University of British Columbia
Electrical and Computer Engineering
Electrical and Biomedical Engineering Design Studio
ELEC291/ELEC292

The STM32L051 Microcontroller System

Introduction

This document introduces a minimal microcontroller system using the ST Microelectronics' STM32L051 ARM Cortex-M0 microcontroller.

Recommended documentation

RM0451 Reference manual: *"Ultra-low-power STM32L0x0 advanced Arm®-based 32-bit MCUs":*

https://www.st.com/resource/en/reference_manual/dm00443854-ultra-low-power-stm32l0x0-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf

Datasheet - production data: "STM32L051x6 STM32L051x8 Access line ultra-low-power 32-bit MCU Arm®-based Cortex®-M0+,up to 64 KB Flash, 8 KB SRAM, 2 KB EEPROM, ADC"

https://www.st.com/content/ccc/resource/technical/document/datasheet/9a/75/bd/16/0f/fd/49/19/DM00108219.pdf/files/DM00108219.pdf/jcr:content/translations/en.DM00108219.pdf

Assembling the Microcontroller System

Figure 1 shows the circuit schematic of the STM32L051 microcontroller system used in ELEC291/ELEC292. It can be assembled using a bread board. Table 1 below lists the components needed to assemble the circuit.

Quantity	Digi-Key Part #	Description
2	BC1148CT-ND	0.1uF ceramic capacitors
2	BC1157CT-ND	1uF ceramic capacitors
2	270QBK-ND	270Ω resistor
1	330QBK-ND	330Ω resistor
1	67-1102-ND	LED 5MM RED
1	67-1108-ND	LED 5MM GREEN
1	MCP1700-3302E/TO-ND	MCP17003302E 3.3 Voltage Regulator
1	N/A	BO230XS USB adapter
1	497-14901-ND	STM32L051K8T6
1	1528-1065-ND	LQFP32 to DIP32 adapter board
2	A26509-16-ND	16-pin header connector
1	P8070SCT-ND	Push button switch

Table 1. Parts required to assemble the STM32L051 microcontroller system.

Before assembling the circuit in the breadboard, the STM32L051 microcontroller has to be soldered to the LQFP32 to DIP32 adapter board. This task can be accomplished using a solder iron as described in this video:

https://www.youtube.com/watch?v=8yyUlABj29o

Once the microcontroller is soldered to the adapter board, solder the two 16-pin header connectors. The assembled bread-boarded circuit should look similar to the one in figure 1. Notice that the STM32L051 works with a VDD voltage between 2.0V and 3.6V. For this reason a voltage regulator is required to convert the USB adapter 5V to 3.3V.

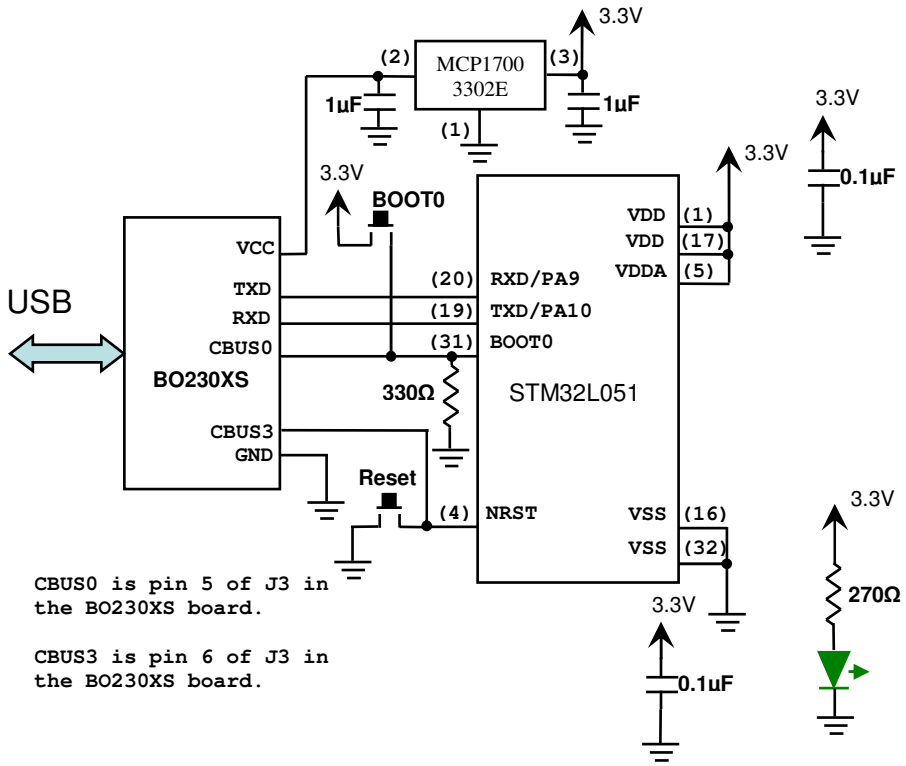


Figure 1. Circuit schematic of the microcontroller system.

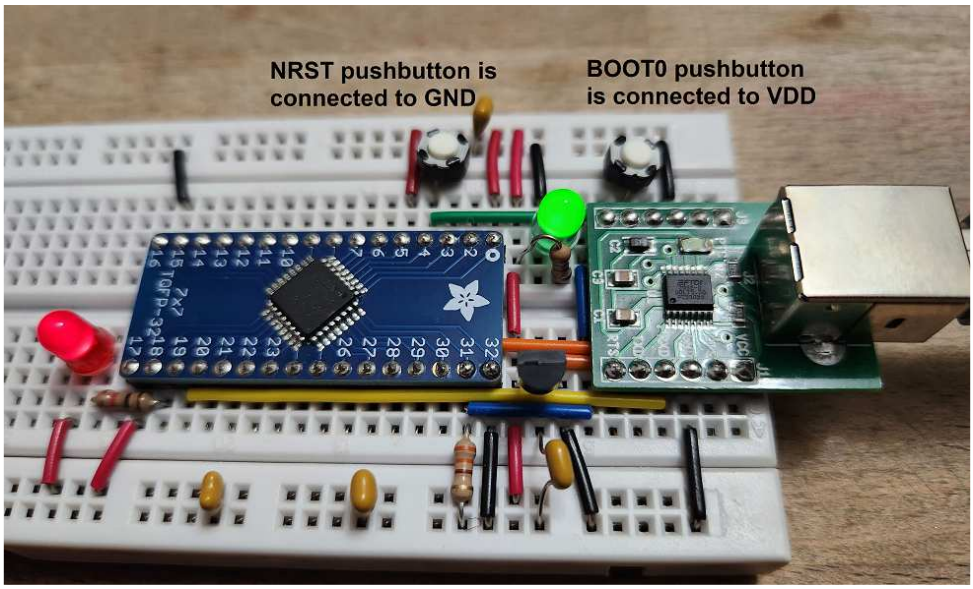


Figure 2. Bread boarded STM32L051 microcontroller system. The red LED is connected to pin 18 (PA8) as needed by the 'blinky' example available from the course webpage. WARNING: The RESET pushbutton to the left is connected to GND. The BOOT0 pushbutton to the right is connected to VDD.

Setting up the Development Environment

To establish a workflow for the STM32 we need to install the following three packages:

1. **CrossIDE V2.25 (or newer) & GNU Make V4.2 (or newer)**

   Download CrossIDE from: http://ece.ubc.ca/~jesusc/crosside_setup.exe and install it. Included in the installation folder of CrossIDE is GNU Make V4.2 (make.exe, make.pdf). GNU Make should be available in one of the folders of the PATH environment variable in order for the workflow described bellow to operate properly. For example, suppose that CrossIDE was installed in the folder "C:\crosside"; then the folder "C:\crosside" should be added at the end of the environment variable "PATH" as described here[1].

2. **GNU ARM Embedded Toolchain.**

   Download and install the GNU ARM Embedded Toolchain from:

   https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads.

   The "bin" folder of the GNU ARM Embedded Toolchain must be added to the environment variable "PATH" in order for the workflow described bellow to operate properly. For example, if the toolchain is installed in the folder "C:\Programs\GNU Tools ARM Embedded", then the folder "C:\Programs\GNU Tools ARM Embedded\5.4 2016q2\bin" must be added at the end of the environment variable "PATH" as described here[1].

   Notice that the folder "5.4 2016q2" was the folder available at the time of writing this document. For newer versions of the GNU ARM Embedded Toolchain this folder name will change to reflect the installed version.

3. **STM32 Flash Loader.**

   Available with the examples provided in the course web page is "stm32flash"[2]. This the program used to load the flash memory of the STM32L051.

Workflow.

The workflow for the STM32L051 microcontroller includes the following steps.

1. **Creation and Maintenance of Makefiles.**

   CrossIDE version 2.26 or newer supports project management using **simple** Makefiles by means of GNU Make version 4.2 or newer. A CrossIDE project Makefile allows for easy compilation and linking of multiple source files, execution of external commands, source code management, and access to microcontroller flash programming. The typical Makefile is a text file, editable with the CrossIDE editor or any other editor, and looks like this:

[1] http://www.computerhope.com/issues/ch000549.htm

[2] 'stm32flash' is also available from http://stm32flash.sourceforge.net/. The program was slightly modified in order to integrate it with better both with CrossIDE and GNU Make.

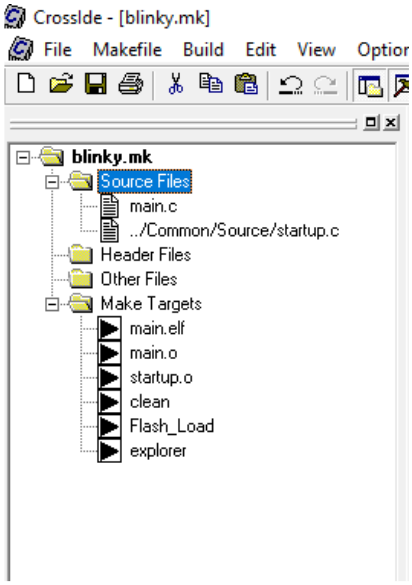
```
# Since we are compiling in windows, select 'cmd' as the default shell.  This
# is important because make will search the path for a linux/unix like shell
# and if it finds it will use it instead.  This is the case when cygwin is
# installed.  That results in commands like 'del' and echo that don't work.
SHELL=cmd
# Specify the compiler to use
CC=arm-none-eabi-gcc
# Specify the assembler to use
AS=arm-none-eabi-as
# Specify the linker to use
LD=arm-none-eabi-ld

# Flags for C compilation
CCFLAGS=-mcpu=cortex-m0plus -mthumb -g
# Flags for assembly compilation
ASFLAGS=-mcpu=cortex-m0plus -mthumb -g
# Flags for linking
LDFLAGS=-T ../Common/LDscripts/stm32l051xx_simple.ld -cref

# List the object files used in this project
OBJS= startup.o main.o

# The default 'target' (output) is main.elf and 'depends' on
# the object files listed in the 'OBJS' assignment above.
# These object files are linked together to create main.elf.
# The linked file is converted to hex using program objcopy.
main.elf: $(OBJS)
        $(LD) $(OBJS) $(LDFLAGS) -Map main.map -o main.elf
        arm-none-eabi-objcopy -O ihex main.elf main.hex
        @echo Success!

# The object file main.o depends on main.c. main.c is compiled
# to create main.o.
main.o: main.c
        $(CC) -c $(CCFLAGS) main.c -o main.o

# The object file startup.o depends on startup.c.  startup.c is
# compiled to create startup.o
startup.o: ../Common/Source/startup.c
        $(CC) -c $(CCFLAGS) ../Common/Source/startup.c -o startup.o

# Target 'clean' is used to remove all object files and executables
# associated with this project
clean:
        del $(OBJS)
        del main.elf main.hex main.map
        del *.lst

# Target 'Flash_Load' is used to load the hex file to the microcontroller
# using the flash loader.
Flash_Load:
#If putty is running, stop it
        @taskkill /f /im putty.exe /t /fi "status eq running" > NUL
# Create a '.bat' file to call the flash loader with the correct serial port
        @echo ..\stm32flash\stm32flash -w main.hex -v -g 0x0 ^^>loadf.bat
# Activate the bootloader and attach the serial port name to the '.bat' file
        @..\stm32flash\BO230\BO230 -b >>loadf.bat
# Run the '.bat' file
        @loadf

explorer:
        @explorer .
```

The preferred extension used by CrossIDE Makefiles is ".mk". For example, the file above is named "blinky.mk". Makefiles are an industry standard. Information about using and maintaining Makefiles is widely available on the internet. For example, these links show how to create and use simple Makefiles.

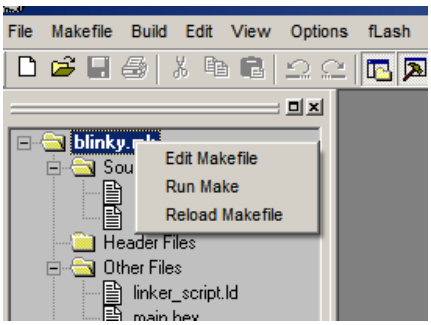
http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/
https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html
https://en.wikipedia.org/wiki/Makefile

2. Using Makefiles with CrossIDE: Compiling, Linking, and Loading.

To open a Makefile in CrossIDE, click "Makefile"→"Open" and select the Makefile to open. For example "blinky.mk". The project panel is displayed showing all the targets and source files:



Double clicking a source file will open it in the source code editor of CrossIDE. Double clicking a target 'makes' that target. Right clicking the Makefile name shows a pop-up menu that allows for editing, running, or reloading of the Makefile:



Additionally, the Makefile can be run by means of the Build menu or by using the Build Bar:

Clicking the 'wall' with green 'bricks' makes only the files that changed since the last build. Clicking the 'wall' with colored 'bricks' makes all the files. Clicking the 'brick' with an arrow, makes only the selected target. You can also use F7 to make only the files that changed since the last build and Ctrl+F7 to make only the selected target.

Compiling & Linking

After clicking the build button this output is displayed in the report panel of CrossIDE:

```
----------------- CrossIde - Running Make -----------------
arm-none-eabi-gcc -c -mcpu=cortex-m0plus -mthumb -g ../Common/Source/startup.c -o startup.o
arm-none-eabi-gcc -c -mcpu=cortex-m0plus -mthumb -g main.c -o main.o
arm-none-eabi-ld startup.o main.o  -T ../Common/LDscripts/stm32l051xx_simple.ld -cref -Map main.map
-o main.elf
arm-none-eabi-objcopy -O ihex main.elf main.hex
Success!
```

Loading the Hex File into the Microcontroller's Flash Memory

To load the program into the microcontroller double click the 'Flash_Load' target. This output is then displayed in the report panel of CrossIDE and the program starts running.

```
C:\Source\crosside\Make_Projects\STM32L051\Blinky>..\stm32flash\stm32flash -w main.hex -v -g
0x0 COM53
stm32flash 0.7

http://stm32flash.sourceforge.net/

Using Parser : Intel HEX
Location     : 0x8000000
Size         : 712
Interface serial_w32: 57600 8E1
Version      : 0x31
Option 1     : 0x00
Option 2     : 0x00
Device ID    : 0x0417 (STM32L05xxx/06xxx)
- RAM        : Up to 8KiB  (4096b reserved by bootloader)
- Flash      : Up to 64KiB (size first sector: 32x128)
- Option RAM : 32b
- System RAM : 4KiB
Write to memory
Erasing memory
Writing and verifying...
0%    10%   20%   30%   40%   50%   60%   70%   80%   90%   100%
|.....|.....|.....|.....|.....|.....|.....|.....|.....|.....|
##############################################################
Done.

Starting execution at address 0x08000000... done.
```

A file named "COMPORT.inc" is created after running the flash loader program. The file contains the name of the port used to load the program, for example, in thie example above COM118 is stored in the file. "COMPORT.inc" can be used in the Makefile to create a target that starts a PuTTy serial session using the correct serial port:

```
PORTN=$(shell type COMPORT.inc)
.
.
putty:
        @taskkill /f /im putty.exe /t /fi "status eq running" > NUL
        @cmd /s start putty -serial $(PORTN) -sercfg 115200,8,n,1,N -v
```

For more details about using "COMPORT.inc" check the project examples available to download from the course web page

Setting up the Development Environment on macOS

Download and install Visual Studio Code (VS Code) for macOS from:

https://code.visualstudio.com/Download

In VS Code install the extension "Make support and task provider" by carlos-algms.

Install xcode and homebrew for macOS. I used the instructions from this link:

https://phoenixnap.com/kb/install-homebrew-on-mac

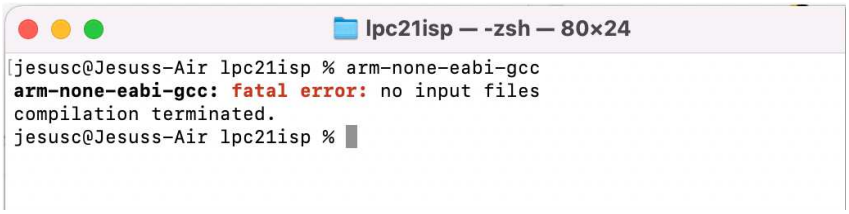
Install arm-none-eabi-gcc[3] using homebrew. In a terminal type:

```
$ brew tap ArmMbed/homebrew-formulae

$ brew install arm-none-eabi-gcc
```

To test that the ARM compiler is installed correctly, type in a terminal the following:

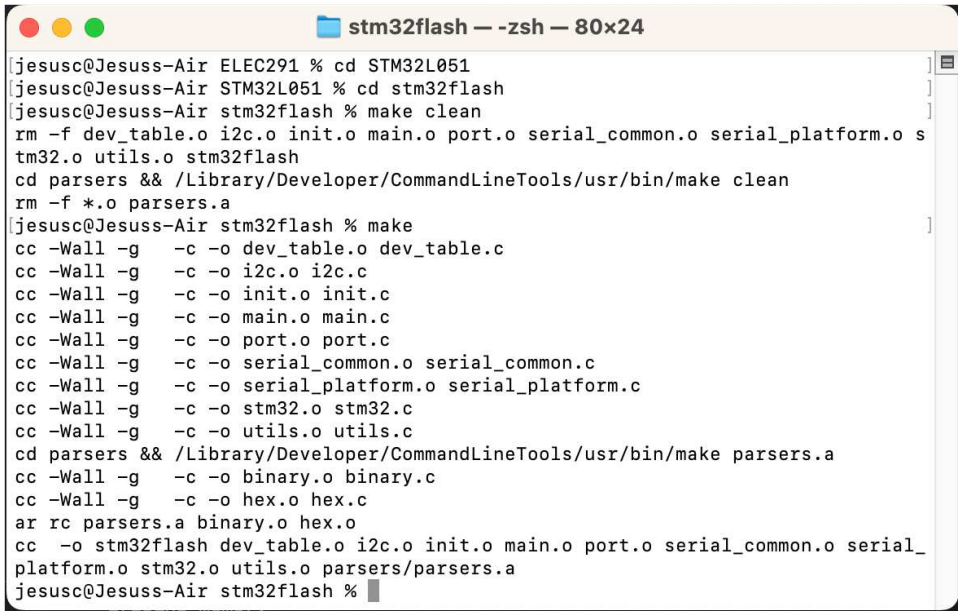
```
$ arm-none-eabi-gcc
```

The program runs and complains that there are no source files to compile (this is a good thing because it means that the compiler is now installed!):



Download the project examples for the STM32L051 from the course web page (Canvas) and unzip it somewhere in the hard drive of your Mac computer. The flash loader included with the zip file for the STM32L051 has to be re-compiled for macOS. In a shell terminal navigate to the directory stm32flash and type 'make clean' followed by 'make'. It looks like this:

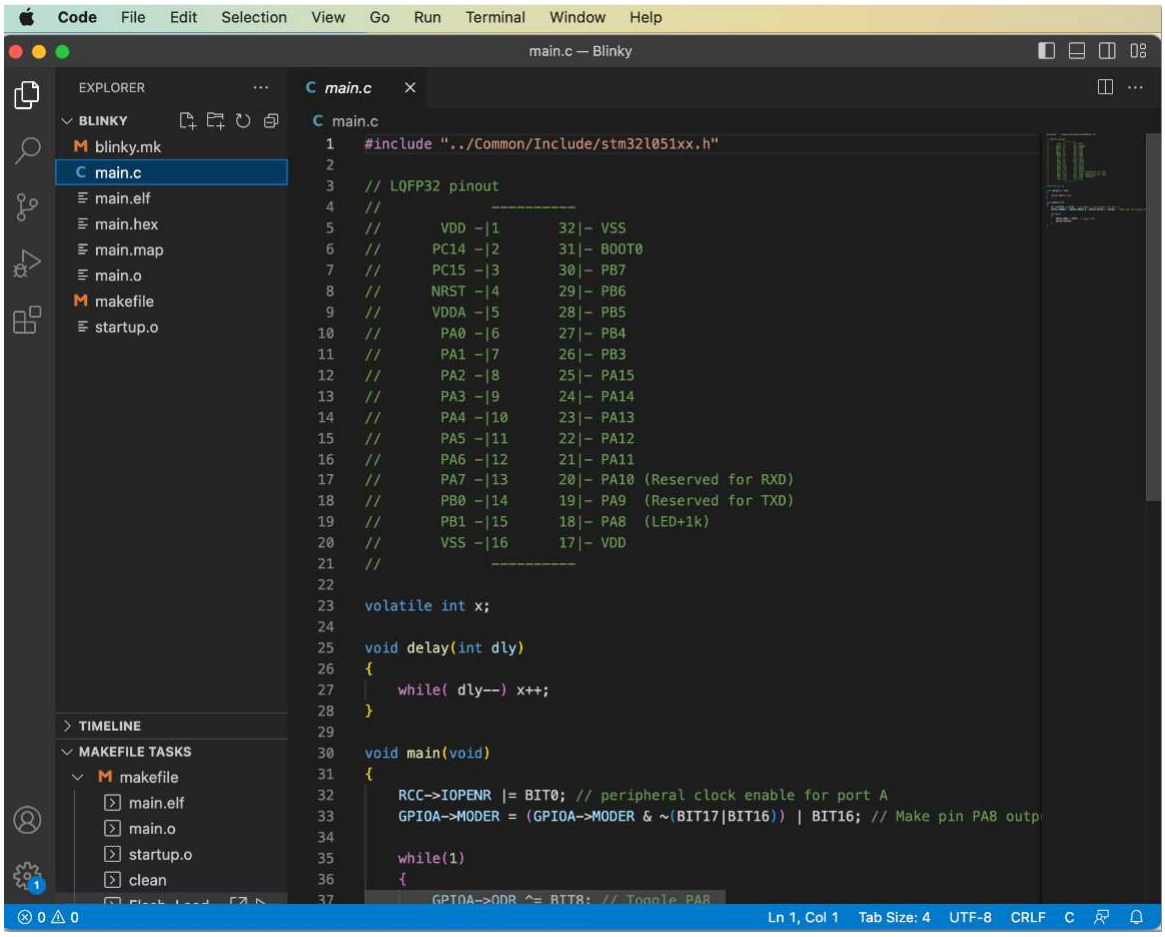
[3] I followed the instructions here
https://dev.to/lewuathe/how-to-achieve-arm-cross-compilation-on-macos-3b08



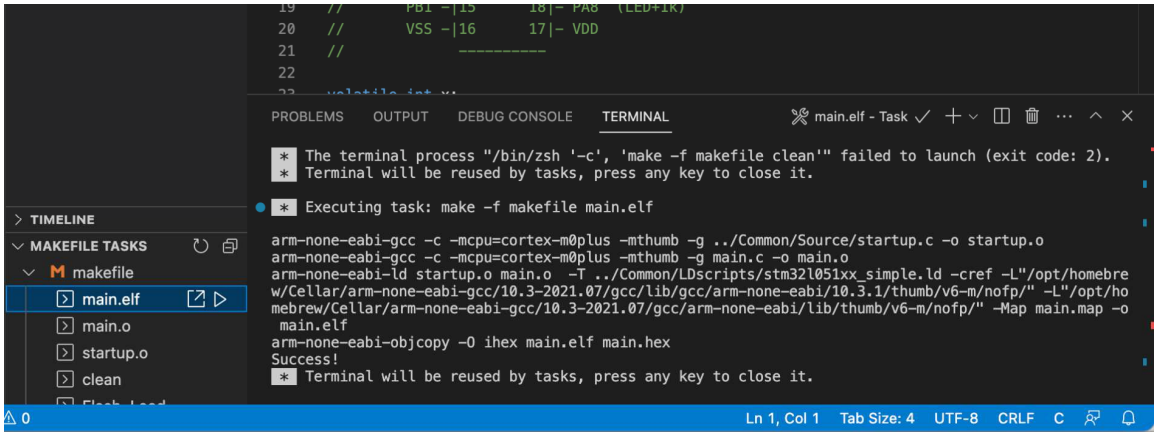
Each STM32L051 project example should include a ‘makefile’ for macOS (if not you can make one based on the ‘makefile’ for Windows). The file is named ‘makefile.mac’. For example, for project ‘Blinky’, the file ‘makefile.mac’ looks like this:

```
CC=arm-none-eabi-gcc
AS=arm-none-eabi-as
LD=arm-none-eabi-ld
PORTN=/dev/$(shell ls /dev | grep "cu.usbserial")

CCFLAGS=-mcpu=cortex-m0plus -mthumb -g
ASFLAGS=-mcpu=cortex-m0plus -mthumb -g
LDFLAGS=-T ../Common/LDscripts/stm32l051xx_simple.ld -cref

OBJS= startup.o main.o

# Search for the path of libraries.
LIBPATH1=$(shell find /opt -name libgcc.a | grep "v6-m" | sed -e "s/libgcc.a//g")
LIBPATH2=$(shell find /opt -name libc_nano.a | grep "v6-m" | sed -e "s/libc_nano.a//g")
LIBSPEC=-L"$(LIBPATH1)" -L"$(LIBPATH2)"

main.elf: $(OBJS)
          $(LD) $(OBJS) $(LDFLAGS) $(LIBSPEC) -Map main.map -o main.elf
          arm-none-eabi-objcopy -O ihex main.elf main.hex
          @echo Success!

main.o: main.c
          $(CC) -c $(CCFLAGS) main.c -o main.o

startup.o: ../Common/Source/startup.c
          $(CC) -c $(CCFLAGS) ../Common/Source/startup.c -o startup.o

clean:
          rm $(OBJS)
          rm main.elf main.hex main.map
          rm *.lst

Flash_Load:
          ../stm32flash/stm32flash -w main.hex -v -g 0x0 $(PORTN)
```

Rename the file ‘makefile.mac’ to ‘makefile’. Open the ‘Blinky’ directory in VS Code. It should look something like this:



To compile and link the program, under 'MAKEFILE TASKS' click 'main.elf' (in some makefiles you'll have to click in 'make.hex'):

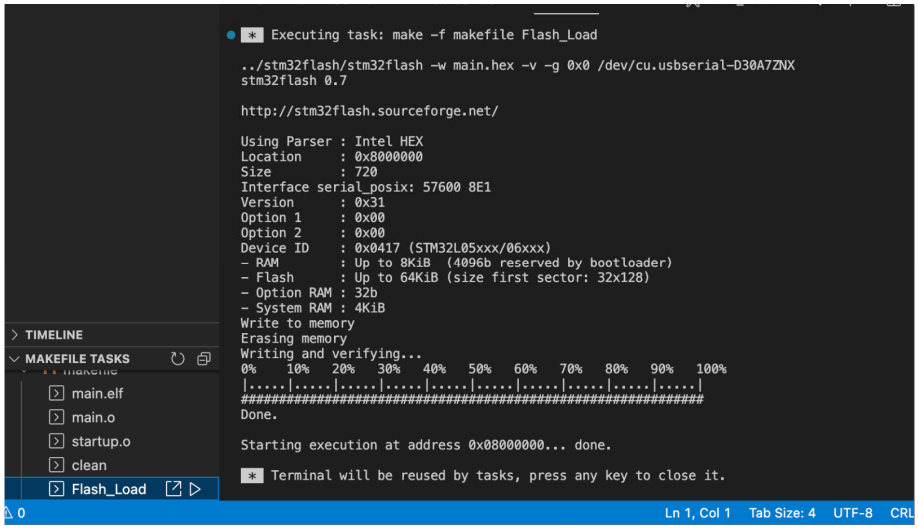


To load the new program into the STM32L051, first you'll need to activate the serial boot loader:

1. Press and hold the BOOT0 push button.
2. Press and release the NRST push button.

3. Release the BOOT0 push button.

Now, under ‘MAKEFILE TASKS’ click ‘LoadFlash’:



That is all. The program should be running in the STM32L051.