Modelling of Computer Systems

Computer systems include three basic types of entities:

Hardware

Software

Data

Hardware

- Continuous simulation is commonly used to check if the synchronisation paradigms work between the hardware components. It is not very useful because a simulator must believe written specifications and the error, if any, usually lies in the design.
- Both discrete and continuous simulation are used in modelling the flow of data through hardware components. The goal is to monitor the timing (and errors caused by timing issues) as opposed to the properties of the data being pumped around.

Clock issues

When modelling hardware it is tempting to assume that

1. All the devices access the same central clock.
2. The clock can be accessed with a delay of 0 (or at least with a fixed delay for all the devices).

Both these assumptions are patently wrong.

Clocks

Each device has its own timing mechanism (**“clock”**). No two clocks operate at exactly the same frequency hence no two devices can be perfectly synchronised.

Data transfer with two clocks

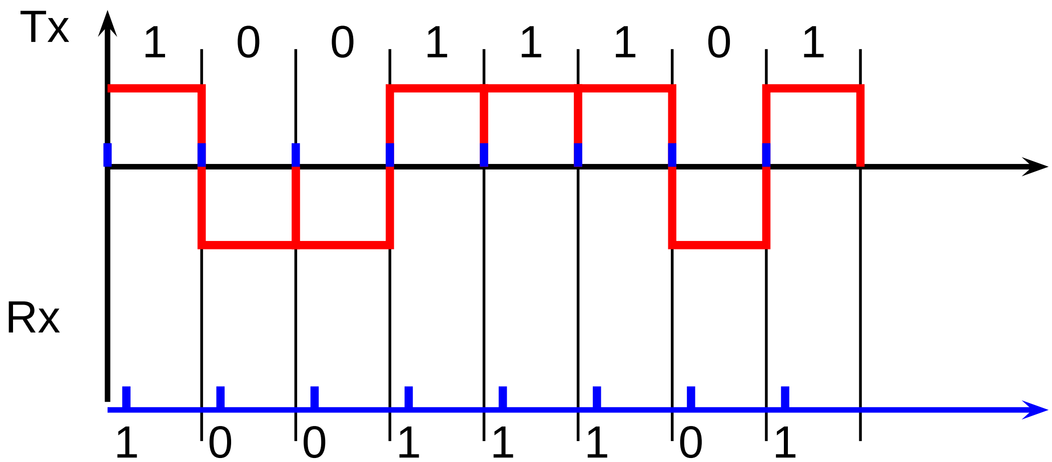
One device (disk controller, NIC, etc.) sends bits down a physical line. At the other end, another device receives them. Each has its own clock.

The receiver uses its clock to sense the flow of bits. the transmission rate is known to the receiver, so its clock ticks an integer number of times per pulse (or bit). The most common ticking rate is 32 ticks per bit. The problems:

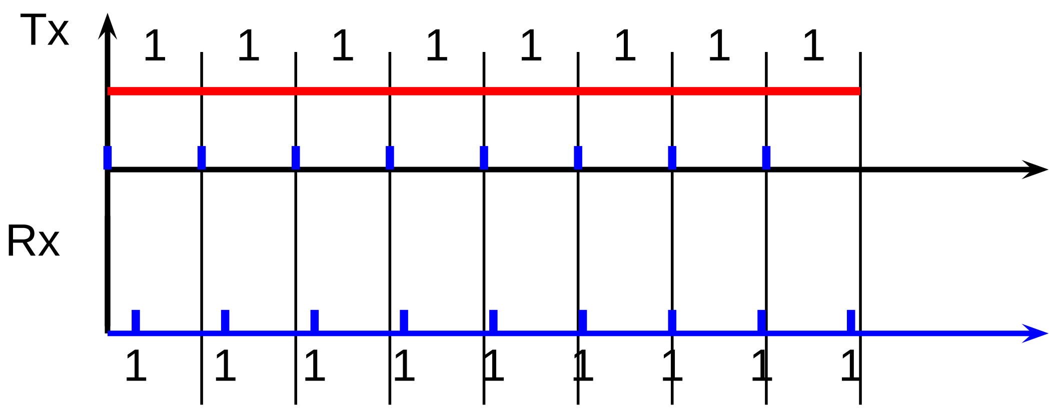
- There is an unknown propagation delay between the sender and the receiver, so it is impossible to synchronise their clocks permanently. Hence, the receiver must figure out when a bit starts.
- No two clocks have an identical ticking rate, at least because of crystal impurities. Hence, the receiver must continuously adjust the position of the tick that indicates the start of a bit.

Non–Return–to–Zero

Two voltage levels are used (usually a positive and a negative voltage).



Clock drift will result in incorrect decoding (occurs for any bit sequence).

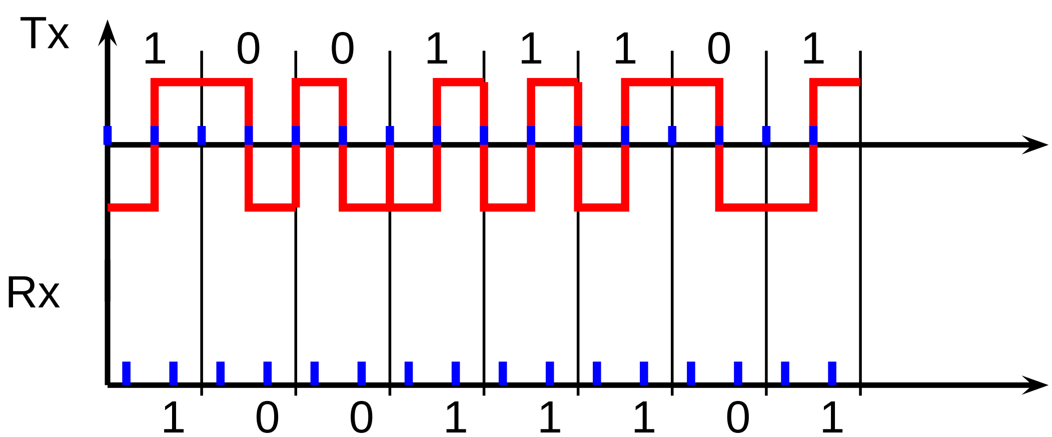


The receiver received 9 ones although only 8 were sent.

Manchester encoding

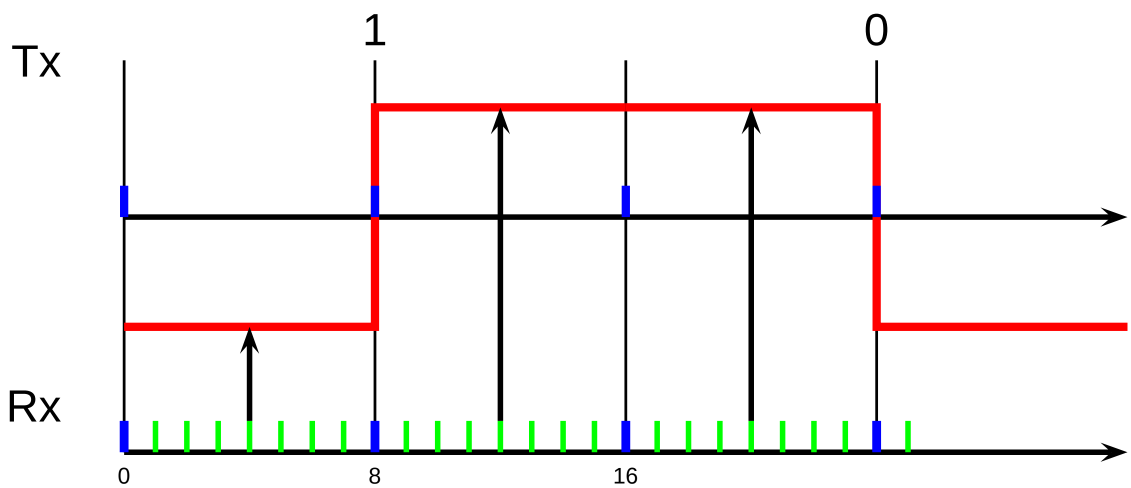
There is a transition in the middle of every bit. The transition serves a mechanism for clock synchronisation; it also gives the value of the bit:

- A low to high level transition means a “1” bit.
- A high to low level transition means a “0” bit.



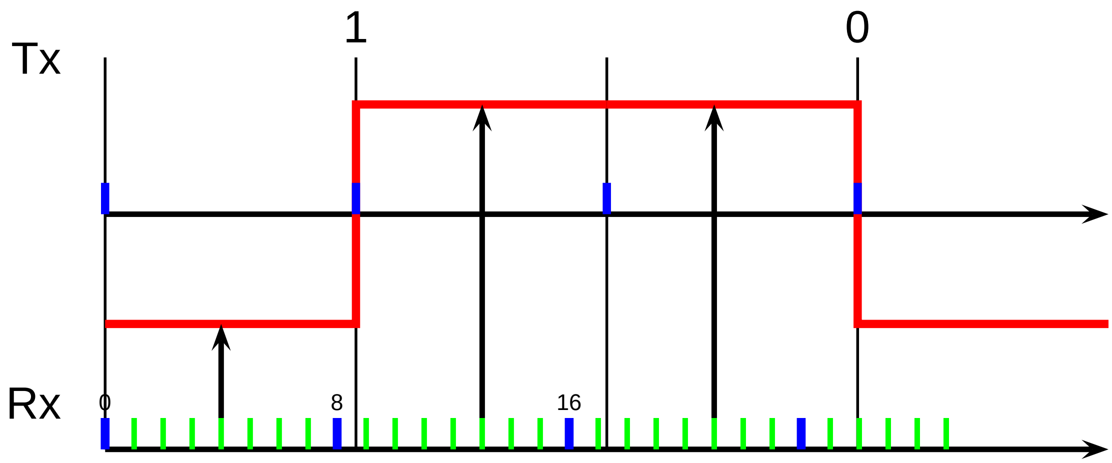
Clock synchronisation

An ideal scenario with Manchester encoding and 16 ticks per bit:

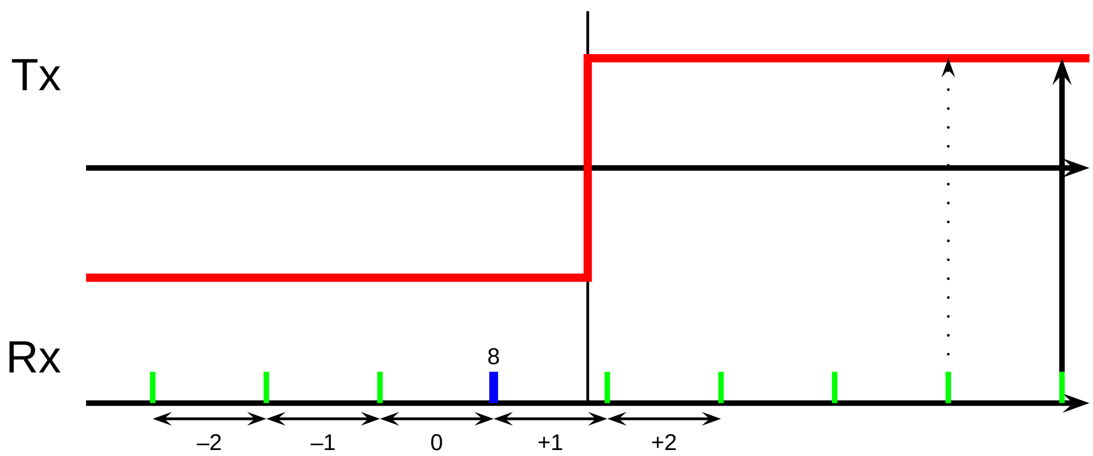


Clock synchronisation

A not so ideal scenario with Manchester encoding and 16 ticks per bit:



The transition part magnified:



Simulating clocks

The common method is the use of Universal Time, a fictitious but useful concept.

- The simulator maintain one **UT** clock that ticks in universal time units (**UTU**).
- Every clock in the model has its own ticking rate which is expressed in **UTU**s; it is a value close to, but not equal, 1.
- All the time intervals in the simulators are converted into UTUs for event handling purposes.
- The local clock rates are used in one place only inside the simulator: in the function that converts time intervals to **UTU**s.

Simulating errors

By tradition, computer components are supposed to break down at a rate proportional to the time they are in use.

We distinguish several types of errors:

1. Permanent failures such as a disk head crash or a power supply failure. These are described by a statistic known as MTBF.
2. Stochastic errors, excluding external interference. The most famous is the **BER** (bit error) associated with transmission channels.
3. Transient errors, including errors induced by external causes.

Mean Time Between Failures

Time Between Failures is a random variable describing the interarrival times of failures ("failure" is a customer arriving to a repair server).

The mean, called MTBF, is often defined as:

$$\theta = \lim_{t \to \infty} \frac{t - \sum_{i=1}^{n} f_i}{n}$$

where n is the number of failures in the time interval $[0, t]$ and f_i the duration of the i^{th} failure.

The above definition assumes falsely that a device is always up or down.

If the Time Between Failures is an exponentially distributed variable (firmly believed) with a mean of **MTBF**, the probability that a device will fail before its **MTBF** is about 63%..

More on MTBF

The general belief is that the failure rate of devices is not constant, but generally goes through three phases over the lifetime of a device. In the first phase the failure rate is relatively high, but decreases over time – this is called the ”infant mortality” phase. In the second phase the failure rate is low and essentially constant–this is the ”constant failure rate” phase. In the third phase the failure rate begins increasing again, often quite rapidly–this is the ”wearout” phase (the graphic representation of failure rate as a function of time gave name to “bathtub” or “bowl” curve)..

TBF is the inverse of the failure rate in the constant failure rate phase. MTBF is, therefore an excellent characteristic for determining how many spare hard drives are needed to support 1000 PC’s, but a poor characteristic for guiding you on when you should change your hard drive to avoid a crash.

Modelling TBF

There are many studies of TBF distributions (for various devices). Their common property is that they show MTBFs that are much higher than those “experienced” by the general public (note that the scientific term for “experienced” is either “anecdotical” or “hearsay” when described politely).

- Tests are conducted in “correct” environment by skilled people.
- In real life a mistreatment of a device will very often shorten its lifetime but not result in a failure in the immediate future.
- Ignorant folks confuse MTBF with Mean Time To Fail.

Stochastic errors

When a device fails, it is not operational anymore (until fixed or replaced). This differs from a device operating but giving an erroneous output (faulty part or corrupted packet).

The two kinds of stochastic errors differ in their probability distributions:

Channel error rate is an inherent property of the channel (caused by its imperfection) and has a “nice” probability distribution. Many distributions are considered; the most common approach is to use the Bernoulli process in discrete cases and the Poisson process in continuous cases.

Transient errors are totally unpredictable and occur at truly “random” moments. Surprisingly the distribution that describes such phenomena is well–known: it is the **....**.

How to model?

Failures: this is a common topic in modelling of manufacturing systems.

BER type errors: use exponential arrivals or a Bernoulli process for every bit.

Transient errors: the key issue is to guess the upper bound of the interarrival time interval (the lower is 0).

Modelling software

We are talking about models of software products, not software products for modelling anything else.

There is a lot of literature on this subject under the label of "**Software Engineering**" (consider UML, Extreme Programming, etc.).

Additionally, we have Quality assurance and its models.

This has nothing to do with modelling and simulation.

Simulation of software behaviour

- Software runs in an emulator provided by the kernel of the O.S. Incorrect behaviour is trapped and diagnosed.
- A debugger is a typical software emulator.
- The notion of virtual machine has been around since 1964.

Modelling software performance

Given two pieces of software it is natural to compare their performance, usually expressed as the volume of system resources needed to complete a task.

This is most often applied to algorithms. Two methods of comparing algorithms stand out:

Asymptotic analysis giving the algorithm's resource requirements as a function of the problem size.

Timing which gives the CPU time used by an algorithm to solve a particular instance of a problem.

Asymptotic analysis

Asymptotic analysis gives the complexity of an algorithm as a function of the problem size. It is defined only for problems of sizes approaching infinity and has a limited usefulness in practice.

Algorithm	complexity		
	Average	Worst	Best
Qsort	$O(n \log n)$	$O(n^2)$	$O(n \log n)$
Hsort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
DPS	$O(n)$	$O(n \log n)$	$O(n)$
Insertion	$O(n^2)$	$O(n^2)$	$O(n)$

Timing

I found this jewel in an old publication:

Table 1. Average execution times (uniform distribution)

Sequence length	LP	MLI	DP	Q	H
1000	60	93	85	113	220
2000	61	89	86	124	240
3000	62	96	87	132	260
4000	62	94	88	140	280
5000	63	99	89	142	285

Table 3. Average execution times (parabolic distribution)

Sequence length	LP	MLI	DP	Insertion
200	700	426	167	717
400	1513	841	173	1402
600	2330	1266	178	2045
800	3117	1756	179	2751
1000	3930	2326	183	3431

Standard deviations?

Sequence length	LP	MLI	DP	Q	H
	Uniform distribution				
1000	100	80	45	90	30
	Parabolic distribution				
1000	12000	2600	80		

Confidence intervals with $\alpha = 0.01$

LP	60 ± 26
MLI	93 ± 21
DP	85 ± 12
H	220 ± 7.3

Modelling data

There is no general theory of modelling data. The basic recipe is well known:

1. Gather a sample as large as possible.
2. Use a χ^2 or similar test to select a distribution that cannot be rejected.
3. Use thie selected distribution.

As an example, let us look at a paper by L. Breslau et al.

This paper uses a distribution called Zipf's distribution which gives the probability of access to the i^{th} most popular object in a set of n objects.

The Zipf's distribution has a pdf:

$$f(i) = \frac{1}{i}\frac{1}{H_n}$$

where $H_n = \sum_{i=1}^{n} \frac{1}{i}$.