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

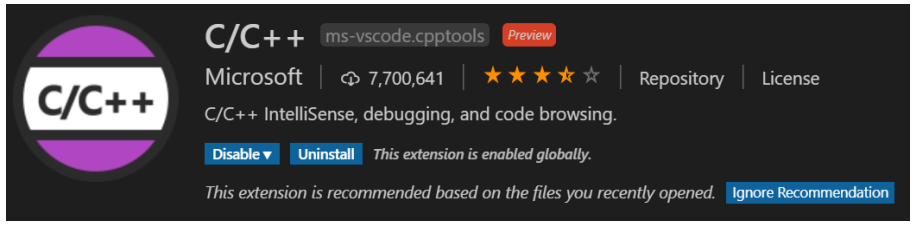
Sample with graphics using STM32F769 Discovery Kit

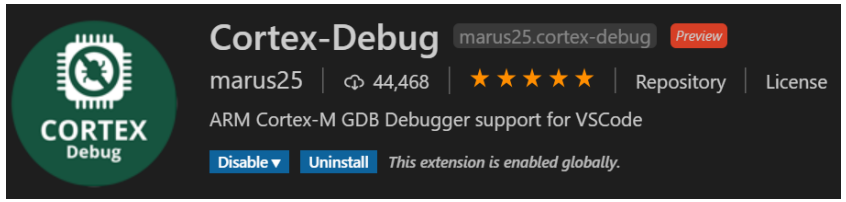
by Tomasz Jastrzębski, October 2019

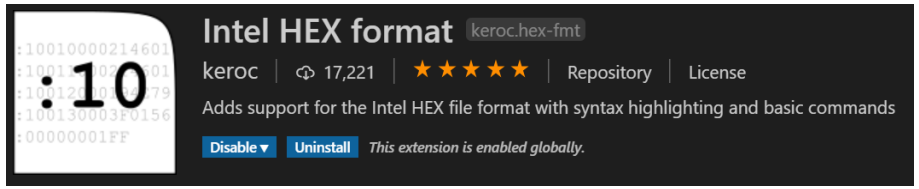
https://github.com/tdjastrzebski/DISCO-F769NI_LCD_demo

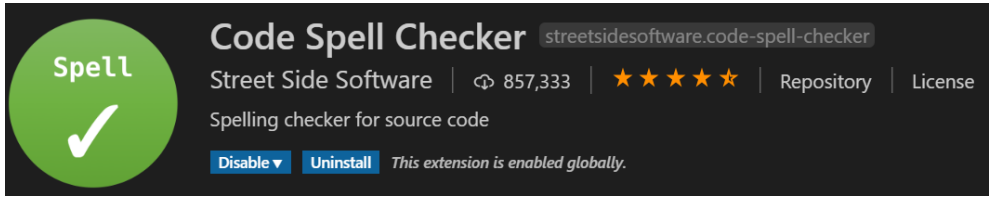
VS Code Installation

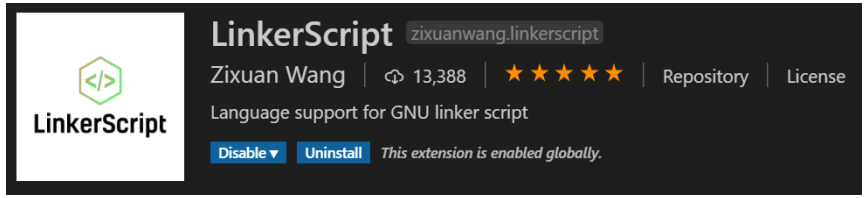
- Install VS Code https://code.visualstudio.com/
- Install VS Code Extensions:
  - C/C++ VS
  - Cortex-Debug
  - Intel HEX format
  - LinkerScript
  - Code Spell Checker











GNU Arm Embedded Toolchain Installation

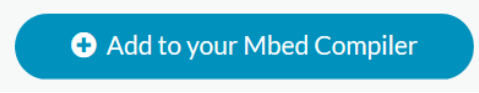
1. Download and install **GNU Arm Embedded Toolchain** installer, version **8-2019-q3-update**
   https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads
2. During the installation check “Add path to environment variable”

Alternatively

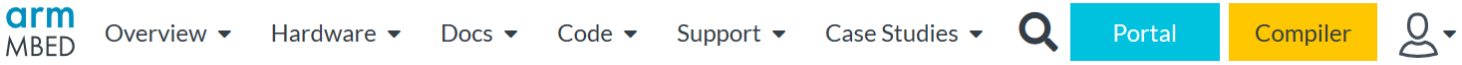
1. Download zip package, unpack it to selected folder, e.g.
   C:/Program Files (x86)/GNU Tools ARM Embedded/8 2019-q3-update
   - this is the default location if you use installer, it contains white space but that’s OK
2. Add **bin** folder to the **PATH** system environment variable
   Example: **C:/Program Files (x86)/GNU Tools ARM Embedded/8 2019-q3-update/bin**

Sample Source Code

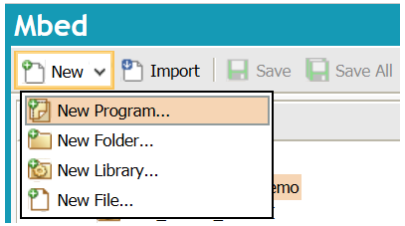
- Login to https://www.mbed.com/
  - you need to create an account first
- Go to Hardware/Boards website menu, find **DISCO-F769NI** board (STM32F769I-DISCO), select **Add to your Mbed Compiler** option in the right panel

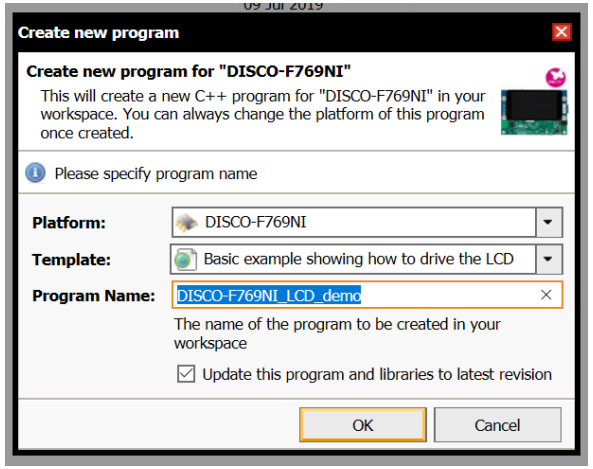


- Go to Compiler

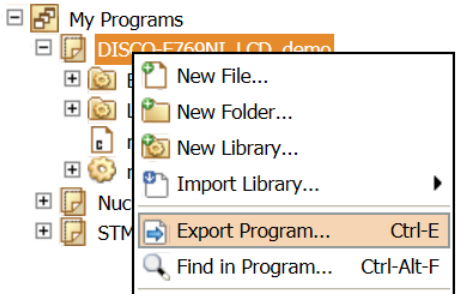


- Select **New/New Program** menu option. In the config form select your board and **Basic example showing how to drive the LCD** template

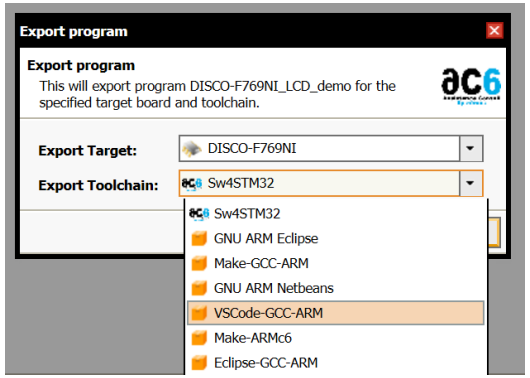




- Right-click your new program, select **Export Program** option



- 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`

2. From 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 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 build is finished BUILD bolder should contain

DISCO-F769NI_LCD_demo.bin output file.

get **stlink** utility
https://github.com/texane/stlink/

1. Download **stlink** version 1.5.1 from (Windows)
   https://github.com/tdjastrzebski/DISCO-F769NI_LCD_demo
   or https://github.com/texane/stlink/releases/tag/1.3.0 (Mac & Windows)
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

Linux: build latest **stlink** utility

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

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.

Debug

1. Obtain **STM32F7x9.svd** file from https://github.com/posborne/cmsis-svd/tree/master/data/STMicro and place 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

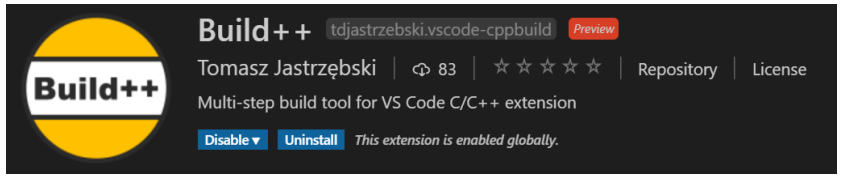
3. Add **Build & Deploy** task to **tasks.json** file:

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

Finally, set breakpoint within **main.cpp** file, press **F5**

Build++

1. Install **Node.js** from https://nodejs.org/en/download/
2. Install **Build++** VS Code extension
3. Install **CppBuild**: `npm install cppbuild -g`
4. 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
    }
}
```

Build++ cont’d

1. Modify **lunch.json** file

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

2. Add **c_cpp_build.json** file

```
{
    "version": 1,
    "params": { "buildDir": "BUILD" },
    "configurations": [
        {
            "name": "GCC",
            "problemMatchers": ["$gcc"],
            "buildTypes": [
                {
                    "name": "debug",
                    "params": { "buildTypeParams": "-O0 -g" }
                },
                {
                    "name": "release",
                    "params": { "buildTypeParams": "-Os -g1" }
                }
            ],
            "params": { "buildOutput": "${buildDir}/${buildTypeName}" },
            "buildSteps": [
                {
                    "name": "C Compile",
                    "filePattern": "**/*.c",
                    "outputFile": "${buildOutput}/${fileDirectory}/${fileName}.o",
                    "command": "arm-none-eabi-gcc -std=gnu11 -c ${buildTypeParams} -Wall -Wextra -Wno-unused-parameter -Wno-missing-field-initializers -fmessage-length=0 -fno-exceptions -ffunction-sections -fdata-sections -funsigned-char -MMD -fno-delete-null-pointer-checks -
fomit-frame-pointer -mcpu=cortex-m7 -mthumb -mfpu=fpv5-d16 -mfloat-abi=softfp (-I[$${includePath}]) (-D$${defines}) (-include [$${forcedInclude}]) [${filePath}] -o [${outputFile}]"
                },
                {
                    "name": "C++ Compile",
                    "filePattern": "**/*.cpp",
                    "outputFile": "${buildOutput}/${fileDirectory}/${fileName}.o",
                    "command": "arm-none-eabi-g++ -std=gnu++14 -c ${buildTypeParams} -fno-rtti -Wvla -Wall -Wextra -Wno-unused-parameter -Wno-missing-field-initializers -fmessage-length=0 -fno-exceptions -ffunction-sections -fdata-sections -funsigned-char -MMD -fno-delete-
null-pointer-checks -fomit-frame-pointer -mcpu=cortex-m7 -mthumb -mfpu=fpv5-d16 -mfloat-abi=softfp (-I[$${includePath}]) (-D$${defines}) (-include [$${forcedInclude}]) [${filePath}] -o [${outputFile}]"
                },
                {
                    "name": "Build link script",
                    "command": "arm-none-eabi-cpp -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 -mfpu=fpv5-
d16 -mfloat-abi=softfp -DMBED_ROM_START=0x8000000 -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 [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}/object_list.txt]"
                },
                {
                    "name": "Build object list 2",
                    "command": "find mbed -type f -name '*.o' >> [${buildOutput}/object_list.txt]"
                },
                {
                    "name": "Link to elf",
                    "command": "arm-none-eabi-gcc -Wl,--gc-sections -Wl,--wrap,main -Wl,--wrap,_malloc_r -Wl,--wrap,_free_r -Wl,--wrap,_realloc_r -Wl,--wrap,_memalign_r -Wl,--wrap,_calloc_r -Wl,--wrap,exit -Wl,--wrap,atexit -Wl,-n -mcpu=cortex-m7 -mthumb -mfpu=fpv5-d16 -
mfloat-abi=softfp -DMBED_ROM_START=0x8000000 -DMBED_ROM_SIZE=0x200000 -DMBED_ROM1_START=0x200000 -DMBED_ROM1_SIZE=0x200000 -DMBED_RAM_START=0x20020000 -DMBED_RAM_SIZE=0x60000 -DMBED_RAM1_START=0x20000000 -DMBED_RAM1_SIZE=0x20000 -DMBED_BOOT_STACK_SIZE=4096 -
T [${buildOutput}/${workspaceRootFolderName}.link_script.ld] -L[mbed/TARGET_DISCO_F769NI/TOOLCHAIN_GCC_ARM] -o [${buildOutput}/${workspaceRootFolderName}.elf] @[${buildOutput}/object_list.txt] -lmbed -Wl,--start-group -lstdc++ -lsupc++ -lm -lc -lgcc -lnosys -lmbed -Wl,--end-
group"
                },
                {
                    "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! Most likely some characters (-) will be missing.
Get this code from: https://github.com/tdjastrzebski/DISCO-F769NI_LCD_demo