STM32MP1 hands on

OpenSTLinux Distribution

Version: 1.2

life.augmented

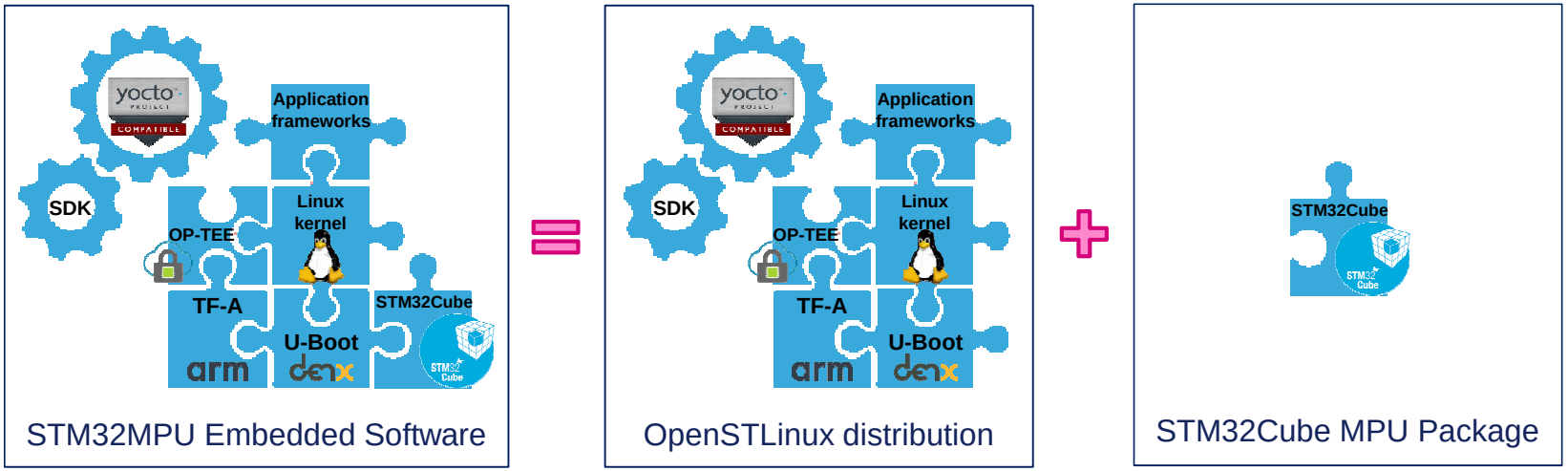
ST Restricted

OpenSTLinux distribution hands on

FOCUS

Dual Cortex-A7 cores

Cortex-M4 core



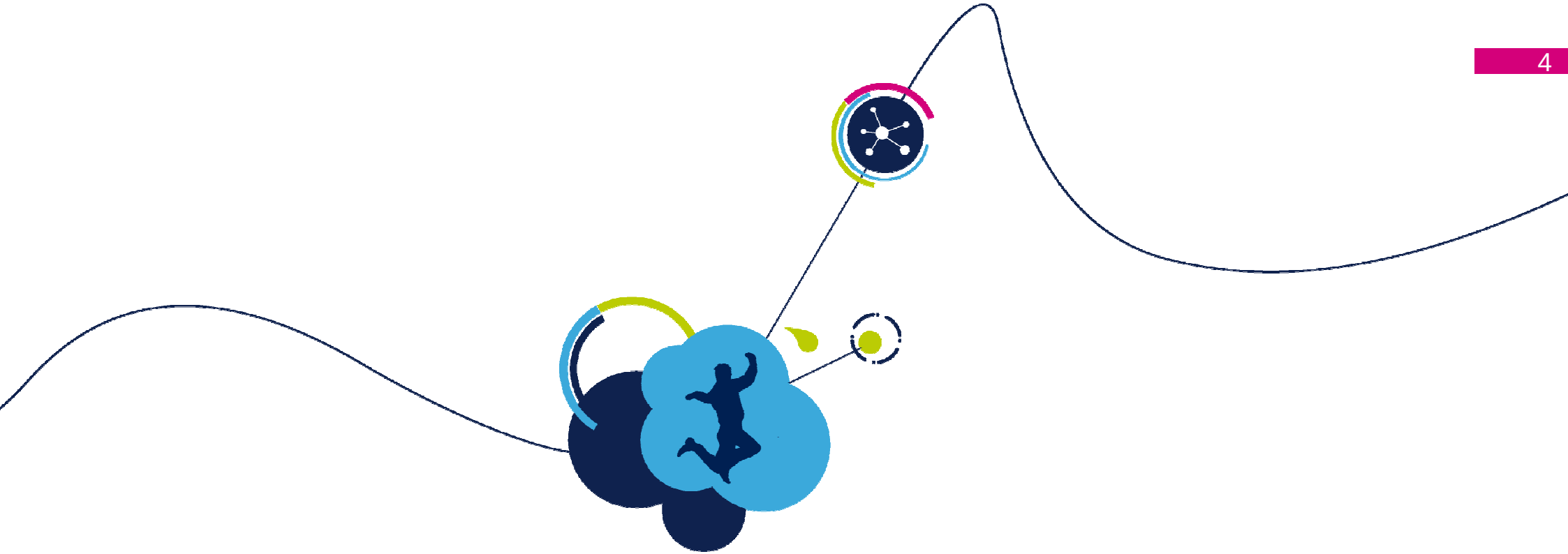
Content

Purpose of this presentation is to give insight/knowledge/tips about OpenSTLinux to STM32MPU platform.

"Hands on" means **learning by doing**.

This presentation describes the following sections:

- **Learning program**: What are the objectives and benefits of this training. What is the overall plan and expected duration.
- **Prerequisites**: What is mandatory to study, to have (in term of material) and required (other Training, other Hands On, etc...) before starting this training.
- **Theoretical school**: Documentation reference (to be consulted on request), self learning or presentation to follow.
- **Practicing school**: Learning by doing part (sequence of exercises and practical work)
- **Evaluation**: How to make sure all the objectives of the training were learnt?



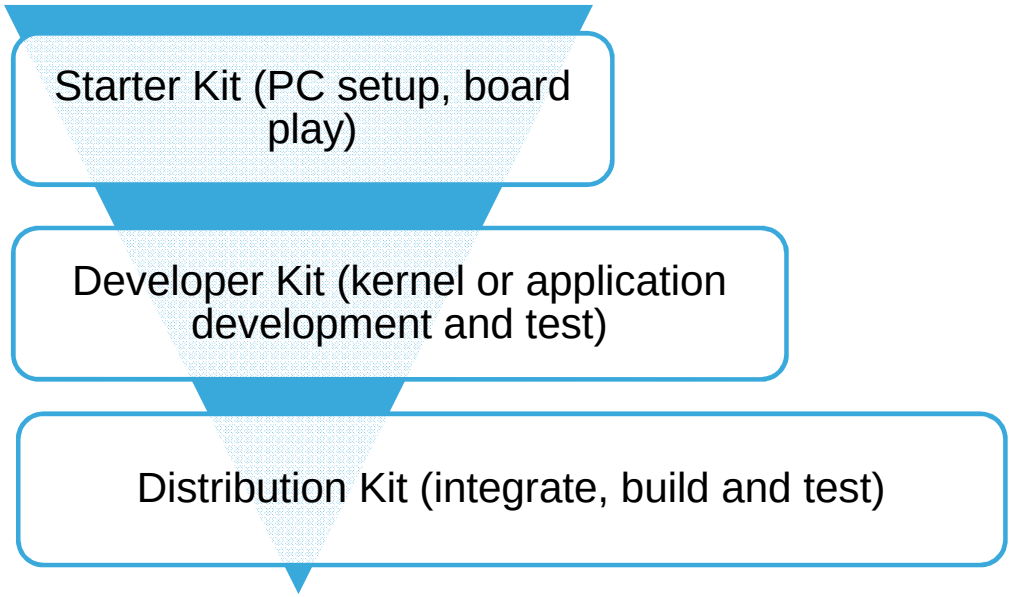
Learning program

life.augmented

Objectives

The first objective is to get an overview of OpenSTLinux distribution basic offering by managing (get, build, play, debug) the different kits.

The second objective is to get familiar with the wiki user guide, how it is structured and how to find answers to a question.



Benefits

The benefits for the trainee are:

- To validate PC setup, OpenSTLinux delivery accessibility and content: validate you have all to start your own development.
- Get started with OpenSTLinux distribution offering to start development: What is in and how to add your own development.
- Get an overview of the different kits to adapt the best development cycles to your situation and needs: save time !

Plan

Discover first what is OpenSTLinux distribution, how to get the release and the setup required to play with it.

You will then manage the different kits available with always the same simple type of use case.

Theoretical presentation: What is OpenSTLinux distribution and how to get it?

Theory: 4h

Setup your environment: PC setup, HW setup and connexions. Focus on Starter kit.

Practical: 4h

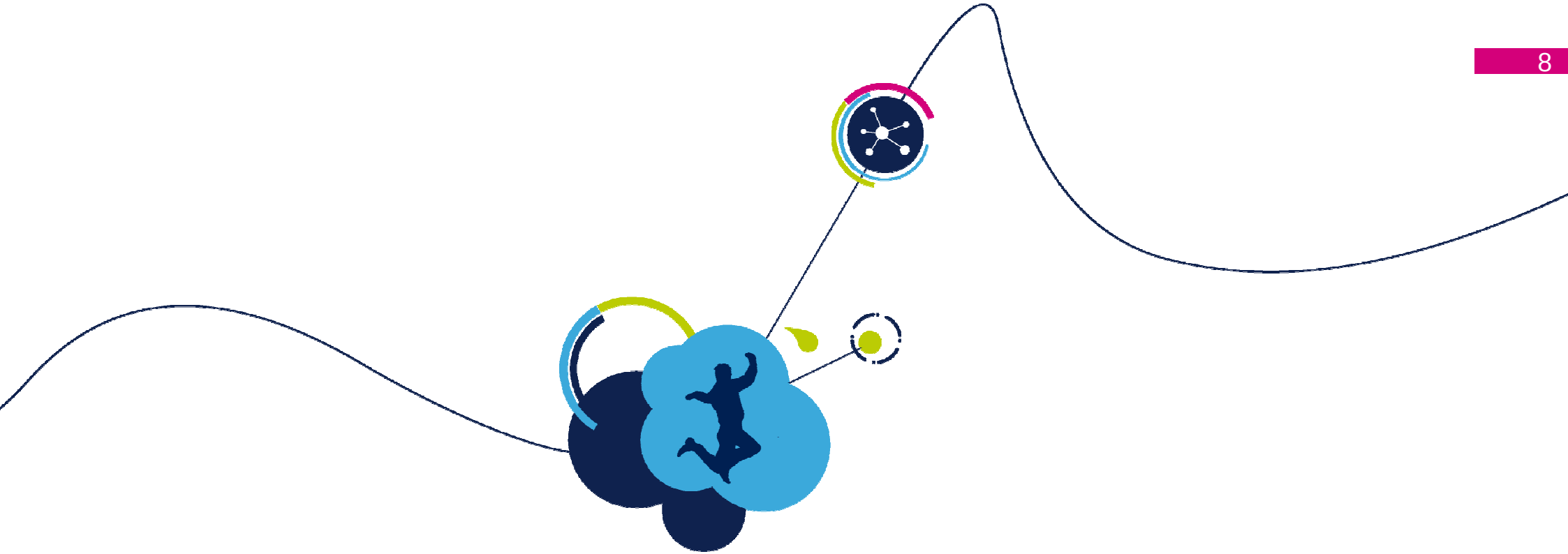
Starter kit: 3 labs

Play with developer then distribution kit for the same use cases.

Practical: 4d

Developer kit: 3 labs

Distribution kit: 3 labs



Prerequisites

Prerequisites 1/4

You should know ST delivery area where you can get full content of the delivery including the user guide (wiki format), trainings, tools and documents:

- Review the complete information given in this area and check you have access.
- Install the wiki user guide (if required) and point to the main page.
- In the rest of this presentation all the references will be quoted as below:
  - {element}: reference to a deliverable present is this area. For example {STM32MP1 Presentation} is the marketing training for STM32MP1.
  - [article]: reference to an article of the user guide (wiki). For example [Linux_online] is the article with the same name.
  - "element" or "article" are not direct link but keyword you should easily find with a search

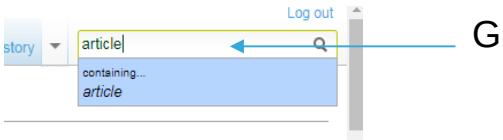
Prerequisites 2/4

TIPS:

- It is recommended for the wiki user guide navigation to keep your main article open in a separate window and to open additional window is you need to go to sub articles:

For example, if you want to run the starter kit article, keep it in a separate web browser window.

- Search window on top right of the wiki user guide allow to search on any pages title and content (global search). See below:



Global search

Prerequisites 3/4

A good knowledge about Embedded system is also required. It has to be completed with STM32MP1 training presentations:

1. {STM32MP1 Presentation} = What is the product, the associated tools and ecosystem
2. {STM32MP15-System- HW Architecture} = HW Architecture
3. {STM32MP15-Software- Software architecture} = SW Architecture

A good knowledge about Embedded Linux is required. If you think you are limited for some topics, you can check this article in the wiki user guide:

- [Linux_online]

Prerequisites 4/4

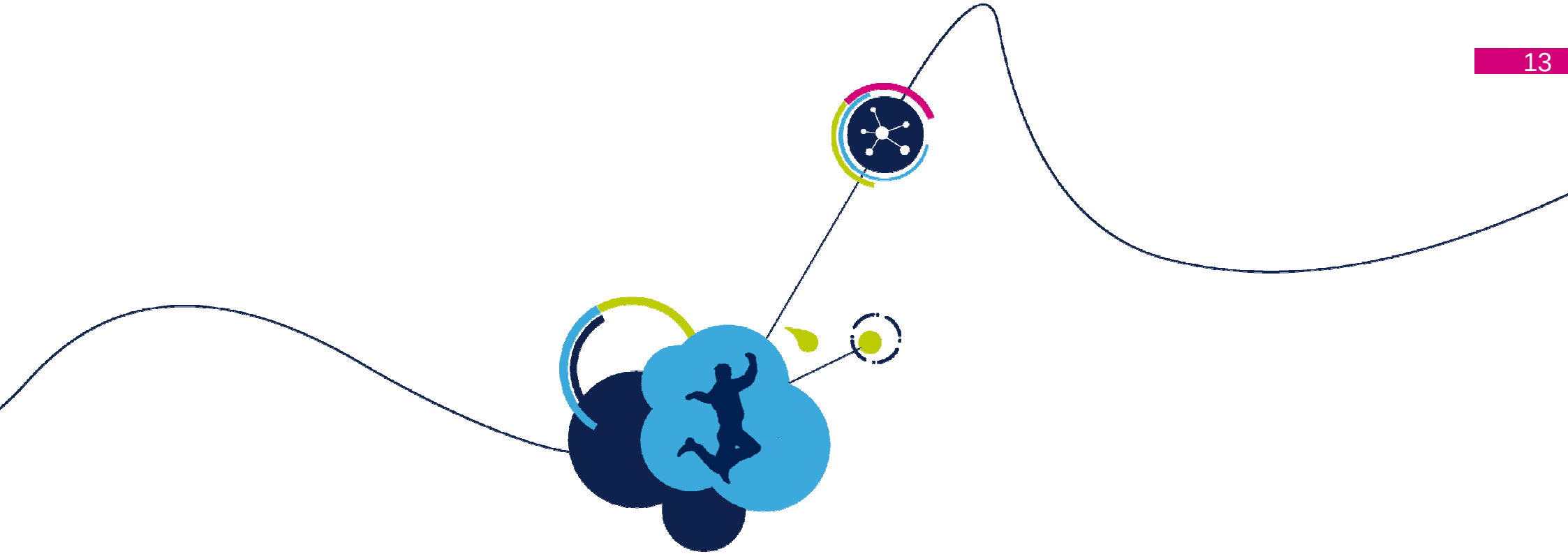
Some basis about Yocto (or Open Embedded framework) is mandatory. This build system is very specific so need a specific training. Some standard mechanisms are described here:

- [OpenEmbedded] is giving only few vocabulary and interesting links.

A STM32 MP1 board with some standard connectors (USB)

A standard PC running in priority (will come back in details later on):

- a UBUNTU LTS (latest one is better)
- Windows PC able to run a VMWare
- at least 4 cores, 100GB free disk and 8 GB DDR is recommended



Theoretical school

Duration: 4h

life.augmented

Theoretical school 1/2

HW Reference documentation:

Type	Link to document
Getting started with STM32MP15 Series hardware development	AN5031.pdf
STM32MP157C Data Brief	DB3372_STM32MP157Cxx.pdf
PP0058_STM32MP157Cxx Product preview	PP0058_STM32MP157Cxx_V0.3.pdf
RM0436 - Reference manual - STM32MP157xxx advanced Arm®-based 32-bit MPUs	RM0436_STM32MP157xx.pdf
STM32MP15xxx EVAL Mother Board schematics	MB1262C-01.pdf
STM32MP15xAA EVAL Daughter Board for 18x18 LFBGA448 schematics	MB1263B-01.pdf

Check for updates: https://github.com/OpenSTLinux/openstlinux_main#hardware-reference-documentation

Theoretical school 2/2

Reference Tools:

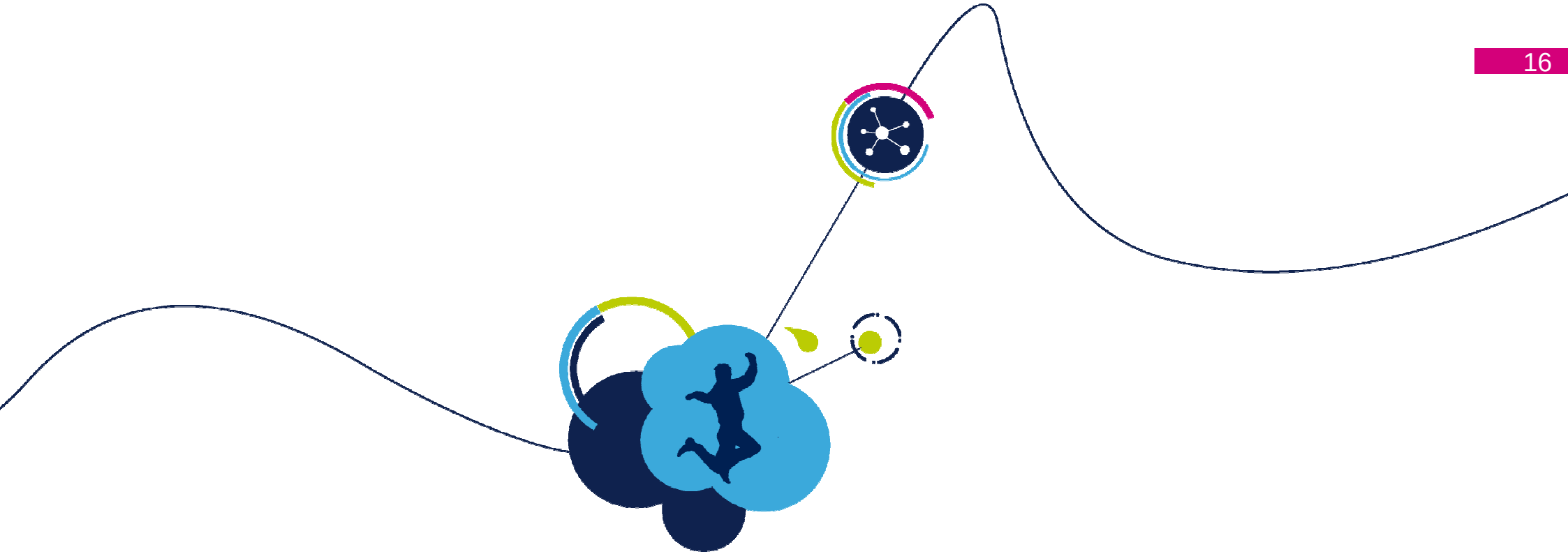
Type	Link to document	Version
STM32CubeProgrammer for LINUX	stm32_programmer_package_v1.0.3-MPU_64-bit.tar.gz To unpack: **PC $>** tar xvf stm32_programmer_package_v1.0.3-MPU_64-bit.tar.gz	1.0.3
STM32CubeProgrammer for WINDOWS	stm32_programmer_package_v1.0.3_MPU.zip	1.0.3

Check for updates:
https://github.com/OpenSTLinux/openstlinux_main#reference-tools

OpenSTLinux Training:

- {ST Distribution and build environment}
- [Which_kit_better_suits_your_needs] without going to sub link of the article

life.augmented



Practicing school

Practicing school: Starter kit 1/8

- Duration: 4h
- Objective: Enable the starter kit of your STM32MP1 board, understand what it is and what you can do (and not do) with it

✓ Lab 1: Starter Kit in wiki user guide

✓ Lab 2: Boot log analysis

✓ Lab 3: Load new application on your board and play with it

Starter kit Lab 1 objectives:

- Understand what is the starter kit and how to use it
- Practicing the starter kit with simple example
- Be familiar with starter kit wiki user guide

Practicing school: Starter kit 2/8

Lab 1: Starter Kit in wiki user guide

- Follow the starter kit of your board [STM32MP15 Starter kit]. Make sure you did these mandatory steps:

- ✓ Check your board assembly and connections.
- ✓ Identify the boot switch.
- ✓ Do all the "PC prerequisites" items.
- ✓ Download the delivered images.
- ✓ Install the STM32 Cube programmer and flash the images in your board.
- ✓ Install and use a remote terminal on your host PC to get a linux console.

- It is convenient to give access to the Cube Programmer binary anywhere on your PC. For example if "/home/bin" was on your $PATH:

```
PC $> ln -s /home/bin/STM32_Programmer_CLI
<Cube_programmer_install_dir>/stm32_programmer_package_v1.0.3-MPU_64-bit/bin/STM32_Programmer_CLI
```

Starter kit Lab 2 objectives:

- Realize all interesting logs given by the boot
- Few basis for debug

Practicing school: Starter kit 3/8

Lab 2: Boot log analysis

- Have a look at the boot log of the board. Your remote console is connected to the board. Press the reset button of the board:

- ✓ log starts with Reset reason
- ✓ U-boot log starting with parts of device init
- ✓ kernel image load and start (check the kernel command line)
- ✓ kernel init, memory layout and kernel device drivers init (interrupts, gpio, dma, peripherals, …)
- ✓ finish with network devices and Kernel console prompt

Practicing school: Starter kit 4/8

Lab 2 TIPS:

- You can easily search backward in the console with <ctrl> <shift> f (practical to find events from the most recent events). For example try to find "stm32-ipcc" (mailbox init).
- You can do the same but from the beginning with dmesg command:

```
Board $> dmesg | grep stm32-ipcc
Board $> dmesg –C (to clear the buffer)
```

- Check again all the information you can see in the boot log. Many issues are revealed during the boot and many information useful for debug too.
- You can also check the clock setting after boot from the console (for advanced investigations):

```
Board $> cat /sys/kernel/debug/clk/clk_summary
```

Starter kit Lab 3 objectives:

- Load and play a reference application with the starter kit
- Few basis of interesting commands

Practicing school: Starter kit 5/8

Lab 3: Load new application on your board and play with it

- Check [STM32MP15 Starter kit] article for your board, you should find the way to load any file from your host PC to your running board. Two ways:
  - "scp" command: you need Ethernet (same network) connection between your host and your board, this is the more common way to work.
  - "kermit" command: backup solution through the linux console see [File transfer over serial console: kermit].

In the following, we consider only "scp" use case.

- Upload the applications provided in your hands-on package (gpio-tools from the kernel):

```
PC $> scp -r openstlinux-hands-appli/kernel_install_dir/* root@<ip address>:/
```

- List all the gpio lines of the board (it will be useful for some further steps):

```
Board $> lsgpio
```

- Launch gpio-hammer application (Ctrl+c to exit):

```
Board $> gpio-hammer -n gpiochip0 -o 14
```

You will see the user led linked to PA14 gpio blinking at 1 Hz.

life.augmented

Practicing school: Starter kit 6/8

- Launch gpio-hammer application (Ctrl+c to exit):

```
Board $> gpio-event-mon -n gpiochip0 -o 13
```

By pushing the PA13 user button on the board, you will see GPIO EVENT information in the console.

- Launch a new application (you will compile in Developer kit section):

```
Board $> openstlinux-hands
```

Press the PA13 user button on the board and you will toggle the green user led. This Application has limited interest but will be used afterwards as the reference use case for the developer kit.

- Launch a new application in background:

```
Board $> openstlinux-hands &
Board $> lsgpio
```

You can see the 2 gpios PA13 and PA14 associated to the use case.

Practicing school: Starter kit 7/8

Lab 3 TIPS:

- It is convenient to use 2 terminals (see [STM32MP15 Starter kit] article for your board) :
  - one linux console for command entry
  - one ssh console for trace output

Practicing school: Starter kit 8/8

Q&A session (if possible with support contact):

- Duration: 1/2h
- Objective: Check all the topics mentioned in starter kit section were well done and there was no issue. It is mandatory for the next steps, especially:
  - connection and boot of the board (boot switch, interfaces and user buttons)
  - Linux console connection and boot sequence overview (dmesg log for debug)
  - load and run applications on the board
- Attendees: From beginners to experts

Practicing school: Developer kit 1/11

- Duration: 2d
- Objective: Enable the developer kit for your STM32MP1 device, understand what it is and what you can do (and not do) with it

✓ Lab 1: Developer Kit in wiki user guide

✓ Lab 2: Build your own application: openstlinux-hands

✓ Lab 3: Develop your own kernel driver

Developer kit Lab 1 objectives:

- Understand what is the developer kit and how to use it
- Practicing the developer kit with simple example
- Be familiar with developer kit wiki user guide

Practicing school: Developer kit 2/11

Lab 1: Developer Kit in wiki user guide

- Follow the developer kit of your STM32MP1 device [STM32MP1_Developer_kit]. Focus on new steps you did not already performed during the previous starter kit stage:

- ✓ Understand the composition of your OpenSTLinux BSP (AFT, U-boot and Linux Kernel).
- ✓ What is the SDK and how to install/use it?
- ✓ Download and compile at least the kernel Linux source package.
- ✓ Cross-compile and deploy a simple user space application.
- ✓ Cross-compile and deploy an external out-of-tree Linux kernel simple module.

- You can only update some parts of your starter kit (BSP) or upload a user application.
- You cannot rebuild the whole starter kit (the rootfs or user land framework).

Practicing school: Developer kit 3/11

Lab 1 TIPS:

- It is convenient to install the developer kit together with starter kit (they are closely linked together), for example:

```
PC $> mkdir ALPHA_DEVKIT
PC $> cd ALPHA_DEVKIT
```

Then untar what you need: SDK, starter kit, Linux kernel sources, … All will be well organized and inline with Open Embedded organization (you will see with the distribution kit)

- For linux kernel sources installation and build (see README.HOW_TO.txt), it is recommended:
  - To use git to track your changes (if you know git), it will ease the distribution kit lab.
  - To configure and build in a separate output folder if your development is substantial.
  - Anyway to use a common kernel installation folder (image of what you will upload on your board).

Developer kit Lab 2 objectives:

- Use the developer kit to develop your own application
- Understand developer kit and Makefile links

Practicing school: Developer kit 4/11

Lab 2: Build your own application: openstlinux-hands

- This application is the one you ran with the starter kit, here you will develop it.

✓ It will use the IOCTL interface of GPIOLib framework see [GPIOLib_overview].

- On top of this article, all you need to know is available in these files:
  - https://elixir.bootlin.com/linux/latest/source/tools/gpio/gpio-event-mon.c
  - https://elixir.bootlin.com/linux/latest/source/tools/gpio/gpio-hammer.c
  - https://elixir.bootlin.com/linux/latest/source/tools/gpio/gpio-utils.c

- Architecture of the main program is this one:

```
/*  Open device: gpiochip0 for GPIO bank A */
/* request GPIO line: GPIO_A_14 for Led switching and GPIO_A_13 for Button activation*/
/*  Start main loop */
while(1) {
        /* read GPIO_A_13 input event */
        /* process the event received  and update the led value */ }
/*  Close device: gpiochip0 */
```

Practicing school: Developer kit 5/11

- Note: GPIO_A_14 and GPIO_A_13 are defined in the device tree of the board but not activated by default (status = "disabled"). Do not change it. If you activate them they will be taken by kernel frameworks and not available for your application.

- Try to develop this application using the developer kit and run it on the board.
  - The interest here is to understand how to structure both your makefile with your application and do some debug around it.

✓ One solution is provided in your hands-on package in "openstlinux-hands-appli" directory. Please check it.

Practicing school: Developer kit 6/11

Lab 2 TIPS:

- To understand "patsubst" function (commonly used in Makefiles): https://www.gnu.org/software/make/manual/html_node/Text-Functions.html
- Makefiles have to be customized matching your needs and your habits of development but the structure is quite similar from one to another.
- You can see the different options of a local make with the tab key:

```
PC $> make <tab>
All          clean          install          openstlinux-hands
```

- Any time you open a new terminal, you will need to run the SDK env setup source to position your tool chain variable. A quick way to check you are with right setup is:

```
PC $> env | grep ARCH
ARCH=arm
```

- Standard debugger for application development is GDB. See [GDB] and especially [gdbgui] usage.

Developer kit Lab 3 objectives:

- Use the developer kit to develop your own kernel driver
- Understand developer kit and kernel links

Practicing school: Developer kit 7/11

Lab 3: Develop your own kernel driver

- The purpose here is to do the same use case as previous one but within the kernel.
- We will need for that the kernel in full source (as provided in the developer kit delivery).
- Many different ways are possible:
  - built-in kernel module = kernel internal module (part of kernel uimage)
  - kernel external module = module is dynamically loaded after kernel init
  - out-of-tree external module = module source is outside the kernel source tree (over way is in-tree external module)
- We will focus on the last one for this lab.

Practicing school: Developer kit 8/11

- [STM32MP1_Developer_kit] article is providing an example of such use case (this gives you a template of your program). The new driver name can be "push_led_driver".
- A specific node will first need to be added to your device tree (device tree of your board):
  - It will initialize the 2 gpios used: GPIO_A_14 and GPIO_A_13. These labels will be used by gpiolib interface.
  - It will create also the link between these gpios and the driver (« compatible » field).
  - You can check [How_to_control_a_GPIO_in_kernel_space] for help.

- Your program may need to use these functions:
  - gpiolib: « devm_gpiod_get », « gpiod_to_irq », « gpiod_set_value » and « gpiod_get_value »
  - irqchip: « devm_request_any_context_irq » to handle interrupt generated by GPIO_A_13 button

Practicing school: Developer kit 9/11

- With https://elixir.bootlin.com/linux/latest/source, you will easily find with the identifier search field:
  - The definitions of the functions and associated structures.
  - Many examples of usage provided by kernel community drivers.
  - The best way to use them and manage errors.

- Try to develop this driver using the developer kit and run it on the board.
  - The interest here is to understand how to structure both your makefile with your driver, the interconnection with the device tree and the relationship with the kernel source.
  - Do not forget to upload your new compiled device tree on the board.

✓ One solution is provided in your hands-on package in "openstlinux-hands-driver" directory. Please check it.

Practicing school: Developer kit 10/11

Lab 3 TIPS:

- The delivered Linux kernel source code is a git extraction, then by default your module will get a dirty version (not compatible with the starter kit version). See the "README.HOW_TO.txt" file (".scmversion" file).
- After you updated the ".scmversion" file, you will need to rebuild your kernel first then your module to make it clean.
- Don’t forget to reboot the board to take device tree modifications into account.
- To make your external module probed at boot, use "vi" to create .conf file like below :

```
Board $> vi /etc/modules-load.d/openstlinux-hands-driver.conf
Board $> cat /etc/modules-load.d/openstlinux-hands-driver.conf
openstlinux-hands-driver
```

Practicing school: Developer kit 11/11

Q&A session (if possible with support contact):

- Duration: 1h
- Objective: Check all the topics mentioned in developer kit section were well done and there was no issues. Mandatory steps:
  - SDK installation and use of cross compilation to upload embedded SW programs
  - rebuild the kernel and device tree then update the starter kit (rapid cycle)
  - makefile organization related to the developer kit (SDK + source code packages)
- Attendees: developers in Embedded Linux

Practicing school: Distribution kit 1/15

- Duration: 2d
- Objective: Use the distribution kit to integrate the development done with the developer kit:
  - First exercise is to integrate your user land application in STM32 MP1 distro.
  - Second is to integrate your device driver in STM32 MP1 machine.
  - Last is to integrate some Open Embedded layers on STM32 MP1 distribution.
- Reference: Yocto project is providing many online documentation regularly updated, see:
  - https://www.yoctoproject.org/docs/, search for "mega-manual" link.

Practicing school: Distribution kit 2/15

- All the following labs will start from the Distribution Kit delivered by ST
  - See [STM32MP1 Distribution kit] and focus on “OpenSTLinux distribution”:

Download the kit and build the latest release.

✓ Lab 1: Integrate your user land application in STM32 MP1 distro.

✓ Lab 2: Integrate your device driver in STM32 MP1 machine.

✓ Lab 3: Integrate some Open Embedded layers to ST distribution.

Distribution kit Lab 1 objectives:

- Understand what is the distribution kit and how to use it
- Be familiar with distribution kit wiki user guide
- Use the distribution kit to integrate your own application

Practicing school: Distribution kit 3/15

Lab 1: Integrate your user land application in STM32 MP1 distro

- Remind the application in "openstlinux-hands.c" you developed with the developer kit. We are going to create a new generic layer for this application and add it in the standard build image (st-image-Weston).

- Use devtool (see [OpenEmbedded - devtool]) to develop and validate your recipe (openstlinux-hands):
  - Take back your hands-on package installed in your <working_dir> directory.

PC $> devtool add openstlinux-hands <working_dir>/hands-on/openstlinux-distribution/openstlinux-hands-appli

- ✓ Devtool has created a recipe based on what it has analyzed from your <working_dir> path. It has also added a workspace layer to be able to work with it.
- ✓ Check conf/bblayers.conf (see the new "workspace" layer) then "workspace" directory content.
- ✓ Check the proposed « openstlinux-hands.bb » recipe:
  - ✓ SRC_URI is empty: Check the bbappend recipe to understand how the src are found.
  - ✓ Oe_runmake was proposed to the compile and install tasks (as a Makefile was present in the source directory). Yocto will call it to build and install the component.

Practicing school: Distribution kit 4/15

- Try to build it, load and test:

```
PC $> devtool build openstlinux-hands
PC $> devtool deploy-target openstlinux-hands root@<board ip@>
```

```
Board $> openstlinux-hands
```

- The recipe is now validated with devtool, we will have to integrate it in the final layer

- Create your new layer: See [How to create a new open embedded layer].
  - Create a new layer named "meta-open-hands-layer" in meta-st directory and add it to your build system ("-p 6" for priority and "–e openstlinux-hands" for recipe name).

✓ Check content (from build dir):

```
PC $> tree ../meta-st/meta-open-hands-layer/
├── conf
│   └── layer.conf
├── COPYING.MIT
├── README
└── recipes-openstlinux-hands
    └── openstlinux-hands
        └── openstlinux-hands.bb
```

life.augmented

Practicing school: Distribution kit 5/15

- ✓ Check "conf/bblayers.conf":
  - A new layer was added to the BBLAYERS variable (if you did "bitbake-layers add-layer" as recommended).
- ✓ Check your layer "conf/layer.conf": There might be a bug in the "create-layer" command. You need to replace the "\\" with single one "\" (if bug not solved). It will be required for the next steps.

- The layer you have created is a template (so almost empty). Now we have to finalize this layer so that it can be used like others.
- Copy the recipe designed with devtool (in "workspace" directory) to this new layer, edit this recipe.
- For the SRC_URI, You need to tell Yocto what to process and where to find them:
  - There are numerous solutions for that and "Openembedded-core" package is full of examples especially the "meta-skeleton" layer.

Practicing school: Distribution kit 6/15

- Most of the SRC_URI rely on a git repository but for this hands on, we will store the files directly inside the layer. The method below is commonly used:

```
SRC_URI =
"file://openstlinux-hands.c \
                file://Makefile"
S = "${WORKDIR}"
```

"$S" variable is the source directory for Yocto.

- Then the sources will be stored like below (directory name is recipe name):

```
└── openstlinux-hands
    ├── openstlinux-hands
    │   ├── Makefile
    │   └── openstlinux-hands.c
    └── openstlinux-hands.bb
```

- Copy your developer kit files like mentioned above (new "openstlinux-hands" and copy files) in your "meta-open-hands-layer" layer

- Remove your devtool workspace:

```
PC $> devtool reset openstlinux-hands
```

life.augmented

Practicing school: Distribution kit 7/15

Test your new layer

- Build your recipe (for a quick check) then your full image:

```
PC $> bitbake openstlinux-hands
PC $> devtool build-image -p openstlinux-hands st-image-weston
```

- Flash the full image and test on the board:

```
Board $> openstlinux-hands
```

- OR, You may use « devtool deploy-target » as a quicker way like previously

Finalize your layer

- Update your recipe with the license information (LICENSE and LIC_FILES_CHKSUM): GPLv2 in our case. The recipe license is MIT, do not mix it with the source license. The checksum is the one from the license text and it is always checked by bitbake before building.
- Add your recipe to the selected image:
  - Create a new bbappen file: st-image-weston.bbappend.
  - Add IMAGE_INSTALL_append = " openstlinux-hands" in this file.
- Add your layer to ST distribution:
  - Check: meta-st/meta-st-openstlinux/conf/template/bblayers.conf.sample.
  - Add your layer to ADDONSLAYERS variable like done for others.

Practicing school: Distribution kit 8/15

Lab 1 TIPS:

- Your developer kit application makefile contains this line:

"DESTDIR ?= ./kernel_install_dir/usr/local/bin"

This is very common and match how devtool/yocto is working (defining DESTDIR and bindir).

- You can get reference of "meta-skeleton" layer to get licensing recipe information up to date in your recipe
- After you have successfully integrated your application or framework, you can regenerate your complete developer kit:
  - Generate SDK see [populate_sdk].
  - Generate Starter and Developer kits see [Generating your own Starter and Developer kits].
  - The use case is to go back in a developer kit cycle with your own distribution or a customized ST distribution.

Distribution kit Lab 2 objectives:

- Use the distribution kit to integrate your own kernel driver
- Be familiar with devtool Yocto tool

Practicing school: Distribution kit 9/15

Lab 2: Integrate your device driver in STM32 MP1 machine

- As for lab 1, use devtool to make a recipe template but using kernel driver code
- From yocto build dir:

```
PC $> devtool add openstlinux-hands-machine <working_dir>/hands-on/openstlinux-distribution/openstlinux-hands-driver/
```

- ✓ Check the recipe, you will see:
  - ✓ KERNEL_DIR variable automatic setting
  - ✓ inherit module (to inherit module.bbclass)
- Build and deploy it like on lab 1.
- It is not working, Why?

Practicing school: Distribution kit 10/15

- Previous trial was not working because we did not handle the device tree modifications: we will use for that "devtool modify" tool.

- In your build directory:

```
PC $> devtool modify linux-stm32mp
PC $> tree -L 3 workspace/
```

You will see a new sources directory with the kernel sources and the linux-stm32mp directory contains a .git directory: you are on a git extraction of the ST kernel code !

- Patch the device tree with the patch from the developer kit hands on, and commit your modification (you may “git am” to do it in one step).

✓ You can see (for example with gitk tool): a new commit has been added on "devtool" branch. You can also check with "git status" command, there is nothing left to commit.

Practicing school: Distribution kit 11/15

- Build and test the new device tree modifications on your board (“devtool build” then “deploy-target”)
- Do not forget to build the new dtb files (not obvious in yocto)

- Now, you have validated your modifications with devtool, you need to integrate them in your layer:
  - Integrate "openstlinux-hands-machine" recipe in your layer like done in lab 1 (add directory tree and files)
  - Add your "openstlinux-hands-machine" recipe to "st-image-Weston" like lab 1
  - Integrate your modified recipe in your layer:

PC $> devtool finish linux-stm32mp meta-open-hands-layer

✓ Check what the previous command did for you. This command is magic !

- Test build then deploy on target and test

Practicing school: Distribution kit 12/15

Lab 2 TIPS

- List what devtool will deploy on target (when you are working with "workspace"):

```
PC $> devtool deploy-target -n linux-stm32mp root@<board ip@>
```

- “Deploy-target” of the kernel leads to error for the moment: you have to remove wmlinux (automatically generated by the kernel recipe) from the image directory before the deploy.
- List all the tasks supported by a recipe:

```
PC $> bitbake -c listtasks linux-stm32mp
```

- To force the rebuild of the kernel device tree, you can "touch" the kernel recipe and rebuild it with devtool (when you are working with "workspace").
- "%" wildcard for recipe version means it will apply to all the versions.
- You can cleanup your build space with bitbake command:

```
PC $> bitbake openstlinux-hands-machine -c cleanall
```

Practicing school: Distribution kit 13/15

- You may redefine "STAGING_KERNEL_BUILDDIR" variable (to build outside source code directory).
- OR you can use "--no-same-dir" option of devtool.

Distribution kit Lab 3 objectives:

- Integrate and configure a new layer coming from Yocto community
- Basics for debugging

Practicing school: Distribution kit 14/15

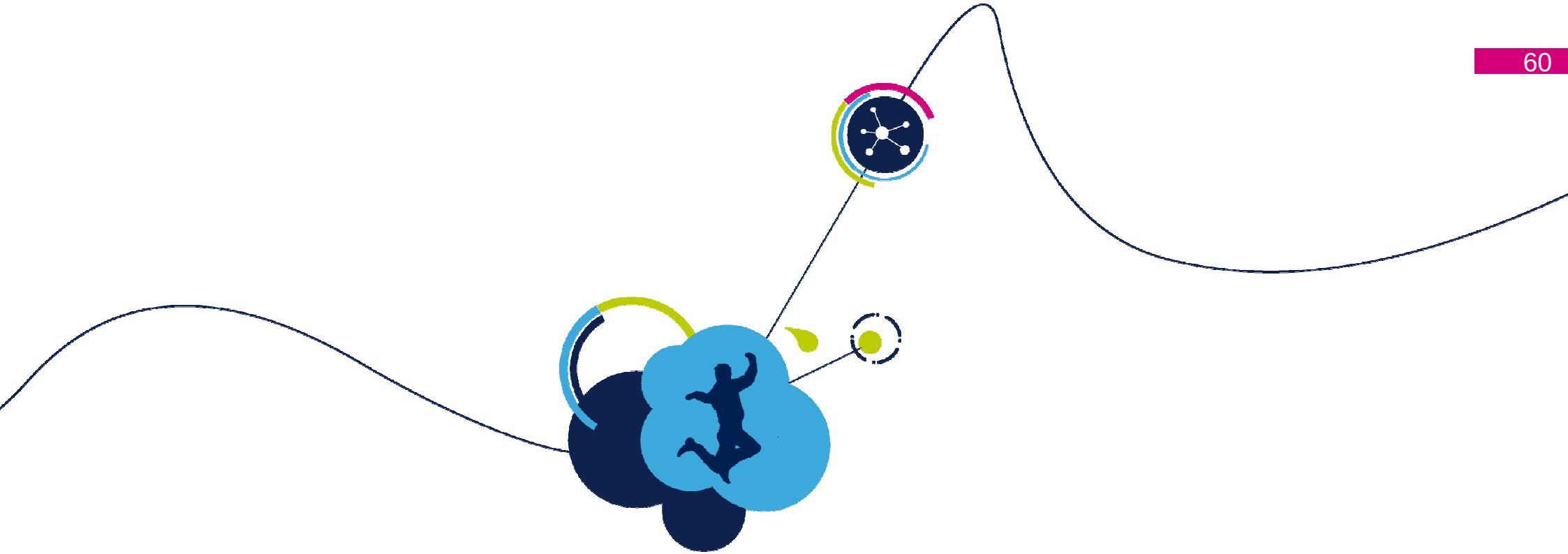
Lab 3: Integrate some Open Embedded layers to ST distribution

- See [How_to_install_JAVA_JDK]. Only compatible from Alpha2 release.

Practicing school: Distribution kit 15/15

Q&A session (if possible with support contact):

- Duration: 1h
- Objective: Check all the topics mentioned in distribution kit section were well done and there was no issues. Mandatory steps:
  - Add own layer and own recipes based on ST distribution kit.
  - Ability to produce a custom distribution and to regenerate a new developer kit.
  - Integration of other open embedded layers coming from community.
- Attendees: developers/integrators in Embedded Linux



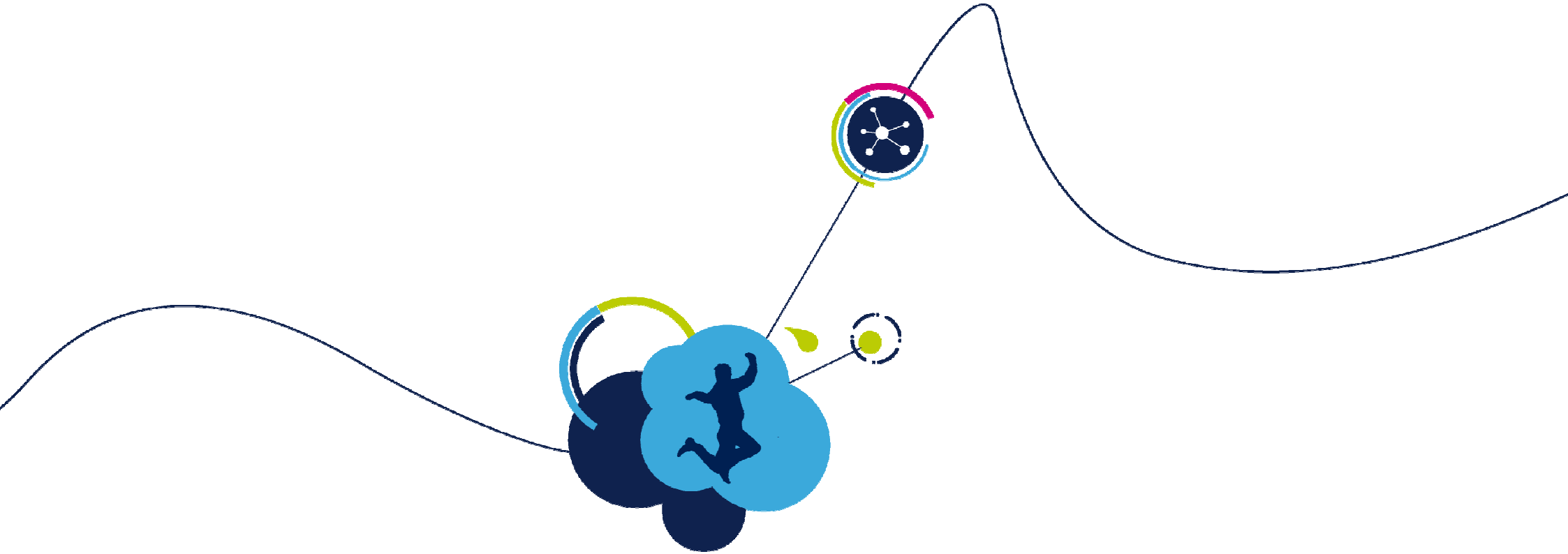
Evaluation

Evaluation

Goal is to check the trainee has understood most of the kits. He knows how to manage them and he has all the knowledge to start working with them.

This evaluation can be formalized with a synthesis of the three Q&A sessions for the different kits with specific focus on:

- Were the prerequisites enough to play the labs and if not done some additional trainings should be planed?
- Is the structure and way to find information (wiki principally) well understood and used?
- Check what could not be done and which appropriate corrective action need to be taken?
- What were the blockers? What are the way to improve the hands on and ramp up of the trainee?



Congratulations !

life.augmented