Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Systèmes Embarqués 1 & 2

tp.01 - Introduction

Classes T-2/I-2 // 2016-2017

Daniel Gachet | HEIA-FR/TIC

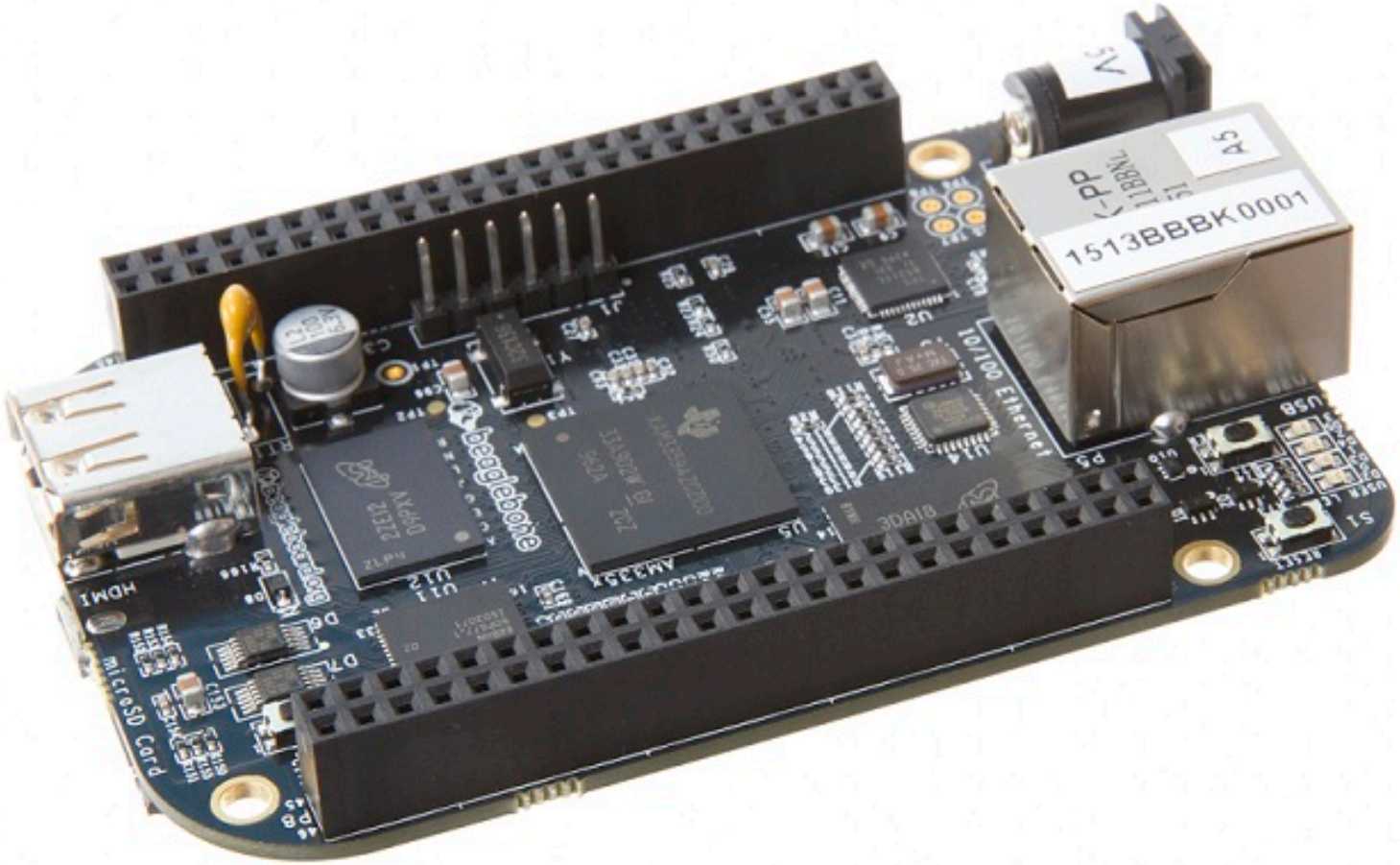
tp.01 | 20.09.2016

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz

Objectifs

- A la fin du laboratoire, les étudiant-e-s seront capables de
  - Décrire les processus de développement en micro-informatique
  - Appliquer les règles de fonctionnement du travail en laboratoire
  - Trouver le matériel et la documentation dans le laboratoire
  - Décrire les processus d'assemblage et d'édition de liens d'une application compilée (ou assemblée)
  - Manipuler et connecter correctement les cibles
  - Manipuler les fonctions de base de l'environnement de développement
  - Décrire les caractéristiques principales du microprocesseur
  - Analyser le déroulement d'un programme élémentaire codé en assembleur
- Durée
  - 1 séance de laboratoire (4 heures)
- Rapport
  - Rapport de laboratoire

Beaglebone Black – Cible



Beaglebone Black - Features

	Feature	
Processor	Sitara AM3358BZCZ100 1GHz, 2000 MIPS	
Graphics Engine	SGX530 3D, 20M Polygons/S	
SDRAM Memory	512MB DDR3L 800MHZ	
Onboard Flash	4GB, 8bit Embedded MMC	
PMIC	TPS65217C PMIC regulator and one additional LDO.	
Debug Support	Optional Onboard 20-pin CTI JTAG, Serial Header	
Power Source	miniUSB USB or DC Jack	5VDC External Via Expansion Header
PCB	3.4" x 2.1"	6 layers
Indicators	1-Power, 2-Ethernet, 4-User Controllable LEDs	
HS USB 2.0 Client Port	Access to USB0, Client mode via miniUSB	
HS USB 2.0 Host Port	Access to USB1, Type A Socket, 500mA LS/FS/HS	
Serial Port	UART0 access via 6 pin 3.3V TTL Header. Header is populated	
Ethernet	10/100, RJ45	
SD/MMC Connector	microSD , 3.3V	
User Input	Reset Button Boot Button Power Button	
Video Out	16b HDMI, 1280x1024 (MAX) 1024x768,1280x720,1440x900 ,1920x1080@24Hz w/EDID Support	
Audio	Via HDMI Interface, Stereo	
Expansion Connectors	Power 5V, 3.3V , VDD_ADC(1.8V) 3.3V I/O on all signals McASP0, SPI1, I2C, GPIO(69 max), LCD, GPMC, MMC1, MMC2, 7 AIN(1.8V MAX), 4 Timers, 4 Serial Ports, CAN0, EHRPWM(0,2),XDMA Interrupt, Power button, Expansion Board ID (Up to 4 can be stacked)	
Weight	1.4 oz (39.68 grams)	
Power	Refer to Section 6.1.7	

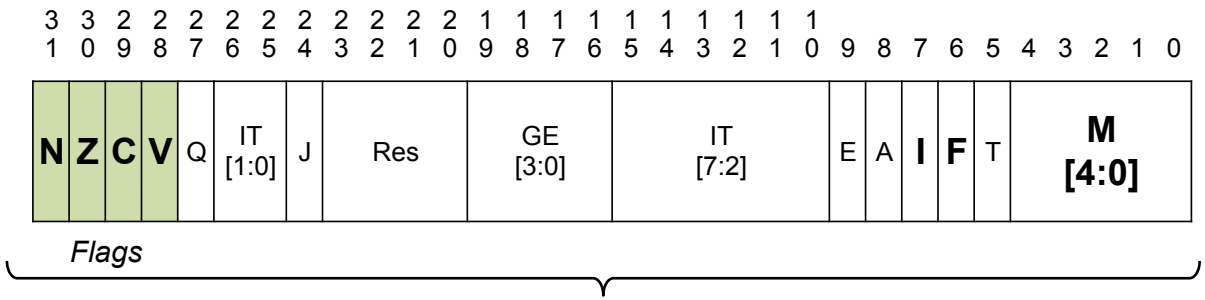
ARM Cortex-A8 et ses registres internes

µP Registers

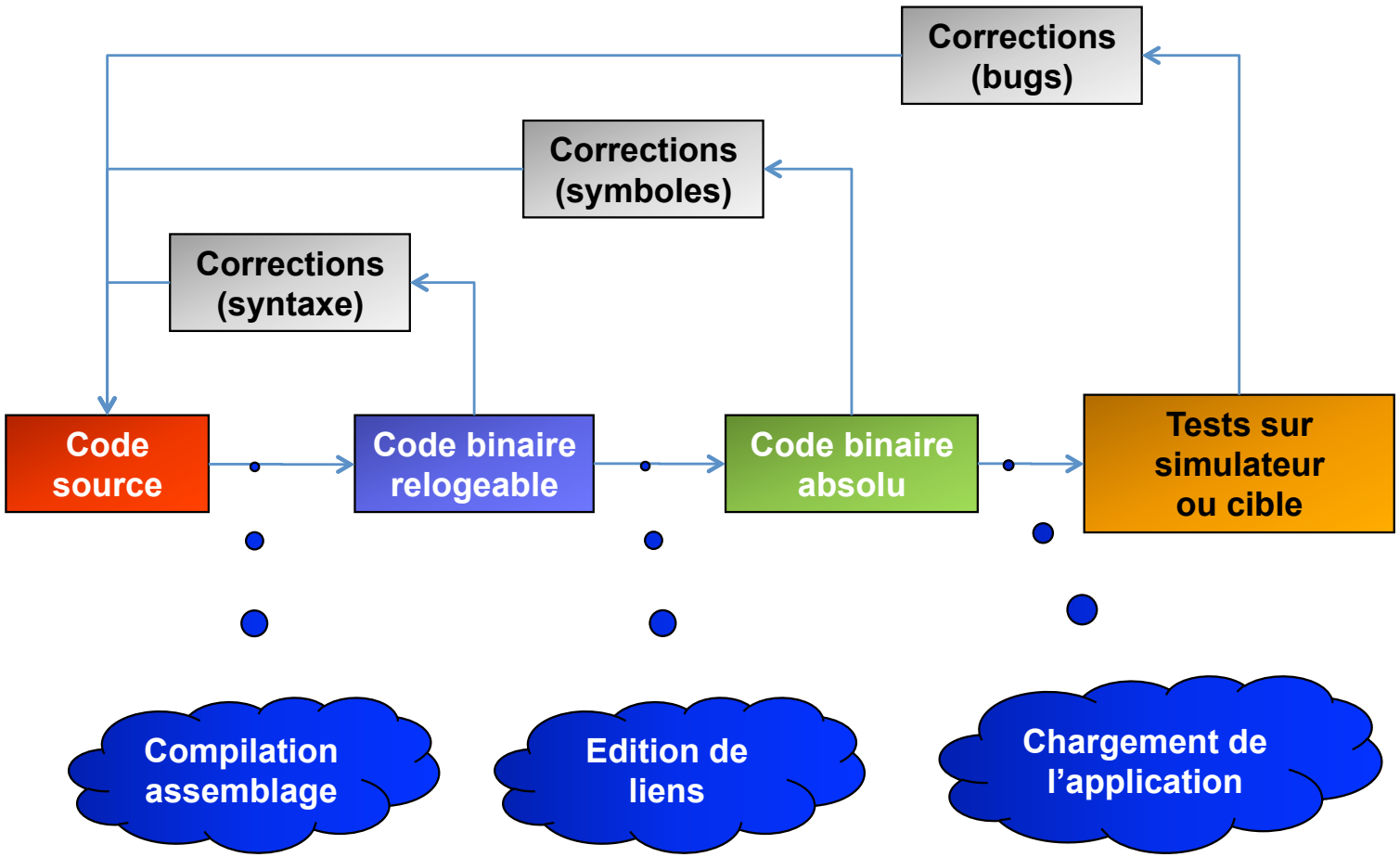
R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13 (sp)
R14 (lr)
R15 (pc)

CPSR

Current Program Status Register (CPSR)



Processus de développement logiciel



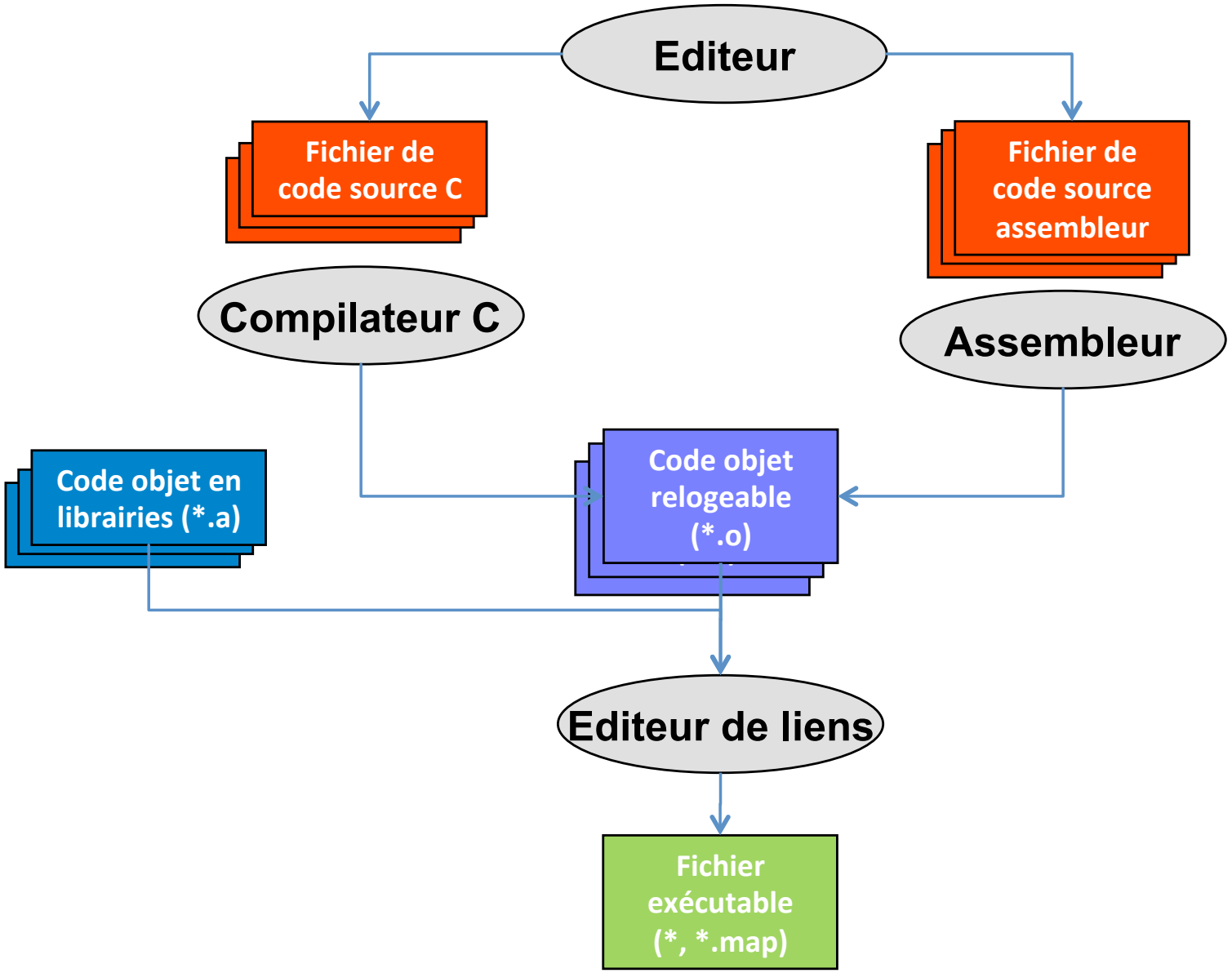
Processus de développement logiciel (II)

- Le développement de logiciel suit les étapes suivantes
  - Le code source assembleur, C ou autre est écrit dans des fichiers texte (ascii) à l'aide d'un éditeur de texte.
  - Le code source est ensuite compilé pour obtenir le code binaire « relogeable » exécutable par le µP. Une phase de correction liée à l'écriture correcte du code source (syntaxe) est naturellement comprise dans cette partie du processus.
  - Le fichier binaire est lié à l'architecture matérielle. Cette étape se divise en deux phases. La première consiste à résoudre les liens entre les différents symboles (p.ex. appel de fonction, adresses de variables globales,...). Si des symboles restent indéfinis, une phase de correction devra être entreprise. La deuxième consiste à reloger le code selon l'organisation mémoire de la cible. En fonction des systèmes d'exploitation et des environnements de développement, cette phase peut intervenir lors du chargement du code sur la cible.

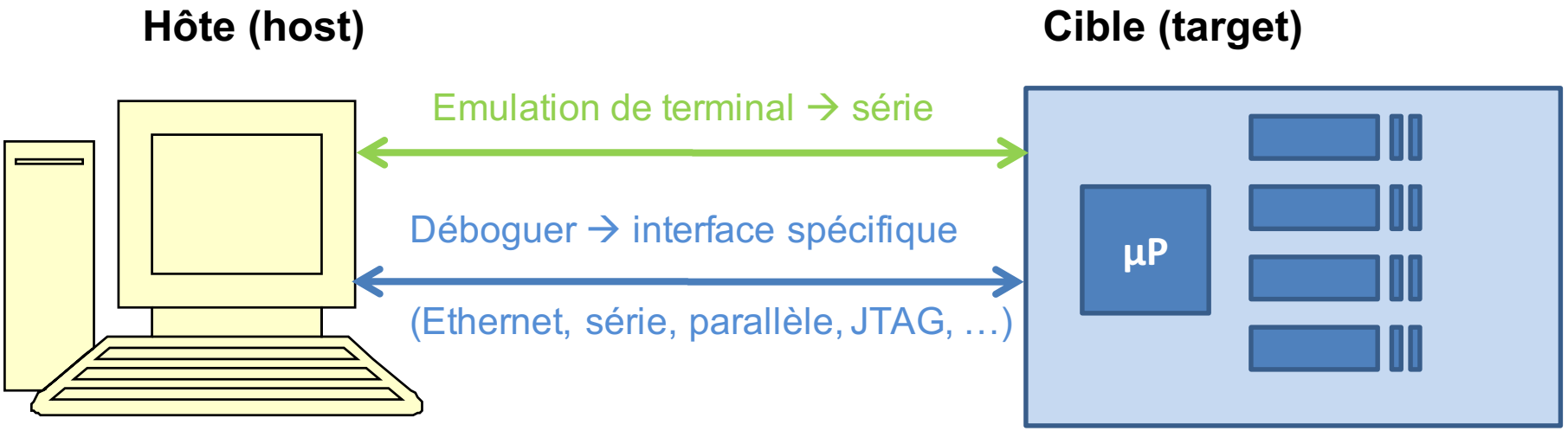
- Le code est chargé en mémoire sur le système cible (système de test ou simulateur). Il est ensuite testé de manière exhaustive. Une série de corrections liées au mauvais fonctionnement du programme (bugs) peuvent intervenir à cette étape.

Attention Des erreurs dues à une mauvaise conception (design) de l'application qui apparaissent lors de sa vérification peuvent être très complexes à identifier et coûter beaucoup d'énergie et d'argent pour être corrigées !

Processus d'implémentation du logiciel



Développement de logiciel pour systèmes embarqués



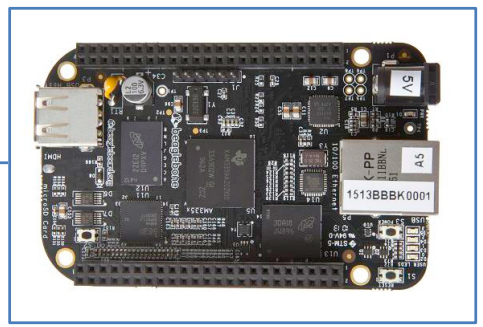
- Développement de l'application
  - Edition code source
  - Compilation / Assemblage
  - Edition de liens (linking)
- Débogueur
  - Code source
  - Symboles / adresses
- Terminal (optionnel)
  - Commandes
  - Tracing / logging
- Exécution de l'application
  - Code binaire
- Serveur de débogage
  - Agent (i/f avec débogueur)

Développement avec le Beaglebone Black

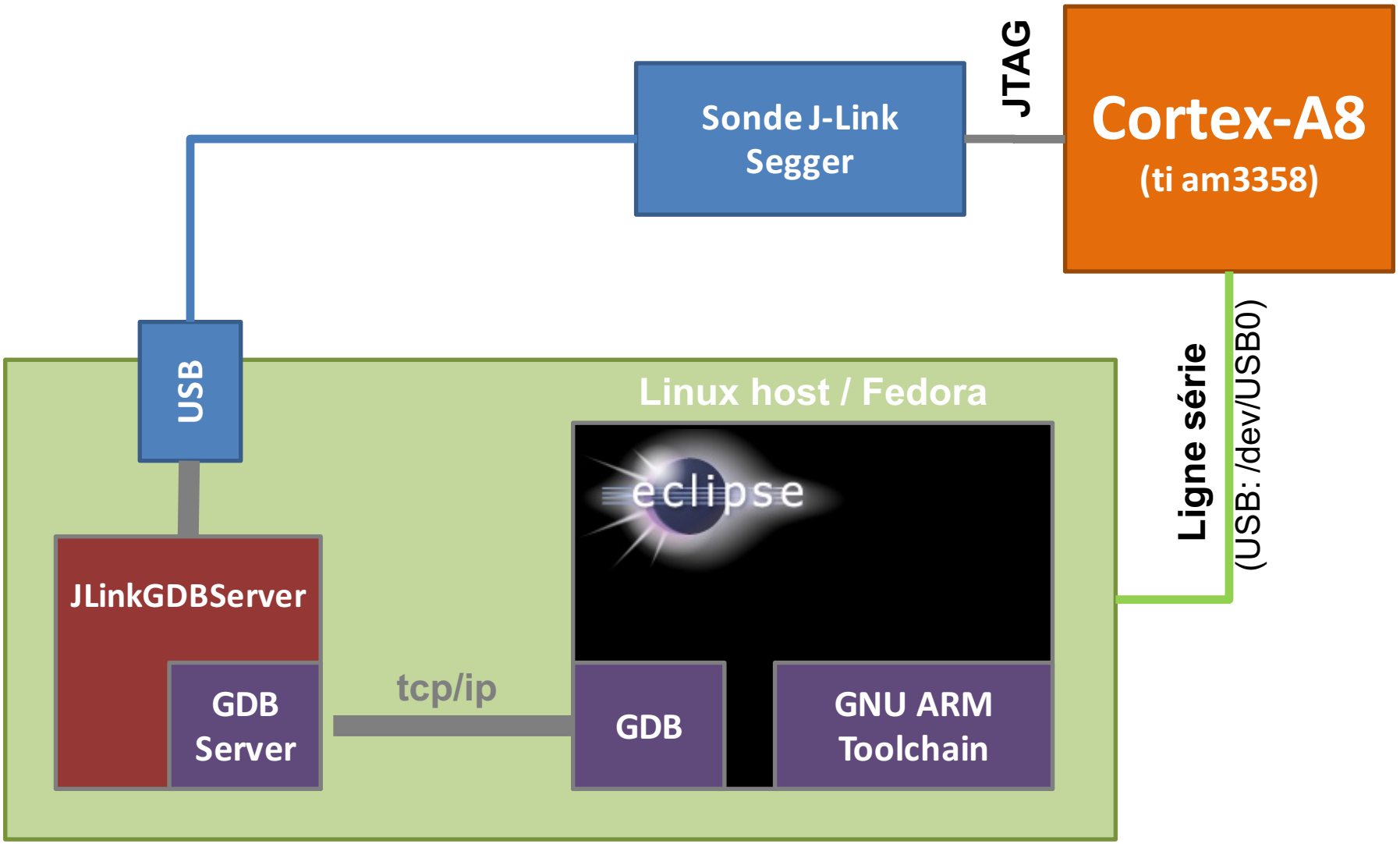
Linux PC
Fedora

J-Link
Segger

Beaglebone Black
TI AM3358 / Cortex-A8



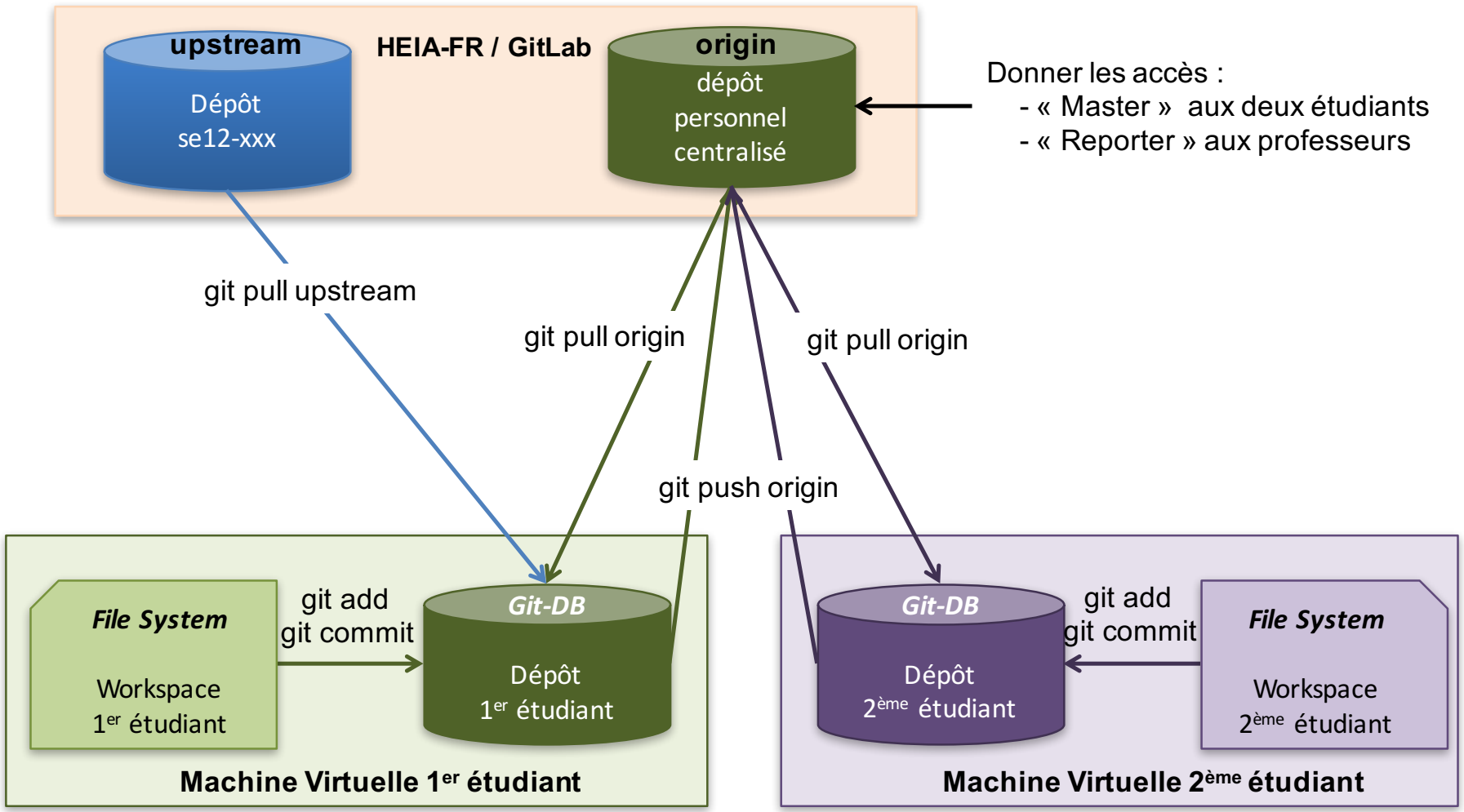
Vue détaillée de la chaîne d'outils de développement



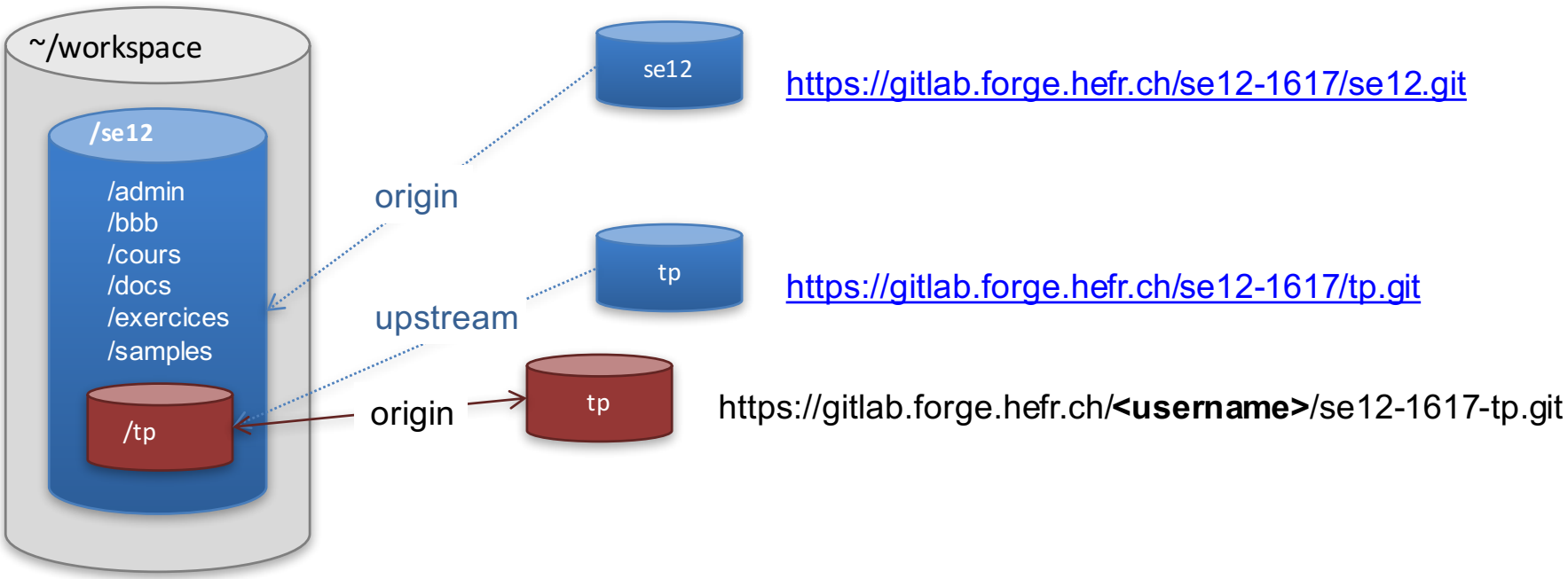
Outils de développement (IDE)

- Pour le développement,nous n'utiliserons principalement des outils libres sous Linux Fedora 24
  - Eclipse C/C++ 4.6 (Neon), utilitaire de gestion de projet de développement informatique (IDE – Integrated Development Environment)
  - GNU-toolchain, outils de développement d'applications, avec
    - Les binutils (assembler, linker, ...) version 2.25
    - Le compilateur C version 5.2.0
    - Le débogueur GDB version 7.6.2
    - Le make version 4.1
    - La librairie standard newlib 2.2.0-1 de RedHat
  - Git pour la gestion de tout le code source
  - J-Link Software V6.00i, utilitaire permettant d'interfacer la chaine d'outil GNU avec la cible au travers d'une interface USB/JTAG
  - Une configuration de projet spécifique HEIA-FR pour notre cible

Organisation de l'espace de travail et des dépôts Git



Organisation de l'espace de travail et des dépôts Git (II)



- Chaque étudiant peut choisir entre 2 options
  - Option 1
    - Travailler avec une machine virtuelle VMware sur sa propre machine
  - Option 2
    - Travailler avec sa machine personnelle sous Linux (Fedora 24)
- Installation de la machine
  - L'option 1 est favorisée, car elle offre une plus grande flexibilité
  - Pour l'option2, suivre les instructions des slides Backups
- Nommage de l'espace de travail
  - Option 1 : /home/lmi/workspace/se12
  - Option 2 : /home/**<username>**/workspace/se12

Option 1 : Installation sur machine virtuelle VMware

- Copier de la clef USB le VMware souhaité sur la machine personnelle
  - VMware-Fusion-8.xxx-...dmg pour Mac OS X
  - VMware-workstation-full-12.xxx-...exe pour Windows
  - VMware-Workstation-Full-12.xxx-...x86_64.bundle pour Linux
- Ouvrir le lien : http://hes-so.onthehub.com
- Se loger avec le compte HES-SO
- Aller sous « Students » / « VMware »
- Choisir le logiciel correspondant à la machine personnelle
- Ajouter le logiciel dans le panier « Add To Cart »
- Effectuer le login Switch « HES-SO - Haute école spécialisée de Suisse occidentale »
- Effectuer l'achat « Check Out » / « Proceed With Order »
- Installer la licence

Création et installation de l'environnement

- Créer le dépôt de groupe pour les tp (1 par groupe de 2 personnes)
  - Avec un browser aller sur le Git de l'école (https://gitlab.forge.hefr.ch/)
  - Sélectionner l'onglet « Projects » et cliquer « + New Project »
  - Nomer le projet « se12-1617-tp » et cliquer « Create project »
  - Noter l'URL du projet (https://gitlab.forge.hefr.ch/<username>/se12-1617-tp)
  - Ajouter le professeur comme « Reporter » (au minimum) au projet (https://gitlab.forge.hefr.ch/<username>/se12-tp/project_members)
- Ouvrir l'espace de travail (workspace)
  - Ouvrir un terminal (une shell Linux) et entrer dans le workspace

    ```
    $ cd ~/workspace/se12/tp
    ```
- Configurer Git

    ```
    $ git config --global user.name "<User Name>"
    $ git config --global user.email user.name@edu.hefr.ch
    ```

Création et installation de l'environnement (II)

- Mettre à jour le dépôt local

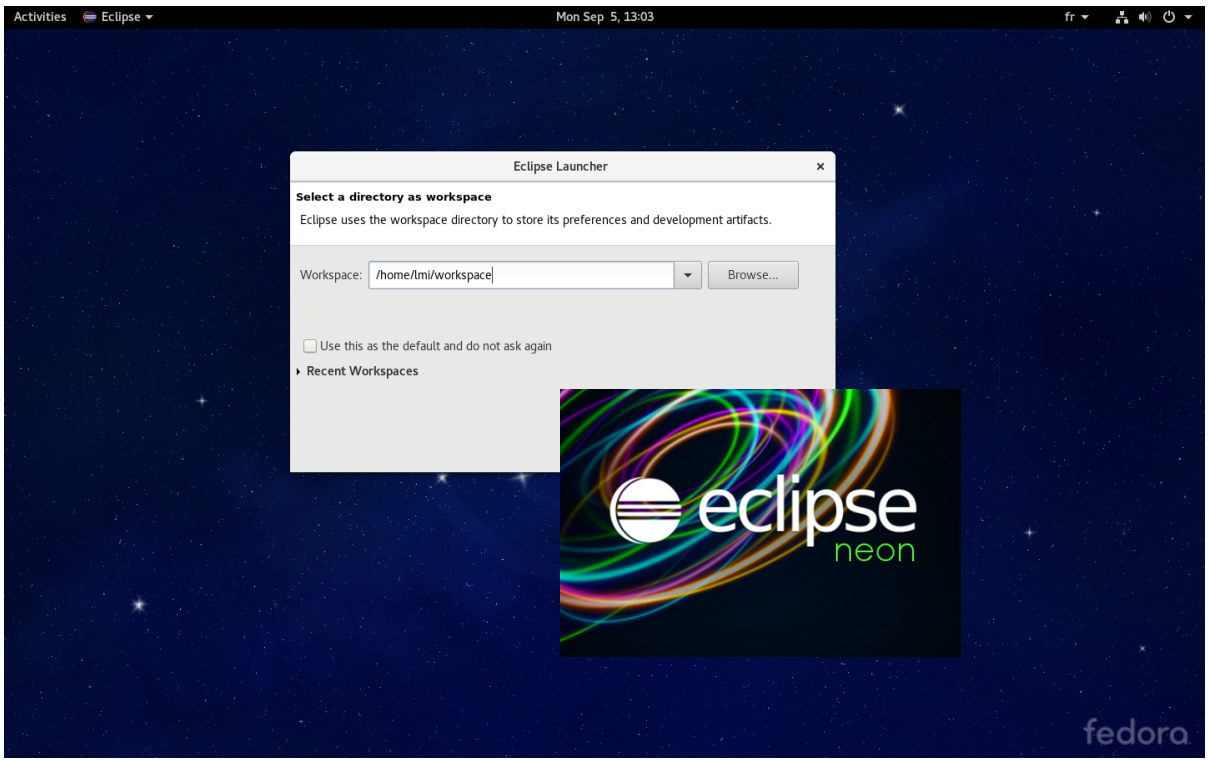
```
$ git pull upstream master
```

- Ajouter les « tp » sur le dépôt local et synchroniser avec le dépôt de groupe

```
$ git remote add -t master -m master origin
https://gitlab.forge.hefr.ch/<username>/se12-1617-tp.git
$ git push origin master
```

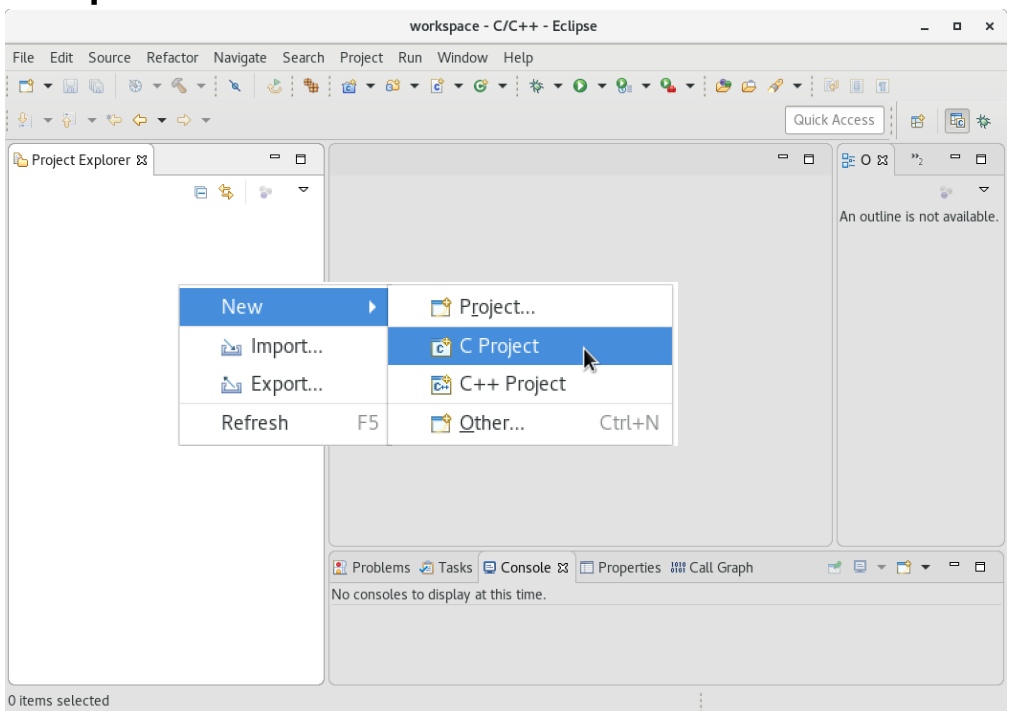
Démarrage de l'IDE

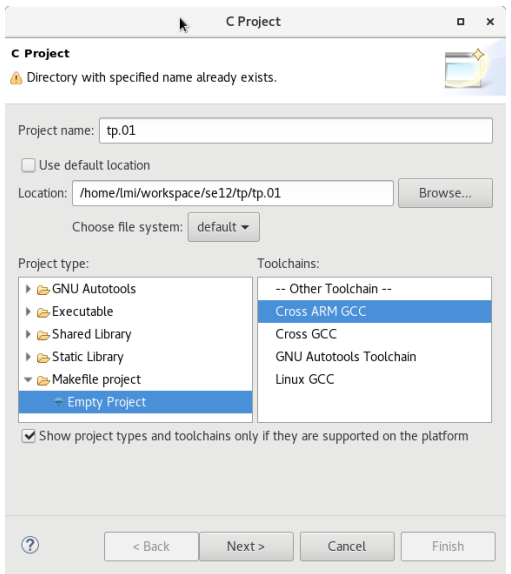
- A partir de votre station de travail Linux, démarrer l'IDE« Eclipse ».
- Placer le workspace sur le répertoire
  - Option 1 : /home/lmi/workspace
  - Option 2 : /home/<username>/workspace



Création d'un nouveau projet

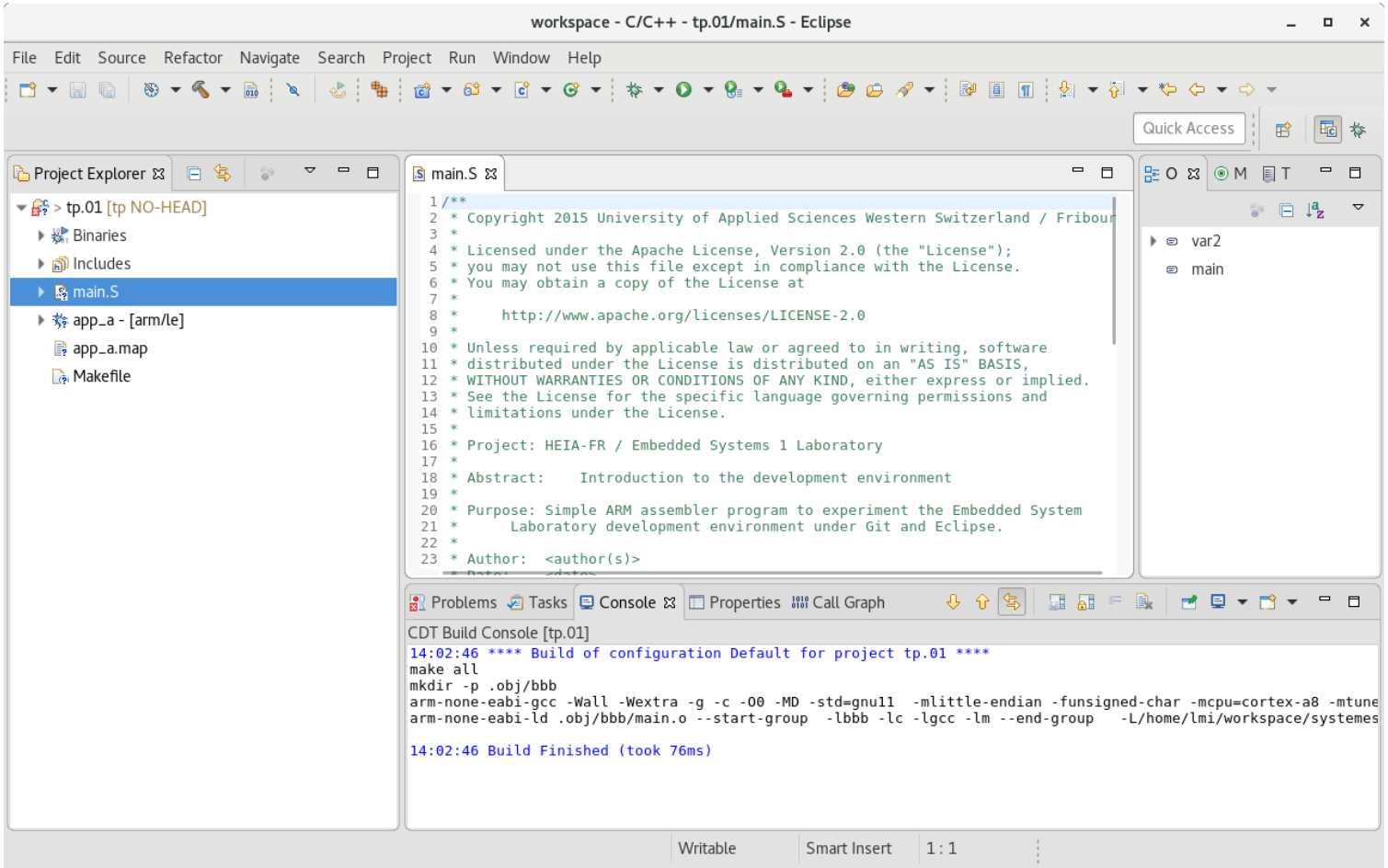
- Aller dans le « *Project Explorer* », cliquer le button droit de la souris et choisir « *New* → *C Project* »
- Sélectionner la location du projet « *<.../se12/tp/tp.01>* »
- Donner un nom « *tp.01* »
- Choisir « **Makefile Project → Empty Project → Cross ARM GCC** »
- Cliquer « *Next* » → « *Next* » → « *Finish* »





Génération de l'application

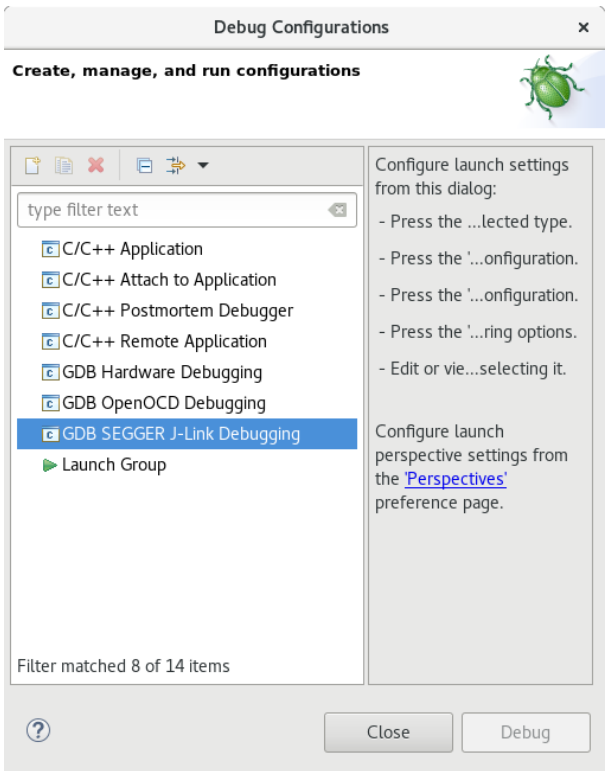
- Cliquer sur le marteau ou « CTRL-B »

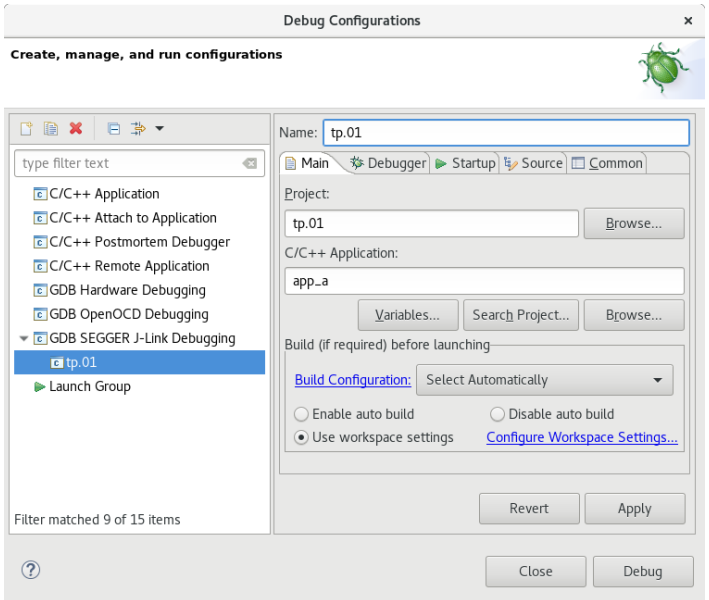


Attention : il est important de contrôler qu'aucune erreur ne soit survenue durant la génération de votre application !

Configuration du débogueur (I – main)

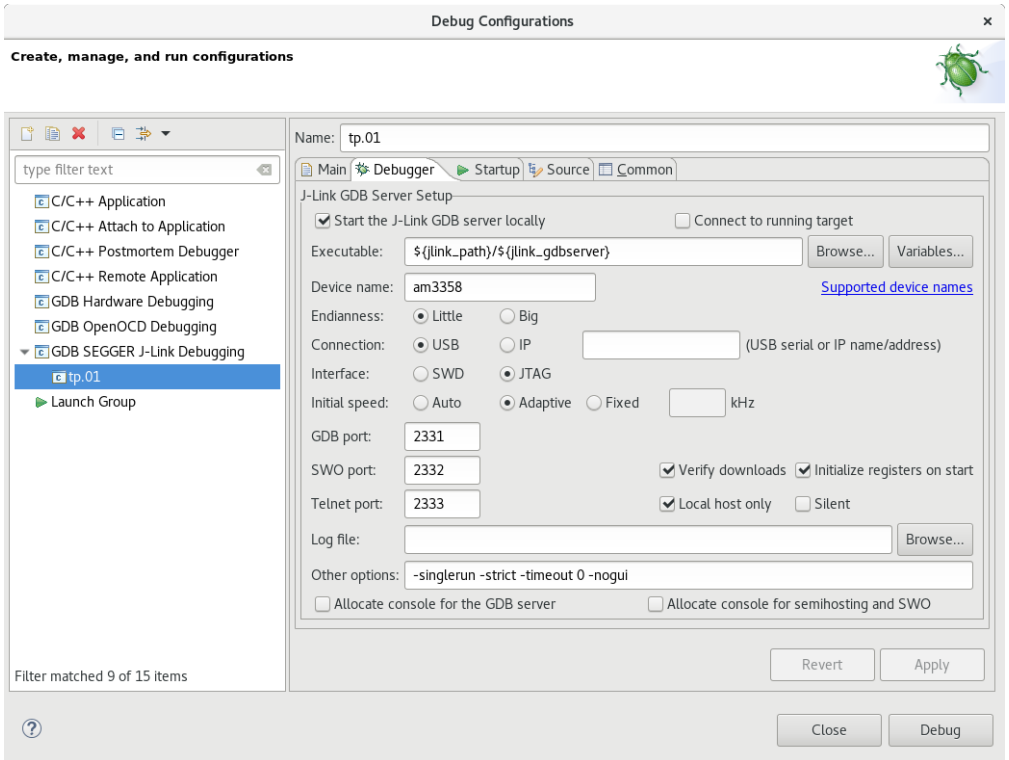
- Cliquer avec la flèche le petit insecte
- Choisir « *Debug Configurations* »
- Double-cliquer sur « *GDB SEGGER J-Link Debugging* »
- Choisir le projet « *tp.01* » et l'application « *app_a* » et nommer « *tp.01* »





Configuration du débogueur (II – Debugger)

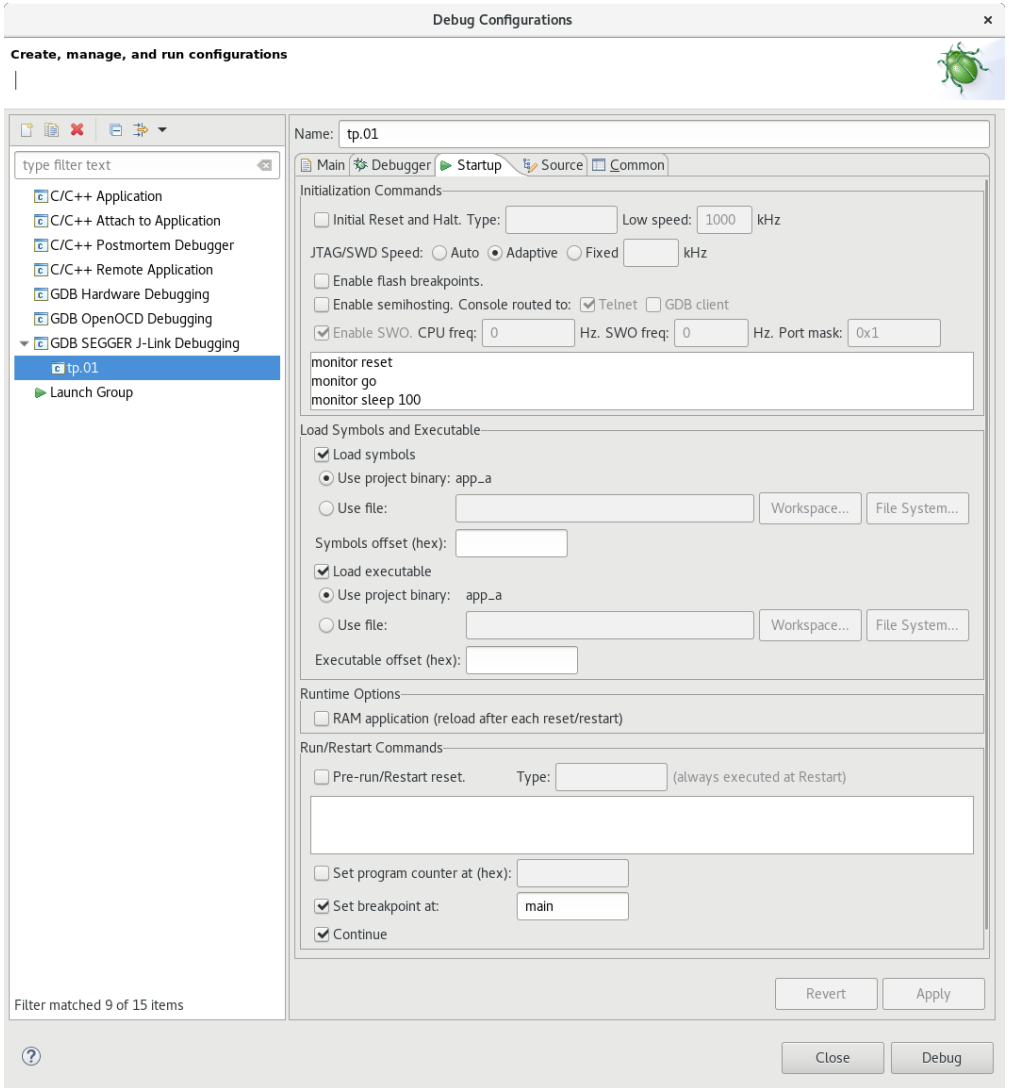
- Ajouter dans « *Device name :* » « *am3358* »
- Choisir dans « *Interface :* » « *JTAG* »
- Choisir dans « *Initial speed :* » « *Adaptive* »
- Décocher « *Allocate console for the GDB server* »
- Décocher « *Allocate console for semihosting and SWO* »



Configuration du débogueur (III – Startup)

- Décocher
  - *Initial Reset…*
  - *Enable flash breakpoints.*
  - *Enable semihosting…*
- Choisir dans « *JTAG/SWD Speed : » « Adaptive* »
- Introduire les commandes

  monitor reset
  monitor go
  monitor sleep 100
  monitor halt
- Décocher
  - *Pre-run/Restart reset.*



Travail à réaliser

- Editez le code suivant à l'intérieur du fichier « main.S »
- Compilez votre code à l'aide de l'environnement de développement
- Exécutez votre code en mode « pas à pas » en l'ayant chargé préalablement sur la cible
- Indiquez dans le code source la signification de chaque instruction
- Ecrivez dans le code en commentaire, l'algorithme équivalent en langage évolué (JAVA ou C)
- Optimisez le code mais sans changer l'algorithme
- Répondez aux questions
- Rédigez votre journal de laboratoire
- Rendez votre code et votre journal au travers de Git

Impératifs

- Le travail pratique doit être exécuté par chaque étudiant
- Le dépôt Git d'un groupe doit contenir le tag de chacun des membres du groupe
- A la fin du TP, le workspace sur la machine locale de chacun des membres du groupe doit être synchronisé avec le dépôt Git commun au groupe

Code...

```
/** <copyright & heading...>  */
// Export public symbols
        .global main, res, incr, i
// Declaration of the constants
#define LOOPS 8
// Initialized variables declaration
        .data
        .align     8
res :   .long     16
incr :  .short    32
// Uninitialized variables declaration
        .bss
        .align  8
i :     .space  4
// Assembler functions implementation
        .text
main :  nop
```

```
        mov     r0, #LOOPS
        ldr     r1, =incr
        ldrh    r1, [r1]
        ldr     r3, =res
        ldr     r4, =i
        mov     r5, #0
        str     r5, [r4]
next :  ldr     r2, [r3]
        add     r2, r1
        str     r2, [r3]
        ldr     r5, [r4]
        add     r5, #1
        str     r5, [r4]
        cmp     r5, r0
        bne     next
1 :     nop
        b       1b
```

Questions…

- Quelle est la taille de chacune des variables ?
- Quelle est la taille du code ?
- Comment procéder pour obtenir ces tailles ?
- Où se trouve chaque variable en mémoire (adresse absolue) ?
- Où se trouve le code en mémoire ?
- Est-il possible d'améliorer l'algorithme ?

Journal

- Le travail effectué durant le laboratoire devra être résumé et synthétisé dans un journal de laboratoire de 1 à 2 pages
  - En-tête
    - Etablissement : HEIA-FR (logo), institut, ...
    - Titre : Systèmes Embarqués I, journal, sujet (TP.01 : Introduction)
    - Auteur (nom, email, classe, ...)
    - Lieu et date
  - Heures de travail en dehors des heures de classe pour ce TP
  - Synthèse de l'étudiant sur ce qu'il a appris/exercé durant le TP
    - Non acquis
    - Acquis, mais à exercer encore
    - Parfaitement acquis
  - Réponses aux questions
  - Remarques / choses à retenir
  - Feedback sur le TP

Journal (II)

- Remarque
  - Le journal doit être rendu sous le format PDF.
  - Il peut être rédigé en français, allemand ou anglais.
  - Il doit être stocké dans le dépôt Git avec le code source sous
    - *journal : .../se12/tp/tp.01/doc/report.pdf*
    - *sources : .../se12/tp/tp.01*
- Délai
  - Le journal et le code doivent être rendus le soir même du TP au plus tard à 24h00

Sauvegarde des sources dans le dépôt Git

- Sauvegarder les modifications dans le dépôt local
  - Ouvrir un terminal (une shell Linux)
  - Consulter l'état du dépôt

    ```
    $ git status
    ```
  - Ajouter éventuellement les nouveaux fichiers

    ```
    $ git add *
    ```
  - Commiter les modifications

    ```
    $ git commit -a -m "un commentaire..."
    ```
- Synchroniser le dépôt local avec les dépôts centralisés (serveurs)
  - Synchroniser avec le dépôt du cours

    ```
    $ git pull upstream master
    ```
  - Synchroniser avec le dépôt personnel

    ```
    $ git pull origin master
    ```
- Sauvegarder le résultat du travail dans le dépôt centralisé (serveur)
  - Pousser la branche sur le dépôt

    ```
    $ git push origin master
    ```

Backup slides

Option 2

Installation de l'environnement sur machine personnelle native

Option 2 : Installation d'une machine personnelle

- Créer l'espace de travail (workspace)
  - Ouvrir un terminal (une shell Linux) et créer le répertoire de travail (workspace)

    ```
    $ mkdir -p ~/workspace
    ```
- Configurer GIT

  ```
  $ git config --global user.name <Firstname Lastame>
  $ git config --global user.email user.name@edu.hefr.ch
  ```
- Créer une copie du dépôt GIT contenant le support de cours

  ```
  $ cd ~/workspace
  $ git clone \
  https://gitlab.forge.hefr.ch/se12-1617/se12.git
  ```
- Installer l'environnement et générer les bibliothèques pour la cible

  ```
  $ ~/workspace/se12/bbb/scripts/install_se12_environment
  ```
- Ajouter le dépôt pour les travaux pratiques

  ```
  $ cd ~/workspace/se12
  $ git clone -o upstream \
  https://gitlab.forge.hefr.ch/se12-1617/tp.git
  ```

Option 2 : Installation d'une machine personnelle (II)

- Installer le plug-in C/C++ GDB Hardware Debugging
  → Help
  → Install new software
  → Work with : *Neon - http ://download.eclipse.org/releases/neon*
  → Mobile and Device Development
  ☒ TM Terminal

  → Press *Add...*
  → Name : *GNU ARM C/C++ Cross Development Tools*
  → Location : *http ://gnuarmeclipse.sourceforge.net/updates*
  ⊟ GNU ARM C/C++ Cross Development Tools
  ☒ GNU ARM C/C++ Cross Compiler
  ☒ GNU ARM C/C++ J-Link Debugging
  ☒ GNU ARM C/C++ OpenOCD Debugging