



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

Programming the EFM8LB12 on macOS using C

Copyright © 2021-2023, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

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

<https://code.visualstudio.com/Download>

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

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

Install WINE for macOS by typing this in a terminal:

```
brew tap homebrew/cask-versions
brew install --cask --no-quarantine wine-stable
```

Download this file: “EFM8_prog.zip” (It should be available on Canvas) and copy it into your home directory. This is the program we use to load compiled code to the EFM8LB12 microcontroller. In a macOS terminal type:

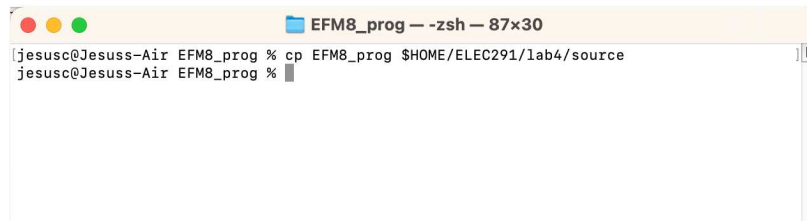
```
unzip EFM8_prog.zip
cd EFM8_prog
make
```

It should look like this:

```
EFM8_prog -- zsh -- 87x30
jesusc@Jesuss-Air ~ % unzip EFM8_prog.zip
Archive:  EFM8_prog.zip
  creating: EFM8_prog/
   inflating: EFM8_prog/EFM8_prog.c
   inflating: EFM8_prog/makefile
jesusc@Jesuss-Air ~ % cd EFM8_prog
jesusc@Jesuss-Air EFM8_prog % ls -l
total 64
-rw-r--r--  1 jesusc  staff   26031 29 Dec 12:44 EFM8_prog.c
-rw-r--r--  1 jesusc  staff    55 16 Feb 08:54 makefile
jesusc@Jesuss-Air EFM8_prog % make
cc      -c -o EFM8_prog.o EFM8_prog.c
gcc -o EFM8_prog EFM8_prog.o
jesusc@Jesuss-Air EFM8_prog %
```

Copy ‘EFM8_prog’ to where your source files for the EFM8LB12 are. For example, I have my files in \$HOME/ELEC291/lab4/source:

```
cp EFM8_prog $HOME/ELEC291/lab4/source
```

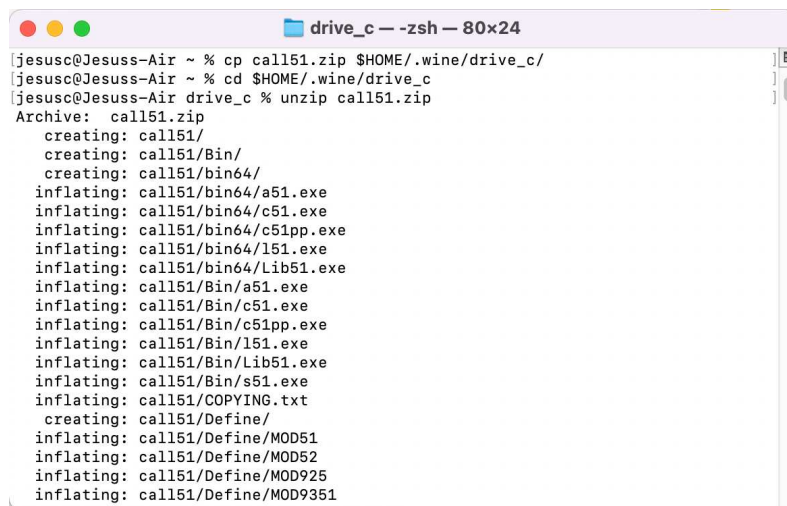
A terminal window titled "EFM8_prog -- zsh -- 87x30". The prompt is "jesusc@Jesuss-Air EFM8_prog %". The command "cp EFM8_prog \$HOME/ELEC291/lab4/source" has been entered and executed. The prompt is now "jesusc@Jesuss-Air EFM8_prog %".

```
jesusc@Jesuss-Air EFM8_prog % cp EFM8_prog $HOME/ELEC291/lab4/source
jesusc@Jesuss-Air EFM8_prog %
```

Download this file: “call51.zip” (it should be available on Canvas) into your home directory. This zip file contains both the ‘C’ and assembly compilers (c51 and a51) for the EFM8LB12 (or any 8051 compatible microcontroller for that matter). Now copy the file to the WINE ‘C:’ drive, and expand it by typing:

```
cp $HOME/call51.zip $HOME/.wine/drive_c/
cd $HOME/.wine/drive_c/
unzip call51.zip
```

It should look similar to image below. Note: I forgot to put \$HOME/ before call51.zip, but it worked anyhow because the working directory was already set to \$HOME.

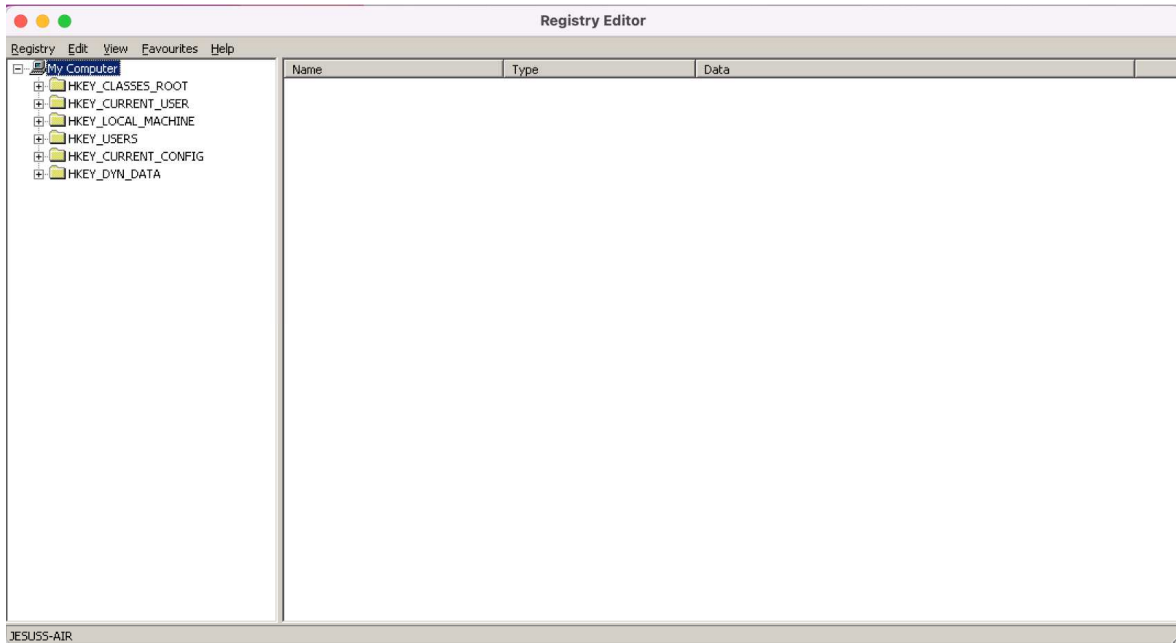
A terminal window titled "drive_c -- zsh -- 80x24". The prompt is "jesusc@Jesuss-Air ~ %". The command "cp call51.zip \$HOME/.wine/drive_c/" has been entered and executed. The prompt is now "jesusc@Jesuss-Air ~ %". The next command "cd \$HOME/.wine/drive_c" has been entered and executed. The prompt is now "jesusc@Jesuss-Air drive_c %". The final command "unzip call51.zip" has been entered and executed. The output shows the files being created and inflated.

```
jesusc@Jesuss-Air ~ % cp call51.zip $HOME/.wine/drive_c/
jesusc@Jesuss-Air ~ % cd $HOME/.wine/drive_c
jesusc@Jesuss-Air drive_c % unzip call51.zip
Archive:  call51.zip
  creating: call51/
  creating: call51/Bin/
  creating: call51/bin64/
  inflating: call51/bin64/a51.exe
  inflating: call51/bin64/c51.exe
  inflating: call51/bin64/c51pp.exe
  inflating: call51/bin64/l51.exe
  inflating: call51/bin64/lib51.exe
  inflating: call51/Bin/a51.exe
  inflating: call51/Bin/c51.exe
  inflating: call51/Bin/c51pp.exe
  inflating: call51/Bin/l51.exe
  inflating: call51/Bin/lib51.exe
  inflating: call51/Bin/s51.exe
  inflating: call51/COPYING.txt
  creating: call51/Define/
  inflating: call51/Define/MOD51
  inflating: call51/Define/MOD52
  inflating: call51/Define/MOD925
  inflating: call51/Define/MOD9351
```

Now we need to configure the PATH in WINE, so it can always find the compilers just copied. In a terminal window type:

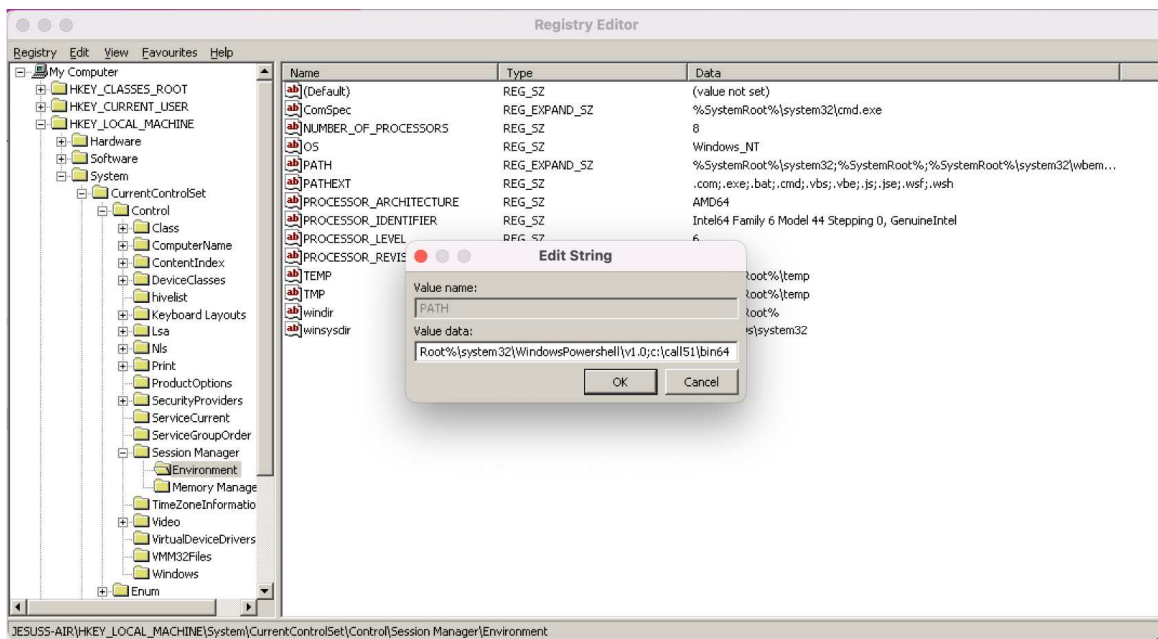
```
wine64 regedit
```

A window opens that looks like this:



Navigate to the 'PATH' environment variable by clicking: HKEY_LOCAL_MACHINE -> SYSTEM -> CurrentControlSet -> Control -> Session Manager -> Environment

Double click 'PATH' and add 'C:\call51\bin64' at the very end:



Click 'OK' and close the Registry Editor.

To compile using the 8051 assembler, in the terminal type:

```
wine64 a51 -l Blinky_EFM8.asm
```

```
source — C:\windows\system32\cmd.exe — -zsh — 104x30
jesusc@Jesuss-Air source % wine64 a51 Blinky_EFM8.asm
preloader: Warning: failed to reserve range 000000000010000-0000000000110000
preloader: Warning: failed to reserve range 000000000010000-0000000000110000
preloader: Warning: failed to reserve range 000000000010000-0000000000110000
preloader: Warning: failed to reserve range 000000000010000-0000000000110000
preloader: Warning: failed to reserve range 000000000010000-0000000000110000
preloader: Warning: failed to reserve range 000000000010000-0000000000110000
preloader: Warning: failed to reserve range 000000000010000-0000000000110000
preloader: Warning: failed to reserve range 000000000010000-0000000000110000
preloader: Warning: failed to reserve range 000000000010000-0000000000110000
preloader: Warning: failed to reserve range 000000000010000-0000000000110000
preloader: Warning: failed to reserve range 000000000010000-0000000000110000
00c4:fixme:heap:RtlSetHeapInformation 00000000006D0000 0 000000000021FE00 4 stub
No errors found
jesusc@Jesuss-Air source %
```

NOTE: the message ‘00c4:fixme:heap:....’ and ‘preloader:...’ come from WINE and have nothing to do with a51 or your code. They can be safely ignored. These messages can be suppressed from the output by running the compiler using this command line:

```
wine64 a51 -l Blinky_EFM8.asm 2>&1 | grep -v preloader | grep -v fixme
```

The output of the command above then looks like this:

```
source — C:\windows\system32\cmd.exe — -zsh — 104x30
jesusc@Jesuss-Air source % wine64 a51 Blinky_EFM8.asm 2>&1 | grep -v preloader | grep -v fixme
No errors found
jesusc@Jesuss-Air source %
```

To load to the EFM8LB12 microcontroller, first press and hold the BOOT push button, press and release the RESET button, then release the BOOT button. The EFM8LB12 enters boot mode. We need to find the name of the port macOS assigned to the BO230XS USB adapter. In the terminal type:

```
ls -l /dev/ | grep "usb"
```

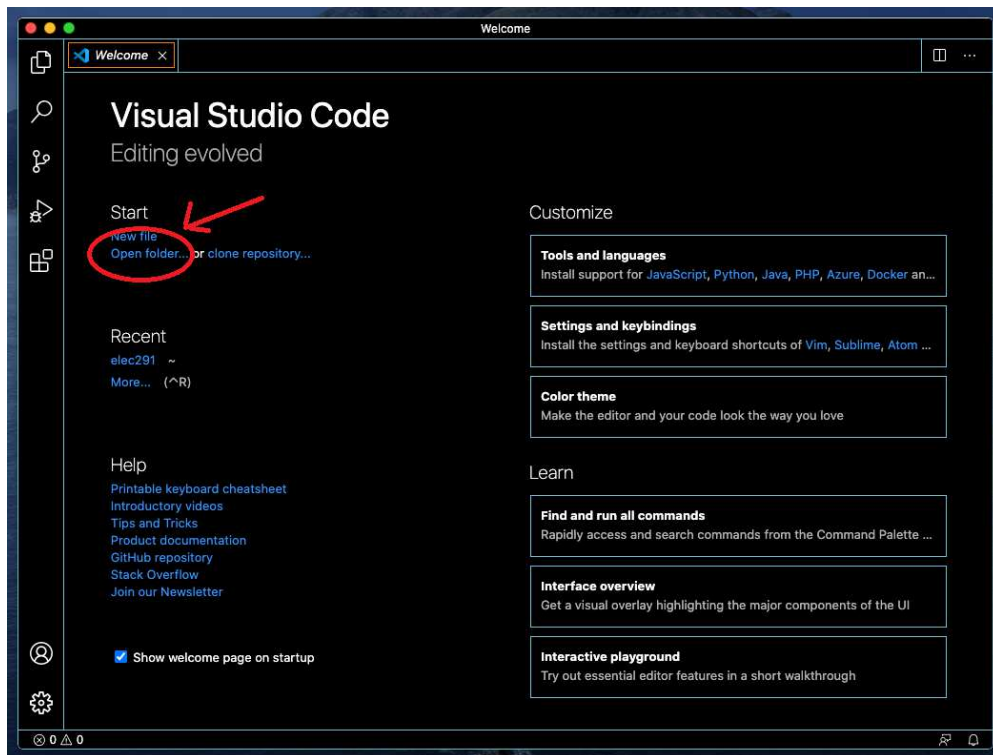
That will print the two names of the usb-to-serial port assigned by macOS for the BO230XS adapter. The names are different for different BO230XS adapters. In my computer the first name returned was **cu.usbserial-DN05C8LH** as shown below. Now run the ‘EFM8_prog’ loader in order to send the compiled object of the program to the EFM8LB12 microcontroller (notice the preceding "-p/dev/"):

```
./EFM8_prog -r -p/dev/cu.usbserial-DN05C8LH Blinky_EFM8.hex
```

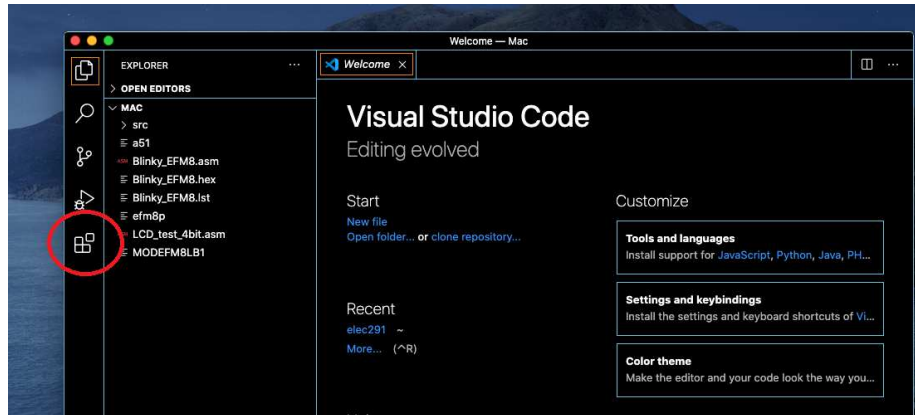
```
source -- C:\windows\system32\cmd.exe -- -zsh -- 108x34
jesusc@Jesuss-Air source % wine64 a51 Blinky_EFM8.asm 2>&1 | grep -v preloader | grep -v fixme
No errors found
jesusc@Jesuss-Air source % ./EFM8_prog
Serial flash programmer for the EFM8LB1. (C) Jesus Calvino-Fraga (2012-2022)
Usage example:
./EFM8_prog -p/dev/cu.usbserial-DN05FVT8 somefile.hex
Options available:
-p[portname] (use this serial port)
-ttx (boot loader activation timeout is xx seconds)
-r (reset processor after loading flash)
-h (this help)
To check what serial ports are available on a Mac terminal type:
ls -l /dev | grep "usb"
jesusc@Jesuss-Air source % ls -l /dev | grep "usb"
crw-rw-rw-  1 root  wheel   0x9000005 16 Feb 09:19 cu.usbserial-DN05C8LH
crw-rw-rw-  1 root  wheel   0x9000004 16 Feb 09:19 tty.usbserial-DN05C8LH
jesusc@Jesuss-Air source % ./EFM8_prog -r -p/dev/cu.usbserial-DN05C8LH Blinky_EFM8.hex
Serial flash programmer for the EFM8LB1. (C) Jesus Calvino-Fraga (2012-2022)
Blinky_EFM8.hex: loaded 44 bytes
.Found EFM8LB12F64E_QFP32. Id: 0x3442
Sending 'setup' command... Done.
Erasing flash memory...
#####
##### Done.
Writing flash memory...
# Done.
Verifying... Done.
Running program... Done.
Actions completed in 2 seconds.
jesusc@Jesuss-Air source %
```

These tasks can be also performed in the terminal of Visual Studio Code, while having access to editor for the source code. For convenience, you should first install syntax highlighting for the 8051 assembler in VS Code:

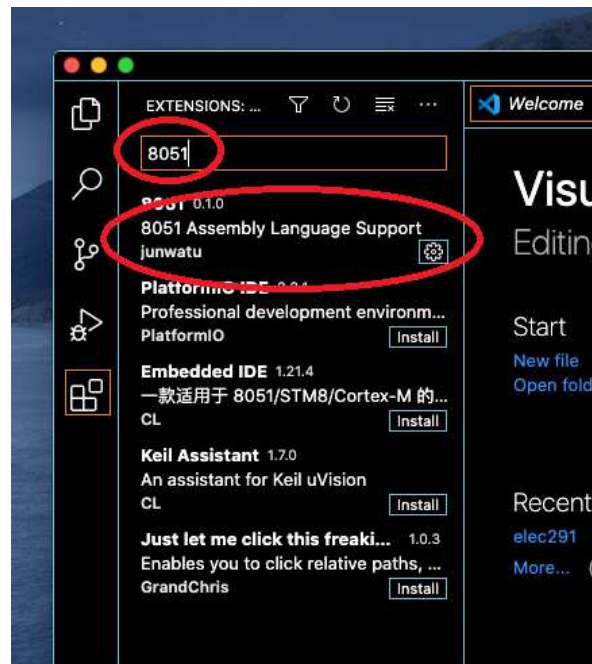
Run VS code, and click 'Open folder...' and select the directory with your source files:



To install syntax highlighting for the 8051, click the extension button:



Type "8051" and pick the first one. The others may work also but I hadn't tried.

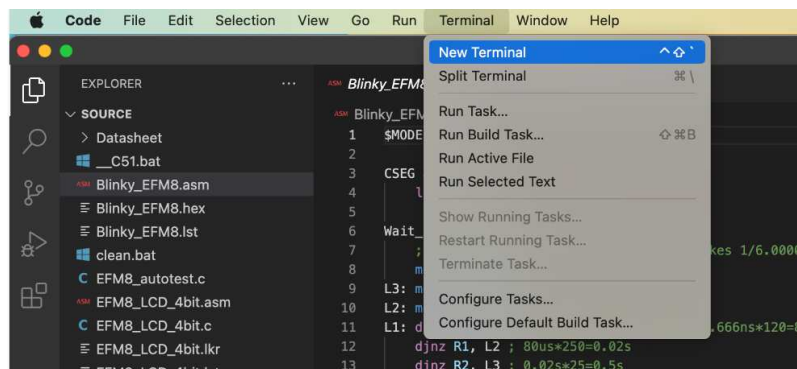


Click the explorer button and the double click the ".asm" file you want to edit. The file opens and we can make changes:

```

1  $MODEFM8LB1
2
3  CSEG at 0H
4      ljmp main
5
6  Wait_half_second:
7      ;For a 6MHz clock one machine cycle takes 1/6.000MHz=166.666ns
8      mov R7, #25
9      L3: mov R1, #250
10     L2: mov R0, #120
11     L1: djnz R0, L1 ; 4 machine cycles-> 4*166.666ns*120=80us
12     djnz R1, L2 ; 80us*250=0.02s
13     djnz R2, L3 ; 0.02s*25=0.5s
14     ret
15
16 main:
17     ; DISABLE WDT: provide Watchdog disable keys
18     mov WDTCON, #0x0E ; First key
19     mov WDTCON, #0xAD ; Second key
20
21     mov SP, #7FH
22     ; Enable crossbar and weak pull-ups
23     mov XBR0, #0x00
24     mov XBR1, #0x00
25     mov XBR2, #0x00
26     mov P2MDOUT, #0x02 ; make LED pin (P2.1) output push-pull
27
28 M0:
29     cpl P2.1 ; Led off/on
30     lcall Wait_half_second
31     sjmp M0
32 end
  
```

To compile click the "Terminal" menu entry, and them "New Terminal":

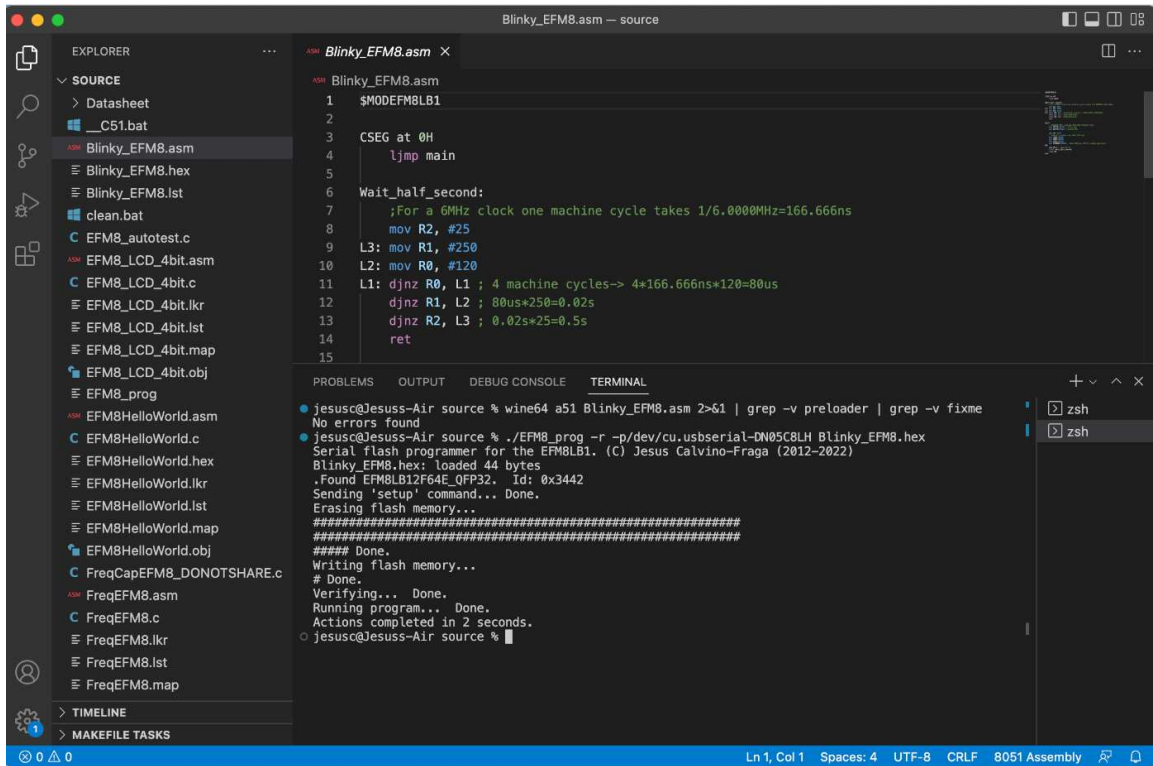


A terminal shell opens inside VSCode. From there we can compile the source code and load the memory of the EFM8LB12 microcontroller using the generated “hex” file. Let us compile first as before:

```
wine64 a51 -l Blinky_EFM8.asm 2>&1 | grep -v preloader | grep -v fixme
```

Before loading, we need to activate the boot-loader: first press and hold the BOOT push button, press and release the RESET button, then release the BOOT button. Then call the loader program as before (I had to re-adjust the panes sizes to show the whole output text, but you don’t really have to do that):

```
./EFM8_prog -r -p/dev/cu.usbserial-DN05C8LH Blinky_EFM8.hex
```

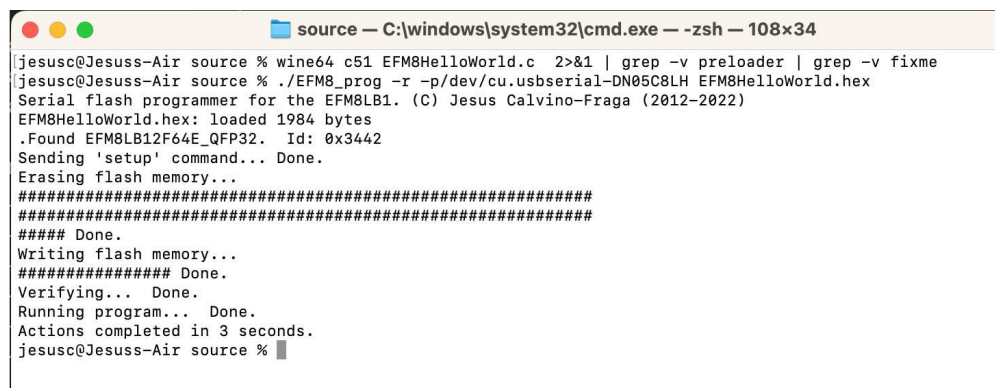



To compile a 'C' file and load the resulting 'hex' file, the steps are almost identical to what was done already with the assembler compiler. In a terminal type:

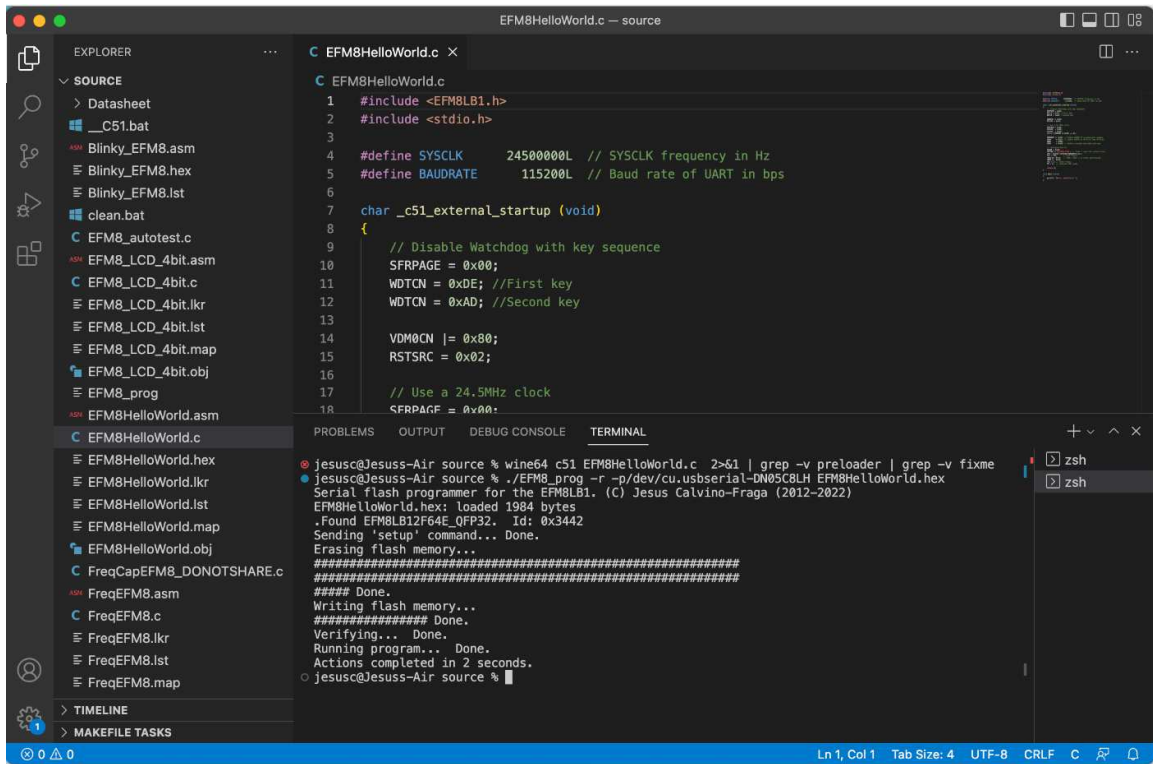
```
wine64 c51 EFM8HelloWorld.c 2>&1 | grep -v preloader | grep -v fixme
```

followed by:

```
./EFM8_prog -r -p/dev/cu.usbserial-DN05C8LH EFM8HelloWorld.hex
```



We can do something similar from inside the terminal of VS code:



Since compiling and loading are very repetitive tasks, you should consider writing a couple of shell scripts to speed up the process. These two scripts can be used to quickly compile the source “asm” or “C” files and load the resulting “hex” file to the EFM8LB1 microcontroller:

Script 1, for compiling with a51:

```
#!/bin/bash
prompt="Please select a file to compile:"
options=( $(find *.asm | xargs -0) )

PS3="$prompt "
echo ""
select opt in "${options[@]}" "Quit" ; do
    if (( REPLY == 1 + ${#options[@]} )) ; then
        exit

    elif (( REPLY > 0 && REPLY <= ${#options[@]} )) ; then
        echo "Compiling $opt"
        wine64 a51 -l $opt 2>&1 | grep -v preloader | grep -v fixme
        break

    else
        echo "Invalid option. Try another one."
    fi
done
echo ""
```

Script 2, for loading:

```
#!/bin/bash
prompt="Select usb to serial port:"
options=( $(find /dev/*usbserial* | xargs -0) )

PS3="$prompt "
echo ""
select opt1 in "${options[@]}" "Quit" ; do
    if (( REPLY == 1 + ${#options[@]} )) ; then
        exit

        elif (( REPLY > 0 && REPLY <= ${#options[@]} )) ; then
            echo "Selected usb to serial: $opt1"
            break

        else
            echo "Invalid option. Try another one."
        fi
done

prompt="Select hex file:"
options=( $(find *.hex | xargs -0) )

PS3="$prompt "
echo ""
select opt2 in "${options[@]}" "Quit" ; do
    if (( REPLY == 1 + ${#options[@]} )) ; then
        exit

        elif (( REPLY > 0 && REPLY <= ${#options[@]} )) ; then
            echo "Selected hex file: $opt2"
            ./EFM8_prog -r -p$opt1 $opt2
            break

        else
            echo "Invalid option. Try another one."
        fi
done

echo ""
```

Script 3, for compiling with c51:

```
#!/bin/bash
prompt="Please select a file to compile:"
options=( $(find *.c | xargs -0) )

PS3="$prompt "
echo ""
select opt in "${options[@]}" "Quit" ; do
    if (( REPLY == 1 + ${#options[@]} )) ; then
        exit

        elif (( REPLY > 0 && REPLY <= ${#options[@]} )) ; then
            echo "Compiling $opt"
            wine64 c51 -l $opt 2>&1 | grep -v preloader | grep -v fixme
            break

        else
            echo "Invalid option. Try another one."
        fi
done
echo ""
```

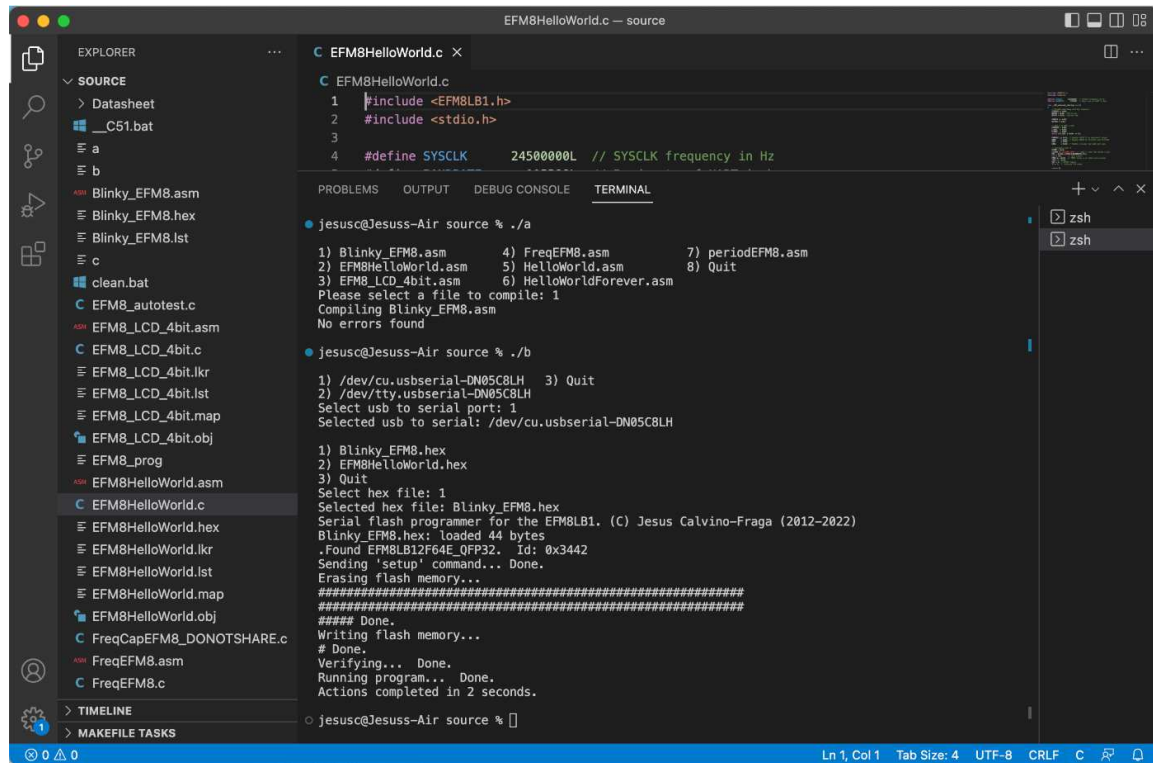
I named the scripts “a”, “b”, and “c”. You’ll need to change the mode of the scripts so they can be executed:

```
chmod a+x a
```

```
chmod a+x b
```

```
chmod a+x c
```

This is how the scripts look like when called from VScode. To compile ‘Blinky_EFM8.asm’ with ‘a51’ and load ‘Blinky_EFM8.hex’ with ‘EFM8_prog’:



The screenshot shows the VS Code interface with the file explorer on the left, the source code editor in the center, and the terminal on the right. The source code is for EFM8HelloWorld.c, which includes EFM8LB1.h and stdio.h, and defines the SYSCLK frequency as 24500000L Hz. The terminal shows the execution of the 'a' script, which lists the files to be compiled and the 'b' script, which lists the files to be loaded and the serial port to be used. The terminal output shows the successful compilation of Blinky_EFM8.asm and the successful flashing of Blinky_EFM8.hex to the EFM8LB1 microcontroller.

```
EFM8HelloWorld.c -- source
C EFM8HelloWorld.c
1 #include <EFM8LB1.h>
2 #include <stdio.h>
3
4 #define SYSCLK 24500000L // SYSCLK frequency in Hz

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
jesusc@Jesus-Air source % ./a
1) Blinky_EFM8.asm 4) FreqEFM8.asm 7) periodEFM8.asm
2) EFM8HelloWorld.asm 5) HelloWorld.asm 8) Quit
3) EFM8_LCD_4bit.asm 6) HelloWorldForever.asm
Please select a file to compile: 1
Compiling Blinky_EFM8.asm
No errors found

jesusc@Jesus-Air source % ./b
1) /dev/cu.usbserial-DN05C8LH 3) Quit
2) /dev/tty.usbserial-DN05C8LH
Select usb to serial port: 1
Selected usb to serial: /dev/cu.usbserial-DN05C8LH

1) Blinky_EFM8.hex
2) EFM8HelloWorld.hex
3) Quit
Select hex file: 1
Selected hex file: Blinky_EFM8.hex
Serial flash programmer for the EFM8LB1. (C) Jesus Calvino-Fraga (2012-2022)
Blinky_EFM8.hex: loaded 44 bytes
.Found EFM8LB12F64E_QFP32. Id: 0x3442
Sending 'setup' command... Done.
Erasing flash memory...
#####
#### Done.
Writing flash memory...
# Done.
Verifying... Done.
Running program... Done.
Actions completed in 2 seconds.

jesusc@Jesus-Air source %
```

Similarly to compile 'EFM8HelloWorld.c' with 'c51' and load 'EFM8HelloWorld.hex' with 'EFM8_prog':

The screenshot shows an IDE window titled 'EFM8HelloWorld.c — source'. The left sidebar contains an 'EXPLORER' panel with a file tree under 'SOURCE'. The main editor displays the source code for 'EFM8HelloWorld.c', which includes headers for 'EFM8LB1.h' and 'stdio.h', and a definition for 'SYSCLK' as 24500000L. The 'TERMINAL' panel at the bottom shows the execution of the 'c51' compiler and the 'EFM8_prog' programmer. The compiler output shows the compilation of 'EFM8HelloWorld.c' into 'EFM8HelloWorld.hex'. The programmer output shows the selection of the hex file, the loading of the file (1984 bytes), the erasing of flash memory, and the successful writing and verification of the program. The status bar at the bottom indicates 'Ln 1, Col 1', 'Tab Size: 4', 'UTF-8', 'CRLF', and 'C'.

```
EFM8HelloWorld.c
1  #include <EFM8LB1.h>
2  #include <stdio.h>
3
4  #define SYSCLK      24500000L  // SYSCLK frequency in Hz

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

• jesus@Jesuss-Air source % ./c
1) EFM8HelloWorld.c      6) HelloWorld.c
2) EFM8_LCD_4bit.c      7) HelloWorldForever.c
3) EFM8_autotest.c      8) PeriodEFM8.c
4) FreqCapEFM8_DONOTSHARE.c  9) Quit
5) FreqEFM8.c
Please select a file to compile: 1
Compiling EFM8HelloWorld.c

• jesus@Jesuss-Air source % ./b
1) /dev/cu.usbserial-DN05C8LH  3) Quit
2) /dev/tty.usbserial-DN05C8LH
Select usb to serial port: 1
Selected usb to serial: /dev/cu.usbserial-DN05C8LH

1) Blinky_EFM8.hex
2) EFM8HelloWorld.hex
3) Quit
Select hex file: 2
Selected hex file: EFM8HelloWorld.hex
Serial flash programmer for the EFM8LB1. (C) Jesus Calvino-Fraga (2012-2022)
EFM8HelloWorld.hex: loaded 1984 bytes
.Found EFM8LB12F64E QFP32. Id: 0x3442
Sending 'setup' command... Done.
Erasing flash memory...
##### Done.
##### Writing flash memory...
##### Done.
Verifying... Done.
Running program... Done.
Actions completed in 3 seconds.

• jesus@Jesuss-Air source %
```