# Uppgift 1401a

*Programmering med handledningstid i labbsalen*
*Ulrik Eklund, 2015-06-12*

## Syfte med uppgiften:

Studenterna ska använda moderna utvecklingshjälpmedel för programutveckling för inbyggda system, som IDE (Integrated Development Environment), utvecklingskort och versionshanteringssystem.
Uppgiften görs normalt två och två, men det går bra att jobba ensam också. Det finns tid med handledare i labbsalen för frågor.

Det rekommenderas att ha ett eget USB-minne för katalogerna som med dina personliga kod repositories 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 tre moment enligt denna handledning:

1. Skapa ett eget lokalt repository baserat på det gemensamma repot på github som finns för uppgiften.
2. Starta Atmel Studio, modifiera källkoden i projektet, kompilera och ladda ner den till utvecklingskortet och köra programmet.
3. Committa filerna till ditt personliga repo, synka med github, och göra en ”pull request” till det centrala kursrepot.

Läraren kommer att kolla alla pull requests vid veckans slut, om du lämnar in senare bör du meddela via e-post.

## Lämpliga förberedelser

1. Labben kommer att innehålla grundläggande C-programmering, så repetera vad som gåtts igenom i kursen datateknik från första året. Mer specifikt kommer du att behöva
    a. Förstå deklarationer, tilldelning och bit-operationer på variabler (sektion 1 i [1]) + integers med fix storlek [2].
    b. Känna till vilka hårdvaruregister som används att styra digitala utgångar på en SAM3X8E Cortex-M3 mikrocontroller (sektion 31 i [3]).
    c. Förstå pekare (allmänt i sektion 3 i [1]), framförallt hur man använder pekare för att komma åt hårdvaruregister [5].
    d. Känna till hur pinnarna på Arduino Due förhåller sig till utgångarna på SAM3X8E [4]
    e. Hur man skapar större projekt med flera filer (sektion 5 i [1]).
    f. Tester för C-funktioner [6]
2. Det finns ett diagnostiskt test på It’s learning. Gör det innan du går till labbsalen som en koll på att du kan grunderna!

3. Läs igenom grunderna i versionshantering med git, t.ex. denna självstudiekurs: http://try.github.io/levels/1/challenges/1
4. Du skall också ha läst igenom *hela* denna handledning innan du går till labbsalen!

## Förutsättningar för att kunna börja på uppgiften

1. Om annan dator än labbsalens dator används (t.ex. egen dator) måste följande program finnas installerat på den datorn:
   a. Atmel Studio 6.2 eller senare (finns endast för Windows)
   b. Bossac flashprogrammeringsdriver, inklusive DOS-macro (finns om man laddar ner uppgiftens repo från Github)
   c. En Git-klient, t.ex. Github client[1] eller TortoiseGit[2] (för andra operativsystem än Windows finns andra Git-klienter)
2. Alla studenter i labbgruppen måste också ha registrerat en användare på github[3]! Om du redan har en, använd den!
   Meddela Ulrik på 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

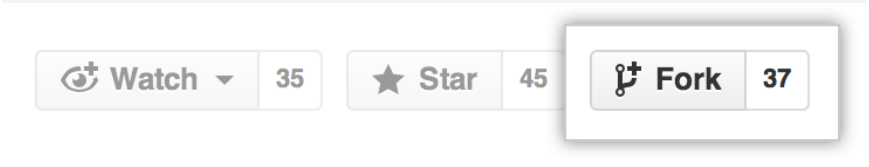
## Själva uppgiften

### Skapa ett personligt repo för denna programmeringsuppgift med hjälp av Git

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).

Det finns ett centralt repo för kursen på github med webbadressen:
https://github.com/MalmoUniversity-DA139A/Task1401a

1. Skapa en lokal ”fork” av detta repository[4] på kursuppgiftens webbsida. Då kan du jobba vidare på ditt eget projekt i ett eget repository utan att oroa dig för att förstöra kursens gemensamma repot. Enklast är att skapa en fork från githubs webbgränssnitt till kursrepot

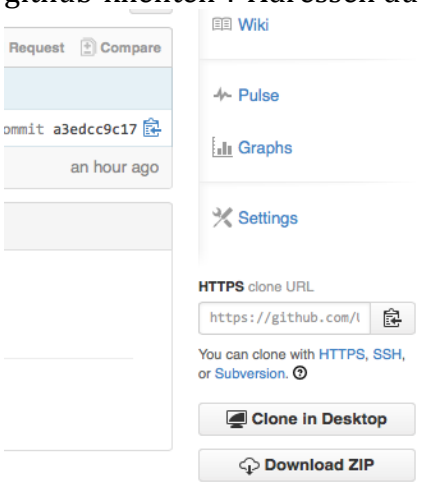



Figur 1: Fork-knappen på githubs webbsida

[1] http://windows.github.com/
[2] http://code.google.com/p/tortoisegit/
[3] http://www.github.com/
[4] https://guides.github.com/activities/forking/

2. Ange att du skall forka repot till din egen användare på github (den har du redan skapat). I resten av handledningen kallas denna användare *StudentNN.*

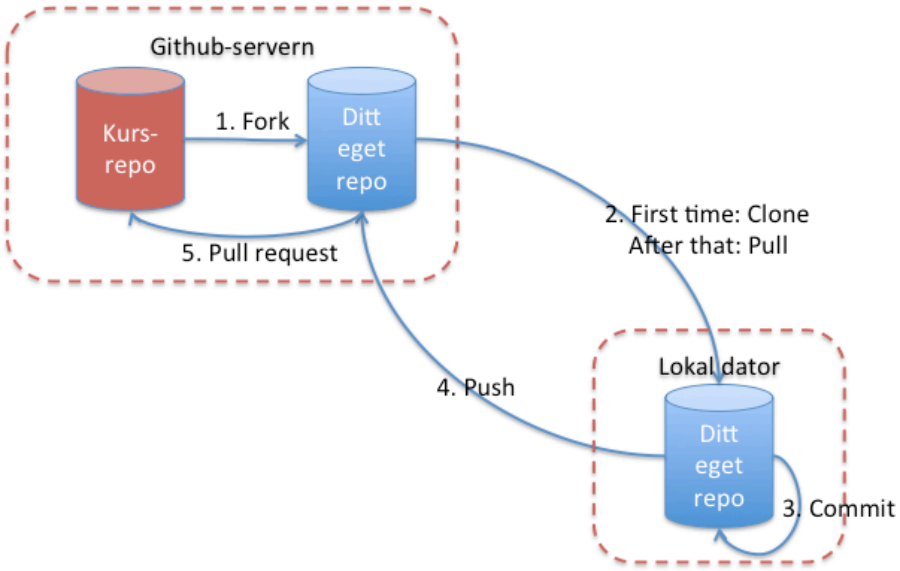
3. Än så länge finns bara ditt nya repo på github-servern, men du vill ju kunna jobba med projektet lokalt på din dator. Därför måste du *klona* det nyligen skapade repot till din lokala dator, vilket betyder att du skapar en identisk kopia som hela tiden kan synkas med github med kommandona *pull* och *push*.
   1. Välj en katalog på datorn där du vill ha din lokala kopia av repot.
   2. Klona repot på github repo till din valda katalog på datorn med hjälp av github-klienten[5]. Adressen du ska använda finns under ***HTTPS*** *clone URL*.




**Figur 2: Adressen till ett kodrepo på githubs webbsida.**

Under labben är det enklast om man jobbar mot ett och samma forkade repo i hela labben, även om det tillhör bara en specifik student. Om ni är två som vill jobba med varsitt lokalt repo kan ni ge båda studenterna rättigheter att jobba mot samma repo och sen klona repot på github till två olika datorer.

[5] Kommandoraden blir i Unix eller gitbash om man står i önskad katalog:
`>git clone https://github.com/StudentNN/Task1401a.git`
där *StudentNN* är ditt användarnamn på github.




**Figur 3: Översikt över hur man forkar, klonar och synkar kodrepot i kursen. Kursens gemensamma repo är rött, ditt privata repo är blått.**

### Arbetskatalogen för denna uppgiften

I katalogen `Task1401a` finns denna handledning och en projektmapp för Atmel studio vid namn `Project1401a`

### Använda Git

Med jämna mellanrum vill man committa sina ändringar till sitt lokala repo (Steg 3 i Figur 3)[6]. 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[7].
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[8]. 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: Utvecklingskort Arduino Due
USB-kabel (USB <-> USB micro)

Vi ska använda några av de digitala utgångarna, vara den kopplad till pinne 13 driver den oranga lysdioden som finns på Due-kortet.

---

[6] Git-shell-kommando: `>git commit -am ”mywork”`
[7] Ett exempel: https://github.com/akka/akka/commit/adfeb2c1f07153b7eec11705fda956f62b1bbb04
[8] `>git add filnamn.xxx`

Vilken I/O-port på processorn driver denna utgång?[9] _____

Due-kortet kan drivas med strömmen 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`. När man programmerar eller ”flashar” kortet gör man det också genom samma USB-kabel.

## Labbens programmeringsuppgift

Om du lyckats med att klona git-repot finns det nu i katalogen `Task1401a\Project1401a` på din lokala dator en projektfil för Atmel Studio: `Project1401a.atsln`
Öppna den genom antingen genom att dubbelklicka eller genom ”open project” inifrån Atmel Studio.

Labben går ut på att ni ska skriva funktioner som passerar de testfall som finns definierade i projektet.
Testa att kompilera projektet under menyn ”Build”. Det här kollar främst att syntaxen är rätt, variabler är definierade och lite annat. C har annars få begräsningar vad som är rätt eller fel i språket.

Du bör inte få några varningar om du använder samma version på utvecklingsmiljön som projektet gjort hittills[10].

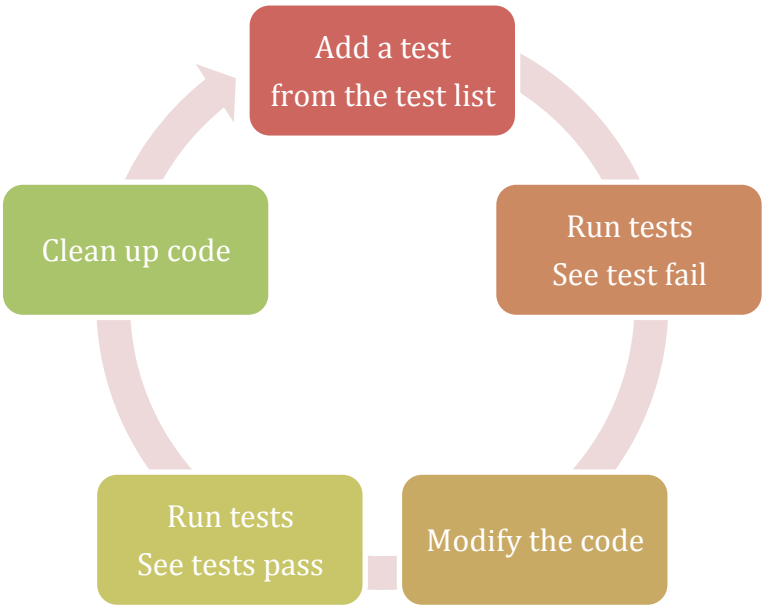
Ladda ner den färdiga binärfilen till Due-kortet via USB-kabeln med kommandot `BossacArduinoDue(Debug)` under Tools-menyn[11].

### 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.

---

[9] Se http://arduino.cc/en/Hacking/PinMappingSAM3X
[10] Varningar som kan finnas är ”bara” för att en del variabler är definierade som aldrig används (i den här labben), så vi bryr oss inte om dem.
[11] Finns inte det kommandot i menyn följ instruktionerna här, eller som kommentarer i bat-filen: https://github.com/ctbenergy/BossacArduinoDue
Både bossac-programmet och bat-filen finns i katalogen i uppgiftens git-repo.




I den här labben är några av stegen redan givna för alla testerna. I detalj så kommer ni att genomföra följande steg för varje test.

*Förberedelse*

1. Skissa på en testlista (redan gjort för den här labben!)
2. Skapa en .c–fil och en .h-fil med funktionerna som skall testas, detta är modulen som skall testas (också redan gjort: `digitalIO.h` och `digitalIO.c`)

*Testcykeln*

3. Välj ett test från testlistan (gör det i samma ordning som de står i `test_digitalIO.c`)
4. Skriv/uppdatera testfilen för funktionen som ska testas (redan gjort för alla tester)
5. Skriv/uppdatera en ”runner-fil” som exekverar valda tester (den första testen körs, de andra är bortkommenterade i det ursprungliga projektet)
6. Kompilera
7. Ladda ner det kompilerade programmet till Due-kortet
8. Kör programmet på Due-kortet
9. Se testen fallera, för det finns ju inget innehåll i den funktion som ska testas. För att rsultatet ska synas måste terminalfönstet inuti Atmel Studio vara installerat.
10. Skriv programkoden så att funktionen klarar av testet (lägg till kod i `digitalIO.c`)
11. Kompilera och kör
12. Se testen/testerna passera, annars upprepa 8-9.
13. Städa och eventuellt kommentera din kod om det inte redan är gjort.

*Samla ihop resultaten från TDD-cykeln*

14. Commit till ditt personliga repo (valfritt)
15. Upprepa testcykeln tills alla tester i testlistan är implementerade. Då är även de två funktionerna färdigprogrammerade!

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

Projektet har två kataloger som man jobba mot, `src` och `test`. Funktionerna man utvecklar ligger i `src`. Testrunner-filen och filen(erna) med testerna ligger i katalogen `test`. Testramverket unity ligger i katalogen `unity`.
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 `test_Module.c`
I det här projektet heter filen med c-koden för testerna `test_digitalIO.c`
Den innehåller totalt 9 tester:

1. Det första testet kollar att initieringsfunktionen för utgång PB27 har initierats korrekt för att kunna användas som en digital utgång. Det görs genom att läsa av en status-bit för port B.
2. Det andra testet kollar att funktionen som sätter utgången till hög faktiskt gör det, också genom att läsa av en status-bit som ger det faktiska värdet på rätt utgång.

Resterande tester är uppbyggda på samma sätt, men är bortkommenterade. Så fort du har klarat av ett test kan du inkludera nästa test genom att ta bort kommenteraren för det testet.

### Testrunner-filen som kör testerna

Filen som kör testerna döps till samma sak som testfilen med tillägget *`runner`*. I vårt fall heter den `test_digitalIO_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 det första testet körs.
När man lägger till fler tester görs det efter befintliga tester, men innan `UnityEnd();`

### Se resultatet från 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 `printf()` 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 (i Atmel Studio 6.2 görs detta automatiskt).

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 resultatet från testerna i terminalfönstret.

### Själva koden som utvecklas

De funktioner som ska programmeras ligger i src-katalogen. I filen `digitalIO.c` finns två funktioner definierade

```
void pinMode(int pinNumber, mode_definition mode)
void digitalWrite(int pinNumber, int value)
```

men det första testfallet gäller bara den första av dessa funktioner.

Titta på tabell 31-2 på sidan 629 i referensmanualen för SAM3X8E-processorn [3] som sitter på Due-kortet. Där finns ett programmringsexempel för vilka register som används för att styra de digitala utgångarna

### void pinMode(int pinNumber, mode_definition mode)

Konfigurerar en (1) utpekad pinne på Due-kortet att fungera som antingen en in- eller utgång. I den här uppgiften behöver funktionen bara programmeras för att konfigurera utgångar.
Behöver man konfigurera två pinnar så får man anropa samma funktion två gånger med olika argument på `pinNumber`. Givetvis så skall det andra anropet inte ändra konfigurationer för den första pinnen (om det inte är samma pinne förstås).
De register som behöver användas är (sida 642 + 643 i [3]):

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x0000 | PIO Enable Register | PIO_PER | Write-only | – |
| 0x0010 | Output Enable Register | PIO_OER | Write-only | |

För att komma åt register på en speciell minnesadress kan man använda en pekare med följande deklaration

```
/* defines the address for enabling the output pins of port B
register with base adress 0x400E1000*/
uint32_t *const p_PIOB_OER = (uint32_t *) (0x400E1000U +
0x0010U);
```

Sen kan man sätta en enstaka bit i detta register med maskning på följande sätt:

```
*p_PIOB_OER |= 0x000000040;
```

### int digitalWrite(int pinNumber, int value)

Funktionen ska sätt en utgång på Due-kortet till hög eller låg, förutsatt att pinnen tidigare har satts som en utgång med `pinMode`.
De register som behövs för detta är (sida 660-661):

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x0030 | Set Output Data Register | PIO_SODR | Write-only | – |
| 0x0034 | Clear Output Data Register | PIO_CODR | Write-only | - |

*När du har uppdaterat de två funktionerna i `digitalIO.c` så att alla nio testerna passeras har du klarat av all programmering (men det är lite kvar med själva inlämningen*

*för att få godkänt). Glöm inte att spara ner testutskriften från terminalen till en en fil* `testResult.txt` *i test-katalogen och committa det färdiga programmet till git på din dator.*

Om du vill se med blotta ögat att dina funktioner fungerar kan du lägga till följande loop efter `UnityEnd();` i `test_digitalIO_runner.c`

```
int i; /* loop counter for the delay */
volatile int j; /* Dummy volatile to prevent compiler optimising the
variable away */
int delay_length = 400000; /* variable determining the length of a
delay */

while(1) /* repeat blink in infinity */
{
    j=0; /* makes sure j doesn't overflow */

    digitalWrite(13, 1); /* sets a bit of I/O port B to high */
    for (i=0; i<delay_length; i++) /* The delay counter */
    {
        j++; /* some easy predictable operation */
    }

    digitalWrite(13, 0); /* clears a bit of I/O port B */
    for (i=0; i<delay_length; i++) /* The delay counter */
    {
        j++;
    }
}
```

## Lagra arbetet på github och lämna in för bedömning

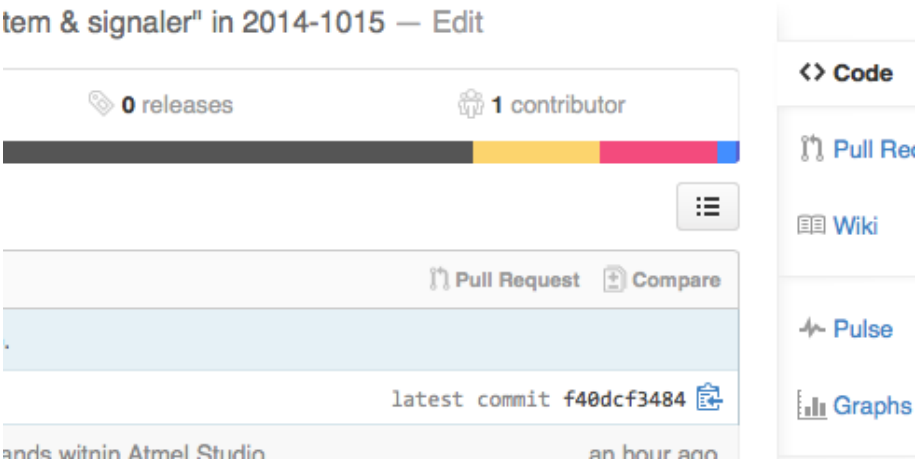
Om du är klar med all programmering är det vara dags att synka ditt lokala repo med det som du har på github.
En viktig sak som måste göras är att lägga till eventuella nya filer till ditt repo (som `testResultat.txt`), man måste alltså tala om explicit vilka filer som git ska hålla reda på[12]. Det går att fråga git-klienten om några nya filer har tillkommit i repots katalog[13]. Innan man har lagt till filerna så uppdaterar en commit bara de filer som redan finns i repot.

Om du har gjort en slutlig commit, inklusive nya filer, kan det vara dags att görs med en push tillbaka till github[14] (steg 3 i Figur 3: Översikt över hur man forkar, klonar och synkar kodrepot i kursen. Kursens gemensamma repo är rött, ditt privata repo är blått.). Din gitklient är oftast så smart att den kommer ihåg varifrån man klonade repot, därför kan man oftast använda *origin* istället för den långa webbaddressen[15].

---

[12] `>git add –all *` säger till git att hålla reda på alla nya filer som skapats om man står i repots huvudkatalog.
[13] `>git status`
[14] `>git push origin`
[15] https://github.com/StudentNN/Task1401a.git

Har du kommit så långt har du ett uppdaterat repo som du kan se på githubs webbsida. Nu ska du skicka in en begäran att ägaren till det gemensamma kursrepot ska titta på dina ändringar genom en *pull request* (steg 5 i Figur 3). Där anger du en rubrik (till exempel att du är klar med en viss uppgift), och fyller i en beskrivning av vad dina ändringar består av.
*Glöm inte att ange namnen alla som bidragit med att göra klart uppgiften! Det går inte att komma i efterhand och påstå att du varit med om ditt namn inte finns med på originalinlämningen! I så fall får du göra en egen pull request efteråt.*




Ulrik kommer att titta på alla som har lämnat in till fredag kväll och de som är godkända får bonuspoäng till tentan. Om du lämnar in senare kommer de att bedömas vid terminens slut i samband med tentan.

## Referenser


[1] Nick Parlante, Essential C, 2003, http://cslibrary.stanford.edu/101/
[2] C data types, https://en.wikipedia.org/wiki/C_data_types
[3] Atmel SAM3X / SAM3A Series SMART ARM-based MCU DATASHEET, 2015, http://www.atmel.com/devices/sam3x8e.aspx
[4] SAM3X-Arduino Pin Mapping, http://www.arduino.cc/en/Hacking/PinMappingSAM3X
[5] Dan Saks, Representing and Manipulating Hardware in Standard C and C++, Embedded Systems Conference San Francisco, 2002, http://www.open-std.org/jtc1/sc22/wg21/docs/ESC_SF_02_465_paper.pdf
[6] Mark VanderVoord, Embedded Testing With Unity And Cmock - A Book For Those Who Code C And Want Awesome Well-Tested Products Using Free Tools And For Those Who Enjoy Short Books with Long Titles, 2010