I Embedded systems and microcontrollers

System: a way of working, organizing or doing one or many tasks according to a fixed plan, program, or set of rules. It's an arrangement in which all its units assemble and work together according to the plan or program.

Embedded systems: a special-purpose computer designed to perform one or few dedicated functions. Common characteristics: low cost, low power, high performance, real-time.

Constraints: available system memory, available processor speed, need to limit power dissipation

Classification: small scale (8 or 16 bit, little complexity), medium scale (single or few 16 or 32 bit), sophisticated embedded system.

Stack: hardware (processors, memory, I/O devices), OS software, application software.

Microprocessor: CPU on single chip

Von-Neumann computer architecture: control unit, ALU, memory, I/O

Microcontroller: CPU + Timers + I/O + RAM + ROM + RAM + ADC/DAC

ROM (read only memory), PROM (programmable ROM), EPROM (erasable PROM by UV), EEPROM/Flash (electronically erasable PROM)

RAM: stores programs or data, static vs. dynamic

Memory-mapped I/O: regular memory cell

IO-mapped: special I/O ports

DMA: allows I/O devices to access memory directly without involving the CPU, parallel processing, but only one can access bus at a time

Processors:

General purpose processor (GPP), application specific system processor (ASSP), multi-processor system using GPPs

While microcontrollers are basically tiny computers, PLCs are a bunch of relays, specifically designed for programming simple logic, timing, arithmetic and I/O for industrial processes.

DEL I: UML

Introduction

Sekvensdiagrammer

Klassediagrammer

Use Cases

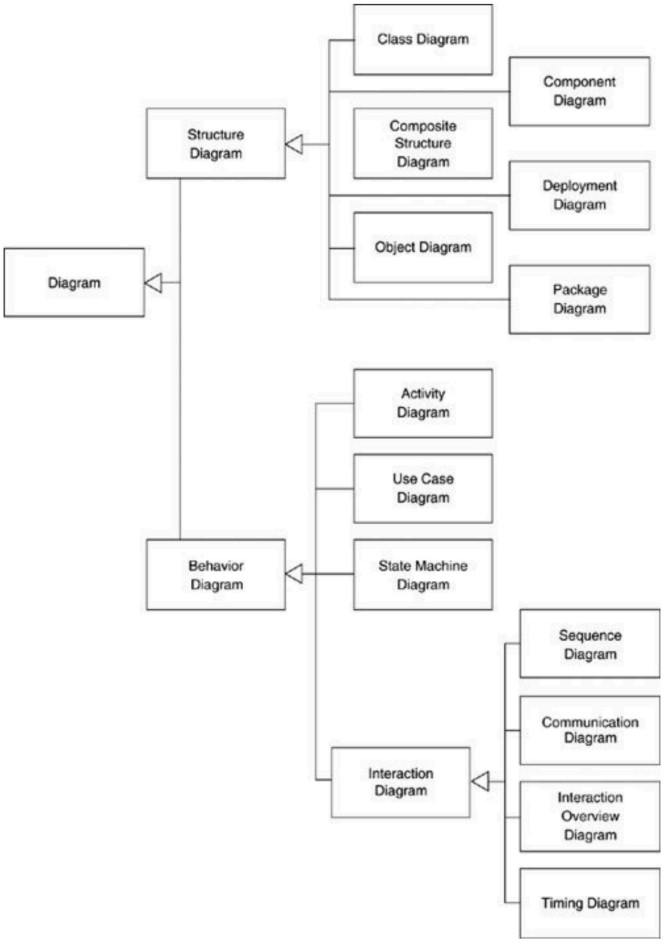
Kommunikasjon- og timingdiagrammer

Øvrige UML-diagrammer, komplett liste under "Syllabus"

Tilstandsmaskiner

UML is a family of graphical notations, backed by a single metamodel, that help in describing and designing software systems, particularly object-oriented software systems. Defines a notation (the graphical syntax) and a meta-model (diagram that defines the concepts of the language)

Three modes: sketch (communicate ideas, selective, forward engineering or reverse-engineering), blueprint (complete model), programming language



Class diagrams

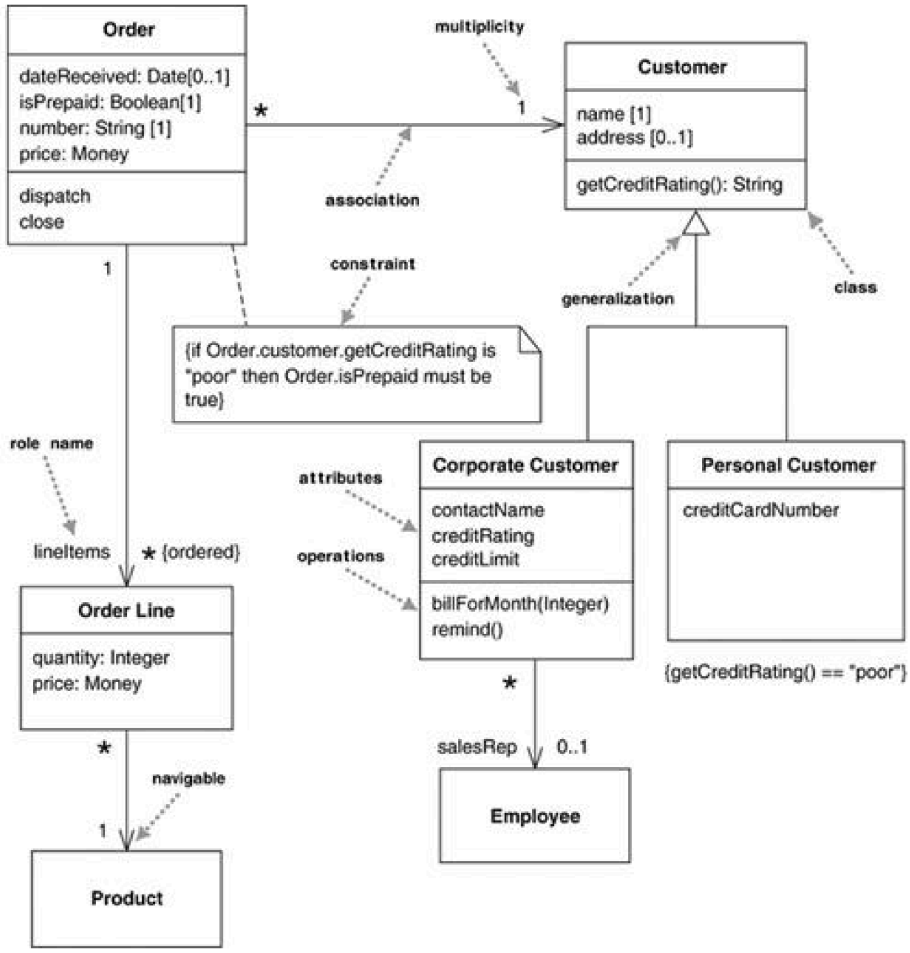


Diagram for classes, their properties and relationships. Properties: attributes and associations.

Attribute: `visibility name: type multiplicity = default {property-string}`

Association: specify relationships between classes, unidirectional or bidirectional

Operation: `visibility name {parameter-list} : return-type {property-string}`

Parameter: `visibility direction name: type = default value`

Visibility: public `(+)` or private `(-)`

Property-string: additional properties, like `read only`, or `query` (getter, we call operations that do change state modifiers).

Multiplicity: indication of how many objects may fill the property, e.g. `0…1, 1, *`

Direction: `in, out or inout.`

White arrows indicate generalizations (inheritance).

Dashed lines indicate dependencies, UML has a list of keywords to better describe the dependency.

We can also create notes.

Sequence diagrams

Interaction diagrams describe how classes collaborate in some behavior. The most common one, the sequence diagram, captures the behavior of a single scenario.

Note the difference in centralized vs. distributed control in the two figures.

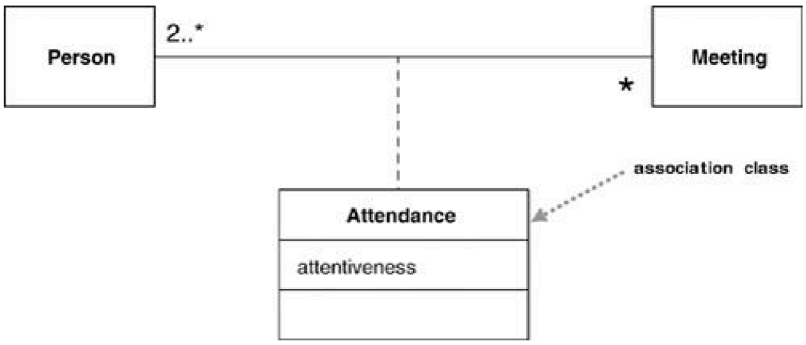
Sequence diagrams are good at showing interaction, not loops, conditionals, logic etc.

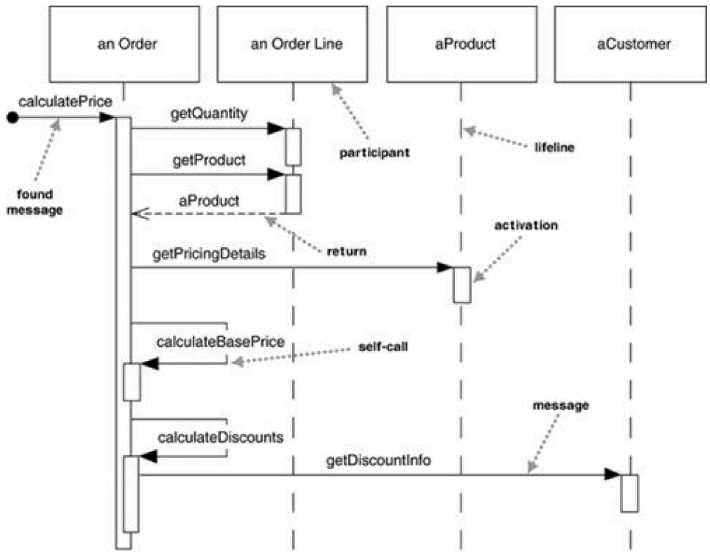
Creation is indicated by drawing the message arrow directly into the participant box.

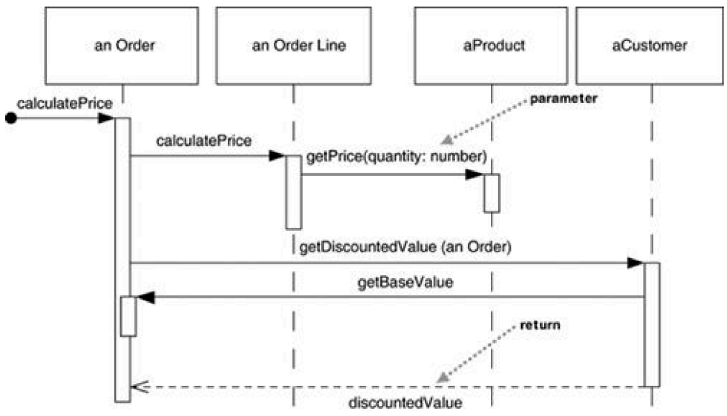
Deletion is indicated with an X.

You can add interaction frames to add loops, conditionals (alt) or other logic. They contain an operator and an optional guard, like else.

Filled arrowheads show synchronous messages (two-way communication, wait involved), stick arrowheads show asynchronous messages (one-way flow).



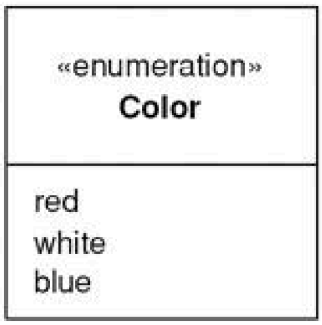




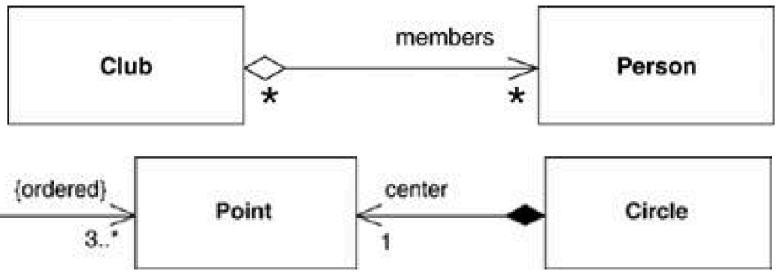
Class diagrams: advanced concepts

Interface: only public operations, with keyword "`interface`".

Classification refers to the relationship between an object and its type, single classification where an object belongs to a single type, or multiple classification where an object may be described by several types not necessarily connected by inheritance.



Association classes allow you to add attributes, operations and other features to associations.



Enumerations are used to show a fixed set of values that don't have any properties other than their symbolic value, using "`enumeration`" keyword.

Can also add responsibilities on a class through comment strings.

Static variables inside a function keeps its value between invocations. Static variables are underlined.

Aggregation is the part-of relationship.

Composition is similar, but we have a "no-sharing" rule.

Derived attributes are denoted with /, and can be calculated from other variables.

A qualified association is equivalent to associative arrays, maps and dictionaries.

Abstract classes are classes that are not directly instantiable, but instantiable through a subclass. Indicated with italics or a label. Interfaces are classes where all their features are abstract.

Template classes have a box with a T in it, in the class header.

Use case diagrams

Captures the functional requirements of a system. Describing typical interactions between users and the system. A scenario is a sequence of steps describing an interaction between a user and the system. A use case is a set of scenarios grouped together by a common user goal. Users are referred to as actors, a role that the user plays with respect to the system. When you make a use case you start by writing the main success scenario as a numbered list, and you then write other scenarios as extensions, describing them in terms of how they diverge from the main success scenario.

Buy a Product

Goal Level: Sea Level

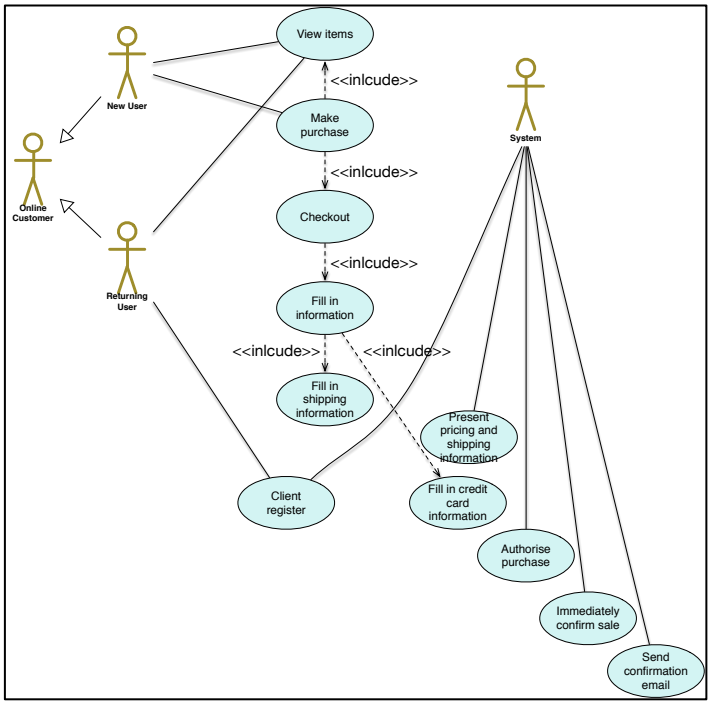
Main Success Scenario:

1. Customer browses catalog and selects items to buy
2. Customer goes to check out
3. Customer fills in shipping information (address; next-day or 3-day delivery)
4. System presents full pricing information, including shipping
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming e-mail to customer

Extensions:

3a: Customer is regular customer
.1: System displays current shipping, pricing, and billing information
.2: Customer may accept or override these defaults, returns to MSS at step 6

6a: System fails to authorize credit purchase
.1: Customer may reenter credit card information or may cancel



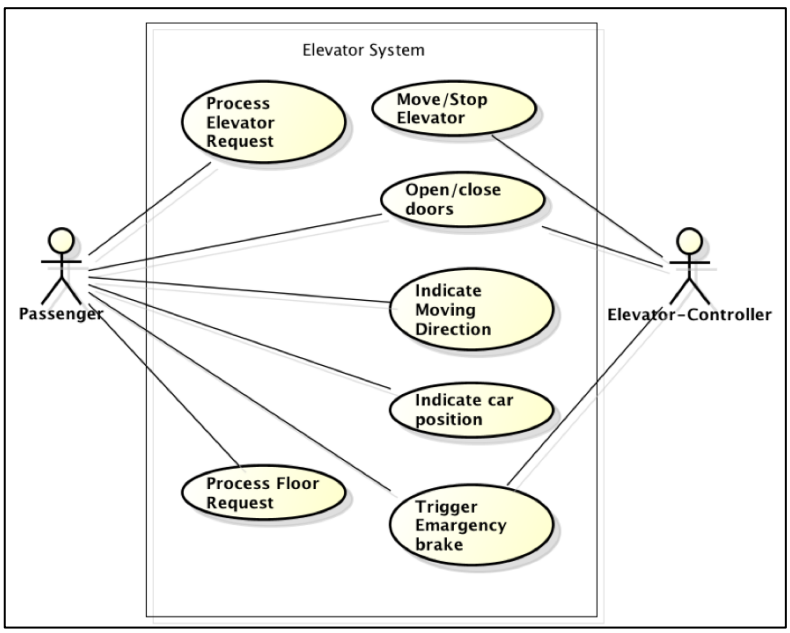
Each use case has a primary actor, which calls on the system, and is the actor with the goal the use case is trying to satisfy. Other actors the system interacts with in the use case are secondary actors.

You can also add:

Pre-conditions: what the system should ensure is true before the system allows the use case to begin.

Guarantees: what the system will ensure at the end of the use case, you can separate them in success guarantees and minimal guarantees.

Trigger: the event that triggers the use case.

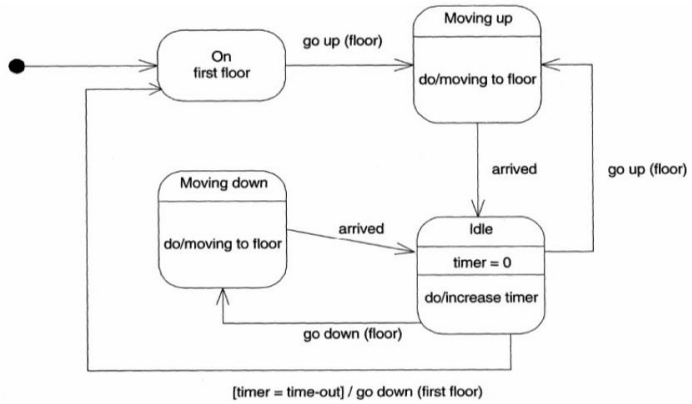


State machine diagrams

Describes the behavior of a system by organizing a system such that an entity is always in one of a number of possible states, with well-defined conditional transitions. Works especially well for event-driven systems. Typically implemented in software with a switch statement.

Transition syntax: `trigger-signature [guard]/activity` (all optional)

The trigger-signature is usually an event that triggers the transition. The guard is a Boolean condition for the transition. The activity is some behavior that is executed during the transition. Guards must be mutually exclusive for the same event, so the system always takes a single transition only.



A black dot indicates the starting point.

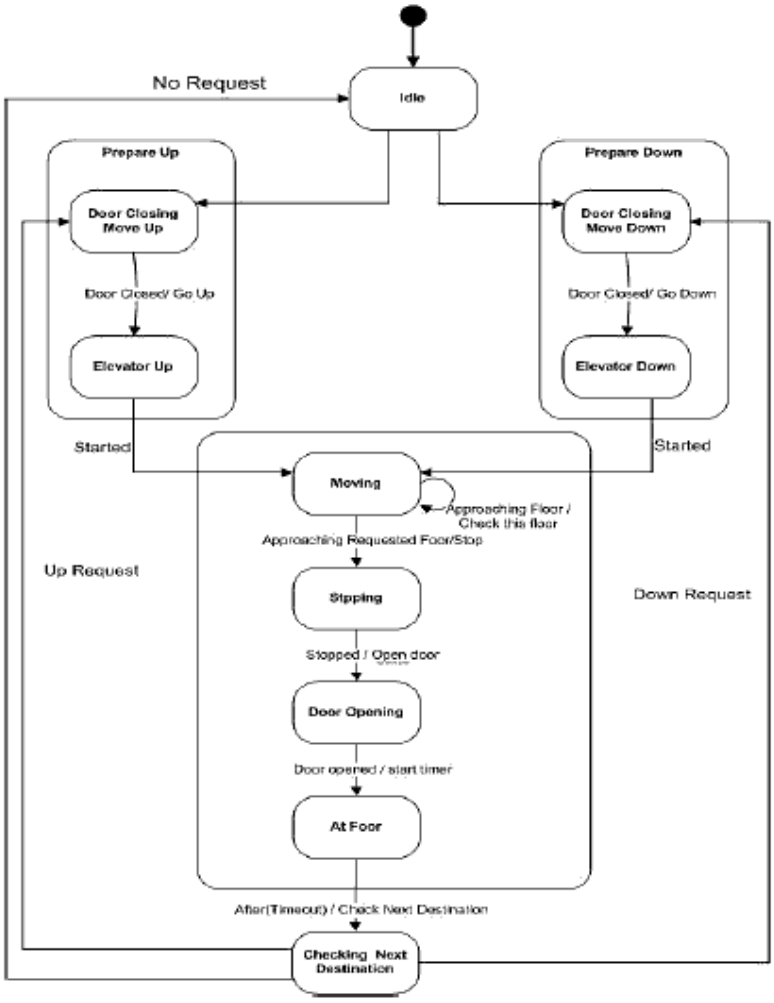
The **final state** indicates that the state machine is completed, so a termination. Indicated by ⊙.

You can define **internal activities** within a state by using the transition syntax inside a state box. There are two special internal activities: the `entry` and `exit` activity. The `do` trigger signature indicates some ongoing work within a state, called a do-activity.

Self-transitions loop back to the same state.

When several states share common transitions and internal activities you can group them as substates in a **superstate** with shared behavior. You can also break a state into several orthogonal state diagrams that run in parallel. You can use a history pseudostate, that indicates that when the system is turned on it goes back to the previous state when it was turned off.

You can also include conditionals.



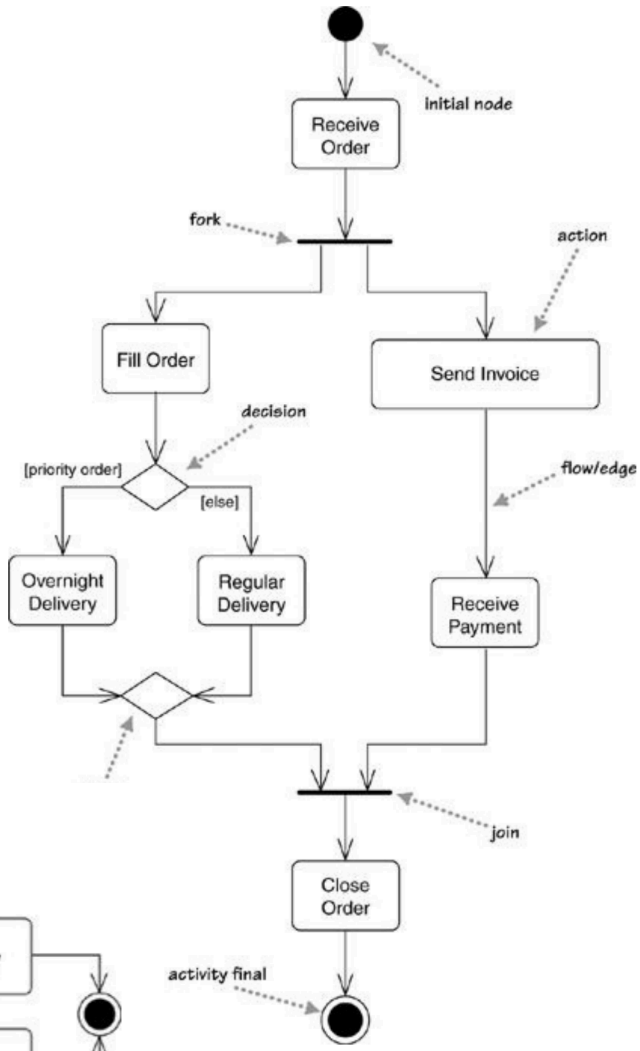
Activity diagrams

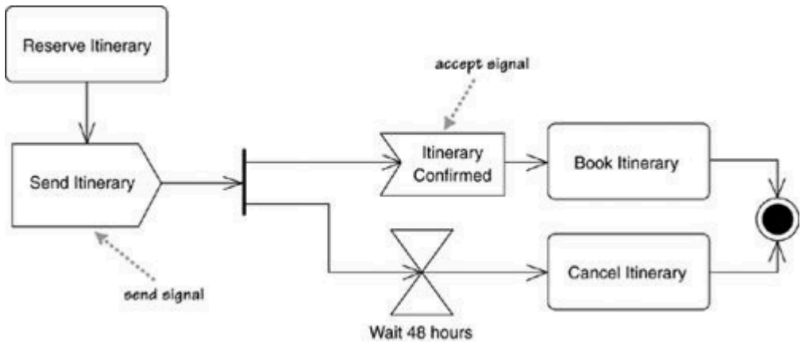
Used to describe procedural logic and work flow. Notice how we can use fork + join to run different logical processes in parallel. The join is only executed when all the incoming flows reach it. Conditional behavior is achieved with decisions and merges. Branches in conditionals does not support parallelism.

You can decompose an action into subactivities in its own activity diagram, denoted in the original action with a rake.

You can also partition the diagram to indicate which classes that does what, one-dimensionally as lanes or two-dimensionally in a grid.

We can also use different signals instead of a simple initial node. Time signals are indicated with an hourglass symbol.

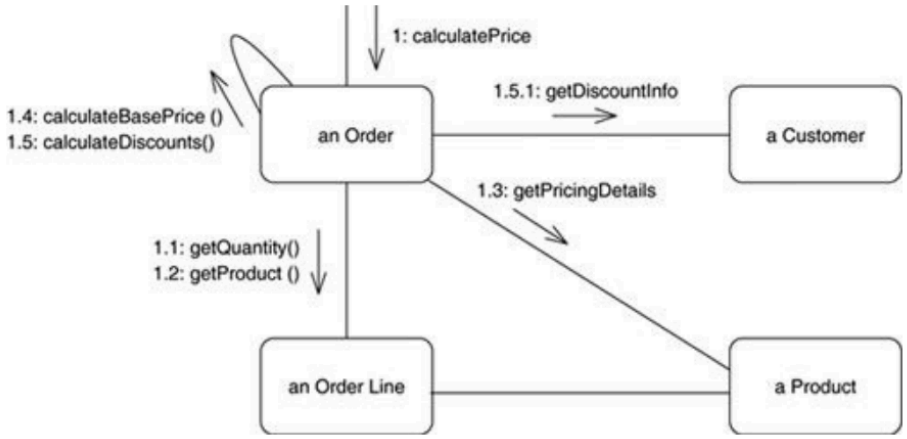




We also have accept signals that listens for an accept from an event, and we can send signals too. Accept and send signals can be incorporated to model two-way communication.

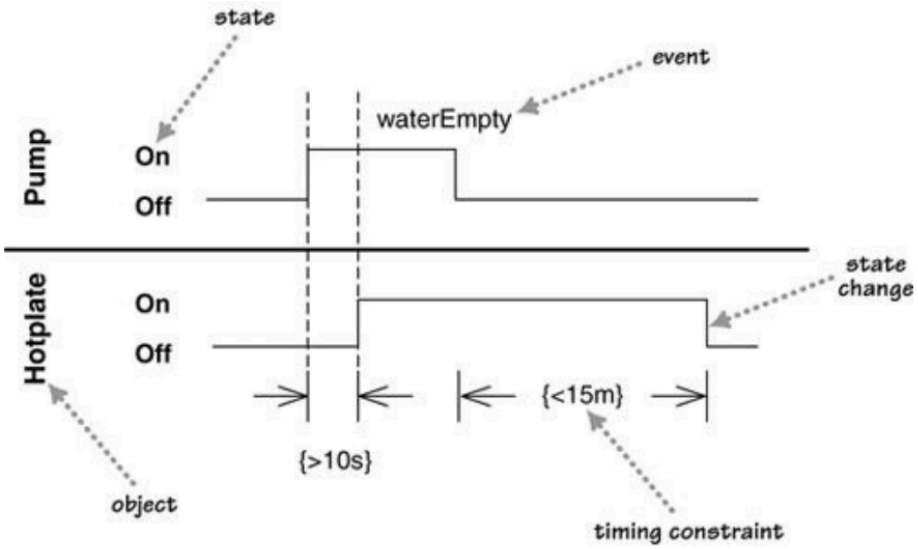
Communication diagrams

Used to emphasize data links between various participants in the interaction. Participants are freely placed, and links are numbered to show the sequence of messages. When to use them instead of sequence diagrams? When you want to emphasize the links, not the sequence of calls. Sequence diagrams are better for timing, communication diagrams are better for information flow.



Timing diagrams

Another interaction diagram where the focus is on timing constrains between state changes. This is very relevant for embedded systems and multi-threading.



II: Analog and digital signals

Introduction

Discrete vs continuous time-varying quantities. Digital signals are usually binary, but not necessarily. Embedded systems have to deal with both analog and digital signals as inputs and outputs. Communication between ICs is usually digital. Microcontrollers do have circuitry to allow them to interface with analog circuitry, like ADCs, DACs and PWM. Most fundamental electronic components are inherently analog. Analog circuits are more difficult to design than the digital equivalent and are more prone to noise, but offer high resolution and efficient transmission. Digital circuits are more expensive and have a processing delay.

Sensor $\Rightarrow$ *Buffer amplifier* $\Rightarrow$ *Low pass filter* $\Rightarrow$ *Sample and hold* $\Rightarrow$ *ADC* $\Rightarrow$ *Memory*

Quantizing: breaking down analog value in a set of finite states.

Encoding: assigning a digital word or number to each state and matching it to the input signal.

A **digital-to-analog converter** allows a microcontroller to produce analog voltages, which for instance is useful for speakers. An **analog-to-digital converter** allows the microcontroller to connect to an analog sensor to read in an analog voltage.

Communication protocols

A set of formal rules describing how to exchange data. Low-level protocols define the electrical and physical standards, like byte-ordering, transmission, error detection etc. High-level protocols deal with data formatting, like syntax, character sets, sequencing of messages etc.

Requirements: agreed code space (how data is encoded), agreed time space (when data is valid) and agreed name space (what the data means).

Message-based data exchange send data through a channel, while **shared memory-based data exchange** has a shared memory used for communication. No need for control flow with shared memory, while a bus needs control flow as well as data flow. Data is more vulnerable though.

State messages: periodic exchange of state, time-triggered <name, value, t_{obs}>.

Event messages: sporadic exchange of event information, event-triggered <name, Δvalue, t_{event}>.

Latency: time interval between start of message transmission at sender and time of reception of message at receiver.

Maximum delay: d_{max}

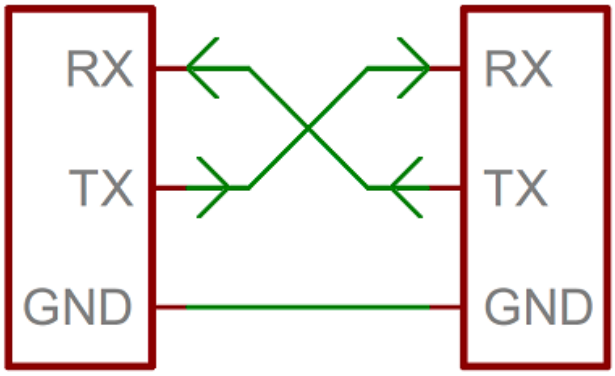
Minimum delay: d_{min}

Jitter: $\varepsilon = d_{max} - d_{min}$

Total bandwidth: maximum throughput

Throughput: the actual dataflow

Fault handling: message retransmission, forward error correction (redundancy), message redundancy (several copies of same message)



Control flow: push style (sender pushing out information e.g. email) or pull style (receiver pulling information e.g. podcast)

ICs must share a common communication protocol to exchange information. Protocols can be separated in **parallel** and **serial** protocols. Parallel interfaces transfer multiple bits at the same time. This requires data buses. Serial interfaces stream their data, a single bit at a time. Parallel is fast, straightforward and relatively easy to implement, but requires more I/O lines. We therefore often opt to serial, sacrificing speed for pin real estate. Common serial interfaces include USM, Ethernet, SPI, I2C and the serial standard. They can either be **synchronous** or **asynchronous**. Synchronous serial interfaces pair its data line with a clock signal, so that all devices on the bus share a common clock. Asynchronous means that data is transferred without the external clock signal. This minimizes the number of pins, but requires extra effort to transfer and receive data.

Asynchronous serial protocol

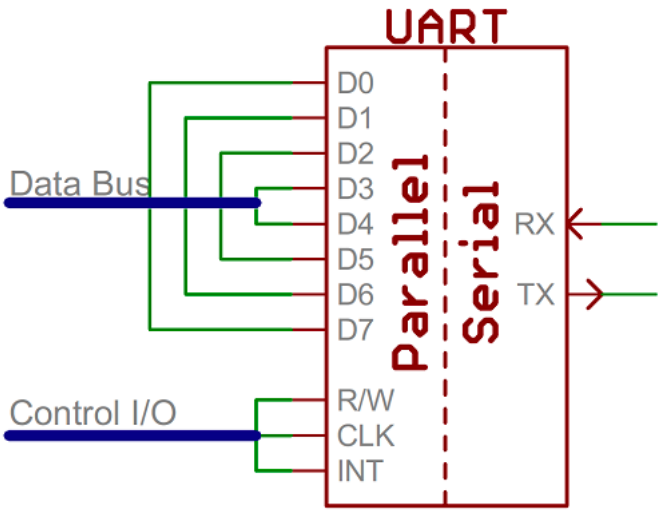
The asynchronous serial protocol adds data bits, synchronization bits, parity bits and baud rate to ensure robust and error-free transfer without the clock signal. The baud rate specifies how fast data is sent for the serial line (bps). Important that sender and receiver has the same baud rate. 9600 bps is a common baud rate for noncritical signals.

Each block of data is sent in a frame of bits, which includes synchronization bits and optionally a parity bit as well as the data. Serial devices has to agree on the endianness of the data, i.e. msb to lsb or lsb to msb. The stop bit will send the line to idle state by holding the line at 1, and then the start bit will indicate a new data packet by having the idle line go from 1 to 0. The parity bit is used for low-level error checking, by sending the evenness of the sum of the data bits.

A serial bus consists of two wires, one for sending and one for receiving data.

A serial interface where both devices may send and receive data simultaneously is full-duplex, while in a half-duplex interface the devices must take turns sending and receiving.

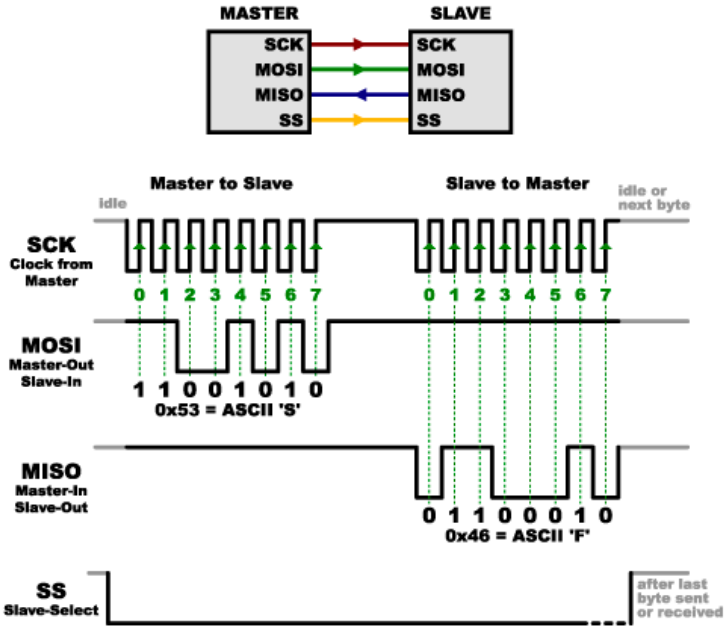
A universal asynchronous receiver/transmitter (UART) is a circuit responsible for implementing serial communication. It is an intermediary between parallel and serial interfaces. Most commonly found inside microcontrollers. The UART creates the data packets and sends the packet out on the TX line. It also samples the RX line and enters the data on the data bus.



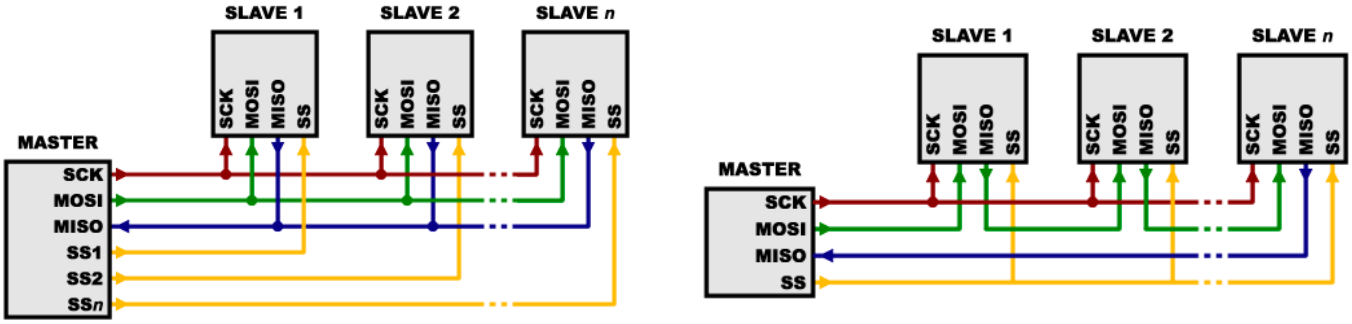
Problems: no guarantee that both sides run at precisely the same rate, a lot of overhead and computers are usually synced to a master clock, so two systems with slightly different clocks will not match.

SPI

SPI is a synchronous data bus. The clock is an oscillating signal that tells the receiver exactly when to sample the data line. On a SPI bus you have a master that generates the clock signal, and one or multiple slaves. Data is sent from the master to a slave on the MOSI line (master out, slave in). If the slave needs to respond it will put the data on the MISO line (master in, slave out). Notice that SPI is full-duplex. The SS line (slave select) is used to select the slave which should send/receive. When the SS line is high, the slave is disconnected from the bus, and when data is about to be sent, the SS line goes low to activate the slave.



When there are multiple slaves they can either be chained or have a separate SS line each.



When they are chained the MISO of one slave goes to the MOSI of the next. The data will overflow from one slave to the next. This is usually used in an output-only situation.

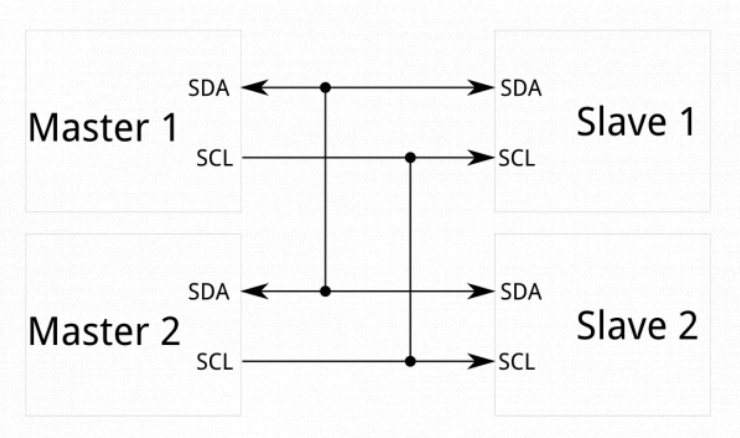
Advantages: faster than asynchronous serial, supports multiple slaves, simple logic.

Disadvantages: requires more lines (usually requires a lot of SS lines), master controls all communication.

I2C

Devices with asynchronous serial ports must agree ahead of time on a baud rate, and the clocks must be close to the same rate. There is also a lot of overhead in hardware and transmission, and communication is between two devices only. SPI's biggest drawback is the number of pins required. SPI also only supports one master on the bus. I2C deals with a lot of these shortcomings.

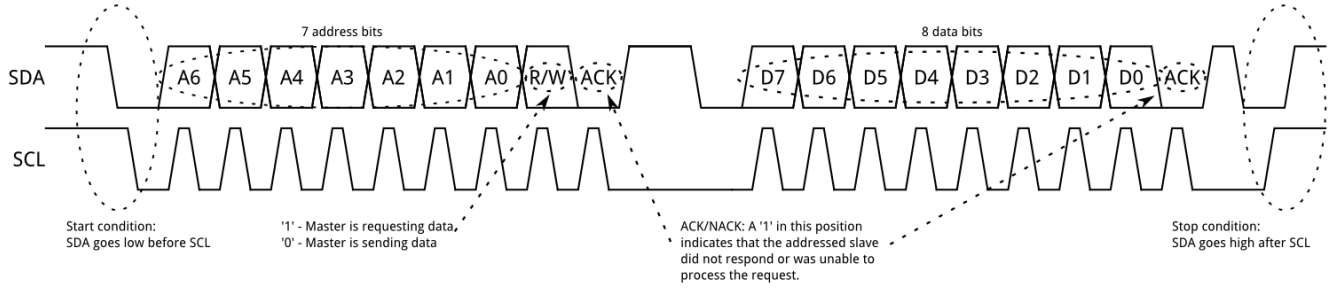
The Inter-integrated Circuit (I2C) is a protocol intended to allow multiple slaves communicate with multiple masters. Only two wires, short distance communication. One overhead bit for meta data. Data rates between async serial and SPI, hardware complexity between SPI and async serial, it's a best of both worlds situation.



Each I2C bus consists of a SCL (clock) and an SDA (data) signal. SCL is generated by the current bus master, but a slave may force the clock low at times to delay the master from sending more data (clock stretching).

Unlike UART or SPI connections, the I2C drivers are open drain, they can pull the signal line low, but cannot drive it high. Each line has a pull-up resistor on it, which restores the signal to high when no device is asserting it low. This means that there can be no bus contention on the line.

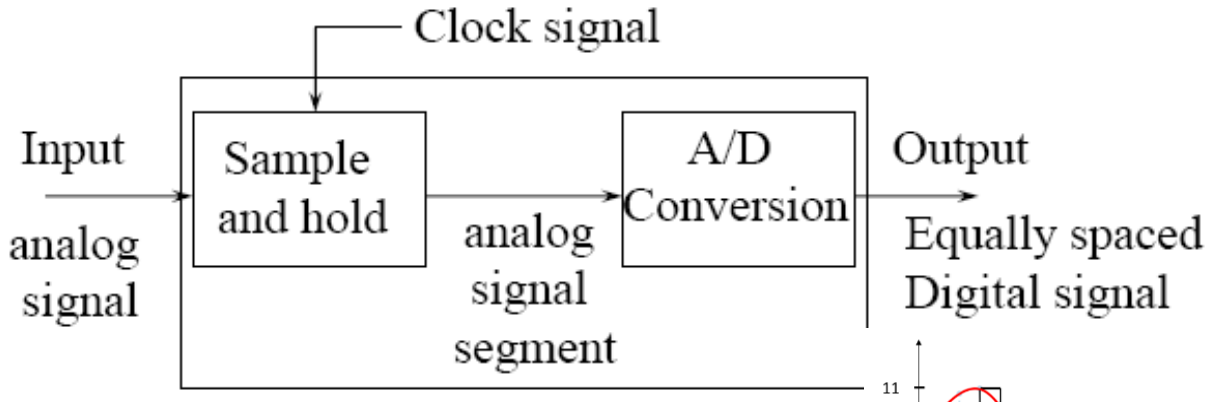
Communication via I2C is more complex than the previous solutions. Messages are broken up into an address frame, where the master indicates the slave receiver, and one or more data frames, which are 8-bit data messages passed from the master to the selected slave or vice versa. Data is placed on SDA when SCL goes low, and is sampled when SCL goes high.

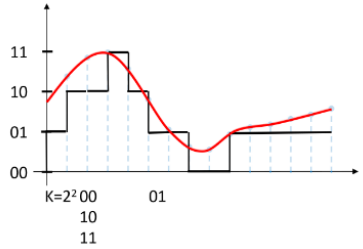


The master leaves SCL high and pulls SDA low to initiate the address frame. This notifies all slaves that transmission is starting. If several masters wish to take ownership of the bus at the same time, the device that pulls SDA low first gains control. The address frame goes first, with the 7-bit address (MSB first), followed by a R/W bit, and then the NACK/ACK bit. Once the first 8 bits are sent, the receiver gains control of SDA. If the device does not pull SDA low before the next pulse, the device did not receive or understand the data, and the exchange halts. The master then decides what's next. Otherwise the transmission can continue. Data frames will be put on SDA either by slave or master. When all the data frames have been sent, the master will generate a stop condition, defined by low to high on SDA after the clock goes high. Normally SDA will never change while the clock is high.

ADC

Analog to digital converters convert analog signals into binary words, from continuous to discrete form. This IC provides a link between analog transducers and the digital world of signal processing and data handling.





The two main steps are sampling and holding, and quantization and encoding. The sample and hold process measures the analog signal at a fixed time interval.

We then quantize the signal, we separate the input signal into discrete states with K increments. $K = 2^n$, where n is the number of bits of the ADC. The resolution is $Q = (V_{max} - V_{min})/K$.

Finally, we encode the signal, assigning a unique digital code to each state for input into the microprocessor.

We can improve the accuracy of the conversion by increasing the sampling rate T_s or the resolution Q.

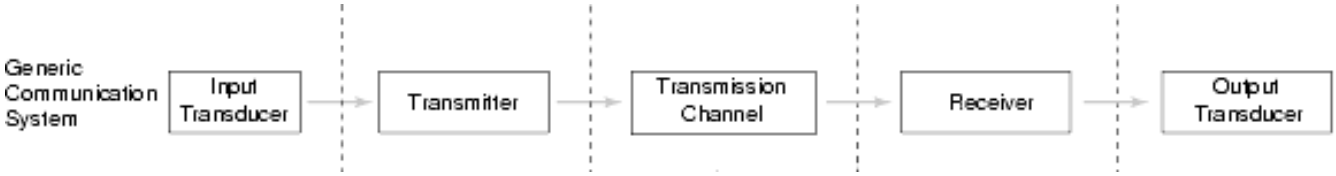
Example: 3 bit ADC $\Rightarrow$ n = 8, Q = 1.25V when U = 10V. We then assign each voltage step to an output state from 0-7, which is then encoded to a binary equivalent from 000 – 111. Pretty basic stuff.

Nyquist sampling theorem: in order to do a proper sampling (you can exactly recreate the sinusoidal signal), and avoid aliasing, the sampling frequency must be at least double the signals highest frequency.

Sampling considerations: workspace (interval from lowest to highest analogue value), scope (O, the difference between highest and lowest value), LSB (O / 2^n, really the same as resolution), quantization and dynamic range (ratio between largest and smallest values in Db, max dynamic range will be $20\log(2^n)$)

Communications systems

Modulasjonsteknikker (AM, FM, PWM) - hva de er godt for, og hvorfor vi bruker dem



Input transducer: device that converts physical signal from source to electrical, mechanical or electromagnetic signal more suitable for communicating.

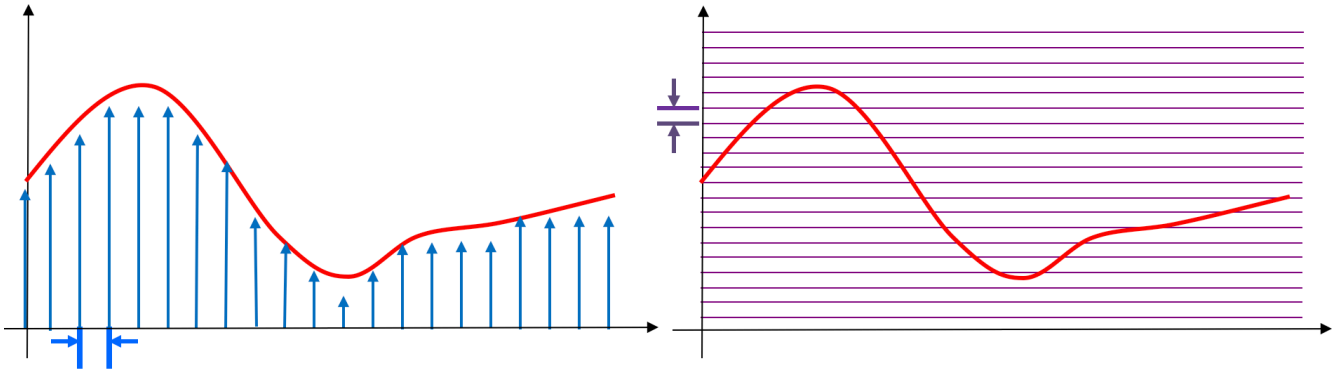
Transmitter: the device that sends the transduced signal.

Transmission channel: physical medium on which the signal is carried.

Receiver: the device that recovers the transmitted signal from the channel.

Output transducer: device that converts signal to a useful quantity.

Baseband: band of frequencies in the original signal delivered to the transducer.

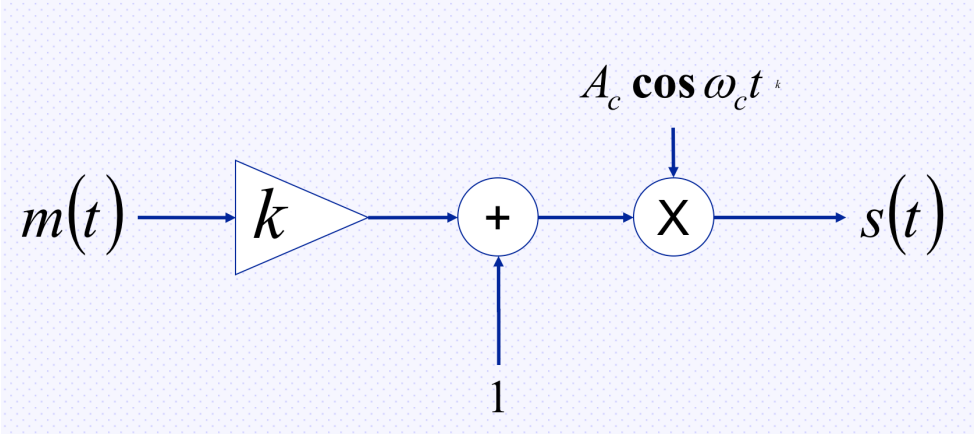


Often have to shift the range of the baseband for transmission using modulation and demodulation. The transmitter will then include a modulator device, and the receiver will include a demodulator device. The modulator modulates the carrier wave using the modulating signal.

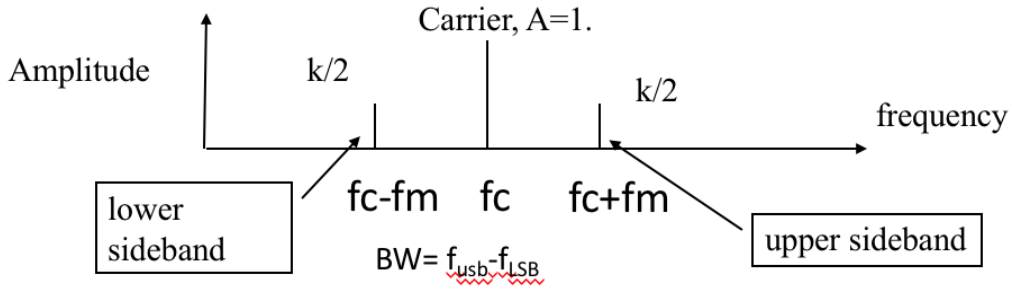
Amplitude modulation

Amplitude modulation (AM): varying carrier wave amplitude in proportion to baseband signal amplitude, with a fixed frequency.

The AM Signal: $s_{AM}(t) = [A_c + m(t)]\cos(2\pi f_c t)$



Modulation index: $k = A_m / A_c$



AM power: $P_t = P_c (1 + k^2 / 2)$

Important that $V_m < V_c$, otherwise we get distortion from overmodulation ($k > 1$).

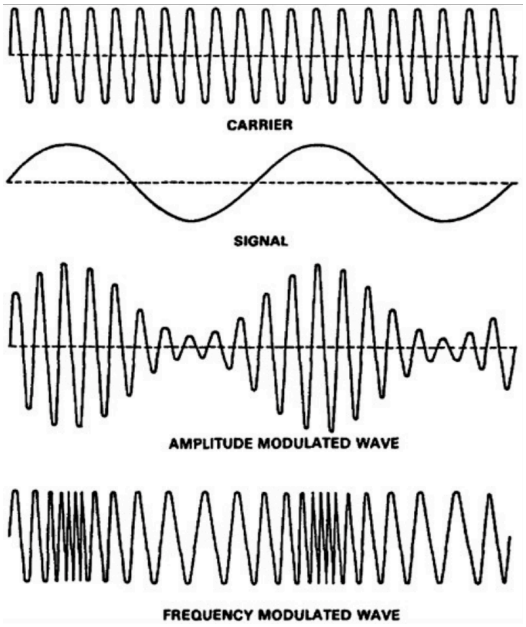
$V_m = (V_{max} - V_{min}) / 2$, $V_c = (V_{max} + V_{min}) / 2$

2 main methods of AM demodulation: **envelope**/non-coherent detection or **synchronized**/coherent demodulation. Envelope detector uses a half-wave rectifier with smoothing capacitor, requires that m << 1. Otherwise we must use a synchronous demodulator, multiplying input with oscillator signal at carrier frequency and low pass filtering it, which is relatively more complex and expensive.

Frequency modulation

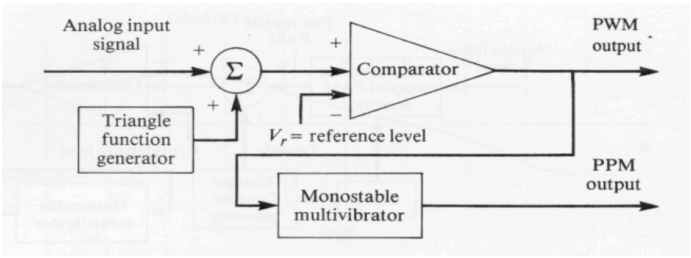
Frequency modulation (FM): varying carrier wave frequency in proportion to baseband signal amplitude. Need a much higher bandwidth than original signal, typically a magnitude. While the total bandwidth required for AM is simply two times the original bandwidth, the FM bandwidth is not so easy to calculate. An estimate (Carsons's rule) states that:

$B_T = 2(\Delta f + f_m)$ Hz, where Δf is the peak frequency deviation of the modulated signal and f_m is the max baseband frequency.

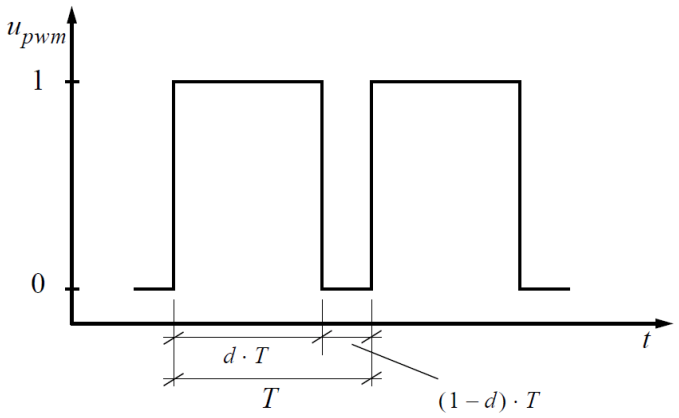


PWM

In PWM the sample values of the analog waveform are used to determine the width of a pulse signal. In PPM the values determine the position of a narrow pulse relative to the clock.



Example: controlling a servo motor to a specific angle. Uses less energy than a linear motor driver. Also very robust against noise, and easy to implement digitally or analogously.



Duty cycle d is the ratio between the pulse width and the clock period.

Demodulation can be done with an LPF, but since most physical processes already have this dynamic it's often not necessary.

Practical aspect: f = 1/T > 20kHz to avoid noise.

Control

Open loop control (no sensors) is ok for simple low-precision systems. Closed loop: use sensor data to estimate the system state and make a control command. Issues: imperfect state information, stability from delay. **Bang-bang control** is the simplest controller: actuator set to increase when error is negative and set to decrease when error is positive. Analog values have noise though, so we introduce a dead band around the set point where no changes are made, i.e. hysteresis. An example is thermostats. We can use an RTD, a resistance that is proportional to temperature. This is a simple system, but always uses max control energy. PID control is an option to ON/OFF control. With PWM we can achieve a stable state that does not oscillate. Shorter cycles are naturally better, but shortens the lifespan of the relay.

Industrial instrumentation

A measurement instrument is a device capable of detecting change in a particular process. It then converts these changes into some form of information understandable to the user. We can get the information using direct indication (monitoring the current value) or a recorder (stores the information, allows us to see the past states as well).

A measurement instrument consists of a sensor (device that changes its physical properties as a result of changes in the process), an amplifier/conditioner and a display.

When an instrument has the ability to send information, we call it a transmitter.

We can classify instruments in in-field and panel instruments. In-field instruments is installed close to the process, while panel instruments are in a controlled-environment room. We can also classify instruments in pneumatic and electronic instruments. Pneumatic do not consume electricity, but are imprecise and limited in output and range. Electronic instruments can be classified in analog, digital and smart analog. Smart analog instruments do self-diagnosis. Digital electronic instruments can transmit multiple values on a bus.

Range: minimum and maximum value interval.

Span: difference between max and min value.

Elevation: the lower limit of the range, if it is positive.

Depression: the absolute value of the lower limit of the range, if it is negative.

Overrange: if the measured values are outside the range.

Error: difference between measured and actual value

Reference value: actual, expected or desired value of a variable.

Accuracy: defines the limits of the error, e.g. if the accuracy is 1% then the readings do not differ from the actual value by more than 1% of the span. "Resolution" of error.

Static characteristics are considered for instruments used to measure an unvarying process condition.

Dynamic characteristics are concerned with the measurement of quantities that vary with time.

Sensitivity: ratio between output change and input change for an instrument.

Levels of measurement:

Nominal: states are only names, e.g. male=1, female=2.

Ordinal: states can be ordered, e.g. grades.

Interval: distance is meaningful, differences between the numbers reflect differences in the attribute, e.g. temperature scale.

Ratio: absolute zero is required, ratios between numbers reflect ratios of the attribute, e.g. meters.

Log interval: ratios reflect attribute, but not differences, e.g. density.

Absolute: absolute reference is required; all properties reflect analogous properties of the attribute.

Error in measurements

Error is the difference between the true value of the measurand and the measured value. Can be expressed as absolute error or as a percentage of excepted value.

Relative accuracy: $A = 1 - \text{percentage error} = 1 - \text{abs}((Y_n - X_n)/Y_n)$

Systematic errors come from instrument malfunction and human error.

Accidental error comes from imprecision in measurements.

Deviation: a data point's deviation from the arithmetic mean.

Average deviation: the arithmetic mean of the deviations, an indication of the precision of the instrument.

The standard deviation is the square root of the sum of all the individual deviations squared, divided by the number of readings.

If the errors are purely random, the deviations should follow definite statistical laws. When n is very large, the distribution will approach the normal distribution.

We sum the error when we sum/subtract quantities. We sum the percentage errors when we multiply/divide quantities.

Transmission lines

When the physical dimension of a circuit approaches the magnitude of a wavelength of the signal, we get transmission line effects. If the rise time is less than twice the propagation delay, transmission line effects must be considered.

Consider a line, with inductance, capacitance, resistance and conductance distributed along the transmission line. By analyzing an infinitesimal part of the line, we find the telegraph equations that describe the dynamics of the transmission line.

DEL III: Programmering i C

Kunne bruke C idiomatisk, og kunne benytte mer avanserte konsepter som structer og pekere

Types, Operators and Expressions

Control Flow

Functions and Program Structure

Pointers and arrays

Structures

Input and Output

Code verification

Ideally want to prove that any given program is correct. More pragmatically want to convince yourself that a program probably works.

Path testing: make sure all logical paths are executed, not possible with complex programs, only realistic to path test fragments of code.

The further we get into the development cycle, the more expensive corrections are.

Unit tests are additional software functions you write to test the different units in your system, typically a single source file in C. The unit test calls functions in different orders and with varying parameters etc. to verify that the modules behave as the spec says. In unit testing we treat the unit as a black box.

Problem: dependencies across modules. To solve this, we have to create a mock, a fake implementation of a module that allows us to simulate the interactions. For instance, in order to test a temperature sensor driver with conversion logic, we need to create a mock of the I2C driver that talks to the temperature sensor in order to test the logic in isolation. This also removes any hardware dependencies from the test!

IV: The development process

Waterfall process: breaks down a project based on activity: requirements analysis, design, coding, testing.

Iterative process: breaks down a project by subsets of functionality, more modular, gives better feedback on progress, time boxing

Predictive and adaptive planning: predictive is preferred but depends on a precise, accurate and stable set of requirements.

Requirement analysis:

Functional specifications: includes the functionality of the system, description of interface requirements (such as definition of data entry fields), details like data consistency, workflow, user experience etc.

Non-functional specifications: extensibility of the system (ability to add features later), scalability (will the current build have capacity issues later on), performance and response time, speed, resource constraints (like bandwidth, memory).

Quality attributes like maintainability, security, portability.

Design constraints on the implementation, like policies, environments, languages etc.

The V-model:

The V-model is an extension of the waterfall model, where every phase of the SDLC has a corresponding testing phase. The model is split in a verification side (dev) and a validation side (testing).

Timeline:

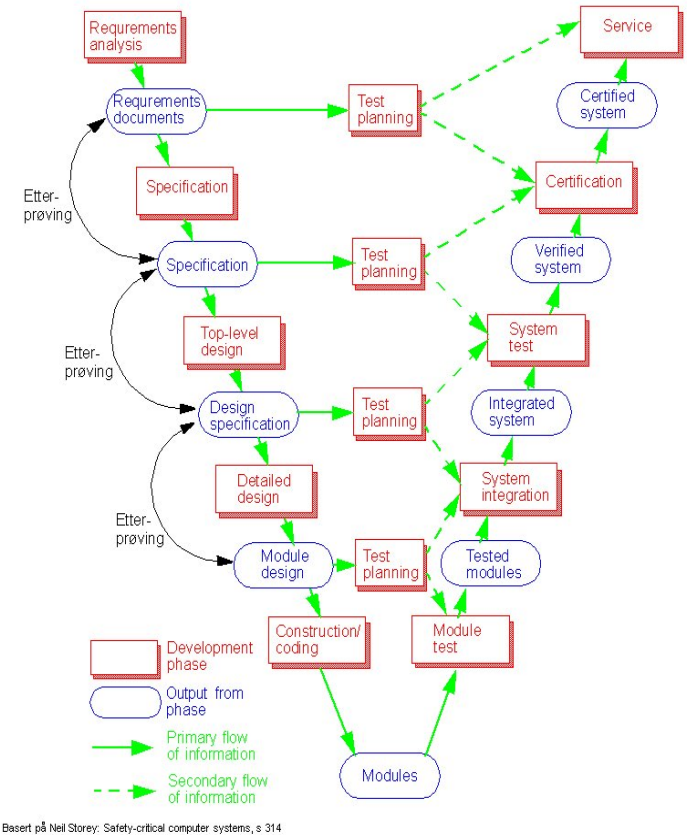
Requirements analysis: information is gathered from end user, creating software requirement specification.

System design / functional design: prepare functional design of the system.

Architecture design: interface relationships, database tables, dependencies and similar high-level design.

Low-level design: communication protocols, memory management etc.

Then we *code* the different modules.



Basert på Neil Storey: Safety-critical computer systems, s 314

Unit testing: testing the individual modules.

Integration testing: integrate the different modules and test.

System testing: testing out the entire system from the perspective of the end user.

User acceptance testing: check if the system satisfies the requirement specification.

In the V model, testing activities start from the very beginning of design, while the waterfall model carries out testing after development is over. The resulting system is usually less faulty when testing plays such an important role in the V-model.