# Laboration 1a

**Syfte med labben:**

Studenterna ska använda moderna utvecklingshjälpmedel för programutveckling för inbyggda system, som IDE (Integrated Development Environment), utvecklingskort och versionshanteringssystem.

Labbarna görs normalt två och två, men det går bra att jobba ensam också.

Det rekommenderas starkt att ha ett eget USB-minne för sitt personliga repo, även om det inte är nödvändigt, all kod du gjort i labben kommer att finnas ”i molnet” när du är klar.

**Krav för godkänt:**

Ha genomfört följande fyra labbmoment enligt denna handledning:

1. Klona det centrala git-repot till din lokala dator och skapa en personlig branch för denna och framtida laborationer och programmeringsuppgifter i kursen
2. Bygga en enkel labbsetup med kretskort och lysdiod.
3. Starta Atmel Studio, kopiera en källkodsfil från repot till din egen katalog och modifiera, kompilera och ladda ner den till utvecklingskortet och köra programmet. Visa resultatet för labbhandledaren.
4. Committa filerna till ditt personliga repo, och pusha det som en branch till det centrala kursrepot.

**Väl godkänt:**

Ovanstående avklarat på ordinarie laborationstillfälle.

**Lämplig förbreredelse**

Labben kommer att innehålla grundläggande C-programmering, så repetera vad som gåtts igenom i kursen datateknik från första året.

Läs också gärna igenom grunderna i versionshantering med git, t.ex. denna självstudiekurs: <http://try.github.io/levels/1/challenges/1>

Det är också bra om du läst igenom *hela* labbhandledningen innan du börjar!

**Krav för att kunna starta laborationen**

Om annan dator än labbsalens dator används (t.ex. egen dator) måste följande program finnas installerat på den datorn:

* Atmel Studio 6.1 (finns endast för Windows)
* Bossac flashprogrammeringsdriver, inklusive DOS-macro
* En Git-klient, t.ex. Github client[[1]](#footnote-1) eller TortoiseGit[[2]](#footnote-2) (för andra operativsystem än Windows finns andra Git-klienter)

Alla studenter i labbgruppen måste också ha registrerat en användare på github[[3]](#footnote-3) (om du redan har en, använd den)!

Meddela Ulrik på [ulrik.eklund@mah.se](mailto:ulrik.eklund@mah.se) vilka era githubanvändare är.

## Versionshantering med Git

Grunderna hur man arbetar med git-klienten för Windows finns på

<https://help.github.com/articles/getting-started-with-github-for-windows>

<https://help.github.com/articles/synchronizing-repositories>

## Skapa ett personligt repo för programmering med hjälp av Git

Det finns ett centralt repo för kursen på github med adressen:

<https://github.com/MalmoUniversity-DA139A/HT13.git>

Ange detta som ”primary remote repository (origin)” i er git-client under ”Settings”.

För att fortsätta jobba och ändå ha kontroll på vem som gör vad så låter vi Git hålla reda på det för alla studenter och lärare i kursen, vilket är en av Gits styrkor (och varför det används i projekt med hundratals utvecklare, t.ex. open-source-projektet Linux).

1. Välj en katalog där du vill ha din personliga kopia av kursens repot (lämpligtvis på en USB-sticka så du kan jobba flytta det lätt mellan datorerna i labsalen och din egna dator). Vi kommer att jobba mot samma repo i hela kursen. Du kan döpa katalogen till vad du vill, men i det centrala repot heter den HT13.
2. Att skapa en personlig kopia av det centrala repot kallas att klona ett git-repo. Klona kursens centrala git-repo till din valda katalog med hjälp av github-klienten[[4]](#footnote-4). Under kursens gång kommer du att synka dina ändringar med det centrala repot.
3. Checka ut en ny branch[[5]](#footnote-5) så att du kan jobba i och uppdatera denna branch oberoende av vad dina kurskamrater gör. Detta görs under branches i git-klienten, klicka på plusset till höger om Master.  
   Branchen namnges med StudentNN, t.ex. Student13.

Under labben är det enklast om man jobbar mot en och samma katalog och branch i hela labben, även om det tillhör en specifik student. Det finns instruktioner i slutet av handledningen hur man delar resulterande kod-filer från labben till alla studenterna i labbgruppen.

## Sätta upp arbetskatalogen för denna labben

*I resten av denna labbhandledning förutsätts det att ni jobbar mot er egen student-branch och inte gör några ändringar i origin- eller master-branch.*

I mappen HT13\CourseMaterial\Lab1a finns denna labhandledning och en projektmapp för Atmel studio vid namn Lab1a\_project.

Skapa en egen användarkatalog i git-repot, t.ex. StudentNN, och skapa en underkatalog för denna labben i filutforskaren i Windows[[6]](#footnote-6)

Kopiera katalogen med labbprojektet till din nyskapade katalog för labben, från HT13\CourseMaterial\Lab1a\Lab1a\_project\ till HT13\StudentNN\Lab1a\Lab1a\_project\

*Visa upp vad du har gjort för labbhandledaren så har du klarat av punkt 1 i listan för godkänt!*

## Använda Git

Med jämna mellanrum vill man committa sina ändringar till sitt lokala repo (i sin lokala branch)[[7]](#footnote-7). Varje commit bör vara en ”unit of functionality” som alltid kompilerar, t.ex. refaktorera namnet på en funktion, en ny feature, eller förbättrad dokumentation. Varje commit måste också ha en beskrivande kommentar så man vet vad det faktiskt var man ändrade[[8]](#footnote-8).

Git sparar alla tidigare commits, så man kan närsomhelst backa till ett tidigare commit-tillfälle om det skulle behövas.

Om en fil aldrig har blivit committad tidigare måste man ange att den ska adderas till Git-repot. Efter man gjort det första gången håller Git reda på att filen ”ingår” i repot (men den måste committas första gången ändå!). Git-klienterna burkar hålla reda på nyskapade filer i katalogerna som tillhör repot och frågar om de ska adderas.

## Bygga labbsetup

Utrustning: Kopplingsdäck

Utvecklingskort Arduino Due

Lysdiod

Vi ska driva lysdioden från en av de digitala utgångarna på Due-kortet, nämligen utgång 22.

Vilken I/O-port på processorn driver denna utgång?[[9]](#footnote-9) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Due-kortet använder sig av 3,3 V som arbetsspänning

Hur stor ström orkar den digitala utgången på Due-kortet driva (se datablad[[10]](#footnote-10))? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

För att få en lämplig ström genom lysdioden så seriekopplas den med ett motstånd

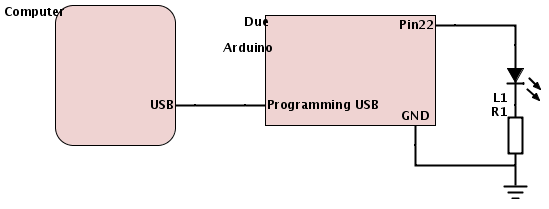
Lysdioden har en framfallsspänning på ca 2,0 V. Vad blir i så fall spänningen över motståndet? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Vilken storlek bör då motståndet ha om strömmen genom motståndet (och lysdioden) ska *högst* bli vad pinnen på kortet orkar driva (Ohms lag)?[[11]](#footnote-11) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Klarar ljusdioden av den strömmen?

När man konstruerar inbyggdasystem så måste man gör en avvägning så att båda dessa villkor är uppfyllda.

Koppla upp lysdiod och valt motstånd på en kopplingsbräda så att den drivs att rätt utgång på Due-kortet och kopplas till GND.



Due-kortet kan drivas genom en USB-kabel från datorn om man inte behöver för mycket effekt till utgångarna och det räcker gott för denna labben. USB-kabeln kopplas till USB-micro-kontakten märkt PROGRAMMING.

*Visa dina uträkningar och kopplingen för labbhandledaren innan du kopplar kortet till datorn via USB-kabeln! Nu har du klarat av punkt 2 i listan för godkänt.*

## Labbens programmeringsuppgift

Om du lyckats med att klona det centrala git-repot och kopiera rätt filer finns nu i katalogen HT13\StudentNN\Lab1a\Lab1a\_project\ en projektfil för Atmel Studio: Lab1a\_project.cproj

Öppna den genom antingen genom att dubbelklicka eller genom ”open project” inifrån Atmel Studio.

Labben går ut på att ni ska göra om existerande kod till att göra en snarliknande uppgift. Något som är betydligt vanligare än att mans skriver all kod på ett vitt papper.

All kod som är relevant för labben finns i main.c

Testa att kompilera projektet under menyn ”Build”. Det här kolla främst att syntaxen är rätt, variabler är definerade och lite annat. C har annars få begräsningar vad som är rätt eller fel i språket.

Du kommer att få två felmeddelanden och ett antal varningar. Varningarna är ”bara” för att en del variabler är definierade som aldrig används (i den här labben), så vi bryr oss inte om dem.

Felmeddelandena måste däremot åtgärdas, och de beror på att datatyperna som behövs för 32-bitarsregisterna inte är definierade. Läser du koden i main.c noga ser du att den raden som inkluderar inttypes.h är bortkommenterad. Avkommentera och till att den läses av kompilatorn och kompilera på nytt.

Bygg programmet under ”Build”-menyn.

Ladda ner den färdiga binärfilen till Due-kortet via USB-kabeln med kommandot ”BossacArduinoDue(Debug)” under Tools-menyn[[12]](#footnote-12).

När programmering är klar ska lysdioden blinka regelbundet. Om den inte gör det är det dags att börja felsöka och förstå vad som gått fel.

Om allt funkar som det ska kan det vara dags att göra en commit i din branch mot ditt personliga repo.

### Skapa separata filer för definitioner och funktioner

För att begränsa programmeringen i labben till bara just de ställen det behövs ska du klippa bort de definitioner och funktioner som inte kommer att behöva ändras och skapa en egen fil med dem.

Börja med att skapa en header-fil ( som får ändelsen .h) genom att ställa dig i src-katalogen i ”Solution Explorer”-fönstret inuti Atmel Studio. Högerklicka och välj Add/New Item…

Atmel kallar header-filer för ”Include File”. Glöm inte att döpa filen till något meningsfullt, på formen abc.h längst ner, innan du trycker på Add-knappen.

Nu ska du klippa ur alla deklarationer ur main.c och in i filen du nyss skapat. Det står kommenterat var du ska klippa om du är osäker.

Eftersom deklarationera fortfarande behövs i main-programmet så måste du tala om för kompilatorn var de finns nu genom att inkludera din nyskapade header-fil på samma ställe i main.c som du klippte ur dem:

#include ”abc.h”

Testa att kompilatorn fortfarande tycker att alla definitioner fortfarande stämmer innan du fortsätter.

Nu ska du göra samma sak för funktionerna efter main(). Skapa en ”C File” i src-katalogen inifrån Atmel Studio genom att högerklicka och ”Add/New Item…”. Döp den till samma sak som dinh-fil ovan (om header-filen heter abc.h skall kod-filen heter abc.c, det är praxis i all C-programmering).

Klipp ur funktionerna som definieras efterhuvudprogrammet main()

void PIOB\_init(int PinNumber) och

register\_data pin(int n)

ur main.c och klistra in dem in din nya c-fil.

Eftersom dessa två funktionera också använder sig av deklarationerna i headerfilen måste den också inkluderas med #include

Bygg hela projektet. Om det funkar utan felmeddelande kan du flasha ner programmet till Due-kortet och se att lysdioden fortfarande blinkar likadant.

Om allt funkar som det ska kan det vara dags att göra en commit igen i din branch mot ditt personliga repo.

### Ändra hur lysdioden blinkar

Nu är uppgiften att ändra det regelbundna blinkandet av lysdioden så att den istället blinkar morsekoden för SOS, tre korta, tre långa, tre korta blink.

Blinkningarna ska uppfylla följande krav:

* Ett långt blink ska vara tre gånger så långt som ett kort blink.
* Pausen mellan varje blink inom en bokstav ska vara lika lång som ett kort blink
* Pausen mellan varje bokstav ska vara lika lång som ett långt blink.
* Pausen mellan varje SOS ska vara lika lång som fem korta blink utan paus.

Du behöver endast ändra koden som finns i kvar i main.c för att lyckas med ovanstående!

Det kan vara bra att regelbundet göra en commit i din branch mot ditt personliga repo när du har något som funkar, t.ex. den första bokstaven. På så sätt kan du alltid backa tillbaka till en fungerande version om något skulle gå fel.

*När du är klar med SOS visar du setupen med din blinkande diod för labbhandledaren. Nu har du klarat av punkt 3 i listan för godkänt. Glöm inte att committa det färdiga programmet till din personliga branch (som du ju jobbat i hela tiden).*

## Lagra arbete i det centrala kursrepot

När man är klar med labben eller inlämningsuppgiften så kan man dela arbetet man gjort i en branch med de andra studenterna i labgruppen.

Först skapa en katalog där den andra studenten vill ha sitt repo. Sen finns det två sätt:

1. Den andra studenten klonar den första studentens repo direkt till sin önskade katalog. Gör det som behöver göras. Sen pushar båda sina resepktive brancher till repot på Github
2. Den första studenten pushar sitt repo som sin branch till github. En annan student klonar repot på github till sin katalog, och gör vad som behöver göras innan den committar till repot på github.

### 1) Klona direkt från ett student-repo

Den här metoden är den som kanske är elegantast, men är lite svår att göra från github-klienten: Däremot går den alldeles utmärkt att göra från kommandoraden i Git-shell.

Du behöver inte vara speciellt orolig att göra fel. Git sparar i pricnip allt som händer och om man har committat regelbundet kan man alltid återskapa alla filer som fanns vid sin senaste commit.

1. Flytta dig till katalogen där student 2 vill ha sitt repo

>cd ”repo-katalog hos student 2”

1. Klona student 1s repo

>git clone ”repo-katalog hos student1”

1. Skapa en ny branch för student 2

>git checkout –b Student02

1. Skapa en mapp HT13/Student02/Lab1a/

>mkdir HT13/Student02/Lab1a/

1. Ställ dig i den nya katalogen

>cd Student02/Lab1a/

1. Kopiera filerna filerna i HT13/Student01/Lab1a/ till katalogen du står i

>cp -r HT13/Student01/Lab1a/\* .

1. Committa den nya versionen

>git commit –am ”Lab 1 completed”

1. Tryck den lokala branchen för student 2 till det centrala git-repot

>git push <https://github.com/MalmoUniversity-DA139A/HT13.git> Student02

### 2) Klona via github

Den här metoden kräver att student 1 redan har committat sin branch till github.

1. Student 2 klonar repot till sin valda lokala katalog precis om Student 1 gjorde första gången.
2. Student 2 skapar sen en ny branch, lämpligtvis med namnet Student02.
3. Sen mergar Student 2 branchen som student 1 skapat in i sin egen branch.
4. Därefter skapar Student 2 en ny katalog för sitt arbete, HT13/Student02/Lab1a och kopierar filerna i labben från student 1s katalog till sin egen katalog.
5. Sen committar student 2 sina ändringar och pushar sin branch tillbaka till repot på github.

Nu finns alltså det två brancher, en vardera för student 1 och 2.

*Säg till labbhandledaren så kollar den om brancherna finns på github. Du har nu klarat av punkt 4 i listan för godkänt.*

1. <http://windows.github.com/> [↑](#footnote-ref-1)
2. <http://code.google.com/p/tortoisegit/> [↑](#footnote-ref-2)
3. <http://www.github.com/> [↑](#footnote-ref-3)
4. Kommandoraden i Unix eller gitbash/git-shell om man står i önskad katalog:  
   >git clone https://github.com/MalmoUniversity-DA139A/HT13.git [↑](#footnote-ref-4)
5. Git-shell-kommando: >git checkout –b StudentNN [↑](#footnote-ref-5)
6. Git-shell-kommando: >mkdir HT13\StudentNN\Lab1a\ [↑](#footnote-ref-6)
7. Git-shell-kommando: >git commit –am ”mywork” [↑](#footnote-ref-7)
8. Ett exempel: [https://github.com/akka/akka/commit/adfeb2c1f07153b7eec11705fda956f62b1bbb04](https://www.google.com/url?q=https%3A%2F%2Fgithub.com%2Fakka%2Fakka%2Fcommit%2Fadfeb2c1f07153b7eec11705fda956f62b1bbb04&sa=D&sntz=1&usg=AFQjCNEAoVw-89UGUd04-j6H8_3gTtYjYw" \t "_blank) [↑](#footnote-ref-8)
9. Se <http://arduino.cc/en/Hacking/PinMappingSAM3X> [↑](#footnote-ref-9)
10. <http://arduino.cc/en/Main/arduinoBoardDue> [↑](#footnote-ref-10)
11. Repetera grundläggande elektronik på <http://sunburst.usd.edu/~schieber/psyc770/transistors101.html> om du int ekommer ihåg. [↑](#footnote-ref-11)
12. Finns inte det kommandot i menyn följ instruktionerna här: <https://github.com/ctbenergy/BossacArduinoDue>

    Både bossac-programmet och bat-filen finns i Lab1a-katalogen i kursens git-repo. [↑](#footnote-ref-12)