Inbyggda realtidssystem

**Dynamic range**

Digitala sensorer kan inte skilja mellan två nära avstånd från fysiska aktiviteter. Den dynamiska längden avgörs:

$$D=\frac{High-Low}{precision}\rightarrow D\_{db}=20 Log(\frac{High-Low}{precision})$$

D är oftast mätt i decibel.

**Quantization**Den faktiska fysiska kvantitet kan beskrivas som ett tal, men för sådana måste sensorn välja ifrån 2n för att representera det. Den ”omskrivningen” är quantization.
Tex:
Om vi har ett batteri på 3.5V. Så är bit:

2^24 – 1 = 3.5V

2^0 -1 = 0V

*Acuators are a subject for quantization error*. Eftersom omvandlingen sker från digital till analog, och digitala kan bara ha ett begränsat antal värden. Precisionen med vilken en analog åtgärd kan vidtas beror därför på antalet bitar av den digitala signalen och aktuatorns intervall.

En ADC kan också göra att det blir quantization fel. Det går dock att lösa med hjälp av ”signal conditining”. Om samplefrekvesnen är tillräckligt hög kan ”noise” bli reducerat utav ett låg-pass filter, det kallas ”oversampling”.

**Noise**

Den delen i signalen vi inte vill ha. It is useful to be able to characterize how much noise there is in a measurement. The root mean square (RMS) N ∈ R+ of the noise is equal to the square root of the average value of n(t)2.

**Sampling**A physical quantity x(t) is a function of time t. A digital sensor will sample the physical quantity at particular points in time to create a discrete signal. In uniform sampling, there is a fixed time interval T between samples; T is called the sampling interval.

A physical quantity x(t) is a function of time t. A digital sensor will sample the physical quantity at particular points in time to create a discrete signal. In uniform sampling, there is a fixed time interval T between samples; T is called the sampling interval.

An important concern when sampling signals is that there are many distinct functions x that when sampled will yield the same signal s. This phenomenon is known as **aliasing**.

**Embedded systems**

När man bestämmer processor är det viktigt att förstå skillnaden mellan

* Instruction set architecture (ISA)
* Processor realization
* Chip

ISA är en definition utav instruktioner som processorn kan utföra och vissa strukturella begräsningar. En enda ISA kan förekomma i många olika chip av olika tillverkare och har därför ofta varierande prestationsprofiler. Fördelarna med ISA är bland annat att utvecklingsverktygen kan användas för flera projekt, man behöver inte utveckla på nytt.

**Microcontrollers**

En microcontroller är en liten dator som är integrerad på ett bräde med en relativt simpel CPU och kombinerade med kringutrustning som minne, I/O och timers. Microcontrollers kan förbruka väldigt små mängder energi och har oftast även ett viloläge som gör att förbrukningen blir ännu mindre. Vissa sensor nät och övervakningssystem har visats kunna fungera i flera år med ett litet batteri på grund av detta.

**DSP processor**

DSP (digital signal processor) processorer är designade för specifika numeriska signalbehandlingar. För att få insikt i strukturen hos sådana processorer och konsekvenserna för den inbyggda system är det värt att förstå strukturen hos typiska signalbehandlingsalgoritmer.

**FIR (Finite impulse response)**

FIR är ett filter för digitala system. Kännetecknande för dessa är att impulssvaret är noll utanför ett begränsat intervall.

**GPU**

Specialiserad på beräkningar som användas i grafisk rendering.

**Parallellism**

**Concurrency** är möjligheten för ett program att delas upp i delar som kan köras oberoende av varandra. Det innebär att uppgifter kan utföras i ordning och resultatet blir fortfarande detsamma som om de körs i ordning. Det är förmågan hos en algoritm eller ett program kan köra mer än en uppgift åt gången.

**Multitasking** är något som tillåter att processer körs samtidigt, medan **multithreading** tillåter delprocesser köras samtidigt.

**Thread** är ett unikt utförande av en sekvens maskininstruktioner, som kan interfolieras med andra trådar som utförs på samma maskin. Thread delar minne.

**Process** can have one more multiple threads. Processes have their own memory.

**Lecture 4**

**Critical section** where shared resources usages.

**Context switching** is a stack pointer and program counter.

**Stack pointer** small register that stores the address of the last program request in a stack.

**Lecture 5**

**Scheduling**

When there are fewer processors than tasks, or tasks must be performed at a particular time, a *scheduler* is needed.

**Lecture 6**

**POSIX** – Portable Operating System Interface

**Memory model** – how memory is handled, what sort of guarantees in terms of executon order and atomicity can one get.

**Volatile** example)

Int c;

C = c+1;

C = c-1; 🡨 no-op

Volatile int c;

C = c+1;

C = c-1; 🡨 the compiler will not remove this even if it is useless.

**Race condition** – two threads ”fight” about a variable. Two threads trying to change it att the same time.