for matching `''  
/usr/bin/sh: -c: line 1: syntax error: unexpected end of file  
make[1]: *** [h:/DISCO-F769NI_LCD_demo/Makefile:530: DISCO-F769NI_LCD_demo.elf] Error 1  
make: *** [Makefile:26: all] Error 2  
The terminal process terminated with exit code: 2
```

While I was unable to determine the root cause, it seems the problem is related to shell command length limits.

On Windows you may try to use cmd.exe or PowerShell.

The above problem does not occur if you use **Build++**. Continue reading.

get stlink utility

<https://github.com/texane/stlink/>

1. Download **stlink** version 1.3.0 (Windows & Mac)
<https://github.com/texane/stlink/releases/tag/1.3.0>
2. Windows:
 1. Extract content to **C:/Program Files/StLink**
 2. Add **C:/Program Files/StLink/bin** to the system **PATH** environment variable
 3. Download libusb from <https://libusb.info/>
 4. Copy files from **MS64/dll** folder to **C:/Program Files/StLink/bin**
- 7Zip utility needed from <http://www.7-zip.org>

Note: the latest version 1.6.0 and Linux executables are not available from author's Git repo
– you have to build them yourself.

Executable **stlink** version 1.6.0 for Windows is available from my repo:
https://github.com/tdjastrzebski/DISCO-F769NI_LCD_demo

Linux: build latest **stlink** utility

1. Download source code for the latest stable version as zip package from <https://github.com/texane/stlink/releases>

2. Follow build steps
<https://github.com/texane/stlink/blob/master/doc/compiling.md>

create **Deploy** task

Add **Deploy** task definition to **tasks.json** file:

```
{
  "label": "Deploy",
  "type": "shell",
  "command": "st-flash",
  "args": [
    "write",
    "${workspaceRoot}/BUILD/${workspaceRootFolderName}.bin",
    "0x08000000"
  ],
  "group": {
    "kind": "build",
    "isDefault": true
  },
  "problemMatcher": [],
  "presentation": {
    "clear": true
  }
}
```

At this point you should be able to deploy your program – run **Deploy** task

Debug

1. Obtain **STM32F7x9.svd** file from <https://github.com/posborne/cmsis-svd/tree/master/data/STMicro> and place it in the project root folder
2. Replace **launch.json** file content with:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Debug (ST-Util)",
      "type": "cortex-debug",
      "request": "launch",
      "serverType": "stutil",
      "cwd": "${workspaceRoot}",
      "executable": "./BUILD/${workspaceRootFolderName}.elf",
      "device": "STM32F769NI",
      "v1": false,
      "svdFile": "${workspaceRoot}/STM32F7x9.svd",
      "preLaunchTask": "Build & Deploy"
    }
  ]
}
```

Debug – cont'd

3. Add **Build & Deploy** task to **tasks.json** file:

```
{
  "label": "Build & Deploy",
  "dependsOn": ["Build", "Deploy"],
  "dependsOrder": "sequence",
  "group": {
    "kind": "build",
    "isDefault": true
  },
  "problemMatcher": []
}
```

Finally, set breakpoint within **main.cpp** file, press **F5** to build, deploy and start

Build++

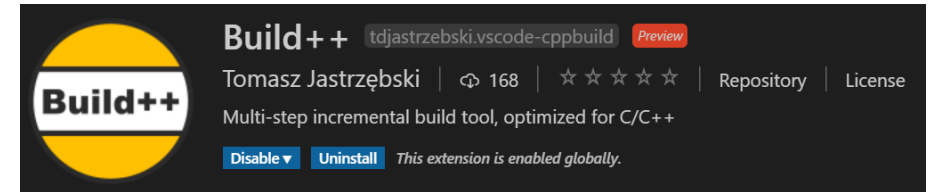
1. Install **Node.js** from <https://nodejs.org/en/download/>
2. Install **CppBuild**: `npm install cppbuild -g`
3. You should be able to build your project from VS terminal command line:
`cppbuild gcc debug -w`
note:
 - use Bash shell,
 - run “cbbpulid --help” for more options
4. Install **Build++** VS Code extension

5. Add **Build++** task to **tasks.json** file:

```
{
  "label": "Build++",
  "type": "shell",
  "command": "cppbuild",
  "args": ["gcc", "debug", "-w" ],
  "presentation": {
    "clear": true
  },
  "problemMatcher": ["$gcc"],
  "group": {
    "kind": "build",
    "isDefault": true
  }
}
```

5. Modify **lunch.json** file:

```
"executable": "./BUILD/debug/${workspaceRootFolderName}.elf",
"preLaunchTask": "Build++"
```



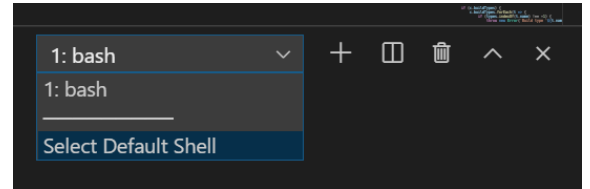
6. Add c_cpp_build.json file to .vscode folder

```
{
  "version": 1,
  "params": {
    "buildDir": "BUILD",
    "buildOutput": "${buildDir}/${buildTypeName}"
  },
  "configurations": [
    {
      "name": "gcc",
      "problemMatchers": ["$gcc"],
      "buildTypes": [
        {
          "name": "debug",
          "params": { "buildTypeParams": "-O0 -g" }
        },
        {
          "name": "release",
          "params": { "buildTypeParams": "-Os -g1" }
        }
      ]
    },
    "buildSteps": [
      {
        "name": "C Compile",
        "filePattern": "**/*.c",
        "outputFile": "${buildOutput}/${fileDirectory}/${fileName}.o",
        "trimIncludePaths": true,
        "params": {
          "flags": "-fdata-sections -ffunction-sections -fmessage-length=0 -fno-delete-null-pointer-checks -fno-exceptions -fomit-frame-pointer -funsigned-char -mcpu=cortex-m7 -mfloat-abi=softfp -mfpv5-d16 -MMD -mthumb -Wall -Wextra -Wno-missing-field-initializers -Wno-unused-parameter",
          "command": "arm-none-eabi-gcc -std=gnu11 -c ${buildTypeParams} ${flags} (-I${includePath}) (-D${defines}) (-include ${forcedInclude}) [${filePath}] -o [${outputFile}]"
        },
        {
          "name": "C++ Compile",
          "filePattern": "**/*.cpp",
          "outputFile": "${buildOutput}/${fileDirectory}/${fileName}.o",
          "trimIncludePaths": true,
          "params": {
            "flags": "-fdata-sections -ffunction-sections -fmessage-length=0 -fno-delete-null-pointer-checks -fno-exceptions -fno-rtti -fomit-frame-pointer -funsigned-char -mcpu=cortex-m7 -mfloat-abi=softfp -mfpv5-d16 -MMD -mthumb -Wall -Wextra -Wno-missing-field-initializers -Wno-unused-parameter -Wvla",
            "command": "arm-none-eabi-g++ -std=gnu++14 -c ${buildTypeParams} ${flags} (-I${includePath}) (-D${defines}) (-include ${forcedInclude}) [${filePath}] -o [${outputFile}]"
          },
          {
            "name": "Build link script",
            "params": {
              "flags": "-E -P -Wl,--gc-sections -Wl,--wrap,main -Wl,--wrap,malloc_r -Wl,--wrap,free_r -Wl,--wrap,realloc_r -Wl,--wrap,memalign_r -Wl,--wrap,calloc_r -Wl,--wrap,exit -Wl,--wrap,atexit -Wl,-n -mcpu=cortex-m7 -mthumb -mfpv5-d16 -mfloat-abi=softfp",
              "memDefines": "-DMBED_ROM_START=0x80000000 -DMBED_ROM_SIZE=0x200000 -DMBED_ROM1_START=0x200000 -DMBED_ROM1_SIZE=0x200000 -DMBED_RAM_START=0x20020000 -DMBED_RAM_SIZE=0x60000 -DMBED_RAM1_START=0x20000000 -DMBED_RAM1_SIZE=0x20000 -DMBED_BOOT_STACK_SIZE=4096"
            },
            "command": "arm-none-eabi-cpp ${flags} ${memDefines} [mbed/TARGET_DISCO_F769NI/TOOLCHAIN_GCC_ARM/STM32F769xI.ld] -o [${buildOutput}/${workspaceRootFolderName}.link_script.ld]"
          },
          {
            "name": "Build object list 1",
            "command": "find [${buildOutput}] -type f -name '*.o' > [${buildOutput}/objects.txt]"
          },
          {
            "name": "Build object list 2",
            "command": "find mbed -type f -name '*.o' >> [${buildOutput}/objects.txt]"
          },
          {
            "name": "Link to elf",
            "params": {
              "flags": "-Wl,--gc-sections -Wl,--wrap,main -Wl,--wrap,malloc_r -Wl,--wrap,free_r -Wl,--wrap,realloc_r -Wl,--wrap,memalign_r -Wl,--wrap,calloc_r -Wl,--wrap,exit -Wl,--wrap,atexit -Wl,-n -mcpu=cortex-m7 -mthumb -mfpv5-d16 -mfloat-abi=softfp",
              "memDefines": "-DMBED_ROM_START=0x80000000 -DMBED_ROM_SIZE=0x200000 -DMBED_ROM1_START=0x200000 -DMBED_ROM1_SIZE=0x200000 -DMBED_RAM_START=0x20020000 -DMBED_RAM_SIZE=0x60000 -DMBED_RAM1_START=0x20000000 -DMBED_RAM1_SIZE=0x20000 -DMBED_BOOT_STACK_SIZE=4096",
              "sysLibs": "-lmbed -Wl,--start-group -lstdc++ -lsupc++ -lm -lc -lgcc -lnosys -lmbed -Wl,--end-group"
            },
            "command": "arm-none-eabi-gcc ${flags} ${memDefines} -T [${buildOutput}/${workspaceRootFolderName}.link_script.ld] -L[mbed/TARGET_DISCO_F769NI/TOOLCHAIN_GCC_ARM] -o [${buildOutput}/${workspaceRootFolderName}.elf] @[${buildOutput}/objects.txt] ${sysLibs}"
          },
          {
            "name": "elf -> bin",
            "command": "arm-none-eabi-objcopy -O binary [${buildOutput}/${workspaceRootFolderName}.elf] [${buildOutput}/${workspaceRootFolderName}.bin]"
          },
          {
            "name": "elf -> hex",
            "command": "arm-none-eabi-objcopy -O ihex [${buildOutput}/${workspaceRootFolderName}.elf] [${buildOutput}/${workspaceRootFolderName}.hex]"
          },
          {
            "name": "Deploy",
            "command": "st-flash write [${buildOutput}/${workspaceRootFolderName}.bin] 0x08000000"
          }
        ]
      }
    ]
  }
}
```

Do NOT just copy-paste this code. Very likely some characters (-) will be missing.
Get this code from: https://github.com/tdjastrzebski/DISCO-F769NI_LCD_demo

Build++ cont'd

Set your default VS Code default shell to Bash.



Press **F5**. Your project should build, deploy and start in debug mode.

Note: make sure your current VS Code terminal shell supports Unix-style “find” command used in Build++ file. Use Git Bash or similar, do NOT use Windows Command Prompt or PowerShell.

Happy coding 😊

Tomasz Jastrzębski

<https://www.linkedin.com/in/t-jastrzebski/>