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

The EFM8 Microcontroller System

Introduction

This document introduces the EFM8 microcontroller system using Silicon Lab's EFM8LB1 microcontroller. The EFM8LB1 IC is an 8-bit 8051 compatible microcontroller, 72MHz, 64KB Flash, 4kB of RAM, 14-bit ADC, and 4 channel 12-bit DAC in a QFP32 package.

Recommended documentation

EFM8LB1 Family Reference Manual:
(*https://www.silabs.com/documents/public/reference-manuals/EFM8LB1-RM.pdf*)

EFM8LB1 Family Datasheet:
(*https://www.silabs.com/documents/public/data-sheets/efm8lb1-datasheet.pdf*)

Assembling the Microcontroller System

Figure 1 shows the circuit schematic of the EFM8 board microcontroller system used in ELEC291/ELEC292. The EFM8 board was assembled as part of project 1 in ELEC291/292 using surface mount components with the reflow oven controller. Figure 2 shows the EFM8 plugged in a bread board.

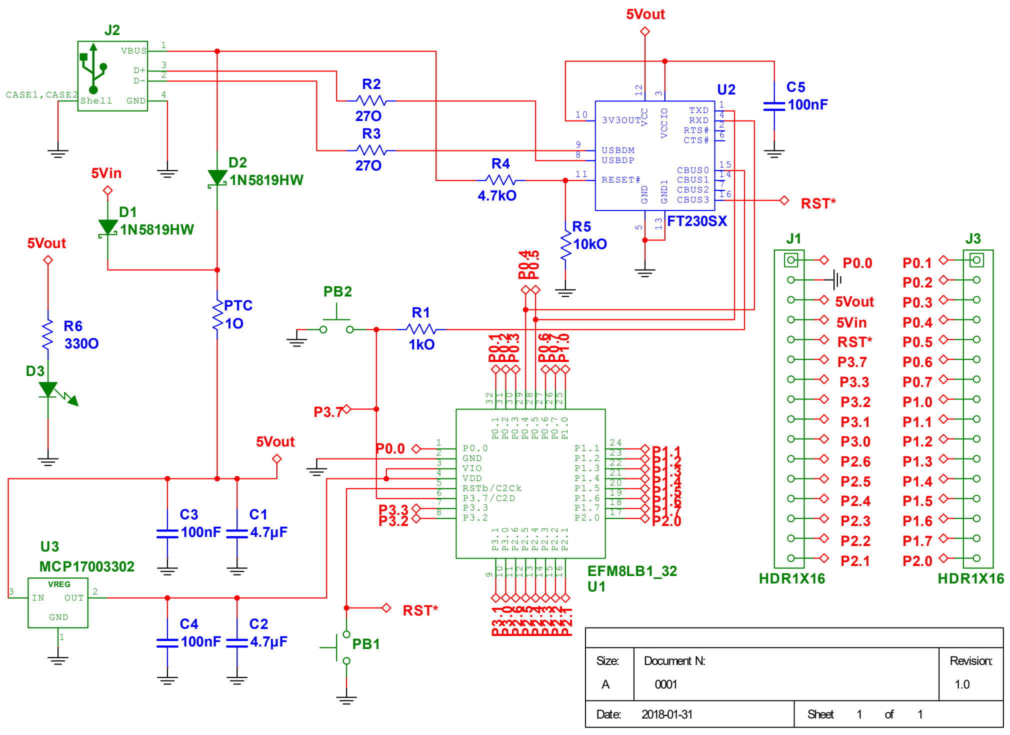


Figure 1. Circuit schematic of the EFM8 microcontroller system.

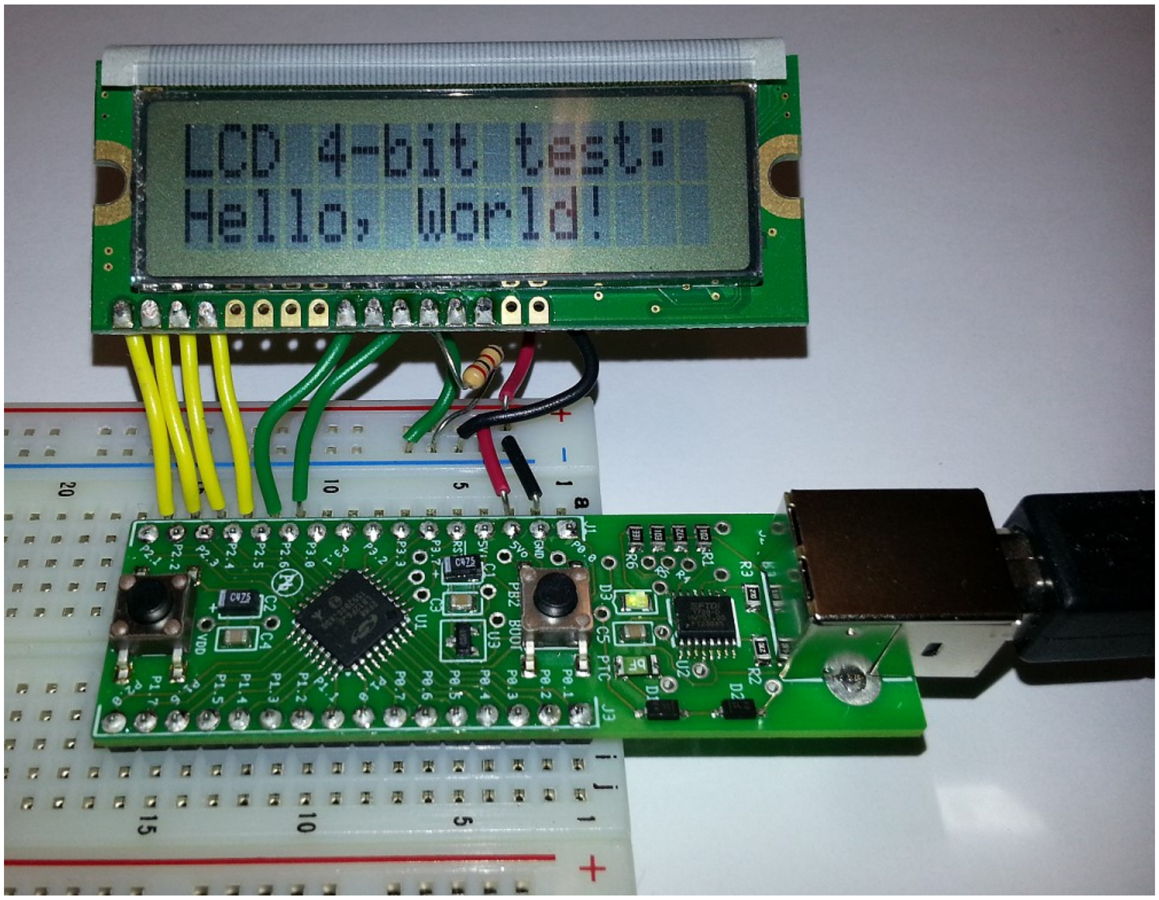


Figure 2. EFM8 board in a bread board with a 16 x 2 LCD.

Setting up the Development Environment

To establish a workflow for the EFM8 we need to install and configure 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) or newer. 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].

   Some of the Makefiles used in the examples below may use a "wait" program developed by the author. This program (and its source code) can be downloaded from the course

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

web page and must be copied into the CrossIDE folder or any other folder available in the environment variable “PATH”.

2. CALL51 Toolchain for Windows.

CALL51[2] (C compiler, Assembler, Linker, Librarian for the 8051) is included with the installation of CrossIDE. The “bin” folder of the CALL51 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:\CrossIDE\Call51”, then the folder “C:\CrossIDE\Call51\Bin” must be added at the end of the environment variable “PATH” as described here[1].

3. EFM8 Flash Loader: EFM8_prog.

Available in the web page for the course is the program “EFM8_prog.zip” developed by the author. Download and decompress the archive file “EFM8_prog.zip” somewhere in your hard drive.

The folder of the EFM8 flash loader must be added to the environment variable “PATH” in order for the workflow described bellow to operate properly. For example, if the EFM8 flash loader is installed in the folder “C:\CrossIDE\EFM8_prog”, then the folder “C:\CrossIDE\EFM8_prog” must be added at the end of the environment variable “PATH” as described here[1]. Alternatively, the full path of the EFM8_prog.exe can be provided in the Makefiles.

4. PuTTy

PuTTy (http://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html) is used to display information sent from the serial port of the microcontroller. The Makefiles provided assume that the PuTTy is available in the “PATH” environment variable. To add PuTTy to the “PATH” follow the instructions here[1].

[2] Derived work from SDCC (https://sourceforge.net/projects/sdcc/). The author used to contribute to this project but branched off to make the compiler more suitable for an educational environment. The assembler, linker, and librarian, as well as many of the C library functions were developed by the author.

Workflow.

The workflow for the EFM8 microcontroller includes the following steps.

1. Creation and Maintenance of Makefiles.

CrossIDE version 2.24 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:

```
# 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=c51
# Object files to link
OBJS=Blinky.obj

# The default 'target' (output) is Blinky.hex and 'depends' on
# the object files listed in the 'OBJS' assignment above.
# These object files are linked together to create Blinky.hex.
Blinky.hex: $(OBJS)
        $(CC) $(OBJS)
        @echo Done!

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

# Target 'clean' is used to remove all object files and executables
# associated wit this project
clean:
        @del $(OBJS) *.asm *.lkr *.lst *.map *.hex *.map 2> nul

# Target 'FlashLoad' is used to load the hex file to the microcontroller
# using the flash loader.  If the folder of the flash loader has been
# added to 'PATH' just 'EFM8_prog' is needed.  Otherwise, a valid and
# complete path must be provided.
LoadFlash:
        EFM8_prog.exe -ft230 -r Blinky.hex

# Phony targets can be added to show useful files in the file list of
# CrossIDE or execute arbitrary programs:
Dummy: Blinky.hex Blinky.Map

explorer:
        explorer .
```

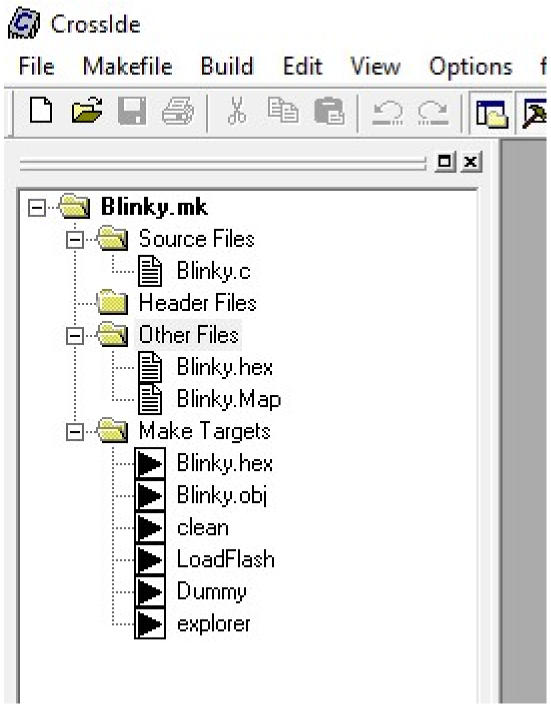
The preferred extension used by CrossIDE Makefiles is ".mk". For example, the file above is named "blinky.mk".

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.

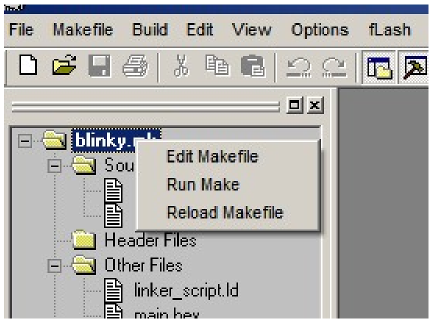
https://www.gnu.org/software/make/manual/make.html
http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/
https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html
https://en.wikipedia.org/wiki/Makefile

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

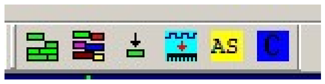
To open a Makefile in CrossIDE, click "Makefile"→"Open" and select the Makefile to open. For example "Blinky.mk". The project panel is displayed showing all the targets and source files:



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 -----------------
c51 -c Blinky.c
c51 Blinky.obj
Done!
```

Loading the Hex File into the Microcontroller's Flash Memory

To load the flash memory into the microcontroller, double click the 'FlashLoad' target. This output is then displayed in the report panel of CrossIDE:

```
----------------- CrossIde - Running Make -----------------
EFM8_prog.exe -ft230 -r Blinky.hex
Serial flash programmer for the EFM8LB1. (C) Jesus Calvino-Fraga (2012-2017)
Connected to COM7
Blinky.hex: loaded 208 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 2.2 seconds.
```

Once loaded, the program starts running automatically.

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 the example above COM4 is stored in the file. "COMPORT.inc" can be used in the Makefile to create a target that starts a PuTTy serial terminal session using the correct serial port:

```
PORTN=$(shell type COMPORT.inc)
.
.
putty:
	@Taskkill /IM putty.exe /F 2>NUL | wait 500
	c:\putty\putty.exe -serial $(PORTN) -sercfg 115200,8,n,1,N -v
```

For more details about using "COMPORT.inc" check the project examples below.

Project Examples

The following 'Make Project' examples are available in the web page of the course.

7SegDisp: Uses a timer interrupt to handle a multiplexed three 7-segment display.

ADC: Reads four channels of the built in 14-bit ADC (P2.2, P2.3, P2.4, and P2.5) and displays the result using printf() via PuTTy.

All_timers: Uses the six timers and the five channels of the Programmable Counter Array (PCA) with their corresponding Interrupt Services Routines (ISRs) to generate eleven different frequencies at 11 pins.

Autotest: Test that all the solder joints in the EFM8 board are correct.

Blinky: Toggles an LED connected to pin P2.1. This is the same project used in the examples above.

BlinkyISR: Similar to "Blinky" but instead of using a delay loop, it uses a timer interrupt. Timer 0 and its corresponding interrupt service routine (ISR) are used in this example.

DAC: Demonstrates how to use the 12-bit DAC of the EFM8LB1. Also shows how to generate different wave forms: sine, ramp, triangle, and square using the DAC.

EFM8_Servo: Shows how to control a standard servo motor (HS-422) using a timer interrupt to generate the required 50 Hz PWM signal. The pulse width can be changed via the serial port and PuTTy to arbitrary values between 0.6 ms and 2.4 ms.

EFM8_UUID: Shows how to read the "Universally Unique Identifier" available in the EFM8LB1 microcontroller. The identifier is displayed via the serial port and PuTTy.

Frequency: Shows how to measure frequency using one of the counters in the EFM8LB1 microcontroller.

HelloWorld: Uses printf() to display "Hello, World!" via the serial port and PuTTy.

I2C_24C02: Shows how to use the built-in I^2C/SMB controller to access and test a 24C02 I^2C EEPROM.

I2C_Nunchuck: Shows how to use the built-in I^2C/SMB controller to access and use the Nintendo nunchuck that came with the WII game console. A nunchuck and connector are required to use this program.

LCD: Shows how to configure and use a Hitachi compatible 16 x 2 LCD in four bit mode.

Period: Shows how to measure period using one of the timers in the EFM8LB1 microcontroller and an arbitrary input pin.

Quadrature_Encoder: Uses a timer interrupt to read and process the input of a 2-phase quadrature encoder. The encoder is used to increment/decrement a variable that is printed via the serial port to PuTTy.

SPI_ADC: Shows how to use the built-in SPI controller to access the MCP3008 10-bit SPI ADC.

SPI_EEPROM: Shows how to use the built-in SPI controller to access and test the FT93C66A SPI EEPROM.

Tetris: Implements the game Tetris via PuTTy. Sorry, no next move preview!

Tone: Shows how to produce a square wave at one of the pins of the microcontroller. The frequency is selected by using the scanf() function.

Uart1: Show how to configure and use the second uart (UART1) available in the EFM8LB1 microcontroller.