STM32 Abend 1

u23 2013

andy, florob, gordin, ike, meise, tobix, zakx

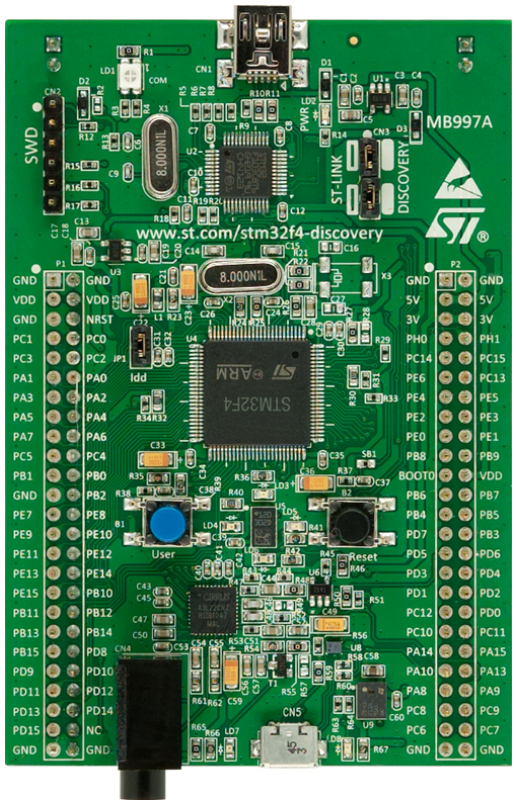
Chaos Computer Club Cologne e.V.
http://koeln.ccc.de

Cologne
2013-10-21

andy <andy@koeln.ccc.de>

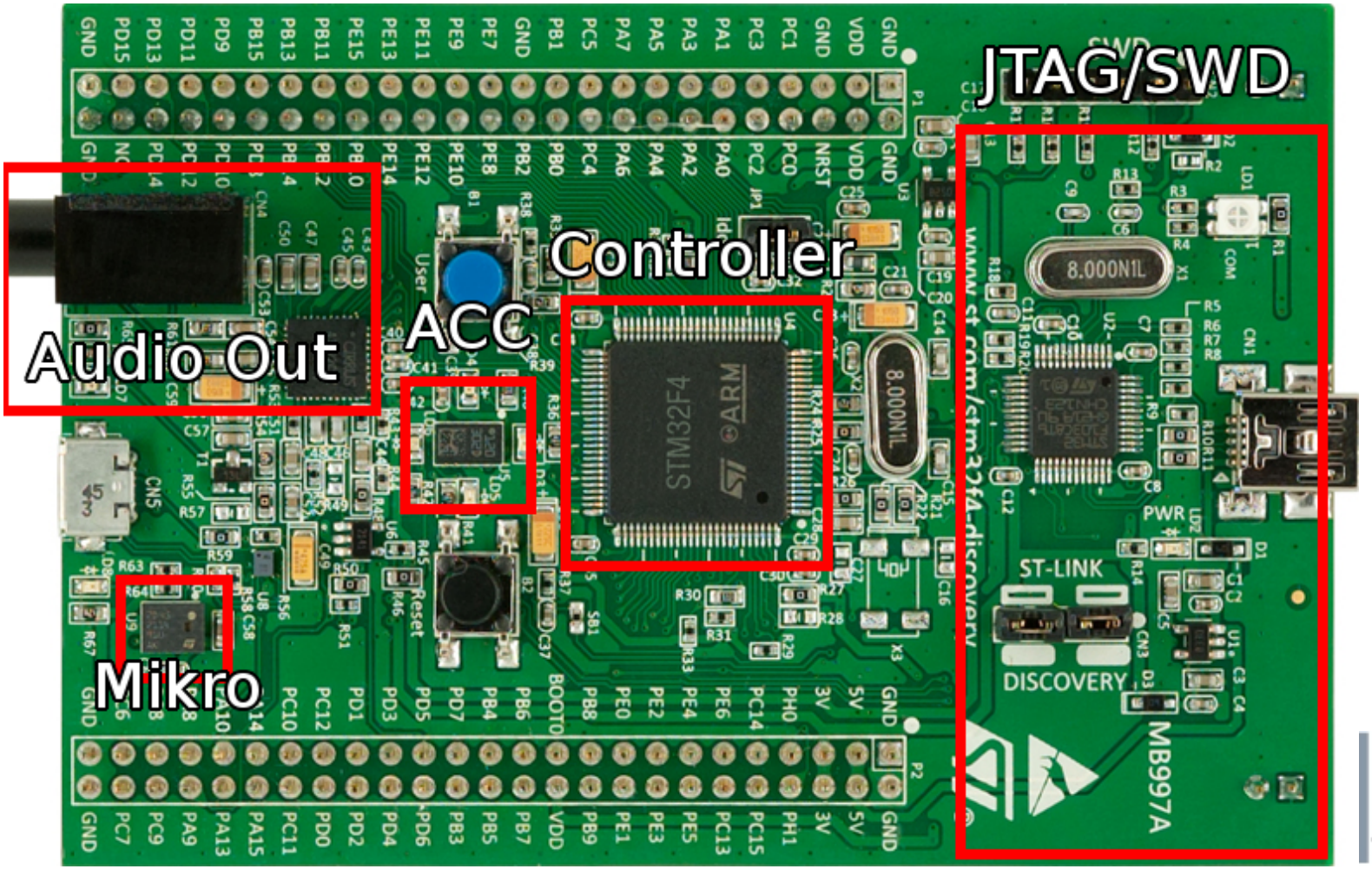
- Einführung
  - 2013-10-19 11:00 — C99 Einführung
- Reguläre Termine
  - 2013-10-21 19:30 — STM32-Einführung (heute)
  - 2013-10-28 19:30 — Peripherie des STM32
  - 2013-11-04 19:30 — Kommunikation mit anderen Bausteinen
  - 2013-11-11 19:30 — Kommunikation mit anderen Bausteinen (2)
  - 2013-11-18 19:30 — Projektarbeit
  - 2013-11-25 19:30 — Projektarbeit
  - 2013-11-28 19:30 — OpenChaos

STM32F4 Discovery



- STM32F407VGT6 32-bit ARM Cortex-M4F
- 1 MB Flash
- 192 KB RAM (64KB CCM, 128KB SRAM)
- JTAG via ST-Linkv2
- USB OTG
- 100pin LQFP

Boarddetails



STM32F407

- SoC = System on a Chip
- Lauffähiges Rechensystem komplett in einem Chip
- Nicht nur CPU + RAM, auch andere Peripherie:
  - USARTs
  - SPI-Controller
  - I2C-Controller
  - DCMI-(Kamera) Controller
  - DMA-Engines
  - GPIOs
  - USB-Controller
  - ... (siehe Datasheets später)

Bei nem "richtigen"PC ist oberer Kram alles im Chipsatz

Chipübersicht

Datasheets sind eure Freunde! Ihr müsst allerdings auch erstmal lernen, wie man sie liest.

- Blockdiagramm: Datasheet STM32F407xx Seite 18
- Alternate Pin Functions: Datasheet STM32F407xx Seite 45
- Memory Map: Datasheet STM32F407xx Seite 63

Ist leider alles zu groß für die Slides :(
Datasheets liegen im Libarary-Repo unter docs/

Externe Peripherie

Wir haben einiges an externen Bauteilen bestellt. China sollte irgendwann liefern.

- Servos
- Character- und (ein) Grafikdisplay
- Funkmodule
- Real-time clocks
- SD-Kartenadapter
- Magnetometer
- Ultraschall Entfernungssensoren
- 7-Segmentanzeigen
- Breadboards
- Kabel zum $stuff zusammenstecken
- LEDs
- Buttons/Joysticks

Projekte

Am Ende sollt ihr mit diesen Bausteinen was hübsches bauen.
Erste Idee (vllt. etwas übertrieben): 3D-Scanner

- Zu scannendes Objekt sitzt auf einem Drehteller
- Ultraschall-Entfernungsmesser misst Entfernung zum Objekt
- Objekt wird etwas gedreht, nochmal gemessen
- Wenn eine Zeile fertig ist, Sensor in nächste Zeile schieben
- Irgendwann hat man dann vielleicht ein gescanntes 3D-Objekt

Sowas in der Art, gerne auch einfacher. Seid kreativ!

Unsere Library

- Übernimmt Startup-Krempel und springt in main()
- Stell das Buildsystem zur Verfügung (Makefiles)
- Zu finden unter
  `https://github.com/cccc/U23_2013_examples`
- Beim Bauen wird ein Fehler bei stlink-trace auftreten (libncurses nicht gefunden)
- *sudo apt-get install libncurses-dev* behebt das in der VM

Linux Setup

- Dependencies siehe README.md
- libncurses (und das -dev Paket dazu) installieren, fehlt in der README
- gcc-arm-embedded muss im *PATH* liegen
- Zusätzlich noch git und make installieren
- Dann sollte das einfach so funktionieren (Magie!)

Ordnerstruktur

apps/	Eure selbstgeschriebenen Apps
bare_metal/	Beispiele ohne unsere lib
build/	Makefiles für das Buildsystem
docs/	Dokumentation, Datasheets, Schaltplan des Boards
tools/	Utilities (stlink-trace)
examples/	Beispielprogramme die unsere Library nutzen
libs/	Die Startup- und Peripherie-Libraries

Neues Projekt anlegen

1. Kopiert den Ordner *hello_word* in *apps*
2. Öffnet die Datei *target.mak* in eurem **grade kopierten Ordner** und ändert die Variable *TARGET* so, dass sie genau so heisst wie euer neuer Ordner
3. Öffnet im Ordner *apps* die Datei *target.mak* und fügt zur Variable *SUBDIRS* euren neuen (getrennt durch Leerzeichen) Ordnernamen hinzu
4. Ihr könnt jetzt Sourcecode hinzufügen oder schreiben. Solltet ihr neue Dateien haben, die ihr kompilieren wollt, tragt sie in die Variable *CCSOURCES* in der *target.mak* ein.

Kompilieren und Flashen

- Im Stammordner reicht ein *make* um alles zu kompilieren
- *make upload* lädt die Firmware 01_blink auf euer Board
- Mit *make upload-firmwarename* oder *make upload FIRMWARE=firmwarename* ladet ihr eine bestimmte Firmware auf das Board
- In der Datei config.mk könnt ihr die Standardfirmware ändern
- Es gibt noch *make upload-fast* was etwas schneller ist, was ihr benutzt spielt keine Rolle

Debuggen via gdb

```
$ ./debug.sh
Connection to STM32 established. You can start debugging now
Ctrl+C or unplug USB to kill Connection
```

Auf anderer Shell:

```
$ arm-none-eabi-gdb u23_lib/examples/01_leds/obj/01_leds.elf
GNU gdb (GNU Tools for ARM Embedded Processors) 7.4.1.20130312-cvs
[... license bla bla ...]
Reading symbols from /home/andy/Documents/CCC/U23_2013/examples/u23_lib/examples/01_leds/obj/01_leds.elf.

(gdb) target extended-remote :3333
Remote debugging using :3333
0x00000000 in ?? ()

(gdb) monitor reset init
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x080004c4 msp: 0x10010000

(gdb) b main
Breakpoint 1 at 0x80001d4: file /home/andy/Documents/CCC/U23_2013/examples/u23_lib/examples/01_leds/src/ma

(gdb) c
Continuing.
Note: automatically using hardware breakpoints for read-only addresses.

Breakpoint 1, main () at /home/andy/Documents/CCC/U23_2013/examples/u23_lib/examples/01_leds/src/main.c:8
8 InitializeSystem();

(gdb) <weitere Befehle>
[...]
```

Alternativ Eclipse. Gordin? Ach, sowas will doch niemand...

Ein neues Programm

```
#include <System.h>
#include <stdio.h>

int main()
{
    // Do some basic initialization tasks
    InitializeSystem();

    // Initialize pins for LEDs
    InitializeLEDs();

    // Enable printf via trace macrocell (get output with 'make trace')
    EnableDebugOutput(DEBUG_ITM);

    //Turn on all LEDs
    SetLEDs(1 | 2 | 4 | 8);

    iprintf("Hello, World!\r\n");

    while(1);
}
```

LEDs

- LEDs kann man mit *SetLEDs(int);* setzen
- Parameter ist eine Bitmaske
- LED 1 und 4 einschalten: (1<<0) | (1<<3)

Accelerometer

- Ein Accelerometer misst auf 3 Achsen (X, Y und Z) die Beschleunigung des Boards relativ zur Erde
- Zu gut Deutsch: Wie liegt das Board grade im dreidimensionalem Raum
- 3 Werte: Für jede Achse einen

Accelerometer Demo-Code

```
#include <System.h>
#include <Accelerometer.h>

int main()
{
    // Do some basic initialization tasks
    InitializeSystem();

    // Initialize pins for Accelerometer
    InitializeAccelerometer();

    // Calibrate the Accelerometer on the Start
    CalibrateAccelerometer();

    while (1)
    {
        int led = 0;
        int8_t Data[3] = {};

        // Read new Data
        ReadCalibratedAccelerometerData(Data);

        //... do some fancy WiiMote-Style shit ...
    }
}
```

Debug Output per ITM

- STM hat eine Trace-Makrozelle (ITM = Integrated Trace Macrocell)
- Einschalten per *EnableDebugOutput(DEBUG_ITM);*
- *make trace* startet dann das trace utility
- Controller wird resettet und ihr seht alles, was ihr per *printf()* ausgebt

Debug Output per Serial

- Debugging per Serial einschalten mit *EnableDebugOutput(DEBUG_USART);*
- Danach kann man *printf()* benutzen
- Benutzt wird UART2 mit TX-Pin PA2 und RX-Pin PA3

git

- git ist ein Versionskontrollsystem
- Verwaltet alle Arten von Quellcode
- Lässt euch Änderungem, die ihr an Quellcode gemacht habt, verteilen
- Wir benutzen es für unsere Software
- Hier nur kurze Einführung
- Im Wiki stehen zwei längere Howtos:
- http://try.github.com/
- http://githowto.com/

git clone <url>

- Lädt das Repository vom Server und legt euch eine lokale Kopie an

```
~ git clone git://github.com/cccc/U23-Library.git
Cloning into 'U23-Library'...
remote: Counting objects: 971, done.
remote: Compressing objects: 100% (713/713), done.
remote: Total 971 (delta 455), reused 739 (delta 223)
Receiving objects: 100% (971/971), 1.50 MiB | 650 KiB/s, done.
Resolving deltas: 100% (455/455), done.
```

git init

- Erstellt euch ein neues lokales Repository

```
~ git init
Initialized empty Git repository in /home/andy/Desktop/test/.git/
```

git status

- Zeigt einem Dateistatus an

```
~ git status
# On branch master
#
# Initial commit
#
# Untracked files:
# (use "git add <file>..." to include in what will be committed)
#
# testfile
nothing added to commit but untracked files present (use "git add" to track)
```

git add <filename>

- Fügt eine Datei dem Index hinzu
- Damit weiß git, dass es diesen Inhalt der Datei beachten soll
- Muss für alle Dateien gemacht werden mit deren Inhalt man was gemacht hat
- Geht auch direkt auf ganzen Ordnern

```
~ git add testfile
```

git commit

- Fasst den aktuellen Index zusammen zu einem commit
- Commit = ein Änderungssatz zu einer bestimmten Sache
- Jeder Commit bekommt eine Commitmessage der ihn beschreibt, sodass andere Leute nachvollziehen können, was ihr gemacht habt

```
~ git commit -m "Neues file hinzugefuegt"
[master (root-commit) d9ca032] Neues file hinzugefuegt
 0 files changed
 create mode 100644 testfile
```

git rm <filename>

- Löscht eine Datei

```
~ git rm testfile
rm 'testfile'
```

git mv <source> <destination>

- Verschiebt eine eine Datei
- Wird auch zum Umbenennen verwendet

```
~ git mv testfile neuesfile
~ git status
# On branch master
# Changes to be committed:
# (use "git reset HEAD <file>..." to unstage)
#
# renamed: testfile -> neuesfile
#
```

Remotes

- Remotes sind Zeiger auf andere Repositories des selben Projekts
- Ihr könnt dahin pushen (falls ihr dir Rechte habt) oder davon pullen
- Der C4 hat ein Repo, ihr habt alle Repos
- Ihr pusht also immer nur in euer eigenes Repo und pullt von allen anderen in eurem Team oder vom C4 um ihre neuen Änderungen zu bekommen
- Remote hinzufügen geht so: *git remote add <remotename> <remoteurl>*

git push

- Schiebt eigene Änderungen auf den Server
- Server ist nicht zwingend notwendig, aber praktisch um Änderungen zu verteilen
- Wenn jemand anders in der Zwischenzeit Änderungen gepusht hat, die ihr noch nicht habt, kann es sein, dass es abgelehnt wird – dann einmal pullen (ggf. mergen) und wieder pushen
- Beispiel: *git push [remote [branchname]]*

git pull

- Holt Änderungen vom Server
- Beispiel: *git push [remote [branchname]]*

git log

- Zeigt euch alle Commits der Reihe nach an
- gitk ist das selbe in grafisch

Aufgaben

1. Falls ihr noch kein Board habt, geht zum Andy und holt euch eins
2. Legt euch nen github Account an
3. Forkt das U23 Projekt des C4s (C4-Account: cccc)
4. Clonet euren neuen Fork
5. Fügt das C4-Repo als upstream remote hinzu (*git remote add upstream* *https://github.com/cccc/U23_2013_examples.git*)
6. Legt ein neues Projekt an
7. Lasst LEDs blinken
8. Gebt euch mal Debugging output aus (per ITM)
9. Vielleicht schafft ihr es schon Accelerometerwerte per Debugoutput auszugeben
10. Probiert mal mit dem GDB zu Debuggen