

Advanced System Design

Zusammenfassung

Joel von Rotz /  [Quelldateien](#)


Inhaltsverzeichnis

Crypto

2

Cesar Cipher / Substitution Cipher

2

Enigma Maschine 

2

Stream & Block Cipher


3

Konfusion


3

Diffusion

3

Feistel Network 

3

DES 

3

Initial & Final Permutation

4

f-Funktion

4

Expansion E

4

Key Scheduling Transform X


4

Das Coole

5

Das Problem

5

TEA 

5

XTEA

5

AES

6

Cipher Modi [W](#)

6

Electronic Code Book (ECB)

6

Cyber Block Chaining (CBC)

6

Cipiher FeedBack (CFB)

6

Output FeedBack (OFB)

6

CounTeR (CTR)

6

Galois Counter Mode (GCM)

6

Meet-In-The-Middle Attack

6

Double Enrcpytion

6

Triple Enrcpytion

6

Key Whitening

6

Hash Functions


6

Block

6

weitere Begriffe

6

Docker 

6

Was'n Docka?

6

Begriffe

6

Docker Pfade

7

Images

7

>_ Auflisten

7

>_ Löschen

7

>_ Neue Version veröffentlichen

7

>_ Grösse

7

>_ Informationen abrufen

7

Container

7

Container-Layer

7

>_ Erstellen/Ausführen/Starten

7

>_ Auflisten

8

>_ Attach/Detach

8

>_ Interaktivität

8

>_ Löschen

8

>_ (Um-)benennen

8

>_ Dateien kopieren Host ⇌ Container

8

>_ Ausgabe

8

>_ Prozesse ausführen/abfragen

8

>_ Logging

9

Lebenszyklus

9

>_ Grösse --size

9

Image kreieren aus Container

9

Volume

9

Host Volumes

9

Named Volumes

9

Anonyme Volumes

9

Datenaustasch

9

Speicherpfade

10

>_ Auflisten

10

>_ Volumen löschen

10

>_ Erstellen

10

Dockerfile

10

Anweisungen

10

Startkommando

11

Image builden

11

Image Build History zeigen

11

Multistage

11

Netzwerke

11

Docker Hub

11

Images suchen & herunterladen

11

Image Reference Format

11

Performance

12

Cache

12

Trashing

12

Thrashing / [Seitenflattern](#)

12

Cache Struktur

12

GCC Optimization

12

12

Safety

13

Terms

13

Requirements

13

V&V&C

13

Computers in Safety Related Systems?

13

Silver Bullet

13

Hazard Analysis

13

FMEA: Failure mode and effects analysis

13

HAZOP: Hazard and operability studies

13

ETA: Event tree analysis

14

FTA: Fault tree analysis

14

Risk Analysis

14

Severity

14

Frequency

14

Risk

14

Integrity

14

Integrity Levels

15

Allocating Levels

15

Design for Safety

15

Types of Fault

15

Fault Tolerance

15

Triple Modular Redundancy (TMR)

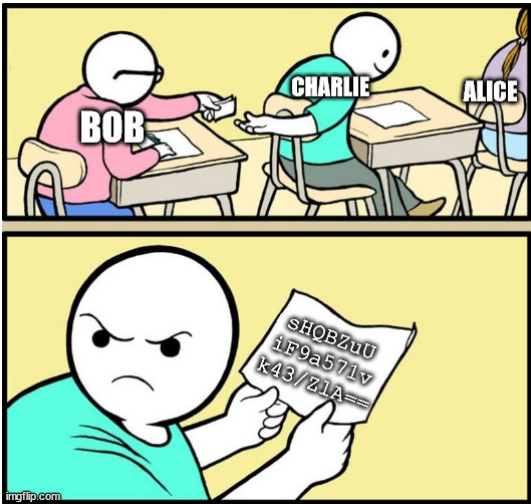
15


Voter

15

TMR with triple voting	15
Multistage TMR	16
NMR - N Modular Redundancy	16
Dynamic Redundancy	16
Self Checking Pair	16
Redundancy Combining	16
Software Faults	16
Reliability	16
(Un)-Reliability R (Q)	16
Failure Rate z(t)	17
Bathtub Curve	17
Useful-Life	17
Time-Variant Failure Rates	17
Mean times	17
Mean Time to Failure	17
Mean Time to Repair	17
Failure in time	17
Reliability modelling	17
Triple Modular Redundancy	18
Dynamic Redundancy	18
Cut and Tie Sets	18
Reliability prediction	18
Resistor (DoD MIL-Handbook 217)	18
Capacitor (DoD MIL-Handbook 217)	18
Prediction of software reliability	19
Reliability assessment	19
Software - Formal Methods	19
Spark	19
Hardware - Safety Processors	19
Hercules RM42 MCU	19
Dual CPU	20
Memory	20
Watchdog	20
Trends - Low Power	21
Firm-/Software Optimierungen	21
Anhang	21
Crypto	21
Permutation Tabellen	21

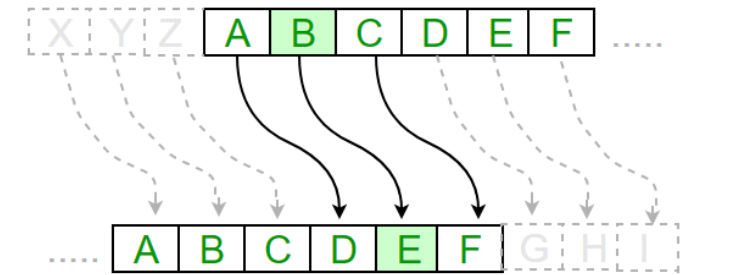
Crypto



 Hinweis

Wenn Daten Sequenzen in Blöcke geteilt werden (z.B. 64-Bit), dann wird davon ausgegangen, dass bei unvollständigen Blöcken die restlichen Bits mit z.B. 0 ausgefüllt werden.

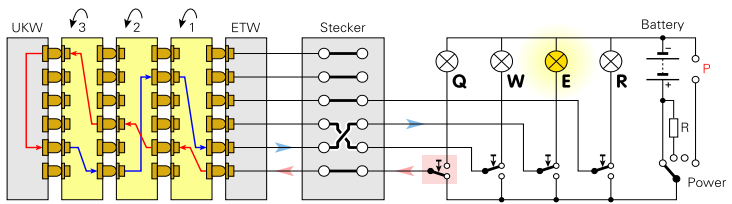
Cesar Cipher / Substitution Cipher



Buchstaben werden um x Positionen verschoben (z.B. $A \xrightarrow{+2} C$). Nachteil ist, dass die Entschlüsselung sehr einfach ist.

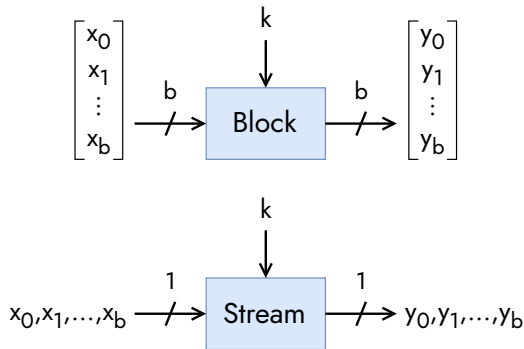
Enigma Maschine

Die Enigma Maschine ist ein komplexes Ent- & Verschlüsselungs System, welches während den Weltkriegen von den Nazis hauptsächlich verwendet wurde (und durch Alan Turing geknackt).



Nachjedem Tastendruck leuchtet ein Buchstabe auf und die Rotoren drehen sich, damit der nächste gleiche Tastendruck nicht den gleichen Buchstabe ergibt. Mit den Steckern können die Buchstaben umkonfiguriert werden (bei Doppelstecker wird z.B. $A \rightarrow B$ & $B \rightarrow A$ und dadurch halbiert sich die Möglichkeiten zu 13).

Stream & Block Cipher



! Ver- & Entschlüsseln

Verschlüsselte Informationen können mit dem genau gleichen Ablauf wieder entschlüsselt werden.

DES

Data Encryption Standard ist ein Cipher, der 64 Bit lange Blöcke mit einem 56 Bit langen Schlüssel verschlüsselt.

Konfusion

Konfusion ist eine Verschlüsselungsoperation, bei der die **Beziehung zwischen Key und Ciphertext verschleiert** wird. Ein gängiges Element zur Erzielung von Konfusion ist heute die Substitution.

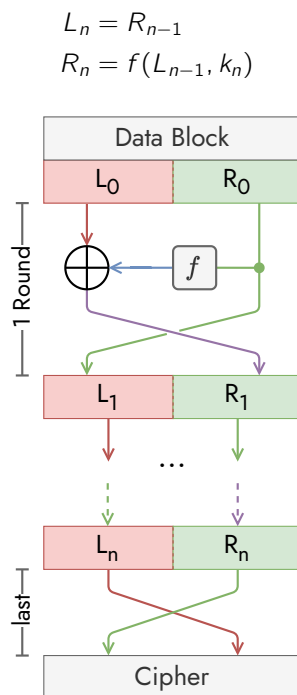
Konfusion erhöht die Mehrdeutigkeit des Ciphertextes und wird sowohl von Block- als auch von Stream-Ciphern verwendet.

Diffusion

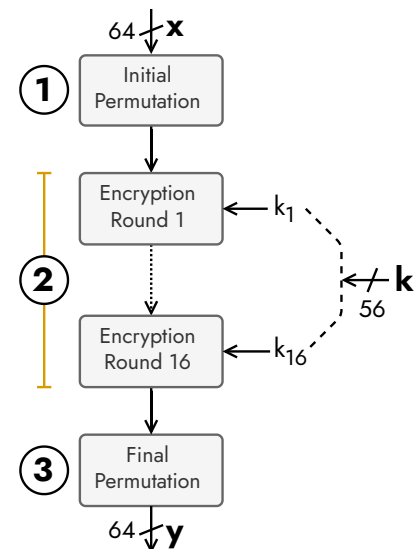
Diffusion ist eine Verschlüsselungsoperation, bei der der Einfluss eines Klartextsymbols auf viele Ciphertext-Symbole verteilt wird, um die statistischen Eigenschaften des Klartextes zu verbergen.

Feistel Network

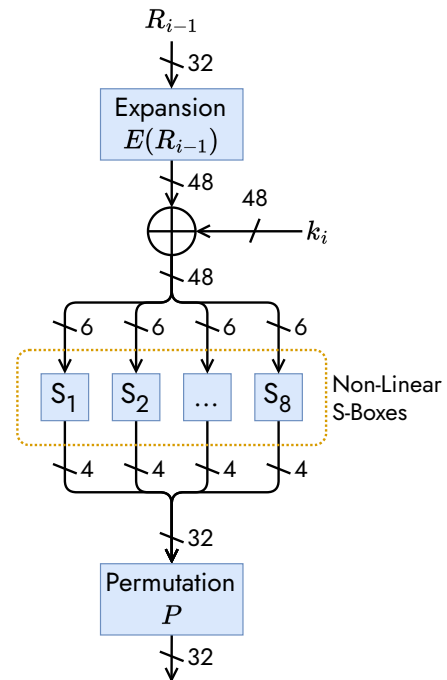
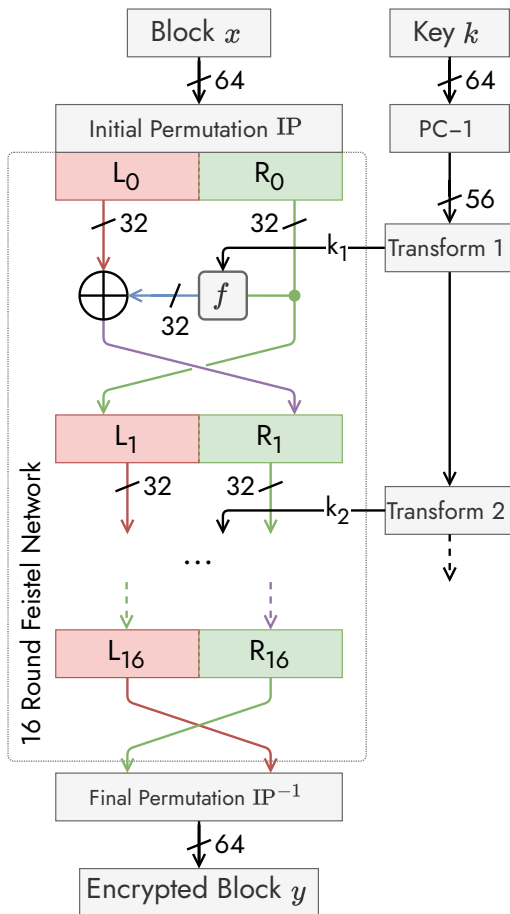
Ein Feistel Netzwerk wird zum Ver- und Entschlüsseln von Datenpaketen verwendet. Folgend ist ein symmetrisches Feistel Netzwerk \rightarrow Datenblock wird halbiert (64-Bit $\rightarrow 2 \times 32$ -Bit). Eine Runde entspricht:



Funktion f ist **wichtig**. Wenn diese sicher gegen Attacks ist, dann wird das Feistel Netzwerk mit jeder Runde und Key-Segment sicherer!



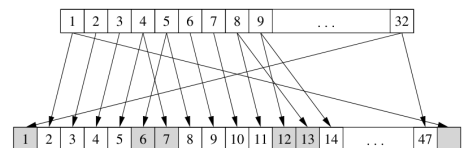
1. Data x wird mit einer *Initial Permutation* transponiert (**Diffusion**)
2. Diffusierte Data wird im Feistel Netzwerk **16x** verschlüsselt
3. Die verschlüsselte Data wird mit einer *Final Permutation* wieder transponiert (**Diffusion**)



1. Das Datenbyte R_{i-1} wird mit expandiert (Doppelzuweisung) um auf 48 Bit zu kommen. . .
2. . . danach wird dies mit dem Key **ver**xort und. . .
3. . . mit der jeweiligen Substitutionsbox (LUT) S_x verarbeitet.
4. Schlussendlich

Expansion E

Expansion E ist eine spezielle Permutationsfunktion. Die Expansion wird von 32-Bits zu 48-Bits *expandiert*.



Initial & Final Permutation

Vor und nach dem Feistel Netzwerk werden die Blöcke bitweise permutiert, also wie kreuzverdrahtet (Enigma Steckerbrett) → Bit **Diffusion**

Hinweis

Diese Permutationen sind in Hardware einfacher implementierbar als in der Software.

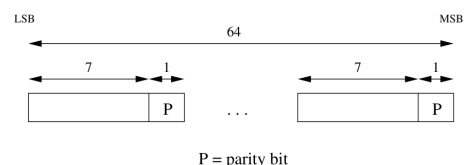
Wichtig

16-Bit der 32 Input-Bits kommen doppelt vor. **ABER** ein Input-Bit kommt **nicht** zweimal vor im selben 6-Bit Block → Diffusion wird verbessert, da gewisse Input-Bits zweimal vorkommen.

Key Scheduling Transform X

Der *Key Scheduler* generiert 16 Subkeys k_i vom Hauptkey k .

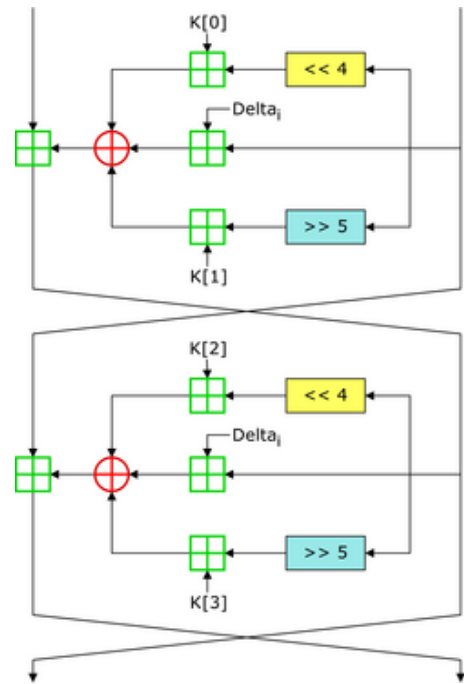
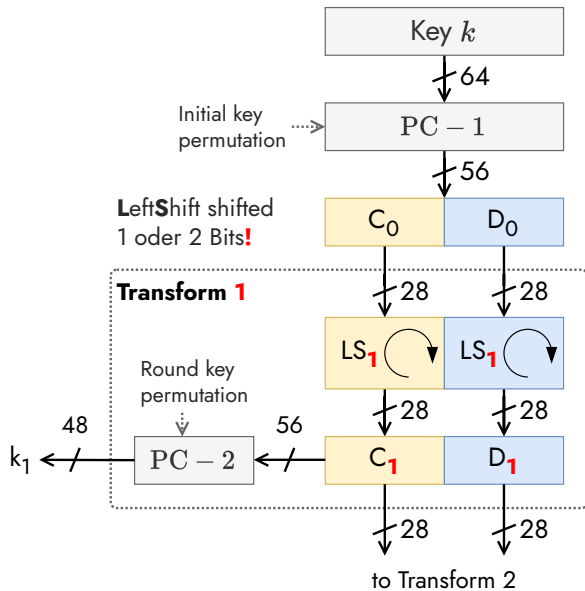
1. Der Key wird in $PC - 1$ auf 56-Bits gekürzt. Die *Parity* Bits 8, 16, 24, 32, 40, 48, 56 & 64 werden entfernt → sinnlose Bits



f -Funktion

Die f -Funktion vom DES ist das Herz des Algorithmus.

2. In Runden $i = 1, 2, 9, 16$ wird C_n **ein** Bit nach links geschiftet, ansonsten **zwei** Bits.
3. In $PC - 2$ werden erneut 8 Bits verworfen \rightarrow 48 Bit Subkey k_i



! (Fast) kein Feistel Netzwerk

Die verschlüsselten Blöcke können nicht so einfach wieder entschlüsselt, da im Algorithmus nebst XOR noch Additionen stattfinden. Für die Entschlüsselung werden die **Addition rückgängig** mit Subtraktionen gemacht.

XTEA

e**X**tended **TEA** ist eine Erweiterung von TEA, welcher die Verschlüsselung besser macht. Gleiche Eigenschaften + Konstantwert $\Delta = 0x9E3779B9$.

Das Coole

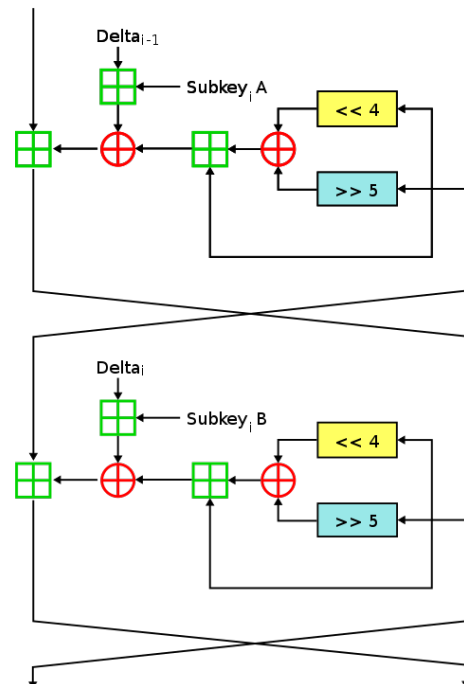
Der Cipher kann mit dem gleichen System wieder entschlüsselt werden. DES verwendet als Grundelement das Feistel Netzwerk, welches diese Eigenschaft hat.

Das Problem

DES ist wegen der kleinen 56 Bit Key nicht sicher (Fall: 1993 & 2008 Brute Force), was ja auch nid so doll is'. Ein Ansatz dafür ist den Key auf 112 Bits zu erweitern durch **triple DES** ($DES(k_1) + DES^{-1}(k_2) + DES(k_2)$).

TEA

Tiny **E**ncryption **A**lgorithm ist ein bereits geknackter und daher auch einfacher Verschlüsselungsalgorithmus. Er verwendet ein *Feistel Network*, 64 Bit Datenblöcke und einen 128 Bit Key. Es werden 32 Runden gemacht, damit die Informationen mehr verschlüsselt werden. Er erfordert sehr wenig Rechenleistung.



! XTEA sicher?

XTEA ist ein sicherer Verschlüsselungsalgorithmus, wenn auch nicht so sicher wie RSA oder andere.

AES

Cipher Modi W

Electronic Code Book (ECB)

Cyber Block Chaining (CBC)

Cipher FeedBack (CFB)

Output FeedBack (OFB)

Counter (CTR)

Galois Counter Mode (GCM)

Meet-In-The-Middle Attack

Double Encryption

Triple Encryption

Key Whitening

Hash Functions

Block

weitere Begriffe

Keyspace Anzahl möglichen & relevanten Keys

Brute Force Alle Keykombination versuchen (Erfolg \approx Keyspace/2)

Frequency Analysis Gewisse Zeichen(kombinationen) werden häufiger verwendet

Docker

“Dad why is my sisters name Rose?”
 “Because your Mother loves roses”
 “Thanks Dad”
 “No Problem”

DOCKER



! Wichtig

Bash Befehle via Host sind mit **\$** gekennzeichnet.

```
$ echo "this happens on the host"
```

Bash Befehle in einem Docker Container sind mit **#** gekennzeichnet.

```
# echo "this happens in a Docker container"
```

Was'n Docka?

Docker ist eine Plattform zur Software-Virtualisierung mit Fokus auf Wiederverwendbarkeit und Containerisierung.

Die Idee ist, eine Anwendung mit der nötigen Konfiguration, Runtime und Bibliotheken in ein Paket zusammenzustellen und dann als **portables** Produkt weitergegeben, verarbeitet, etc. ausgeführt werden.

Begriffe

Image

eine schreibgeschützte und vorgefertigte Vorlage, welche alle nötigen Software und Dateien beinhaltet. Es ist eine "Momentaufnahme" des Filesystems.

Container

ist die ausgeführte Instanz eines Images und ist eine isolierte Umgebung, welche die entsprechenden Prozesse ausführt, **ohne** andere Systeme zu stören.

Dockerfile

beschreibt wie ein Docker Image zusammengebaut wird anhand Schritten → welche Tools installiert und Dateien kopiert werden

Layers

Ein Image ist auf Layern aufgebaut, welche während einem Build-Prozess mit einem Dockerfile erstellt werden. Jede Anweisung im Dockerfile erzeugt einen neuen Layer mit einer eindeutigen ID.

Container Layer

Wenn ein Container kreiert und gestartet wird, entsteht ein Container Layer, welcher alle Änderungen verfolgt. Mit `commit` kann ein neues Image mit diesen Änderungen kreiert werden.

Docker daemon dockerd (Service)

Auch bekannt als *Docker Engine*. Zuständig für Ausführungen der Container und Docker Kommandos.

Volumes

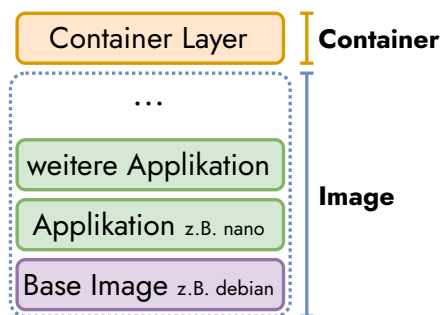
Dateien in Volumes bleiben nach Beendigung/Löschung des Containers erhalten (für Datenaustausch z.B. zwischen Host & Cont. oder seriell/parallel Cont. zu Cont.)

Docker Pfade

- `/var/lib/docker/`: Hauptverzeichnis von Docker
- `/var/lib/docker/images/`: enthält Metadaten zu den Images
- `/var/lib/docker/<overlay-driver>/`: enthält ausgepackte Layer (Images & Container)
- `/var/lib/docker/volumes/`: enthält Volumes

Images

Images sind **schreibgeschützte** Pakete, welches schichtenweise mit Software und Strukturen aufgebaut ist.



>_ Auflisten

Listet alle heruntergeladenen Images auf

```
$ docker images
$ docker image ls
$ docker image list
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
debian	latest	5027089adc4c	3 weeks ago	117MB

>_ Löschen

```
$ docker rm <image>
$ docker images rm <Image>
$ docker rmi <image>
```

Löscht ein Image (**WICHTIG** kein Container mit diesem Image sollte dabei existieren, sonst gehts nicht)

```
$ docker image prune
```

löscht *dangling* (nicht gekennzeichnete / Tag = none) Images. `-a` löscht alle Images, welche nicht verwendet werden

>_ Neue Version veröffentlichen

Nachdem eine Version kreiert wurde, kann diese auf DockerHub veröffentlicht werden.

```
$ docker image push [options] <name>[:tag]
```

>_ Grösse

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
alpine	latest	1d34ffeaf190	2 weeks ago	7.79MB
hello-world	latest	d2c94e258dcb	13 months ago	13.3kB

>_ Informationen abrufen

```
$ docker inspect <image>
```

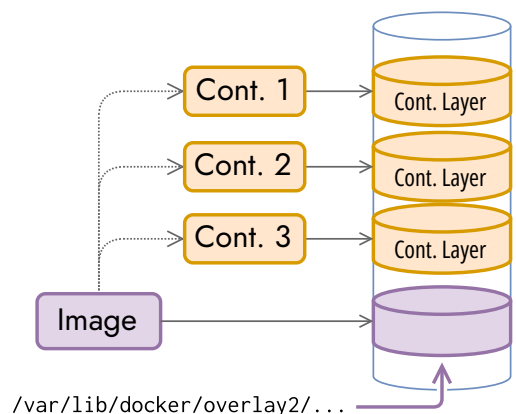
Container

Container sind ausgeführte, **isolierte** Images.

Container-Layer

Da das Image schreibgeschützt ist, werden alle Änderungen, Löschungen und Hinzufügungen am Image in der **Container-Layer** verfolgt → das Image bleibt heil und unversehen.

Dies bedeutet auch, dass das Image nur **einmal** pro Version existiert.



>_ Erstellen/Ausführen/Starten

```
$ docker run <opt> <image> <args> ①
$ docker create <opt> <image> <args> ②
$ docker start <container>
```

- ① Direkt erstellen & ausführen
- ② Container erstellen und dann ausführen

>_ Auflisten

listet alle **momentan** ausgeführten Container auf. Mit `-a` listet alle Container auf

```
$ docker ps # kurz für 'docker container ls'
CONTAINER ID   IMAGE     COMMAND   ... STATUS      ...
→ NAMES
79f3f8a71b46   debian   "bash"    ... Up 56 minutes ...
→ musing_wozniak
6201bc70ef8c   debian   "bash"    ... Up 58 minutes ...
→ magical_villani
```

>_ Attach/Detach

Ein Container mit einer Shell, z.B. Bash, kann via `attach` oder `-it` zugegriffen werden.

```
docker attach <container>      ①
docker run -it <image>         ②
```

- ① Bei bereits aktiven Container
- ② Die Shell des Containers wird an den Vordergrund gebracht + Pseudo Terminal

Mit `CTRL+Q CTRL+P` hängt man sich vom Container ab, ohne ihn zu beenden. `CTRL+C` oder im Shell `exit` beendet den Container.

>_ Interaktivität

Ein Container kann auf verschiedene Arten gestartet werden.

```
$ docker start <container>      ①
$ docker start -i <container>   ②
```

- ① im Hintergrund
- ② im Vordergrund

`-i/--interactive`

Macht den Container interaktiv und verbindet die Standardeingabe.

`-t/--tty`

Aktiviert einen Pseudo-Terminalsimulator für den Container (...:/# <cmd>).

```
$ docker run -i debian          ①
echo hello world
hello world

$ docker run -it debian         ②
root@containerID:/# echo hello world
hello world

$ docker create -t debian       ③
$ docker start -i <container>
```

- ① Interaktiv ohne TTY
- ② Interaktiv mit TTY
- ③ In Einzelschritten

>_ Löschen

```
$ docker rm <container> [container ...]
```

Es können nur *stopped* und *created* Container gelöscht werden.

💡 Löschen nach Beendigung

```
$ docker run --rm <image>
```

💡 Inaktive Container löschen

Löscht alle Container, welche nicht mehr verwendet werden.

```
$ docker container prune
```

>_ (Um-)benennen

Ein Name kann auf zwei Arten einem Container zugewiesen werden.

```
$ docker rename <old> <new>      ①
$ docker run --name peter_enis debian  ②
```

- ① Ein bereits existierender Container wird umbenannt
- ② Bei **Erstellung** eines Containers kann direkt ein Name zugewiesen werden

>_ Dateien kopieren Host ↔ Container

Der Docker Host kann Dateien in und aus dem Container kopieren.

```
$ docker container cp <container>:<path> <dest>  ①
$ docker cp <src> <container>:<dest>             ②
```

- ① kopiert dateien vom Container zum Host
- ② kopiert vom Host zum Container

>_ Ausgabe

Zeigt Log Daten (z.B. Konsolenausgabe) während der Ausführung an (`-f` folgt dem Container/ kontinuierliches Update)

```
$ docker logs <container>
```

>_ Prozesse ausführen/abfragen

Führt Prozesse in einem aktiven Container aus.

```
$ docker exec <container> <cmd>
```

Gibt Prozesse des Containers an (ax als Argument gibt alle laufenden Prozesse an)


```
$ docker top <container> <arg>
$ docker container top <container> <arg>
```

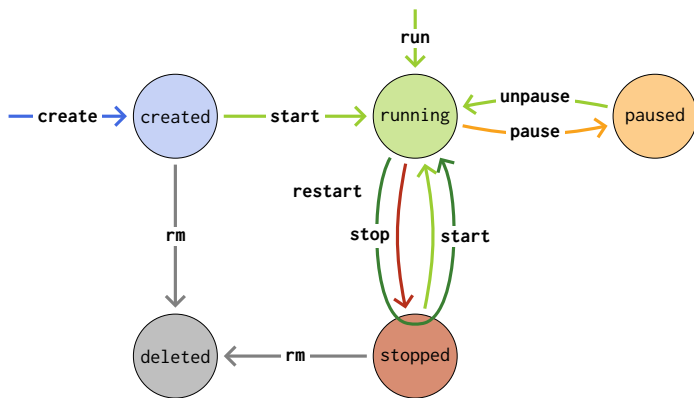
>_ Logging

Zeigt Log Daten während Ausführung an (-f folgt dem Container/ kontinuierliches update).

```
$ docker logs <container>
```

Lebenszyklus

Ein Docker-**Container** kann fünf Zustände annehmen (Created, Running, Deleted, Stopped und Paused) und kann mit folgenden Docker-Befehlen gesteuert werden.



i Unterschied *Stopped* und *Paused*

Wenn ein Container gestoppt wird, werden alle ihm zugewiesenen Ressourcen freigegeben, während bei einem angehaltenen Container kein Speicher, aber die CPU freigegeben wird.

>_ Grösse --size

Da ein Container Änderungen auf einer separaten Layer verfolgt, ist der Speicherplatz selbst meistens klein.

```
$ docker ps -a --size
CONTAINER ID   IMAGE      ... NAMES      SIZE
253be0a7fe55   alpine    ... brave_pascal  8B
↳ (virtual 7.79MB)
68bb3a7fdf37   hello-world ... loving_mcclintock 0B
↳ (virtual 13.3kB)
d20695e9fe4b   alpine    ... unruffled_tesla 25B
↳ (virtual 7.79MB)
```

virtual referenziert auf Container + Image = Totale theoretische Grösse.

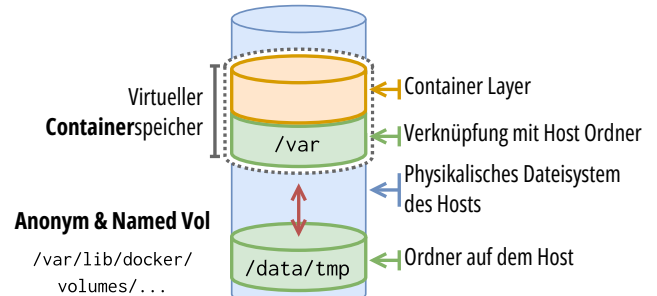
Image kreieren aus Container

Aus einem Container kann ein neues Image erstellt werden:

```
$ docker container commit <container> [image[:tag]]
$ docker commit ...
```

Volume

Container sind voneinander isoliert → Datenaustausch zwischen Host ⇔ Container & Container ⇔ Container wird mit **Volumen** gemacht. Dies muss **explizit** angegeben werden.



Mit dem Parameter **-v** und einem Pfad wird ein Volumen angegeben.

Host Volumes

```
$ docker run -v /path/in/host:/path/in/container
↳ <opt> <image>
```

- /path/in/host: Pfad auf dem Host
- /path/in/container: Pfad im Container, welche mit dem Host-Pfad verbunden wird.

Named Volumes

```
$ docker run -v name:/path/in/container <opt> <image>
```

- name: Name des Volumens
- /path/in/container: Pfad im Container, welche geöffnet wird.

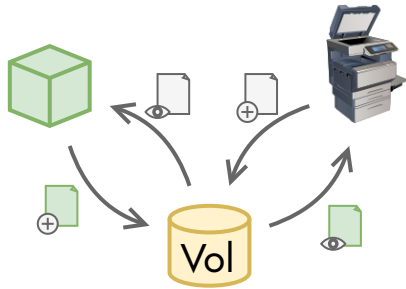
Anonyme Volumes

```
$ docker run -v /path/in/container <opt> <image>
```

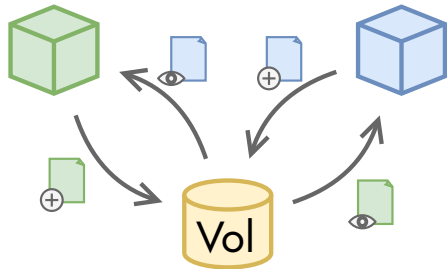
- /path/in/container: Pfad im Container, welche geöffnet wird.

Datenaustasch

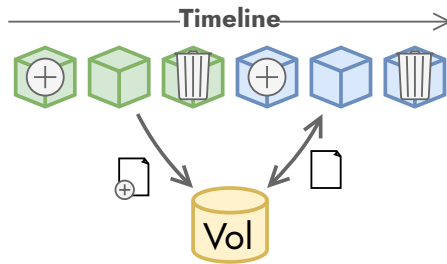
① Host & Container



② Container & Container (parallel)



③ Container nach Container (seriell)



Speicherpfade

Benannte & anonyme Volumes werden unter dem Pfad `/var/lib/docker/volumes/` angelegt.

Host Volumes werden an einem vom Host festgelegten Pfad angelegt.

>_ Auflisten

```
$ docker volume ls
```

>_ Volumes löschen

Um ein Volume zu löschen, muss der Name des Volumes angegeben werden. Ebenfalls darf es von keinem Container verwendet werden.

```
$ docker volume rm <volume>
```

`prune` löscht nicht gebrauchte anonyme Volumes. Mit `-a/--all` können ungebrauchte anonyme und benannte Volumes gelöscht werden.


```
$ docker volume prune
```

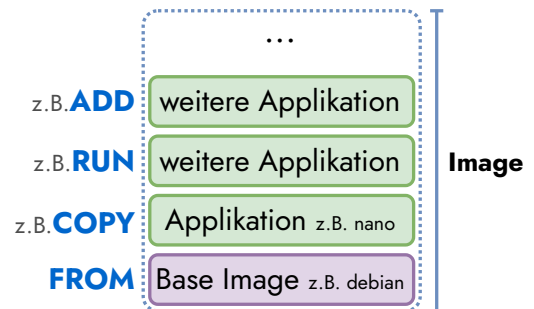
>_ Erstellen

Mit dem `volume create` Befehl können anonyme und benannte Volumes erstellt werden. Einfach keine Pfad/Host Volumes.

```
$ docker volume create [name]
```

Dockerfile

Mit einem  Dockerfile kann ein Image erstellt werden anhand Anweisungen. Jede Anweisung erzeugt eine eigene Layer.



Anweisungen

```
FROM <image> # image oder 'scratch'
```

Mit der Anweisung `FROM` setzt man das Parent- oder Basis-Image. Es ist die erste Anweisung in einem Dockerfiles. Es können mehrere `FROM` verwendet werden, wenn man Multistage-Building machen möchte.

```
COPY [OPTIONS] <src> ... <dest>
COPY [OPTIONS] ["<src>", ... "<dest>"]
```

Kopiert die Quelle zur Destination. Wenn die Destination ein Ordner ist, können mehrer Dateien kopiert werden. Mit dem Wildcard-Zeichen `*` und Wildcard-Einzelzeichen `?` können mehrere ähnliche Dateien auf einmal kopiert werden.

```
COPY hom*.txt /mydir/
# sammelt alle Dateien z.B. 'home.txt' und 'homie.txt'

COPY hom?.txt /mydir/
# findet 'home.txt', aber nicht 'homie.txt'
```

```
RUN [OPTIONS] <command> ...
RUN [OPTIONS] [ "<command>", ... ]
```

Führt Befehle im Image aus, z.B. `RUN apt-get update && apt-get install -y curl`. Mit `\` kann der Befehl auf mehrere Zeilen gebrochen werden.

```
ADD [OPTIONS] <src> ... <dest>
ADD [OPTIONS] ["<src>", ... "<dest>"]
```

Ähnlich wie COPY, einfach können URLs angegeben werden und gezippte Dateien werden automatisch entpackt.

Startkommando

Die Startkommandos eines Container können auf zwei Arten gesetzt: CMD und ENTRYPOINT.

```
ENTRYPOINT ["executable", "param1", "param2"]
ENTRYPOINT command param1 param2
```

```
# wenn ENTRYPOINT nicht gesetzt ist, wird CMD als
→ Startkommando verwendet.
CMD ["executable", "param1", "param2"]

# als default Parameter zu ENTRYPOINT
CMD ["param1", "param2"]
```

ENTRYPOINT	CMD	[command]	ausgeführt wird
["script.sh"]			script.sh
["script.sh"]		/bin/bash	script.sh /bin/bash
["script.sh"]	["mysqld"]		script.sh mysqld
["script.sh"]	["mysqld"]	/bin/bash	script.sh /bin/bash
	["/bin/sh"]		/bin/sh
	["/bin/sh"]	/bin/bash	/bin/bash

```
$ docker run [opts] <image> [command] [args...]
$ docker run --rm -it debian ls -la # startet "ls -la"
→ im Container
```

Um die Startkommandos eines Images anzuzeigen

```
$ docker inspect -f
→ 'ENTRYPOINT:{{.Config.Entrypoint}}';
→ CMD:{{.Config.Cmd}}' <image>
"ENTRYPOINT:[]; CMD:[bash]"

$ docker inspect -f
→ 'ENTRYPOINT:{{.Config.Entrypoint}}';
→ CMD:{{.Config.Cmd}}' kaohslu/01-demo-img
"ENTRYPOINT:[dotnet ASYD_Demo.dll]; CMD:[]"

WORKDIR /path/to/workdir
```

Mit WORKDIR wird der Arbeitspfad gesetzt (ab Aufruf der Anweisung), wo Befehle wie COPY oder ADD ihre Arbeit verrichten.

Image builden

```
$ docker build <path> -t <name>:<tag>
$ docker build -t <name>:<tag> <path>
```

Image Build History zeigen

Um die History eines Images anzuschauen → Wie das Image erstellt wurde.

```
$ docker history <image>
```

Multistage

Mit Multistage Building können komplexe Images erzeugt werden.

```
# syntax=docker/dockerfile:1
FROM golang:1.21
# FROM golang:1.21 as build
WORKDIR /src
COPY <<EOF ./main.go
package main

import "fmt"

func main() {
    fmt.Println("hello, world")
}
EOF
RUN go build -o /bin/hello ./main.go

FROM scratch
COPY --from=0 /bin/hello /bin/hello
# COPY --from=build /bin/hello /bin/hello
CMD ["/bin/hello"]
```

Netzwerke

Da Container per default isoliert sind (laufen in einem privaten Netzwerk, welches sich Docker kümmert), muss man z.B. Netzwerke **explizit** nach aussen öffnen, wenn man das will.

[TODO?]

Docker Hub

Images suchen & herunterladen

```
$ docker search <searchterm>

$ docker pull <image>
```

Image Reference Format

Standardmässig werden Images wie z.B. [hello-world](#) immer vom [DockerHub-registry](#) heruntergeladen, aber es ist möglich andere **repos** anzufragen. Es gilt folgendes Format:

<repo>/<source>/<image>/<tag>

- **<repo>**: Repository/Content-Host (default index.docker.io)
- **<source>** Untergruppe, Hauptprojekt, User, Organisation, etc. (default library)
- **<image>** Projekt, wie z.B. eine Runtime
- **<tag>** Version oder Tag des Projektes (default latest)

```
$ docker run kaohslu/01-demo-img:latest
```

→ Auf dem offiziellen Repository **DockerHub** wird unter dem User **kaohslu** das Image **01-demo-img** der Version **latest** heruntergeladen und gestartet.

Performance

Cache

i Was Cache?

Trashing

Thrashing / Seitenflattern

Cache Struktur

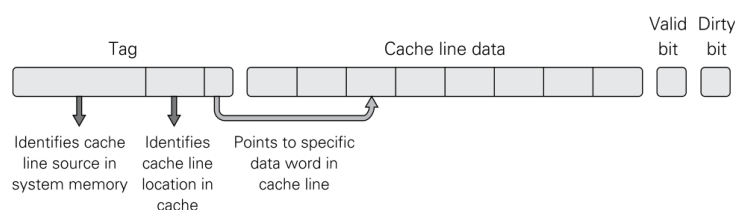


Abbildung 0.1: Struktur einer Cache-Zeilen

GCC Optimization

-O0 no optimization -O3 all optimization (-O1,-O2) + function inlining and more

funroll-loops

💡 Enabling Optimization

```
#pragma GCC optimize ("O0")
```

Safety

Terms

Hazard
A hazard is a situation in which there is actual or potential danger to people or the environment.

Accident
An accident is an unintended event harming people or the environment.

Incident
An incident (or near miss) is an unintended event which does not harm, but has the potential to do so.

Risk
To each hazard, the risk describes the likelihood of occurrence and the likely consequences.

Fault
A fault is a defect within the system. Faults can be categorized into random faults and systematic faults.

Error
An error is a deviation from the required operation of the system or subsystem.

System Failure
A system failure occurs when the system fails to perform its required function.

Causalities
The presence of a fault may lead to an error, which may lead to a system failure, which may lead to an accident.

Requirements

Integrity / Dependability
The *integrity / dependability* is a property of a system that justifies placing one's reliance on it.

- This demands:**
- 1. *Safety* is a property of a system that it will not endanger human life or the environment.
 - 2. *Reliability* is the probability of a system functioning correctly over a given period of time under a given set of operating conditions.
 - 3. *Availability* describes the probability that a system will be functioning correctly at a given time.
 - 4. *Maintainability* it the ability of a system to be maintained.

V&V&C

Verification
Verification is the process of determining that a system, or module, meets its specification.

Validation
Validation it the process of determining that a system is appropriate for its purpose.

Certification
Certification it the process of convincing a regulatory bodies about a systems properties.

Computers in Safety Related Systems?

- Advantages**
- + modern digital devices are extremely reliable
 - + high speed, low power, small physical size
 - + high flexibility, adaptability
 - + sophisticated strategies possible (including e.g. diagnostic)

- Disadvantages**
- complexity, complexity, complexity
 - number of possible states to be considered as infinite
 - bad predictability due to number of states (i.e. possible failure modes)
 - exhaustive testing not possible, detection of failures is unreliable

Silver Bullet

There is good evidence that better processes lead to programs with fewer defect. Some numbers in relation to the CMM (Capability Maturity Model) level defined by Software Engineering Institute (SEI) according to:

CMM Level	Focus	Defects / 1000 LOC
1	None	7.5
2	Project Mngt.	6.2
3	Software Eng.	4.7
4	Quality Processs	2.3
5	Cont. Improvement	1.1

Brooks argues that **“there is no single development, in either technology or management technique, which by itself promises even one order of magnitude [tenfold] improvement within a decade in productivity, in reliability, in simplicity.”** He also states that **“we cannot expect ever to see two-fold gains every two years”** in software development, as there is in hardware development (Moore's law).

Hazard Analysis

⚠️ How to identify the ways in which a system can cause harm?

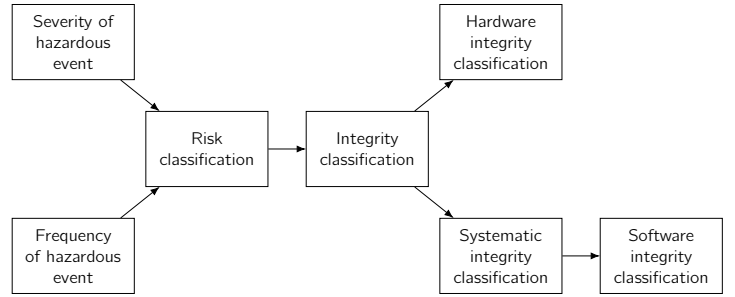
FMEA: Failure mode and effects analysis

Consider the failure of any component within a system and track the effects of this failure to determine its ultimate consequences.

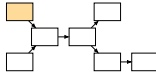
HAZOP: Hazard and operability studies

Use a series of 'guide words' to investigate the effects of deviations from normal operating conditions.

Guide Word	Deviation	Causes	Consequences	Action
NO	No cooling	Cooling water valve malfunction	Temperature increase in reactor	Install high temperature alarm (TAH)
REVERSE	Reverse cooling flow	Failure of water source resulting in backward flow	Less cooling, possible runaway reaction	Install check valve
MORE	More cooling flow	Control valve failure, operator fails to take action on alarm	Too much cooling, reactor cool	Instruct operators on procedures
AS WELL AS	Reactor product in coils	More pressure in reactor	Off-spec product	Check maintenance procedures and schedules
OTHER THAN	Another material besides cooling water	Water source contaminated	May be cooling ineffective and effect on the reaction	If less cooling, TAH will detect. If detected, isolate water source. Back up water source?



Severity
How severe is an accident?



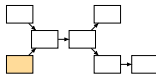
ETA: Event tree analysis

Take the events that can affect the system as starting point and track them forward to determine their possible consequences.

Flow regulator fails open w=0.02	Alarm response Q=0.00087	SIS Loop Q=0.0001656	Pressure relief valve fails closed Q=0.01518	Consequence	Frequency
Failure	Success	Null	Null	Not set	0.01898
	Failure	Success	Null	Not set	0.001017
	Failure	Failure	Success	Minor release	5.925E-07
	Failure	Failure	Failure	Major release	9.182E-09

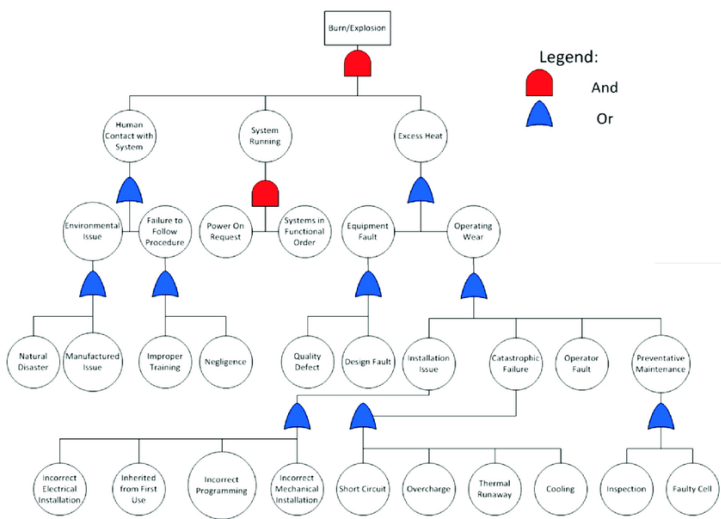
Category	Definition
Catastrophic	Multiple deaths
Critical	Single death, and/or multiple severe injuries or severe occupational illnesses
Marginal	Single severe injury or occupational illness, and/or multiple minor injuries or minor occupational illnesses
Negligible	Single minor injury or minor occupational illness at most

Frequency
How frequent does it occur?



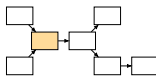
FTA: Fault tree analysis

Start with all identified hazards and work backwards to determine their possible causes. (Reverse to ETA)



Category	Definition	Range (events per hour)
Frequent	Many times in system lifetime	$> 1 \times 10^{-3}$
Probable	Several times in system lifetime	$1 \times 10^{-3} \dots 1 \times 10^{-4}$
Occasional	Once in system lifetime	$1 \times 10^{-4} \dots 1 \times 10^{-5}$
Remote	Unlikely in system lifetime	$1 \times 10^{-5} \dots 1 \times 10^{-6}$
Improbable	Very unlikely to occur	$1 \times 10^{-6} \dots 1 \times 10^{-7}$
Incredible	Cannot believe that it could occur	$< 1 \times 10^{-7}$

Risk
What is the risk associated?



Risk Analysis

! Fundamental Rule

Risk = Severity × Frequency

Frequency	Consequence			
	Catastrophic	Critical	Marginal	Negligible
Frequent	I	I	I	II
Probable	I	I	II	III
Occasional	I	II	III	III
Remote	II	III	III	IV
Improbable	III	III	IV	IV
Incredible	IV	IV	IV	IV

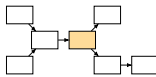
I Intolerable

II Undesirable, tolerable only if risk reduction is impracticable

III Tolerable

IV Negligible

Integrity
Is the risk acceptable?



ALARP-Rule

A tolerable risk (class II & III) is acceptable only if it is as low as reasonably practicable.

Risk reduction

Risks can be reduced by means of safety features. The reduction achieved depends upon the integrity of these features.

Safety integrity

Safety integrity is the likelihood of a safety-related system satisfactorily performing the required safety functions under all the stated conditions within a stated period of time.

Integrity Levels

What failure rate is tolerable?

Safety Integrity Level	Continuous mode (prob. of dangerous failure per year)	Demand mode (prob. of failure to perform on demand)
4	$\geq 1 \times 10^{-5} \dots 1 \times 10^{-4}$	$\geq 1 \times 10^{-5} \dots 1 \times 10^{-4}$
3	$\geq 1 \times 10^{-4} \dots 1 \times 10^{-3}$	$\geq 1 \times 10^{-4} \dots 1 \times 10^{-3}$
2	$\geq 1 \times 10^{-3} \dots 1 \times 10^{-2}$	$\geq 1 \times 10^{-3} \dots 1 \times 10^{-2}$
1	$\geq 1 \times 10^{-2} \dots 1 \times 10^{-1}$	$\geq 1 \times 10^{-2} \dots 1 \times 10^{-1}$

Example – Parachute

Hazard of free fall, severity rated as catastrophic with occasional frequency. Therefore risk is intolerable and a parachute is selected as safety feature. It works on demand mode and has to reduce the frequency by 2 orders of magnitudes, so SIL 2 is required at least.

! Distinguish!

Risk is a measure of the likelihood, and the consequences, of a hazardous event. Safety integrity is a measure of the likelihood of the safety system correctly performing its task.

Allocating Levels

What is contributing?

Hardware integrity is that part of the safety integrity relating to dangerous **random** hardware failures.

Systematic integrity is that part of the safety integrity relating to dangerous **systematic** failures.

Software integrity is that part of the safety integrity relating to dangerous **software** failures.

Design for Safety

General, Iterative Design Process

- Abstraction** Generlization & ID of essentials
- Decomposition** Objects into smaller parts + Analysis
- Elaboration** detailing & adding features
- Decision** identification & selection of alternatives

Types of Fault

What types of fault may occure?

NATURE

- Random (HW)
- Systematic
 - Specification
 - HW Design
 - SW Design

DURATION

- Permanent – most HW faults, design
- Transient – e.g. α particles
- Intermittent – e.g. contacts, interference (EMC)

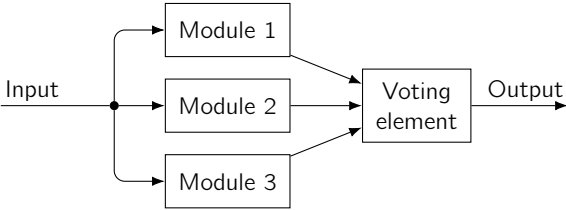
EXTEND

- Localized – affecting only part of the system
- Global – effects which permeate throughout the system

Fault Tolerance

Triple Modular Redundancy (TMR)

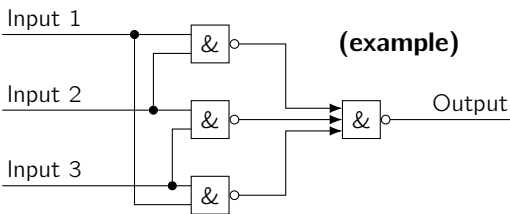
Three identical moduls get fed by the same signal. A voter compares the results and produces an output corresponding to the majority view.



- + simple
- + prevents from failure of a single component, i.e. *single-point failure*
- leaves input and voter as sources for single-point failures
- does not prevent from systematic failures
- voter is dependable

Voter

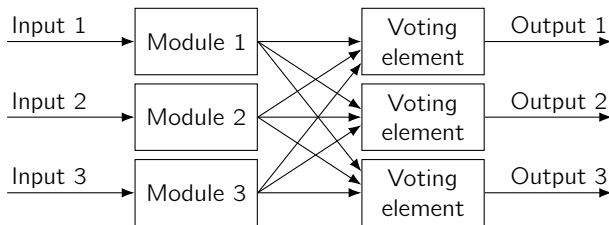
To make voting unit reliable, keep it as simple as possible.



- + simple, low complexity
- + high reliability
- no indication in case of discrepancies

TMR with triple voting

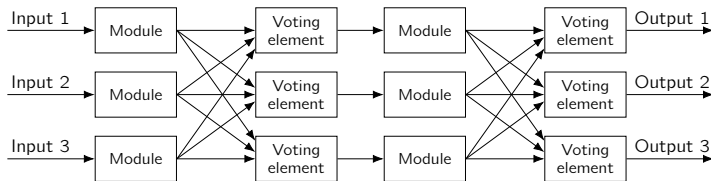
Use triple input signals and triple voting instances.



- + all outputs are correct in case a single module fails
- more components required
- no protection against simultaneous failure of two or more modules
- does not prevent from systematic failures

Multistage TMR

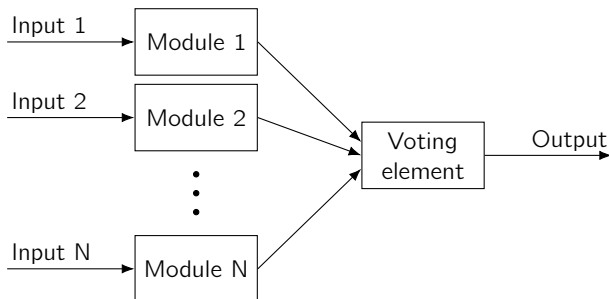
Cascading TMRs with triple voters can deal with failed voting unit.



- + allows a single module to fail at each level
- + allows a single voter to fail at each level
- no protection against simultaneous failure of two or more units at same level
- does not prevent from systematic failure

NMR - N Modular Redundancy

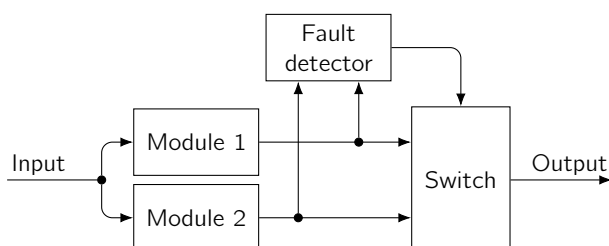
Use a large (odd) number of modules to increase ability to withstand failures.



- + allows $(N-1)/2$ modules to fail
- higher complexity of voter
- higher cost, size, power consumption

Dynamic Redundancy

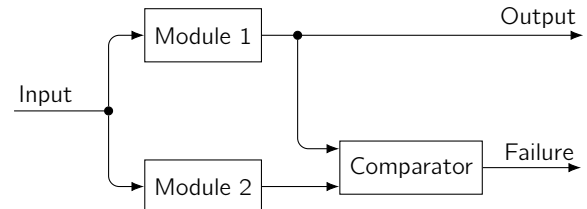
While no fault is detected, one module drives the output. In case of a fault, a switch reconfigures the system such that the output is taken from a 'standby spare' module.



- + allows one modules to fail
- + gives indication of fault
- fault detector required (single-point failure!)
- + either *hot standby* or *cold standby* possible
- + can be extended to N modules

Self Checking Pair

The output of two identical modules are compared to give an indication of failure.



- + simple, reliable
- + gives indication of fault
- no redundancy

Redundancy Combining

- STATIC** Voting to produce *fault masking* at the cost of large amount of redundancy.
- DYNAMIC** Fault detection and some form of switching, but **do not** mask faults.
- HYBRID** Combination of voting, fault detection and module switching
→ Most reduced down to *N-modular redundancy with spares*.

Software Faults

Software faults are systematic by nature, duplicating the systems gives therefore no protection from faults.

- N-Version Programming** Same function gets implemented differently with the same specifications. ($N = 3$ for Airbus, $N = 4$ for Space Shuttle)
- Recovery Blocks** When a module fails, it induces (triggers) the execution of a secondary implementation of the same module.

Reliability

(Un-)Reliability $R(Q)$

Reliability R is the probability of a component/system functioning correctly over a given period of time and a given set of operating condition.

$$R(t) = \frac{n(t)}{N}$$

N : Amount of identical components

$n(t)$: expected number of components operating correctly at some time t

The **un**reliability Q defines how likely it is that the system/component will break.

$$Q(t) = \frac{n_f(t)}{N} = 1 - R(t)$$

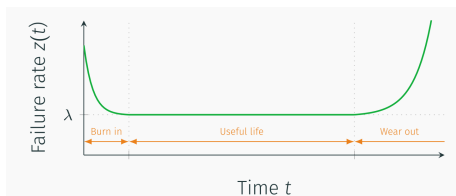
$n_f(t)$: expected number of malfunctioning components at some time t .

Failure Rate $z(t)$

Is the rate at which a device fails. Number of devices failing within a given period of time as a fraction of the devices still functioning.

$$z(t) = \frac{1}{n(t)} \cdot \frac{d n_f(t)}{dt}$$

Bathtub Curve



Burn in high 'infant mortality' due to manufacturing faults.

Useful life the failure rate takes in a fairly constant level λ .

Wear out ageing becomes apart and the failure rate rises.

Useful-Life

If the failure rate is constant, $z(t) = \lambda$,

$$z(t) = \lambda = \frac{1}{n(t)} \cdot \frac{d n_f(t)}{dt}$$

with $n_f(t) = N - n(t)$ we get the differential equation

$$\lambda n(t) = \frac{dN - n}{dt} = -\frac{dn}{dt}$$

with solution for $R(t) = n(t)/N$

$$R(t) = \exp(-\lambda \cdot t)$$

Time-Variant Failure Rates

For software failures, which are systematic and therefore correctable after detection, the failure rate decreases with time. The reliability resulting can be modelled by the *Weibull* distribution:

$$R(t) = \exp(-(t/\eta)^\beta)$$

β : shape parameter

η : characteristic life

Mean times

Mean Time to Failure

❓ What is the expected time that a system will operate before the first failure occurs?

$$MTTF = \int_0^\infty R(t) dt \rightarrow \int_0^\infty \exp(-\lambda \cdot t) dt = \frac{1}{\lambda}$$

🔥 Beware!

A system with $\lambda = 0.001$ failure/h does have a MTTF of 1000 hour. But the reliability at $t = 1000$ hour is $R(t) = e^{-\lambda \cdot t} = e^{-1} \approx 0.37$. Chances that any given system runs for 1000 hour are only $\approx 37\%$!

! Reliability vs. MTTF

Reliability

a function of time, depends on the time for which the system must operate.

MTTF

fixed characteristic that does not change with time.

Mean Time to Repair

❓ What is the average time required to repair a system that has failed?

$$MTTR = \frac{1}{\mu} \quad MTBF = MTTF + MTTR$$

$MTTR$: Mean time to repair

$MTBF$: Mean time between failures

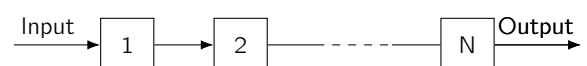
Failure in time

❓ How many failures are to be expected?

Failure in time (FIT) is the number of failures that can be expected in 1×10^9 h of operation.

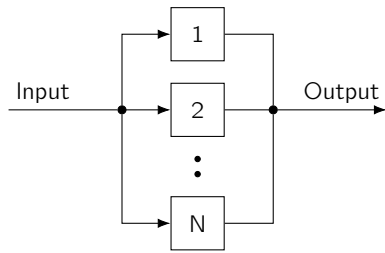
$$FIT = 1 \times 10^9 \cdot \frac{1}{MTBF}$$

Reliability modelling



$$R(t) = R_1(t) \cdot R_2(t) \cdot \dots \cdot R_N(t) = \prod_{i=1}^N R_i(t)$$

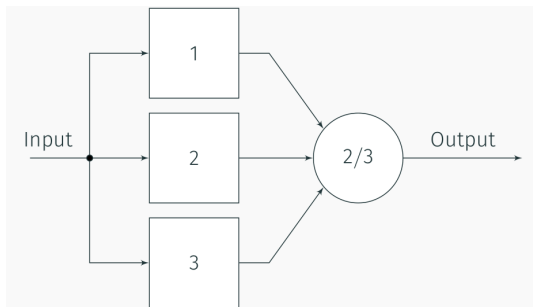
$$\lambda = \lambda_1 + \lambda_2 + \dots = \sum_{i=1}^N \lambda_i$$



$$R(t) = 1 - Q(t) = 1 - \prod_{i=1}^N (1 - R_i(t))$$

$$Q(t) = Q_1(t) \cdot Q_2(t) \cdots Q_N(t) = \prod_{i=1}^N Q_i(t)$$

Triple Modular Redundancy

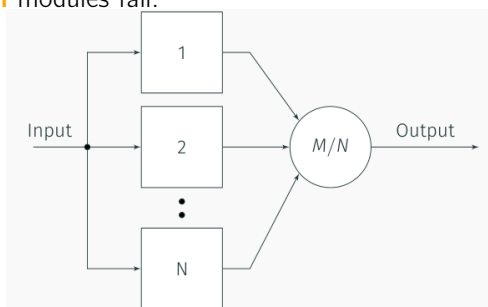


$$R_{TMR} = R_1(t) \cdot R_2(t) \cdot R_3(t) + (1 - R_1(t)) \cdot R_2(t) \cdot R_3(t) + (1 - R_2(t)) \cdot R_1(t) \cdot R_3(t) + (1 - R_3(t)) \cdot R_1(t) \cdot R_2(t)$$

For $R_1 = R_2 = R_3 = R_m$: $R(t) = 3 \cdot R_m^2(t) - 2 \cdot R_m^3(t)$

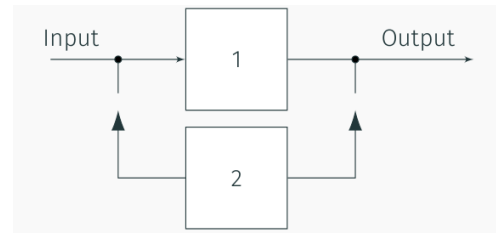
! N-Modular Redundancy

M out of N voting: System works correctly as long as **less than M** modules fail.



$$R_{MtoN}(t) = \sum_{i=0}^{N-M} \frac{N!}{(N-i)! \cdot i!} \cdot R_m^{N-i}(t) \cdot (1 - R_m(t))^i$$

Dynamic Redundancy

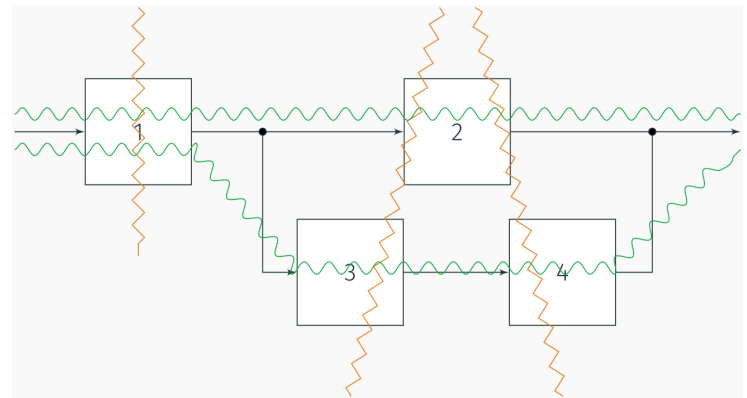


$$R(t) = R_1(t) + (1 - R_1(t)) \cdot C_1 \cdot R_2(t) = R_m(t) + (1 - R_m(t)) \cdot C_m \cdot R_m(t)$$

R_n : Module Probabilities

C_n : Fault Coverage

Cut and Tie Sets



~ cut : sets of simultaneous failures leading to a system failure

~ tie : sets of working modules guaranteeing a working system

$$1 - \sum_{j=1}^{N_c} \prod_{i=1}^{n_j} (1 - R_i(t)) \leq R(t) \leq \sum_{j=1}^{N_T} \prod_{i=1}^{n_j} (1 - R_i(t))$$

Reliability prediction

Resistor (DoD MIL-Handbook 217)

$$\lambda_p = \lambda_b \cdot \pi_R \cdot \pi_Q \cdot \pi_E \quad [\text{failure}/1 \times 10^6 \text{h}]$$

λ_b : Base rate, temperature ($0.7 \times 10^{-3} \dots 6.5 \times 10^{-3}$)

π_R : Resistance range (1.0 ... 2.5)

π_Q : Quality of manufacturing (0.03 ... 15)

π_E : Environment (1 ... 490)

λ_p : Per part ($0.21 \times 10^{-3} \dots 119.4$)

Capacitor (DoD MIL-Handbook 217)

$$\lambda_p = (C_1 \cdot \pi_T + C_2 \cdot \pi_E) \cdot \pi_Q \cdot \pi_L \quad [\text{failure}/1 \times 10^6 \text{h}]$$

C_1 : Die complexity
 C_2 : Packaging
 π_T : Ambient temperature
 π_E : Environment
 π_Q : Quality
 π_L : Learning (production)

Prediction of software reliability

Task: estimate the number of faults within a given piece of software.

Prediction of software reliability

In general, this is a difficult task and still an active field of research.

- Assessment according to the techniques used during development and test.
- Rate the testing scheme used. Therefore, make minor changes (mutations) to the code and check their detection.
- Base upon experience from past projects.
- Estimate according to located faults located per time.

Number of faults \propto unreliability?

It is *not* given that a program with more faults is less reliable than one with fewer faults!

Reliability assessment

Task: demonstrate that a system meets its reliability requirement.

How to ...

... proof that a system fails less then once in 1×10^9 hour (i.e. ≈ 100000 year) of operation?
Trust the development techniques.

Software - Formal Methods

Examples

- **B-Method** – abstract machine notation, became Event-B, Rodin as tool
- **ACSL** – ANSI/ISO C specification language, Frama-C as tool
- **Esterel** – synchronous programming language, generates C code
- **Z notation** – specification language
- **SPIN** – model checker basing on Promela language
- **SPARK** – refinement of Ada

Spark

Guarantees

SPARK analysis can give strong *guarantees* that a program:

- does not read uninitialized data,
- accesses global data only as intended,
- does not contain run-time errors,
- respects key integrity properties,
- is a correct implementation of requirements.

Level of adoption

- Stone – valid SPARK subset of Ada
- Bronze – initialization and correct data flow
- Silver – absence of run-time errors (AoRTE)
- Gold – proof of key integrity properties
- Platinum – full functional proof of requirements

```
procedure Increment (X : in out Integer)
with
    Global => null,
    Depends => (X => X),
    Pre => X < Integer'Last,
    Post => X = X'Old + 1
is
begin
    X := X + 1;
end Increment;
```

①

②

③

④

- ① **Global:** does not read or write any global variables
- ② **Dependance:** Value of **X** after call depends on the (previous) value of **X**
- ③ **Condition:** Increment only callable if $X \leq \text{max value}$
- ④ **Conformance:** Check if the function does actually produce the desired result.

Vorsicht

This check is done by *Spark*, **NOT** *Ada*.
These properties are not only declared, but these are **proved** by a dedicated proof-engine GNATprove!

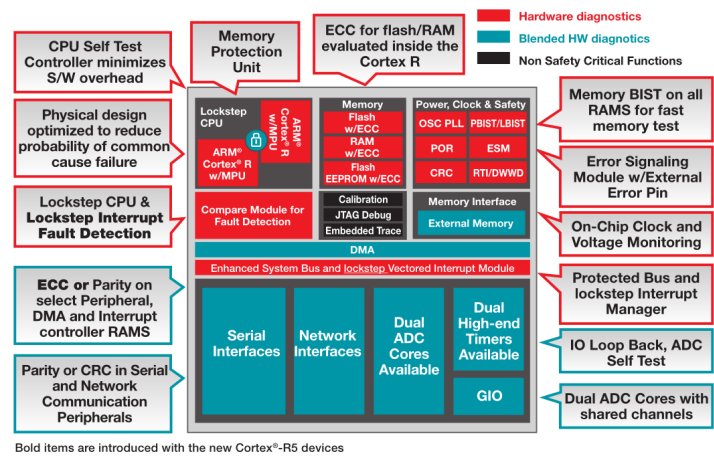
Hardware - Safety Processors

Hercules RM42 MCU

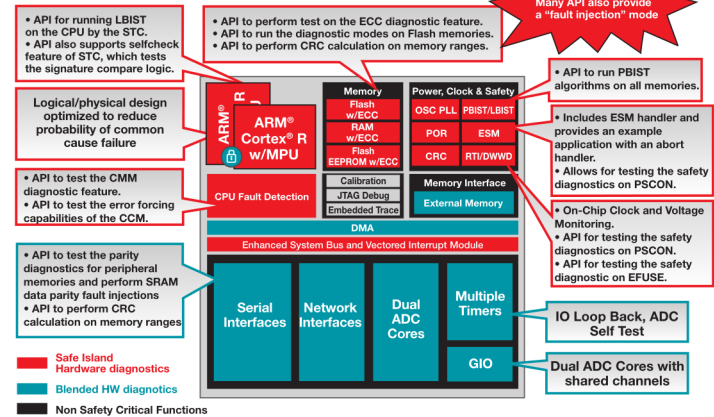
Safety features

- dual ARM Cortex-R CPU
- cycle-to-cycle lockstep operation
- error correction code (EEC) circuit within CPU
- SRAM/Flash single bit error correction and double bit error detection (SECEDED)
- CPU logic built-in self test (LBIST)
- SRAM programmable built-in self test (PBIST)
- lockstep interrupt manager
- advanced clock and voltage monitoring
- error signaling module with dedicated error pin

Hercules™ MCU safety features



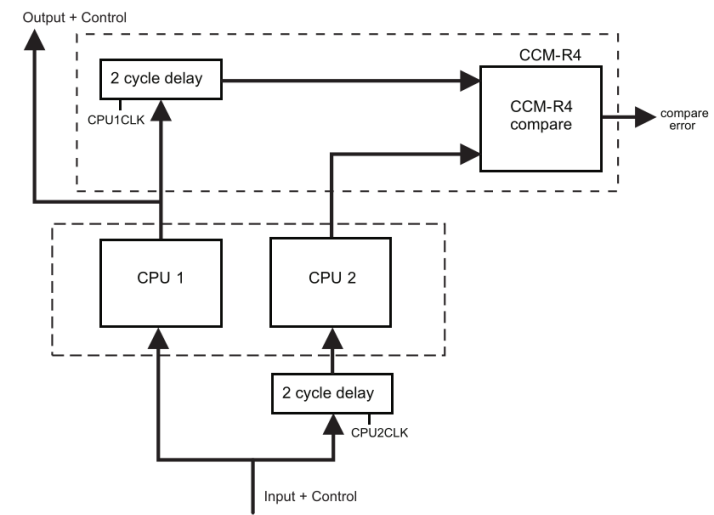
Hercules™ MCU safety features and SafeTI™ Diagnostic Library



Dual CPU

Lockstep operation

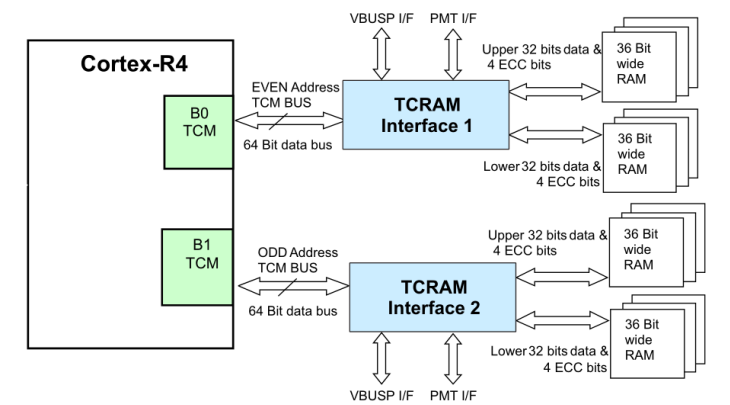
- two CPU, diverse placement ('north' & 'flip-west')
- operation shifted by two clock-cycles
- independent but synchronized clock sources



Memory

Tightly coupled RAM (TCRAM)

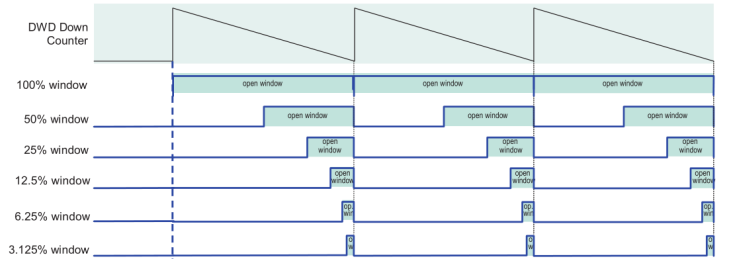
- 64-bit data and 8-bit ECC code (modified Hamming code)
- two 36-bit wide byte-interleaved RAM banks
- stores addresses for single-bit and multibit errors
- CPU address bus integrity checking
- redundant address decoding for chip select



Watchdog

Digital windowed watchdog (DWWD)

- configurable end time window
- reset by writing of correct sequence
- once enabled, it can not be disabled
- signalling via dedicated error-pin



Trends - Low Power



Firm-/Software Optimierungen

- float & double vermeiden → sehr Rechenintensiv bei Systemen ohne FPU (und auch sonst)
- Divisionen sind rechenintensiv, ausser durch Zweierpotenzen 2, 4, 8, 16, ..., wo Bitshifting gemacht wird.
- Optimierungen einschalten beim Kompilieren :-)

Anhang

Crypto

Permutation Tabellen

Initial Permutation IP								Final Permutation IP ⁻¹							
58	50	42	34	26	18	10	2	40	8	48	16	56	24	64	32
60	52	44	36	28	20	12	4	39	7	47	15	55	23	63	31
62	54	46	38	30	22	14	6	38	6	46	14	54	22	62	30
64	56	48	40	32	24	16	8	37	5	45	13	53	21	61	29
57	49	41	33	25	17	9	1	36	4	44	12	52	20	60	28
59	51	43	35	27	19	11	3	35	3	43	11	51	19	59	27
61	53	45	37	29	21	13	5	34	2	42	10	50	18	58	26
63	55	47	39	31	23	15	7	33	1	41	9	49	17	57	25