Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg

HTA-FR – Studiengänge Informatik und Telekom.

Verfasser:
D. Gachet / HTA-FR - Telekommunikation

Embedded systems 1

Praktische Arbeiten PA1
Einführung in die Entwicklungsumgebung

Klassen I-2 / T-2 // 2016-2017

Zielsetzungen:

- **Am Ende der Laborübung sind die Studierenden in der Lage:**
  - Die Entwicklungsprozesse in der Mikroinformatik zu beschreiben
  - Die Funktionsregeln der Arbeit im Labor anzuwenden
  - Die Ausrüstungen und die Dokumentation im Labor zu finden
  - Den Assemblierungs- und Linkprozess einer kompilierten (oder assemblierten) Anwendung zu beschreiben
  - Die Zielsystemen richtig handhaben und anschliessen zu können
  - Mit den Grundfunktionen der Entwicklungsumgebung umzugehen
  - Die Hauptmerkmale des Mikrocontrollers zu beschreiben
  - Den Ablauf eines elementaren, in Assembler codierten Programms zu analysieren

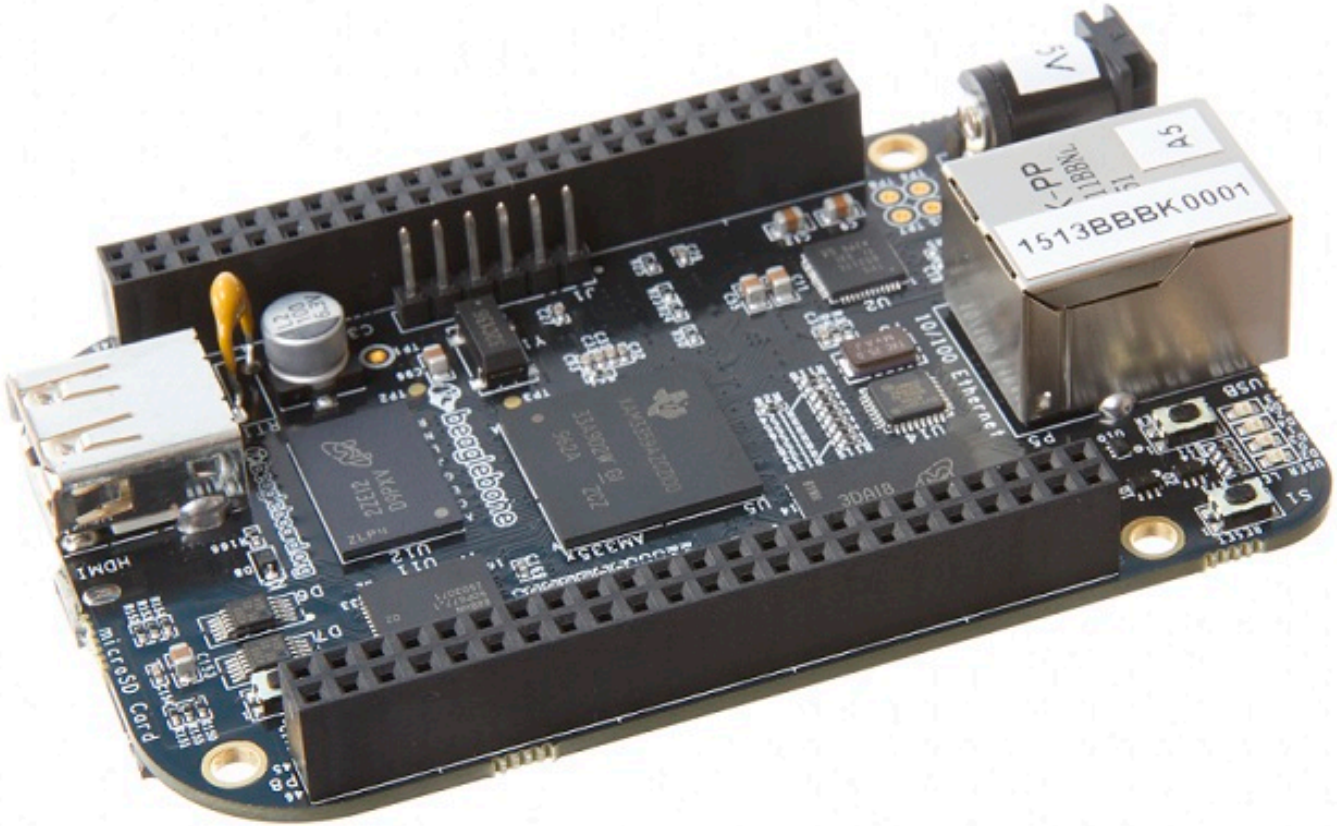
Dauer :

- 1 Laborübung (4 Stunden)

Bericht:

- Lerntagebuch

Zielsystem – Beaglebone Black



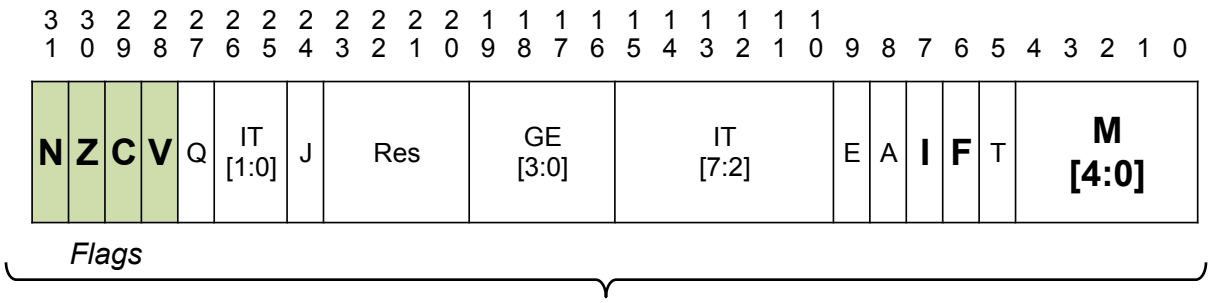
Beaglebone Black - Features

	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	

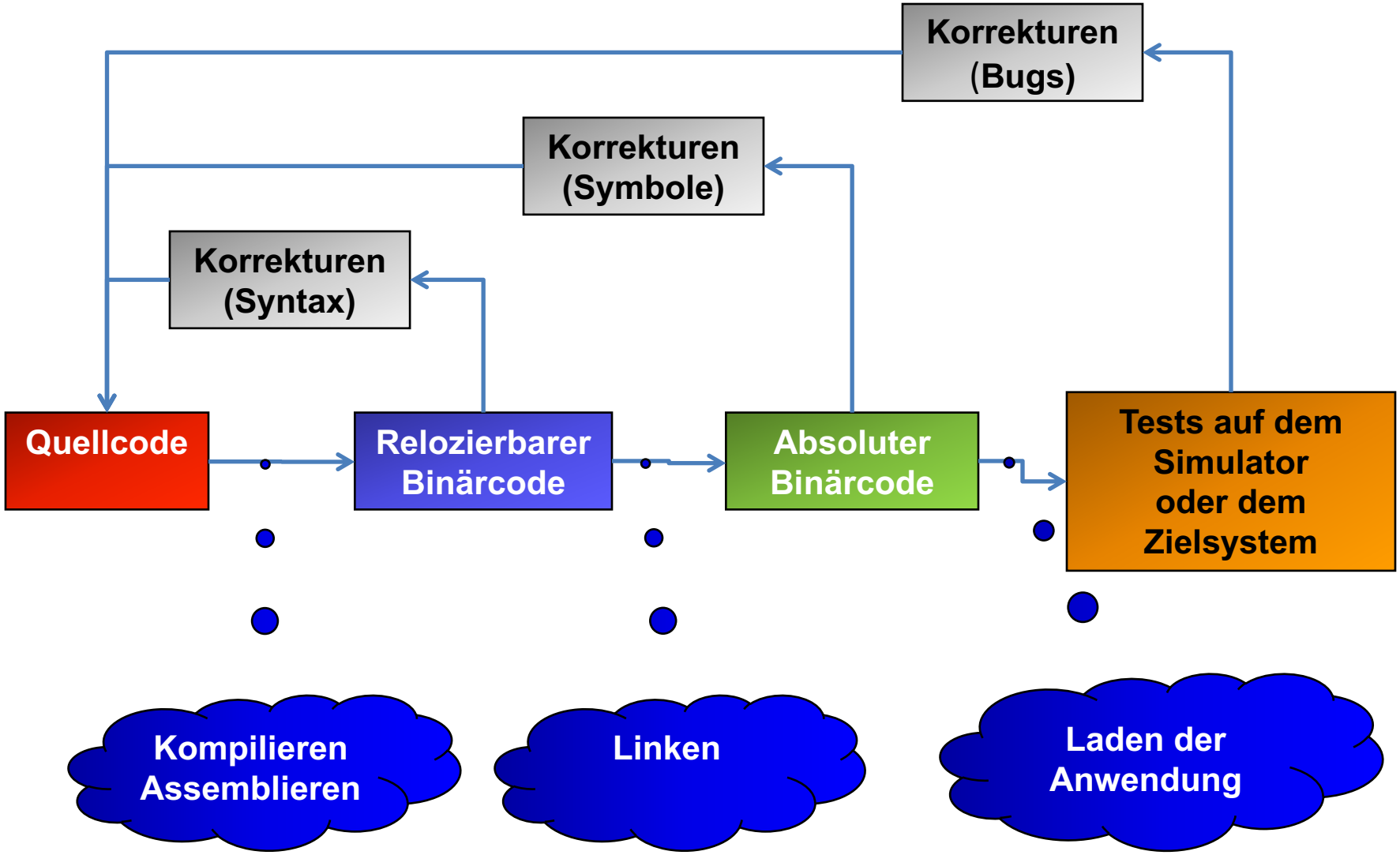
µP Registers

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)



Softwareimplementierungsprozess:



Die Etappen der Softwareentwicklung lassen sich wie folgt einteilen:

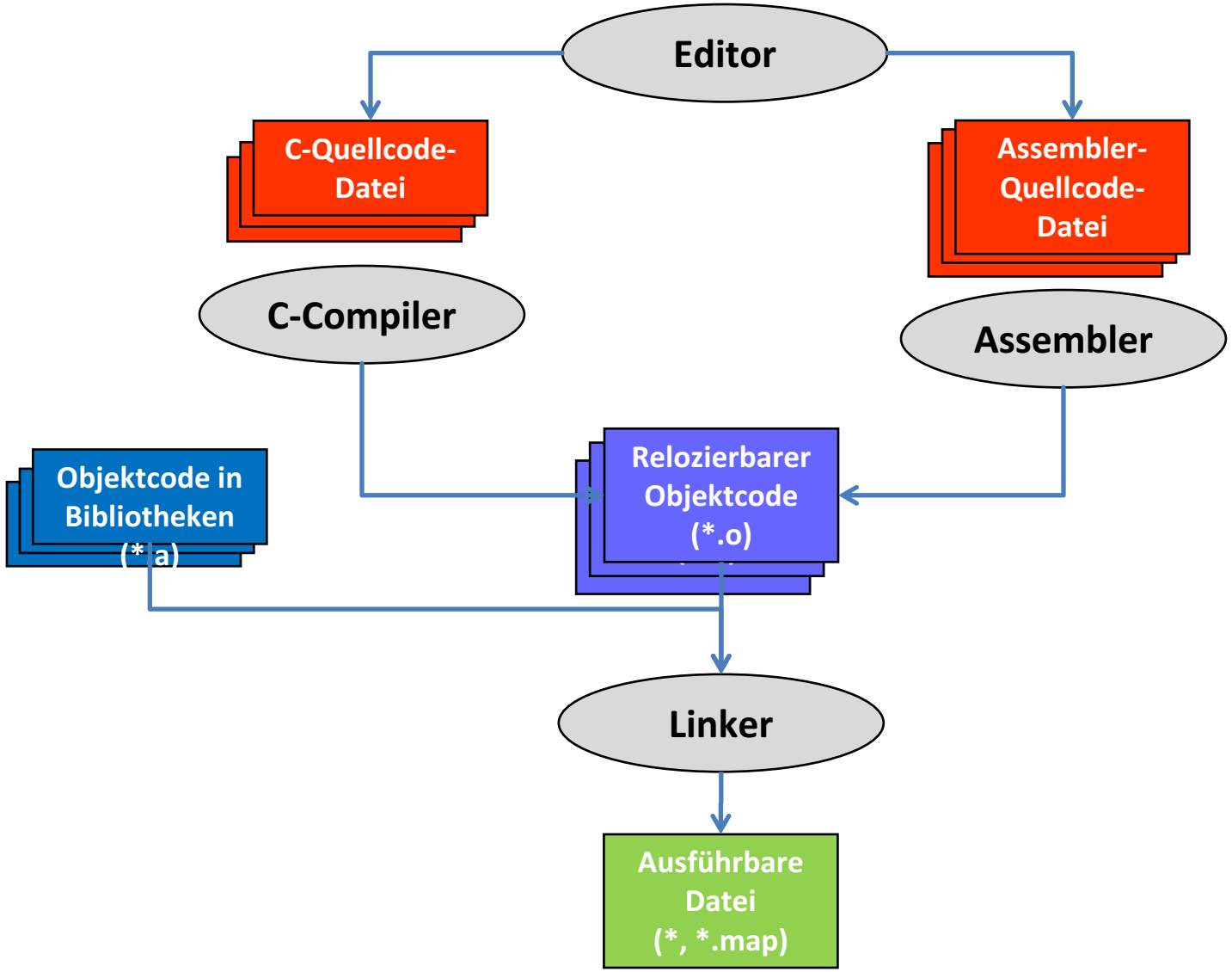
- Der Assembler-, C- oder anderer Code wird mithilfe eines Texteditors in die Quellprogramm-Dateien geschrieben.

- Der Quellcode wird anschliessend kompiliert, um den "relozierbaren" Binärcode zu erhalten, der durch den µP ausgeführt werden kann. Zu diesem Teil des Prozesses gehört selbstverständlich auch eine mit der korrekten Eingabe des Quellcodes (Syntax) verbundenen Korrekturphase.

- Die Binärdatei ist mit der Hardwarearchitektur verbunden. Diese Etappe ist in zwei Phasen unterteilt. In der ersten werden die Verbindungen zwischen den verschiedenen Symbolen
  (z. B. Funktionsaufruf, Adressen der globalen Variablen, ...) gelöst. Wenn Symbole undefiniert bleiben, wird eine Korrekturphase zwingend notwendig. In der zweiten Phase wird der Code entsprechend der Speicherorganisation des Zielsystems reloziert. Abhängig von den Betriebssystemen und der Entwicklungsumgebung kann diese Phase beim Laden des Codes auf das Zielsystem eingefügt werden.

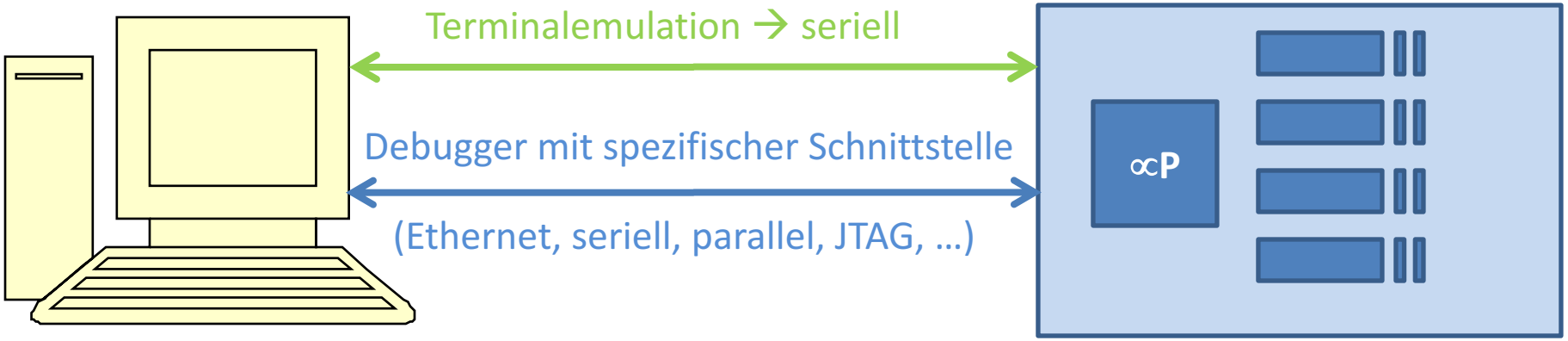
- Der Code wird in den Speicher des Zielsystems geladen (Testsystem oder Simulator). Er wird anschliessend umfassend getestet. In dieser Etappe können wegen Fehlfunktionen des Programms (bugs) eine Reihe von Korrekturen notwendig werden.
  Achtung: Korrekturen aufgrund eines schlechten Konzepts/Designs, die zu diesem Zeitpunkt notwendig werden, können bedeutend sein und viel Zeit und Geld kosten!

Kompilierungs-/Assemblierungskette für embedded systems



Host (host)

Zielsystem (target)



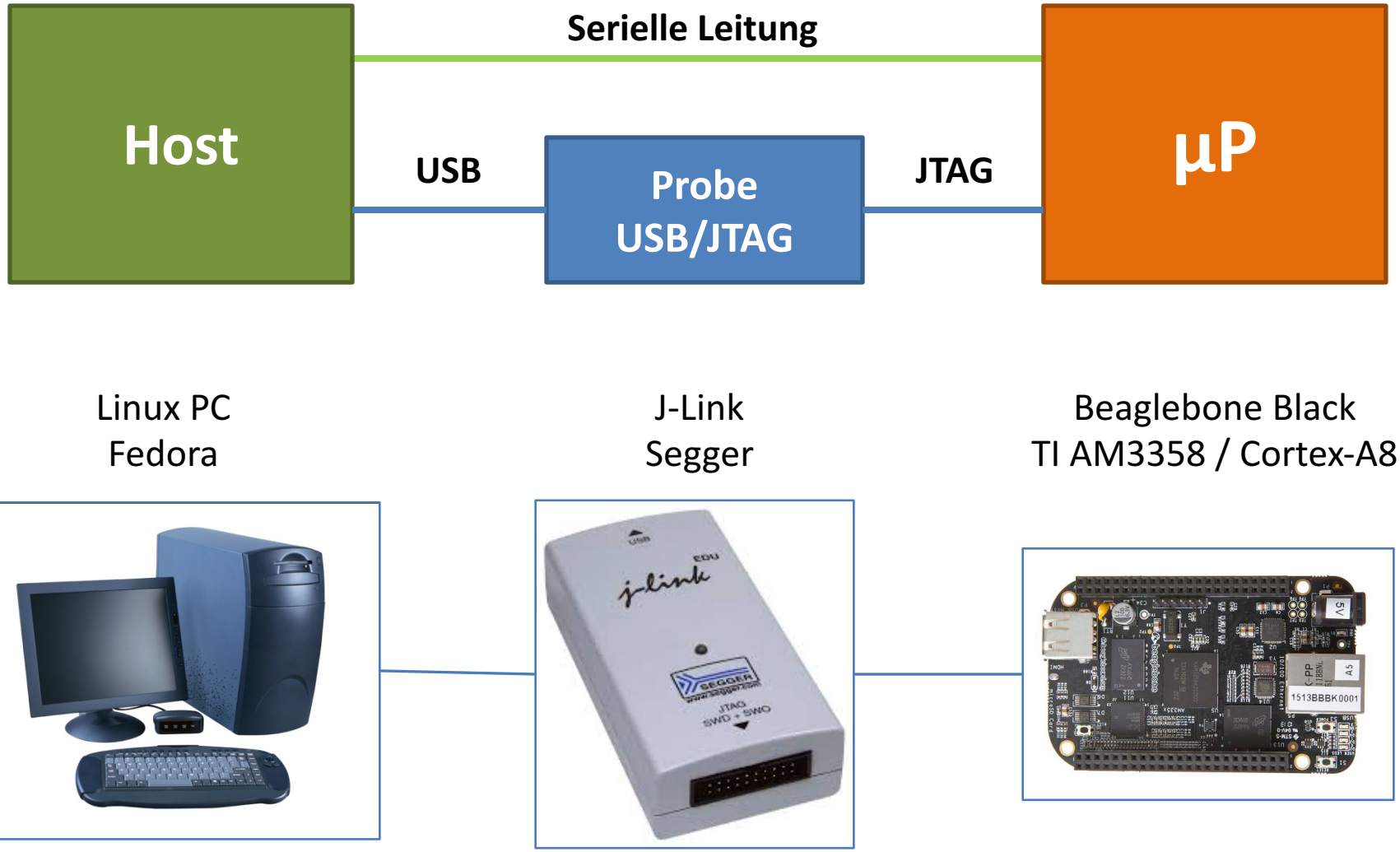
Entwicklung der Anwendung
- Eingabe
- Kompilieren / Assemblieren
- Linken

Debugger
- Quellcode
- Symbole / Adressen

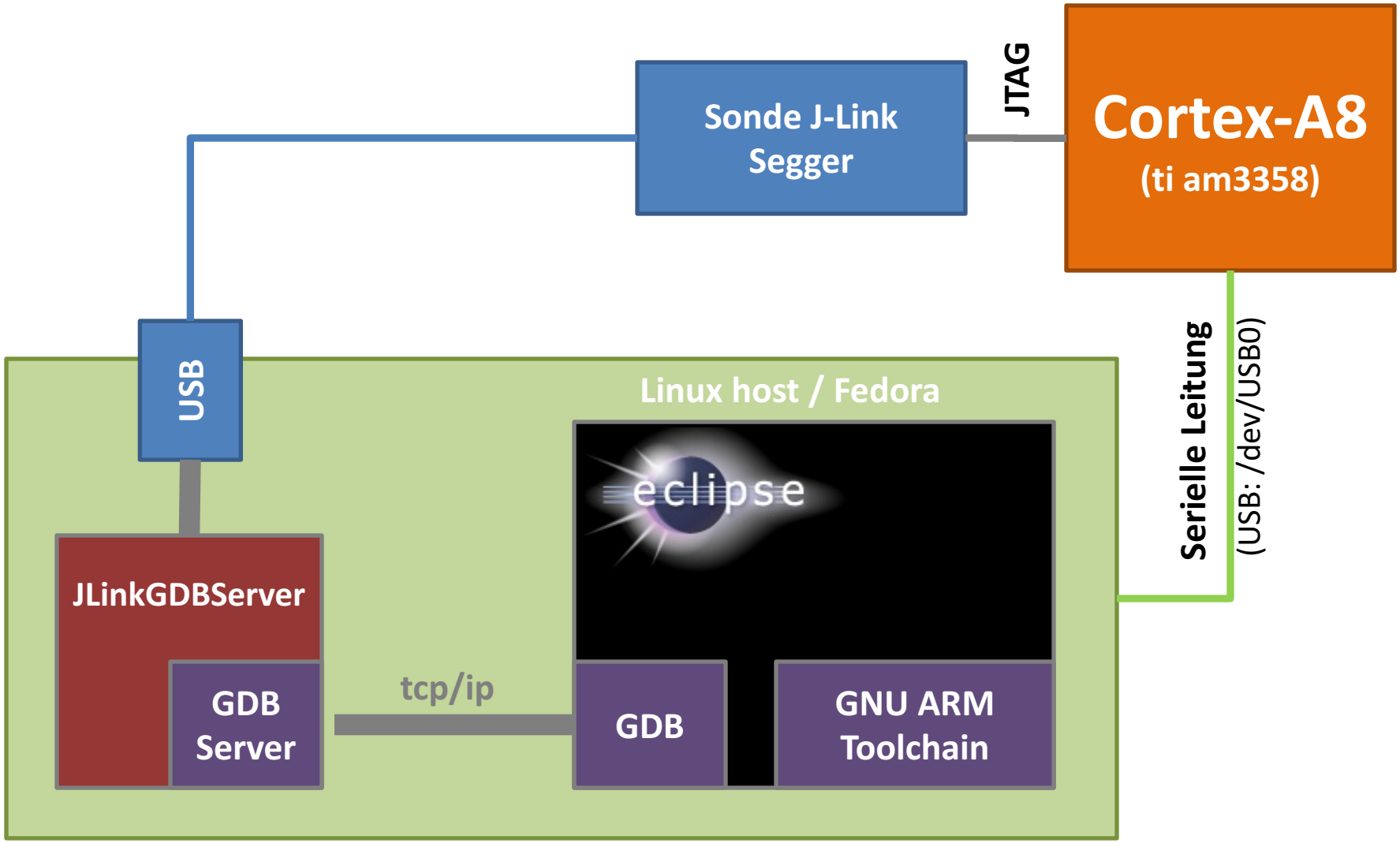
Terminal (optional)
- Befehle
- Tracing / logging

Ausführen der Anwendung
- Binärcode

Debug-Server
- Agent (i/f mit Debugger)


Host
Serielle Leitung
USB
Probe
USB/JTAG
JTAG
µP
Linux PC
Fedora
J-Link
Segger
Beaglebone Black
TI AM3358 / Cortex-A8
j-link
SEGGER
JTAG
SWD + SWO
1513BBBK0001

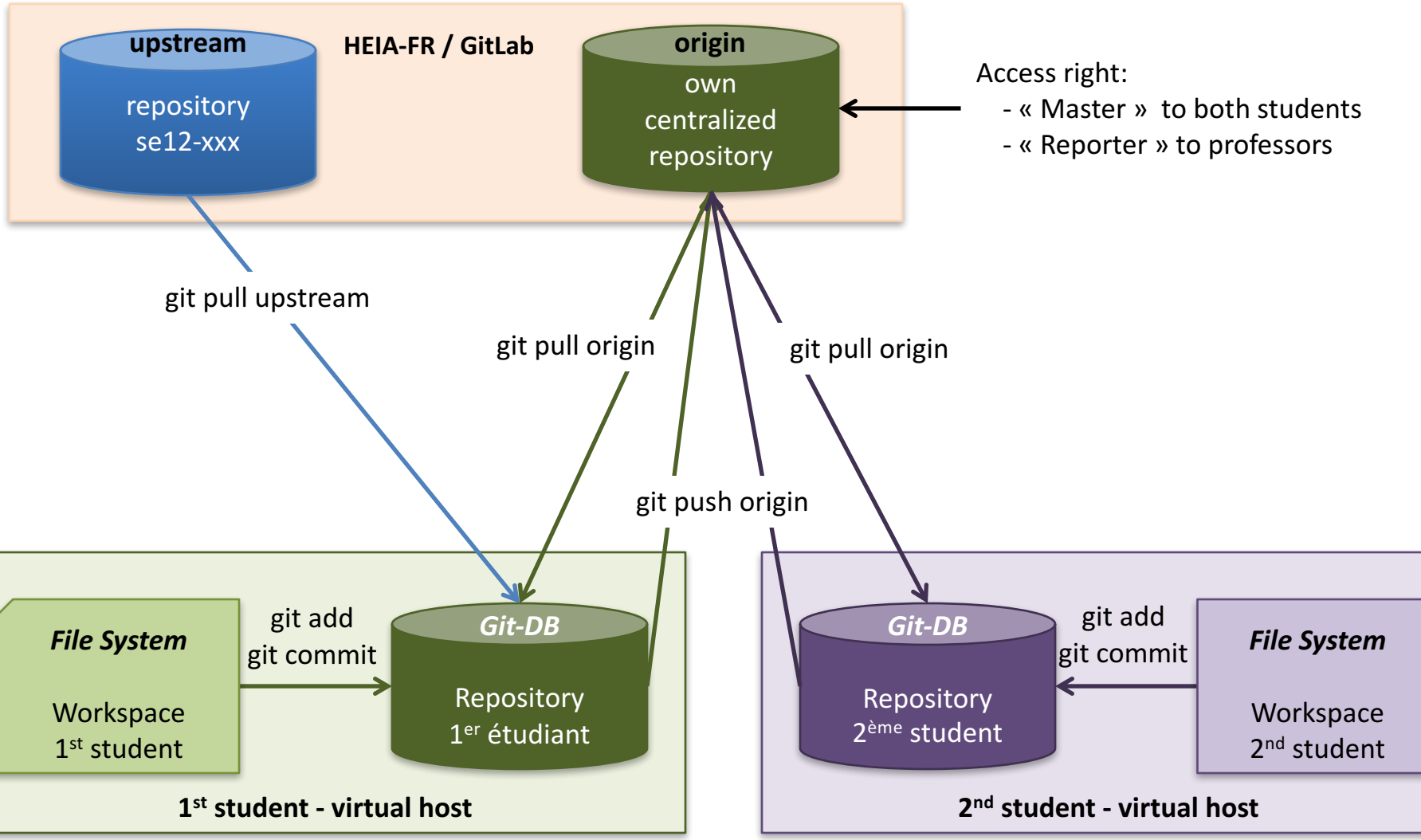
Detaillierte Ansicht der Entwicklungswerkzeugkette



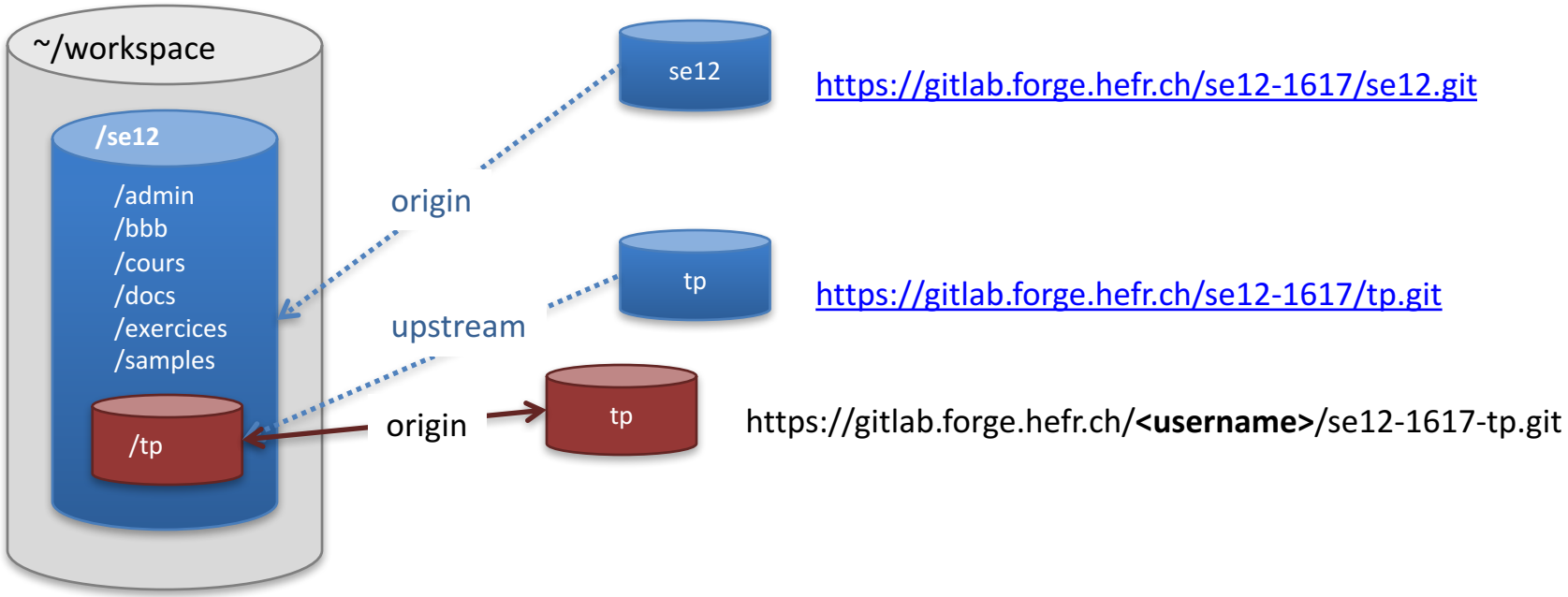
Für die Programmierung der Zielsysteme werden wir in der Kette nur die freien Werkzeuge unter Linux Fedora 24 verwenden, nämlich:

- Eclipse C/C++ 4.6 (*Neon*), Hilfsprogramm für die Projektverwaltung in der Informatik-Entwicklung (IDE – Integrated Development Environment)
- GNU-toolchain, Entwicklungswerkzeuge für Anwendungen, bestehend aus:
  - Binärwerkzeugen (Assembler, Linker, …) Version 2.25
  - C-Compiler 5.2.0
  - GDB-Debugger Version 7.6.2
  - Make Version 4.1
  - Standard Bibliothek newlib 2.2.0-1 von RedHat
- Git wird für die Verwaltung aller Quellcodes verwendet
- J-Link Software V6.00i, Hilfseinrichtung, die über eine USB/JTAG-Schnittstelle erlaubt, die GNU-toolchain mit dem Zielsystem zu verbinden
- Für unser Zielsystem wurde eine spezifische HEIA-FR-Konfiguration erzeugt

Organisation des Arbeitsplatzes und Git-Repositorys



Organisation des Arbeitsplatzes und Git-Repositorys (II)



- **2 Optionen**
  - Option 1
    - Arbeiten mit einer virtuellen VMware-Maschine auf seiner eigenen Maschine
  - Option 2
    - Arbeiten mit seiner persönlichen Linux-Maschine (Fedora 24)

- **Installation der virtuellen Maschine**
  - Option 1 ist bevorzugt, weil es mehr Flexibilität bietet
  - Für die Option 2, die Anweisung auf Backupsslides folgen

- **Benennung des Workspace**
  - Option 1: /home/lmi/workspace/se12
  - Option 2: /home/**<username>**/workspace/se12

Option 1 : Installation einer virtuellen Maschine VMware

Für die Installation von VMware auf der persönlichen Maschine zu befolgende Schritte

- Kopieren Sie den USB-Schlüssel der gewünschten VMware auf die persönliche Maschine
  - *VMware-Fusion-8.xxx-…dmg pour Mac OS X*
  - *VMware-workstation-full-12.xxx-…exe für Windows*
  - *VMware-Workstation-Full-12.xxx-…i386.bundle für Linux*
- Öffnen Sie den Link: **http://hes-so.onthehub.com**
- Loggen Sie sich mit dem Konto HES-SO ein
- Wechseln Sie zu "Students" / "VMware"
- Wählen Sie die für die persönliche Maschine passende Software
- Fügen Sie die Software zum Einkaufskorb hinzu "Add To Cart"
- Führen Sie das Switch-Login "HES-SO - Haute école spécialisée de Suisse occidentale" aus
- Schliessen Sie den Einkauf mit "Check Out" / "Proceed With Order" ab
- Warten Sie auf die E-Mail
- Öffnen Sie die Seite, um die Lizenz zu erhalten
- Installieren Sie die Lizenz

Erstellen und Installation der Umbegung

- **Erstellung des TP-Repositories (1 pro Gruppe von 2 Personen)**
  - Mit einem Browser gehen Sie auf die Git der Schule (https://gitlab.forge.hefr.ch/)
  - Wählen Sie « Projects » und klicken Sie auf « + New Project »
  - Nennen Sie das Projekt « se12-1617-tp » und klicken Sie auf « Create project »
  - Notieren Sie sich die URL des Projekts (https://forge.tic.eia-fr.ch/git/<username>/se12-1617-tp)
  - Fügen Sie den Lehrer als „Reporter“ (mindestens) in das Projekt ein. (https://gitlab.forge.hefr.ch/<username>/se12-tp/project_members)

- **Workspace öffenen**
  - Linux shell öffnen und in der Workspace gehen

```
$ cd ~/workspace/se12/tp
```

- **Git konfigurieren**

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

Erstellen und Installation der Umbegung (II)

- **Lokaler Repository synchronisieren**

```
$ git pull upstream master
```

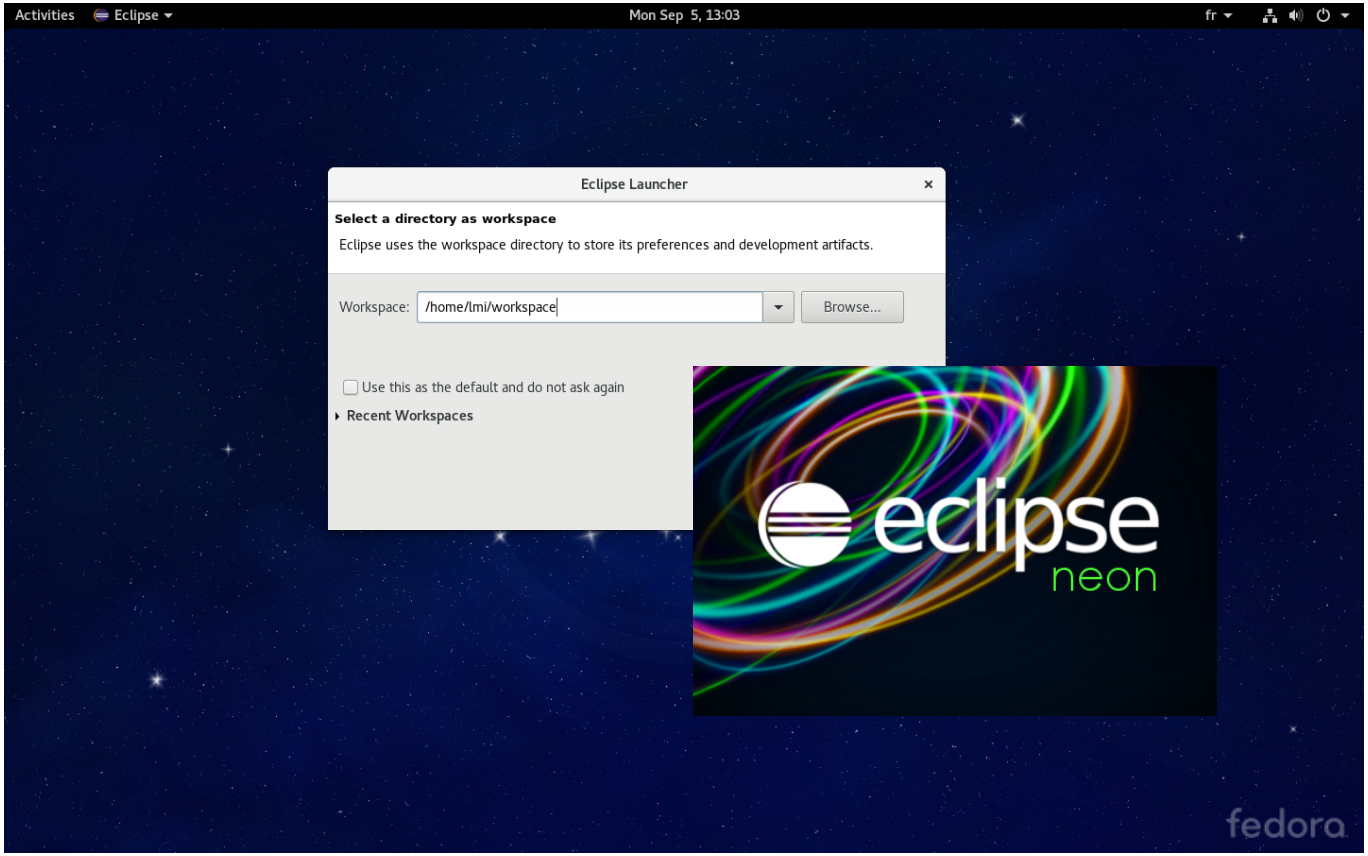
- **Sie die „TPs“ auf dem lokalen Repository einfügen und synchronisieren**

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

$ git push origin master
```

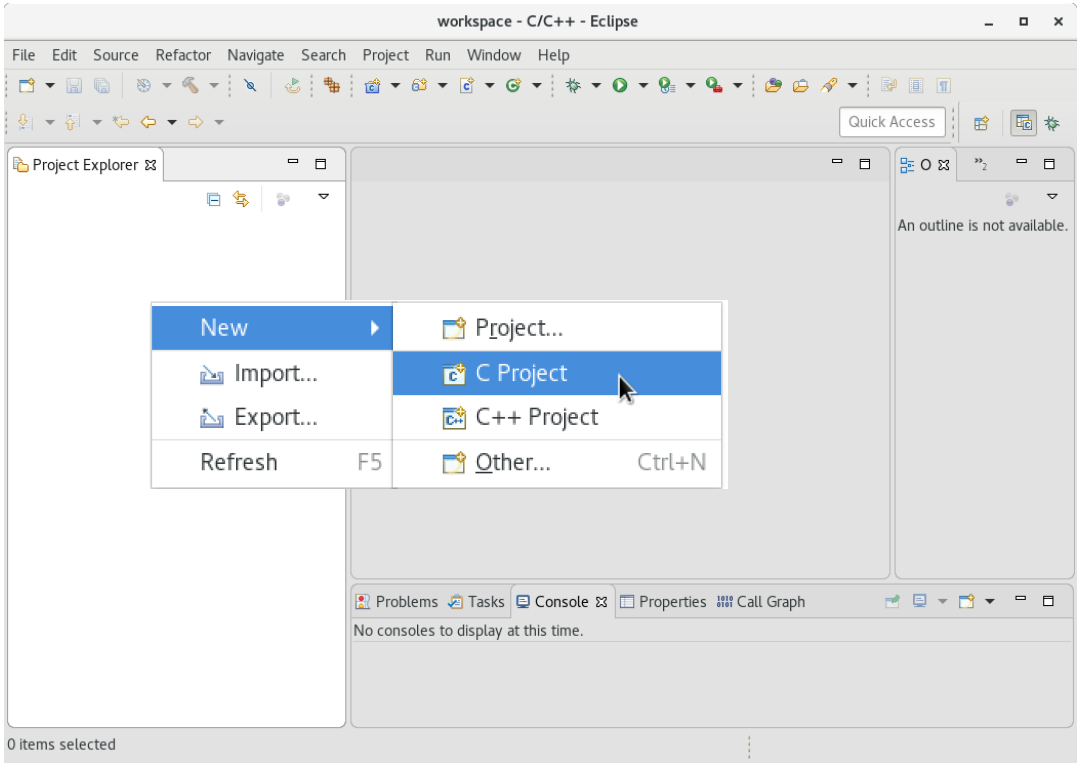
Starten der IDE

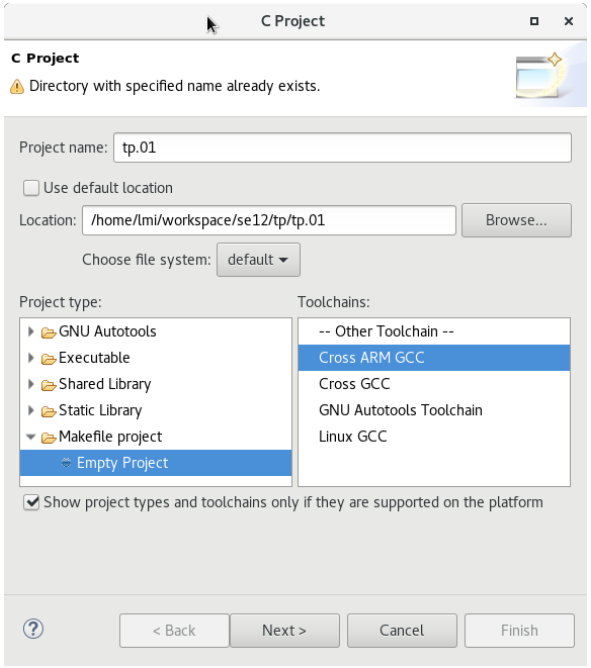
- Starten Sie von Ihrer Linux-Arbeitsstation aus die IDE “*Eclipse*".
- Legen Sie den Workspace in den Ordner:
  - Option 1 (Virtuelle Maschine): /home/lmi/workspace
  - Option 2 (Eigenes Notebook): /home/<**username**>/workspace



Anlegen eines neuen Projekts

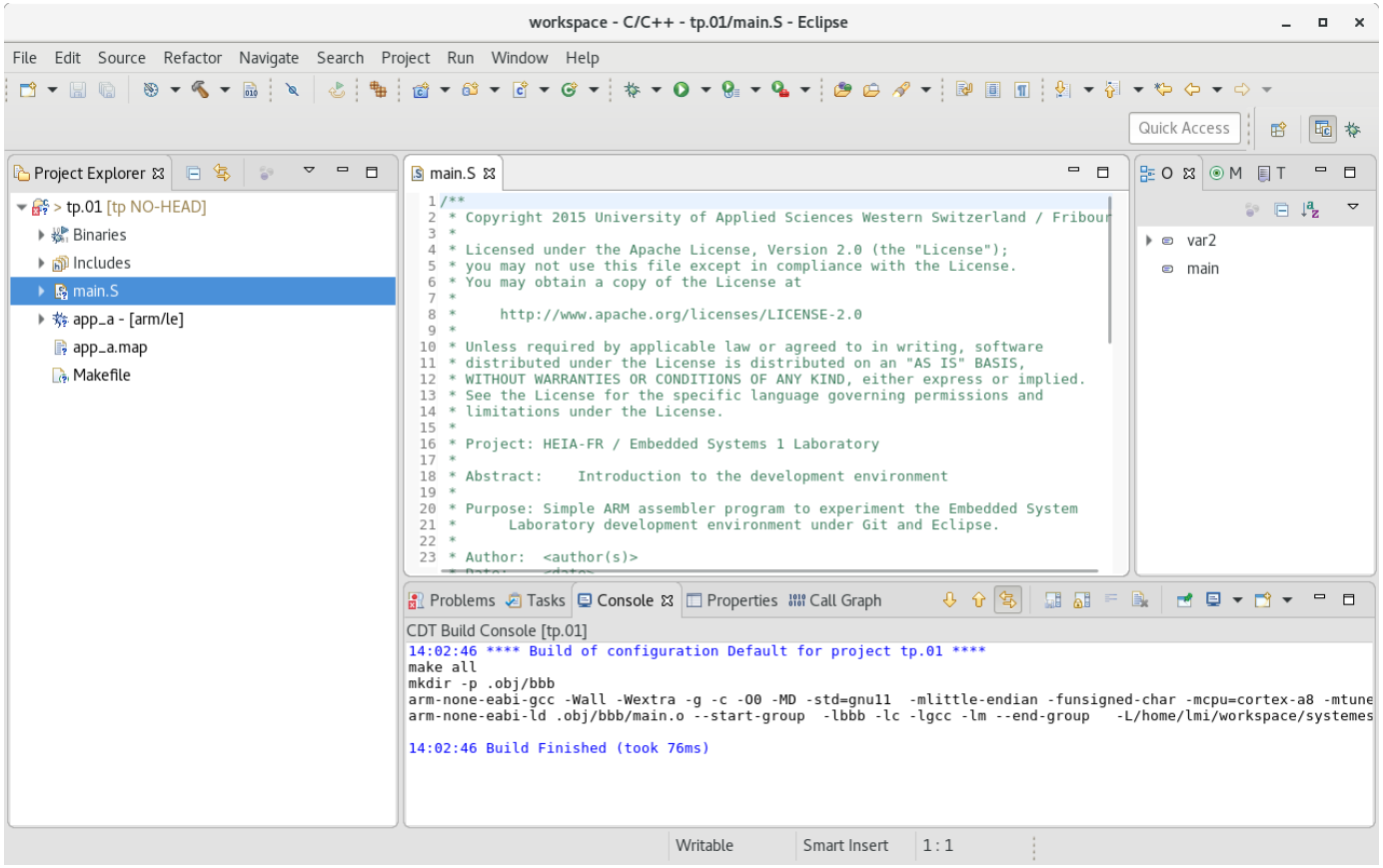
- Wechseln Sie zu „Project Explorer“, klicken Sie auf die rechte Maustaste und wählen „*New → C Project*“
- Wählen Sie den Speicherort des Projekts „*<.../se12/tp/tp.01>*“
- Geben Sie einen Namen ein „*tp.01*“
- Wählen Sie „***Makefile Project → Empty Project → Cross ARM GCC***“
- Klicken Sie auf „*Next*“ → „*Next*“ → „*Finish*“





Erstellen der Anwendung

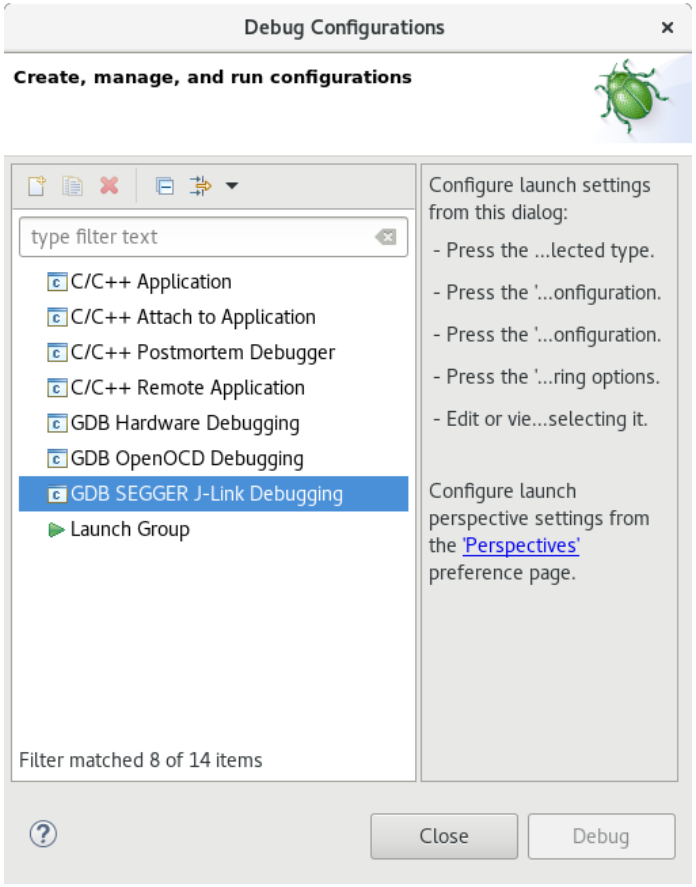
- Klicken Sie auf den Hammer oder geben Sie "CTRL-B" ein.

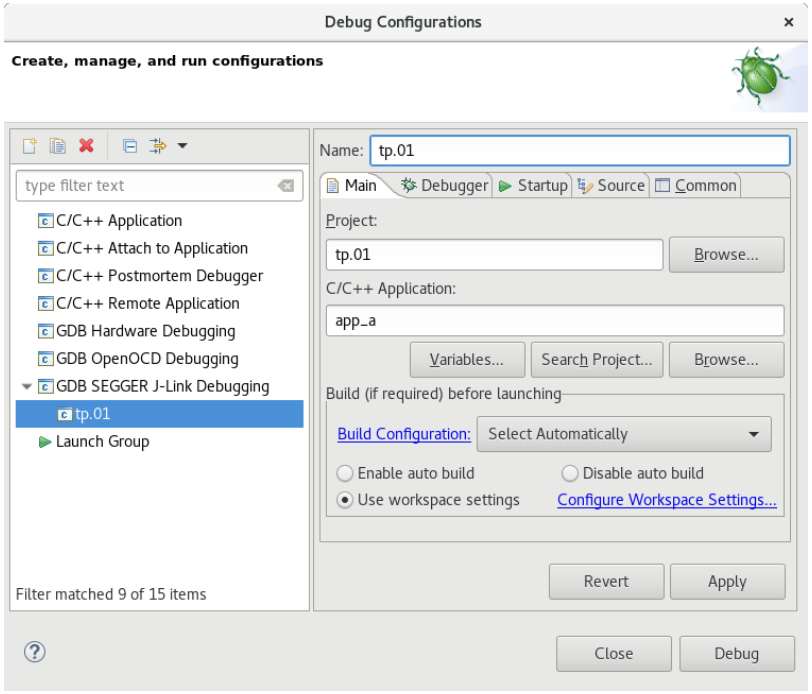


Achtung: Es ist wichtig, sicher zu stellen, dass beim Erstellen Ihrer Anwendung kein Fehler aufgetreten ist!

Konfiguration des Debuggers (I – *main*)

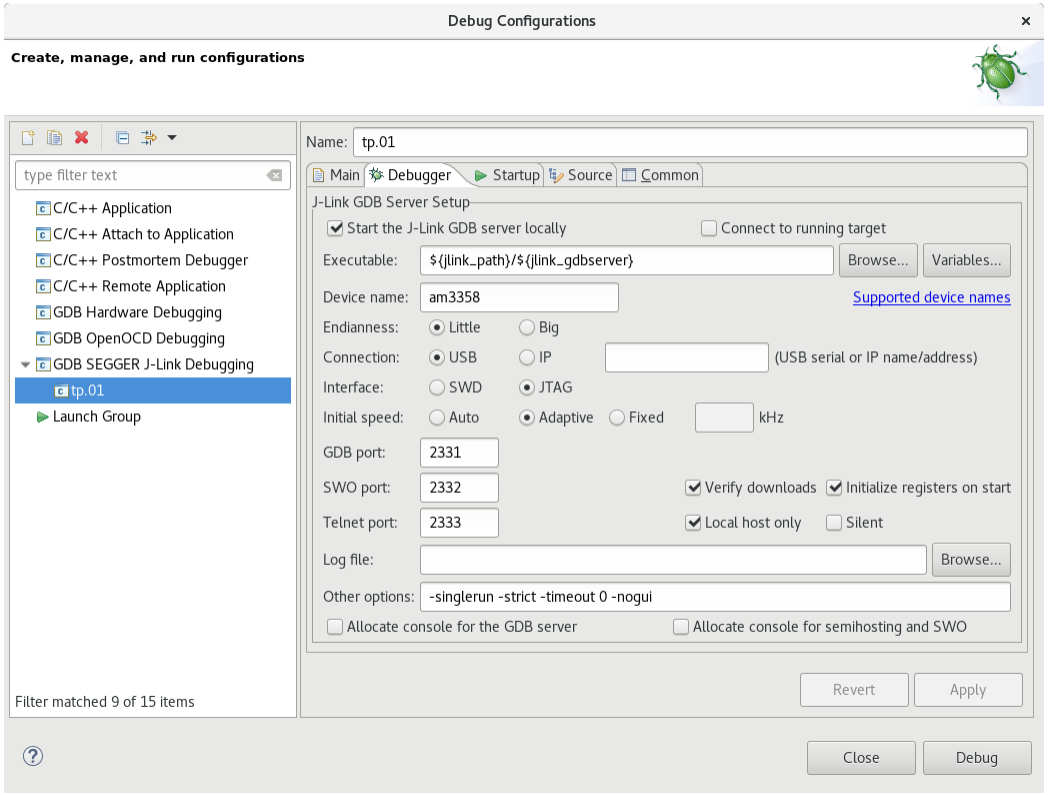
- Klicken Sie auf den kleinen Käfer und wählen Sie „*Debug Configurations*“
- Doppelklicken Sie auf „*GDB SEGGER J-Link Debugging*“
- Wählen Sie das Projekt „*tp.01*“ und die Anwendung „*app_a*“ und nennen Sie es „*tp.01*“





Konfiguration des Debuggers (II – *Debugger*)

- Fügen Sie in „*Device name*“ : „*am3358*“ ein
- Wählen Sie in „*Interface*“ : „*JTAG*“
- Wählen Sie in „*Initial speed*“ : „*Adaptive*“
- Deaktivieren Sie die Option „*Allocate console for the GDB server*“
- Deaktivieren Sie die Option „*Allocate console for semihosting*“



Konfiguration des Debuggers (III – *Startup*)

- Deaktivieren Sie die Option
  - *Initial Reset...*
  - *Enable flash breakpoints.*
  - *Enable semihosting...*

- Wählien Sie in „*JTAG/SWD Speed*“ : „*Adaptive*“

- Fügen Sie folgende Befehle ein:

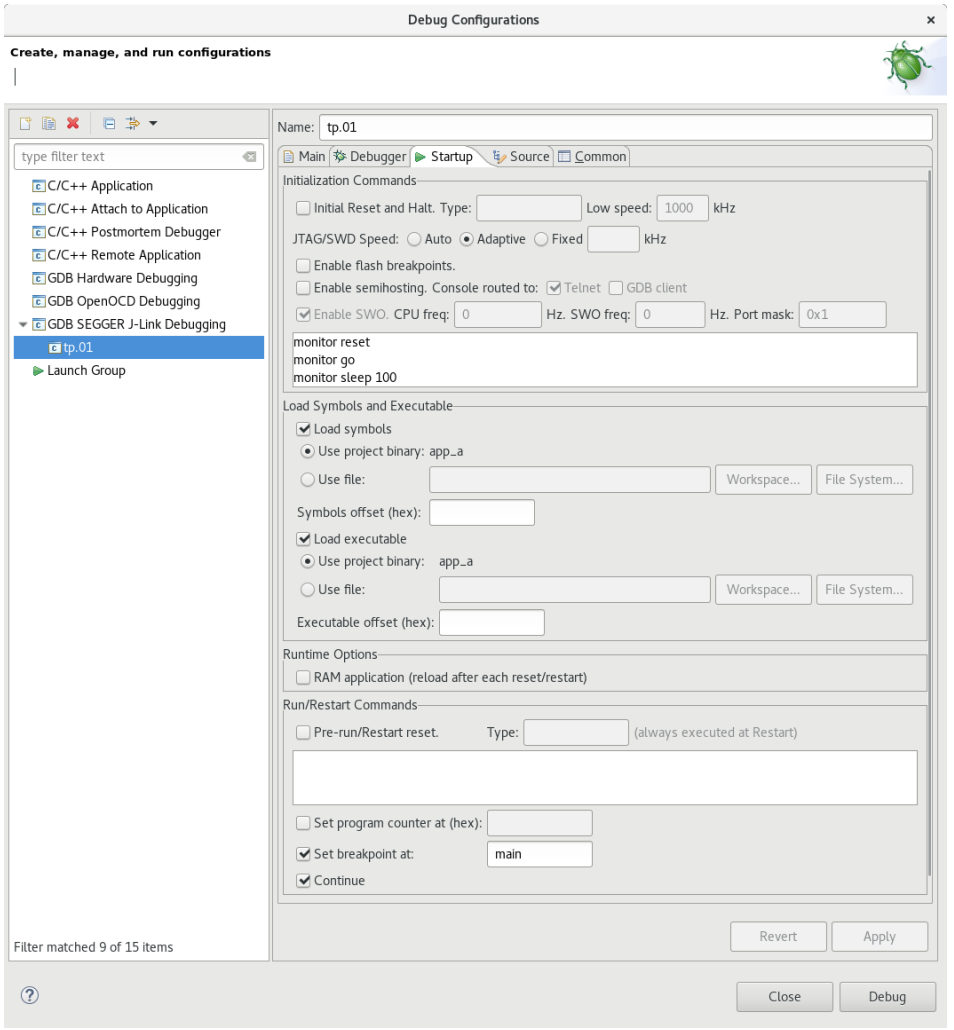
  monitor reset

  monitor go

  monitor sleep 100

  monitor halt

- Deaktivieren Sie die Option
  - *Pre-run/Restart reset.*



Zu realisierende Arbeit

- Geben Sie den folgenden Code in die Datei "*main.S*" ein
- Kompilieren Sie Ihren Code mit Hilfe der Entwicklungsumgebung
- Führen Sie Ihren zuvor in das Zielsystem geladenen Code "schrittweise" aus
- Geben Sie in den Quellcode die Bedeutung jeder Anweisung
- Schreiben Sie der äquivalente Algorithmus in Hochsprache (Java oder C) als Kommentar im Quellencode
- Optimieren Sie das Code aber ohne den Algorithmus zu ändern
- Beantworten Sie die Fragen
- Redigieren Sie Ihr Lerntagebuch
- Übertragen Sie Ihren Code und Ihr Lerntagebuch in Git

Erfordernisse

- Die praktische Arbeit ist von jedem Student durchgeführt werden
- Das Git-Repository einer Gruppe muss den Tag jedes der Gruppenmitglieder enthalten
- Am Ende des Labors, der Workspace auf dem lokalen Computer von jedem der Gruppenmitglieder muss mit dem gemeinsamen Git-Repository der Gruppe synchronisiert werden

Code

```
/** <copyright & heading...> */

// Export public symbols
        .global main, res, incr, i

// Declaration of the constants
#define LOOPS 8

// Declation of initialized variables
        .data
        .align  8
res:    .long   16
incr:   .short  32

// Declation of uninitialized variables
        .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
```

Fragen

- Welche Grösse hat jede der Variablen?
- Welche Grösse hat der Code?
- Wie kann man diese Grösse erhalten?
- Wo befindet sich jede Variable im Speicher (absolute Adresse)?
- Wo befindet sich der Code im Speicher?
- Ist es möglich, diesen Code zu verbessern/optimieren?

- **Die im Labor durchgeführte Arbeit muss in einem Lerntagebuch von 1 bis 2 Seiten zusammengefasst und synthetisiert werden**
  - Kopfdaten (header):
    - Titel der praktischen Arbeit
    - Ort, Datum und Uhrzeit der Laborübung
    - Verfasser / Kurs
  - Arbeitsstunden asserhalb der Schulstunden für dieses Labor
  - Synthese des Studenten auf was er für TP gelernt / geübt hat
    - Nicht erworben
    - Erworben, aber dennoch ausüben
    - Perfekt erworben
  - Antworten auf Fragen
  - Bemerkung / Dinge zu erinnern
  - Feedback zur TP

Lerntagebuch (II)

- **Remarque**
  - Das Lerntagebuch muss als .pdf-Datei eingereicht werden
  - Es muss auf Deutsch verfasst werden
  - Es muss im Git-Repository mit Quellcode gespeichert werden unter
    - Lerntagebuch: *.../se12/tp/tp.01/doc/report.pdf*
    - Quellencode: *.i./se12/tp/tp.01*
- **Abgabetermin**
  - Spätestens um 24.00 Uhr des Datums der besuchten Laborübung

Sichern der Quellen in der Ablage Git

- **Sichern der Änderungen**
  - Öffnen Sie ein Terminal (eine Linux-Shell)
  - Sehen Sie sich den Status der Ablage an
    ```
    $ git status
    ```
  - Fügen Sie eventuell neue Dateien hinzu
    ```
    $ git add *
    ```
  - Führen Sie die Änderungen aus
    ```
    $ git commit -a –m “comment…”
    ```
- **Synchronisieren Sie die lokale Ablage mit der zentralen Ablage (Server)**
  - Synchronisieren mit der Ablage des Kurses
    ```
    $ git pull upstream master
    ```
  - Synchronisieren mit der persönlicher Ablage
    ```
    $ git pull origin master
    ```
- **Sichern Sie das Ergebnis der Arbeit in der zentralen Ablage (Server)**
  - Laden Sie den Pfad in die Ablage hoch
    ```
    $ git push origin master
    ```

Backup-Folien

Option 2

Installation der Umgebung auf einer persönlichen Linux-Maschine

Option 2: Installation der Umgebung auf personnell Linux Maschinen

- **Erstellen des Arbeitsbereichs (workspace)**
  - Öffnen Sie ein Terminal (eine Linux-Shell) und erzeugen Sie einen Arbeitsordner (workspace)

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

- **Konfigurieren Sie GIT**

```
$ git config --global user.name <Firstname Lastname>
$ git config --global user.email <user.name@edu.hefr.ch>
```

- **Erstellen Sie eine Kopie des GIT-Repository mit den Slides des Kurses**

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

- **Installieren Sie die Umgebung und erzeugen Sie die Bibliotheken für die Zielsystem**

```
$ ~/workspace/se12/bbb/scripts/install_se12_environment
```

- **Erstellen Sie eine Kopie des GIT-Repository mit dem Quellcode des Kurses**

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

- **Installieren Sie das Plugin 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 Developpment 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