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

Programming the EFM8LB12 on macOS using C

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

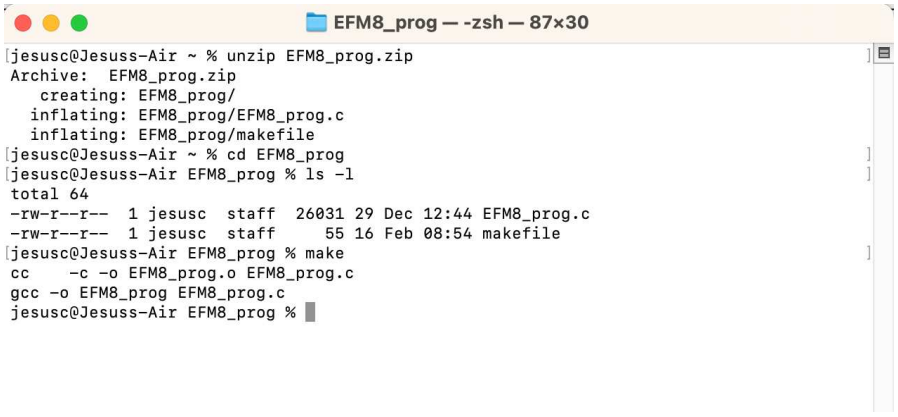
Install WINE for macOS by typing this in a terminal:

```
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:

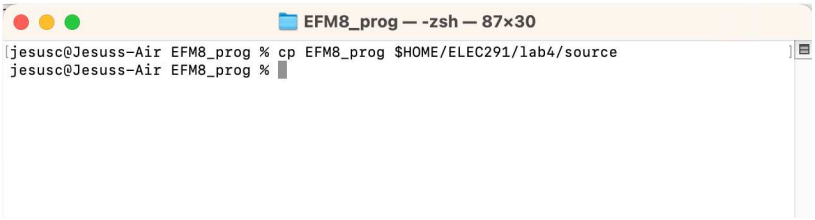
```
unzip EFM8_prog.zip
cd EFM8_prog
make
```

It should look like this:



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

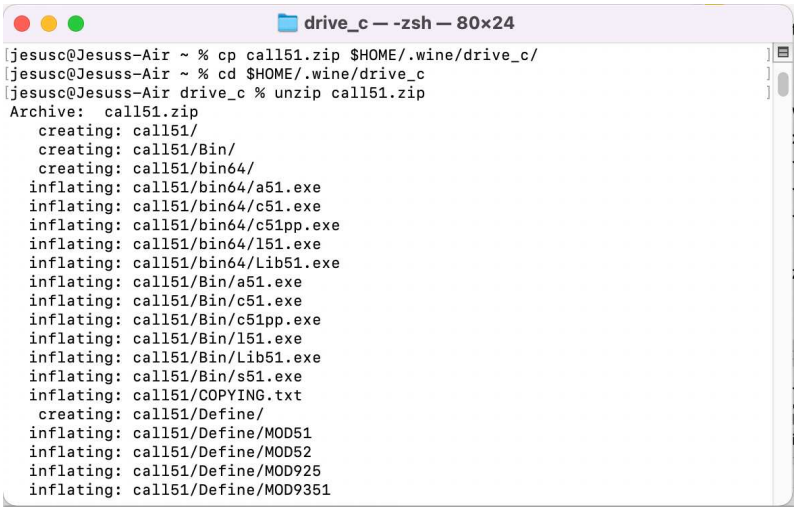
```
cp EFM8_prog $HOME/ELEC291/lab4/source
```



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
```

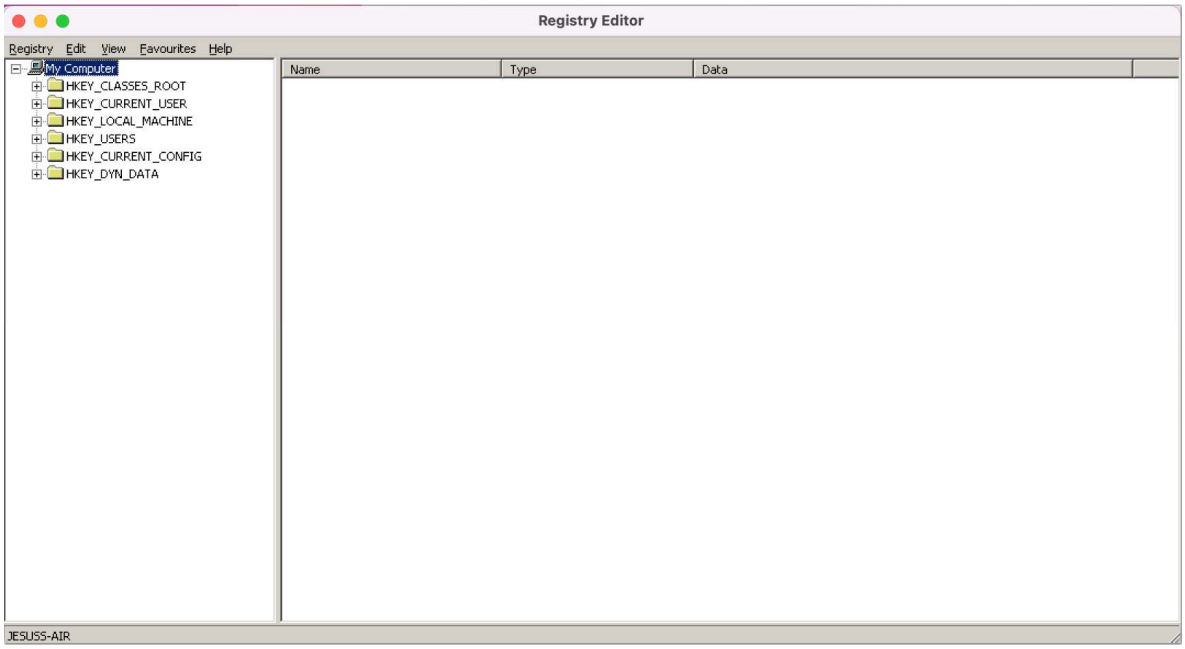
It should look similar to image below. Note: I forgot to put $HOME/ before call51.zip, but it worked anyhow because the working directory was already set to $HOME.



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

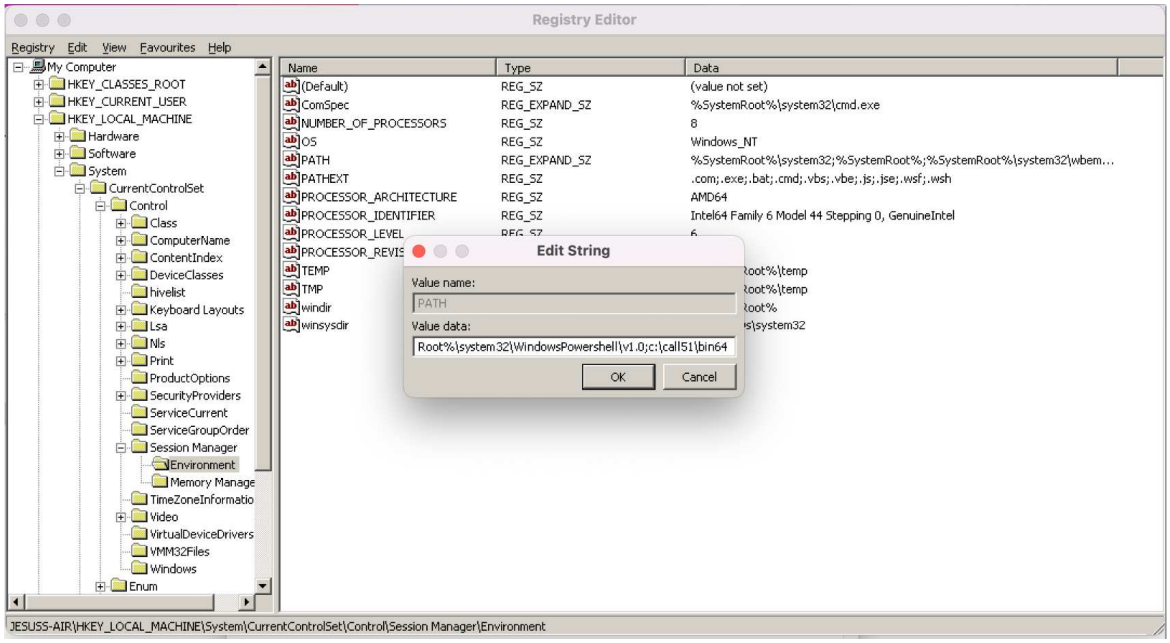
```
wine64 regedit
```

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 — 104×30
jesusc@Jesuss-Air source % wine64 a51 Blinky_EFM8.asm
preloader: Warning: failed to reserve range 0000000000010000-0000000000110000
preloader: Warning: failed to reserve range 0000000000010000-0000000000110000
preloader: Warning: failed to reserve range 0000000000010000-0000000000110000
preloader: Warning: failed to reserve range 0000000000010000-0000000000110000
preloader: Warning: failed to reserve range 0000000000010000-0000000000110000
preloader: Warning: failed to reserve range 0000000000010000-0000000000110000
preloader: Warning: failed to reserve range 0000000000010000-0000000000110000
preloader: Warning: failed to reserve range 0000000000010000-0000000000110000
preloader: Warning: failed to reserve range 0000000000010000-0000000000110000
preloader: Warning: failed to reserve range 0000000000010000-0000000000110000
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 — 104×30
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"
```

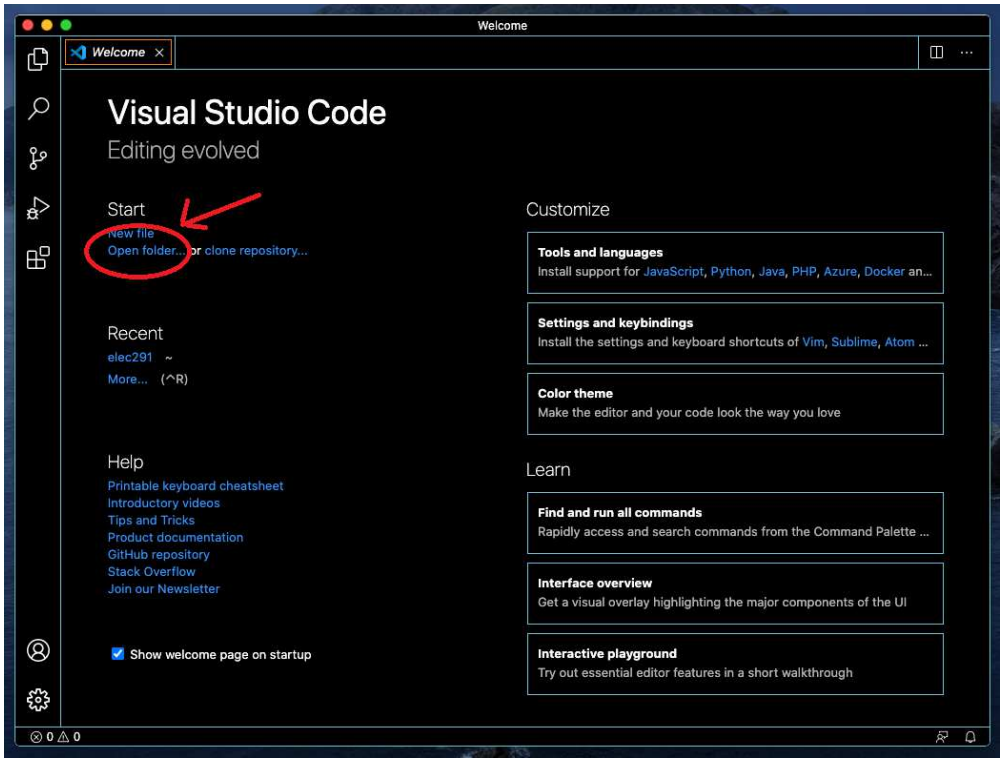
That will print the two names of the usb-to-serial port assigned by macOS for the BO230XS adapter. The names are different for different BO230XS adapters. In my computer the first name returned was **cu.usbserial-DN05C8LH** as shown below**.** Now run the ‘EFM8_prog’ loader in order to send the compiled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

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

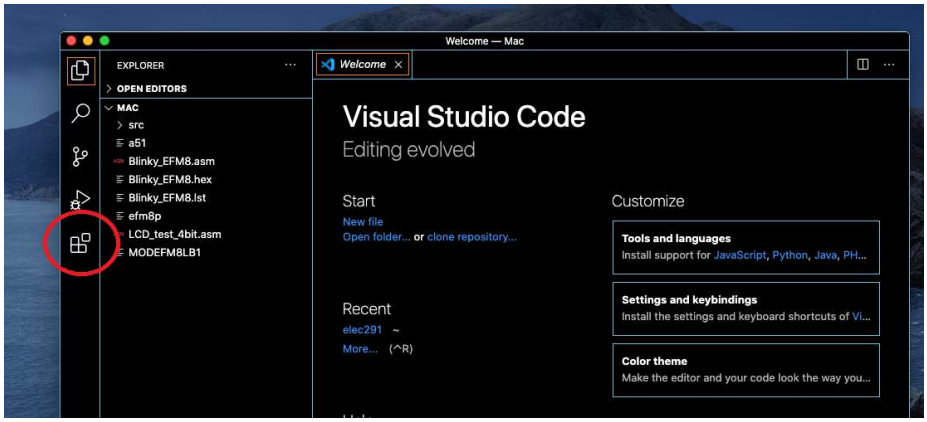
```
source — C:\windows\system32\cmd.exe — -zsh — 108×34
jesusc@Jesuss-Air source % wine64 a51 Blinky_EFM8.asm 2>&1 | grep -v preloader | grep -v fixme
 No errors found
jesusc@Jesuss-Air source % ./EFM8_prog
Serial flash programmer for the EFM8LB1. (C) Jesus Calvino-Fraga (2012-2022)
Usage example:
./EFM8_prog -p/dev/cu.usbserial-DN05FVT8 somefile.hex
Options available:
   -p[portname] (use this serial port)
   -txx (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:

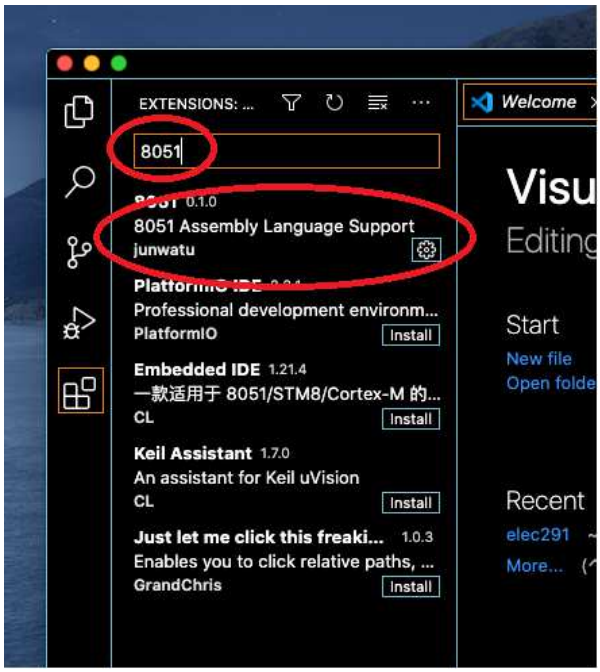
Run VS code, and click 'Open folder..." and select the directory with your source files:



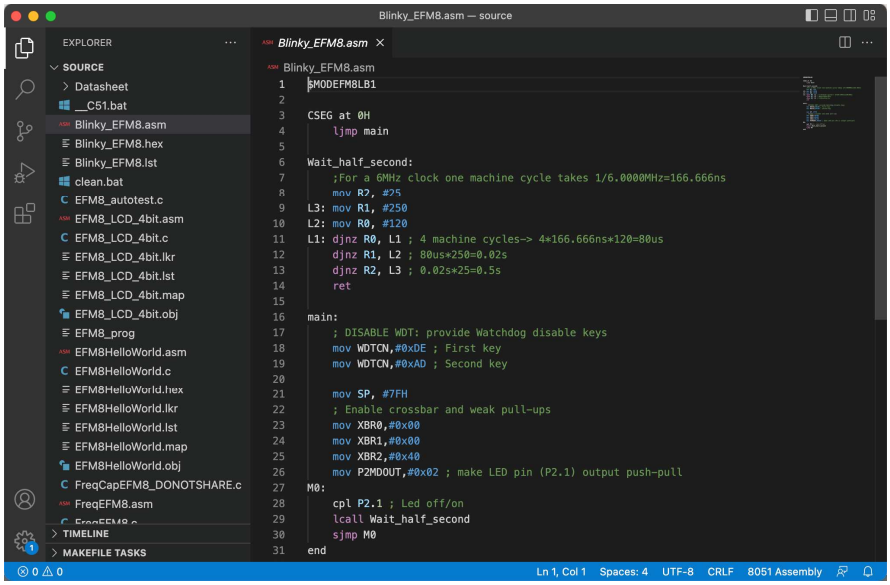
To install syntax highlighting for the 8051, click the extension button:



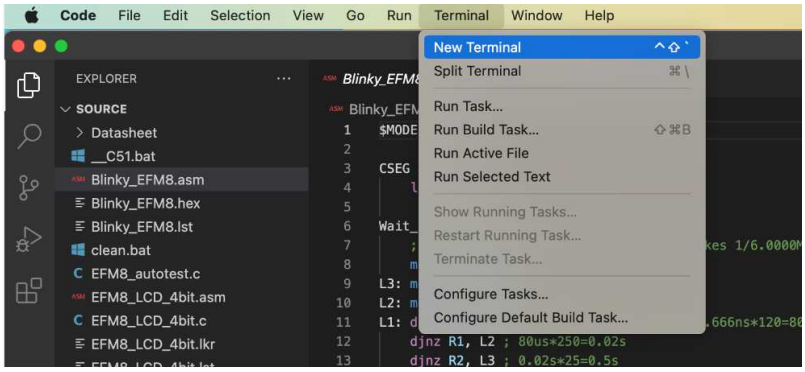
Type "8051" and pick the first one. The others may work also but I hadn't tried.



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



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

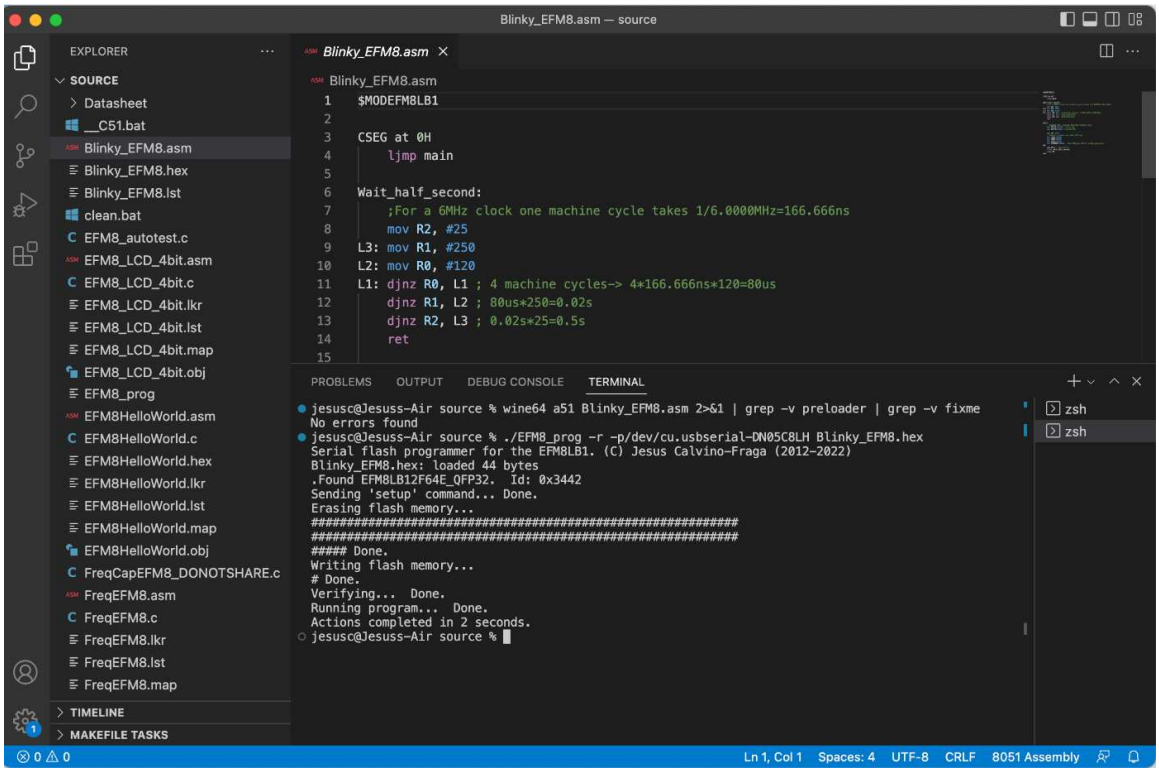


A terminal shell opens inside VScode. From there we can compile the source code and load the memory of the EFM8LB12 microcontroller using the generated “hex” file. Let us compile first as before:

```
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
```

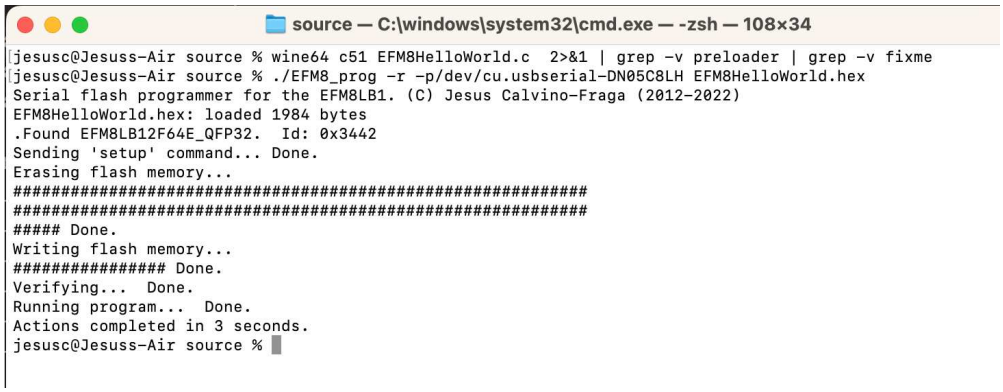


To compile a 'C' file and load the resulting 'hex' file, the steps are almost identical to what was done already with the assembler compiler. In a terminal type:

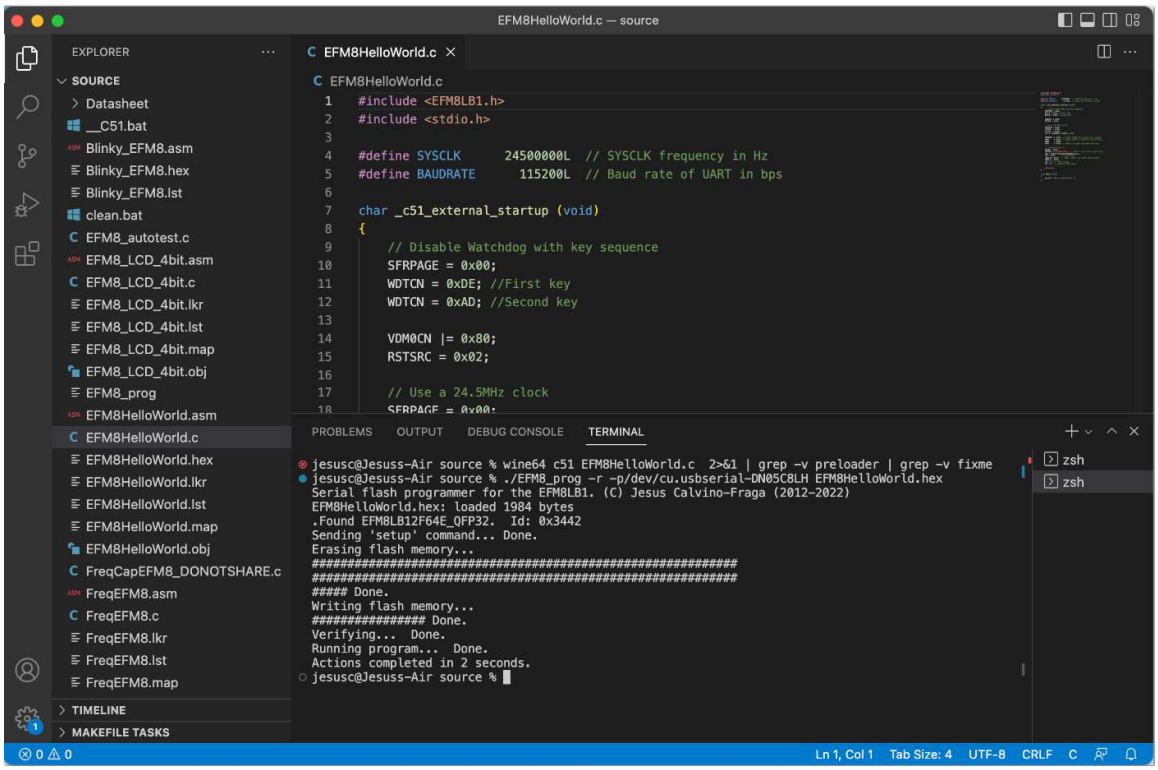
```
wine64 c51 EFM8HelloWorld.c  2>&1 | grep -v preloader | grep -v fixme
```

followed by:

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



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


EFM8HelloWorld.c — source
EXPLORER
SOURCE
Datasheet
__C51.bat
Blinky_EFM8.asm
Blinky_EFM8.hex
Blinky_EFM8.lst
clean.bat
EFM8_autotest.c
EFM8_LCD_4bit.asm
EFM8_LCD_4bit.c
EFM8_LCD_4bit.lkr
EFM8_LCD_4bit.lst
EFM8_LCD_4bit.map
EFM8_LCD_4bit.obj
EFM8_prog
EFM8HelloWorld.asm
EFM8HelloWorld.c
EFM8HelloWorld.hex
EFM8HelloWorld.lkr
EFM8HelloWorld.lst
EFM8HelloWorld.map
EFM8HelloWorld.obj
FreqCapEFM8_DONOTSHARE.c
FreqEFM8.asm
FreqEFM8.c
FreqEFM8.lkr
FreqEFM8.lst
FreqEFM8.map
TIMELINE
MAKEFILE TASKS
EFM8HelloWorld.c
#include <EFM8LB1.h>
#include <stdio.h>
#define SYSCLK 24500000L // SYSCLK frequency in Hz
#define BAUDRATE 115200L // Baud rate of UART in bps
char _c51_external_startup (void)
{
// Disable Watchdog with key sequence
SFRPAGE = 0x00;
WDTCN = 0xDE; //First key
WDTCN = 0xAD; //Second key
VDM0CN |= 0x80;
RSTSRC = 0x02;
// Use a 24.5MHz clock
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
zsh
zsh
jesusc@Jesuss-Air source % wine64 c51 EFM8HelloWorld.c 2>&1 | grep -v preloader | grep -v fixme
jesusc@Jesuss-Air source % ./EFM8_prog -r -p/dev/cu.usbserial-DN05C8LH 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...
Writing flash memory...
Done.
Verifying... Done.
Running program... Done.
Actions completed in 2 seconds.
jesusc@Jesuss-Air source %
Ln 1, Col 1
Tab Size: 4
UTF-8
CRLF
C

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 ""
```

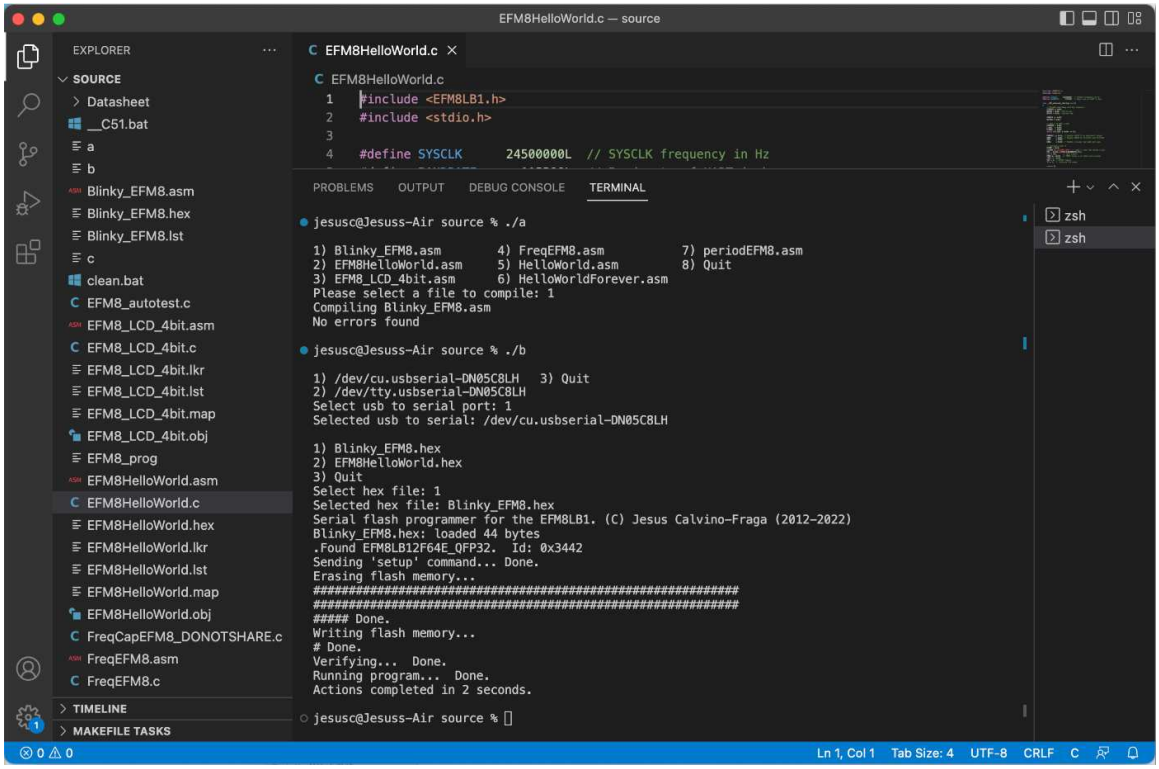
Script 3, for compiling with c51:

```
#!/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’:



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