# Laboration 1b

**Syfte med labben:**

Studenten ska använda moderna utvecklingshjälpmedel, som IDE (Integrated Development Environment), för programutveckling av inbyggda system.

Studenten ska använda sig av metoder för att självständigt konstruera och testa programvara för inbyggda system.

**Krav för godkänt:**

Ha genomfört följande labbmoment enligt denna handledning:

1. Ladda ner labbens kod exempel från github
2. Kört de första testerna inifrån Atmel Studio
3. Implementerat de nya testerna för LCD-displayen och genomfört en TDD-cykel för varje test.
4. Checka in kod + tester till ett centralt GIT-repo

Du visar upp för läraren först när du är helt klar med steg 4 och 5.

**Väl godkänt:**

För väl godkänt skall du inte bara implementera tester och programkod för det uppenbara rättfallet utan också testa att funktionen beter sig vettigt med konstiga indata, se mer detaljer nedan.

## Krav innan laborationen

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

* Atmel Studio 6.1 (finns endast för Windows)
* Github client ellerTortoiseGit (för andra datorer än Windows finns andra Git-klienter)
* Unity testramverk (unity.c, unity.h och unity\_internals.h)
* Rekommenderas också att ha ett USB-minne för att lätt kunna flytta sitt lokala repo

## Bygga labbsetup

Utrustning: Utvecklingskort Arduino Due

LCD display

## Test-driven utveckling (Test-Driven Development TDD)

Wikipedia har en bra introduktion till tesdriven utveckling: <http://en.wikipedia.org/wiki/Test-driven_development>

Rent generellt ser processen ut som nedan. I den här labben är några av stegen redan givna för de två första testerna.

Förberedelse

1. Skissa på en testlista
2. Skapa en .c–fil och en .h-fil med funktionen(erna) som skall testas, detta är modulen som skall testas

Testcykeln

1. Välj ett test från testlistan
2. Skriv/uppdatera testfilen för modulen med testet
3. Skriv/uppdatera en ”runner-fil” som exekverar valda tester
4. Kompilera och kör
5. Se testen fallera (för det finns ju inget innehåll i modul som ska testas)
6. Skriv programkoden
7. Kompilera och kör
8. Se testen/testerna passera, annars upprepa 6-7.

Samla ihop resultaten från TDD-cykeln

1. Commit till ditt personliga repo (valfritt)
2. Upprepa testcykeln tills alla tester i testlistan är implementerade. Då är även modulen färdigprogrammerad!

## Utveckling av funktioner för att använd LCD-display med Arduino

Ladda ner labbens kod till ert eget lokala repo och er egen branch. OBS: Om ni sparade er kod udner förra labben i mappen CourseMaterial kan ni att få kollisioner med vilken versions som är giltig.

1. Ställ er i master-branchen i ert lokala git-repo  
   >git checkout master
2. Syncha mot master-branchen på kursens repo på github  
   >git pull origin master
3. Ställ er i er egen branch på ert lokala repo (behöver ingen –b flagga för ni skapade ju branschen redan i förra labben)  
   >git checkout StudentNN
4. Merga in master-branchen i er egen branch för att få tillgång till labbens projekt i katalogen Coursematerial/Lab1b  
   >git merge master

Testlistan

Följande lista med tester föreslås för laborationen:

1. LCD-displayen initieras korrekt, dvs. att funktionen LCDInit() fungerar (finns redan i projektet)
2. LCD-displayen kan skriva ut ett enskilt tecken (angivet som 8 bitars ASCII), dvs. att funktionen LCDWrite() fungerar
3. Rensa LCD-displayen från alla tecken
4. Skriva ut ett fyrsiffrigt tal på displayen (lämpligtvis genom att först använda sig av funktionen som omvandlar talet till ASCII-tecken)

De två första testerna finns redan implementerade projektet, de tre sista ska ni skriva själva, först testerna och sen koden som uppfyller testerna enligt TDD-cykeln ovan.

Testcykeln

Välj ett test från testlistan. Fundera på om det är fråga om att utöka en befintlig funktion eller om ni bör skriva en ny funktion för att passera testet.

Om ni ska skriva en ny funktion måste ni lägga till funktionsdeklaration i någon befintlig c- och h-fil som redan finns. Ni kan också skapa en ny c- och h-fil med enbart deklarationen av den funktion ni måste utveckla för att passera testen.

Titta på hur testerna i den befintliga testfilen ser ut för inspiration. Läs mer i Unitys manual om det behövs.

Uppdatera testrunner-filen så att det nya testet körs. Bygg, ladda ner och kör projektet (se nästa stycke). Testet ska fallera! Det finns ju ingenting i funktionen som uppfyller testet. Detta är viktigt i TDD-cykeln för det kontrollerar att testet faktiskt funkar, nämligen säger till om funktionen inte gör vad den ska.

Eftersom Due-kortet inte har någon skärm där man kan skriva ut testresultatet så görs det enklast i terminalfönstret inifrån Atmel-Studio (tittar ni noga i testrunner-filen ser ni koden som initialiserar kommunikationen via USB-kabeln).

OBS! Bossac-programmeringen konkurrerar med terminalfönstret om COM-porten. För att programmering ska funka måste termalkommunikationen vara inte vara ”connect”. När man laddat ner binärfilen till kortet med Bossac klickar man i ”connect” igen och sen får man trycka på reste-knappen på Due-kortet så att alla testerna körs från början.

När testet körs men fallerar kan man börja skriva koden som gör att man kan passera testet. Man fyller funktionen med vad som behövs. Bra testdriven utveckling lägger till så små tester som det är möjligt i varje cykel. Då blir det lättare att skriva koden och man kan alltid backa tillbaka till en version där ens funktion passerade alla tester.

När man är säker på att funktionen kommer att passera alla tester, även den sist tillagda, bygger man, laddar ner och kör programmet och ser resultatet i terminalfönstret.

Börja om testcykeln för nästa test.

## Exempel på hur det ser ut för de två tester som redan har implementerad kod

Projektet har två kataloger som man jobba mot, src och test. Funktionerna man utvecklar ligger i src. Testramverket unity, testrunner-filen och filen(erna) med testerna ligger i katalogen test. När man sen vill använda sina testade filer i produktionssystemet behöver man bara filerna i src-katalogen. Testfilerna används alltså bara under utveckling.

### En enkel testfil

Alla testfunktioner är av typen

void test\_TheFunctionsdoesSomethingUseful(void);

Dessa tester deklareras snyggast i en motsvarande h-fil för att inkluderas i runner-filen (se nedan).

Ta för vana att döpa testfilen för att testa Module.c till testModule.c

I det här projektet heter filen med c-koden för testerna testLCDShield.c

Den innehåller två tester

* 1. Det första testet kollar att initieringsfunktionen för displayen ger ifrån sig värdet true (vilket oftast i C är ett positivt tal, i princip vilket som helst). Helst borde testet läsa från själva LCD-shielden att den är i rätt tillstånd[[1]](#footnote-1), men den som konstruerade den har valt att inte ansluta pinnen för läsningen till Due-kortet.
  2. Ett test som kollar att det går att skriva en byte på displayen. Testet kollar att det går att göra detta flera gånger i följd och där bytesen har olika format, som hexadecimalt, som decimalt och som en character. Funktionen som testas ger ifrån sig det faktiska värdet som skrivs på IO-pinnarna.

### Testrunner-filen som kör testerna

Filen som kör testerna döps till samma sak som testfilen med tillägget \_Runner. I vårt fall heter den testLCDShield\_Runner.c

Eftersom det är den här filen som allting körs ifrån måste den innehålla main(). Den måste också innehålla all nödvändig initiering av Due-kortet själv, många av de initialisationsfunktionerna kommer från Atmels ASF-bibliotek.

Det står kommenterat i main() var själva testramverket Unity börjar användas. Det är här de två testerna som redan är implementerats körs.

När ni lägger till fler tester görs det efter befintliga tester, men innan UnityEnd();

### Själva koden som utvecklas

De funktioner som programmeras ligger i src-katalogen. I filen LCDShield.c finns fyra funktioner definierade

void delay\_init(void)

void delay(uint32\_t us)

int LCDInit(void)

uint8\_t LCDwrite(uint8\_t byte, bool type)

men det finns bara testfall för de två sista.

int LCDInit(void)

Den här funktionen är en implementering i C av hur man initierar en LCD-display som styrs av Hitachi HD44780U (Dot Matrix Liquid Crystal Display Controller/Driver).

Själva algoritmen beskriv i Figur 24 på sidan 46 i databladet[[2]](#footnote-2).

LCD-shielden som man sätter på Due-kortet har bara 4 datapinnar kopplade (se tillverkarens hemsida[[3]](#footnote-3)), så efter att den omkonfigurerats till 4-bitas användning används LCDwrite() för att skicka data till displayen.

Tyvärr är inte pinnen som gör att man kan lösa status på LCD-shielden kopplad (pinne R/W), annars hade man använt den för att ge output till funktionen. Nu är funktionen implementerad på enklast möjliga sätt och ger ifrån sig 1 när den löpt igenom alla instruktionerna. Observera att utvariabeln sätts till 0 i början.

uint8\_t LCDwrite(uint8\_t byte, bool type)

Funktionen skriver en byte till LCD-displayen. Om type == HIGH tolkas det som att ett tecken ska skrivas, om type == LOW tolkas det som att ett kommande skickas till displayen[[4]](#footnote-4).

Funktionen abstraherar bort två saker från den som använder den:

* Att LCD-shiledne anvånder bara 4 pinnar i stgället för 8.
* Att LCD-shielden, som är konstruerad för Arduino Uno-kort, har pinnarna i omvänd ordning jämfört med Arduino Due.

Funktionen gör att tecknen och kommandona till LCD-displayen kan programmeras rakt av enligt tabellerna 4 och 6 i referensmanualen till Hitachi-controllern.

Utvärdet från funktionen är den byte som faktiskt skrivs till LCD-displayen på PIOC 23-26 (där bit 0-3 kastas om och bit 4-7 kastas om).

### Köra testerna

Eftersom det inte finns någon naturlig bildskärm kopplad till Due-kortet måste man få ut testresultaten på annat sätt.

I början på testrunner-filen finns hur man initierar att Due-kortet skriver på COM-porten på PC:n via USB-kabeln (configure\_console(void)). Det gör det möjligt att använda terminalfönstret inifrån Atmel Studio som output för prinft(); som Unity använder sig av. Men Terminalfösntet konkurrerar med Bossac om COM-porten så när man laddar ner sin binärfil till DUE-kortet måste terminalfönstret vara o-connectat.

Så fort man laddat ner sin binärfil till Due så startar ju programmet och därför missar man utskriften innan man gjort connect igen. Gör connect och tryck på reset-knappen för att starta testrunner-programmet om igen.

Om allt funkar som det ska ser du att Unity meddelar att två tester är godkända i terminalfönstret.

## Uppgifterna i labben

För godkänt gäller

1. Godkänt resultat på labbprovet
2. Syncha sitt lokala repo mot master-grenen på kursens repo på github, och merga in labbens projekt i sin egen branch
3. Köra minst två TDD-cykler så att koden man skrivit till slut passerar testerna.
4. Committa resultaten i sin egen branch och syncha mot github

När alla fyra stegen är klara visar du för labbhandledaren utskriften från testrunnerfilen och källkodsfilerna på github.

För väl godkänt gäller att testerna skall var mer heltäckande än bara det uppenbara rättfallet, se nedan under varje test.

Test: Rensa LCD-displayen från alla tecken

Skriv test för en enkel funktion som rensar displayen och flyttar tillbaka markören till startpositionen.

Funktionen kan t.ex. ge ifrån sig värdet TRUE (dvs. 1) när den är klar.

Information hur det görs finns i tabell 6 i databladet till controllern för LCD-dispalyen (Hitachi 44780).

Implementera funktionen när testet skrivits och körts. Titta till exempel på de fem sista raderna i funktionen InitLCD() i projektet.

Test: Skriva ut ett fyrsiffrigt tal på displayen

Skriv ett test för en funktion som tar ett fyrsiffrigt tal och skriver ut det som fyra efterföljande bytes kodade i ASCII på displayen, t.ex. med hjälp av LCDWrite().

Funktionen kan t.ex. ge ifrån sig en byte med ASCII-värdet på entalssiffran när den är klar[[5]](#footnote-5), men välj gärna andra lösningar om du vill.

*För väl godkänt gäller att testerna skall var mer heltäckande än bara det uppenbara rättfallet, i detta fallet innebär det att det måste finnas tester för tal som är större och mindre än fyra siffror, samt tester för negativa heltal. Funktionen ska givetvis klara av dessa tester.*

1. Med en assertion av typen

   TEST\_ASSERT\_EQUAL\_HEX8(0x0206, PORT\_C\_OUT); [↑](#footnote-ref-1)
2. <http://www.dfrobot.com/image/data/DFR0009/HD44780.pdf> [↑](#footnote-ref-2)
3. <http://www.dfrobot.com/index.php?route=product/product&path=123_124&product_id=51> [↑](#footnote-ref-3)
4. Dvs. type styr bit RS, som är kopplad till Digital Pin 8 på Arduino Due-kortet, vilket motsvaras av pin PC22 på ARM-processorn. [↑](#footnote-ref-4)
5. Om entalssiffran är 7 blir ASCII-värdet 0x37 [↑](#footnote-ref-5)