Embedded Information Systems

5. Quantities and variables in real-time systems

Power-aware systems

November 10, 2020

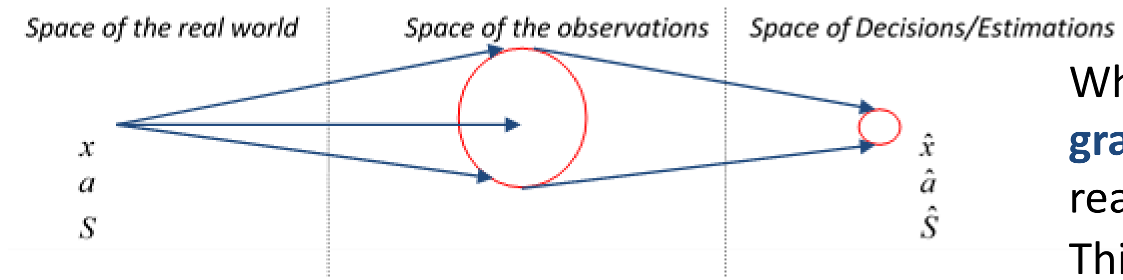
Reminder:

5. Quantities and variables in real-time systems

Known concepts: Real-time variable, real-time image, temporal accuracy, periodic update, observation: state observation, event observation, real-time object, permanence, action delay, idempotence, credibility (Byzantine errors), sphere of control, …

Modelling of the recipient environment: **What can not be avoided:**
The **cognition** of the recipient environment, and its **installation** into the software.

An important tool of this latter is the measurement process: which is an inherent part of the cognition process within which we are increasing and expanding our knowledge.



The figure helps the **interpretation.** While taking measurement, we try to **grasp** the different **phenomena** of the real world.

This is made preferably by quantities which show **stability.**

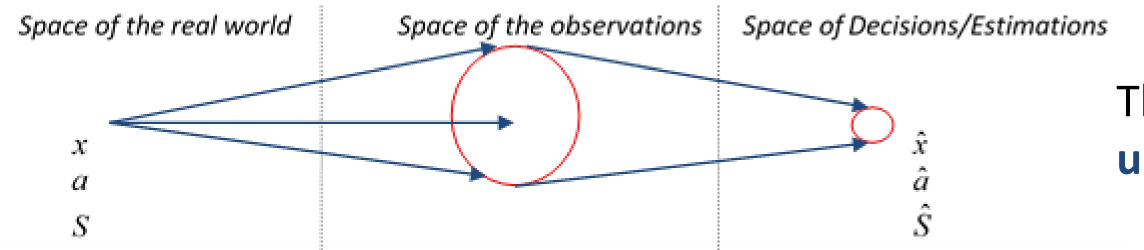
Obviously, such quantities are results of **abstractions.**

The following **quantities/features** play key role:

- ***state variables*** (*x*), the changes of which follow energy processes (voltage, pressure, temperature, speed, etc.) due to interactions;
- ***parameters*** (*a*), which characterize the strength of the interactions;
- ***structures*** (*S*), which describe the relations of the system components.

The ***Space of the real world*** is such an abstraction, where the values of the investigated features correspond to one point of the space.

Reminder:

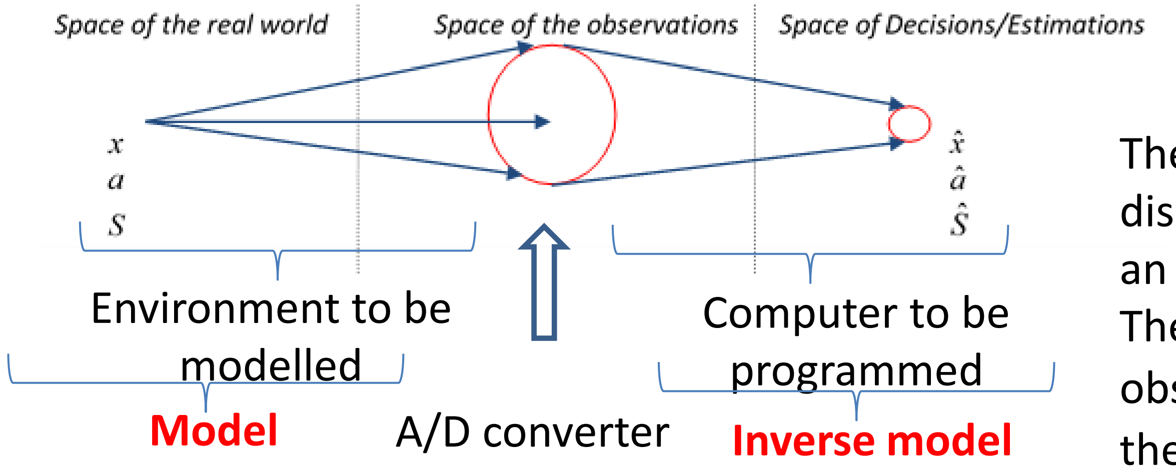


The coordinates of this points **are unknown** before the measurement.

It is well known, that due to measurement errors, only **an estimate** of the measurand can be provided.
Further difficulty, that there is no direct access to the quantity to be measured, only some kind of **indirect mapping** is possible.

This mapping is called **observation.**

The path from the quantity to be measured and the observation is called **measuring channel**.


Space of the real world
Space of the observations
Space of Decisions/Estimations
x
a
S
$\hat{x}$
$\hat{a}$
$\hat{S}$
Environment to be modelled
Computer to be programmed
Model
A/D converter
Inverse model

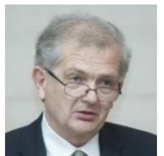
Observation in case of deterministic channel:

The observed reality is described by a discrete model, and it is supposed to be an autonomous system.
The state equations and the observation equation describing the reality and the observation:

$$\boxed{x(n+1) = Ax(n),} \quad dim[x(n)] = N, \quad dim[A] = N * N$$

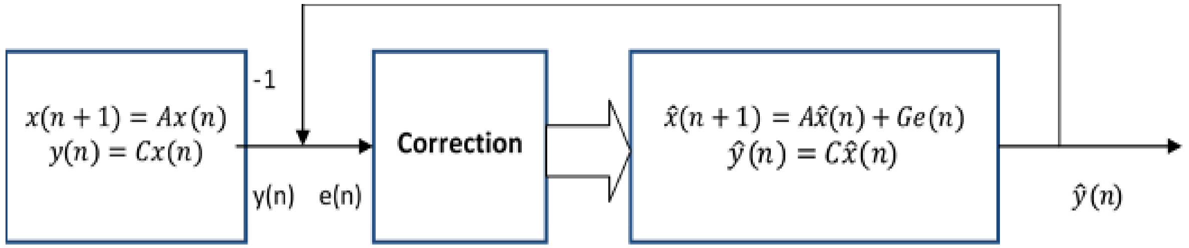
$$\boxed{y(n) = Cx(n)} \quad dim[y(n)] = M, \quad M \leq N, \ dim[C] = M * N$$

Can we invert such a system? **Generally not!** When is it possible? If C^{-1} exists!



If C^{-1} **does not exist**, then something else is to do! The solution is a **simulator!** Reminder:

The model should be built into the computer! This is controlled by the observed values!


$x(n+1) = Ax(n)$
$y(n) = Cx(n)$
-1
y(n)
e(n)
Correction
$\hat{x}(n+1) = A\hat{x}(n) + Ge(n)$
$\hat{y}(n) = C\hat{x}(n)$
$\hat{y}(n)$

Our aim is the estimation of the **state variable *x*(*n*).**

The name of this device is: **observer,** which tries to produce **a copy** of the reality by following it; thanks to a correction/training/adaptation mechanism, and gives **an estimate** of the value to be measured. This estimator $\hat{x}(n)$ **can be read** from the observer.

The state and the observation equations of the observer are:

$\boxed{\hat{x}(n+1) = A\hat{x}(n) + Ge(n)}$ G: **correction matrix**; $dim[G] = N * M$, $e(n) = y(n) - \hat{y}(n)$

$\boxed{\hat{y}(n) = C\hat{x}(n)}$ Matrix G should be designed to get: $\boxed{\hat{x}(n) \rightarrow x(n).}$

The difference of the state variables is to be minimized:

$$\underbrace{x(n+1) - \hat{x}(n+1)}_{\varepsilon(n+1)} = Ax(n) - A\hat{x}(n) - Ge(n) = \underbrace{(A - GC)}_{F}\underbrace{(x(n) - \hat{x}(n))}_{\varepsilon(n)}.$$

$$\boxed{\varepsilon(n+1) = F\varepsilon(n)}$$

Is the state equation of the **error system**.

The design of matrix G: $\boxed{\varepsilon(n) \xrightarrow[n\to\infty]{} 0,}$ therefore $\|\varepsilon(n+1)\| < \|\varepsilon(n)\|$, is forced, possibly for $\forall\, n$. F reduces the size of vector, i.e. it is „**contractive**”. This property can be interpreted also in such a way that the internal energy of the error system is **dissipated**.

If this is the case in every step, then the decrease of the size of the error vector will be a **monotonic process.**

Special cases:

Reminder:

1. $F = A - GC = 0$. In this case $G = AC^{-1}$. This is possible if C is a square matrix, i.e. the observation has as many components as the state vector itself.

$\boxed{\text{The equation can be solved in an explicite way!}}$ The system converges in one step!

2. $F^N = (A - GC)^N = 0$. In this case the error system converges in *N* steps:

$$\boxed{x(N) - \hat{x}(N) = (A - GC)^N (x(0) - \hat{x}(0)) = 0}$$

The matrices wth property $F^N = 0$ can be characterized by the fact that **all they eigenvalues are zero**.

Systems having state transition matrix of this property are of **finite impulse response** (**FIR**) systems, the initial error will disappear in finite steps.

3. If $F^N = (A - GC)^N \neq 0$, then the size of the state vector of a stabile error system will decrease **exponentially**. The error system is stable, if all its eigenvalues are within the unit circle. Systems having state transition matrix of this property have **infinite impulse response** (**IIR** systems), because the initial error **will disappear in infinite steps.**

Observation in the case of noisy observation channel:

In this case our expectation is not $\|\varepsilon(n)\| \xrightarrow[n\to\infty]{} 0$, but the trace of $E\{[\![\varepsilon(n)\varepsilon^T(n)]\!]\} \xrightarrow[n\to\infty]{} min.$

$$\varepsilon(n) = \begin{bmatrix} \varepsilon_0(n) \\ \varepsilon_1(n) \\ \vdots \\ \varepsilon_{N-1}(n) \end{bmatrix}$$

therefore $trace\{E[\varepsilon(n)\varepsilon^T(n)]\} = \sum_{k=0}^{N-1} E\{\varepsilon_k^2(n)\}$.

Instead of $\boxed{\varepsilon(n+1) = F\varepsilon(n)}$

$$\boxed{E[\varepsilon(n+1)\varepsilon^T(n+1)] = FE[\varepsilon(n)\varepsilon^T(n)]F^T}$$

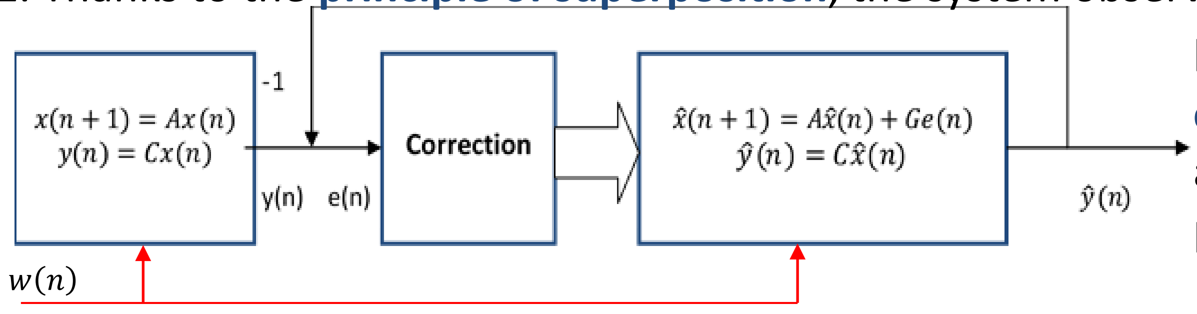
Is used for error system characterization.

Reminder:

Matrix $\boxed{P = E[\varepsilon(n)\varepsilon^T(n)]}$ plays a central role in the famous Kalman predictor and filter.

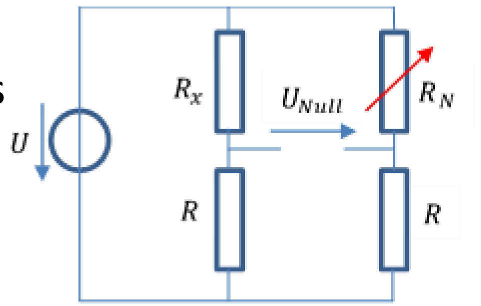
Comments:

1. Thanks to the **principle of superposition**, the system observed and the observer itself can have an additional, **common external excitation** without any change in the convergence properties.



2. The above introduced observer is called Luenberger observer. According to Luenberger **almost any system is an observer**. The only requirement is that the observer should be faster than the observed system; otherwise it will not be able to follow its changes.

3. The bridge-branch containing the **impedance to be measured** within an impedance measuring bridge implements the **physical model of the reality**, while the **tuneable bridge-branch** correspond to the model built into the **observer.**



The difference between the outputs of the voltage divider bridge-branches controls the correction mechanism. Finally the value of the unknown impedance will be computed from the correction value.

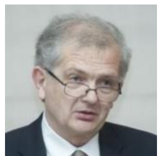
This setup, together with the operator responsible for tuning, implements an observer.

Example:

Given $A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}; C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

How to set matrix G?

$$\boxed{G = AC^{-1} = A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}}$$



Example: Given $A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$; $C = [1 \quad 1]$. How to set matrix G? $\boxed{G = \begin{bmatrix} g_0 \\ g_1 \end{bmatrix} = ?}$ Reminder:

$$GC = \begin{bmatrix} g_0 \\ g_1 \end{bmatrix} [1 \quad 1] = \begin{bmatrix} g_0 & g_0 \\ g_1 & g_1 \end{bmatrix}, \quad [A - GC] = \begin{bmatrix} 1 - g_0 & -g_0 \\ -g_1 & -1 - 1g_1 \end{bmatrix},$$ based on $[A - GC]^2 = 0$

$$\begin{bmatrix} 1 - g_0 & -g_0 \\ -g_1 & -1 - g_1 \end{bmatrix} \begin{bmatrix} 1 - g_0 & -g_0 \\ -g_1 & -1 - g_1 \end{bmatrix} = \begin{bmatrix} 1 - 2g_0 + g_0^2 + g_0 g_1 & -g_0 + g_0^2 + g_0 + g_0 g_1 \\ -g_1 + g_1^2 + g_1 + g_0 g_1 & 1 + 2g_1 + g_1^2 + g_0 g_1 \end{bmatrix} =$$

$= \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ Substituting expressions of the minor diagonal into the main diagonal we get:

$\boxed{1 - 2g_0 = 0,}$ $\boxed{1 + 2g_1 = 0,}$ where from: $g_0 = 0.5$ és $g_1 = -0.5$.

Checking:

$$\begin{bmatrix} 0.5 & -0.5 \\ 0.5 & -0.5 \end{bmatrix} \begin{bmatrix} 0.5 & -0.5 \\ 0.5 & -0.5 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Example: Let us compute eigenvalues of matrix $[A - GC]$: $det[\lambda I - A + GC] = 0$

$$det \begin{bmatrix} \lambda - 0.5 & 0.5 \\ -0.5 & \lambda + 0.5 \end{bmatrix} = (\lambda - 0.5)(\lambda + 0.5) + 0.25 = \lambda^2 - 0.25 + 0.25 = 0.$$

$\boxed{\text{Both eigenvalues are zero.}}$

Comments:

1. This property is **valid** in every system capable to converge **in finite steps**.
2. The transfer function of such systems is a rational function having **all its poles at the origin**:

$$H(z) = a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N} = \frac{a_N + a_{N-1} z + a_{N-2} z^2 + \ldots + a_1 z^{N-1}}{z^N}$$

These are the so-called **Finite Impulse Response** (**FIR**) filters.

The time-domain equivalent: $y(n) = a_1 x(n-1) + a_2 x(n-2) + \ldots + a_N x(n-N)$,

where **due to computability reasons** only previous samples of $x(n)$ are used.

Reminder:

Example: The **condition for the eigenvalues** can be used to compute g_0 and g_1:

$$det[\lambda I - A + GC] = 0,\ det\begin{bmatrix}\lambda - 1 + g_0 & g_0 \\ g_1 & \lambda + 1 + g_1\end{bmatrix} = \lambda^2 + \lambda(g_0 + g_1) + g_0 - g_1 - 1 =$$

$= \lambda^2 = 0.$ Where from: $g_0 + g_1 = 0$ and $g_0 - g_1 = 1,$ thus: $g_0 = 0.5$ és $g_1 = -0.5.$

It happens that the dynamics of recipient environment is not known:
The state equation is not known or $\boxed{A = I.}$ In this case we have only observations.

Linear Least Squares (LS) Estimation: No a priori information is available neither from the parameter to be measured, nor from the channel characteristics/noise. Let us assume that the observation equation is linear: $y(n) = Cx(n) + w(n);\ w(n)$ is the observation noise.
We assume that the unknown $x(n)$ takes value $\hat{x}(n)$, and we set up **the model of the observation.** We compare this with the observation, and we are looking for the best setting of $\hat{x}(n)$ assuming squared error: $J\big(x(n), \hat{x}(n)\big) = \big(y(n) - C\hat{x}(n)\big)^T\big(y(n) - C\hat{x}(n)\big) =$
$= y(n)^T y(n) - y(n)^T C\hat{x}(n) - \hat{x}(n)^T C^T y(n) + \hat{x}(n)^T C^T C\hat{x}(n) =$
$= y(n)^T y(n) - 2\hat{x}(n)^T C^T y(n) + \hat{x}(n)^T C^T C\hat{x}(n)$ The minimum of which is given by:

$$\boxed{\left.\frac{\partial J\big(x(n), \hat{x}(n)\big)}{\partial \hat{x}(n)}\right|_{\hat{x}(n) = \hat{x}_{LS}} = 0} \quad -2C^T y(n) + 2C^T C\hat{x}(n) = 0,\ \boxed{\hat{x}(n) = [C^T C]^{-1} C^T y(n)}$$

Due to $A = I$ here $x(n)$ is practically a constant parameter.
We take more observations, and the (noisy) observed values are collected into vector $y(n)$.
If $C = [1 \quad 1 \quad \cdots \quad 1]^T$, then

$$\boxed{\hat{x}_{LS} = \frac{1}{N}\sum_{k=0}^{N-1} y_k(n)}$$

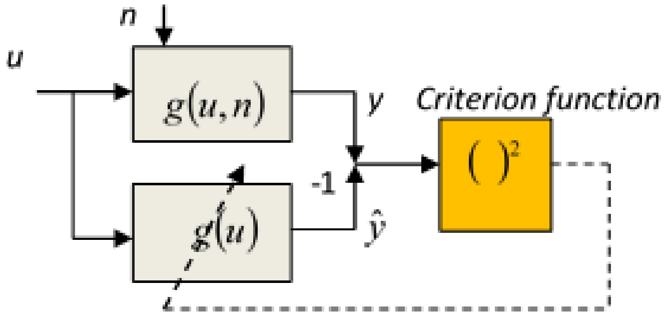
i.e. the result is simple **linear averaging.**

Model fitting:

In the case of LS estimators, we do not have prior informat Reminder: parameter to be measured, therefore what we do is **model fitting**.

The problem of model fitting is manifold.

A classical version is **regression calculus**: The determination of a possibly deterministic relation of independent and dependent variables can be considered as a special case of model fitting.



In the figure below the function to be modelled $y = g(u,n)$ has two types of independent variables: the one denoted by $u(n)$, is known and **can be influenced**, while the other, denoted by $n(n)$, is unknown, and **cannot be influenced.** This latter is typically a noise process, or disturbance modelled as a noise process.

For modelling we use a "tuneable" function $\hat{y} = \hat{g}(u)$ the free parameters of which are tuneable. We aim at applying such setting of parameters which is optimal in some sense. Typically quadratic criterion is applied:

$$\boxed{J = E\{(y-\hat{y})^T(y-\hat{y})\}}$$

Linear regression: The function to be fitted is a scalar linear function $\hat{g}(u) = a_0 + a_1 u$, the parameters of which are set to minimize $E\left\{\left(y-\hat{g}(u)\right)^2\right\}$.

To get the minimum of $J = E\{(y - a_0 - a_1 u)^2\}$ we take the derivative to a_0 and a_1 :

$$\frac{\partial J}{\partial a_0} = -2E\{(y - a_0 - a_1 u)\} = -2(E\{y\} - a_0 - a_1 E\{u\}) = 0$$

$$\frac{\partial J}{\partial a_1} = -2E\{u(y - a_0 - a_1 u)\} = -2(E\{uy\} - a_0 E\{u\} - a_1 E\{u^2\}) = 0$$

Reminder:

Polynomial regression: Important property that it is linear in its parameters.

$$\hat{g}(u) = \sum_{k=0}^{N} a_k u^k$$

We prefer **models linear in their parameters**, because in case of squared error criterion, finding the optimum requires the solution of a set of **linear equations**.

Linear regression based on measured data:

The above development can be carried out also for the case where no a priori information is available. $\boxed{\text{In this case } y_n = a_0 + a_1 u_n + w_n,}$

which is the model of the observation at the n-th time instant.

Let us take N observations! The input and the observed samples are ordered into vectors.

$$\underbrace{\mathbf{z} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{bmatrix}}_{y(n)} = \underbrace{\begin{bmatrix} 1 & u_0 \\ 1 & u_1 \\ \vdots & \vdots \\ 1 & u_{N-1} \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} a_0 \\ a_1 \end{bmatrix}}_{* x(n)} + \underbrace{\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{N-1} \end{bmatrix}}_{+w(n)}$$

We recognize that this structure is the same as the model of the LS estimation!

$$\boxed{\hat{x}(n) = [C^T C]^{-1} C^T y(n)}$$

$$[\mathbf{C}^T \mathbf{C}] = \begin{bmatrix} N & \sum_{n=0}^{N-1} u_n \\ \sum_{n=0}^{N-1} u_n & \sum_{n=0}^{N-1} u_n^2 \end{bmatrix},$$

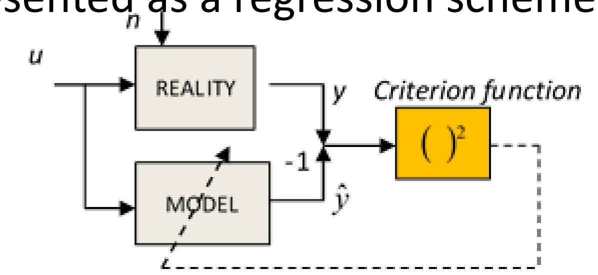
$$\mathbf{C}^T \mathbf{z} = \begin{bmatrix} \sum_{n=0}^{N-1} y_n \\ \sum_{n=0}^{N-1} u_n y_n \end{bmatrix},$$

$$\boxed{\begin{bmatrix} \hat{a}_0 \\ \hat{a}_1 \end{bmatrix} = \frac{1}{\frac{1}{N}\sum_{n=0}^{N-1} u_n^2 - \left(\frac{1}{N}\sum_{n=0}^{N-1} u_n\right)^2} \begin{bmatrix} \frac{1}{N}\sum_{n=0}^{N-1} u_n^2 & -\frac{1}{N}\sum_{n=0}^{N-1} u_n \\ -\frac{1}{N}\sum_{n=0}^{N-1} u_n & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{N}\sum_{n=0}^{N-1} y_n \\ \frac{1}{N}\sum_{n=0}^{N-1} u_n y_n \end{bmatrix},}$$

We recognize the approximations of the statistical characterizations!

Generalization of the regression scheme:

The model fitting problem is presented as a regression scheme:

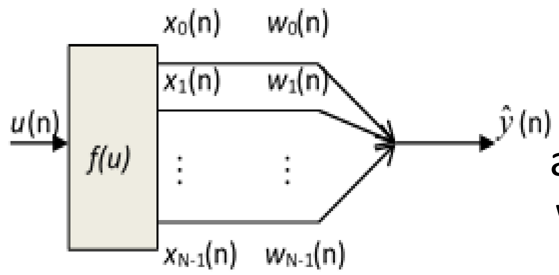


The response y to the input u is to be compared with the response $\hat{y}$ of the model.

It worth comparing this structure with the **observer scheme**: the similarity is obvious; we are fitting a model in both cases.

For the **observer** we know the **parameters**, and the **state variables** are to be estimated, while for the regression scheme the state variables are known and the parameters are to be estimated. Both schemes are **parallel.**

Adaptive linear combiner



A frequently used model-family. The model consists of two parts. The fix part generates a sequence of values

$$\boldsymbol{X}^T(n) = [x_o(n) \quad x_1(n) \quad \dots \quad x_{N-1}(n)]$$

and the linear combination of these values results in $\hat{y}(n)$.

We are looking for the minimum squared error by searching the optimum $\mathbf{W}^T(n) = [w_0(n) \quad w_1(n) \quad \dots \quad w_{N-1}(n)]$ setting.

We are minimizing $J(\boldsymbol{W}(n)) = E\{[y(n) - \boldsymbol{X}^T(n)\boldsymbol{W}(n)]^T[y(n) - \boldsymbol{X}^T(n)\boldsymbol{W}(n)]\}=$

$$= E\{y^T(n)y(n)\} - 2\boldsymbol{W}^T(n)E\{\boldsymbol{X}(n)y(n)\} + \boldsymbol{W}^T(n)E\{\boldsymbol{X}(n)\boldsymbol{X}^T(n)\}\boldsymbol{W}(n).$$

Let us introduce the following notations: $\boxed{E\{\boldsymbol{X}(n)y(n)\} = \boldsymbol{P}}$ $\boxed{E\{\boldsymbol{X}(n)\boldsymbol{X}^T(n)\} = \boldsymbol{R}}$

The minimum will be obtained at

$$\boxed{\frac{\partial J(\boldsymbol{W}(n))}{\partial \boldsymbol{W}(n)} = -2\boldsymbol{P} + 2\boldsymbol{R}\boldsymbol{W}(n) = 0}$$

Thus the optimum setting is: $\boxed{\boldsymbol{W}^* = \boldsymbol{R}^{-1}\boldsymbol{P}}$

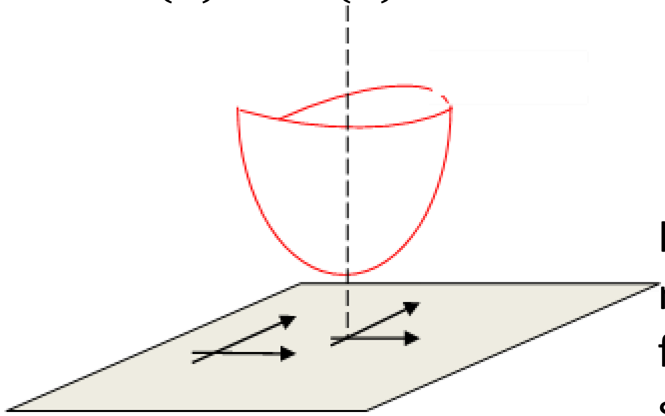
This is called **Wiener-Hopf equation**.

The optimum setting results in minimum error, which can be derived by substitution:

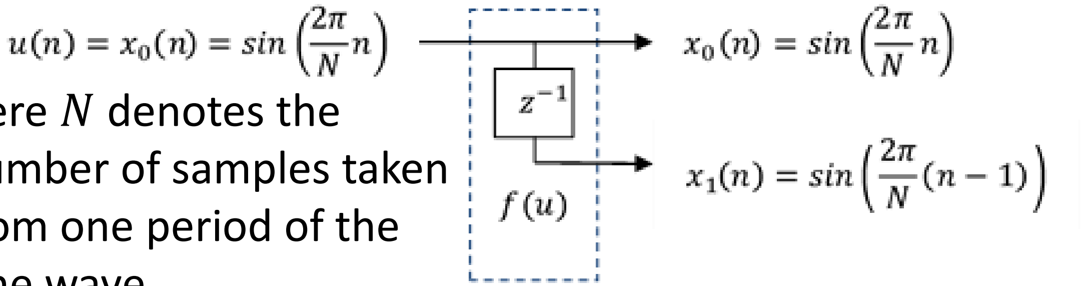
$J_{min} = E\{y^T(n)y(n)\} - \boldsymbol{P}^T\boldsymbol{R}^{-1}\boldsymbol{P} = E\{y^T(n)y(n)\} - \boldsymbol{P}^T\boldsymbol{W}^*$ Using this we get:

$$\boxed{J(\boldsymbol{W}(n)) = J_{min} + (\boldsymbol{W}(n) - \boldsymbol{W}^*)^T\boldsymbol{R}(\boldsymbol{W}(n) - \boldsymbol{W}^*) = J_{min} + \boldsymbol{V}^T(n)\boldsymbol{R}\boldsymbol{V}(n)}$$

where $\boldsymbol{V}(n) = \boldsymbol{W}(n) - \boldsymbol{W}^*$ stands for the parameter error. **Example:**



Imagine $\boldsymbol{X}^T(n) = [sin(2\pi n/N) \quad sin(2\pi(n-1)/N)]$, i.e. two subsequent samples of a discrete sine wave.



$u(n) = x_0(n) = sin\left(\frac{2\pi}{N}n\right)$
z^{-1}
$f(u)$
$x_0(n) = sin\left(\frac{2\pi}{N}n\right)$
$x_1(n) = sin\left(\frac{2\pi}{N}(n-1)\right)$

Here N denotes the number of samples taken from one period of the sine wave.

The signal to be approximated is: $y(n) = 2\,cos(2\pi n/N)$. $\boldsymbol{W}^T(n) = [w_0(n) \quad w_1(n)]$

The matrices $\boldsymbol{R}$ and $\boldsymbol{P}$ can be computed by averaging sine and cosine waveforms for the complete period:

$E\{x_0^2(n)\} = E\{x_1^2(n)\} = E\{(sin^2(2\pi/N)\} = E\{sin^2(2\pi(n-1)/N)\} = 0.5,$

$E\{x_0(n)x_1(n)\} = E\{sin(2\pi n/N)\,sin(2\pi(n-1)/N)\} = 0.5\,cos(2\pi/N)$

$E\{x_0(n)y(n)\} = E\{2\,sin(2\pi n/N)\,cos(2\pi n/N)\} = 0,$

$E\{x_1(n)y(n)\} = E\{2\,sin(2\pi(n-1))/N\,cos(2\pi n/N)\} = -sin(2\pi/N).$

$$\boxed{\boldsymbol{R} = \begin{bmatrix} 0.5 & 0.5\,cos\left(\frac{2\pi}{N}\right) \\ 0.5\,cos\left(\frac{2\pi}{N}\right) & 0.5 \end{bmatrix}}$$

$$\boxed{\boldsymbol{P} = \begin{bmatrix} 0 \\ -sin\left(\frac{2\pi}{N}\right) \end{bmatrix}.}$$

Computing the inverse of $\boldsymbol{R}$:

$$\boldsymbol{R}^{-1} = \frac{1}{0.25\,sin^2\left(\frac{2\pi}{N}\right)}\begin{bmatrix} 0.5 & -0.5\,cos\left(\frac{2\pi}{N}\right) \\ -0.5\,cos\left(\frac{2\pi}{N}\right) & 0.5 \end{bmatrix}$$

$$\boxed{\boldsymbol{R} = \begin{bmatrix} 0.5 & 0.5\,cos\left(\frac{2\pi}{N}\right) \\ 0.5\,cos\left(\frac{2\pi}{N}\right) & 0.5 \end{bmatrix}}$$

$$\boxed{\boldsymbol{W}^* = \boldsymbol{R}^{-1}\boldsymbol{P} = \begin{bmatrix} \frac{2}{tan(2\pi/N)} \\ -\frac{2}{sin(2\pi/N)} \end{bmatrix}}$$

With this setting the output of the model:

$$\boxed{\boldsymbol{P} = \begin{bmatrix} 0 \\ -\,sin\left(\frac{2\pi}{N}\right) \end{bmatrix}.}$$

$$\hat{y}(n) = X^T(n)W^* = 2\frac{sin(2\pi\, n/N)}{tan(2\,\pi/N)} - 2\frac{sin(2\pi(n-1)/N)}{sin(2\,\pi/N)} = 2cos(2\pi n/N)$$

Since cosine waveforms can be generated as linear combination of different phase sine waves, therefore for the case of the example J_{min}, i.e. the lowest point of the error surface (paraboloid) reaches the hyper-plane of the parameters.

Towards adaptive procedures:

At any point of the error surface the relative rate of the achievable error reduction can be measured by the gradient:

$$\boxed{\boldsymbol{\nabla}(n) = \frac{\partial J(\boldsymbol{W}(n))}{\partial \boldsymbol{W}(n)} = 2\boldsymbol{R}[\boldsymbol{W}(n) - \boldsymbol{W}^*]}$$

Multiplying both sides from the left by matrix $\frac{1}{2}\boldsymbol{R}^{-1}$:

$\boldsymbol{W}^* = \boldsymbol{W}(n) - \frac{1}{2}\boldsymbol{R}^{-1}\boldsymbol{\nabla}(n)$, Where we utilized: $\boxed{\boldsymbol{W}^* = \boldsymbol{R}^{-1}\boldsymbol{P}}$

Assuming that our knowledge about matrix $\boldsymbol{R}$ and thus about the gradient is not complete, this equation can be rewritten into an iterative form:

$$\boldsymbol{W}^* = \boldsymbol{W}(n) - \frac{1}{2}\boldsymbol{R}^{-1}\boldsymbol{\nabla}(n) \quad \Longrightarrow \quad \boldsymbol{W}(n+1) = \boldsymbol{W}(n) - \frac{1}{2}\widehat{\boldsymbol{R}}^{-1}\widehat{\boldsymbol{\nabla}}(n),$$

And finally by introducing the convergency factor $\boldsymbol{0 < \mu < 1}$, and replacing the perfect matrix $\boldsymbol{R}$ and the perfect gradient:

$$\boxed{\boldsymbol{W}(n+1) = \boldsymbol{W}(n) - \mu\boldsymbol{R}^{-1}\boldsymbol{\nabla}(n)} \quad (*)$$

Comments:

1. 1. If matrix $\boldsymbol{R}$ matrix and the gradient are perfectly known, then setting $\mu = \frac{1}{2}$ provides one-step convergence from an arbitrary (but finite) initial value $\boldsymbol{W}(n)$.
2. Since $\boldsymbol{\nabla}(n) = 2\boldsymbol{R}[\boldsymbol{W}(n) - \boldsymbol{W}^*]$, therefore substituting it into equation (*) and subtracting from both sides the value of W^* we get:

$$\boxed{\boldsymbol{W}(n+1) - \boldsymbol{W}^* = (1-2\mu)(\boldsymbol{W}(n) - \boldsymbol{W}^*) = \boldsymbol{V}(n+1) = (1-2\mu)^{n+1}\boldsymbol{V}(0),} \quad (**)$$

i.e. the initial error will decrease exponentially, if $\mu \neq \frac{1}{2}$.
If $0 < \mu < 0.5$, then the error will decrease monotonically, otherwise with oscillating sign.

3. The gradient methods of model fitting are distinguished by the a priori knowledge available to evaluate (*).

   If the $\boldsymbol{R}$ and $\boldsymbol{P}$ matrices are known, the equations describing the behaviour of adaptive linear combinator are (*) and (**).
4. Note that matrix $\boldsymbol{R}$ provides „global" information about the error surface, while gradient $\boldsymbol{\nabla}(n)$ gives a „local" characterization of the error surface. Based on this local information we descend on the error surface to be closer and closer to the minimum.

Replica determinism: The reliability of a system can be improved by active redundancy, i.e. by replicated physical components. A set of replicated RT objects is *replica determinate* if all the members of this set have (1) the same externally visible **RAM** state,

(2) and produce **the same output messages** at points in time that are at most an interval of d time unit apart. In a fault-tolerant system, the time interval d determines the time it takes to replace a missing message or an erroneous message from a node by a correct message from redundant replicas.

Example: Consider an airplane with a three-channel flight-control system and a majority voter. Each channel has its own sensors and computers to minimize the possibility of a common-mode error.

Within a specified time after the event "**start of take-off**", the control system must check whether the plane **has attained the take-off speed**.

If yes, the lift-off procedure is initiated, and the engines are **further accelerated**.

If not, the take-off must be aborted, and the engines **must be stopped**.

The table below describes such a situation, where the condition of replica determinism **is not met**, and the faulty channel governs the decision.

Channel	Decision	Action
Channel 1	Take off	Accelerate engine
Channel 2	Abort	Stop engine
Channel 3	Abort	Accelerate engine

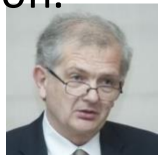
Because of **random effects** (deviation in the sensor calibration, digitalization error, etc.), channels 1 and 2 reach different conclusions:

Channel 1 decides that the **take-off speed has been reached** and that the plane should take off.

Channel 2 decides that the **take/off speed has not been reached**: **abort** the and the take-off.

Channel 3 is **faulty** and decides to abort, and to accelerate the engine.

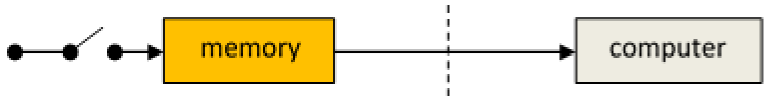
The **faulty channel wins!**



In the majority vote of the action, the faulty channel wins, because **the correct channels are not replica determinate!**

Sampling and polling:

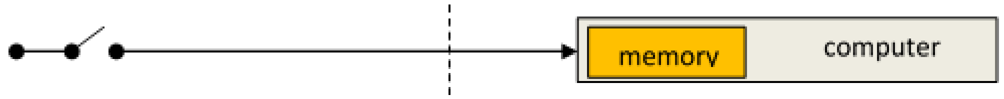
We use the term **sampling**, if the data is written into a memory element at the sensor:



From the point of view of system specification, a sampling system can be seen as protecting a node from more events in the environment than are stated in the system specification.

The memory is at the sensor and thus **outside the sphere of control** of the computer.

We use the term **polling,** if the data memory is resided inside the computer:



From functional point of view, there is no difference between sampling and polling as long as no faults occur. Under fault conditions, the sampling system is more robust than the polling system.

Comment: The **interrupt** mechanism can be characterized by the figure introduced for **polling**. The interrupt mechanisms empower **a device outside the sphere of control of the computer** to govern the temporal control pattern inside the computer. This is a potentially dangerous mechanism that must be used with great care.

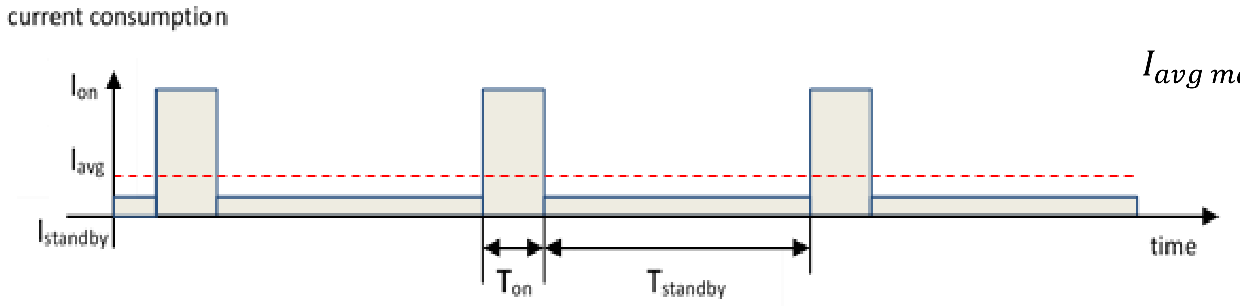
From the **fault-tolerance point of view**, an interrupt mechanism is even **less robust** than the already denounced polling mechanism. **Every transient error** on the transmission line **will interfere** with the temporal control scheme within the computer.

Power Aware Systems – Low Power Design

Example: The capacity/capability of two AA (Mignon) type battery cells in sensor network applications (microcontroller+radio+sensors):

2 pieces of AA battery cells have an average capacity of **3000 mAh.**

How long can we use our system in a day, if we require a total availability of services for minimum **1 year** (8760 hours), while **P_{on}=150 mW** (I_{on}=50mA) and **$I_{standby}$=50µA**?



$$I_{avg\ max} = \frac{3000mAh}{8760h} = 342\mu A$$

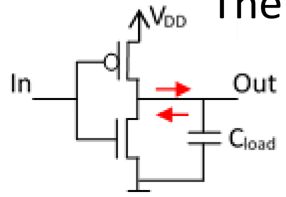
$$I_{avg} = \frac{I_{on}T_{on}}{T_{on}+T_{standby}} + \frac{I_{standby}T_{standby}}{T_{on}+T_{standby}} = I_{on}\lambda + I_{standby}(1-\lambda)$$

$$\boxed{\lambda = \frac{I_{avg\ max} - I_{standby}}{I_{on} - I_{standby}} = 0.0058 \approx 0.6\%}$$

$$\boxed{\approx 8\frac{minutes}{day}.}$$

If we take measurements **in every hour**, then they can take (only) **20 seconds**!

Power Consumption of a CMOS Gate:



The two FETs are alternately open and closed. Main sources of power consumption:

(1) charging and discharging capacitors,

(2) short circuit path between supply rails during switching,

(3) leaking diodes and transistors (becomes one of the major factors due to shrinking feature sizes in semiconductor technology).

Power consumption of CMOS circuits (ignoring leakage): $\boxed{P \sim \alpha C_L V_{DD}^2 f,}$

where V_{DD}: stands for **power supply**, α: **switching activity** (for a clock it is 1), C_L: a **load capacity**, f: **clock frequency**. Delay for CMOS circuits: $\boxed{\tau \sim C_L \frac{V_{DD}}{(V_{DD} - V_T)^2}}$ where V_T: threshold voltage, $V_T \ll V_{DD}$.

It can be stated: Decreasing V_{DD}

- reduces P **quadratically** (f constant);
- The gate delay increases only **reciprocally**;
- Maximal frequency f_{max} decreases **linearly.**

Potential for Energy Optimization (Dynamic Voltage Scaling: DVS): $\boxed{P \sim \alpha C_L V_{DD}^2 f\ ,}$

$\boxed{E \sim \alpha C_L V_{DD}^2 f t = \alpha C_{Lt} V_{DD}^2 (\#cycles).}$ Saving energy for a given task:

- reduce the supply voltage V_{DD};
- reduce switching activity;
- reduce the load capacitance;
- reduce the number of cycles (#cycles).

Use of Parallelism:

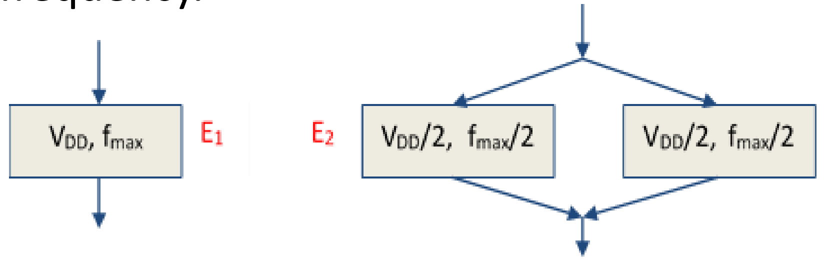
Duplicated hardware with half of the supply voltage, and half of the clock frequency.

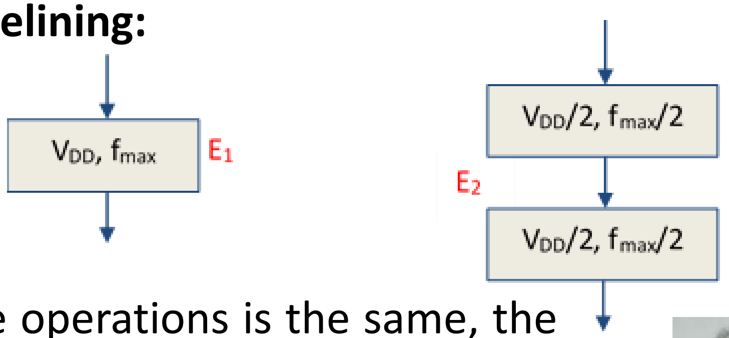
Power Supply Gating:

It is one of the most effective ways of minimizing static power consumption (leakage).

Power Supply Gating cuts-off power supply to inactive units/components: a header switch provides virtual power, while a footer switch provides virtual ground and thus reduces leakage.

Use of Pipelining:

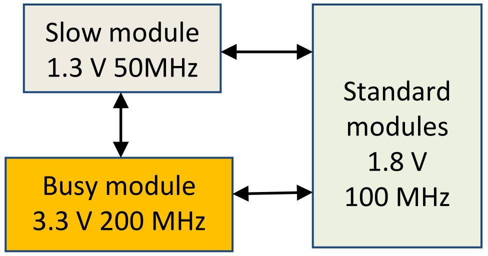


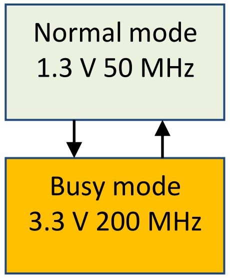


$E_1 \sim V_{DD}^2(\#cycles),$ $E_2 = \frac{E_1}{4}.$ The number of the operations is the same, the energy consumption is lowered to its fourth.

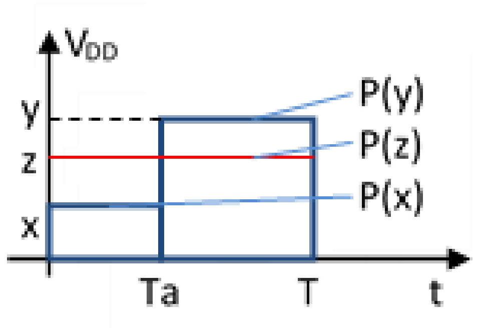
The application of **Very Long Instruction Word** (VLIW) architectures is an alternative: **parallel instruction sets** are applied.

Not all components require same performance. Required performance may change over time:





Optimal strategy (Dynamic Voltage Scaling):



Case A: execute at voltage *x* for *Ta* time units, and at voltage *y* for (1-*a*)*T* time units.

The energy consumption: $\underline{T(aP(x)+(1-a)P(y))}$.

Case B: execute at voltage $\underline{z = ax + (1-a)y}$ for *T* time units.

The energy consumption: $\underline{TP(z)}$.

Since the power is a convex (quadratic) function of V_{DD}, therefore $P(z) < aP(x) + (1-a)P(y)$, i.e. it worth executing at constant voltage. (The linear combination gives a string above the parabola.)

Example:

V_{DD} [V]	5.0	4.0	2.5
Energy per cycle [nJ]	40	25	10
f_{max} [MHz]	50	40	25
cycle time [ns]	20	25	40

Task execution needs 10^9 cycles within **25 seconds.**

a. Complete task **ASAP**: 10^9 cycles @ 50 MHz.

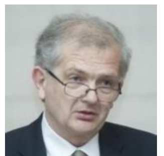
Energy consumption: $E_a = 10^9 * 40 * 10^{-9} = 40$ [J],

time requirement: $10^9 * 20 * 10^{-9} = 20s$.

b.Execution **at two voltages**: $0.75*10^9$ cycles @ 50 MHz + $0.25*10^9$ cycles @ 25 MHz.

Energy consumption: $E_a = 0.75 * 10^9 * 40 * 10^{-9} + 0.25 * 10^9 * 10 * 10^{-9} = 32.5$ [J],

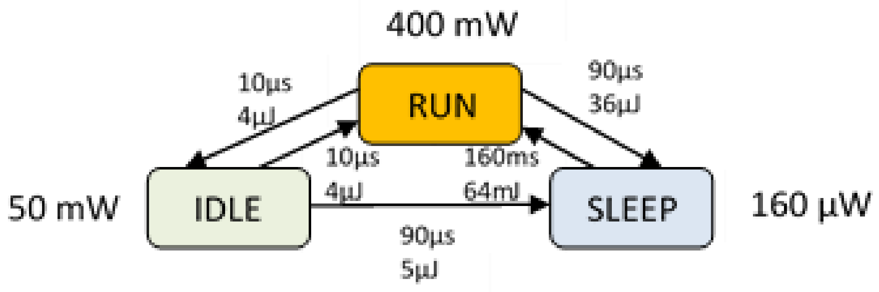
time requirement: $0.75 * 10^9 * 20 * 10^{-9} + 0.25 * 10^9 * 40 * 10^{-9} = \mathbf{25s}$.



c. Execution **at optimal voltage**: 10^9 cycles @ 40 MHz.
Energy consumption: $E_a = 10^9 * 25 * 10^{-9} = \mathbf{25\ [J]}$,
time requirement: $10^9 * 25 * 10^{-9} = \mathbf{25s}$.
Comment: Obviously some spare-time is always required at task executions.

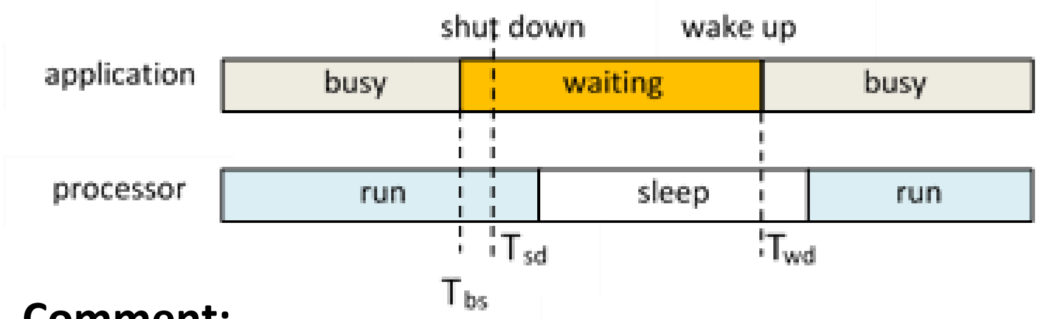
V_{DD} [V]	5.0	4.0	2.5
Energy per cycle [nJ]	40	25	10
f_{max} [MHz]	50	40	25
cycle time [ns]	20	25	40

Dynamic Power Management (DPM): tries to assign optimal power saving states



Requires hardware support.
Example: **StrongARM SA1100**
IDLE: a software routine may stop the CPU when not in use, while monitoring interrupts.
SLEEP: shuts down on-chip activities.

Example: reduce power according to workload:



T_{bs}: time before shutdown
T_{sd}: shutdown delay
T_{wd}: wakeup delay

Comment:
Shutdown only if long idle times occur.
There is a trade-off between savings and overhead “costs”.