Eat – Sleep –Code - Repeat

NXP

University Course

Course Contents

General topics, **Courses** & **Labs**

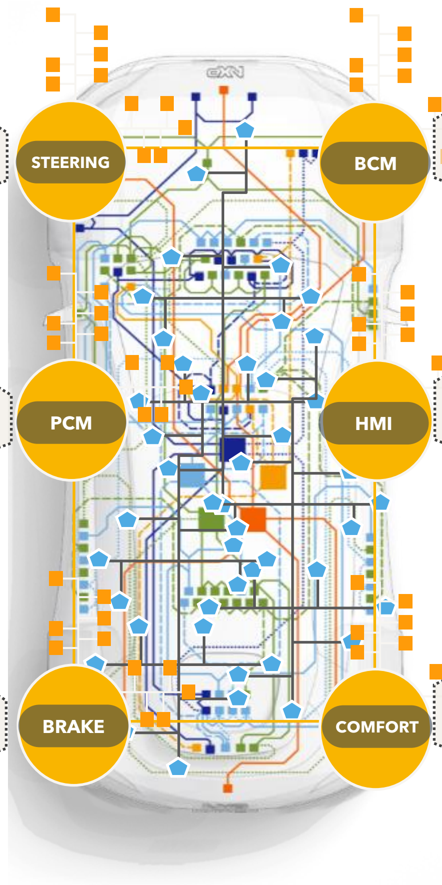
1. General Presentation of the Course
2. V-Model, Requirements Engineering, Process | Understand and Create Requirements
3. Architecture (UML) | Virtual Machine Environment Setup
4. Process | Git, IDE Setup, Compile and Flash the Hello World Project
5. How Hardware and Software are Linked | From Compiling to Electrical Signals and Debugging
6. Microcontroller Features (I/O, PWM, ADC, DAC, Timer, Interrupts) | Hands-on Lab (no module – just the dev board)
7. Node 1: Lights (BCM)
8. Node 2: Steering
9. Node 3: Transmission
10. Node 4: Brakes
11. Node 5: Door Control + HVAC
12. Node 6: HMI and CAN Communication

REPLI-CAR NETWORK

STEERING CONTROL NODE
Replicate Steering Control Module behavior
· steer base on input controller (steer controller) from HMI
· Communications with other nodes: HMI, PCM, BCM

POWERTRAIN CONTROL NODE
Replicate Engine and Transmission Modules behavior
· Accelerate/Decelerate based on input controller from HMI
· Communications with other nodes: HMI, Brake, BCM

BRAKE CONTROL NODE
Replicate Brake Control Module behavior
· brake actuation and brake lamp indication based on input controller from HMI
· Communications with other nodes: HMI, PCM, BCM



BODY CONTROL NODE
Replicate Body Control Module behavior
· Actuate lights (headlamps, turn indicators), horn and washer pump based on input controller from HMI
· Communications with other nodes: HMI, PCM, Brake, Steering, Comfort

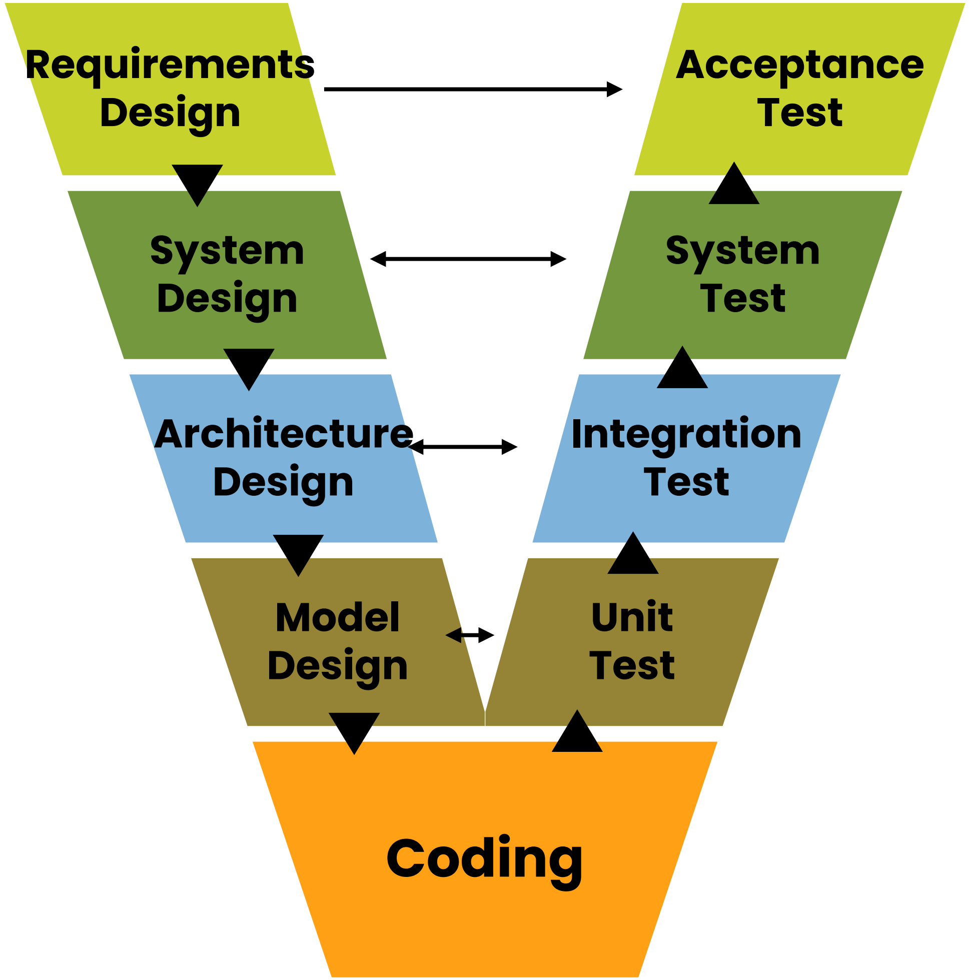
HMI NODE
Replicate Human Machine Interface Module behavior
· handle input/output controllers that actuate different nodes in Repli-Car network. Display information received from the nodes
· Communications with other nodes: Steering, Comfort, PCM, BCM

COMFORT CONTROL NODE
Replicate Comfort Module behavior
· actuate window motor and ventilator using input controller commands from HMI
· Communications with other nodes: HMI, BCM

V-Model

nxp.com

The V-MODEL



- Known as **verification** and **validation** model
- It is similar to waterfall model that follows a sequential path of execution of processes
- Devised by the late Paul Rook in 1980s, V-model was targeting the **improvement of software development**
- It is a **step-by-step process** in which the next phase begins only after the completion of the present phase
- If this model is used to test a product, there is an **assurance** that the final product developed will be **of high quality**

Phases of V-Model – Verification phase

The verification phase of V-model includes:

- **Business requirement** analysis is the stage of having a **detailed communication with the customer** so that it gets easier **to understand his/her exact requirements**.
- **System design** stage involves **understanding and detailing out the entire hardware and communication setup** for the product being developed. System test design can also be planned at this stage
- **Architectural design** stage involves **understanding the technical and financial feasibility of the product before it is actually developed**. The focus is to understand the data transfer that will take place between internal and external modules
- **Module design** stage focuses on **designing a detailed plan for the internal modules** of the system. Also known as low-level design, it is important to ensure that the design is compatible with other modules in system architecture and other external systems

Phases of V-Model – CODING phase

- During this phase, the actual **coding** of the system modules is taken up
- On the basis of system and architectural requirements of the program, the best suitable programming language is selected using which the coding is done at par with the coding guidelines and standards; the code is then reviewed and optimized to ensure the delivery of best performing product

Phases of V-Model – Verification phase

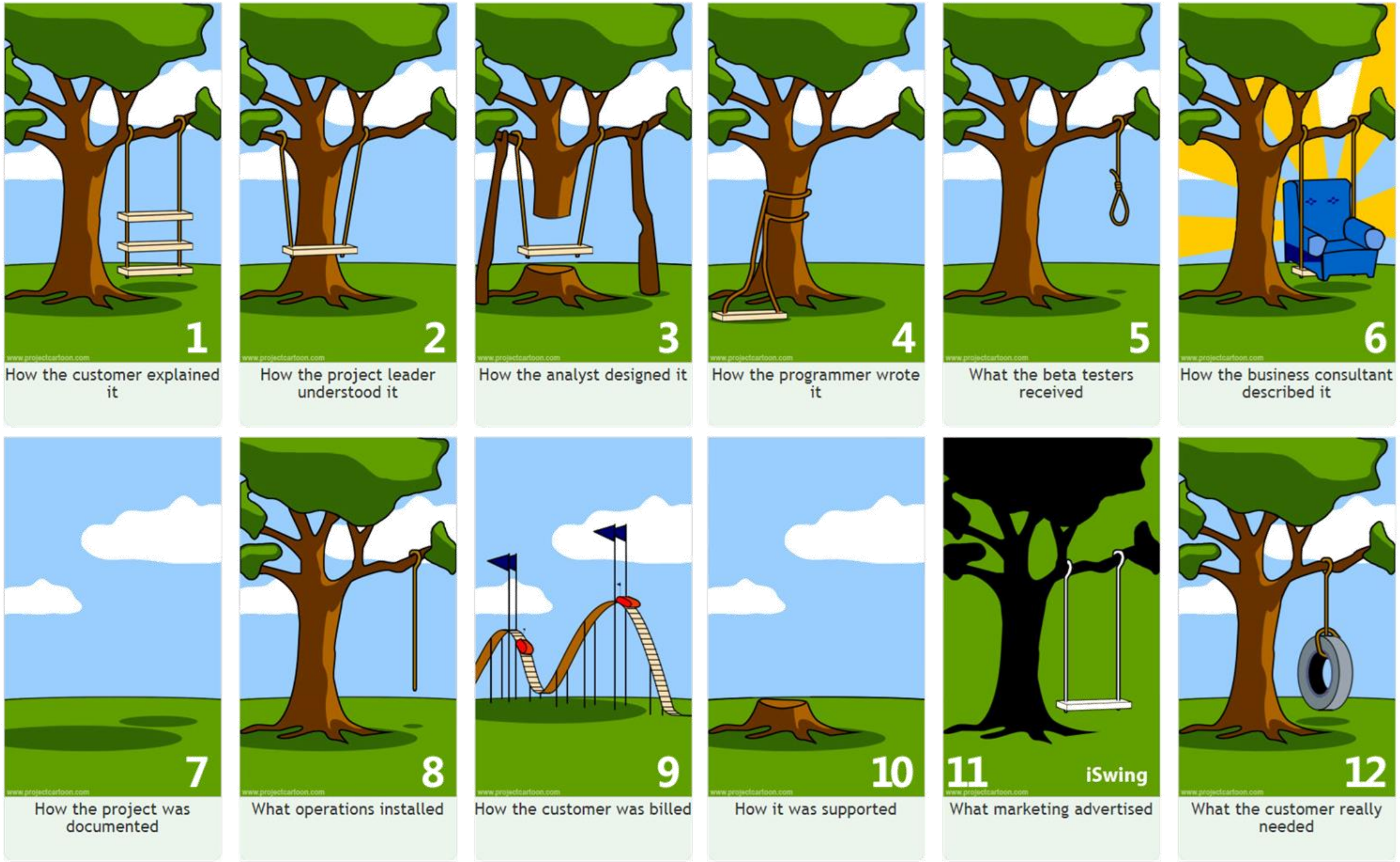
During this phase, the product undergoes various forms of testing.

- **Unit testing** is conducted at an early stage so that the bugs are eliminated at the starting stages of product development
- **Integration testing** is done to check whether there is a valid and proper communication within the internal modules of the system
- **System testing** enables the testing of the entire system and to ensure if the internal modules communicate effectively with the external systems
- **Acceptance testing** is done to test a product in the user's environment and to check if it's compatible with the other systems available in the environment

Requirements Engineering

nxp.com

What is requirements engineering?



What is requirements engineering?

- **Requirements Engineering** (RE) is the **process of discovering, analyzing, documenting and validating** the requirements of the system
- Each software development process goes through the phase of requirements engineering
- The processes used for RE vary widely depending on the application domain, the people involved and the organization developing the requirements
- However, there are a number of generic activities common to all processes:
  - Requirements elicitation
  - Requirements analysis
  - Requirements validation
  - Requirements management

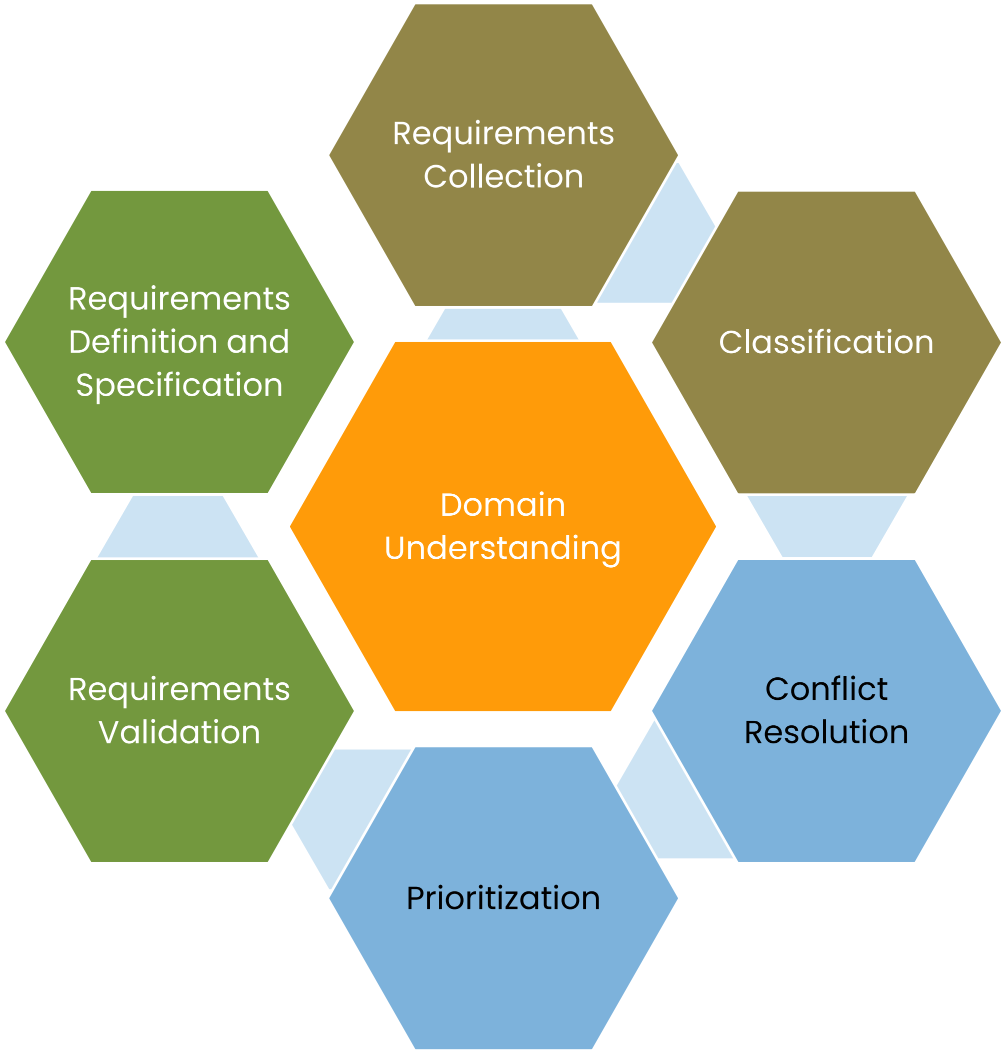
Requirements engineering – Elicitation and analysis

- Sometimes called **requirements elicitation** or **requirements discovery**
- Involves technical staff working with customers to **find out about the application domain**, the services that the system should provide and the system’s operational constraints
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called **stakeholders**

Requirements engineering – Problems of requirements analysis

- Stakeholders **don’t know what they really want**
- Stakeholders **express** requirements **in their own terms**
- Different stakeholders may have **conflicting requirements**
- Organizational and political factors may **influence the** system **requirements**
- The **requirements change** during the analysis process.
- New **stakeholders may emerge** and the business environment change

The requirements analysis process



Process activities & Viewpoint-oriented elicitation

- **Process activities**:
  - Domain understanding
  - Requirements collection
  - Classification
  - Conflict resolution
  - Prioritization
  - Requirements checking

- **Viewpoint-oriented elicitation**:
  - Stakeholders represent different ways of looking at a problem or problem viewpoints
  - This multi-perspective analysis is important as there is no single correct way to analyze system requirements

Requirements validation

- **Validation**
  - Concerned with **demonstrating** that the **requirements define the system** that the customer really wants
  - **Requirements error costs are high**, so validation is very important
  - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error

- Requirements **management** is **the process of managing changing requirements** during the requirements engineering process and system development
  - **Requirements** are inevitably **incomplete and inconsistent**
  - **New requirements** emerge during the process as business needs change and a better understanding of the system is developed
  - Different viewpoints have **different requirements** and these are **often contradictory**

Enduring and volatile requirements

- **Enduring requirements**
  - Stable requirements derived from the core activity of the customer organization. E.g. a hospital will always have doctors, nurses, etc. May be derived from domain models
- **Volatile requirements**
  - Requirements which change during development or when the system is in use. In a hospital, requirements derived from health-care policy

Classification of requirements

- **Mutable requirements**
  - Requirements that change due to the system's environment
- **Emergent requirements**
  - Requirements that emerge as understanding of the system develops
- **Consequential requirements**
  - Requirements that result from the introduction of the computer system
- **Compatibility requirements**
  - Requirements that depend on other systems or organizational processes

Traceability

- **Traceability** is concerned with the **relationships** between requirements, their sources and the system design
- **Source traceability**
  - Links from requirements to stakeholders who proposed these requirements
- **Requirements traceability**
  - Links between dependent requirements
- **Design traceability**
  - Links from the requirements to the design

Writing Good Requirements

- A good requirement states something that is **necessary, verifiable**, and **attainable**.
  - **Need**. If there is a doubt about the necessity of a requirement, then ask: What is the worst thing that could happen if this requirement were not included? If you do not find an answer of any consequence, then you probably do not need the requirement.
  - **Verification**. As you write a requirement, determine how you will verify it.
  - **Attainable**. To be attainable, the requirement must be technically feasible and fit within budget, schedule, and other constraints.
  - **Clarity**. Each requirement should express a single thought, be concise, and simple. It is important that the requirement not be misunderstood – it must be unambiguous.

- *Simple sentences will most often suffice for a good requirement.*

Writing Good Requirements - Use of terms

- In a specification, there are terms to be avoided and terms that must be used in a very specific manner. Authors need to understand the use of shall, will, and should:
  - **Requirements** use **shall**
  - **Statements** of fact use **will**
  - **Goals** use **should**.
- These are standard usage of these terms. You will confuse everyone if you deviate from them. **All shall statement (requirements) must be verifiable**, otherwise, compliance cannot be demonstrated.
- Terms such as **are**, **is**, **was**, and **must** *do not belong in a requirement*. They may be used in a descriptive section or in the lead-in to a requirements section of the specification.
- There are several terms to be avoided in writing requirements, because they confuse the issue and can cost you money, e.g.: support, but not limited to, etc., and/or

Writing Good Requirements - Use of terms

- The word **support** is often used incorrectly in requirements. Support is a proper term if you want a structure to support 50 pounds weight. It is incorrect if you are stating that the system will support certain activities.
  - **WRONG:** The system **shall support the training coordinator** in defining training scenarios.
  - **RIGHT:** The system **shall provide input screens for defining training scenarios.** The system shall provide automated training scenario processes.
- The terms **but not limited to**, and **etc**. are put in place because the person writing the requirements suspects that more may be needed than is currently listed. *Using these terms will not accomplish what the author wants and can backfire.*

Writing Good Requirements – Structure/grammar

- Requirements **should be easy to read and understand**. The requirements in a system specification are either for the system or its next level (e.g.: subsystem). Each requirement can usually be written in the format:
  - The System **shall provide …**
  - The System **shall be capable of…**
  - The System **shall weigh…**
  - The Subsystem #1 **shall provide …**
  - The Subsystem #1 **shall interface with…**
- *Note: The name of your system and the name of each subsystem appear in these locations. If you have a complex name, please use the acronym, or your document will have many unneeded pages just because you have typed out a long name many times*

Writing Good Requirements – Structure/grammar

- Each of these beginnings is followed by **what** the System or Subsystem shall do.
- Each should generally be followed by **a single predicate**, not by a list.
- Requirement statements **should not be complicated** by explanations of operations, design, or other related information. This non-requirement information should be provided in an introduction to a set of requirements or in rationale.
- You can accomplish two things by rigorously sticking to this format:
  - First, you avoid the **Subject Trap**.
  - Second, you will **avoid bad grammar** that creeps into requirements when authors get creative in their writing

Writing Good Requirements – Structure/grammar - Subject trap

- A set of requirements might be written that reads as follows:
    - The **guidance and control subsystem** shall provide control in six degrees of freedom.
    - The **guidance and control subsystem** shall control attitude to 2+/- 0.2 degrees.
- The **subject trap** is created because **the author has defined a guidance and control subsystem**. Controlling attitude and rate is a **system** problem; it requires not only a guidance and control subsystem *but also a propulsion subsystem to achieve these attitudes and rates*. The requirements should be **written from the system perspective**, as follows:
    - The **system** shall provide six degrees of freedom control.
    - The **system** shall control attitude to 2 +/- 0.2 degrees.

Writing Good Requirements – Structure/grammar - Bad grammar

- **Bad grammar** → reader will misinterpret what is stated.
- Avoid it by **writing requirements as bullet charts**. When the content is agreed upon a good writer can convert the information into a sentence for the specification.
- **Authors will also try to put all that they know in a single sentence**. This results in a long complex sentence that probably contains more than one requirement. Bullet charts or one good editor can alleviate this problem.

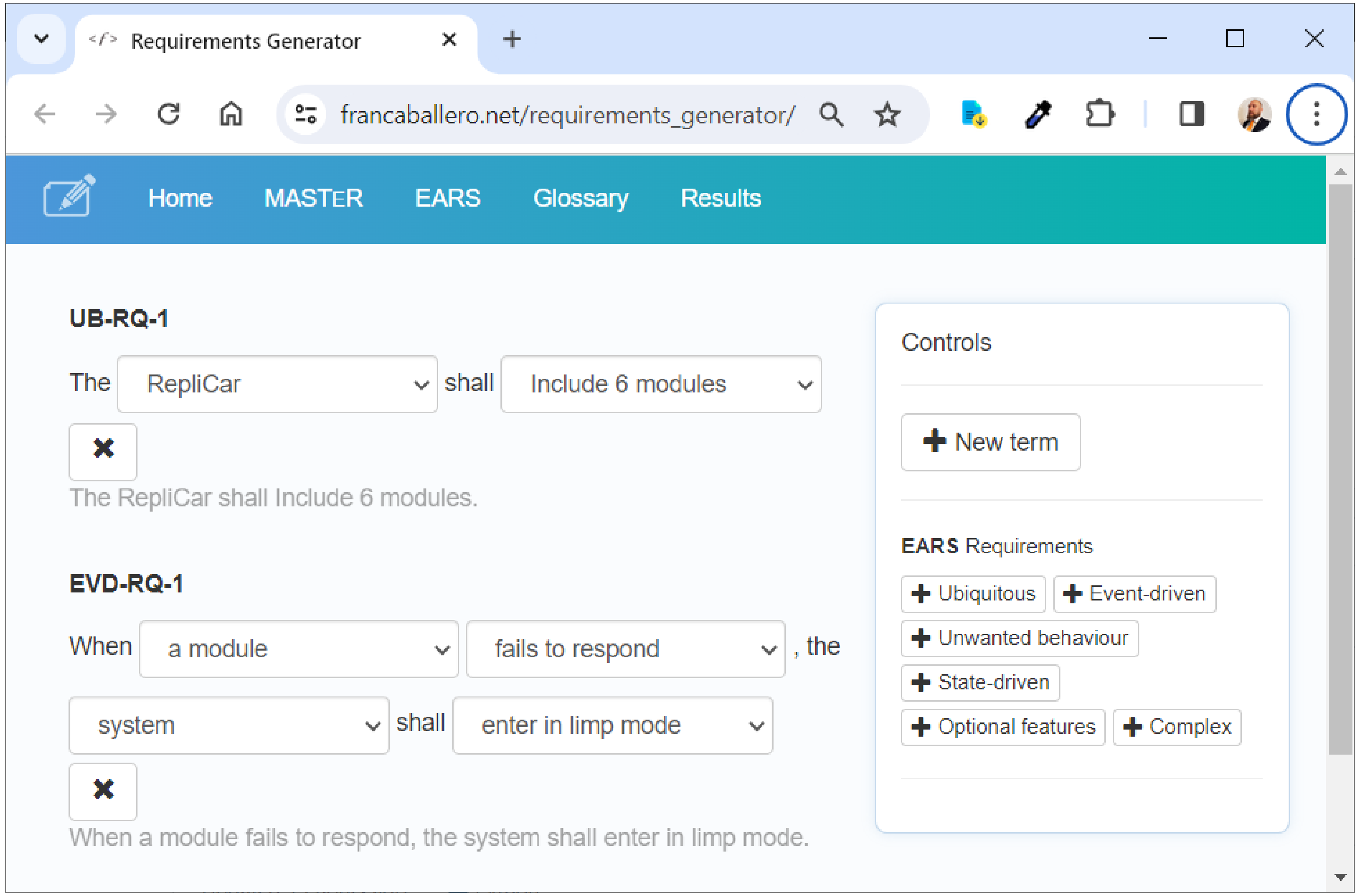
Writing Good Requirements – Verifiable

- Requirements may be unverifiable for several reasons. The following discusses the most common reason: use of **ambiguous terms** – because they are subjective (different meanings to different people).
- **Avoid these words**:
    - Minimize / Maximize
    - Rapid
    - User-friendly
    - Easy
    - Sufficient
    - Adequate
    - Quick

Writing Good Requirements – Tools

- **Requirements generator** https://francaballero.net/requirements_generator/

  Example: **Easy Approach to Requirements Syntax (EARS)**



References

nxp.com

References

- Ian Sommerville, *Software Engineering* (International Computer Science Series), 22 Aug. 2000
- *Writing Good Requirements*, https://spacese.spacegrant.org/uploads/Requirements-Writing/Writing%20Good%20Requirements.pdf
- IEEE Guide for Developing System Requirements Specifications , https://www2.seas.gwu.edu/~mlancast/cs254/IEE_STD_1233-_Requirements_Spec.pdf
- Requirements generator, https://francaballero.net/requirements_generator/

Overview of the change management process

Change/Feature/Bug Requests, Implementation (Coding), Review, Branch, Commit, PullRequest, Testing

Introduction in CHANGE MANAGEMENT PROCESS

The change management process refers to the stages involved in any change management strategy and its implementation. Having a strategy and steps helps transformations become successful and ensure that all factors are considered.

At a glance, the change management process breaks down into the following five steps:

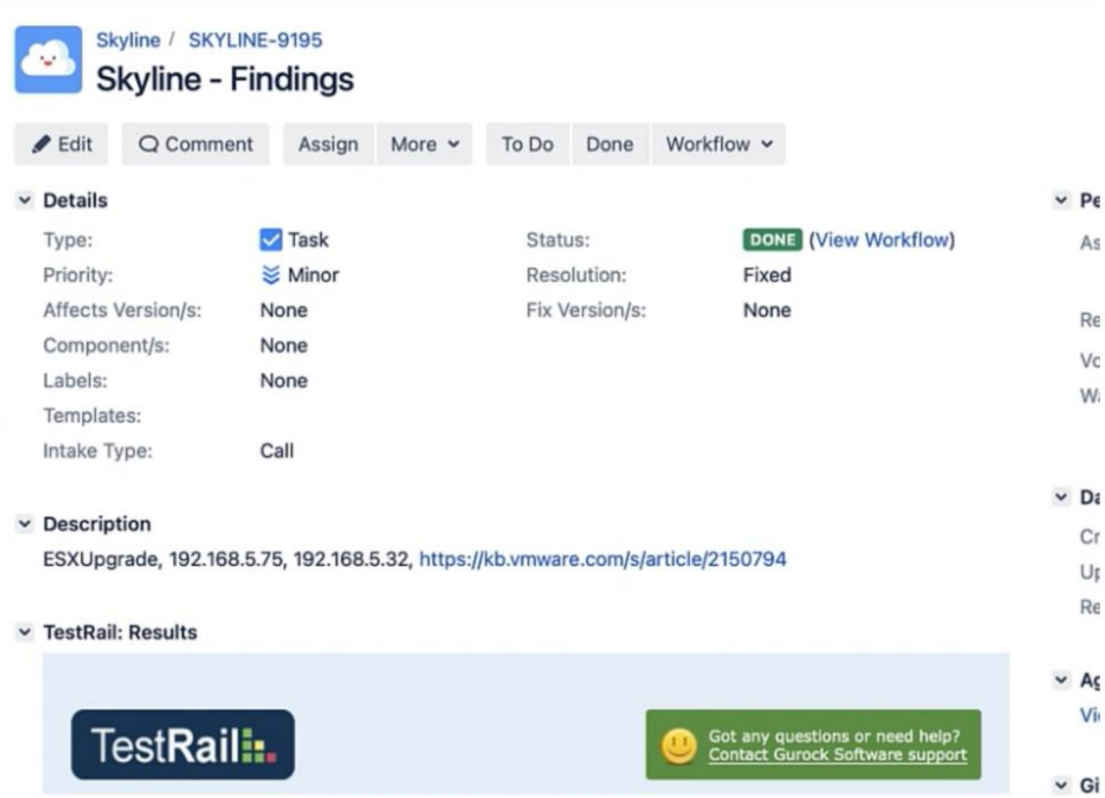
1. Prepare for change.
2. Create a vision for change.
3. Implement changes.
4. Embed and solidify changes.
5. Review and analyze.

Introduction in Jira

What Is Jira?

Jira is a software application developed by the Australian software company Atlassian that allows teams to track issues, manage projects, and automate workflows.

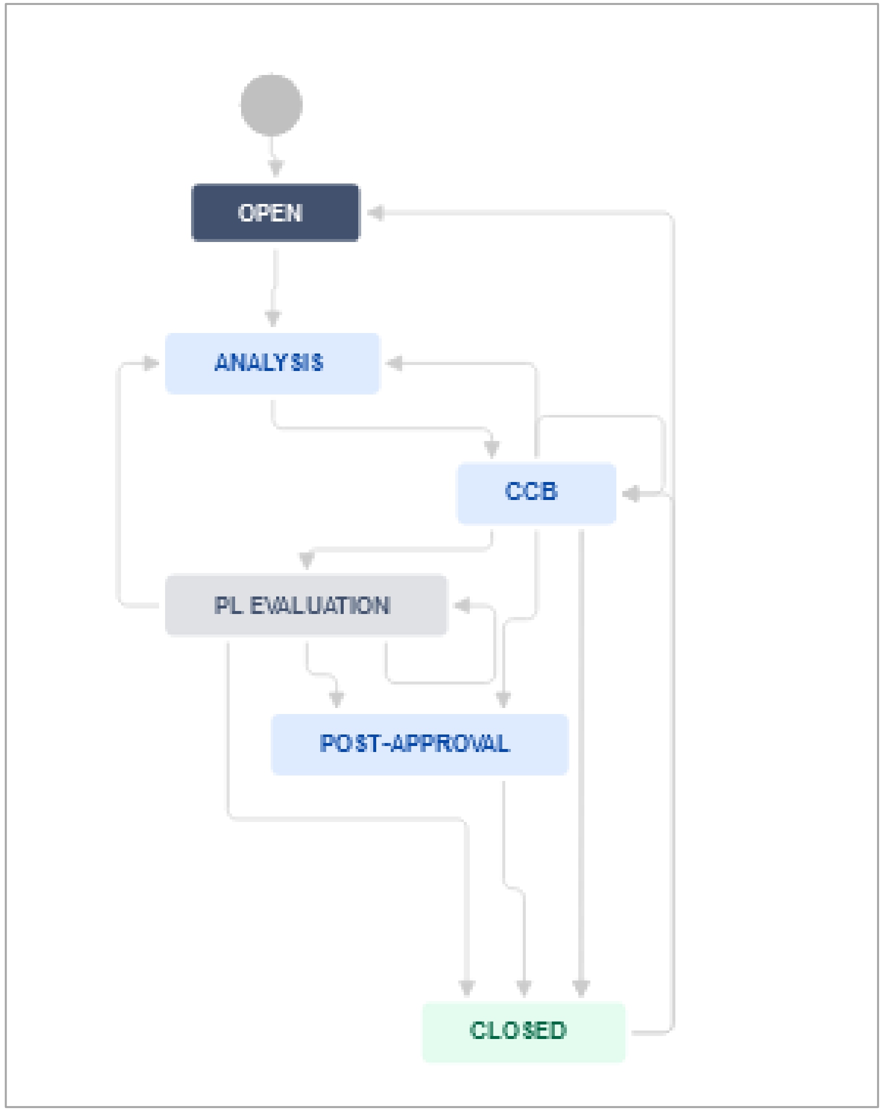
Jira is based on four key concepts: *issue*, *project*, *board*, and *workflow*.



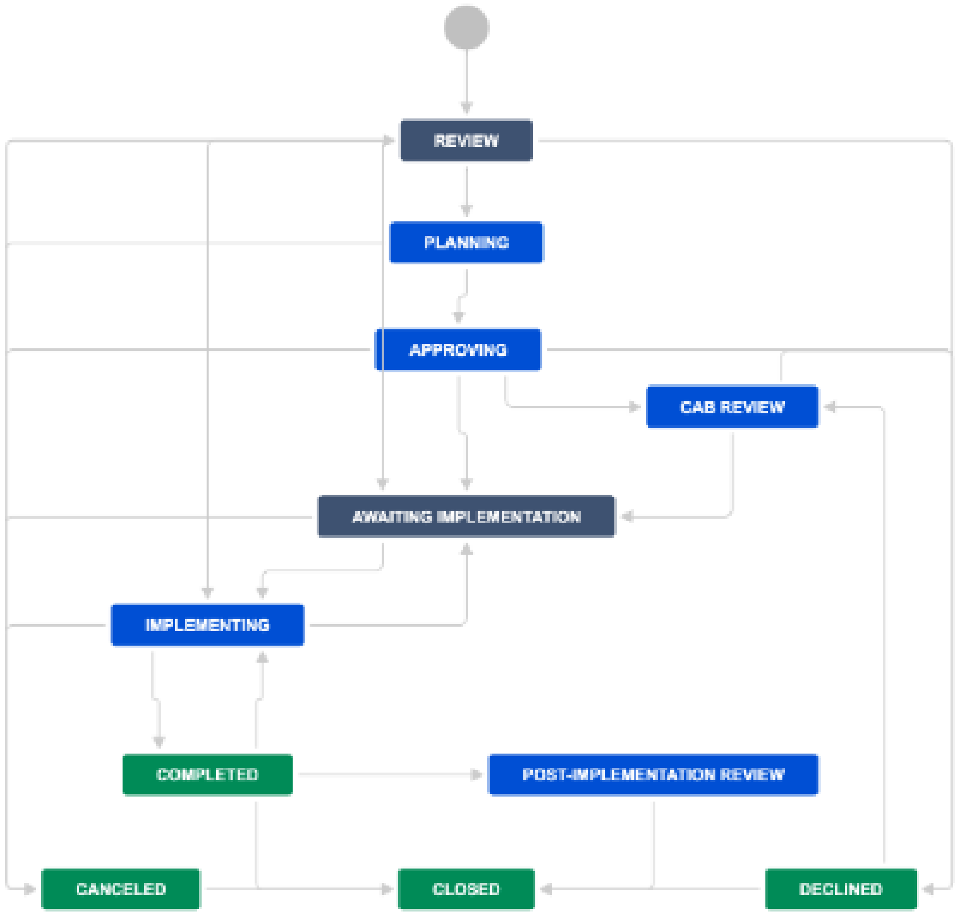
Simple schema

Jira Implementation Workflow

Change request flow



Jira Implementation Workflow



Support Tools

Jira, Git, Bitbucket

nxp.com

Log-in using Windows credentials

What can you do in Jira:

- ❖Plan and track several types of activities like tasks, sub-tasks, bugs, new- features, inquiries
- ❖Plan and track releases
- ❖Use queries (predefined or define your own)
- ❖Use the predefined Jira Reports
- ❖Visualize Jira project status
- ❖Integrate Jira with Git and PullRequest

JIRA

Story points

1, 1 , 2, 3 ,5 , 8, 13…..

JIRA

Story points

- 1 story point - 0.5 day
- 2 story points - 1 day
- 3 story points - 2 days
- 5 story points - 1 week
- 8 story points - 2 weeks
- 13 story points - 3 weeks (maximum)
- Tasks bigger than 13 points must be divided in smaller tasks

Repo from GitHub

Working with Git

Command Line Interface

https://git-scm.com/downloads

git-repo

repo - The multiple repository tool

Used in MCAL and OS projects

https://gerrit.googlesource.com/git-repo/

Source Tree GUI Client

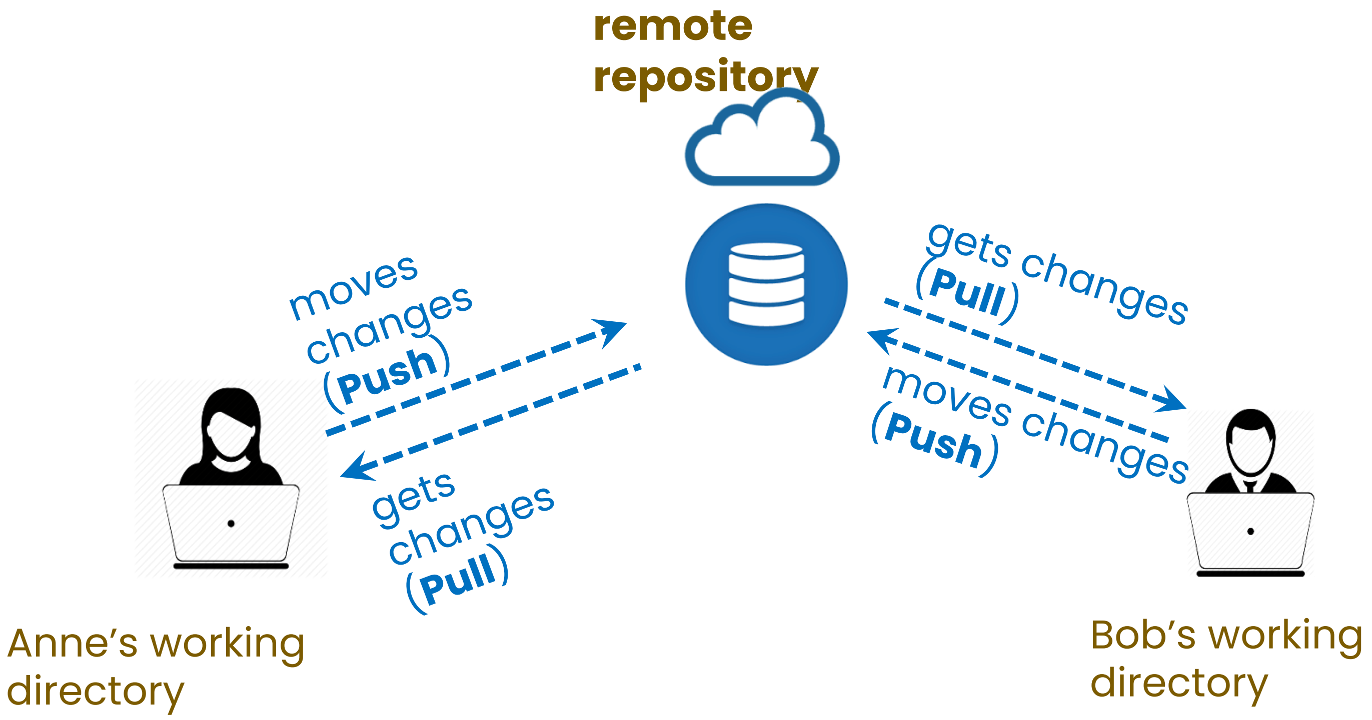
https://www.sourcetreeapp.com/download/

Atlassian Bitbucket

https://bitbucket.sw.nxp.com/dashboard

NXP remote repo servers

Centralized Version Control System



Working with Git BASH & Source Tree

Main steps using Git Bash and Source Tree.

1. **Clone the project repo. One time operation.**
2. **Create a branch from the JIRA ticket**
3. **Pull the branch**
4. **Checkout the branch**
5. **Add/update a source files**
6. **Commit the change**
7. **Pull the changes from remote repo**
8. **Push your change to remote repo, in the feature/ bugfix branch**
9. **Add a tag o the last commit form the feature/bugfix branch and push the tag**
10. **Merge the feature/bugfix branch to the official branch through BitBucket Pull Request**

Git setup on git bash

Syntax highlighting:

```
git config --global color.ui auto
```

Introduce yourself:

```
git config --global user.name “Name“
git config --global user.email email
```

Check your settings:

```
git config --list
```

Or check a specific key’s configuration

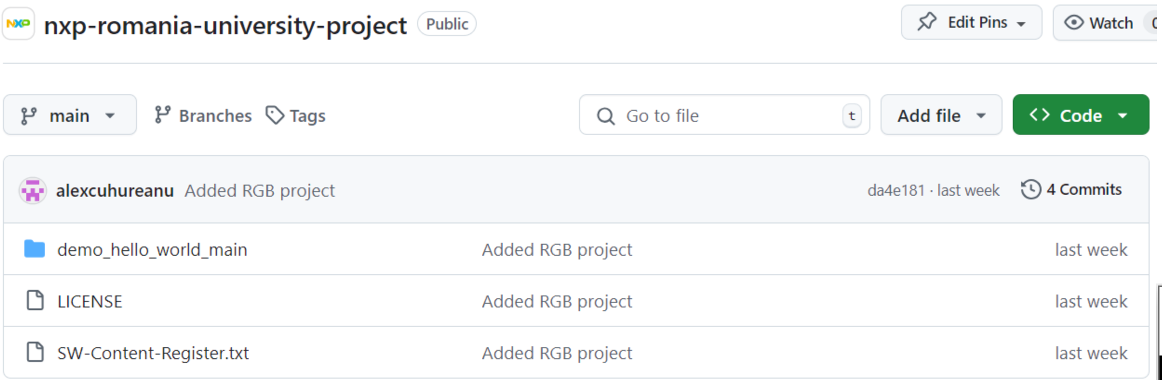
```
git config <key>
```

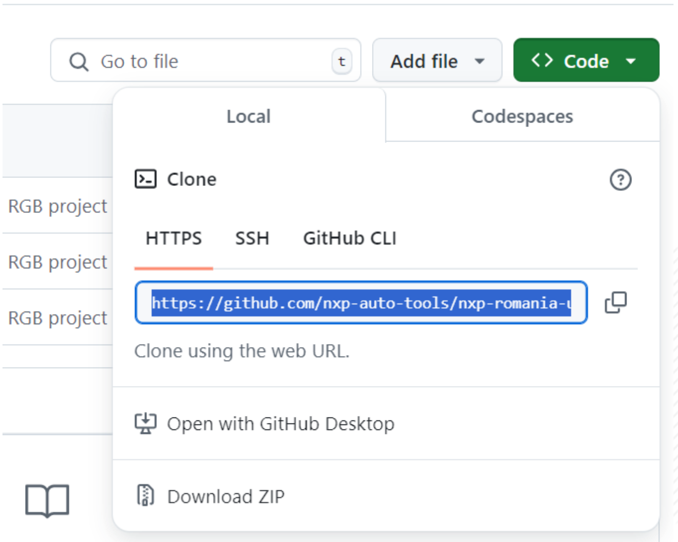
e.g. Make a clone:

```
git clone …..
```

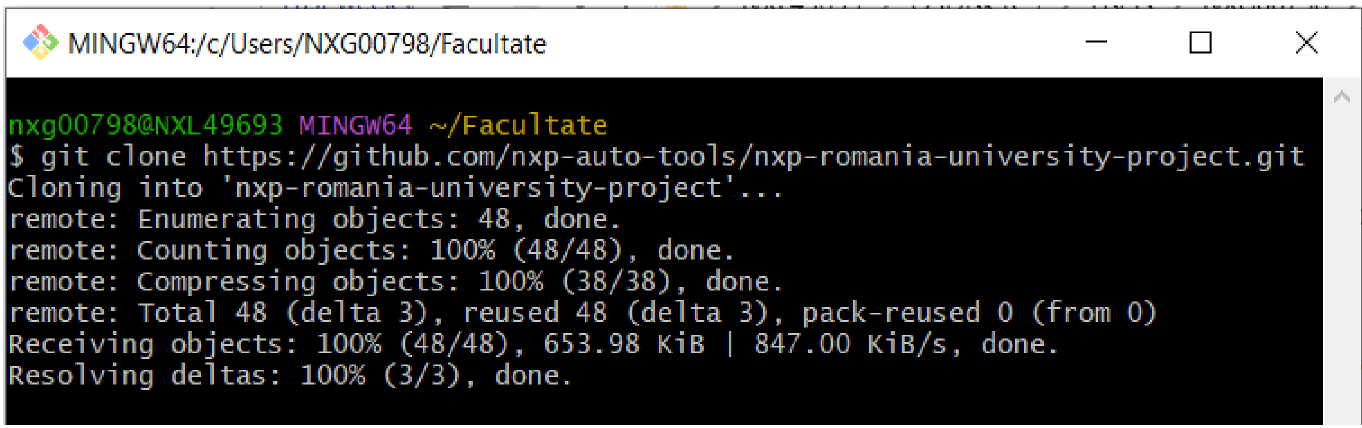
Git ducation

GitHub





Git Bash



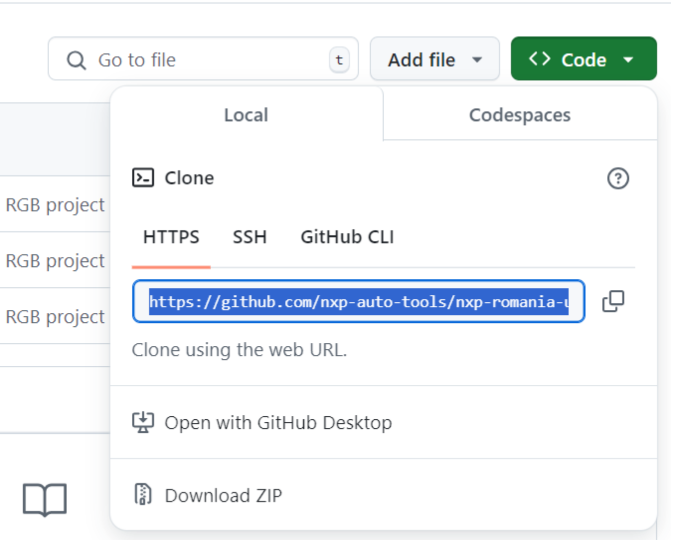
| Clone a repository

Cloning a repo

Someone else's repository can be located on their local file system or on a remote machine accessible via HTTP, HTTPS, SSH, etc.

To clone the repo onto your local machine:

```
git clone <url>
```

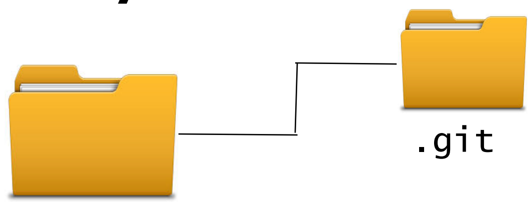


e.g. using SSH protocol:

```
git clone https://github.com/NXP/*repo*.git
```

SSH uses public-key authentication, so you'll have to generate one

Repository status



Now, you can notice a .git directory in the initialized project directory (is hidden).
Deleting it will turn your project into a normal unmanaged collection of files

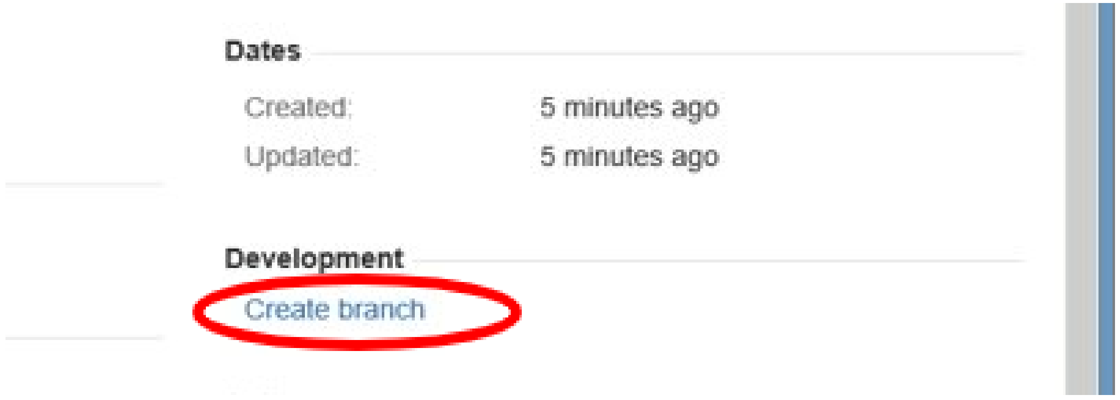
Next, you'll want to see the status of the repository:

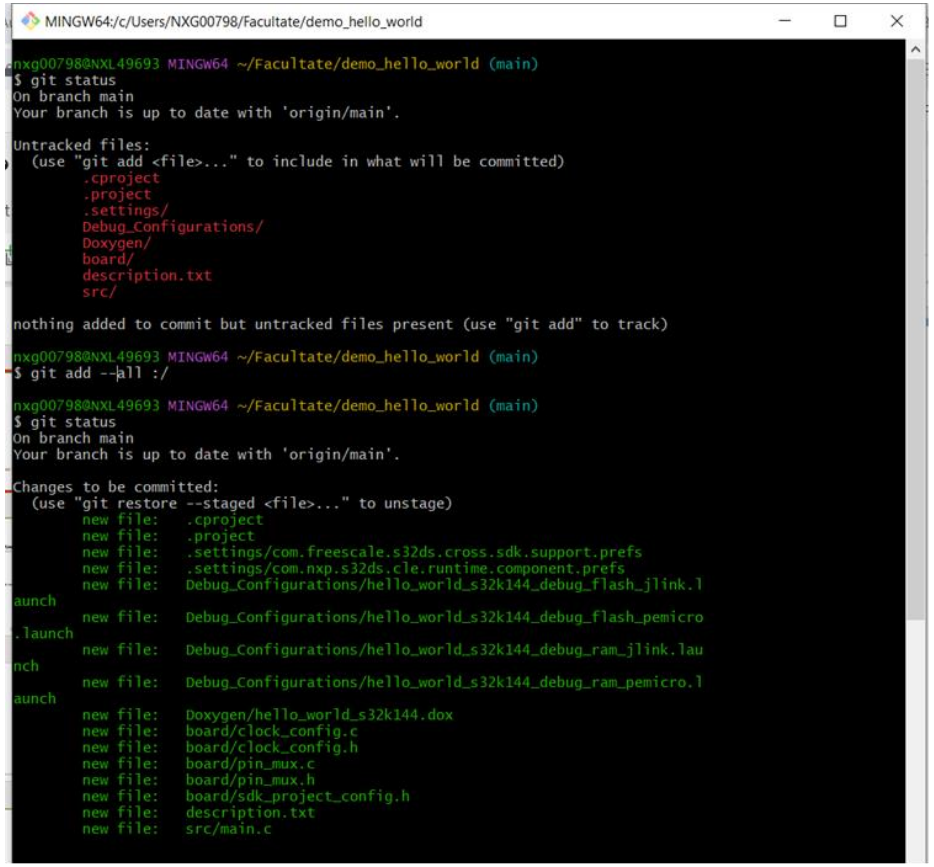
```
git status
```

Output:

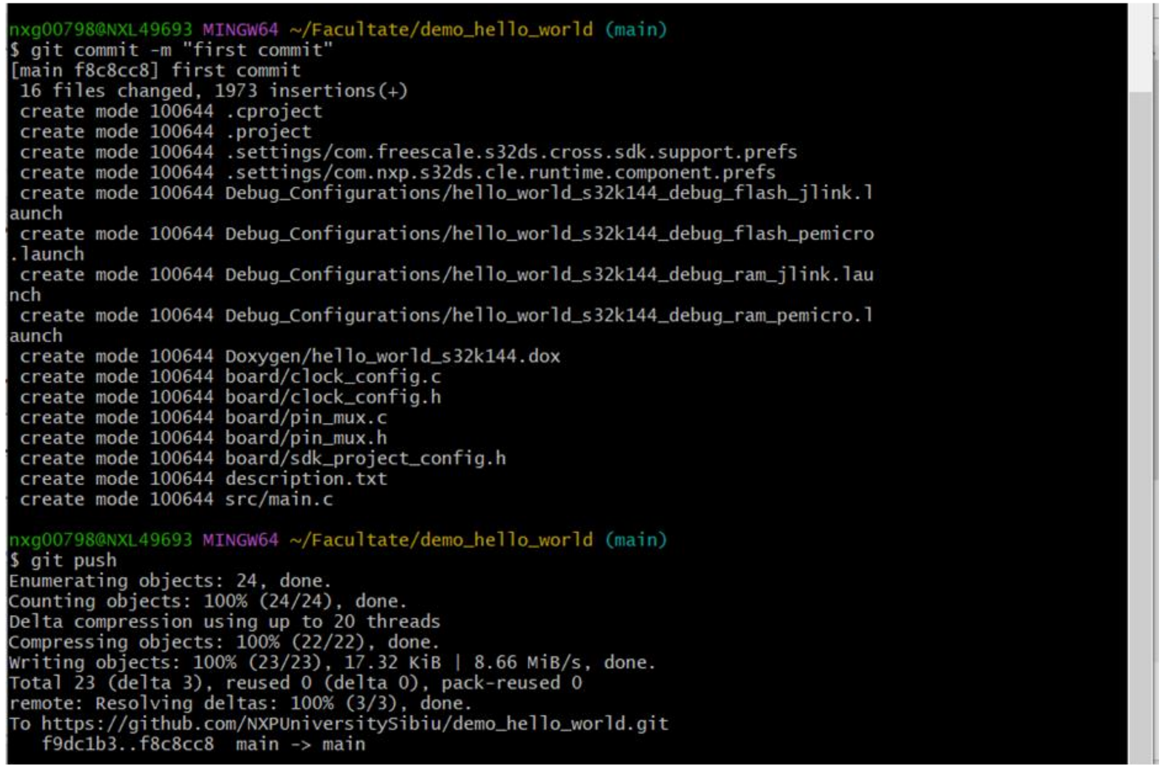
```
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to
track)
```

Create a branch from the JIRA ticket





```
MINGW64:/c/Users/NXG00798/Facultate/demo_hello_world
nxg00798@NXL49693 MINGW64 ~/Facultate/demo_hello_world (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .cproject
        .project
        .settings/
        Debug_Configurations/
        Doxygen/
        board/
        description.txt
        src/

nothing added to commit but untracked files present (use "git add" to track)

nxg00798@NXL49693 MINGW64 ~/Facultate/demo_hello_world (main)
$ git add --all :/

nxg00798@NXL49693 MINGW64 ~/Facultate/demo_hello_world (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   .cproject
        new file:   .project
        new file:   .settings/com.freescale.s32ds.cross.sdk.support.prefs
        new file:   .settings/com.nxp.s32ds.cle.runtime.component.prefs
        new file:   Debug_Configurations/hello_world_s32k144_debug_flash_jlink.l
aunch
        new file:   Debug_Configurations/hello_world_s32k144_debug_flash_pemicro
.launch
        new file:   Debug_Configurations/hello_world_s32k144_debug_ram_jlink.lau
nch
        new file:   Debug_Configurations/hello_world_s32k144_debug_ram_pemicro.l
aunch
        new file:   Doxygen/hello_world_s32k144.dox
        new file:   board/clock_config.c
        new file:   board/clock_config.h
        new file:   board/pin_mux.c
        new file:   board/pin_mux.h
        new file:   board/sdk_project_config.h
        new file:   description.txt
        new file:   src/main.c
```



```
nxg00798@NXL49693 MINGW64 ~/Facultate/demo_hello_world (main)
$ git commit -m "first commit"
[main f8c8cc8] first commit
 16 files changed, 1973 insertions(+)
 create mode 100644 .cproject
 create mode 100644 .project
 create mode 100644 .settings/com.freescale.s32ds.cross.sdk.support.prefs
 create mode 100644 .settings/com.nxp.s32ds.cle.runtime.component.prefs
 create mode 100644 Debug_Configurations/hello_world_s32k144_debug_flash_jlink.l
aunch
 create mode 100644 Debug_Configurations/hello_world_s32k144_debug_flash_pemicro
.launch
 create mode 100644 Debug_Configurations/hello_world_s32k144_debug_ram_jlink.lau
nch
 create mode 100644 Debug_Configurations/hello_world_s32k144_debug_ram_pemicro.l
aunch
 create mode 100644 Doxygen/hello_world_s32k144.dox
 create mode 100644 board/clock_config.c
 create mode 100644 board/clock_config.h
 create mode 100644 board/pin_mux.c
 create mode 100644 board/pin_mux.h
 create mode 100644 board/sdk_project_config.h
 create mode 100644 description.txt
 create mode 100644 src/main.c

nxg00798@NXL49693 MINGW64 ~/Facultate/demo_hello_world (main)
$ git push
Enumerating objects: 24, done.
Counting objects: 100% (24/24), done.
Delta compression using up to 20 threads
Compressing objects: 100% (22/22), done.
Writing objects: 100% (23/23), 17.32 KiB | 8.66 MiB/s, done.
Total 23 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/NXPUniversitySibiu/demo_hello_world.git
   f9dc1b3..f8c8cc8  main -> main
```

| Commit your changes

Committing files

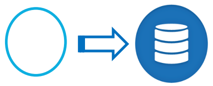
Committing the snapshot

```
git commit -m "Added the module definitions and functionalities"
```

-m flag prevents the additional editor window message to pop up.

Your message should, however, be explicit enough to others.
To exit and save contents, hit Esc and then, enter :wq.

Note that git status only shows the uncommitted changes.
If you run it after a commit, it will show that you have no more files to commit.
(only if you don’t add new files in your working directory in the meantime)

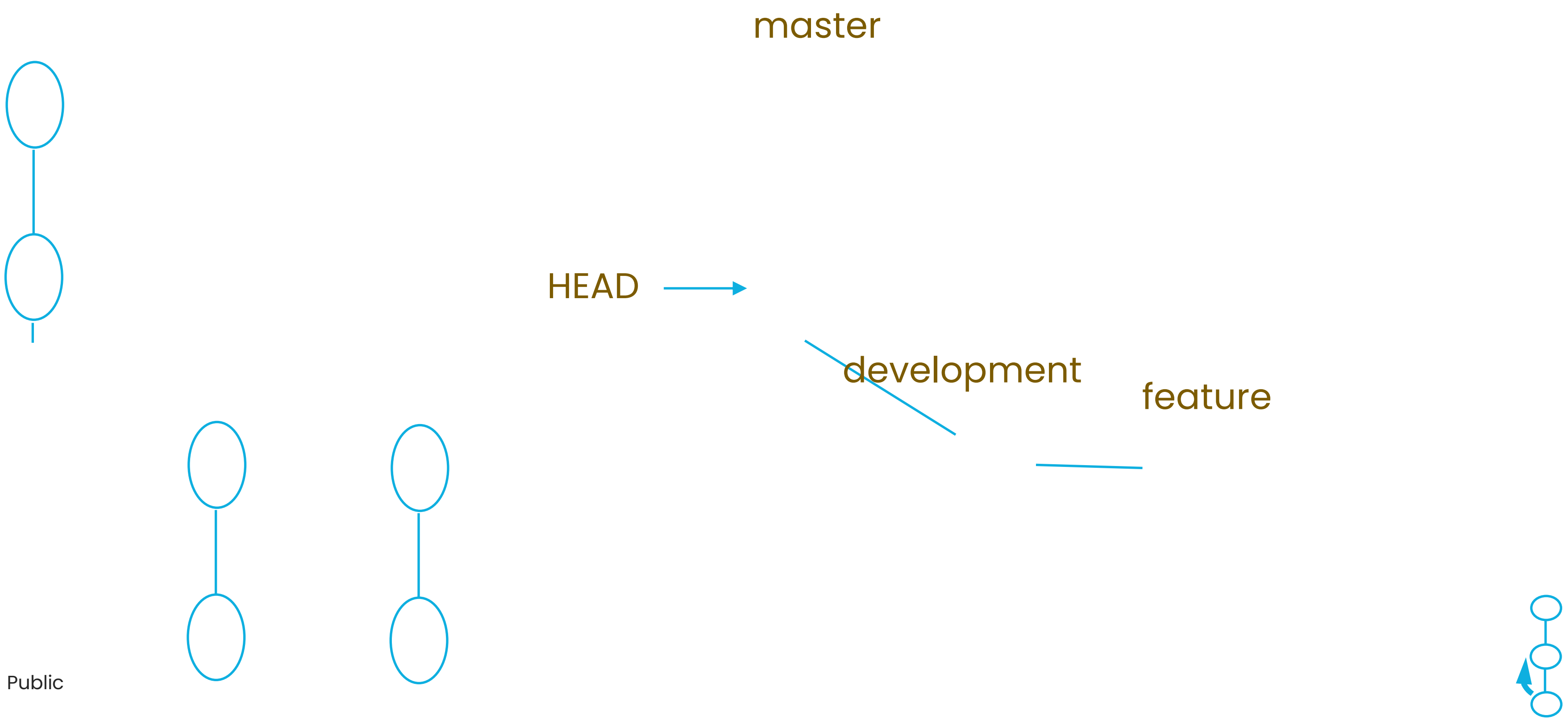


Checkout <branch>

```
git checkout <branch-name>
```

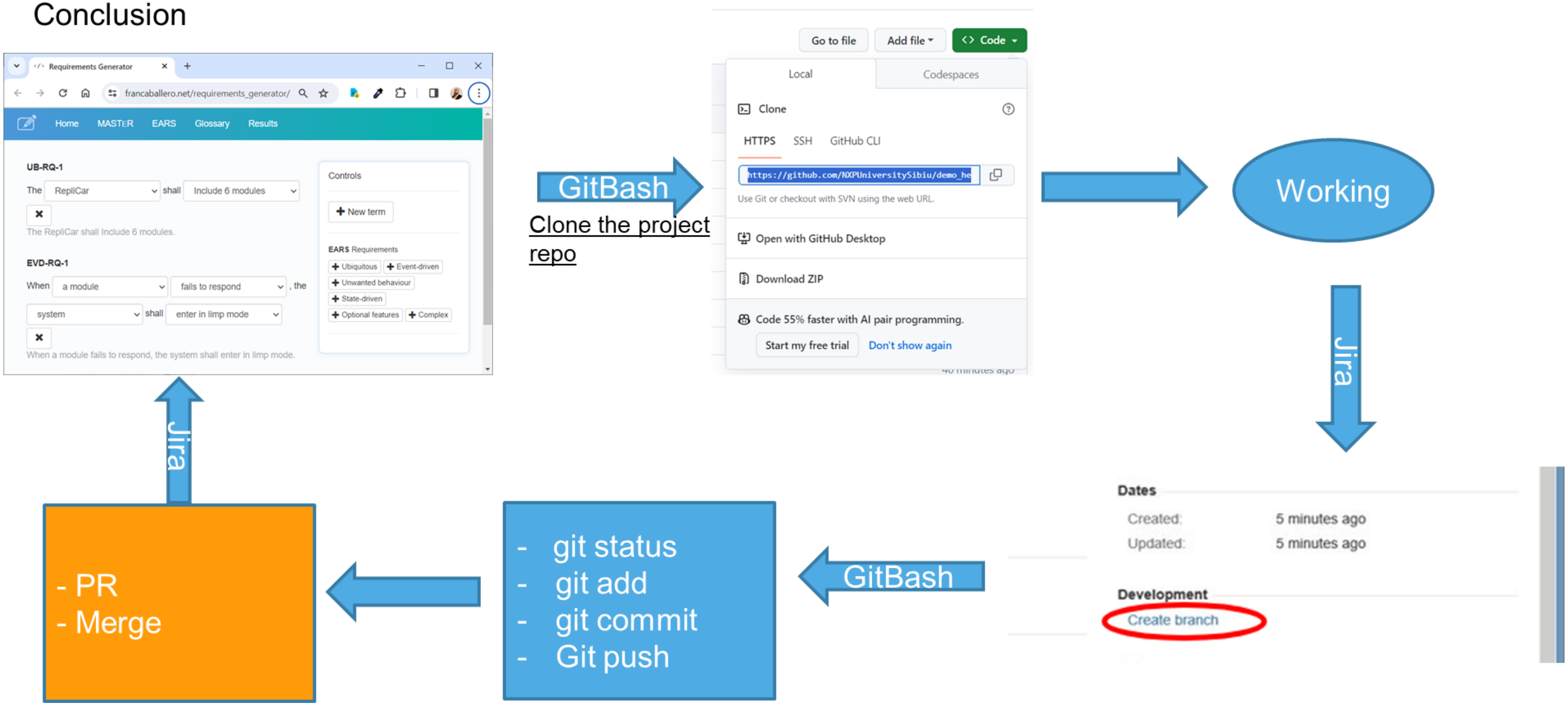
To return to the current (default) branch – master (restores current files in index and working directory):

```
git checkout master
```




blog.amigoscode.com
12 MOST COMMON GIT COMMANDS
git init
START
Creates a new local repository in the current directory
git clone
Copies an existing remote repository to your local machine.
git status
Shows the state of your working directory and staging area.
git add
Adds changes in your working directory to the staging area, which is a temporary area where you can prepare your next commit.
git commit
Records the changes in the staging area as a new snapshot in the local repository, along with a message describing the changes.
git push
Uploads the local changes to the remote repository usually a on a platform like GitHub or GitLab.
git pull
Downloads the latest commits from a remote repository and merges them with your local branch.
git branch
Lists, creates, renames, or deletes branches in your local repository. A branch is a pointer to a specific commit.
git checkout
Switches your working directory to a different branch or commit, discarding any uncommitted changes
git merge
Combines the changes from one branch into another branch, creating a new commit if there are no conflicts
git diff
Shows the differences between two commits, branches, files, or the working directory and the staging area.
git log
LOG
Shows the history of commits in the current branch, along with their messages, authors, and dates.
amigoscode.com

Conclusion


GitBash
Clone the project repo
Working
Jira
Dates
Created: 5 minutes ago
Updated: 5 minutes ago
Development
Create branch
GitBash
- git status
- git add
- git commit
- Git push
- PR
- Merge
Jira
Requirements Generator
francaballero.net/requirements_generator/
Home MASTER EARS Glossary Results
UB-RQ-1
The RepliCar shall Include 6 modules
The RepliCar shall Include 6 modules.
EVD-RQ-1
When a module fails to respond, the system shall enter in limp mode
When a module fails to respond, the system shall enter in limp mode.
Controls
New term
EARS Requirements
Ubiquitous
Event-driven
Unwanted behaviour
State-driven
Optional features
Complex
Go to file
Add file
Code
Local
Codespaces
Clone
HTTPS SSH GitHub CLI
https://github.com/NXPUniversitySibiu/demo_he
Use Git or checkout with SVN using the web URL.
Open with GitHub Desktop
Download ZIP
Code 55% faster with AI pair programming.
Start my free trial
Don't show again

ENVIRONMENT SET-UP

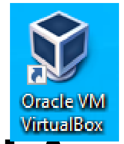
nxp.com

Install VirtualBox

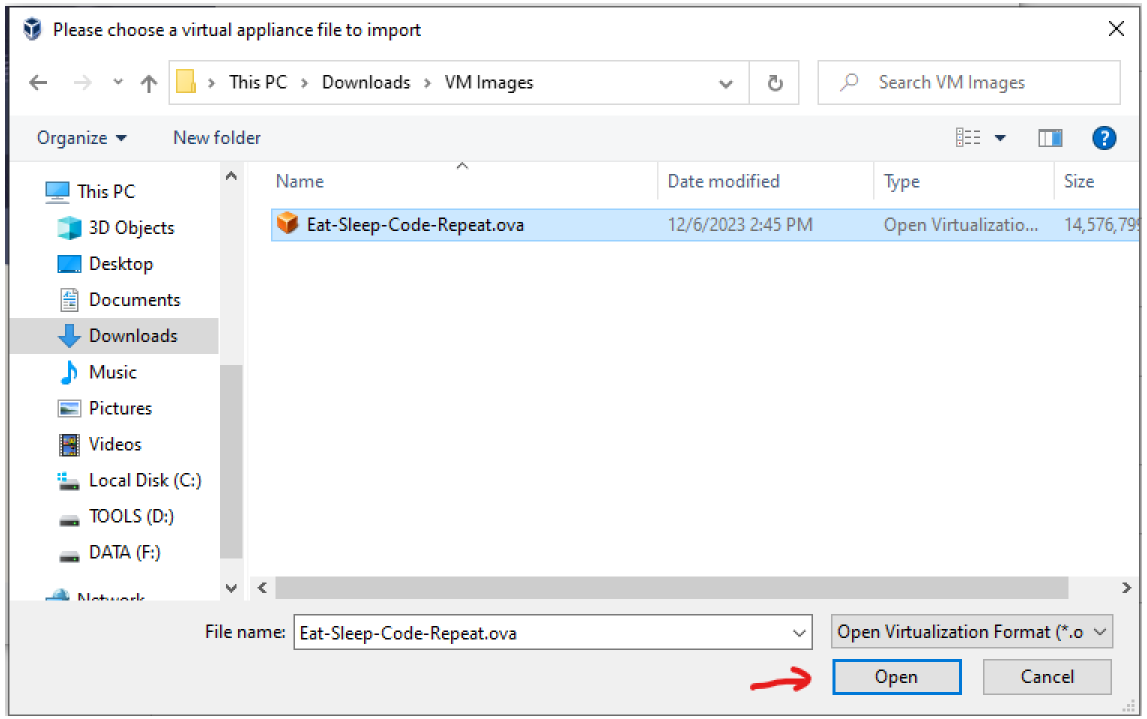
- Prerequisites
  - 40 GB free space on disk
  - Enable Virtualization. If is disabled, please search on google the procedure specific for your own laptop (e.g. search on google *How to Enable Virtualization on HP Latitude xxx*) - Usually this is activated by default on most models.
- Install VirtualBox - v7 is required
- Download VM image - **Eat-Sleep-Code-Repeat.ova**

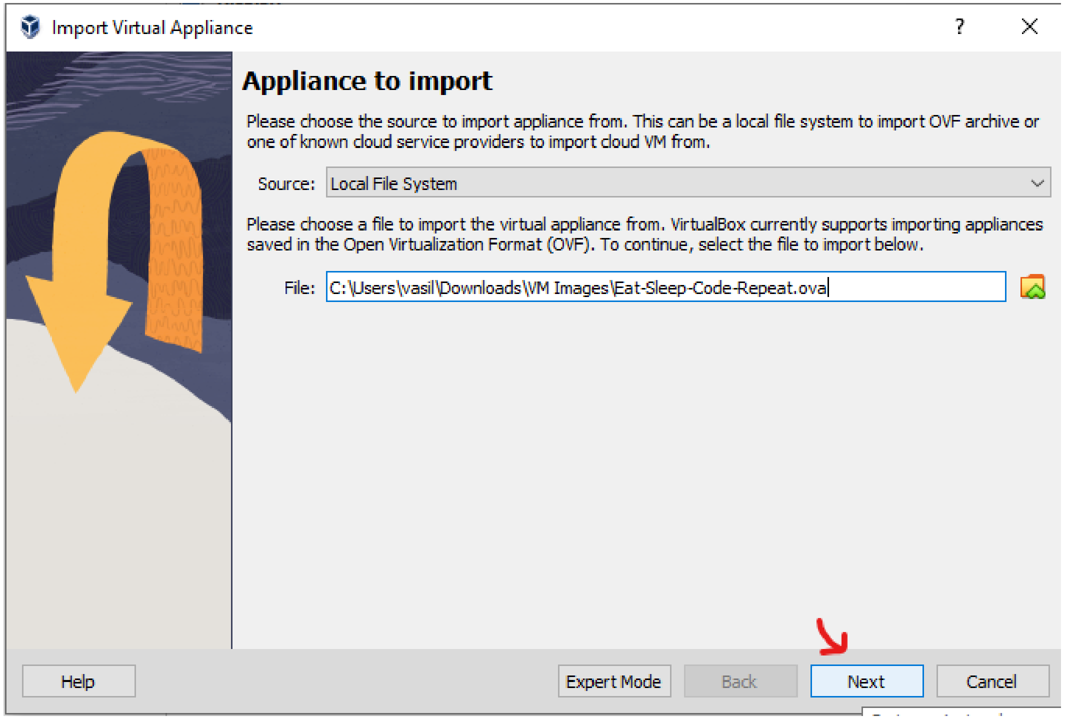
Eat-Sleep-Code-Repeat.ova - Google Drive

Import Image into VirtualBox

- Start VirtualBox 

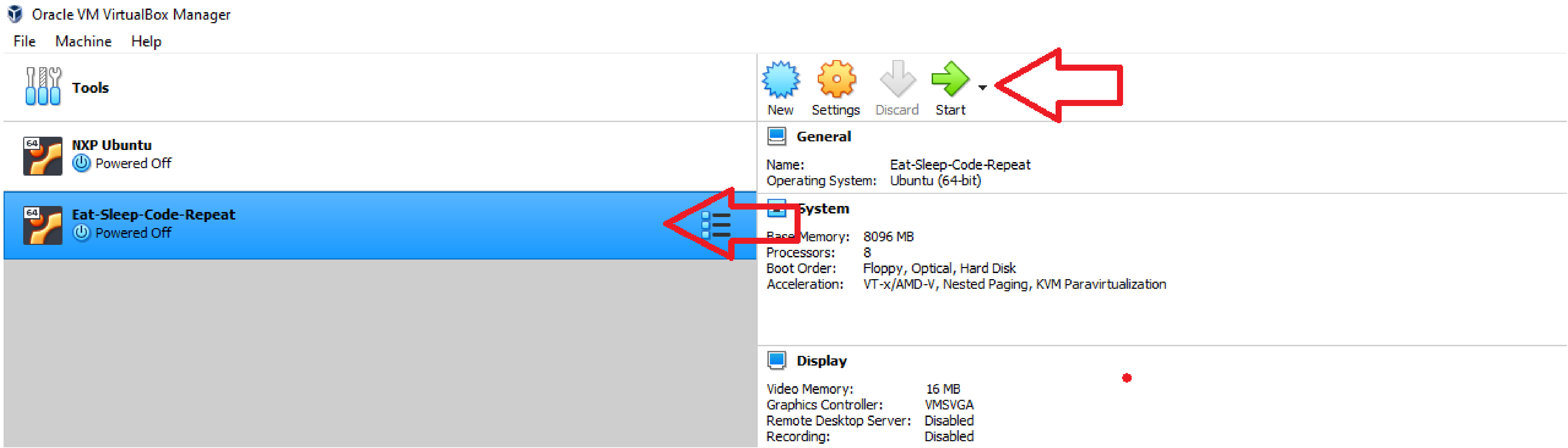
- Go to **File->Import Appliance…** menu (Ctrl+I)
- Browse and select the downloaded **Eat-Sleep-Code-Repeat.ova**



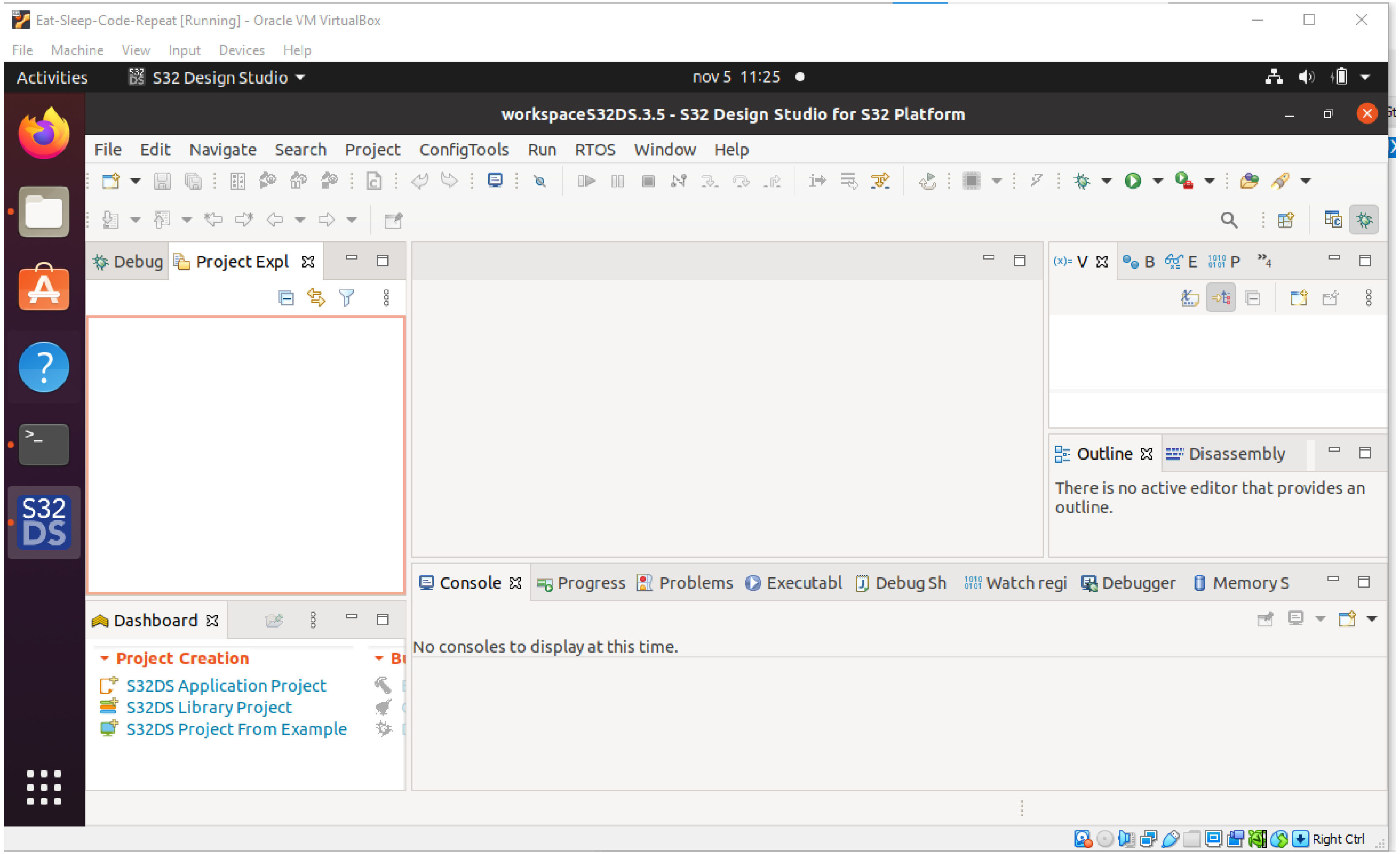


Import Image into VirtualBox

- Select Eat-Sleep-Code-Repeat and then press **Start** button
- Default username: **student**
- Default password: **student**

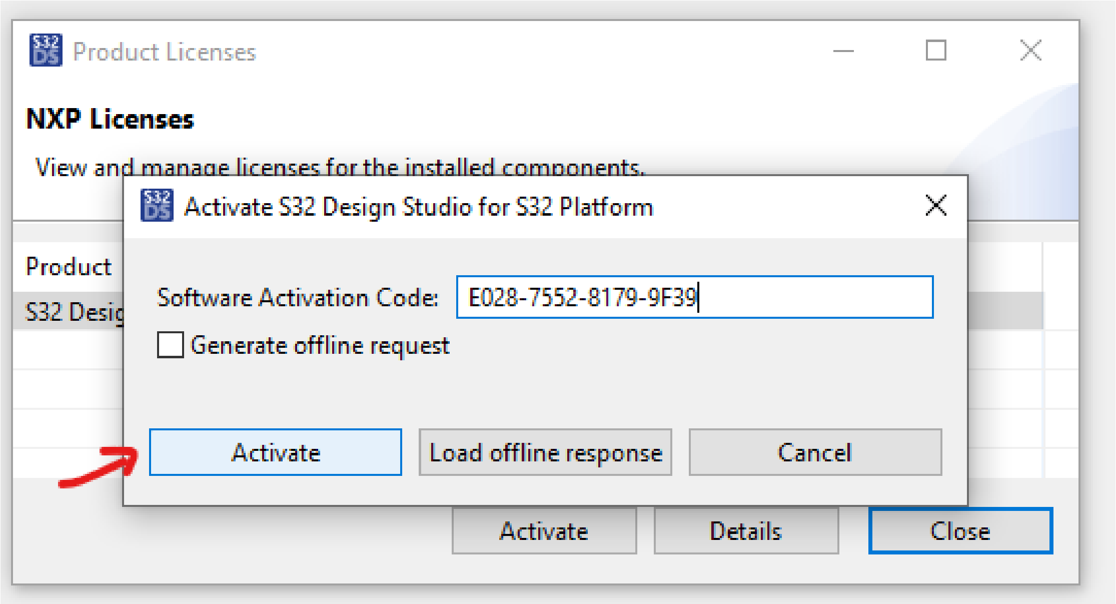


Import Image into VirtualBox



Activate S32DS

- Start **S32 Design Studio for S32 Platform 3.5**
- Open terminal in ***/home/student/NXP/S32DS.3.5***
- *sudo ./s32ds.sh*
- [Design : Product Download : Files]
- Use this activation code **E028-7552-8179-9F39, 084A-7BE8-C06C-A982**
- Use online activation – Internet connection is required
- DISCLAIMER! - Don't share the activation code!



S32K144 board ON VM

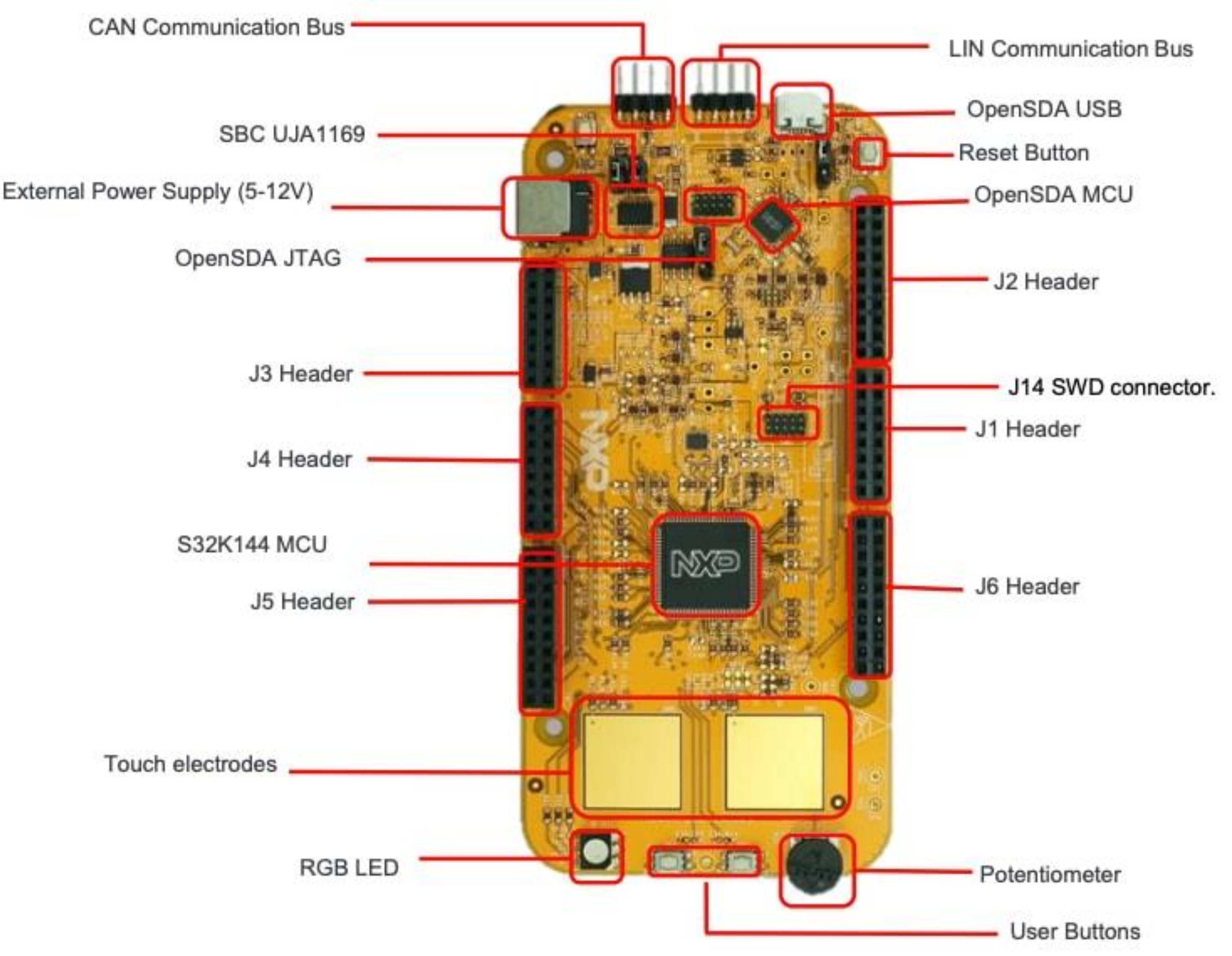
- **Set-up debug configuration & exercise**

nxp.com

02

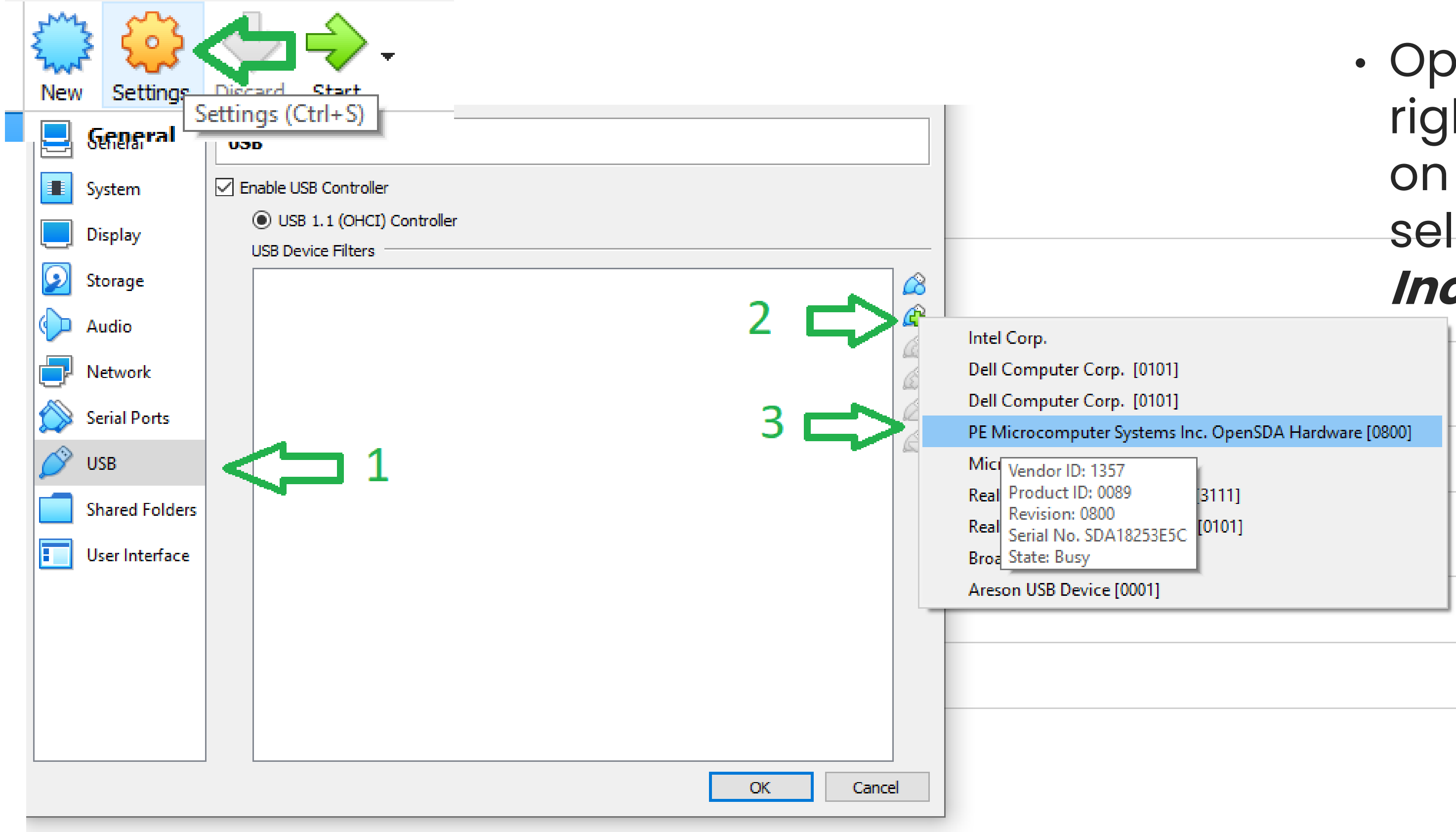
S32K144

Get to Know Your Evaluation Board

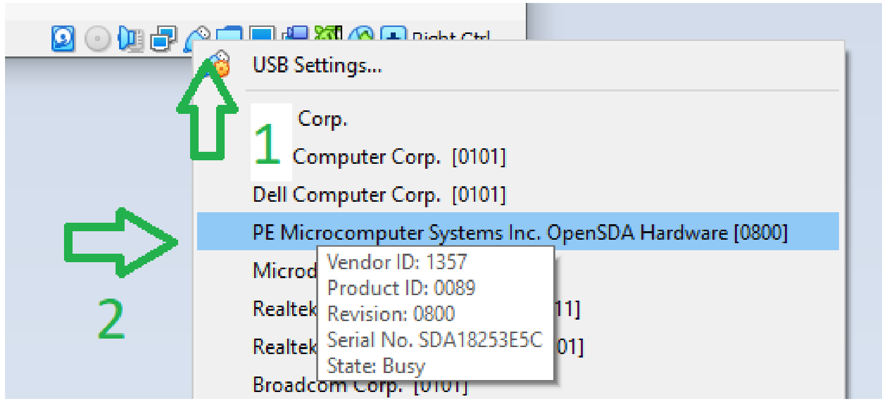


Attach Board to VM

- Option 1: With VM turned OFF, select the VM and click on **Settings** button or press **Ctrl+S** keys
- Select **USB** node then click on **Add new USB filter…** And from the list select ***PE Microcomputer Systems Inc…..*** device

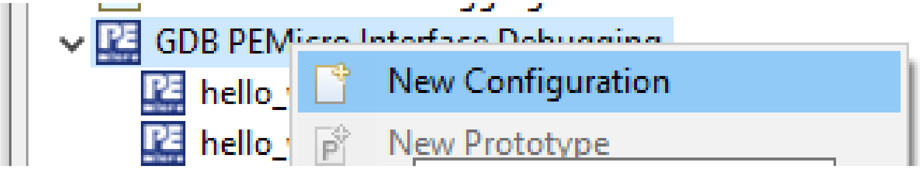


- Option 2: With VM running, from the right-bottom of the VM screen, right click on USB icon and select ***PE Microcomputer Systems Inc…..*** device from the list



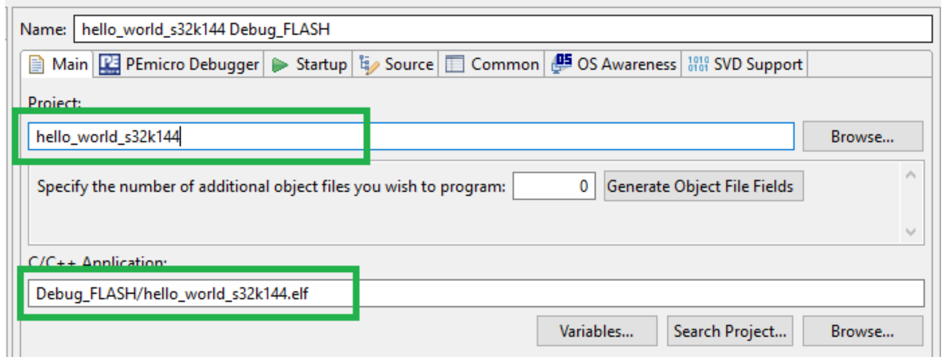
Create DEBUG Configuration

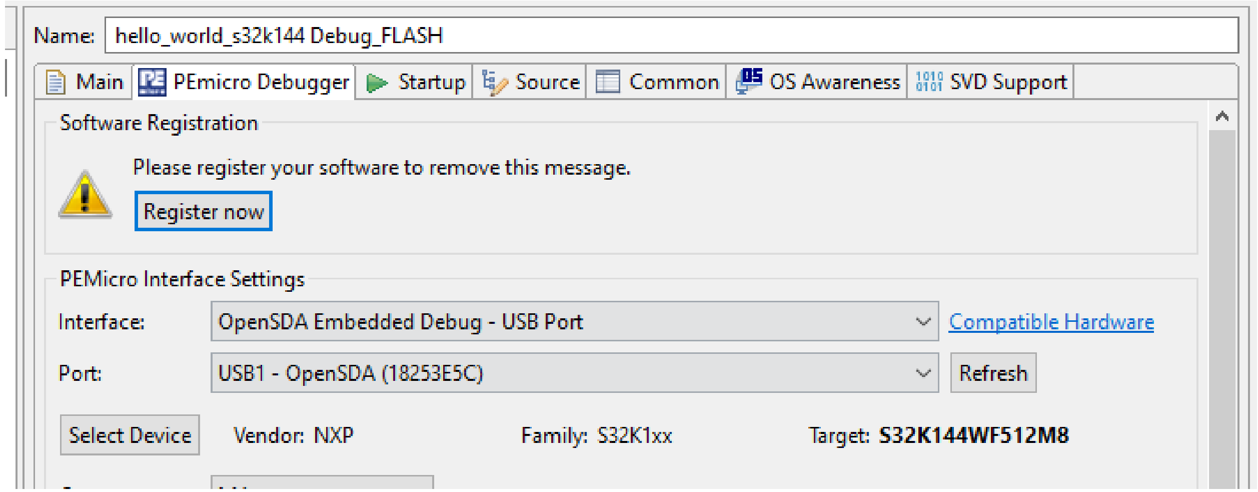
- Start ***S32 Design Studio for S32 Platform 3.5***
- Select **hello world** project
- Build project – click **Project > Build All (Ctrl+B)**
- Click **Run > Run** or **Run > Debug**.
- Expand ***GDB PEMicro Interface Debugging***
- If there is no entry like *{project_name}_Debug_FLASH* create a new configuration -> right click on ***GDB PEMicro Interface Debugging -> New Configuration***



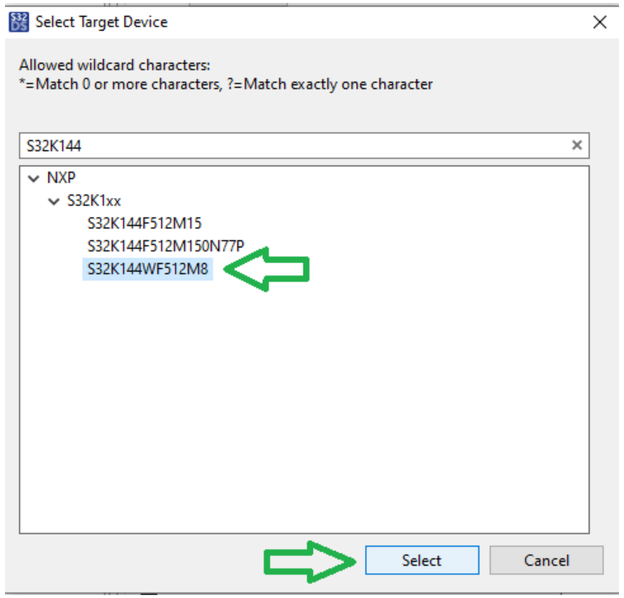
Create DEBUG Configuration

- In the **Main** tab select ***Project*** and *C/C++ Application*
- In the **Pemicro Debugger** tab set **Interface, Port**
- Click on ***Select Device*** and select ***S32K144WF512M8*** device

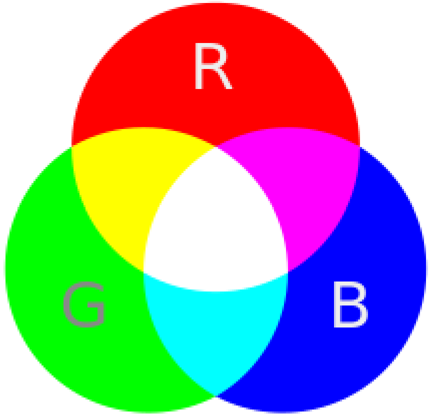




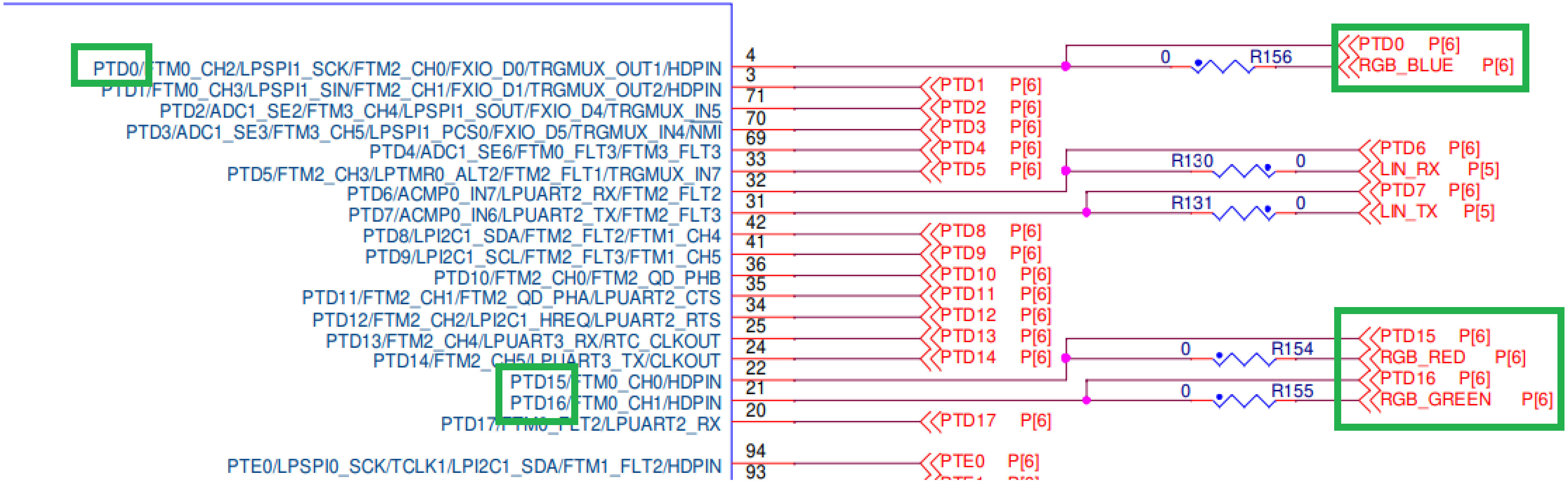
- Click on **Debug** button – the project will be flashed on board and the application will start to run on this.
- From this point, the Debug perspective will be opened and the execution will stop at the first line of **main** function – click **F8** to resume the code execution



Exercise: RED-YELLOW-BLUE



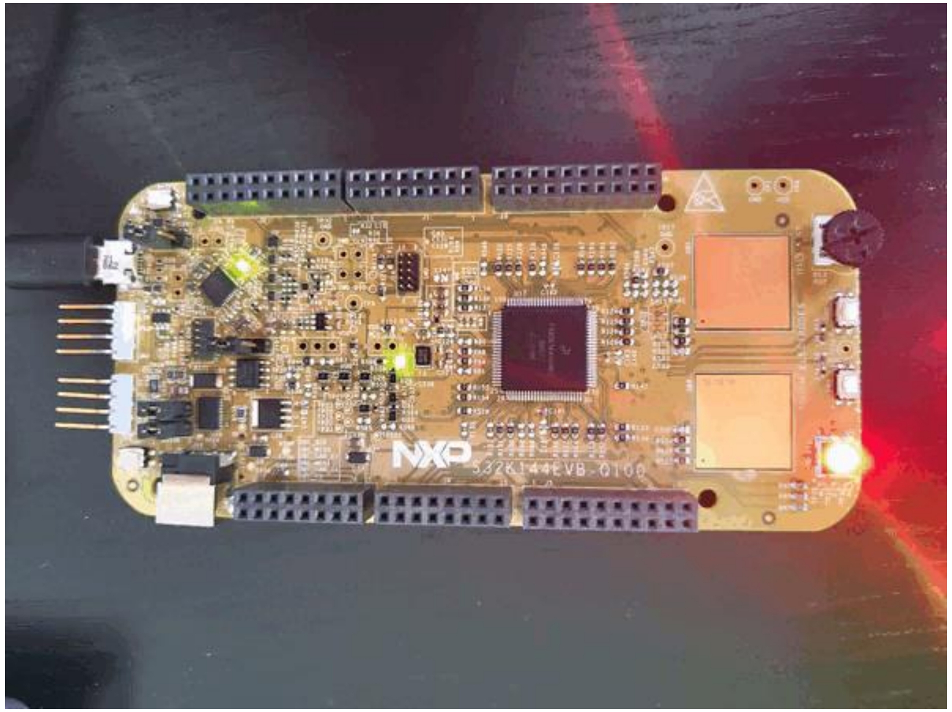
- The initial demo project will make the RGB LED to blink rapidly RED and GREEN colors
- Scope of this exercise is to make the RGB LED to blink on RED, YELLOW and BLUE

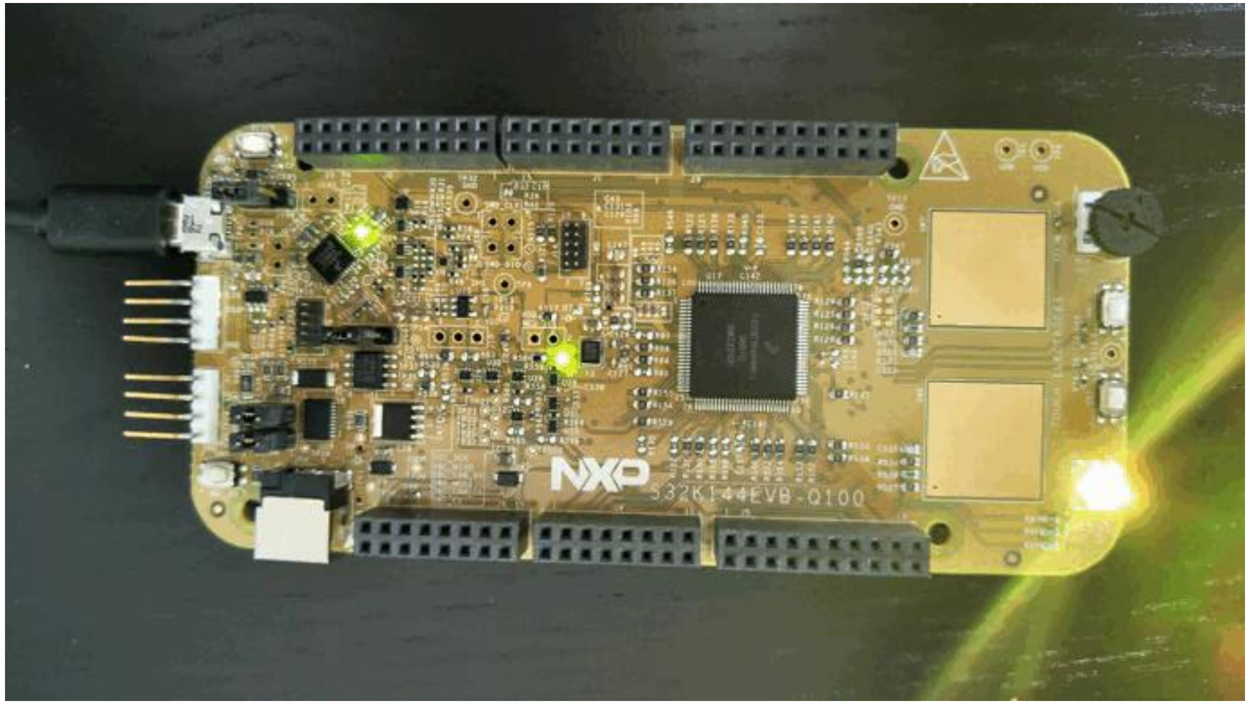


Board pinout

- RGB_BLUE – PTD0 ; RGB_RED – PTD15 ; RGB_GREEN – PTD16

Exercise: RED-YELLOW-BLUE





Before

Target result

How Hardware and Software are Linked

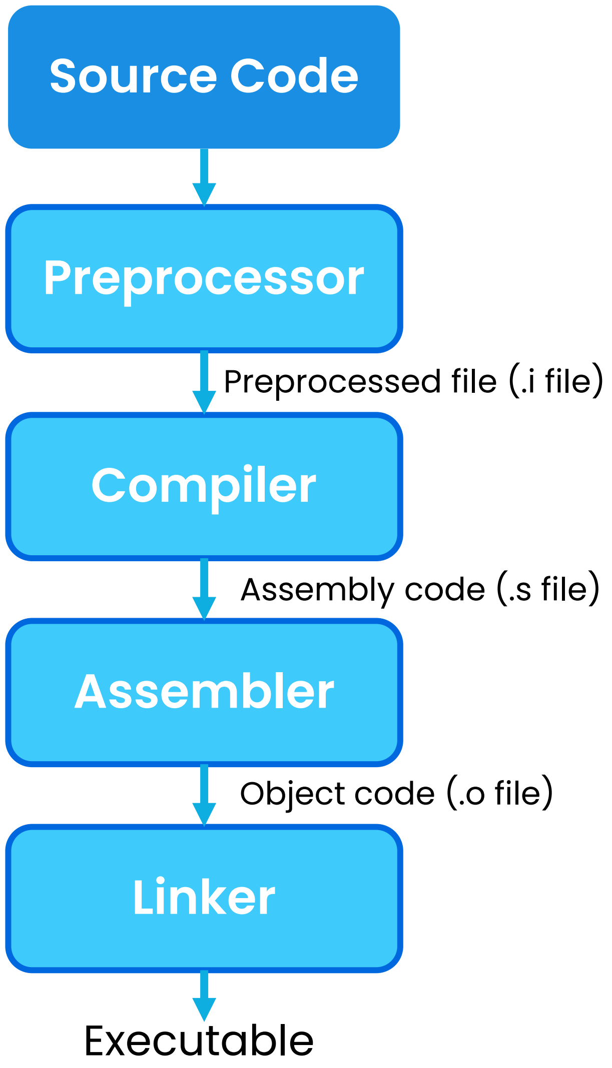
nxp.com

What is Hardware and Software?

Hardware	Software
Hardware refers to the physical, tangible components of a computing system or electronic device. It includes all mechanical, electronic, and electric part that form the foundation of the system's functionality. Essentially, hardware is everything you can touch and see in a computer or device.	Software refers to the set of instructions, programs and data that run on hardware, guiding it to perform specific tasks. Unlike hardware, software is intangible; it cannot be physically touched but it plays a crucial role in controlling and managing the hardware.

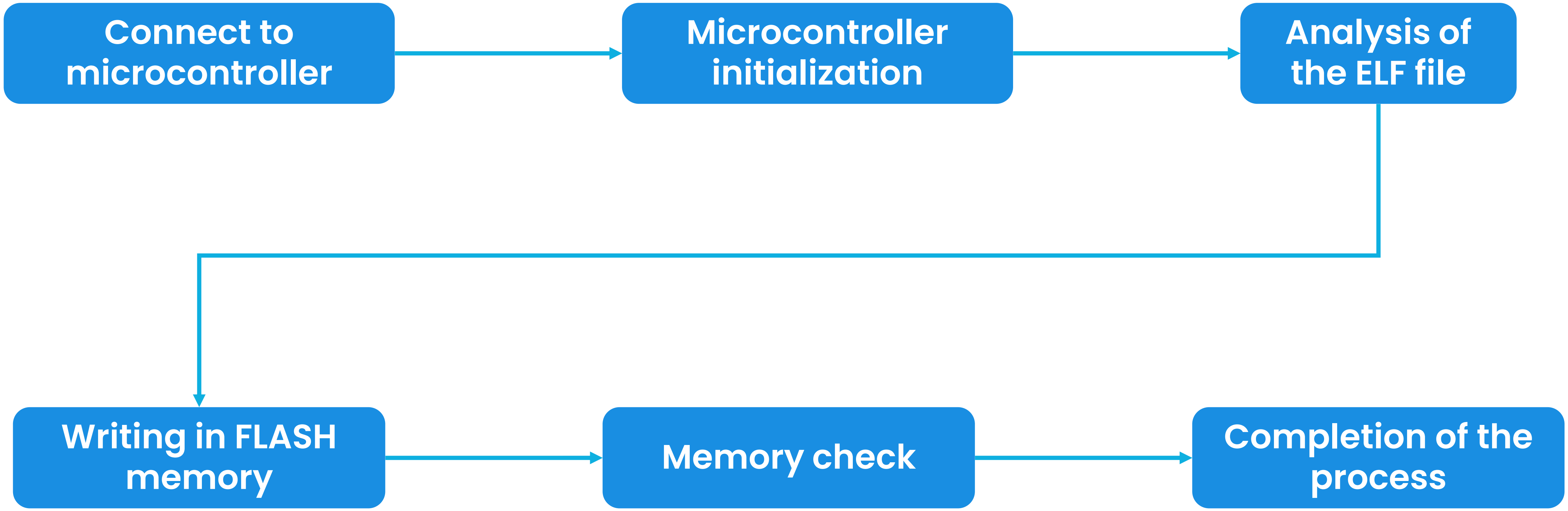
From source code to executable

Compiling



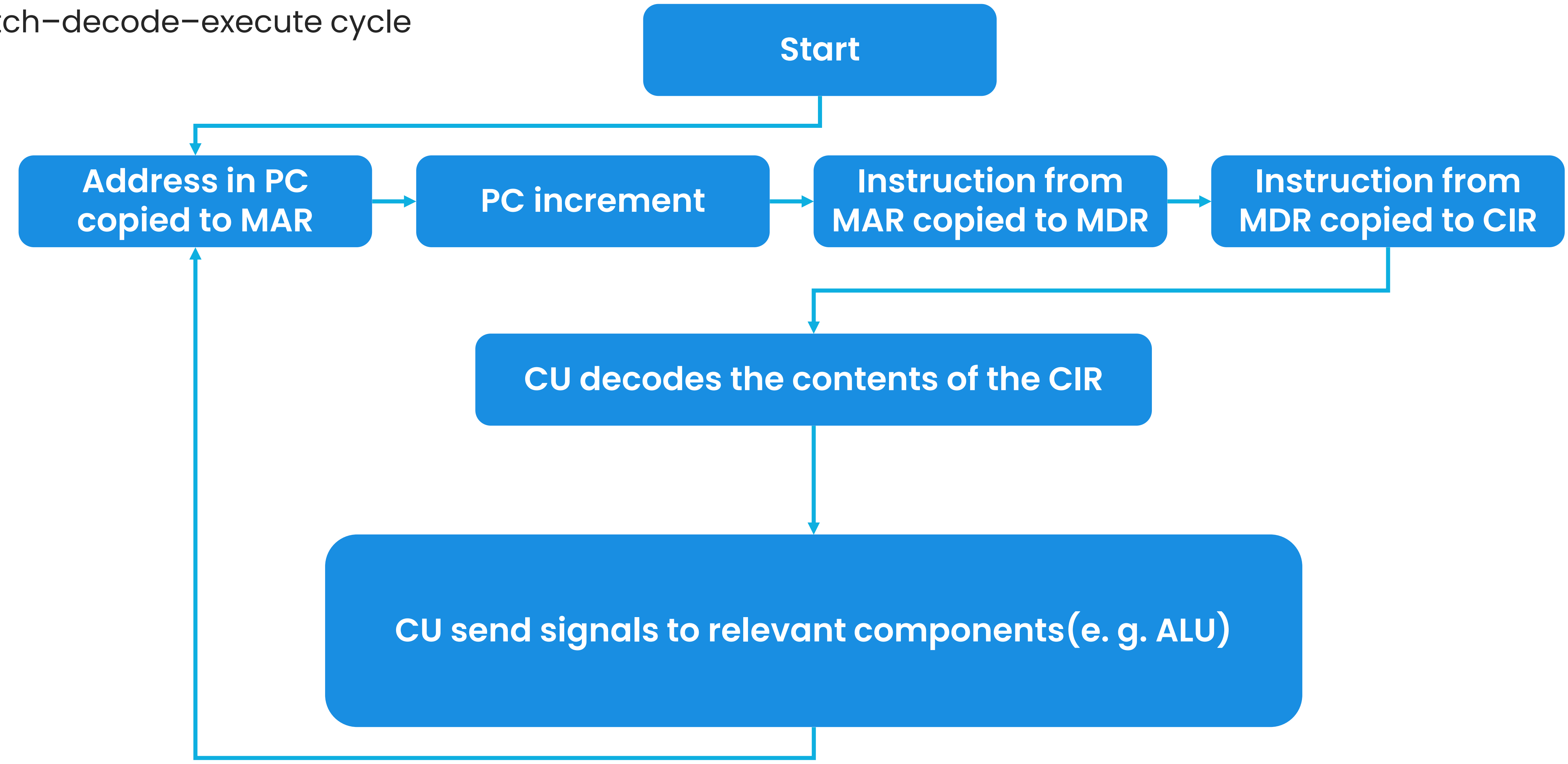
Writing to Flash memory

Flashing the microcontroller



Code execution

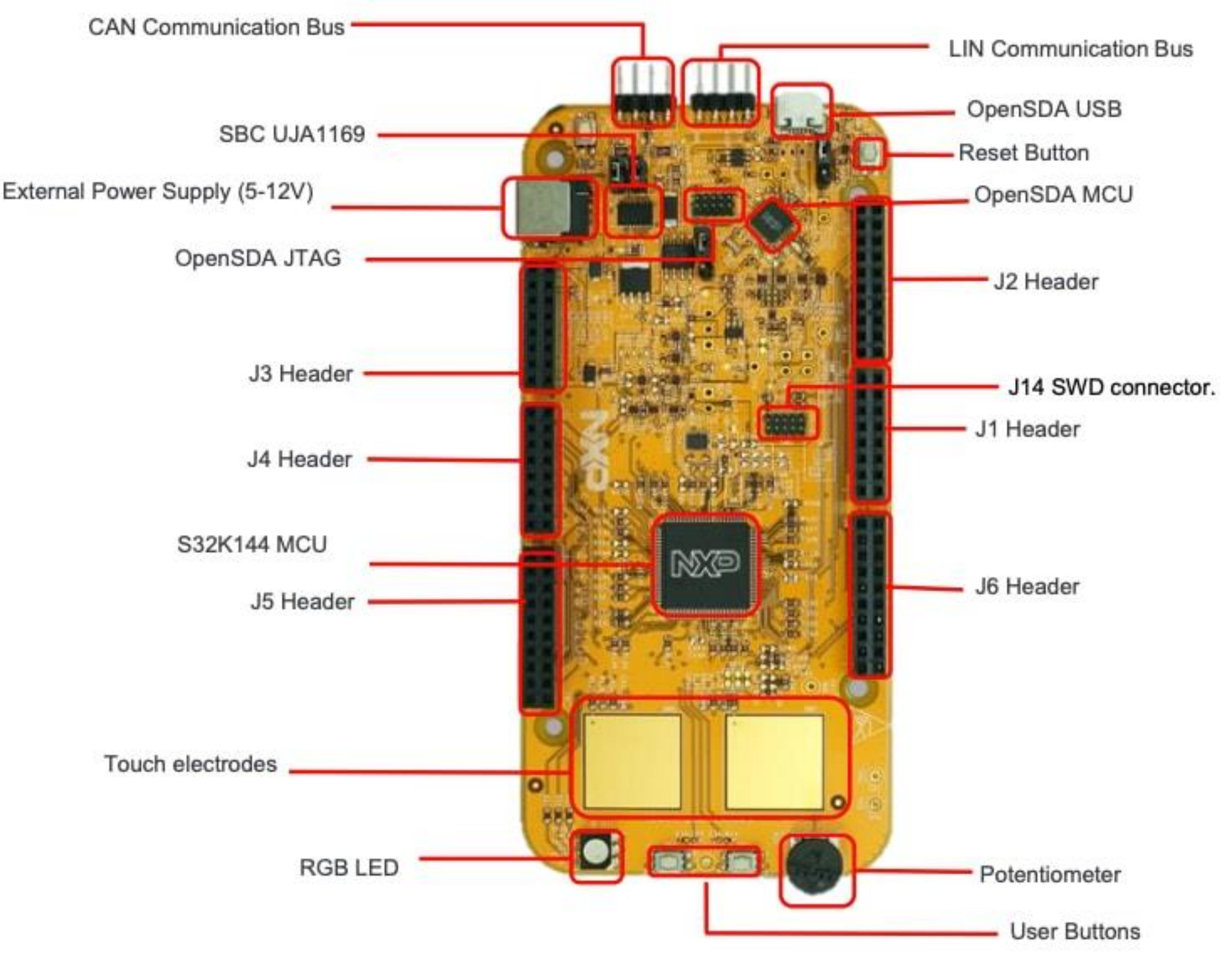
fetch–decode–execute cycle



02

S32K144

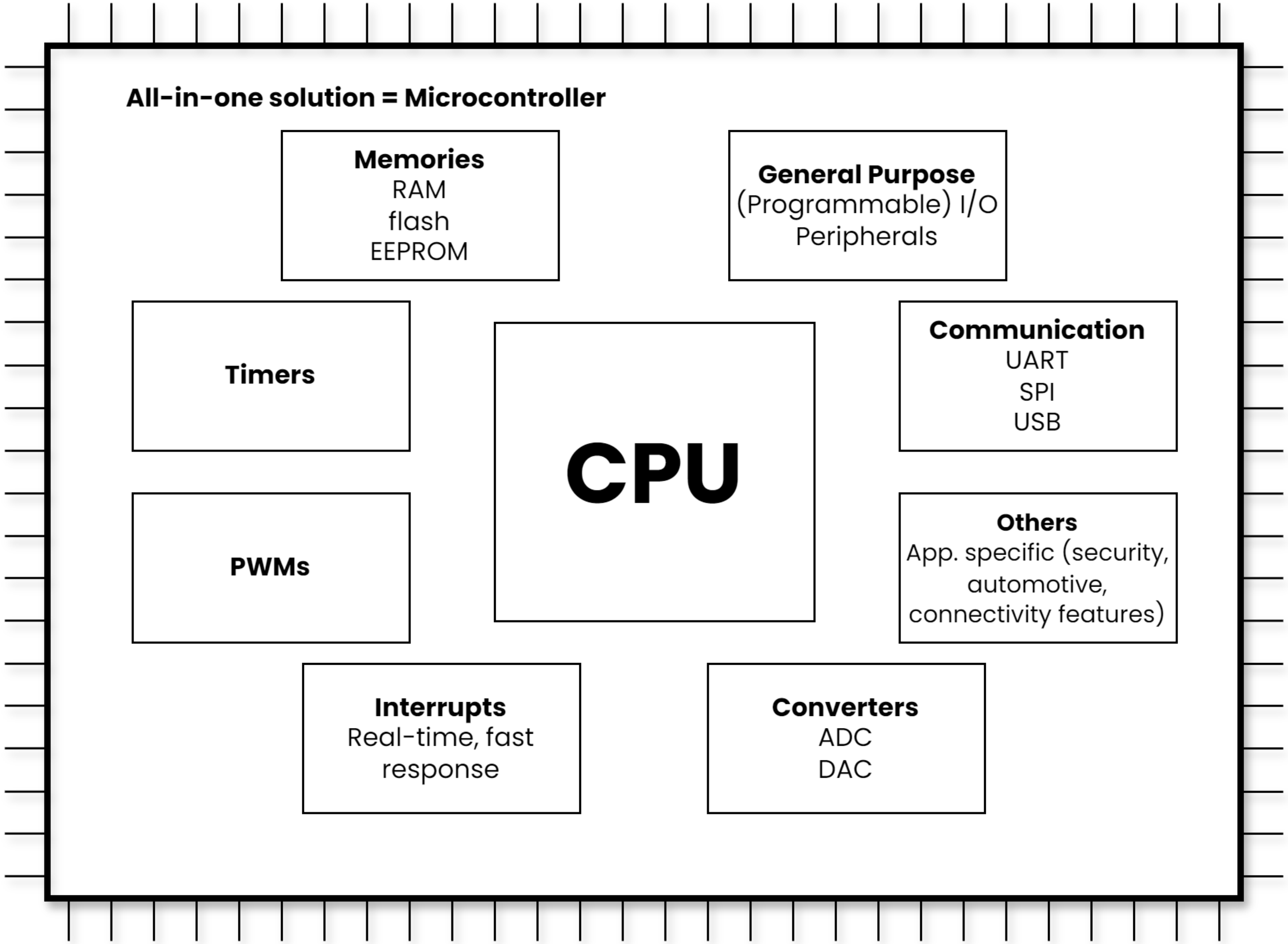
Get to Know Your Evaluation Board



Microcontroller Features | Hands-on Lab I/O, PWM, ADC, DAC, Timer, Interrupts

nxp.com

What *IS* a microcontroller?

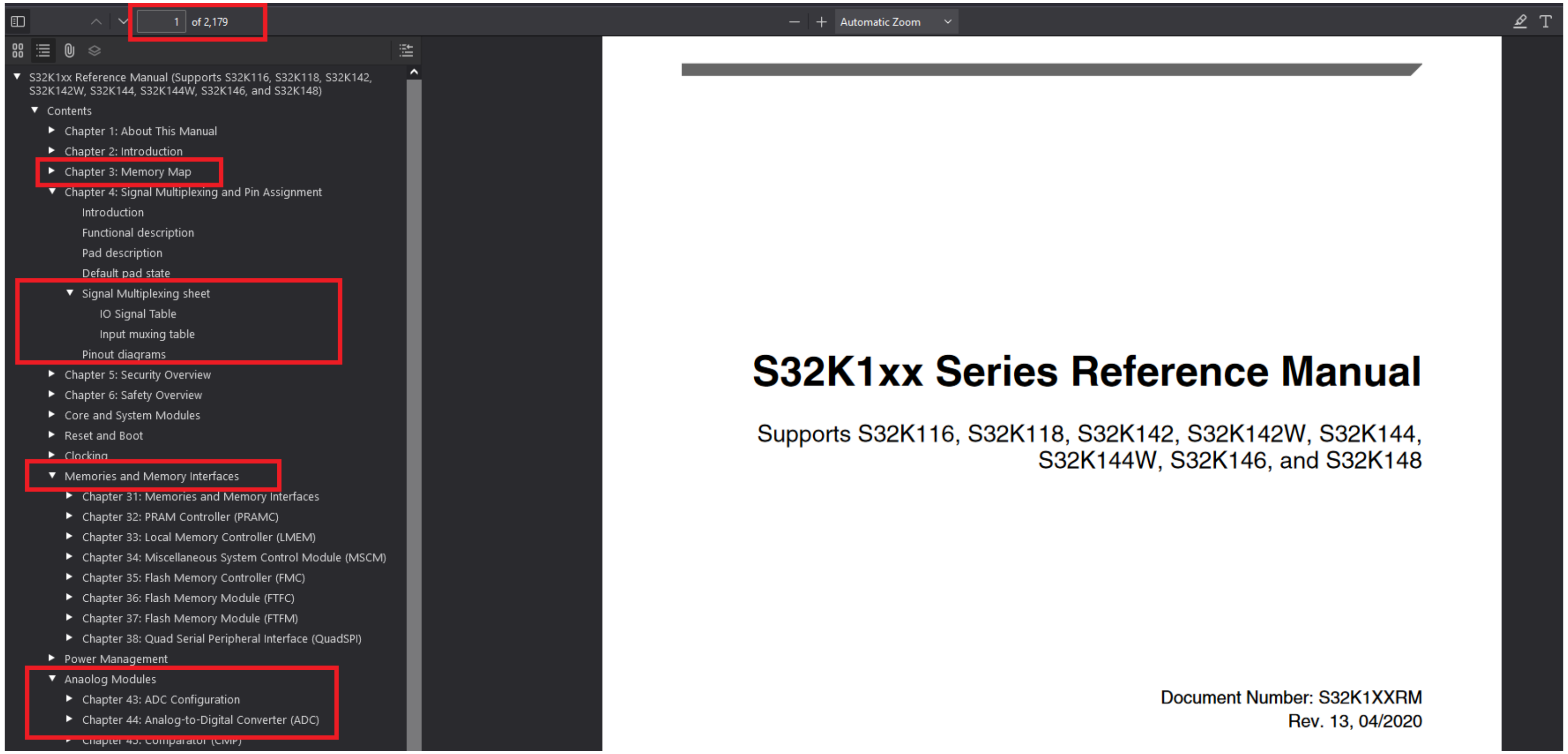


What *IS* a microcontroller?

- [S32K1 Microcontrollers for Automotive General Purpose]
- [S32K1xx Series Reference Manual (PDF Download)]

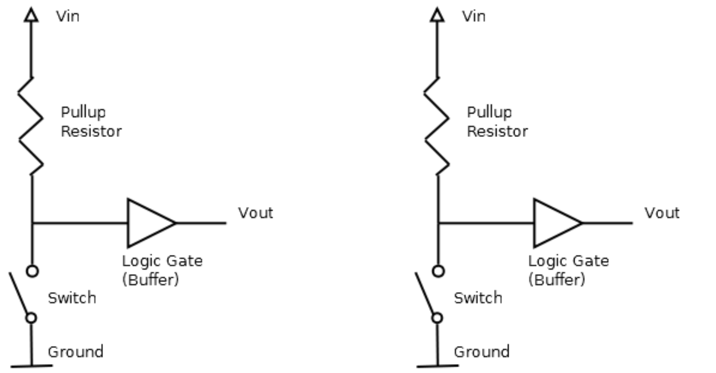
64 LQFP

100 LQFP

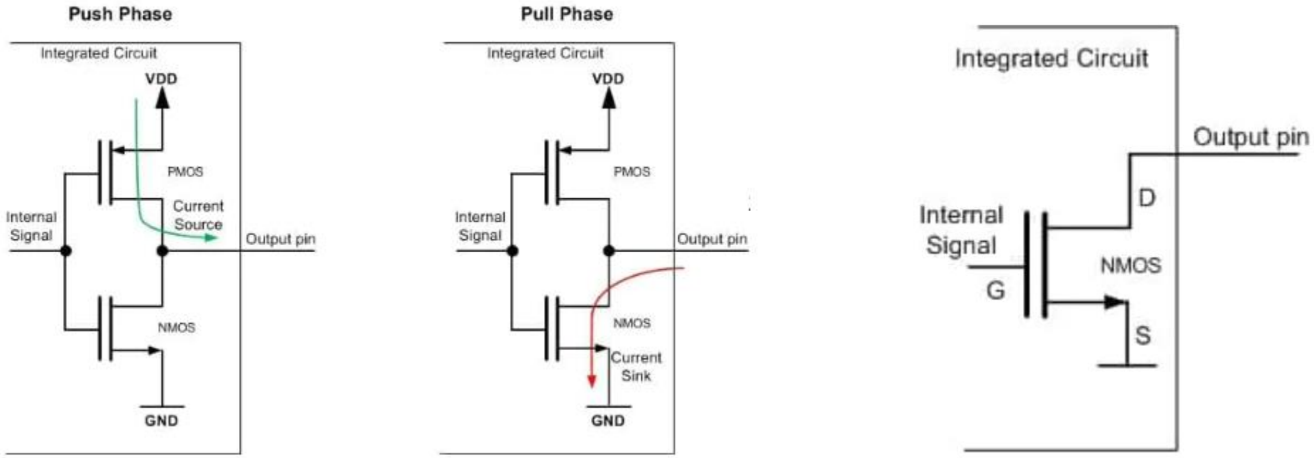


I/O – Input/Output Peripheral

- 4 types of GPIO modes:
  - **Input** mode – receive data from external input device (sensor)
    - Pull-up, Pull-down, Floating point configurations

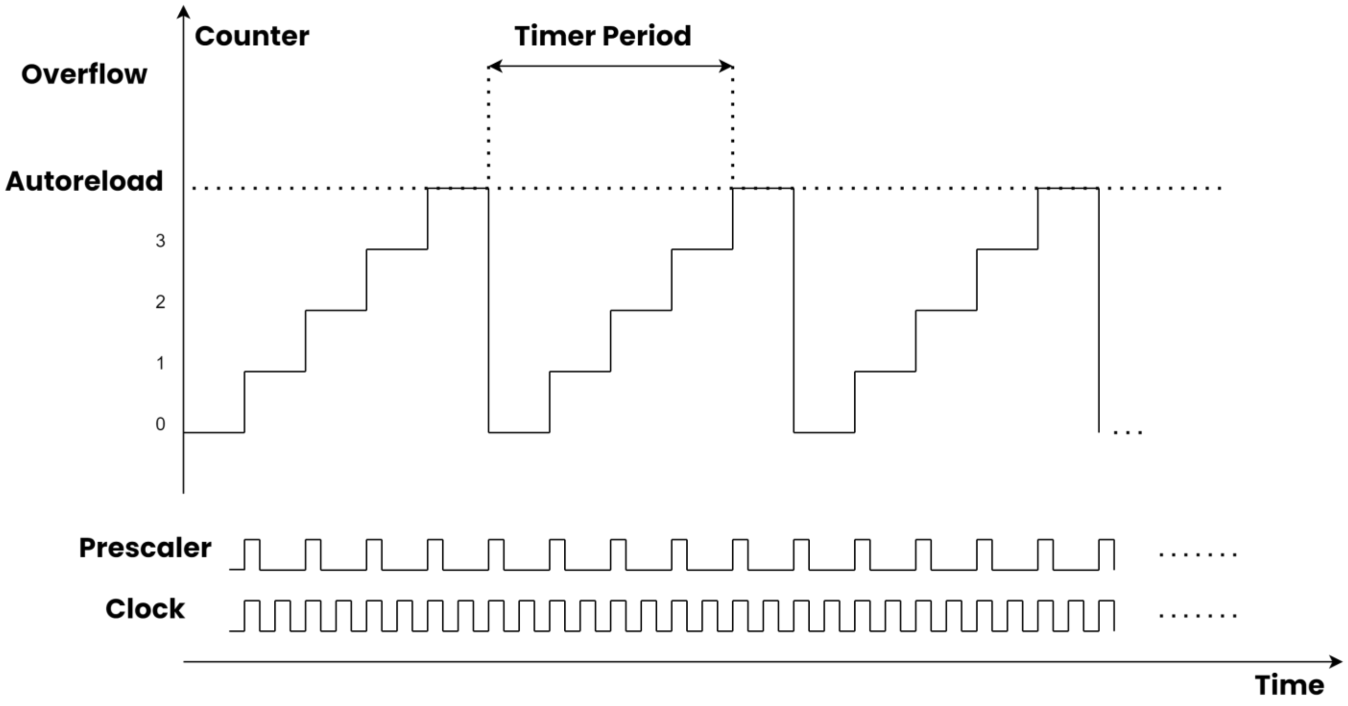


  - **Output** mode
    - Push-pull, Open drain configurations



  - **Analog** mode – **ADC**, **DAC**
  - **Alternate function** mode – **USART**, **I2C** pins, etc.

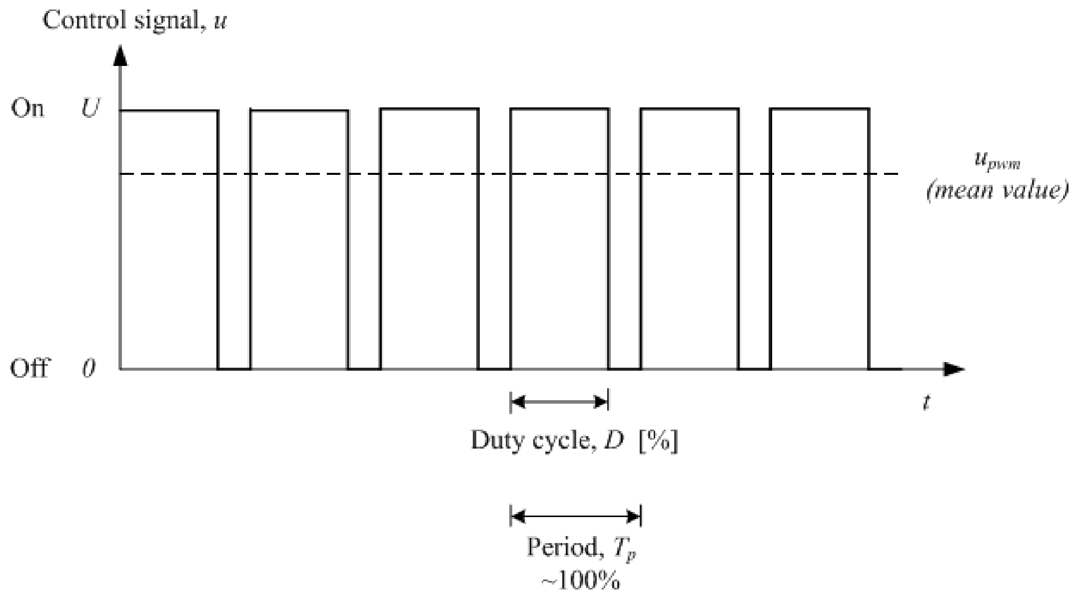
Timers



- Great overview on timers:
  - [Introduction to Microcontroller Timers: Periodic Timers - Technical Articles]
- For further information, please check out this Application Note!
  - [AN5303: Features and Operation Modes of FlexTimer Module on S32K – Application Note]

PWM – Pulse Width Modulation

- **PWM** = generates a **rectangular wave signal** with variable **duty cycle**.



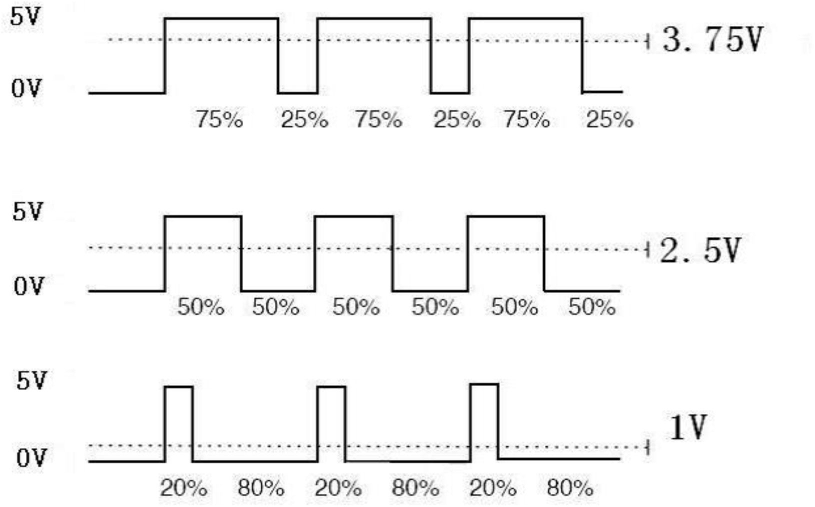
1. Duty Cycle describes the proportion of “on” time to the regular interval or period.

2. Period - the time it takes for a signal to conclude a full ON-OFF cycle

3.The voltage amplitude here is 0V–5V.

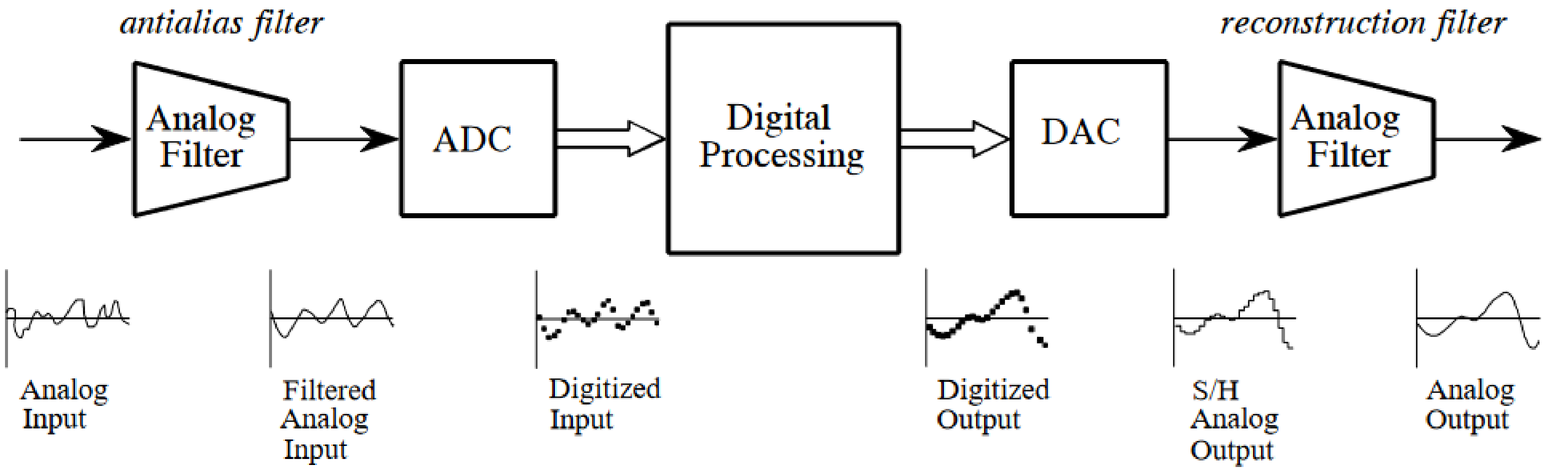
PWM – Pulse Width Modulation

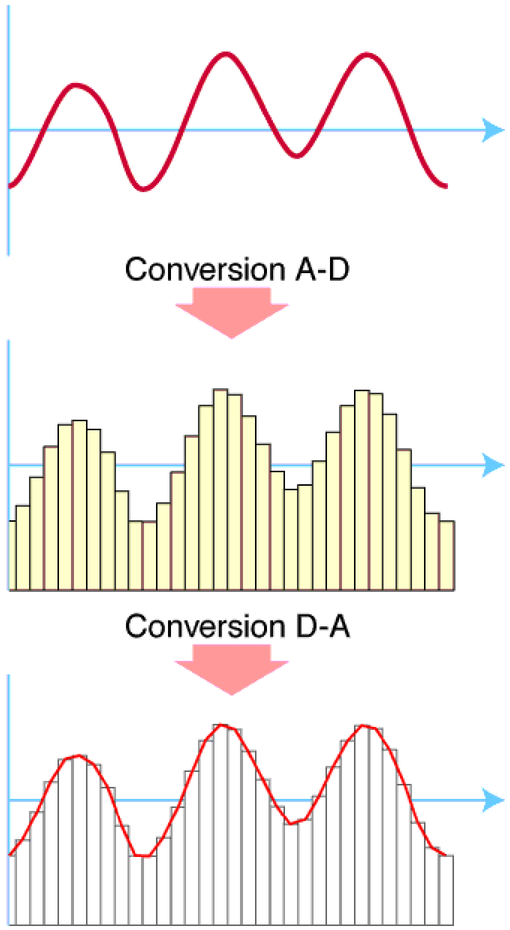
- The PWM can simulate intermediate voltage levels between fully **ON** (5 Volts) and **OFF** (0 Volts) by adjusting the proportion of time the signal remains **ON** within each cycle.



- Use cases:
  - Control the average power or amplitude delivered by a signal
    - Control servomechanisms,
    - Power delivery,
    - Voltage regulation

ADC and DAC – Analog-to-Digital and Digital-to-Analog Converters





- ADC and DAC
- For further information, please check out this Application Note!
  - S32K1xx and S32M24x ADC Guidelines, Spec and Configuration

Interrupts

- [Arm Cortex-M4 Processor Technical Reference Manual Revision r0p1]

7.2 Nested Vectored Interrupt Controller (NVIC) Configuration

This section summarizes how the module has been configured in the chip. Full documentation for this module is provided by Arm and can be found at **arm.com**.

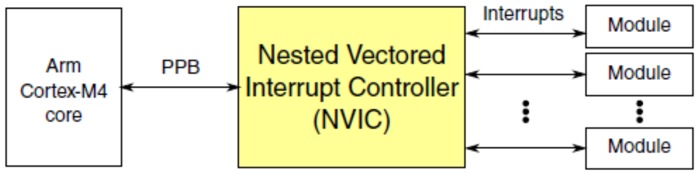


Figure 7-2. NVIC configuration

7.2.1 Interrupt priority levels

This device supports 16 priority levels for interrupts. Therefore, in the NVIC each source in the IPR registers contains 4 bits. For example, the IPR0 diagram is shown below.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	IRQ3				0	0	0	0	IRQ2				0	0	0	0	IRQ1				0	0	0	0	IRQ0				0	0	0	0
W																																

Brighter Together

nxp.com