Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Technik & Architektur

Infotronik

Lab Assignments

HS2015

Contents

1 Assignment #1: Start

We form the teams and get initial hardware. Working in teams allows to share the workload and to share knowledge.

1.1 Goals

1. You have formed a team of two. Ideally the team is build by a CS and an EE engineer.
2. Decide if you want to buy/build your own robot.
3. You have received the initial hardware for the course.

1.2 Hints

1. Carefully select your team mate. Meet and greet ☺.
2. Fill out the INTRO TEAM form: Use a good group name. You can change the name later if needed.
3. Fill in the names of your team members and return one team sheet per team to your instructor.
4. You can keep the other team sheet for your records.
5. You will get a INTRO Kit assigned.
6. Check the material in the INTRO kit. The extra robot material will be provided at a later time.
7. Fill out the INTRO PREORDER form and pass it to your instructor. Mark the hardware you would like to own. If you do not want any hardware, then mark that you are not interested. That way we can plan for the hardware orders.
8. Return the preorder form to your instructor.
9. Have fun ☺

2 Assignment #2: Recap Instructions

You provide a summary of the lecture material from one week ago and present it to the class. Additionally you create a quiz with solutions. And you know what students have provided you as tips. You will need to do the recap itself as homework.

2.1 Goals

1. You consider the tips of the students from previous semester.
2. You provide a summary and presentation to the class what we have learned one week ago.
3. Submit/place your material on GitHub **in advance**.
4. Alternatively, you can present an extended topic. E.g. you think you have some really cool tips and tricks around a topic, then go for it!
5. You create a set of five questions (with solutions). No need to go through the questions during your presentation.
6. The quiz questions could be used for exam preparation or even be used in the exam.

2.2 Hints

1. Read and consider the tips from previous semester. Be prepared to provide tips as well for the next generation.
2. We are using GitHub (a version control system). With the next labs you will get GitHub access.
3. Register for a time slot in the Recap schedule sheet. The Recap schedule will be published and updated on GitHub (`Recaps/recap schedule.txt`). Alternatively you can register in the paper form.
4. The presentation should be short, around 5-10 minutes.
5. It does not have be slide form: the goal is that it is useful for everyone.
6. Prepare as well a short quiz with five questions in writing.
7. Submit your material in advance on GitHub in the *Recaps* folder.

8. Store your files (presentation with quiz) in the `Recaps` folder inside the instructor version control repository.

9. An example from previous semester is stored on GitHub.

10. Acceptance criteria: delivered and presented, useful, material submitted in VCS and accepted.

11. Have fun ☺

2.3 Example Recap Summary and Quiz

Author: Erich Styger
This is an example Recap summary (key points, quiz with answers).

2.3.1 Key Points

1. Need to know/refresh the basic preconditions:
   (a) Eclipse
   (b) Basic C programming
2. Use script to augment learning material
3. Learning goals are in the script
4. Important: prepare, do not fall behind, catch up

2.3.2 Quiz

1. With $X == 0x00EFC000$ and Y=((X >>16)&0xFF)-5, what is Y in hexadecimal?
2. Determine the decimal values for following hexadecimal numbers: 0x10, 0x20?
3. Write in C a function for the HCS08 microcontroller which takes two 16bit arguments and returns the sum of the two arguments.
4. Write in C code a single statement which changes bit number 5 in PORTA and let the other bits untouched (bit number 0 is the least significant bit), in following way: if the bit is already set, it shall be cleared; if the bit is already cleared, then it should be set.

5. Determine the value of register A for following code sequence:
   asm LDAA #5;
   asm STAA i;
   asm INCA;
   asm MULA i;

2.3.3 Solutions

1. 0xEA
2. 16, 32
3. short Add(short a, short b) { return a+b; }
4. $PORTA \hat{}= (1 << 5)$;
5. 30

3 Assignment #3: GitHub

You need a GitHub account to access the course GitHub material. And you need to decide where you want to host your robot group repository.

3.1 Goals

1. Creating a GitHub account to access course material.
2. Verifying login/account information, both for GitHub and your own repository.
3. Creating a repository and adding collaborators.

3.2 Hints

1. If you do not have a GitHub account yet: go to `http://github.com` and create a new account.
   - Provide a GitHub user name, email address, password and sign up.
   - Send your GitHub user name to erich.styger@hslu.ch so he can add you as collaborator to the INTRO repository.
   - Choose the 'Free' plan (no need to setup an organization).
   - Finish the sign up.
   - On the email address you provided, you will receive a notification that you have been added to the INTRO repository.
   - Click on the link to access the repository.
2. Use the web interface to browse through the different folders of the repository.
3. There is a 'Download ZIP' button which can be used to download a snapshot of the repository.
4. Congratulations! You have now access to the INTRO repository ☺

4 Assignment #4: First Projects

We have everything installed and doing our first baby steps.

4.1 Goals

1. You have Eclipse installed and are getting familiar using Eclipse.
2. Having the correct debug firmware loaded on the FRDM board.
3. You can create a project, build, download and debug it on the board.
4. You are getting familiar with the project structure.

4.2 Hints

1. Install the Kinetis Design Studio (KDS). Use the default installation settings.
2. Check your group laboratory hardware material. Put your name on the label of the boxes.
3. Connect the FRDM board with the mini USB cable on the port labeled 'SDA' (on the right hand side if you see the two USB ports in front of you). The green LED near the USB port shall be on. If plugged in the first time, it will install the drivers which were installed with the KDS installation.
4. The boards ship from the factory with an older firmware which does not allow debugging, so you need to upgrade the firmware first. Note that updating the bootloader does not work with Windows 8 or 10, you need someone with a Windows 7 machine. Ask your instructor if you do not have someone with a Windows 7 machine to update the board in the next step.
5. If the board enumerates as FRDM-KL25Z drive, then it is in MSD (Mass Storage Bootloader) mode. In this mode, you could copy S19 files to the board in order to program it. But you cannot use it for debugging. To be able to debug the board, you need to load an OpenSDA debug firmware. Such a firmware `MSDDEBUGFRDM-KL25Z_Pemicro_v114.SDA` is available on the software under OpenSDA. In order to have the board working with Windows 8.x, an update of the bootloader needs be performed. Note that for this you need a Windows

7 or 8.0 machine (otherwise ask your instructor to do this for you). To update the bootloader:

(a) Power the board with reset button pressed. Use the 'OpenSDA' USB port.

(b) The device enumerates as BOOTLOADER drive, and small green LED is blinking.

(c) Copy the file `BOOTUPDATEAPP_Pemicro_v111.SDA` to it.

(d) Power off the board.

(e) Power the board again, the small green LED shall be blinking with a frequency of about 1 Hz.

Now with the bootloader updated, we can load the new firmware which will be a debug interface and a USB MSD bootloader device:

(a) Unpower the board, then power the board with reset button pressed.

(b) The device enumerates as BOOTLOADER drive, and small green LED is blinking.

(c) Copy the `MSD-DEBUG-FRDM-KL25Z_Pemicro_v114.SDA` to the device.

(d) Re-power the board, the small green LED shall be on.

(e) Drivers get installed, and the board enumerates as FRDM-KL25Z, and works with the debugger as P&E OpenSDA debug device.

6. Launch the Eclipse IDE and select/create a new workspace.

7. Create projects for each board you have

   - FRDM: Kinetis L, MKL2x, KL25Z (48 MHz), MKL25Z128xxx4, Processor Expert
   - Robo: Kinetis K, MK20, MK22F (120 MHz), MK22FX512xxx12, Processor Expert

8. Generate code with Processor Expert.

9. To build the project, use the menu *Project > Build*.

10. Create the debug configuration for your board(s).

11. To download and debug, us the menu *Project > Debug*.

12. Download/debug to each board to make sure things are working.
13. By default the debug configuration is not stored as a .launch file. To do this, go to the launch/debug configuration, and in the 'Common' tab, enable the 'Share file' option.
14. By default, parallel builds are not enabled. Go to the project properties, C/C++ build and then the 'Behaviour' tab: enable parallel build and see what the effect is for building your project.
15. Have fun ☺

4.3 Questions

1. Why is it better to separate workspace and project folders?
2. What is a bootloader?
3. Why there are two USB connectors on the FRDM-KL25Z board?
4. What does 'sharing a debug configuration' mean? What happens if that 'share file' option is enabled?
5. What is the effect/impact of using the 'parallel build' option?

4.4 Links

- OpenSDA on the Freedom KL25Z Board
- The Freescale OpenSDA Trap: No Device Available, or something like that
- Sharing Debug Configration with Eclipse
- Using Parallel Build with Eclipse

5 Assignment #5: Git and SourceTree, eGit

We are installing a Git client on the host to work with the repositories.

5.1 Goals

1. Installation of Git, eGit and SourceTree.
2. Creating and using repositories.
3. Ability to commit, push and pull.
4. Resolving conflicts.

5.2 Hints

1. Download and install Git from http://git-scm.com/downloads. Use the latest version.
2. Download and install a client. Common ones are TortoiseGit from http://code.google.com/p/tortoisegit/ or SourceTree from http://www.sourcetreeapp.com/. I recommend to use SourceTree.
3. Which one you use/install is up to you, but SourceTree is a good starting point as it makes password caching easy. Go through the SourceTree installation with the recommended default settings.
4. Create a folder where you want to have your Git repository cloned/created, e.g. `Lectures/INTRO/git_INTRO`
5. Clone the instructor repository and your repository into that folder, so that it is e.g. `Lectures/INTRO/git/INTRO`.
6. Keep in mind that you only can clone into an empty/new folder, and not into a folder which has files in it.
7. Clone your team repository into that folder, so it is e.g. `Lectures/INTRO/git/MyTeam`.
8. Make local commits, and then push the files to the remote repository.
9. Add files/change it. See what happens on the machine of your team mate who pulls the files.

10. Create a conflict. Resolve that conflict. Make sure you are familiar with the process.

11. Inside your team repository, create a folder structure. Create a folder for the projects. Do not place the Eclipse workspace inside the repository!

12. Move the projects you have created with Eclipse in one of the earlier labs to this git project folder:

    (a) With your file manager/Explorer, move your projects folders from the workspace location to your new Git location.

    (b) In Eclipse, use the context menu on the project do Delete the project from the workspace. Do **NOT** (!!!!) select 'Delete project content on disk (cannot be undone)'.

    (c) Add your projects again to the workspace, this time from the Git folder location. For this, use the menu *File* > *Import* and use *General* > *Existing Projects into Workspace*. Browse to the Git project folder and import your projects.

13. Install the eGit Eclipse client.

    - Use the menu `Help` > `Install new Software` and use `http://download.eclipse.org/egit/updates` as update site.
    - Install **Eclipse Git Team Provider**

14. Have fun ☺

5.3 Questions

- Why do we need to ignore files?
- Why ignoring folders in a repository is useful?
- Why should you separate the Eclipse workspace from the projects?
- Why it can make sense to have multiple VCS clients installed (e.g. TortoiseGit and SourceTree)?
- How can you create a conflict yourself?
- Do you need a server to use Git?
- What is the difference between *Clone* and *Pull*?

- What is the difference between *Stage* and *Push*?
- Why do you need to add collaborators?

5.4 Links

- Installing eGit in Eclipse and CodeWarrior for MCU10.4
- Version Control with Processor Expert Projects

6 Assignment #6: Processor Expert

In this assignment we are doing first steps with Processor Expert and adding a LED component.

6.1 Goals

1. Start using Processor Expert.
2. Getting familiar with Processor Expert UI and capabilities.
3. Adding a **LED** component.
4. Practice to share 'binary' or large XML files with a VCS.

6.2 Hints

1. During the course we are using extra Processor Expert components from the following URL: https://sourceforge.net/projects/mcuoneclipse/files/PEx%20Components/
2. That site hosts in each release two ***.PEupd Update Files** in a zip archive: These are archive files containing components in a package.
3. To install the components:
   (a) Use the following URL
   https://sourceforge.net/projects/mcuoneclipse/files/PEx%20Components/
   and download the latest zip file with the components.
   (b) Extract the content of the zip file (it contains two *.PEupd Files)
   (c) In Eclipse, use the menu *Processor Expert > Import Component(s)* to import the *.PEupd files. You can select/import multiple files in one step.
4. If the Processor Expert views are not shown, use the menu *Processor Expert > Show Views*.
5. Use the *Components Library* view to show all available components. Best if you use the alphabetical tab.
6. Perform a refresh in the Components Library view if necessary. You should see components like 24AA_EEPROM, 74HC164 or Bluetooth_EGBT.

7. As you are going to change the `ProcessorExpert.pe` XML file, and as this one will be hard to merge in case of a conflict: agree in your team who is doing the Processor Expert change. Then that person commits and pushes his changes, and everyone else is pulling it. The principle steps are:
   (a) Group is making sure that everyone has the HEAD of the `ProcessorExpert.pe` file
   (b) Group agrees how is doing doing the change.
   (c) Everyone else will not perform a Processor Expert change.
   (d) Everyone else makes sure that his `ProcessorExpert.pe` file is not locked/used. Best way is to close the project in Eclipse (menu *Project > Close Project*).
   (e) Make change of `ProcessorExpert.pe`, save it, commit and push it.
   (f) Then pull/sync the change. This shall update the version of `ProcessorExpert.pe` of every team member.
   (g) Re-open the Eclipse project.
8. Add a *LED* component to your projects. Share it and its settings with your team members. See general procedure as above.
9. You can enable or disable the available methods (or events) with the context menu.
10. You can drag&drop a method name into your source file.
11. Use the *Inspector* view to configure the component, like using a different pin.
12. Switch the Component Inspector view to 'Advanced' mode so you see all properties. You can make that setting permanent for the workspace using the menu *Window > Preferences > Processor Expert > General > Preferred inspector views* settings.
13. Generate code and inspect the generated source files.
14. You can add multiple components of the same kind: add as many components as you have LEDs on your board. Notice they are named `LED1`, `LED2` and so on. It will create source files named `LED1.c` and `LED1.h`. Note that you can change the *Component Name*, and that

way it will affect the source file names of the generated code. Try it out and see how the source file names change of the generated files.

15. Use the view 'triangle' menu to switch between the different modes of the view, e.g. to use the 'Tabs view'. You can configure the default view with the workspace settings (menu *Window* > *Preferences* > *Processor Expert*).

16. Have fun ☺

6.3 Questions

- What is the advantage of using *.PEupd compared to use the files in the repository? What are advantages and disadvantages of using the Git repository files?
- You could use either BitIO or the LED component: What are the pros and cons of each approach?
- What are the advantages of using a tool like Processor Expert? What could be the disadvantages?
- What would be a typical method for an character display component? What would be a typical property or event?
- From the GitHub (`https://github.com/ErichStyger/McuOnEclipse_PEx`) you could download the repository as zip file and then unpack it at the right location. What is the advantage of cloning it instead?

6.4 Links

- McuOnEclipse Releases on SourceForge
- Tutorial: Bits and Pins with Kinetis and the FRDM-KL25Z Board
- Tutorial: Enlightning the Freedom KL25Z Board
- Switching between 'tabs' and 'no-tabs' UI in Processor Expert

7 Assignment #7: LED Driver

We implement a driver for the LEDs on our platform with using low-level BitIO component.

7.1 Goals

1. Use BitIO or LED component for LEDs.
2. Test that LEDs are working.

7.2 Requirements

We have to implement an LED driver controlling our LEDs in the system. The driver needs to support a different number of LEDs as every platform is different. Following functionality shall be supported in a LED driver:

- `LEDx_On()`: turn on LED x
- `LEDx_Off()`: turn off LED x
- `LEDx_Neg()`: toggle LED x
- `LEDx_Get()`: get the status (on or off) of LED x
- `LEDx_Put()`: set the status (on or off) of LED x
- `LED_Init()`: Initialization of the LED driver
- `LED_Deinit()`: Deinitialization of the LED driver

7.3 Hints

1. Locate and inspect the schematics of your boards to find out how the LEDs are connected to your microcontroller.
2. Create the source files `platform.c` and `platform.h`. They serve for platform initialization and configuration.
3. Consider macros for board identification, how many LEDs you have, etc.
4. Think about either adding these configuration macros either to the `Platform.h`, or to the `LED.h`.

5. Add preprocessor macros to your project compiler preprocessor settings to identify the boards. Consider macros like `PL_BOARD_IS_FRDM` or `PL_BOARD_IS_ROBO`.
6. Use a macro like `PL_HAS_LED` to completely switch off the LED module and functionality.
7. In case of Eclipse index problem, check http://mcuoneclipse.com/2012/03/20/fixing-the-eclipse-index/
8. Create `LED.c` and `LED.h` as common files for all your projects as the LED driver module.
9. Be sure you are in the Expert Inspector mode, and give your component a useful name. If you specify a *Pin Signal* name, then the pin assignments will be documented in a text file within the Documentation folder.
10. You can drag&drop a method name into your source file.
11. Even if you leave the `Deinit()` empty, it is not causing a lot of overhead, but keeps your architecture clean and orthogonal.
12. Write a test function to test your driver functionality.
13. Make sure you are able to compile your sources without warnings. If there are compiler or linker warnings, fix them.
14. Have fun ☺

7.4 Questions

- Why would you need a `Deinit()` function beside of an `Init()`?
- Why would you need an `Open()` and `Close()` function?
- You can use macro/inlined version of our LED methods. What would be the advantage using normal methods/functions instead?
- Your LED may have different anode/cathode wiring on the board. What do you need to do?
- What is the purpose of the 'Pin Signal' properties in Processor Expert?

8 Assignment #8: Preprocessor

We are using the preprocessor to know what the compiler is seeing. This that way preprocessor bugs can be debugged.

8.1 Goals

1. You able to generate the preprocessor output and to read it.

8.2 Hints

1. Generate a preprocessor listing for your code from previous lab working on LED. Add the preprocessor option to your project compiler settings.
2. With the Eclipse build tools integration, the output files are where usually the object files are placed, with an object file extension (*.o). You can open these files with a normal text editor, or drag it into the source file view.
3. Inspect the content of the preprocessor output. Notice several defines at the beginning: do you know from where they are coming from?
4. Notice how line and file information is present in the preprocessor output. See https://gcc.gnu.org/onlinedocs/gcc-4.4.0/cpp/Preprocessor-Output.html
5. How are things like `Bit1_ClrVal()` in BitIO or `LED1_Off()` in LED component implemented?
6. Do not forget to remove the preprocessor option at the end of the lab.
7. Have fun ☺

8.3 Questions

- Which compiler option is used to generate a preprocessor listing?
- Why is it preprocessor listing useful? What problems can be solved with it?
- Why does the build fail if you create the preprocessor output?
- Would you be able to compile the preprocessor output by the compiler again?

8.4 Links

- GNU gcc Preprocessor Output