DEVELOPING FOR SILICON LABS EFM32 MICROCONTROLLERS IN LINUX

1 Introduction

The EFM32 microcontroller family of Silicon Labs (former Energy Micro) has many subfamilies with different Cortex-M architectures.

Family	Core	Features	Flash (kB)	RAM (kB)
Zero Gecko	ARM Cortex-M0+		4- 32	2- 4
Happy Gecko	ARM Cortex-M0+	USB	32- 64	4- 8
Tiny Gecko	ARM Cortex-M3	LCD	4- 32	2- 4
Gecko	ARM Cortex-M3	LCD	16- 128	8-16
Jade Gecko	ARM Cortex-M3		128-1024	32-256
Leopard Gecko	ARM Cortex-M3	USB, LCD	64- 256	32
Giant Gecko	ARM Cortex-M3	USB. LCD	512-1024	128
Giant Gecko S1	ARM Cortex-M4	USB, LCD	2048	512
Pearl Gecko	ARM Cortex-M4		128-1024	32-256
Wonder Gecko	ARM Cortex-M4	USB, LCD	64- 256	32

The development board EMF32GG-STK3700 features a EFM32GG990F1024 microcontroller from the Giant Gecko family. It is a Cortex M3 processor with the following features:

Flash (KB)	RAM (KB)	GPIO	USB	LCD	USART/ UART	LEUART	Timer/ PWMRTC	ADC	DAC	OpAmp
1024	128	87	Y	8x34	3/2	2 2	4/12	1(8)	2(8)	3

2 Basic References

The most important references are:

- EFM32GG990 Datasheet : Technical information about the EMF32GG990F1024 including electrical specifications and pinout.
- EFM32GG Reference Manual : Manual describing all peripherals, memory map.

- EFM32GG-STK3700 Giant Gecko Starter Kit User's Guide : Information about the STK3700 Board.

3 Peripherals

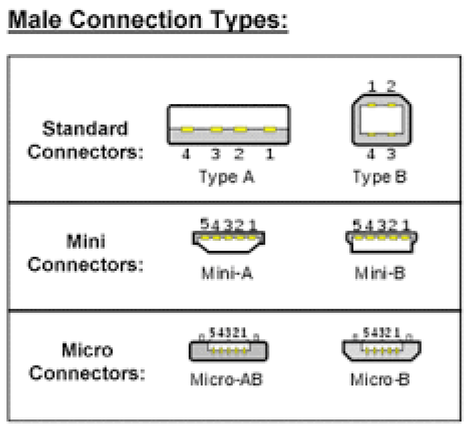
There are a lot of peripherals in the board.

- LEDs: Using pins PE2 and PE3.
- Buttons: Using pins PB9 and PB10
- LCD Multiplexed 20x8 with a 4 character alphanumeric field, a 4 digit numeric field and other symbols at pins PA0-PA11, PA15,PB0-PB6,PD9-PD12,PE4-PE7
- Touch Sensor: Using pins PC8-11.
- Light Sensor: Using PD6,PC6
- LC Sensor: Using PB12,PC7
- NAND Flash: Using PB15, PE8-15, PC1-2, PF8-9, PD13-15
- USB OTG: Using PF11-12, PF5-6

It is also possible to use the header connectors to add more peripherals.

4 Connections

The EMF32GG-STK3700 board has two USB connectors: One, a Mini USB B Connector, for development and other, a Micro B-Type USB Connector, for the application.



For development, a cable (delivered) must be connected between the Mini USB connector on the board and the A-Type connector on the PC. Among the devices listed by the *lsusb* command, the following must appear.

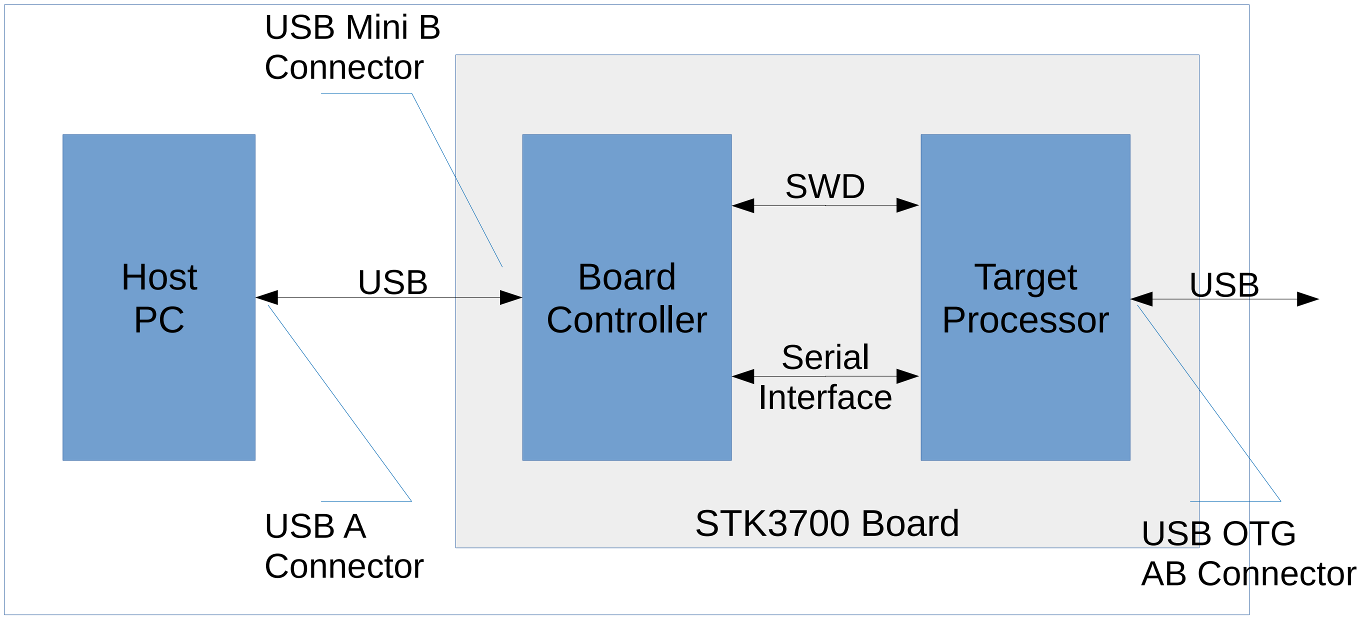
```
Bus 001 Device 019: ID 1366:0101 SEGGER J-Link PLUS
```

The STK3700 board has two microcontrollers: one, called target, is a EFM32GG990F1024 microcontroller, and the other, called Board Controller, implements an interface for programming and debugging.

For a Cortex M microcontroller the following programming interfaces are used:

- SWD : 2 pinos: SWCLK, SWDIO - Serial Wire Debug.
- JTAG : 4 pinos: TCK, TMS, TDI, TDO - Not used in this board

In the STK3700 only the SWD interface is used, and there is a connector on the SWD lines which permits the debugging of off board microcontrollers.



Generally, there is a serial interface between the Target and the Board Controller. It can be implemented using a physical channel with 2 lines or a virtual, using the SWD/JTAG channel. Both appears to the Host PC as a serial virtual port (`COMx` or `/dev/ttyACMx`). In the STK3700 board the serial channel uses the UART0 unit (PE0 and PE1).

5 Tools

The tools needed for developing embedded system for ARM microcontrollers are:

1. Toolchain (Compiler/Linker/Debugger/Tools) supporting the microcontroller family.
2. CMSIS library

3. Files specific to the microcontroller
4. Flash tools
5. Debugging tools
6. Debugging interface

5.1 Toolchain

A toolchain for ARM is the ARM GNU CC, which includes compiler, assembler, linker, debugger, and many utilities. It can be downloaded from ARM GNU GCC or installing using a package manager like apt. Until recently this site was hosted in Launchpad.

The toolchain is a version of the GNU CC ported to ARM processors in a project backed by ARM itself. The are other alternatives: one free alternative comes from Linaro.

After installing the toolchain, the following applications are avaliable:

```
arm-none-eabi-addr2line  arm-none-eabi-gcc-ar      arm-none-eabi-nm
arm-none-eabi-ar         arm-none-eabi-gcc-nm      arm-none-eabi-objcopy
arm-none-eabi-as         arm-none-eabi-gcc-ranlib  arm-none-eabi-objdump
arm-none-eabi-c++        arm-none-eabi-gcov        arm-none-eabi-ranlib
arm-none-eabi-c++filt    arm-none-eabi-gcov-dump   arm-none-eabi-readelf
arm-none-eabi-cpp        arm-none-eabi-gcov-tool   arm-none-eabi-size
arm-none-eabi-elfedit    arm-none-eabi-gdb         arm-none-eabi-strings
arm-none-eabi-g++        arm-none-eabi-gprof       arm-none-eabi-strip
arm-none-eabi-gcc        arm-none-eabi-ld
```

5.2 CMSIS

The CMSIS library is built on many components. It creates a Hardware Abstraction Layer (HAL) for the ARM peripherals but not for the manufactured added ones. It standardizes the access to registers and resources of the microcontroller.

This enables the developing without reliying too much on libraries provided by the manufacturer, that hampers the portability. See CMSIS Site

There are two versions of CMSIS:

- Version 4: older version but still heavily used with a BSD or zlib license.
- Version 5: new version with a broader license (apache) and support for new cores.

The EFM32 support library (see next) comes with a version 4 CMSIS library!

To download the version 4, one have to use the following command on the target folder

```
git clone https://github.com/ARM-software/CMSIS
```

For the version 5, use the command below.

```
git clone https://github.com/ARM-software/CMSIS_5
```

5.3 Files specific to the EFM32 microcontroller

The files needed for development for microcontroller are:

- *headers*: that define the registers to access the hardware.
- *link script*: that tells the linker (ld) how to built the objects files to build an executable
- *initialization files*: according CMSIS, two files, in this case, start_efm32gg.c e system_efm32gg.c, have the code to initializer the processing, including the initialization of data section.

There is a Software Development Kit SDK, provided the manufacturer, but no more supported. It can be still dowloaded from github using the following command.

```
git clone https://github.com/SiliconLabs/Gecko_SDK
```

There is an older version, which can be downloaded from Gecko_SDK.zip. IT is a huge file, about 300 MBytes zipped and and 2,4 GBytes after unzipping.

Silicon Labs provides an Development Environment called Simplicity Studio, based on Eclipse, which downloads the files as they are needed. It is possible to use Simplicity just to download the files and copy them to another folder. this has the advantage of downloading only what is needed.

5.3.1 Header files

The ARM CMSIS files are in `Gecko_SDK/platform/CMSIS/` folder, but there is never a need to include them, because they are used in the component include files.

In the `Gecko_SDK/platform/Device/SiliconLabs/EFM32GG/Include/` folder there are the device specific header files like the `efm32gg990f1024.h`, which defines symbols por registers according the CMSIS Standard for the EFM32GG990f1024 Microcontroller. There are also header files for peripheral like `efm32gg_gpio.h`, `efm32gg_adc.h` and others, which are already included by the device specific header file.

Although it is possible and quite common to include the device specific file directly, a better approach is to use a *generic* header, and define which device using a prepreprocessor symbol during compilation. To do this, copy the file `em_device.h` from the header files folder to the project folder and define the symbol `EFM32GG990F1024` with the `-DEFM32GG990F1024` compiler parameter.

There is other header file, called `em_chip.h` located in the `Gecko_SDK/platform/emlib/inc` folder that defines the `CHIP_Init` function. This function must be called at the very beginning and it corrects some bugs in core implementation. Since it calls other header files from the same folder, a better idea is to add this folder to the include path with the `-IGecko_SDK/platform/emlib/inc/` parameter.

5.3.2 libraries

In folder `Gecko_SDK/platform/CMSIS/Lib/GCC/libarm_cortexM` there are the following libraries:

```
libarm_cortexM0l_math.a     libarm_cortexM7lfdp_math.a
libarm_cortexM3l_math.a     libarm_cortexM7lfsp_math.a
libarm_cortexM4lf_math.a    libarm_cortexM7l_math.a
libarm_cortexM4l_math.a
```

5.4 Ferramenta para gravação da Flash

The communication protocol used by the EFM32 boards is compatible with the J-Link. Two software application implement it:

- JLink
- OpenOCD

5.4.1 JLink

To use the JLink software download from Segger the JLink package (`JLink_Linux_V622b_x86_64.deb` or a more recent one), and if optionally the Ozone package (`ozone_2.54_x86_64.deb`), an GUI interface for JLink. To install them on Linux, run the following commands after downloading them:

```
sudo dpkg -i JLink_Linux_V622b_x86_64.deb
sudo dpkg -i ozone_2.54_x86_64.deb
```

After installing JLink, the following application are available in the /opt/SEGGER/JLink folder, with the corresponding links in /usr/bin folder.

```
JFlashSPI_CL        JLinkExe            JLinkGDBServer     JLinkLicenseManager
JLinkRegistration   JLinkRemoteServer   JLinkRTTClient     JLinkRTTLogger
JLinkSTM32          JLinkSWOViewer      JTAGLoadExe        Ozone
```

5.4.2 OpenOCD

NOT TESTED YET!!!

To use the openocd software, install a openocd using a package manager (apt) or compiling a new version from source code dowloaded from openocd.

Define a symbol `OOCDSCRIPTSDIR` as below.

```
OOCDSCRIPTSDIR=/usr/share/openocd/scripts
```

To run it, it is neccessary to specify interface and board.

```
openocd -f $OOCDSCRIPTSDIR/interface/jlink.cfg -f $OOCDSCRIPTSDIR/board/efm32.cfg
```

5.5 Debugging Tools

In the GCC toolchain, the tool for debugging is the GNU Project Debugger (GDB), in the form specific for ARM (`arm-none-eabi-gdb`). It runa on the PC and since it can not communicate directly with the microcontroller, a relay mechanism is needed in the form of a GDB proxy. The GDB proxy can be the Segger JLinkGDBServer or the openocd.

This is a two step process:

1. In a separated window, start the GDB Proxy
   ```
   $JLinkGDBServer
   ```
2. In other window, start GDB and stabilishes a connection
   ```
   arm-none-eabi-gdb
   (gdb) target remote localhost:2331
   ```

The ARM GNU GDB has a Command Line Interface (CLI) as default and a Text User Interface (TUI), which can be activated with a -ui parameter. There are many Graphic User Interface (GUI) applications for GDB including ddd and nemiver.

6 Integrated Development Environment (IDE)

The are many IDEs for Embedded Development, mostly based on Eclipse). The manufactures provides the Simplicity Studio.

Alternatives are:

- VisualStudio (Windows)
- emIDE (Windows)
- CodeBlocks
- Embedded Studio

7 Using the tools

7.1 Connecting

Connect the board to the PC using the Mini USB cable. When connected a blue LED should lit.

Starting JLinkExe prompt as shown.

```
$JLinkExe
SEGGER J-Link Commander V6.22b (Compiled Dec  6 2017 17:02:58)
DLL version V6.22b, compiled Dec  6 2017 17:02:52

Connecting to J-Link via USB...O.K.
Firmware: Energy Micro EFM32 compiled Mar  1 2013 14:08:50
Hardware version: V7.00
S/N: 440112411
License(s): GDB
VTref = 3.329V
```

Type "connect" to establish a target connection, '?' for help

```
J-Link>si swd
J-Link>speed 4000
```

```
J-Link>device EFM32GG990F1024
J-Link>connect
```

See the transcript below.

```
$JLinkExe
J-Link>si swd
Selecting SWD as current target interface.
J-Link>speed 4000
Selecting 4000 kHz as target interface speed
J-Link>device EFM32GG990F1024
J-Link>con
Device "EFM32GG990F1024" selected.


Connecting to target via SWD
Found SW-DP with ID 0x2BA01477
Found SW-DP with ID 0x2BA01477
Scanning AP map to find all available APs
AP[1]: Stopped AP scan as end of AP map has been reached
AP[0]: AHB-AP (IDR: 0x24770011)
Iterating through AP map to find AHB-AP to use
AP[0]: Core found
AP[0]: AHB-AP ROM base: 0xE00FF000
CPUID register: 0x412FC231. Implementer code: 0x41 (ARM)
Found Cortex-M3 r2p1, Little endian.
FPUnit: 6 code (BP) slots and 2 literal slots
CoreSight components:
ROMTbl[0] @ E00FF000
ROMTbl[0][0]: E000E000, CID: B105E00D, PID: 000BB000 SCS
ROMTbl[0][1]: E0001000, CID: B105E00D, PID: 003BB002 DWT
ROMTbl[0][2]: E0002000, CID: B105E00D, PID: 002BB003 FPB
ROMTbl[0][3]: E0000000, CID: B105E00D, PID: 003BB001 ITM
ROMTbl[0][4]: E0040000, CID: B105900D, PID: 003BB923 TPIU-Lite
ROMTbl[0][5]: E0041000, CID: B105900D, PID: 003BB924 ETM-M3
Cortex-M3 identified.
J-Link>
```

A command line makes all easier.

```
JLinkExe -if swd -device EFM32GG990F1024 -speed 2000
```

Considering that the debugging session will be done with GDB, the only important commands for JLink are related to flashing a program:

- *exit*: quits JLink
- *g*: start the CPU core
- *h*: halts the CPU core
- *r*: resets and halts the target
- *erase*: erase internal flash
- *loadfile*: load data file into target memory

7.2 Flashing

To use the JLinkExe to write the flash contents one has to use the following command:

```
$JLinkExe -Device EFM32GG990F1024 -If SWD -Speed 4000 -CommanderScript
Flash.jlink
```

The Flash.jlink file has the following contents:

```
r
h
loadbin <path>,<base address>
r
```

7.3 Debugging

To use gdb as a debugging tool, the GDB Proxy must be started first using the command

```
JLinkGDBServer -if SWD -device EFM32GG995F1024 -speed 4000
```

In other window to start the gdb the following command is used:

$arm-none-eabi-gdb <execfile>

```
(gdb) monitor reset
(gdb) target remote localhost:2331
(gdb) break main
(gdb) monitor go
```

To debug using GDB see Debugging with GDB.
The most used commands are:

- *cont*: continues the execution
- *break*: sets a breakpoint in a function or in a line
- *list*: lists source file
- *next*: steps skipping over functions
- *step*: steps entering functions
- *display*: Displays the contents of a variable at each prompt.
- *print*: Prints the contents of a variable

The monitor commands enable direct access to JLink functionalities, as shown below.

- *monitor reset*: resets the CPU
- *monitor halt*: halts he CPU

The commands can be abbreviated by typing just enough characters. For example, *b main* is the same of *break main*.

8 Sample project

8.1 Structure

A sample project for a EFM32GG microcontroller consists of:

- application source files (e,g, main.c)
- startup file (startup_efm32gg.c), copied from „„„
- CMSIS system initialization file (system_efm32gg.c) copied from ......
- linker script (efm32gg.ld) copied from ......
- header file (em_device.h) copied from
- Makefile, a build automation tool
- Optionally, README.md, a description of project
- Optionally, Doxyfile, configuration file for doxygen, a documentation generator

8.2 Makefile

The make application can automate a lot of task in the software development. The recipes are specified in a `Makefile`. The `Makefile` provides a lot of options, which can be specified in the command line. In the `Makefile`, it is possible (and sometimes nedded) to adjust compilation parameters and specify where the required folders and applications are. For example, `OBJDIR` is specified as gcc and is the folder where all object files and the executable are generated. `PROGNAME` is the project name

- *make all* generate image file in OBJDIR
- *make flash* write to flash memory in microcontroller
- *make clean* delete all generated files
- *make dis* disassemble output file into OBJDIR)/PROGNAME.S
- *make dump* generate a hexadecimal dump file into OBJDIR/PROGNAME.dump
- *make nm* list symbol table in standard output
- *make size* list size of output file
- *make term* open a terminal for serial communication to board
- *make debug* start an GDB proxy and the GDB debugger
- *make gdbproxy* start the GDB proxy

- *make docs* generates docs using doxygen

9 Notes

The files for EFM32GG are in:

- **headers**: `Gecko_SDK/platform/Device/SiliconLabs/EFM32GG/Include`
- **linker script**: `Gecko_SDK/platform/Device/SiliconLabs/EFM32GG/Source/GCC/emf32gg.ld`
- **initialization**: `Gecko_SDK/platform/Device/SiliconLabs/EFM32GG/Source/GCC/startup_efm32gg.c` `Gecko_SDK/platform/Device/SiliconLabs/EFM32GG/Source/GCC/startup_efm32gg.S}`]

The software Simplicity Commander enables the access to the JLink functionalities using a CLI.

The software Ozone provides a GUI for the JLink system.