

Dependability and Fault Tolerance Zuverlässigkeit und Fehlertoleranz

Chapter 3: **Formal Issues in Reliability**

H. T. Vierhaus

Wintersemester 2018 / 2019

Basic Termination

Reliability- (Zuverlässigkeit): Probability of non-failure over a certain time, can be described formally. Typically not considering any mechanisms that prevent failure.

Typically this parameter is measured / defined without regarding early-life failure or wear-out effects !

Dependability- (Verlässlichkeit): This parameter includes reliability plus extra mechanisms for error / failure protection. Formally much more complex to describe !

Security- (Sicherheit): This is an issue regarding mis-use, abuse, hacking etc. Very difficult to describe formally !

Safety : This is often used as the most comprehensive feature, often including reliability and security. Hardly defined in a formal way.

Modeling Reliability (1)

Assuming we have a number of N of identical components, such as e. g. transistors. They start their job at a time $t = 0$. At a later point of time t only a smaller number $n(t)$ will be functional. Then the formal reliability is defined as:

$$R(t) = n(t) / N$$

It is also possible to model the „Unreliability“ of the system. This is based on the number of failed components $n_f(t)$ at a certain time t :

$$Q(t) = n_f(t) / N = 1 - R(t)$$

Modeling Reliability (2)

For practical use, often a „failure rate“ λ of system components is defined. The „real life“ failure rate is given by the bath-tub-curve, in which several mechanisms are superimposed, such as early-life-failure, the „normal“ error rate plus wear-out-effect.

$$R(t) = e^{-\lambda t} \quad (\text{Reliability})$$

Typically, in this formula only the „normal“ failure rate after early-life failures and before wear-out is considered.

Modeling Reliability (3)

For practical reasons the time until a system is likely to fail for the first time is of great importance. This is the „mean time to failure“ (MMTF).

$$\text{MMTF} = \int_0^{\infty} R(t) dt$$

Assuming that the probability of failure is constant over time, we can also write:

$$\text{MMTF} = \int_0^{\infty} e^{-\lambda t} dt = 1 / \lambda$$

The mean time to failure is just the reverse of formal reliability !

Modeling Reliability (4)

Furthermore it is important to know, with which probability a system will work properly over a certain space of time. In particular, this is the time until MMFT.

This means, the time is $t = 1 / \lambda$.

With: $R(t) = e^{-\lambda t}$ we obtain: $R(t) = 0,37$

That means, until the MMFT the system has a probability of 37 % to work correctly and a 63% chance to fail.

Now we consider the fact that a system may have to undergo a repair process.

For this purpose, we need to define a „Mean Time to Repair“ MTTR. This is the time spent from the point of failure until it is functional again.

Systeme können / müssen auch „repariert“ werden.

Just like the failure rate λ we can now define a repair rate: $\mu = 1 / \text{MTTR}$

Modeling Reliability (5)

Real systems work in such a way that, after a first failure, there is a repair, some time of operation, failure, repair etc. Therefore it makes sense to define a time interval as „Mean Time Between Failures“ (MTBF). This is then the average operational time between two repair / service events. The time that the system needs to be back in service after the first failure and subsequent repair is:

$$\text{MTBF} = \text{MTTF} + \text{MTTR}$$

Typically, the time for repair (MTTR) is much shorter until the time to the first failure. Therefore MTBF is almost equal to MTTF.

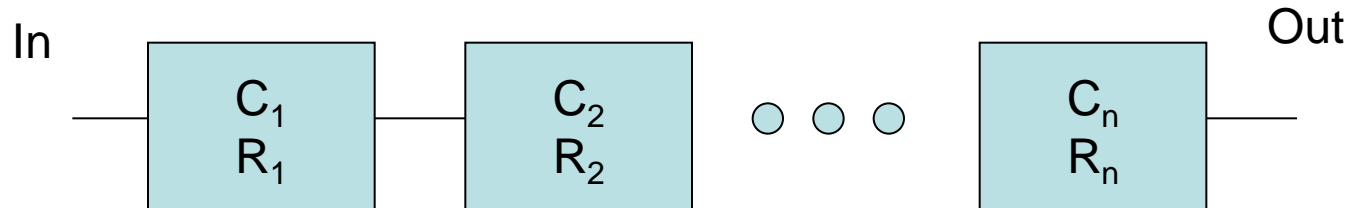
The next important parameter is „Availability“. This is the ration of active run time over run time plus repair time:

$$\text{Availability} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

Then the „**Unavailability**“ is:

$$\text{Unavailability} = 1 - \text{Availability}$$

Combined Chained Reliability



The components $C_1.. C_n$ of a system are connected in a chain. The reliability of the overall systems is equally dependent on these components. Then the total reliability R depends on the reliability of the single components $R_1.. R_n$ in a simple way.

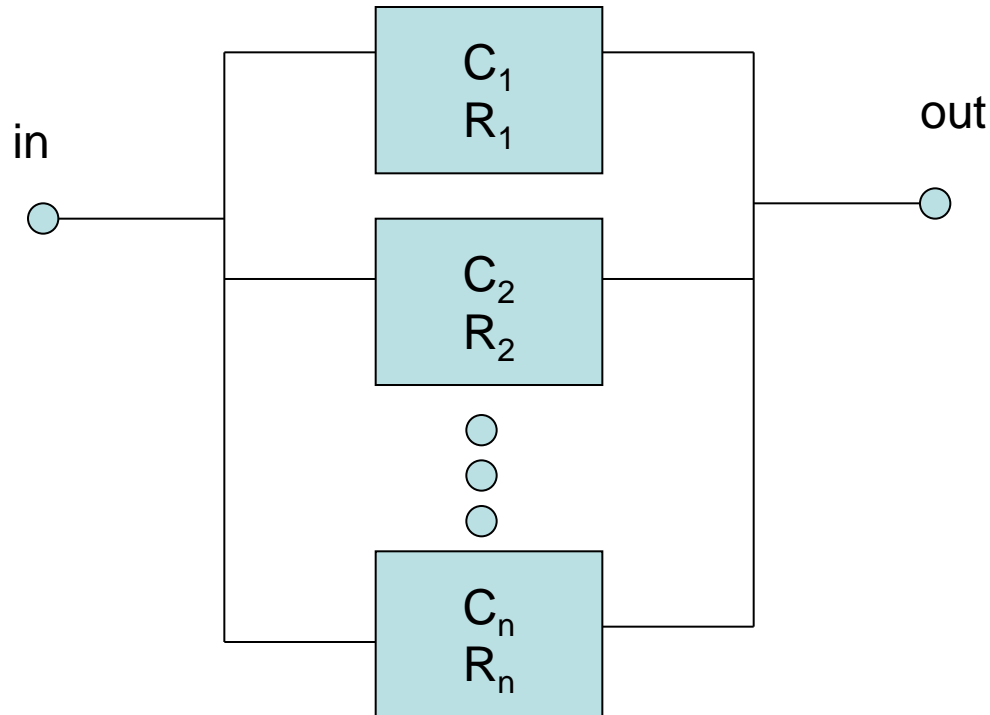
Overall failure rate:

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

Overall reliability: $R(t) = R_1(t) * R_2(t) * \dots * R_n(t)$

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

Combined Parallel Reliability




Boundary condition: - Components C_1, \dots, C_n can each take over the total function of the system !

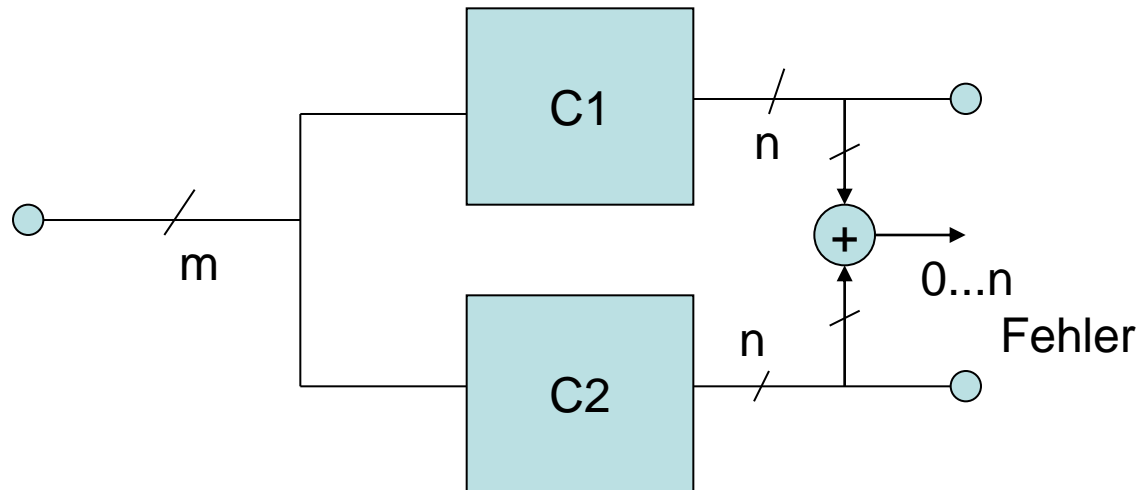
$$\text{Unreliability: } Q(t) = [1 - R_1(t)] * [1 - R_2(t)] * \dots * [1 - R_n(t)]$$

whereby $R_i(t)$ are the reliability values of the parallel modules.

„Redundancy“

- **Hardware-Redundancy:** Hardware is available multiple times either in active operation (hot redundancy) or in an off-line sleeping mode (cold redundancy).
- **Information Redundancy:** More than the necessary amount of information is stored or transmitted / received. This may serve to detect and correct errors.
 Error Coding !!!!!
- **Time-Redundancy:** Information is processed or transmitted multiple times. Then comparison can detect errors.
- Most of the processes that use redundancy also require extra power and extra energy !! This is a real-life obstacle. Only „cold hardware redundancy“ is quite economical in this sense.

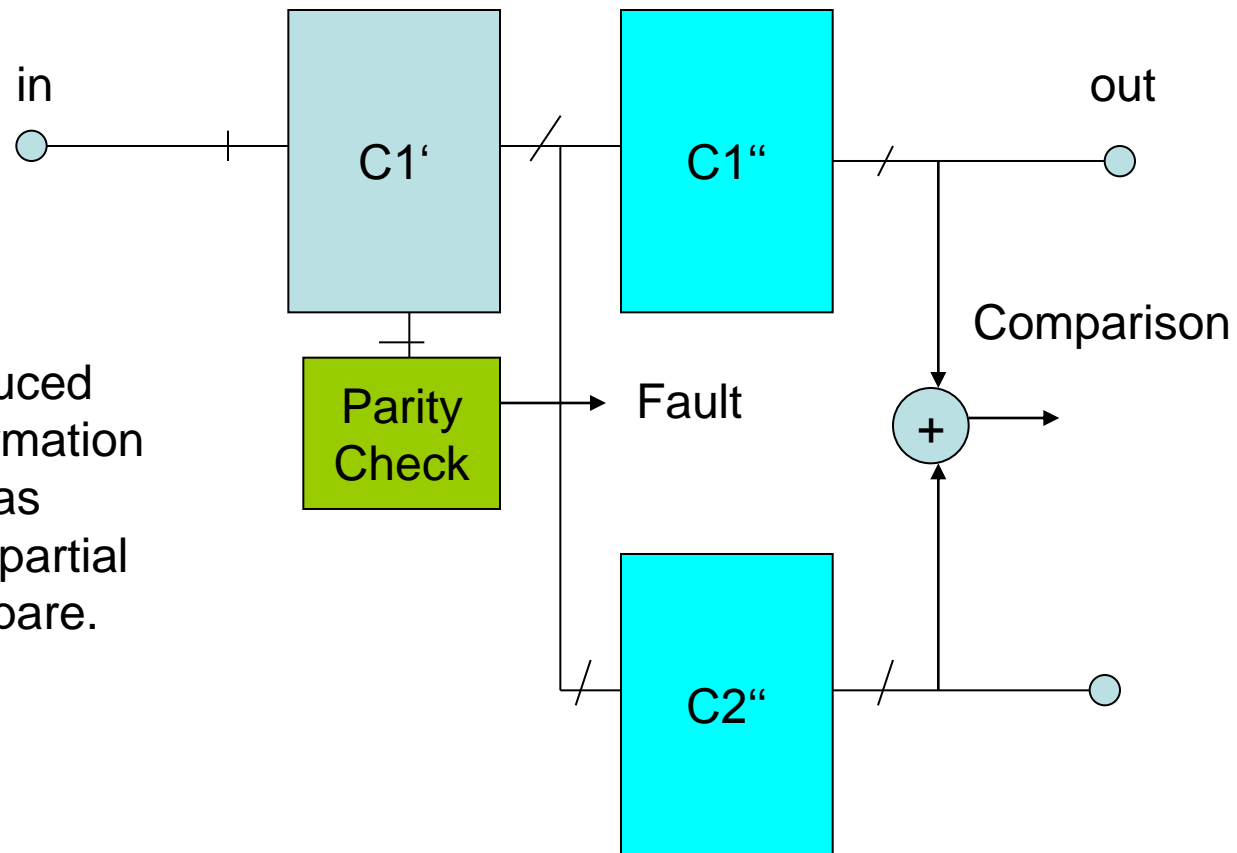
Duplication



Duplication and subsequent comparison by XOR-gates allows for error detection, but not error diagnosis. Uses double peak power, and double energy !

Partial Duplication and Parity-Check

Efforts can be reduced by combining information redundancy such as parity checks and partial duplication & compare.



Duplication

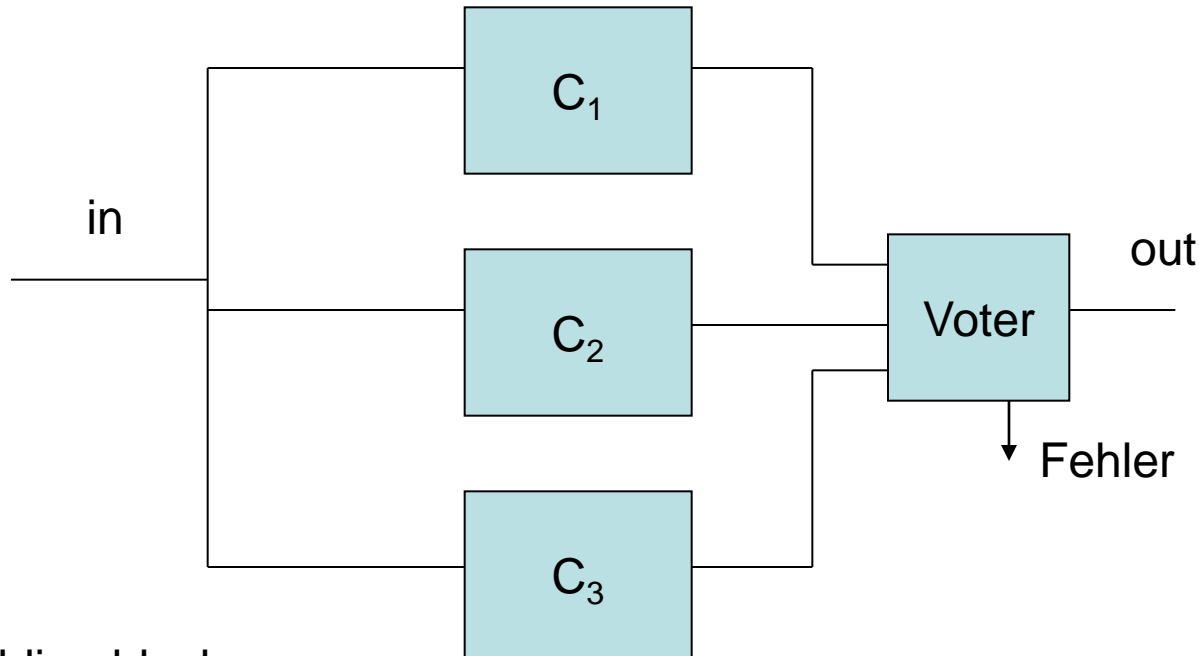
Pro:

- Simple scheme, applicable in running operation, little extra delay
- Can potentially recognize all types of errors
- No fault models necessary

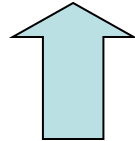
Cons:

- Cannot diagnose the correct / false result
- More than double hardware
- Double power dissipation, double energy.

Triple Modular Redundancy (TMR)



Diverse building blocks
(processors, hardware,
software) will catch even
design errors in HW or SW,
but no specification errors !



Blocks with equal function, may be equal
or diverse in structure !

TMR Pro- / Cons

Pros:

- Universal scheme to detect and correct transient or permanent errors. No fault models ! May work even in case of several faults.
- Diverse implementation of functional units (processors, software) allows to catch even implementation errors.

Cons:

- Relative high effort in extra hardware (over 200 % overhead)
- At least triplication in power and energy consumption
- The comparator / voter is the „single point of failure“ unless it is error-proof by itself (self-checking checkers).
- Extra time delays because of comparator / voter.
- Typically no real gain in system life time, since higher energy often means higher temperatur and faster wear-out !.