Reliable Embedded Systems

Organizing our CanSat project towards success

(Organizing your other work at the same time)

Niels Malotaux

N R Malotaux
Consultancy

• +31-30-228 88 68 niels@malotaux.nl www.malotaux.nl

Niels Malotaux

my recent design

- **Project Coach**
  - Evolutionary Project Management (Evo)
  - Requirements Engineering
  - Reviews and Inspections
- **More than 30 years Embedded Systems Design experience**
- **Expert in helping projects to be successful**

Who are you ?

- Who are you ?
- What do you expect ?

Seminar Program 7 ~ 11 June 2010 - every morning 9:00 - 12:30

Monday: General introduction Reliable Embedded Systems

- What causes failure and what can we do about it

Tuesday: Requirements and Design

- What are we supposed to accomplish
- How to select the best way to do that
- How to document for better understanding

Wednesday: Planning

- How to make sure we'll be ready on time
- You will learn how to accomplish much more in less time

Thursday: Testing, Reviews and Inspections

- Learning to find our mistakes early and never make them again

Friday Master Class: actually organizing our project

- Using what we learnt to set up our project for success

Do you want your project to fail ?

- If you want to make your CanSat project a success, you should attend all sessions
- If you want your project to be a success, make sure you attend all five days !
- Those who attended all sessions may ask Niels for advice by email or Skype during the project
- E-mail: niels@malotaux.nl
- Skype name: nielsmx

This seminar should be highly interactive

- You don't learn much if only listening to a long presentation
- A lot of discussion will teach you more
- So prepare to be very interactive
- Think in advance about:
  - Questions about Embedded Systems Design
  - How much time you have available to do the project
  - All what you think you have to do for the CanSat project
  - Expected problems with CanSat
  - What else you have to do apart from the project
- You may email me a list of these things
  (then I can prepare better)

If you don’t have time

- If you don’t attend, you will use more time
- Attending the seminar will save you a lot of time
- Not only the CanSat project will be more successful:
  your other work will also be more successful in shorter time

- Therefore, if you don’t have time, you should attend !!!

Embedded System ?

- Information processing subsystem of embedding system
- Performing specific functions
- Not visible or directly accessible by the users of the embedding system
- Often switched on only once, and then running for years

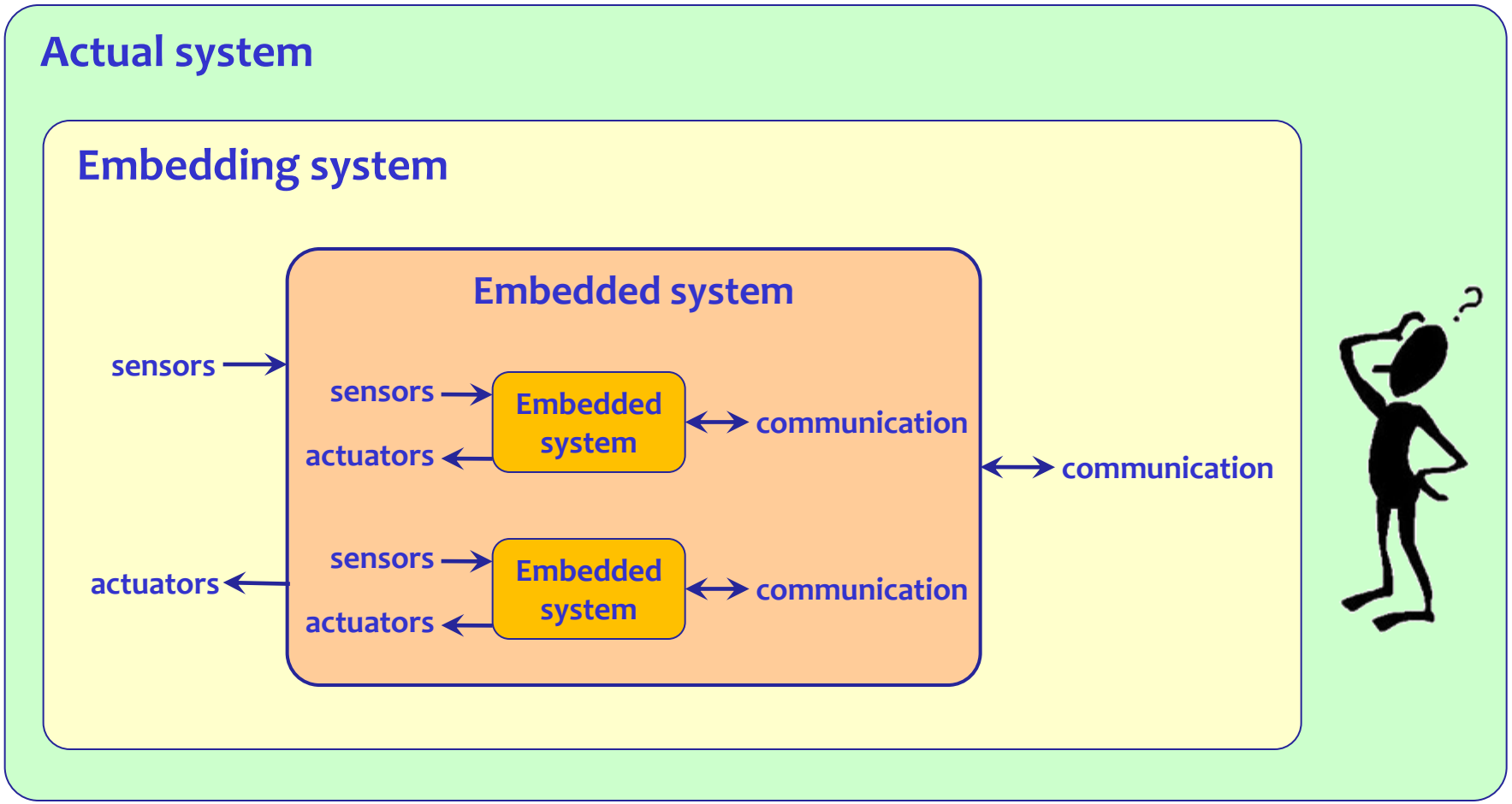

Actel
FUSION
Actel


PHILIPS
5005

How about the *Embedding* System ?

- **Should we talk about Embedded Systems ?**
- **If we don't consider the whole System we will provide inadequate interfaces**

Actual Systems



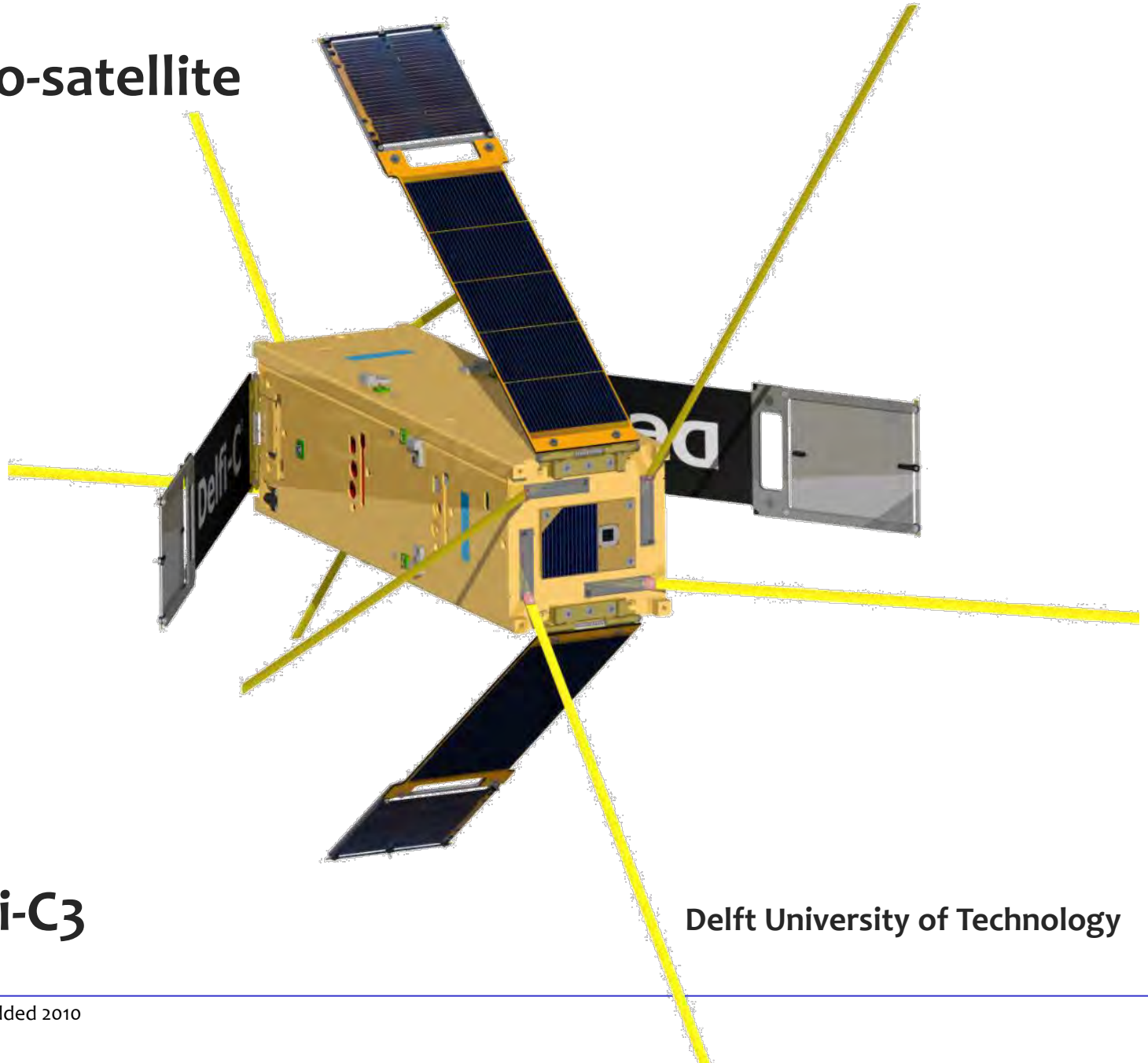
ATM (Automatic Teller Machine)

A lot of Embedded Systems

Washing machine

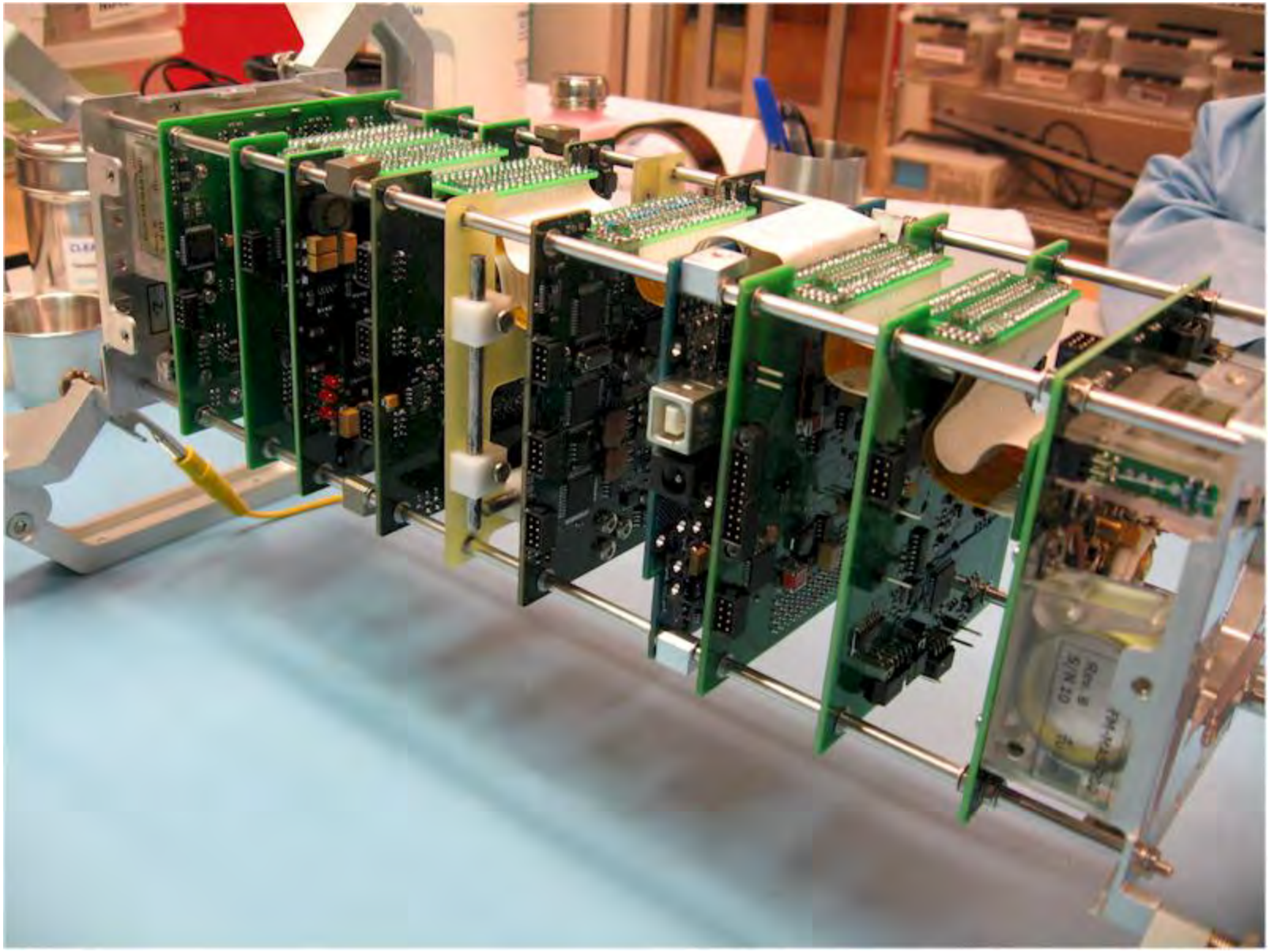
MP3 player

Nano-satellite

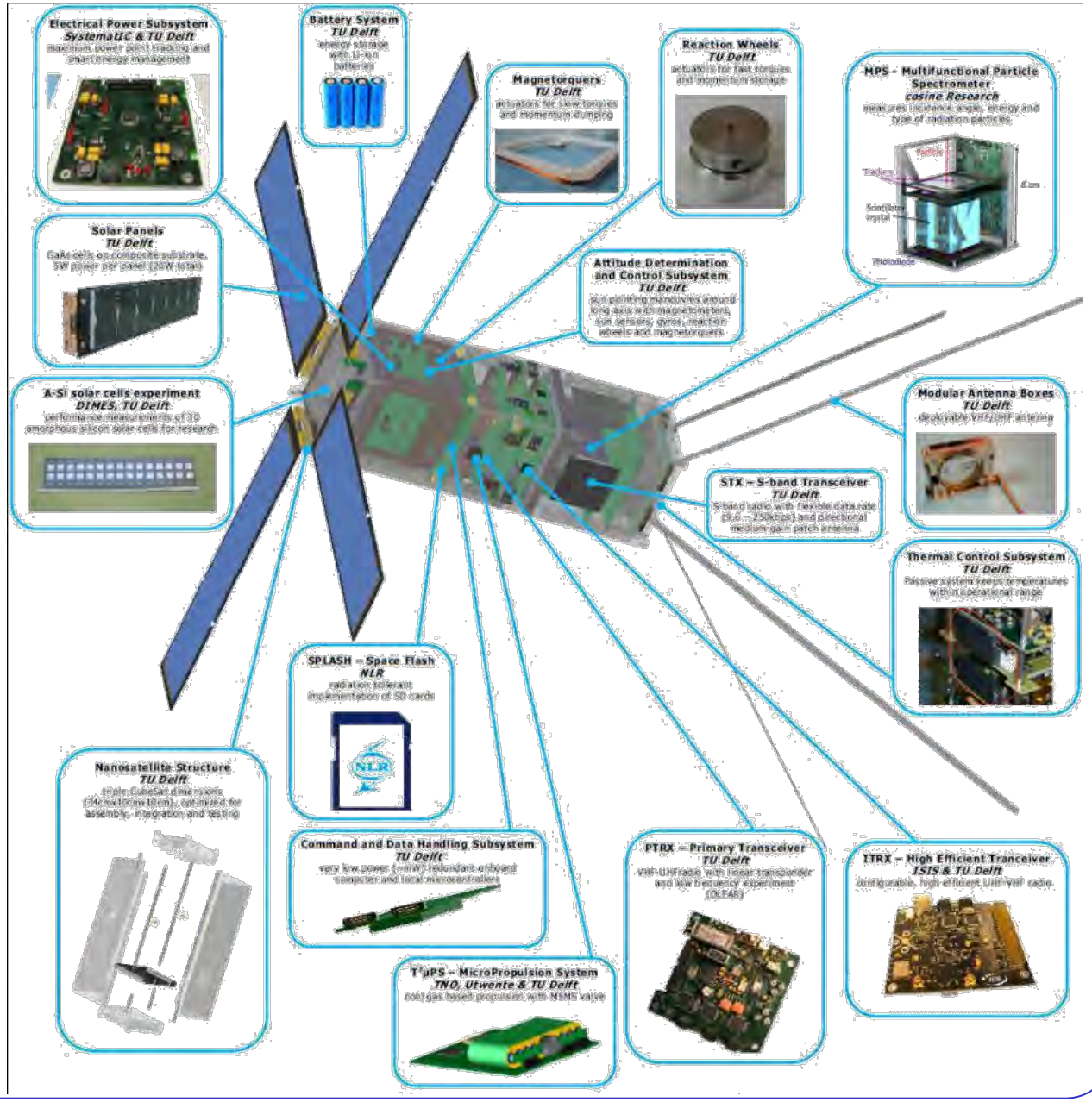


Delfi-C3

Delft University of Technology



Delfi N3xt


Electrical Power Subsystem
SystematIC & TU Delft
maximum power point tracking and smart energy management
Battery System
TU Delft
energy storage with Li-ion batteries
Magnetorquers
TU Delft
actuators for slow torques and momentum dumping
Reaction Wheels
TU Delft
actuators for fast torques and momentum storage
MPS – Multifunctional Particle Spectrometer
cosine Research
measures incidence angle, energy and type of radiation particles
Solar Panels
TU Delft
GaAs cells on composite substrate, 5W power per panel (20W total)
Attitude Determination and Control Subsystem
TU Delft
fine pointing manouvres around 3 axis with magnetometers, sun sensors, gyros, reaction wheels and magnetorquers
A-Si solar cells experiment
DIMES, TU Delft
performance measurements of 10 amorphous silicon solar cells for research
Modular Antenna Boxes
TU Delft
deployable VHF/UHF antenna
STX – S-band Transceiver
TU Delft
S-band radio with flexible data rate (9.6 – 256kbps) and directional medium-gain patch antenna
Thermal Control Subsystem
TU Delft
Passive system keeps temperatures within operational range
SPLASH – Space Flash
NLR
radiation tolerant implementation of SD cards
NLR
Nanosatellite Structure
TU Delft
triple-CubeSat dimensions (34cmx10cmx10cm), optimized for assembly, integration and testing
Command and Data Handling Subsystem
TU Delft
very low power (<mW) redundant onboard computer and local microcontrollers
PTRX – Primary Transceiver
TU Delft
VHF-UHF radio with linear transponder and low frequency experiment (DLFAR)
ITRX – High Efficient Tranceiver
ISIS & TU Delft
configurable, high efficient UHF/VHF radio
T³μPS – MicroPropulsion System
TNO, Utwente & TU Delft
cool gas based propulsion with MEMS valve

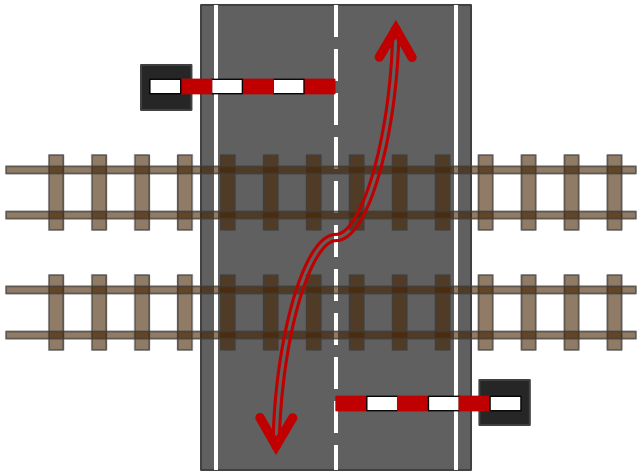
Reliable Embedded Systems ?

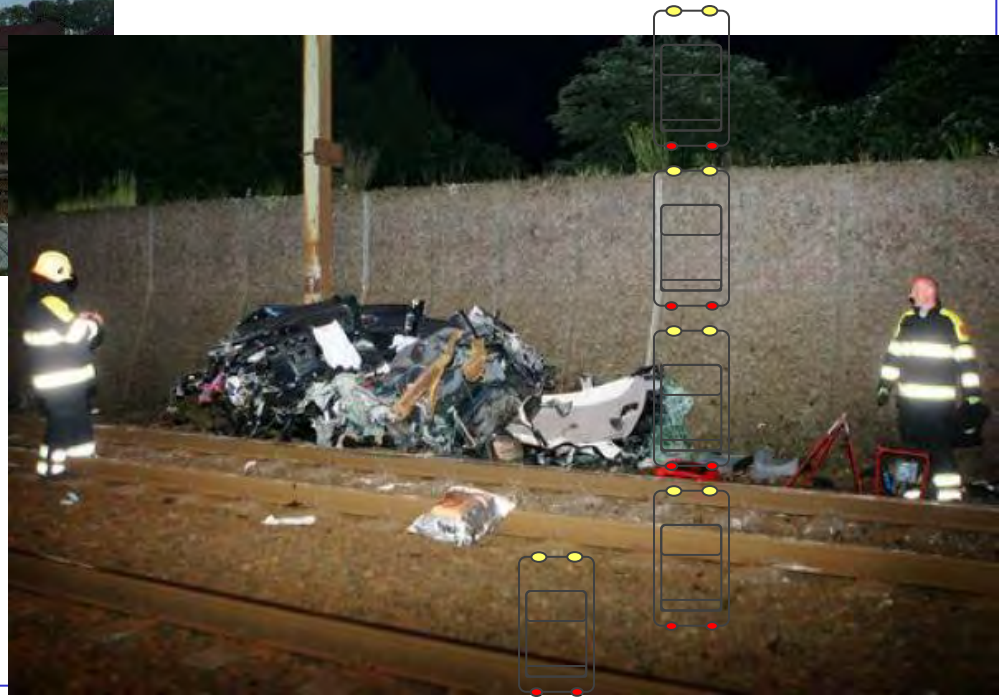
- **It should simply work**
- **How reliable ?**
- **Reliability is an element of Dependability**

Dependability

- Some embedding systems can be switched off
- Many embedded systems we cannot switch off
- We have to trust the correct operation

AHOB (Automatic Half Barrier Crossing)





ADOB

(Automatic Double Barrier Crossing)

1 train every 4 minutes

Few years of trouble before some stability

At >22°C still trouble

Why it didn't work is irrelevant

What we deliver should simply work

Is that so difficult?

Dependability is a complex concept

- **Availability**
  - Readiness for correct service
- **Reliability**
  - Continuity of correct service
- **Safety**
  - Not causing damage
- **Integrity**
  - No improper system alterations
    - Internal
    - External → Security
- **Maintainability**
  - Ease of required alterations

Availability

- **Dependability.Availability**
  - Readiness for correct service
  - Scale: % of <TimePeriod> a <System> is <Available> for its <Tasks>
- **Probability that the system will be functioning correctly when it is needed**
- **Examples**
  - (preventive) maintenance may decrease the availability
  - Telephone exchange (no dial tone) < 5 min per year (99.999%)
  - Snow on the road

Availability

Availability %	Downtime per year	Downtime per month	Downtime per week	Typical usage
90%	36.5 day	72 hr	16.8 hr	
95%	18.25 day	36 hr	8.4 hr	
98%	7.30 day	14.4 hr	3.36 hr	
99%	3.65 day	7.20 hr	1.68 hr	
99.5%	1.83 day	3.60 hr	50.4 min	
99.8%	17.52 hr	86.23 min	20.16 min	
99.9% (three nines)	8.76 hr	43.2 min	10.1 min	Web server
99.95%	4.38 hr	21.56 min	5.04 min	
99.99% (four nines)	52.6 min	4.32 min	1.01 min	Web shop
99.999% (five nines)	5.26 min	25.9 sec	6.05 sec	Phone network
99.9999% (six nines)	31.5 sec	2.59 sec	0.605 sec	Future network

Reliability

- **Reliability**
  - Continuity of correct service
  - Keeps working as intended
  - Scale: Mean time for a <System> to experience <Failure Type> under <Conditions>
- **MTBF – Mean Time Before Failure**
- **MTBR – Mean Time Between Repairs**
- **MTTR – Main Time To Repair**
- **Reliability does not automatically imply safety**

Safety

Safety

- Not causing death, injury, illness, damage to people, equipment, environment
- Probability that x people die per year
- Example: star-system for cars (adult / child, in-car / pedestrian)

- System staying in safe state despite failures
- Safety does not automatically imply reliability
- A safe system may stop functioning, as long as damage is avoided
  - Car ?
  - Airplane ?

Failures

- **Low frequency / low demand rate**
  - Anti-lock braking (ABS)
  - Air bags

Availability

➢ PFD - Probability the Function fails on Demand

- **Frequent or continuous use**
  - Normal braking
  - Steering

Reliability

➢ MTTF - Mean Time To Failure

Safety

➢ After-the-Event Protection

SIL - Safety Integrity Level

Infrequent use, low demand		
SIL Safety Integrity Level	**PFD Probability Fail on Demand**	**Availability**
4	$< 10^{-5}$	99.999%
3	$< 10^{-4}$	99.99%
2	$< 10^{-3}$	99.9%
1	$< 10^{-2}$	99%
Frequent or continuous use		
SIL Safety Integrity Level	**MTTF (years) Mean Time To Failure**	**Failures per hour**
4	> 10,000 yr	$< 10^{-8}$
3	> 1,000 yr	$< 10^{-7}$
2	> 100 yr	$< 10^{-6}$
1	> 10 yr	$< 10^{-5}$

Fault tolerance

- **Fault tolerance**
  - Ability (how much) to function reliably also if faults occur
- **Full Fault tolerance**
  - Absolute Functioning reliably also if faults occur
  - Delivering all services whether errors occur or not
  - Primary mechanism is replication of software, hardware, information, and preventive maintenance
- **Graceful degradation**
  - Continuing delivering services even when errors occur, discarding less important services
- **Fail-safe system**
  - Aborting operation in a safe state

Redundancy

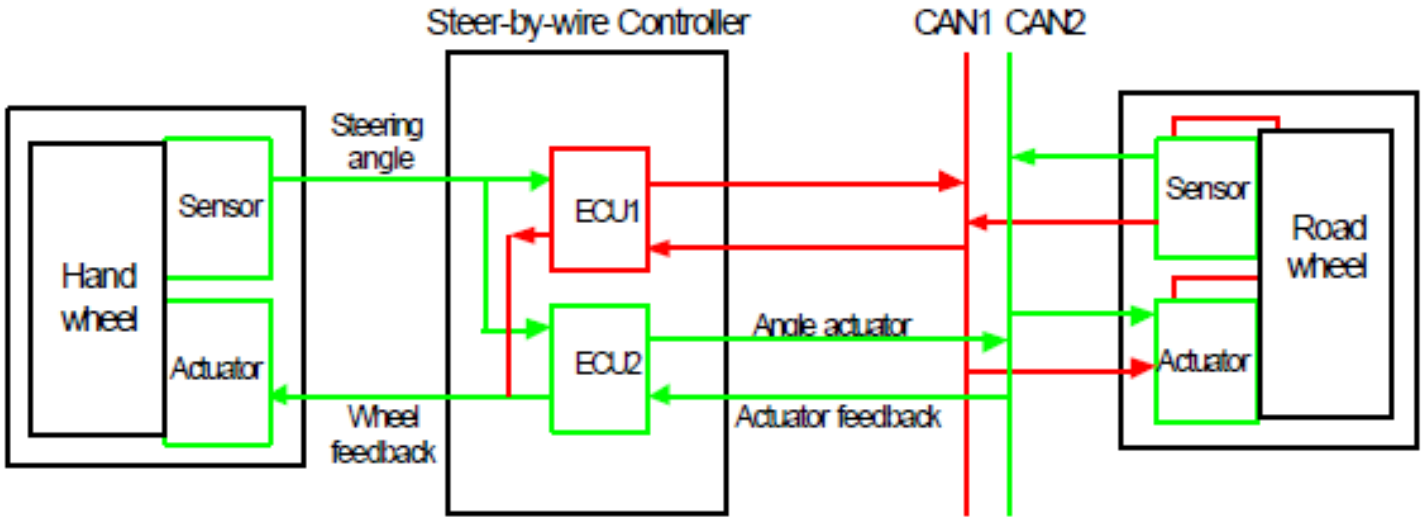


Figure 1. Hardware architecture of a steer-by-wire system.

Security

Dependability.Security

Free from intrusions (theft, alteration)

Scale: Time required to <break into the system>

Can our competition jam our CanSat communication ?

Availability risks of development ?

- **Delivering the right thing**
- **At the right time**

Reliability Engineering

ref Albertijn Barnard

- **Reliability**
  - No failures in products and systems
- **Reliability Engineering**
  - Preventing the creation of failure
  - Engineering analysis
  - Failure analysis
  - Stress test

Reliability engineering actions

ref Albertijn Barnard

- Concurrent engineering
- Integrated project teams
- Design reviews
- Mechanical and electrical stress predictions
- Component derating analysis
- Electrolytic capacitor life calculations
- FMEA and FTA
- System modeling
- Highly-Accelerated Life Testing (HALT) and Highly-Accelerated Stress Screening (HASS)
- Field return root cause analysis

Failure Mode and Effect Analysis

- System - focuses on global system functions
- Design - focuses on components and subsystems
- Process - focuses on manufacturing and assembly process
- Service - focuses on service functions
- Software - focuses on software functions

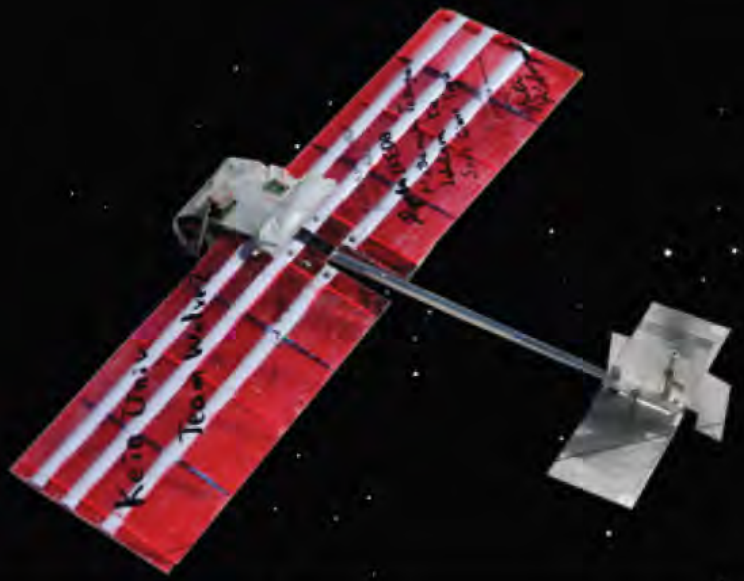
CanSat 2009 project

Our Missions

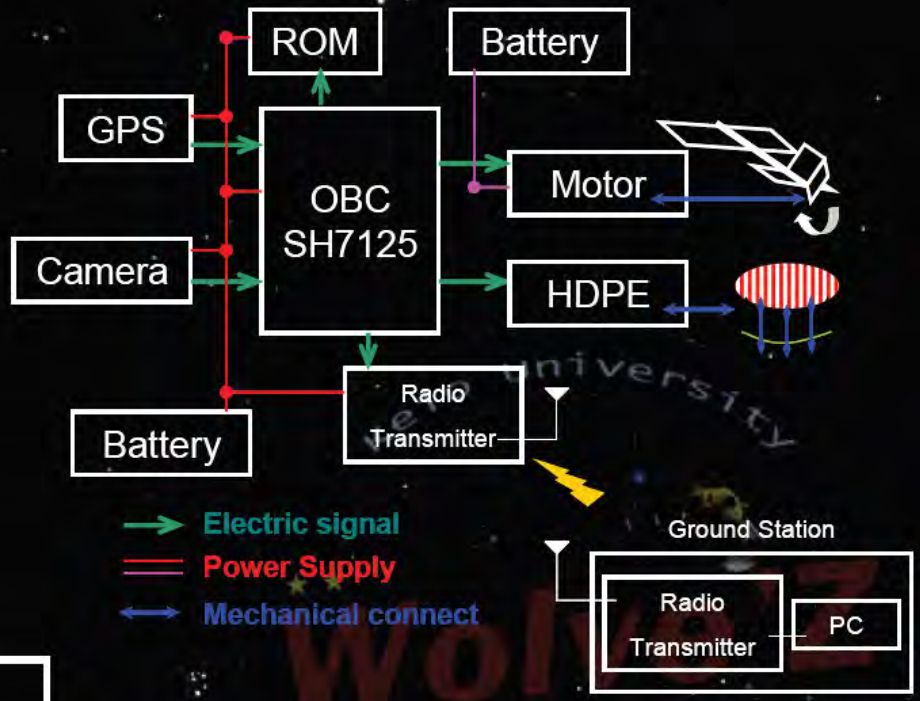
- Fly Back Mission
  - aiming at the target point by autonomous flight.
- Camera Mission
  - taking a moving picture in the sky.
- Soft-landing Mission
  - Decrease speed by the parachute when landing.

Wolve'Z CanSat

WOLVE'Z 09



Weight	600g
Folding Size	100mm × 130mm × 230mm
Opened Size	length530mm × width700mm
Wing Span	700 × 180＝126000mm²



System Diagram

The result of this year's ARLISS flights

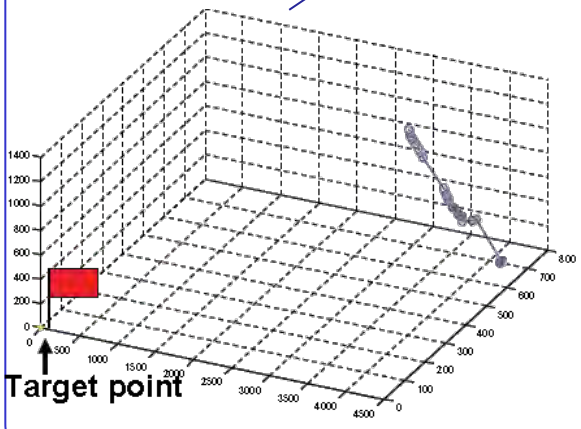
- We launched airplane style CanSat
- First flight's record is 4187 m and second is 301 m
- In third flight, we got a moving picture in the sky
- We couldn't get control record from radio transmitter

Future plans

- We must develop a new radio transmission system so as to get the downlink data from CanSat more certainly

What was the main result ?

	1st flight	2nd flight	3rd flight
Fly back	with control !!	with control !!??	free fall
Camera	fail	fail	success
Soft Landing	fail	fail	fail



Why so much failure ?

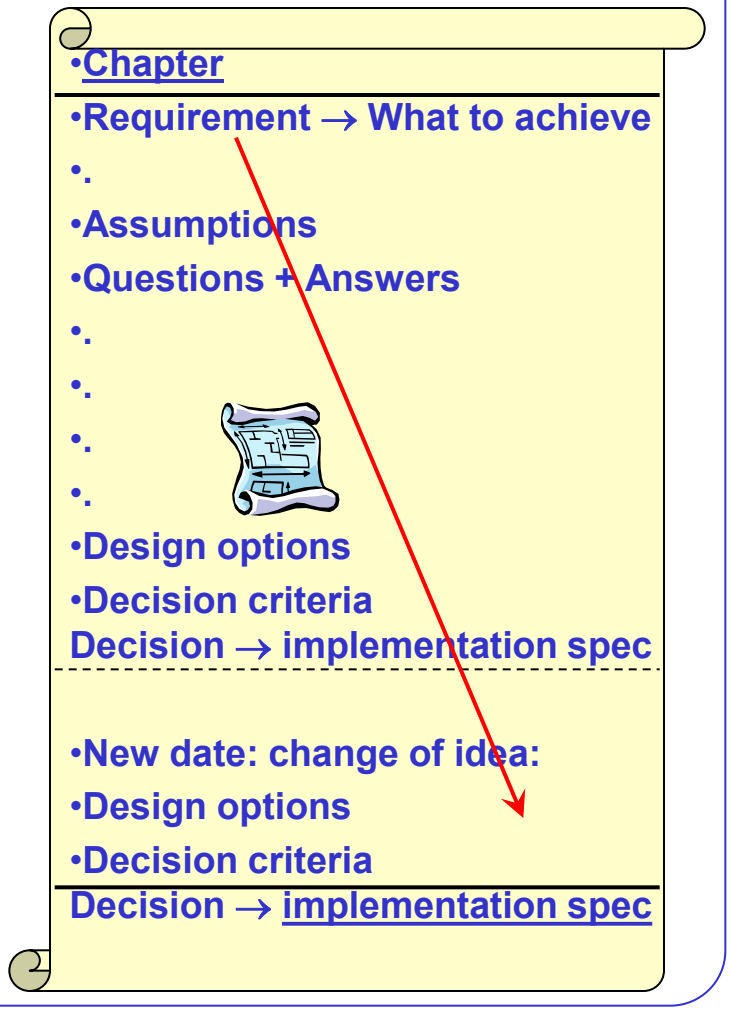
- Not only the Keio team
- Trial and error ?
- Hope ?
- ??

CanSat 2010: Failure is not an option !

- How to make sure it will *simply work*
- Calculate how to achieve our goals *by design* (not hope)
- Learning lessons from previous attempts
- Do root cause analysis on previous attempts
- What went wrong, why and how will we prevent failure ?
- Did the system keep a log of activities (for analysis) ?
- Did we do sufficient fundamental experiments ?
- Did the designers keep a Design Log ?
- Did the designers systematically plan their project ?

DesignLog

- In computer, not loose notes, not in e-mails, not handwritten
  - Text
  - Drawings!
  - On subject order
  - Initially free-format
  - For all to see
- All concepts contemplated
  - Requirement
  - Assumptions
  - Questions
  - Available techniques
  - Calculations
  - Choices + reasoning:
    - If rejected: why?
    - If chosen: why?
- Rejected choices
- Final (current) choices
- Implementation



Risk

An uncertain event or condition that,

if it occurs,

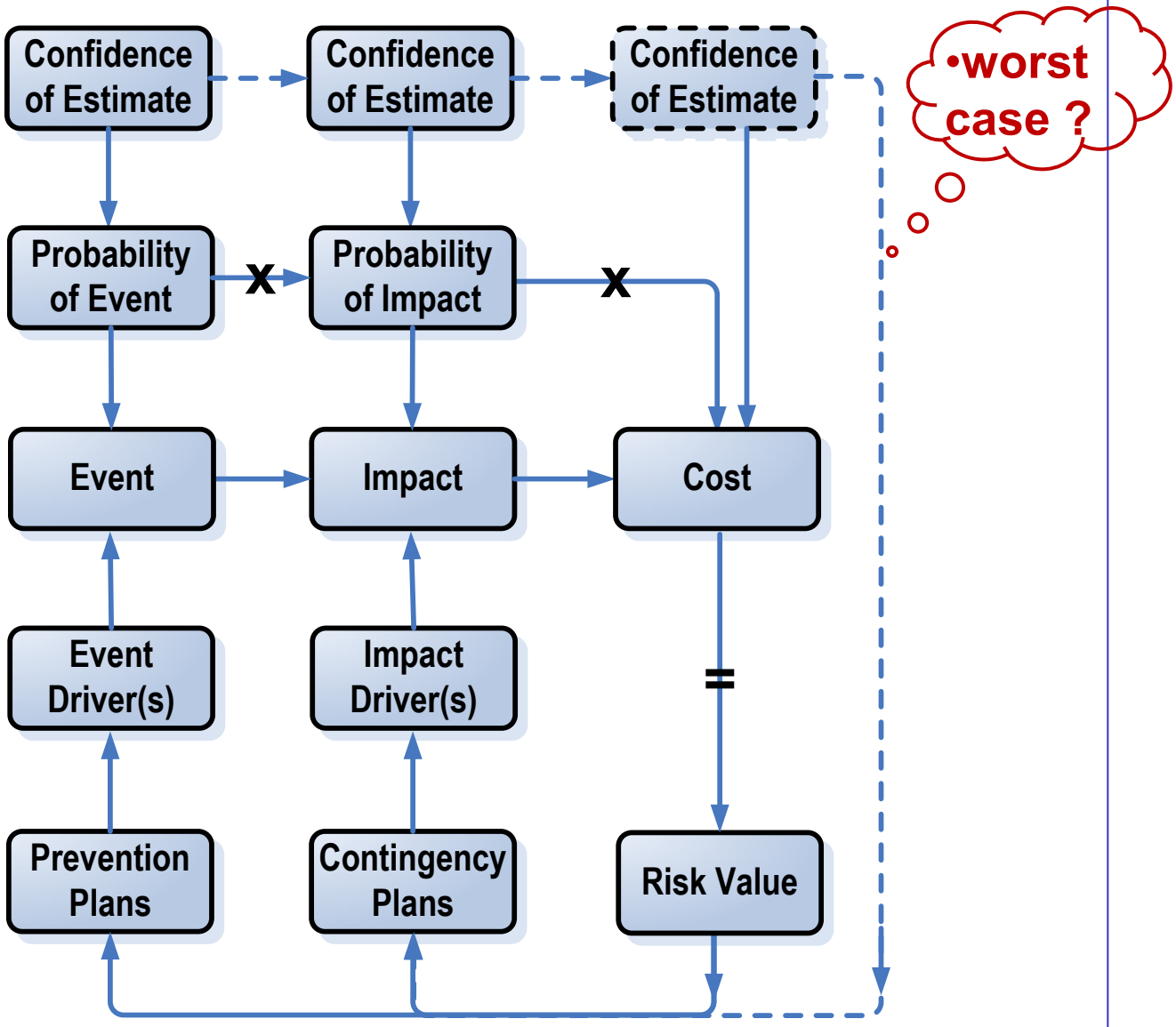
has a negative effect

on a project's objectives

(PMBOK)

- **0% probability is not a risk**
- **100% probability is an issue or a problem**

Risk Model



$$V_R = P_e * P_i * C$$

Checklists for brainstorm

- Human risk
  - In the project
  - After the project
- Technical risk
  - Can we make it
  - Will it survive
- Environmental risk
  - Example: CE
- Regulatory risk
  - Example: CE
- Consequential risk
- …

Each of these can have
it's own checklist
to trigger the recognition
of real risks

What are Risks in our Project ?

…

…

…

Are these really Risks?

0% probability is not a Risk

100% probability is not a Risk

Many known risks are hardly risks

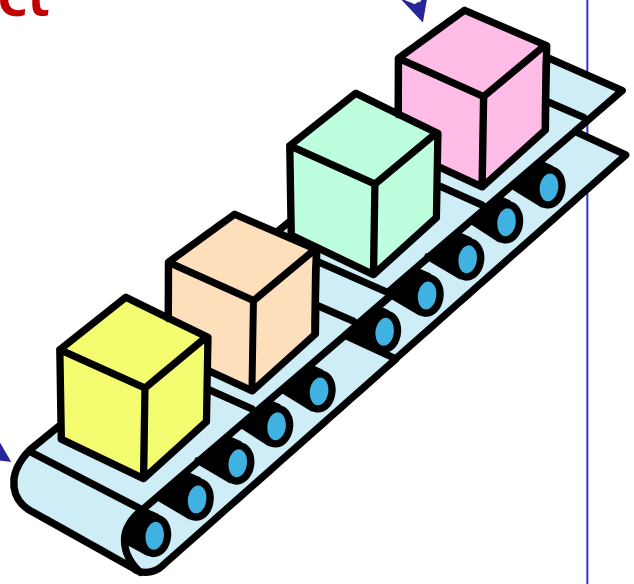
Most of the *real* risks are in the product

Most of the *known* risks are in the project

$V_{Risk} = P_{event} * P_{impact} * C$

$P_{event} = 1$

$P_{impact} \rightarrow 0$



We don't only *design* the product,

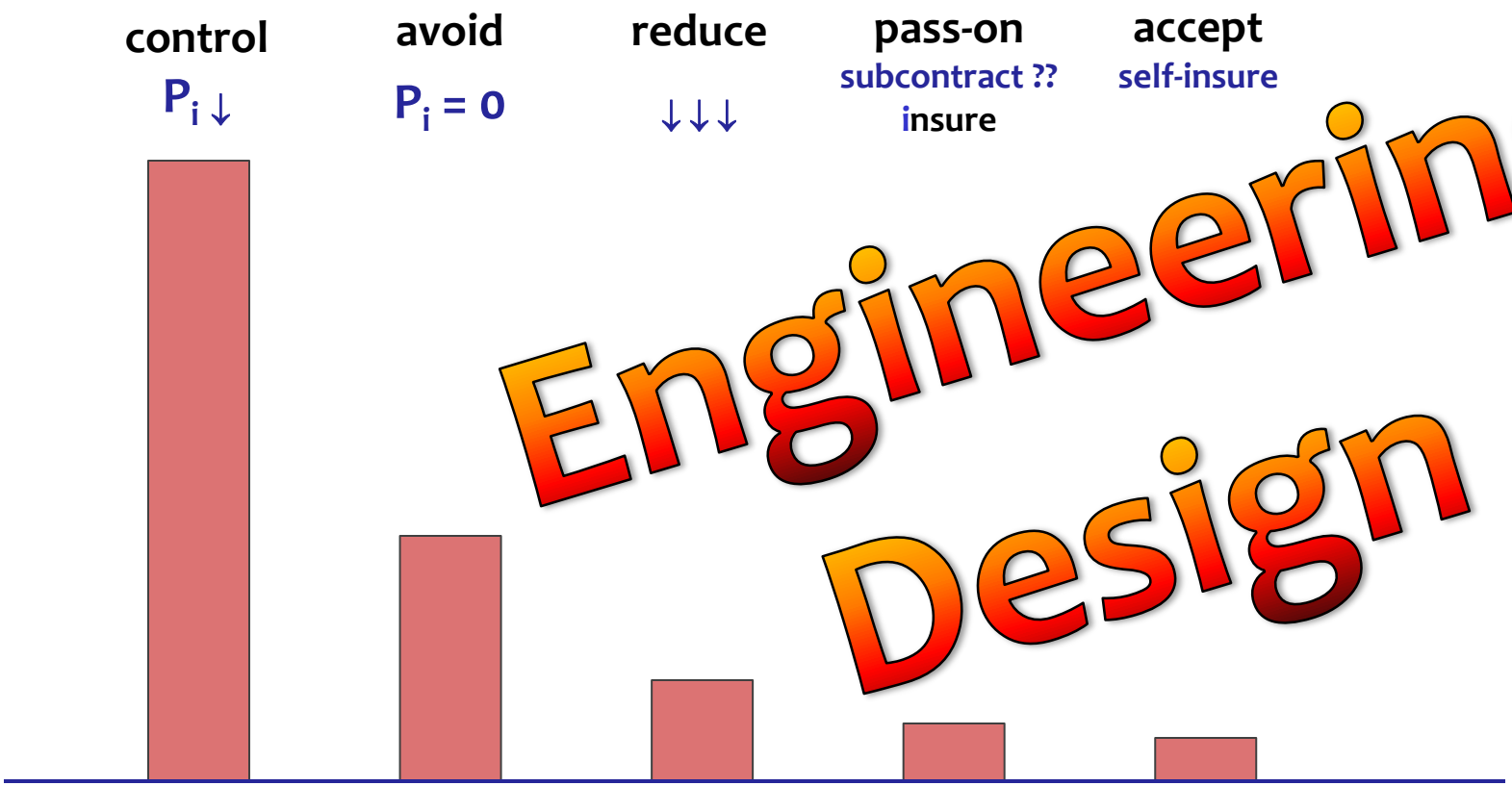
We also *design* the project

If we control 80% of the risks *by design*

We have more time to handle the 20% real risks

Risk mitigation

$$V_{Risk} = P_{event} * P_{impact} * Cost$$



Murphy's Law

- Whatever can go wrong, will go wrong

- Should we accept fate ??

Murphy's Law for Professionals:

Whatever can go wrong, will go wrong …

Therefore:

We should actively check all possibilities that can go wrong and *make sure that they cannot happen*

Exercise

- **Which risks can we expect in our CanSat project ?**
- **What are we going to do about it ?**
- **Save in DesignLog**

Issues for reliability ?

- Radiation
- Temperature, temperature cycles, temperature in vacuum
- Vibration
- Pressure: vacuum, up to 1 atm
- Mass - Weight - Size
- No single point of failure
- Worst case
- Power supply, power consumption
- Components

Worst reliability risks ?

- Mechanics ?
  - Design errors or weaknesses
  - Wear
  - Unexpected behavior
- Electronics ?
  - Design errors or weaknesses
  - Unexpected behavior
- Software ?
  - Design errors or weaknesses
  - Unexpected behavior

Embedded software

- **Embedded**
  - Invisible computing power for specific purpose
- **Concurrent**
  - Several (interacting) processes run at the same time
- **Reactive**
  - Reacts on external signals
- **Real-time**
  - Reactions are appropriately immediate
- **Complex**
  - More complicated than we can oversee

Complexity

- How difficult to understand and verify the design or implementation
- Complexity can be reduced by
  - Methods/thoughts/tools/tables/diagrams/abstractions if they increase our understanding and/or ability to verify
  - First developing the problem
    (understanding the problem is half the solution)
  - Starting small and simple
  - Keeping it small and simple

Growing size of Embedded Software

- Size of software in a TV-set grows exponentially
- I'm hardly impressed
- Using libraries of unknown quality

Start small, keep it small

- ‘Trying’ large program doesn’t work
- Do small steps
  - Skeleton
  - Thin tread
  - Small tree
- Keep it as small and simple as possible

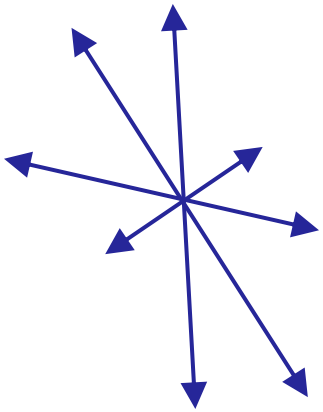
Looking at the whole

- Designers often focus on their sub-system
- Risk of sub-optimizing in stead of optimizing the whole system

Systems Engineering

- **Other Engineers (?)**
  - Silo thinking
  - Sub-optimizing
  - Gold plating (hobbies)
  - Little attention to interfaces
  - Projects are always *multidisciplinary*

- **Systems Engineers**
  - Multi-dimensional thinking
  - Optimizing design decisions over all dimensions
  - Whole life-cycle (cradle to cradle)
  - Balancing requirements
  - Including delivery time
  - All disciplines → *interdisciplinary*



Multidisciplinary ↔ Interdisciplinary

- **Tension between**
  - Technologically possible
  - Economically profitable
  - Socially and psychologically acceptable
  - All kinds of disciplines needed for a good solution
- **Multidisciplinary**
  - Many disciplines work in the project
  - Optimize solution in their own domain
- **Interdisciplinary**
  - Many disciplines work *together* in the project
  - Overall-optimizing
  - First *developing the problem*, before developing the solution, before implementing the solution

Types of Systems Engineering

ref Joe Kasser

Type V

- Can define the problem and then determine what needs to be done to implement an optimal solution

Type IV

- Can define the problem

Type III

- Can be given the problem and then determine what needs to be done to implement an optimal solution

Type II

- Can be told what needs to be done to implement a solution and can work out how to do it

Type I (apprentices)

- Can be told how to implement a solution and can then do it

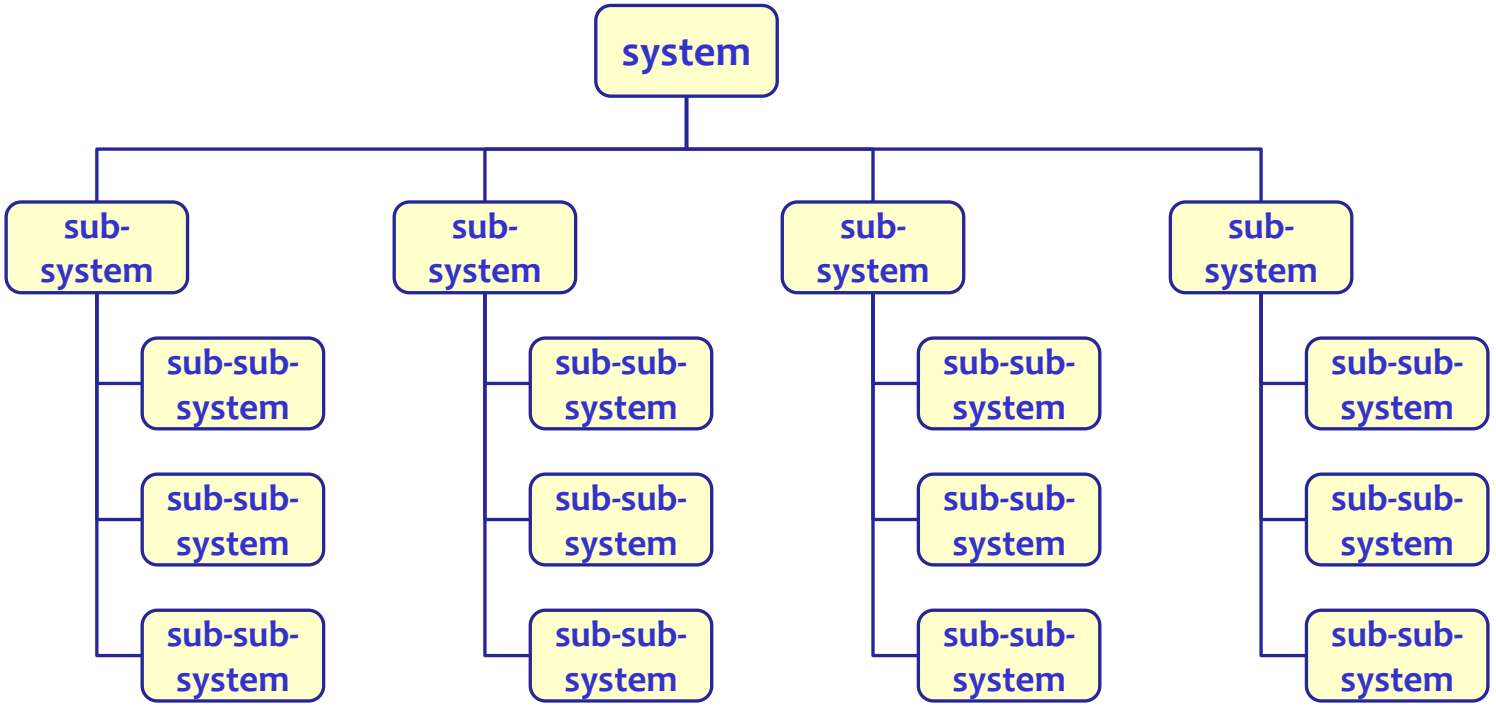
We need sufficient understanding

- Electronics
- Software
- Mechanics
- Aerodynamics
- Interfacing
- Acknowledge that there is a lot we don't know (yet)
- How to find out what we don't know
- Where to find and how to learn what we should know
- We must learn to be Type V Systems Engineers

Reducing complexity

- **Systematically understanding all parts of the system**
  - Relationship drawings
  - Schematic drawings
  - Geometric drawings
  - DesignLog
  - Software architecture
  - Flow diagrams
  - State diagrams
  - Brainstorm
  - Discussion
  - Review

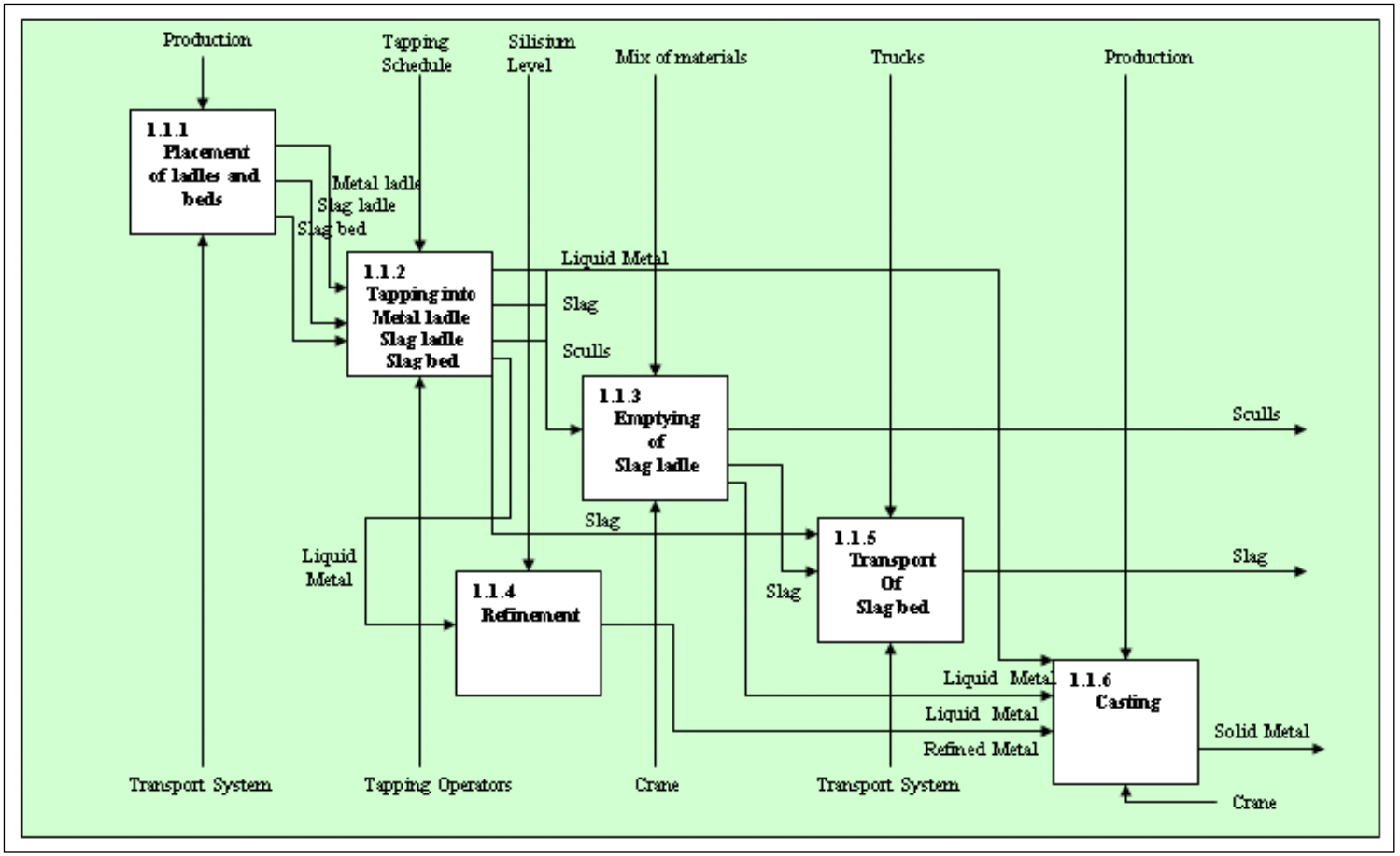
Understanding the relationships

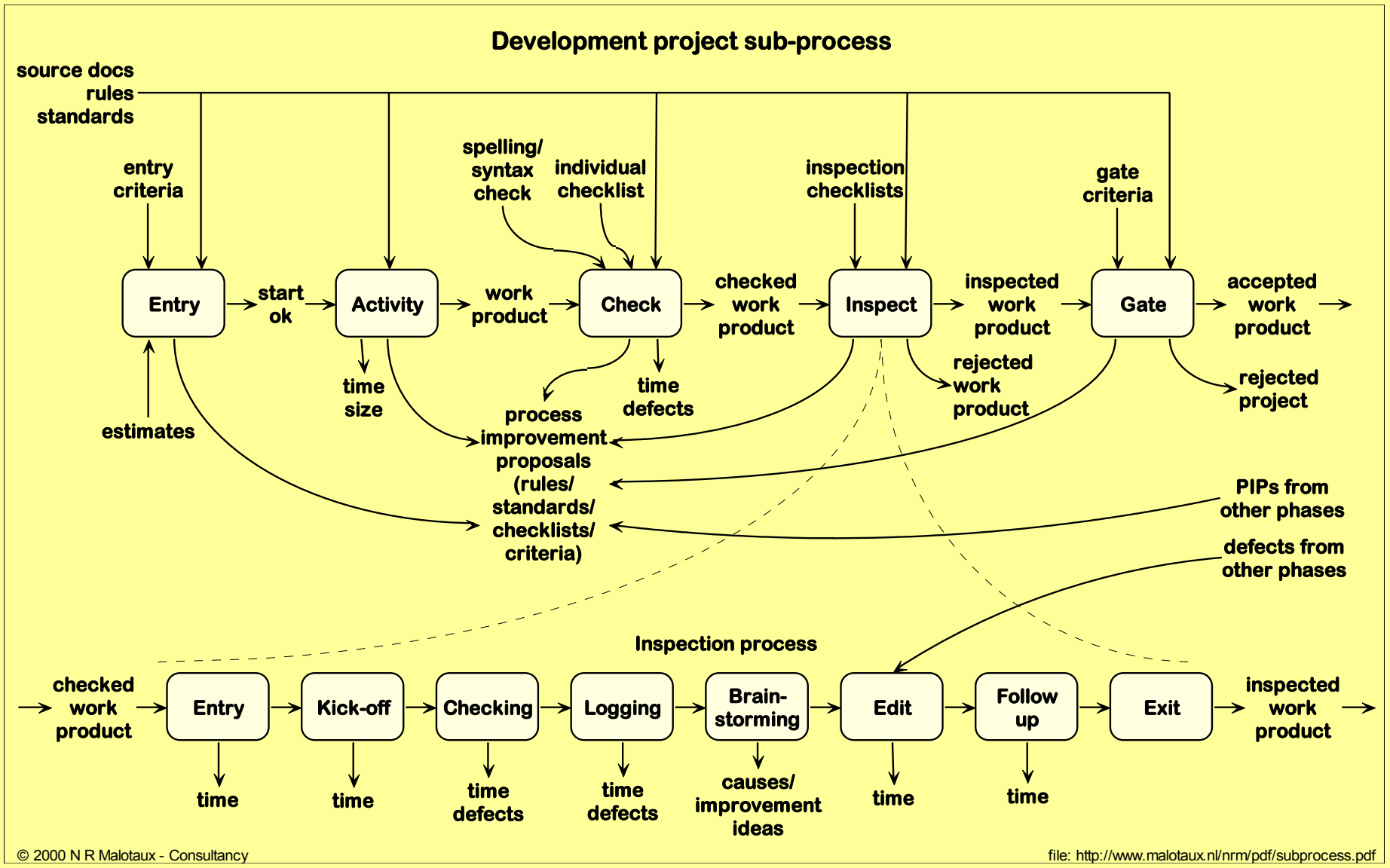


Drawings and diagrams

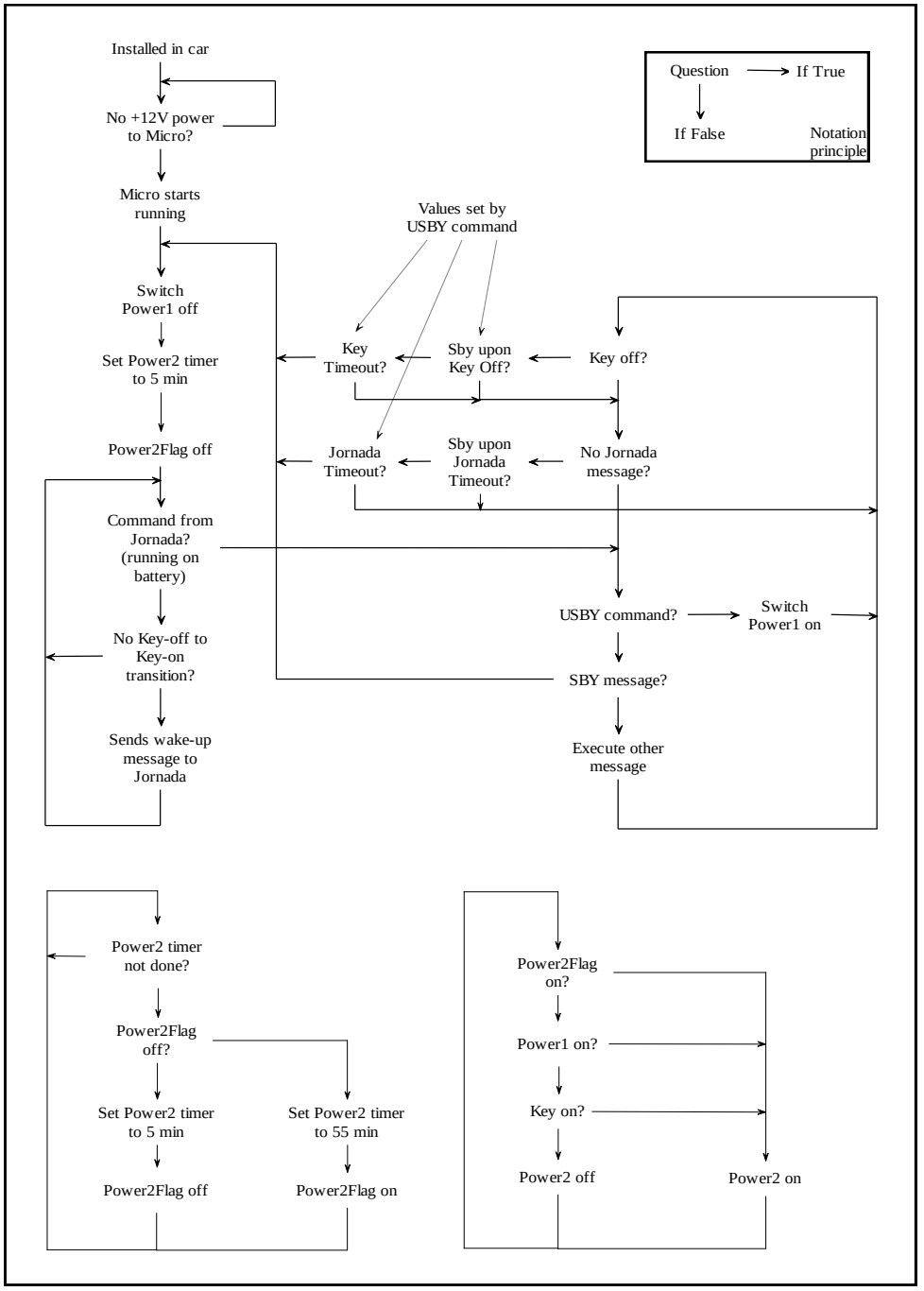
- **Diagrams must be ‘immediately’ clear**
- **If not, they are a risk**

Is this clear ?

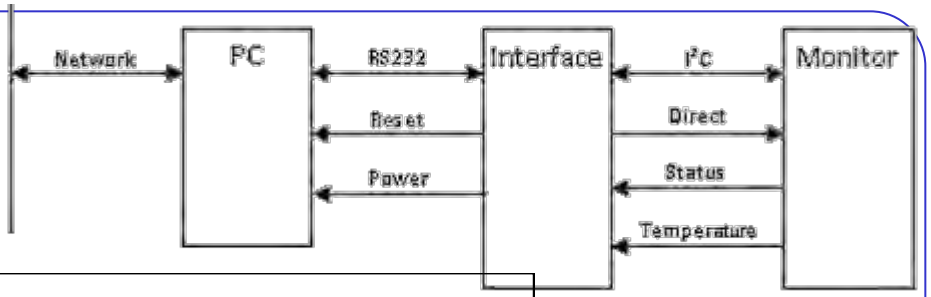


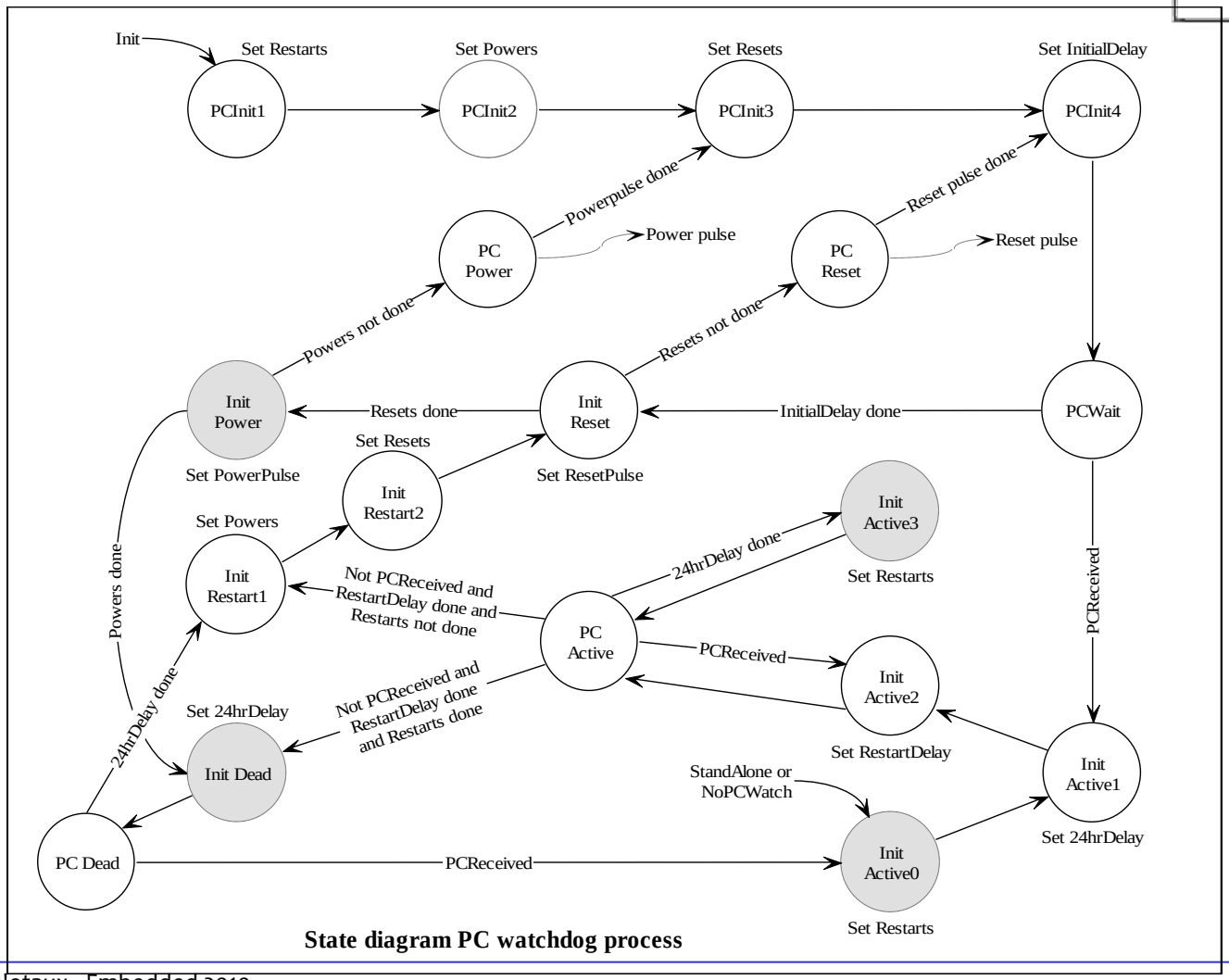

Development project sub-process
source docs
rules
standards
entry
criteria
spelling/
syntax
check
individual
checklist
inspection
checklists
gate
criteria
Entry
start
ok
Activity
work
product
Check
checked
work
product
Inspect
inspected
work
product
Gate
accepted
work
product
time
size
time
defects
rejected
work
product
rejected
project
estimates
process
improvement
proposals
(rules/
standards/
checklists/
criteria)
PIPs from
other phases
defects from
other phases
Inspection process
checked
work
product
Entry
Kick-off
Checking
Logging
Brain-
storming
Edit
Follow
up
Exit
inspected
work
product
time
time
time
defects
time
defects
causes/
improvement
ideas
time
time
© 2000 N R Malotaux - Consultancy
file: http://www.malotaux.nl/nrm/pdf/subprocess.pdf

Flowgrams


Installed in car
No +12V power to Micro?
Micro starts running
Switch Power1 off
Set Power2 timer to 5 min
Power2Flag off
Command from Jornada? (running on battery)
No Key-off to Key-on transition?
Sends wake-up message to Jornada
Values set by USBY command
Key Timeout?
Sby upon Key Off?
Key off?
Jornada Timeout?
Sby upon Jornada Timeout?
No Jornada message?
USBY command?
Switch Power1 on
SBY message?
Execute other message
Question
If True
If False
Notation principle
Power2 timer not done?
Power2Flag off?
Set Power2 timer to 5 min
Set Power2 timer to 55 min
Power2Flag off
Power2Flag on
Power2Flag on?
Power1 on?
Key on?
Power2 off
Power2 on

State Machines





State diagram PC watchdog process

Where are the risks ?

- Everywhere !
- Hardware
  - Timing
  - Voltages
  - Currents
  - Worst case
- Software
  - Architecture
  - Design
  - Operating System
  - Asynchronous events

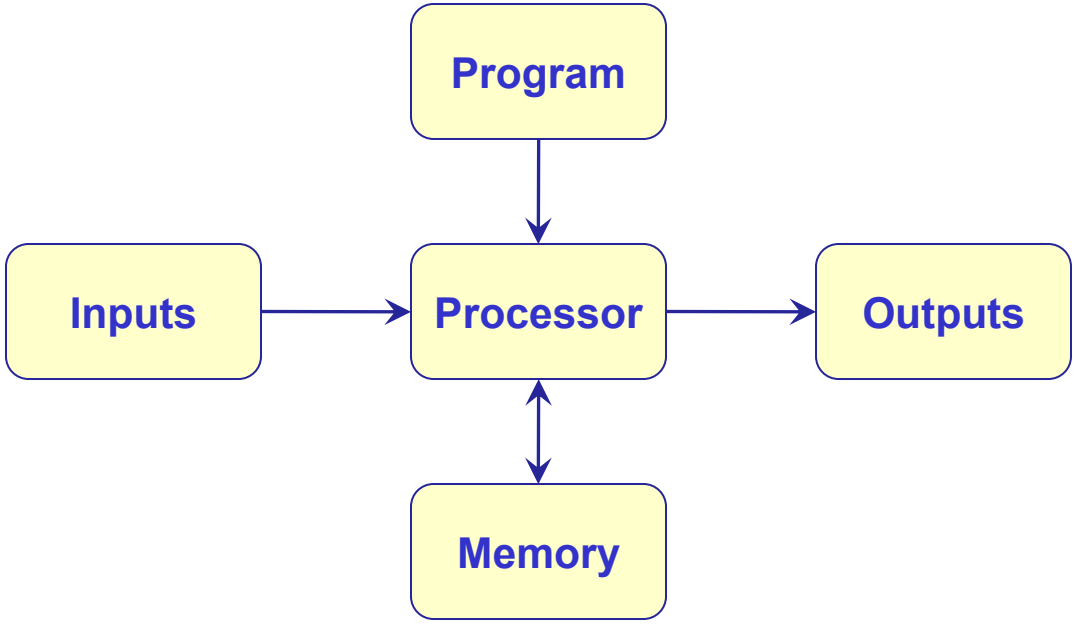
Sunny Scenario ?

- 20% of the software is there to make the computer do what it should do
- 80% of the software is there to make the computer not do what it should not do
- Did we define the 80% part properly ?

Never more than one place

- Code should be at one place and one place only
- Data should be at one place and one place only
- Object Oriented design
- No copy - paste
- Year 2000 problem would have been avoided

Basic Structure of Embedded System



Types of memory

- ROM
- PROM
- EPROM
- FLASH
- EEPROM
- FRAM
- RAM
  - Static
  - Dynamic
  - Serial
  - With battery

Compilers

- Do you know what your compiler does to your software?
- Versions of the compilers
- Optimizers
- Atomic expressions
- Interrupt latency if compiler makes atomic sets

RTOS - Real Time Operating System

- **Organizes concurrent processes**
- ***Event-driven* switches tasks only when an event of higher priority needs service**
- ***Time-sharing* switches tasks on a regular clock interrupt, and on events**

Response times

- Slow
  - No specified deadlines
  - Most PC software
  - Wasting our time
- Interactive
  - Adequate deadlines
  - Productive business software
  - Good CAD systems
- Soft real time
  - Almost always on time
  - Audio/Video encoding/decoding
- Hard real time
  - Always on time
  - Embedded control loops

Issues with processes in Embedded Software

- **Concurrency**
  - Many concurrent (possibly interacting) processes
- **Deadlock**
  - Two processes waiting for each other
  - E.g. higher priority process waiting for lower priority process
  - Causes a (sub)system to stop functioning
- **Livelock**
  - Two processes infinitely communicating with each other preventing other subsystems access
- **Race condition**
  - If the outcome of two signals depends on the order of these signals arriving
- **Determinism**
  - Predictability of cause and effect
  - We must *know* why it works

Memory allocation

- **Static memory allocation**
  - You *know* where your memory is
- **Automatic memory allocation**
  - Stack (LIFO: Last-in – First-out)
- **Dynamic memory allocation**
  - You ask for memory and get it, *if it's available*
  - If it's not available, you're stuck
  - Creates holes in memory: memory eventually gets exhausted !

Memory leak

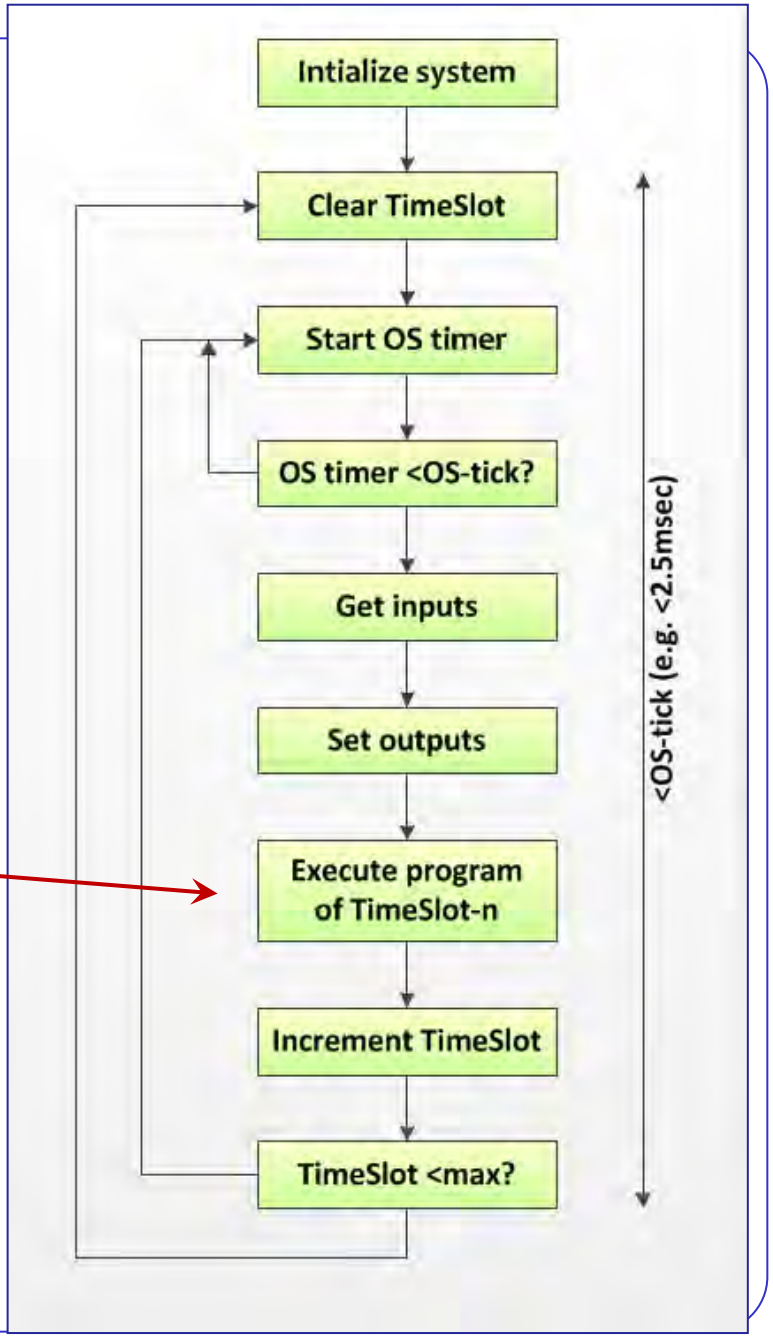
- Memory is claimed at start and not returned at exit
- Example: hand-held device for shopping
  - Device didn't really need dynamically allocatable memory
  - Reliability: MTBF ~ 20 sec
  - Availability: zero
  - Boot-delay: too long
  - After solving all memory leaks: Device simply worked

Garbage collection

- If the program doesn't clean up it's own garbage …
- Clean-up of unused memory
- Used by Java, C# and many other languages
- Not used by C, C++ (that's why most embedded systems use C or C++)
- Reallocation to remove gaps
- Non-deterministic
  - Various approaches
  - How long does it take ?
  - When is it done ?
  - Is it reliable ?
- If reliability means *knowing why* it is reliable …

Time-Slice Hard Real Time OS

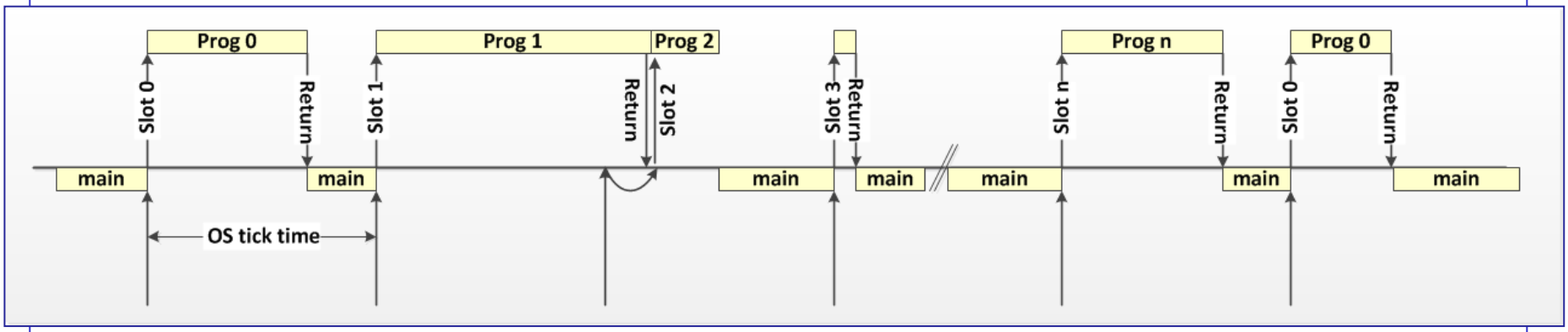
2.5ms slice	Actions
0	Reserved
1	Reserved
2	Reserved
3	Handle outputs
4	Read inputs and filter temperature
5	Prepare I²C to monitor (see steps 15~18)
6	Standby process
7	Degauss process
8	LED's
9	Start-up counters and blank process
10	PC watchdog process
11	Hour counters process
12	Protocol command handler 1
13	Protocol command handler 2
14	Protocol command handler 3
15	I²C message to monitor 1 or 5
16	I²C message to monitor 2 or 6
17	I²C message to monitor 3 or 7
18	I²C message to monitor 4 or 8
19	Keep writing to EEPROM (BusyEE)



20 TimeSlice of 2.5 msec = 50 msec

2.5ms slice	Actions	Software file	Analog measurement	EEPROM access
0	Reserved	timerhnd.asm	Temperature	Writing to EEPROM (if started at step 19)
1	Reserved	timerhnd.asm	Power down	Still writing to EEPROM (if 10msec write)
2	Reserved	timerhnd.asm	Power down	Still writing to EEPROM (if 10msec write)
3	Handle outputs	outputs.asm	Power down	Finishing writing to EEPROM
4	Read inputs and filter temperature	tempfil.asm	Power down	Read from EEPROM possible
5	Prepare I²C to monitor (see steps 15~18)	monitor.asm	Power down	Read from EEPROM possible
6	Standby process	standby.asm	Power down	Read from EEPROM possible
7	Degauss process	degauss.asm	Power down	Read from EEPROM possible
8	LED's	leds.asm	Power down	Read from EEPROM possible
9	Start-up counters and blank process	sutimrs.asm	Power down	Read from EEPROM possible
10	PC watchdog process	pcw.asm	Power down	Read from EEPROM possible
11	Hour counters process	hours.asm	Power down	Read from EEPROM possible
12	Protocol command handler 1	process.asm	Power down	May read from EEPROM for command
13	Protocol command handler 2	process.asm	Power down	May read from EEPROM for command
14	Protocol command handler 3	process.asm	Power down	May write to EEPROM for command
15	I²C message to monitor 1 or 5	monitor.asm	Power down	Writing to EEPROM (if started at step 14)
16	I²C message to monitor 2 or 6	monitor.asm	Power down	Still writing to EEPROM (if 10msec write)
17	I²C message to monitor 3 or 7	monitor.asm	Power down	Still writing to EEPROM (if 10msec write)
18	I²C message to monitor 4 or 8	monitor.asm	Power down	Finishing writing to EEPROM
19	Keep writing to EEPROM (BusyEE)	process.asm	Power down	May write to EEPROM if not yet done

Timing


Prog 0
Prog 1
Prog 2
Prog n
Prog 0
Slot 0
Return
Slot 1
Return
Slot 2
Slot 3
Return
Slot n
Return
Slot 0
Return
main
main
main
main
main
main
main
OS tick time

Interrupts

- Interrupt latency – time from interrupt to start of ISR
- Takes time to finish the current instruction
  - From one to many clock-cycles (eg multiply!)
- Takes time to save the current context
- Takes time if interrupts are disabled
  - By software to protect non-atomic instructions
  - By other running interrupt
- Interrupts introduce non-determinism (unpredictability)
- So we must use them very carefully

SH7125 Exceptions

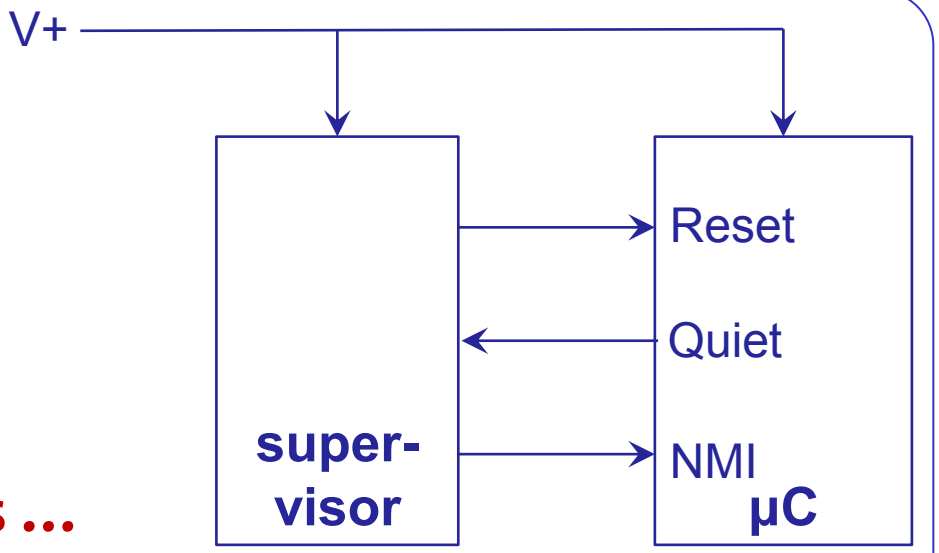
Table 5.1 Types of Exceptions and Priority

Exception	Exception Source	Priority
Reset	Power-on reset	High
	Manual reset	
Interrupt	User break (break before instruction execution)*3	
Address error	CPU address error (instruction fetch)	
Instruction	General illegal instructions (undefined code)	
	Illegal slot instruction (undefined code placed immediately after a delayed branch instruction*1 or instruction that changes the PC value*2)	
	Trap instruction (TRAPA instruction)	
Address error	CPU address error (data access)	
Interrupt	User break (break after instruction execution or operand break)*3	
	NMI	
	IRQ	
	On-chip peripheral modules	Low

Why do we need a watchdog ?

- Design errors (?)
- Software errors (?)
- Hardware errors (interference, wear)
  - Radiation from space
  - Radiation from adjacent sources
  - CE - EMC directive
  - Power-sequences
- Count the number of watchdog restarts for analysis
  - Should stay at zero

Watchdog



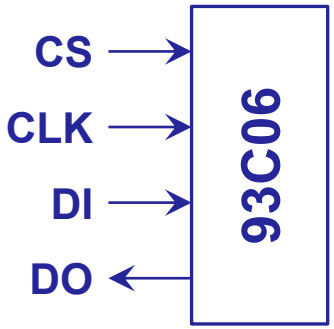
- Resets the system unless …
- Regularly kicking the dog before it barks
- If the system doesn't behave as it should
- Making sure the watchdog is and keeps enabled
  - Can only set at first few instructions
  - Can only reset with specific instruction
- Only hardware isn't enough
- Brown-out
  - What happens if NMI at non-atomic code ?

Software watchdog issues

- What if the program runs outside program space ?
- Within memory
  - empty ROM doesn't exist
  - random RAM
  - memory mapped I/O → can unsafe things happen ?
- Outside memory
- Did the software run in the expected order ?
  - Check that we run the program in the expected order

Debugging

- Don’t !
- First find a theory how the phenomenon could happen
- Check the theory
- Example EEPROM losing data



Example: EEPROM parameter accidentally erased !

- Production since several years
- Factory calibration parameter disappears from EEPROM
- Only read once upon start of program
- How *can* this happen ?

Instruction Set for the NM93C06L and NM93C46L

Instruction	SB	Op Code	Address	Data	Comments
READ	1	10	A5–A0		Reads data stored in memory at specified address.
WEN	1	00	11XXXX		Enable all programming modes.
ERASE	1	11	A5–A0		Erase selected register.
WRITE	1	01	A5–A0	D15–D0	Writes selected register.
ERAL	1	00	10XXXX		Erases all registers.
WRALL	1	00	01XXXX	D15–D0	Writes all registers.
WDS	1	00	00XXXX		Disables all programming modes.

- READ : 110000000
- ERASE: 111000000

Reset → Start READ 1
Reset again → Start READ 110000000
Result ERASE 1110000000

FIGURE 2-7: READ TIMING

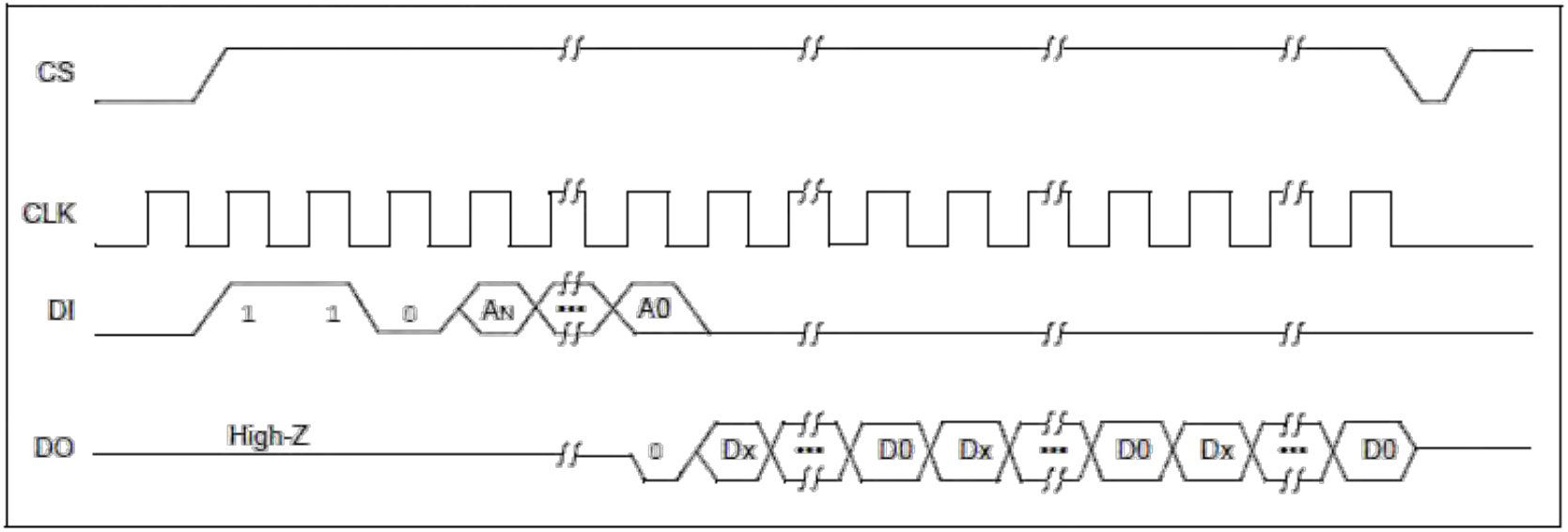


FIGURE 2-1: ERASE TIMING FOR 93AA AND 93LC DEVICES

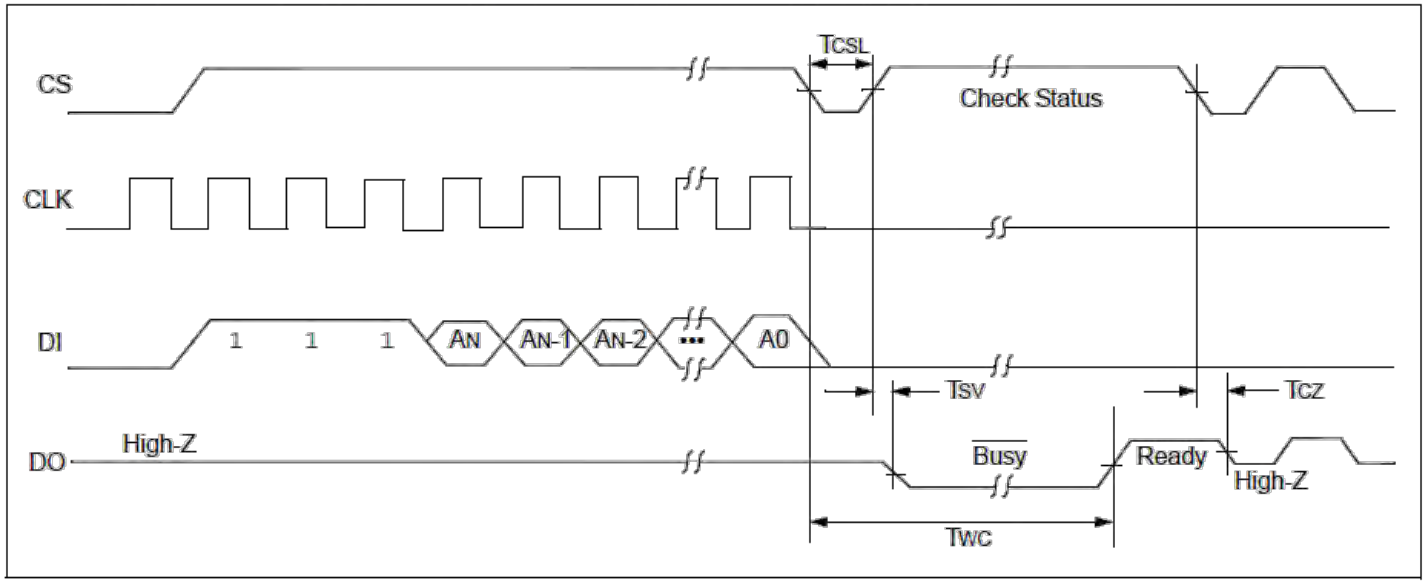
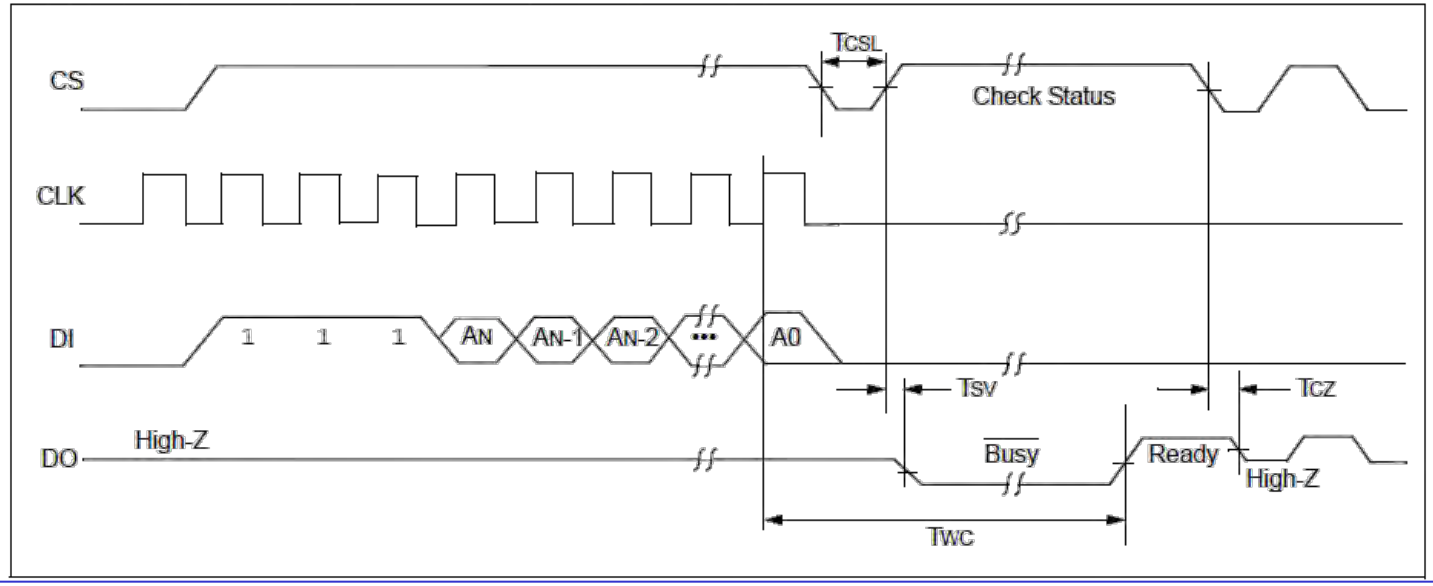


FIGURE 2-2: ERASE TIMING FOR 93C DEVICES



Measuring or testing

- **If one product tests OK, the next product may be not OK**
- **It doesn't prove that all repeat products will work the same**
- **This has to be proven *by design***

Interfaces

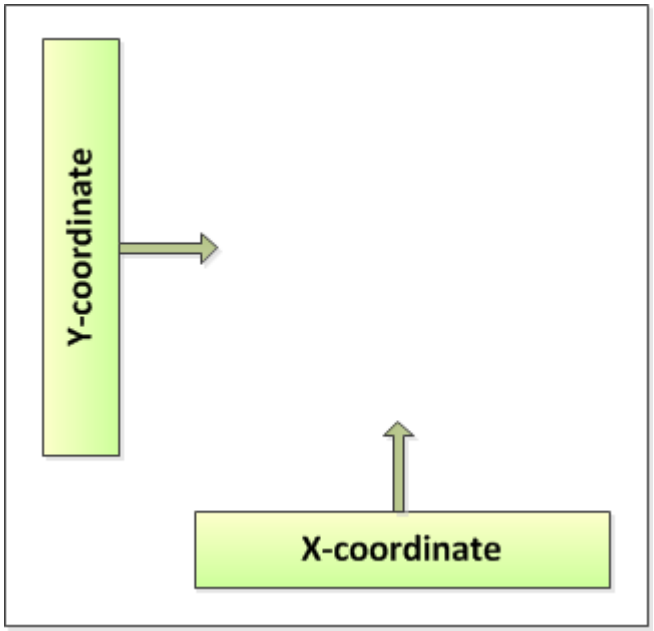
- Digital input / output
- Analog input / output
- Timer/counter (measuring frequency)
- Capture (measuring time)

All components are imprecise

- Voltage levels
- Current levels
- Timing

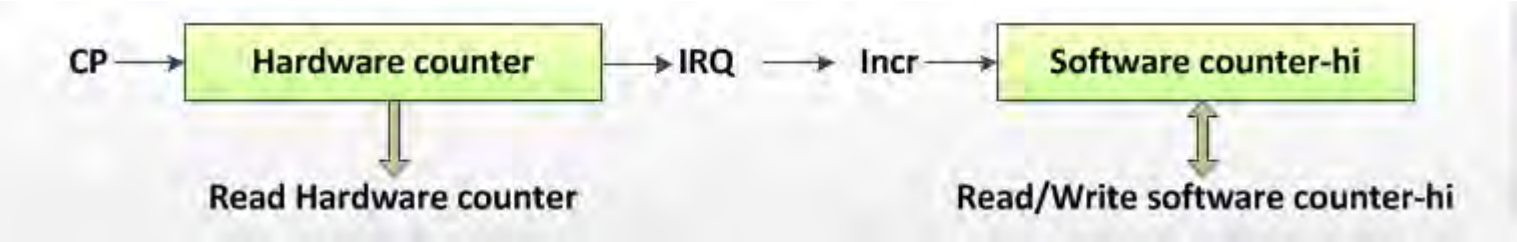
- What is worst case ?
  - Over temperature
  - Over life time
  - Over voltage range

Reading X and Y coordinates



- **More general: reading two or more values (not) at the same time**

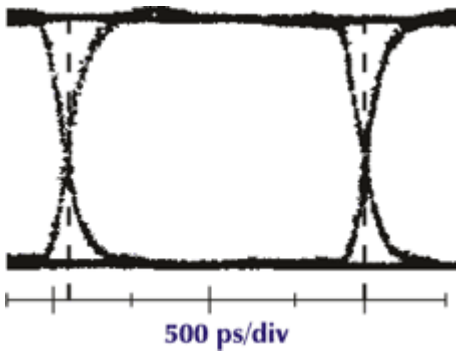
Non atomic

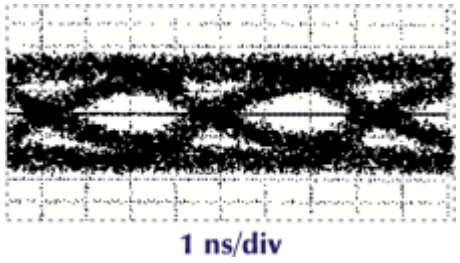


- Overflow Hardware counter 0xff → 0x00 → IRQ
- IRQ → Increment Software counter-hi
- If reading the counter value:
  - Read HWC: 0xff
  - (unaware of IRQ) read SWC-hi: 0x01 (incremented by IRQ)
  - We read: 0x01ff in stead of 0x00ff
- How can we solve this ?
- Such issues should be solved at only one place

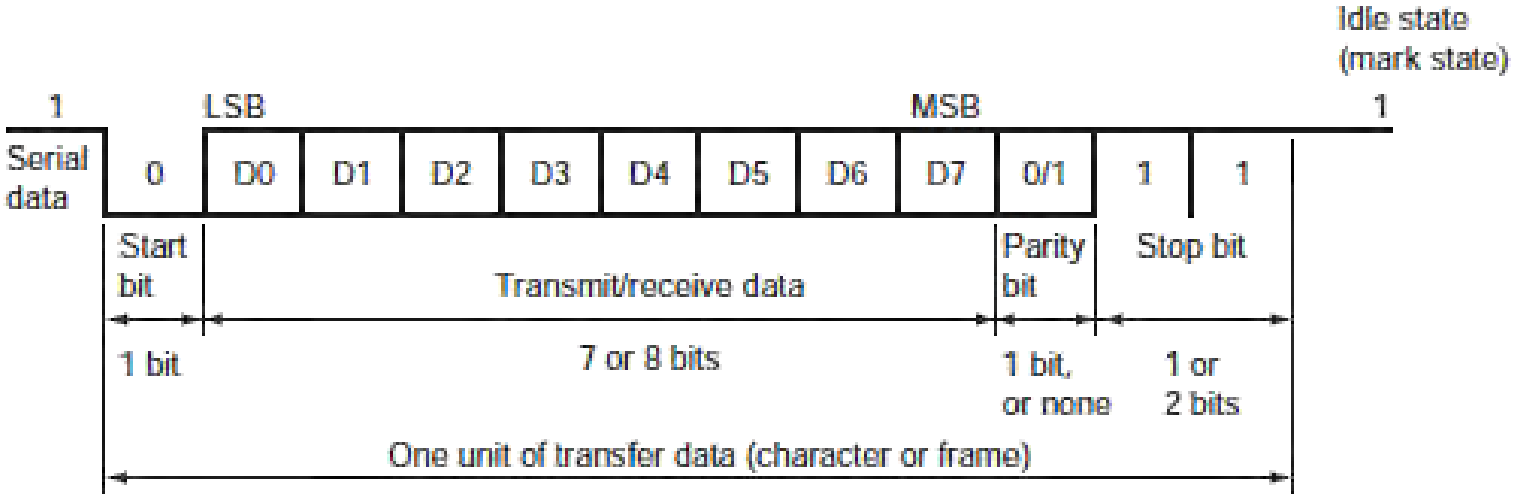
Serial data stream

0110001101100011101101100110100101010110101010010101101101010100110101

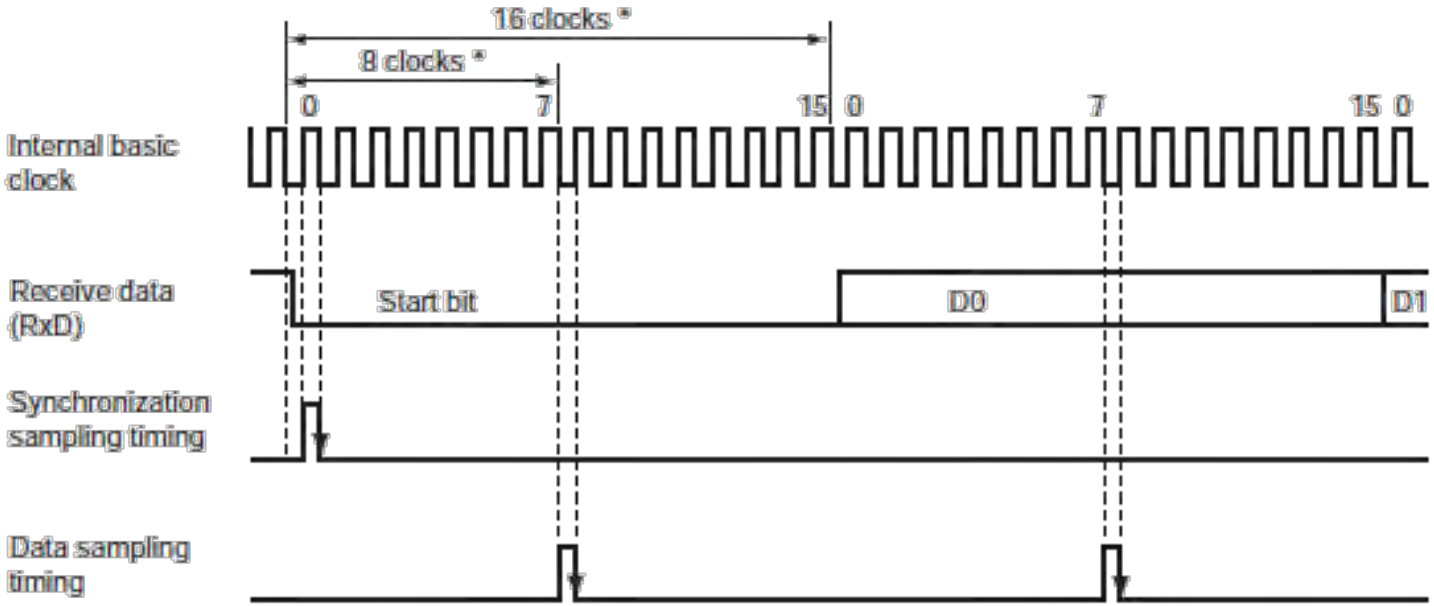




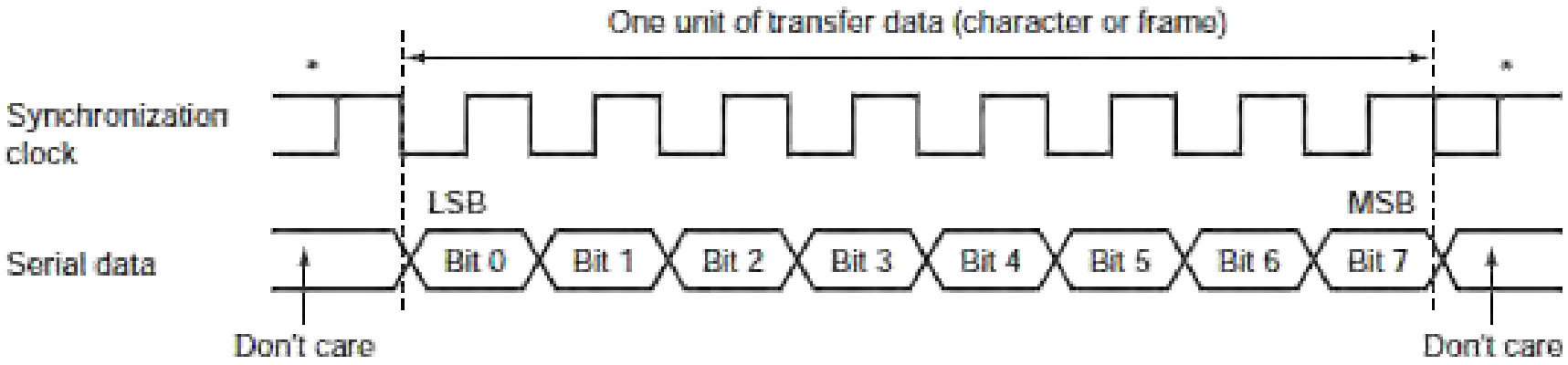
Asynchronous communication



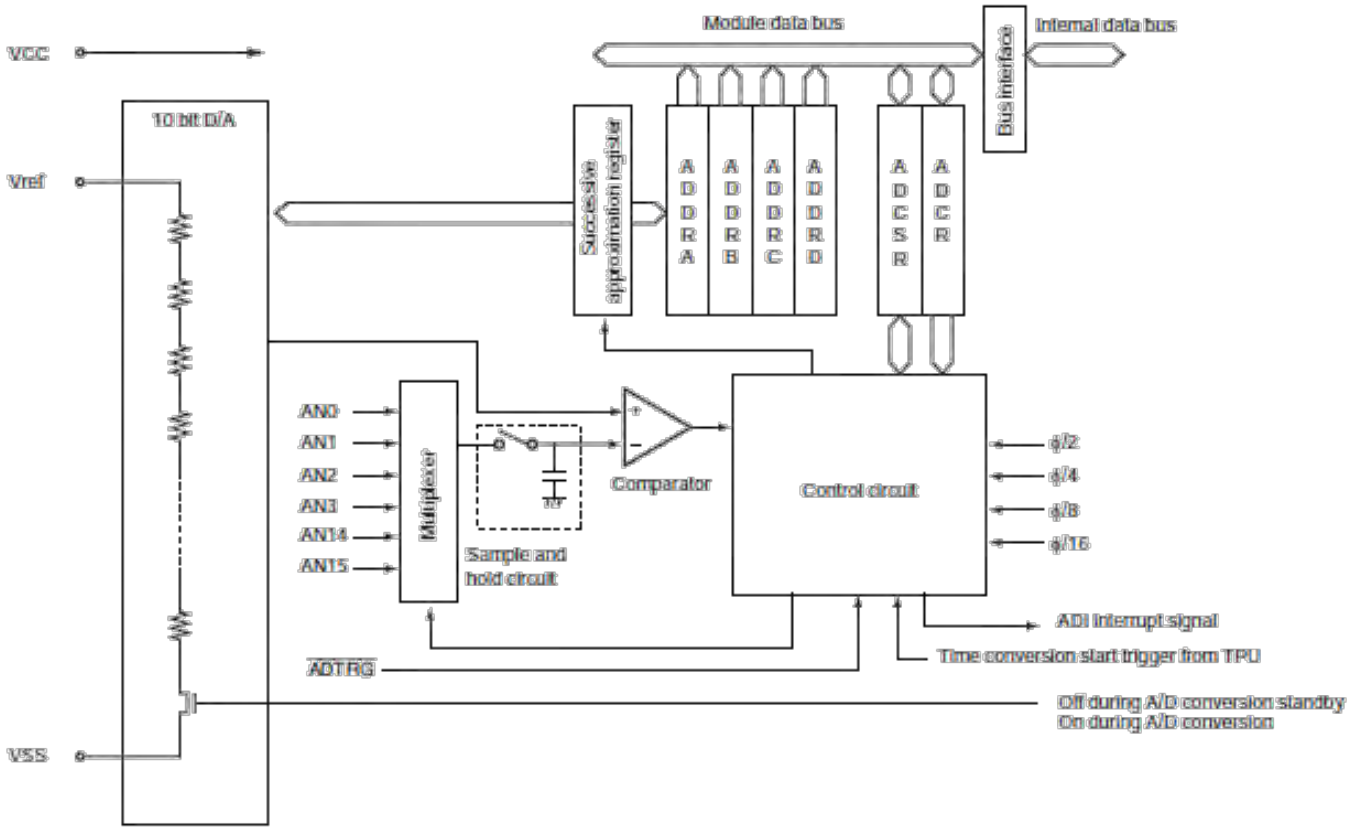
Asynchronous Clock Synchronization



Synchronous communication



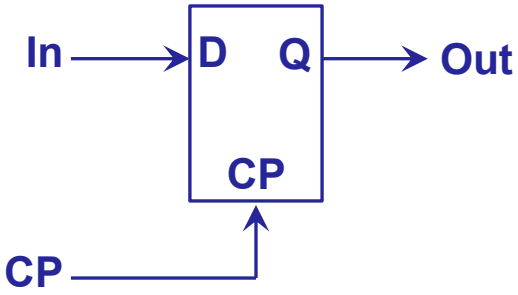
A/D conversion

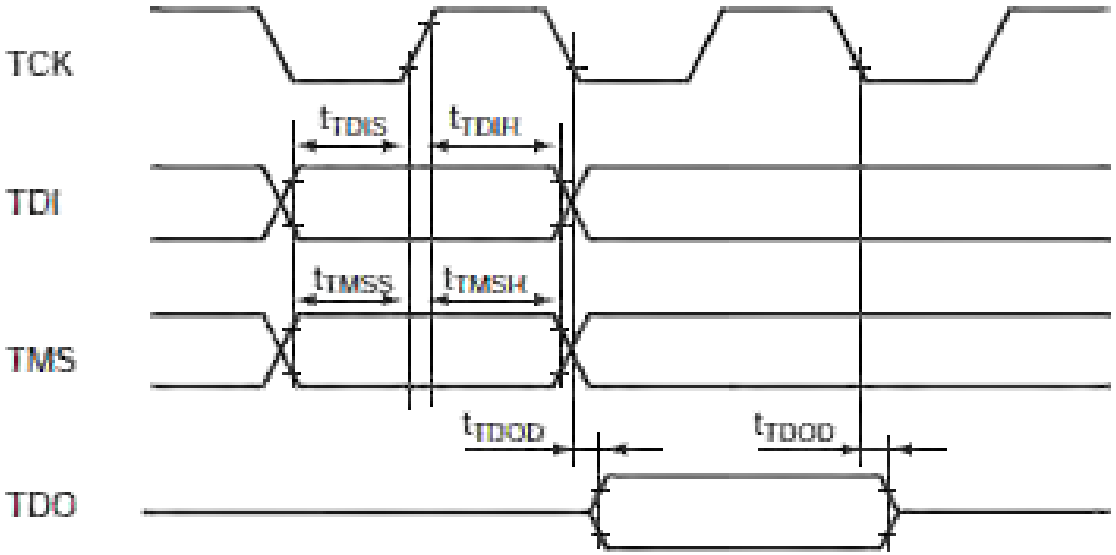


Worst case values

Item		Symbol	Min.	Typ.	Max.	Unit	Test Conditions
Input low voltage	$\overline{RES}$, $\overline{STBY}$, MD2 to MD0, $\overline{TRST}$, TCK, TMS, TDI, EMLE, VBUS, $\overline{UBPM}$, FWE*4	V_{IL}	–0.3	—	$V_{CC} \times 0.1$	V	
	EXTAL, NMI, ports 1, 3, 4, 7, 9, and A to G		–0.3	—	$V_{CC} \times 0.2$	V	
Output high voltage	All output pins	V_{OH}	V_{CC} – 0.5	—	—	V	I_{OH} = –200 µA
			V_{CC} – 1.0	—	—	V	I_{OH} = –1 mA
Output low voltage	All output pins	V_{OL}	—	—	0.4	V	I_{OL} = 0.8 mA
Input leakage current	$\overline{RES}$, VBUS, $\overline{UBPM}$, $\overline{STBY}$, NMI, EMLE, MD2 to MD0, FWE*4, ports 4, 9	$\lvert I_{in} \rvert$	—	—	1.0	µA	V_{in} = 0.5 to V_{CC} – 0.5 V
Three-state leakage current (off state)	Ports 1, 3, 7, and A to G	$\lvert I_{TSI} \rvert$	—	—	1.0	µA	V_{in} = 0.5 to V_{CC} – 0.5 V
Input pull-up MOS current	Ports A to E TDI, TCK, TMS, $\overline{TRST}$	$-I_P$	10	—	300	µA	V_{in} = 0 V
Input capacitance	$\overline{RES}$, NMI	C_{in}	—	—	30	pF	V_{in} = 0 V f = 1 MHz
	All input pins other than $\overline{RES}$, NMI		—	—	15	pF	T_a = 25°C
Current dissipation*1	Normal operation	I_{CC}*2	—	22 V_{CC} = 3.3 V	35 V_{CC} = 3.6 V	mA	f = 16 MHz
	(USB halts)		—	31 V_{CC} = 3.3 V	50 V_{CC} = 3.6 V	mA	f = 24 MHz

Timing

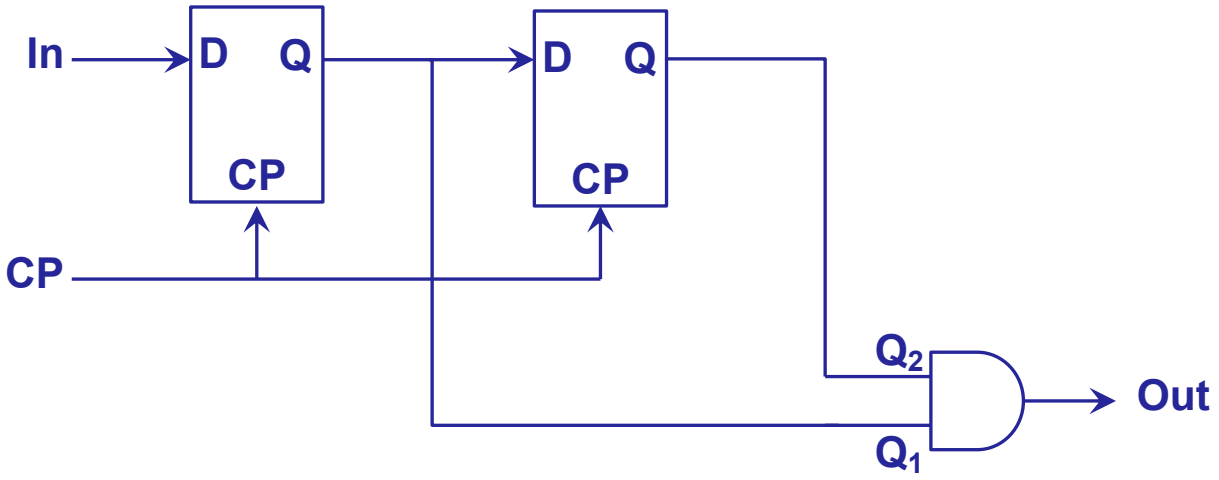

In
D
Q
Out
CP
CP

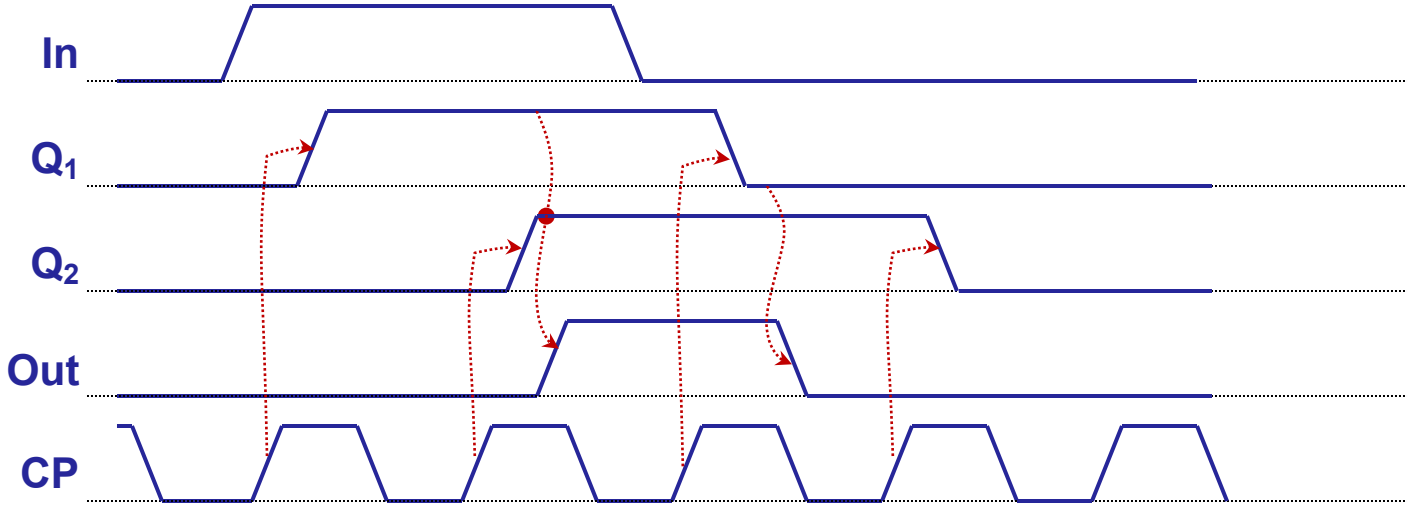

TCK
TDI
TMS
TDO
tTDIS
tTDIH
tTMSS
tTMSH
tTDOD
tTDOD

Metastability

- Digital samples
- If data changes exactly when the sample is taken
- Set-up and Hold times
- Metastability
- Multi-bit issues

Digital debounce

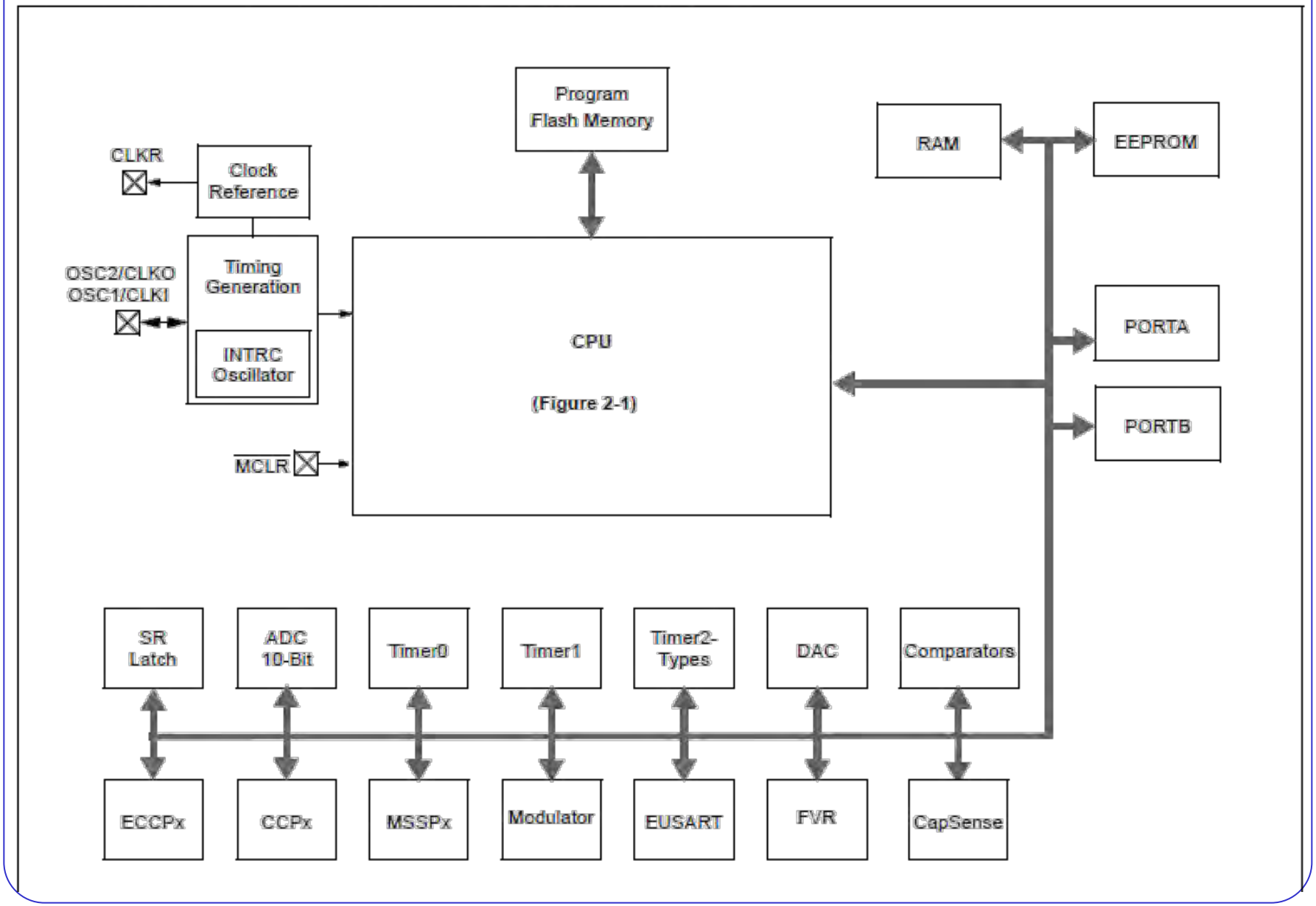




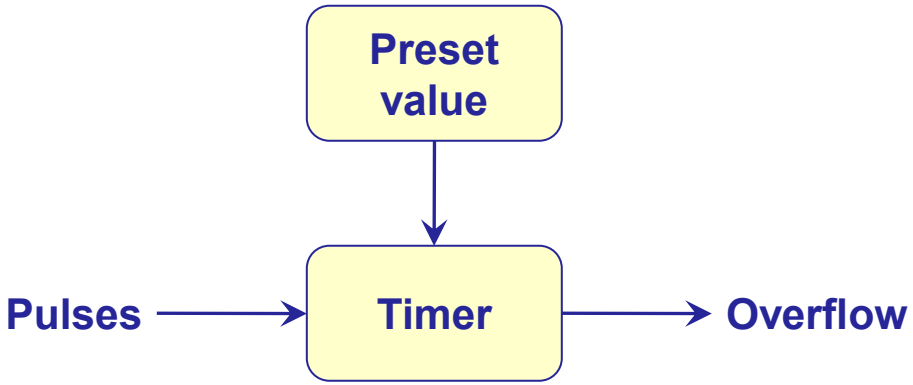
Flash programming

Item	Symbol	Min.	Typ.	Max.	Unit
Programming time*1 *2 *4	t_P	—	10	200	ms/128 bytes
Erase time*1 *3 *5	t_E	—	50	1000	ms/block
Reprogramming count	N_{WEC}	100*6	10000*7	—	Times
Data retention time*8	t_{DRP}	10	—	—	Years

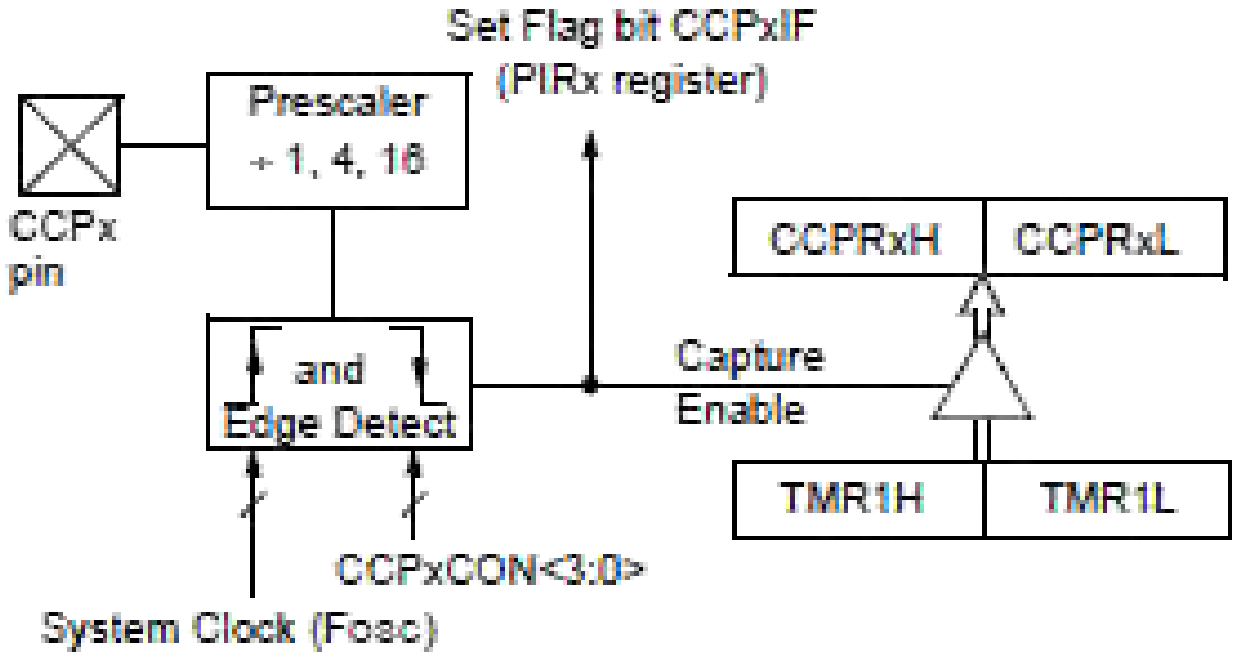
FIGURE 1-1: PIC16F/LF1826/27 BLOCK DIAGRAM



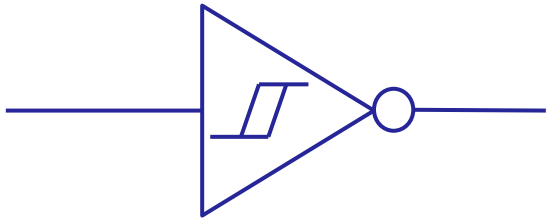
Timer/Counter

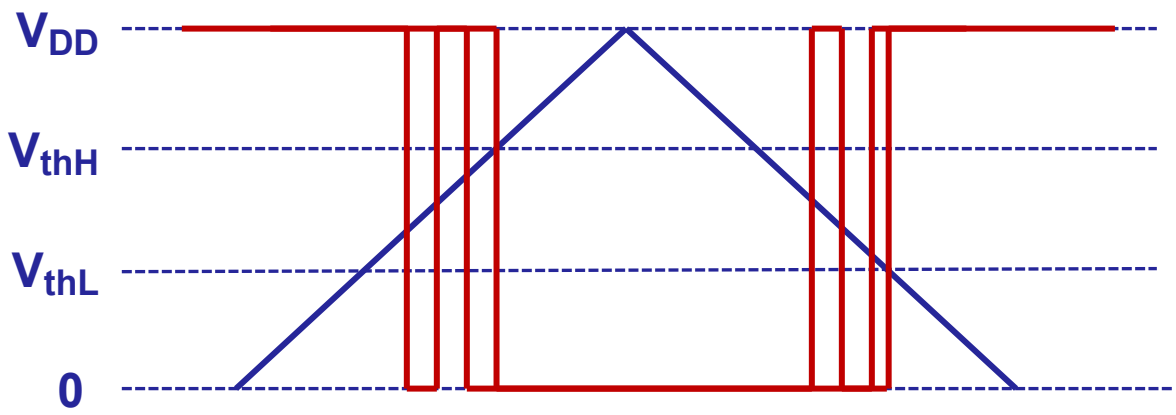


Pulse capture

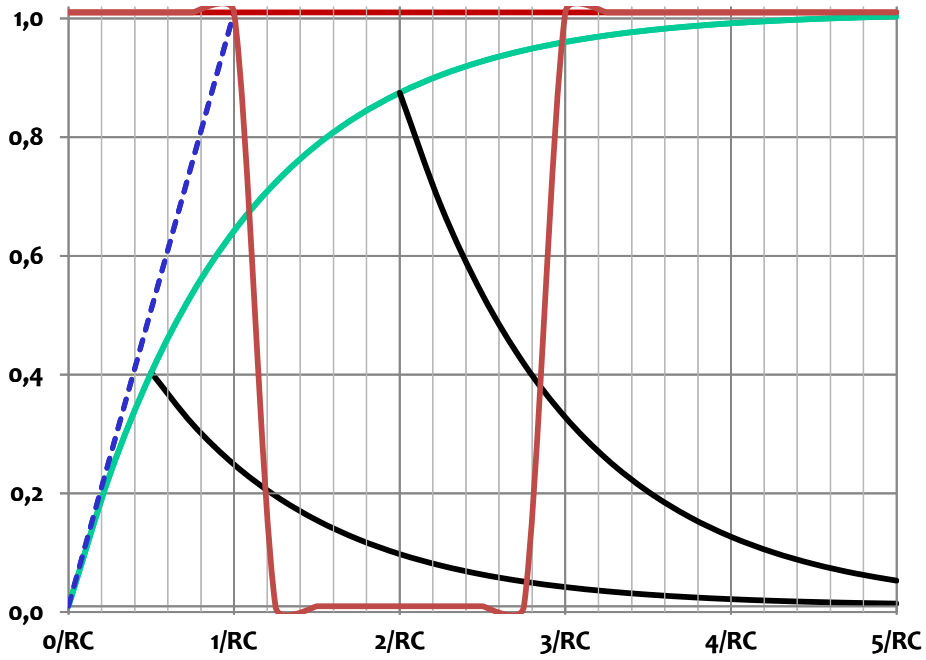


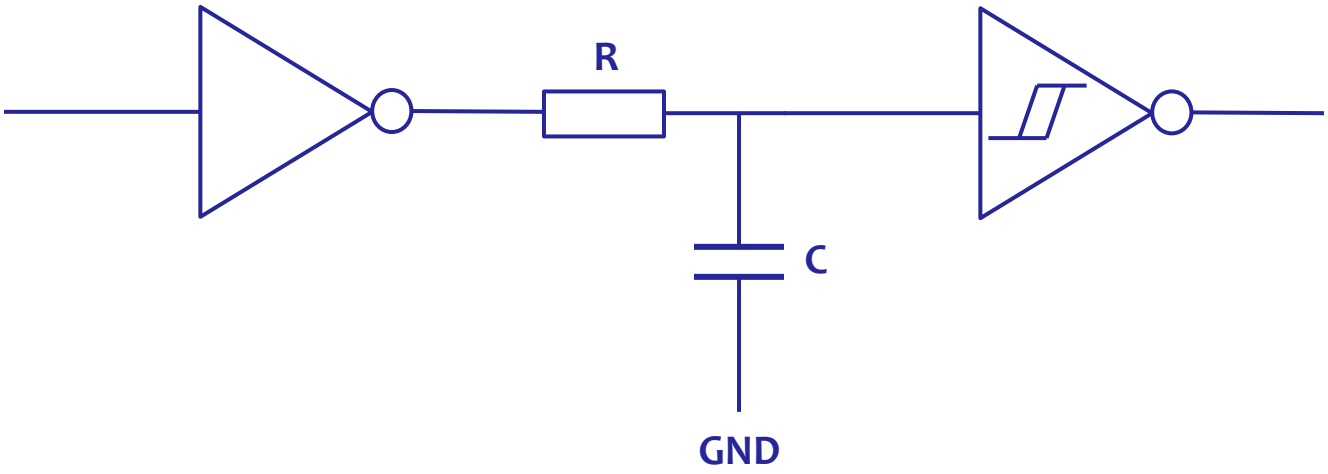
Schmitt-trigger



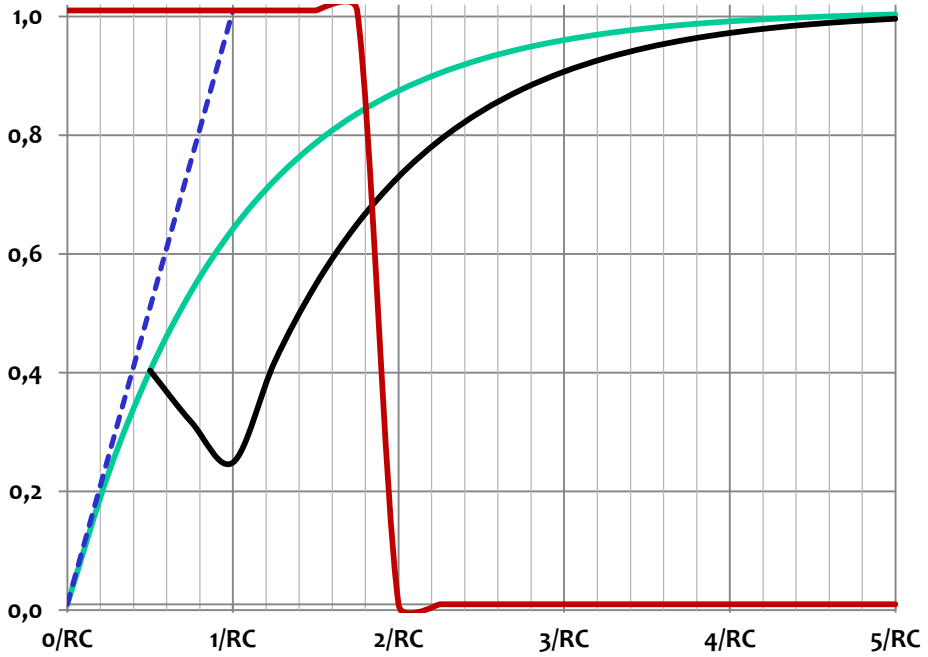


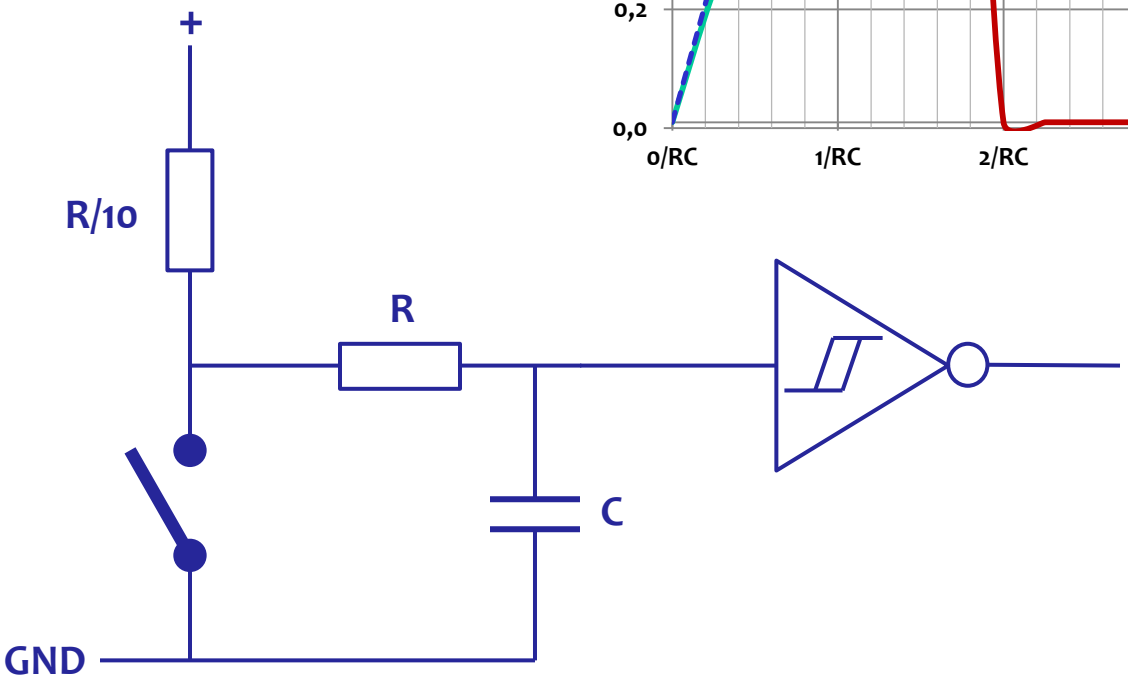
Delay or filter





Debounce





Handshaking

- I have something to send
- OK, you may send it
- Sending data
- I received the data OK
- End of conversation

Time Triggered Architecture

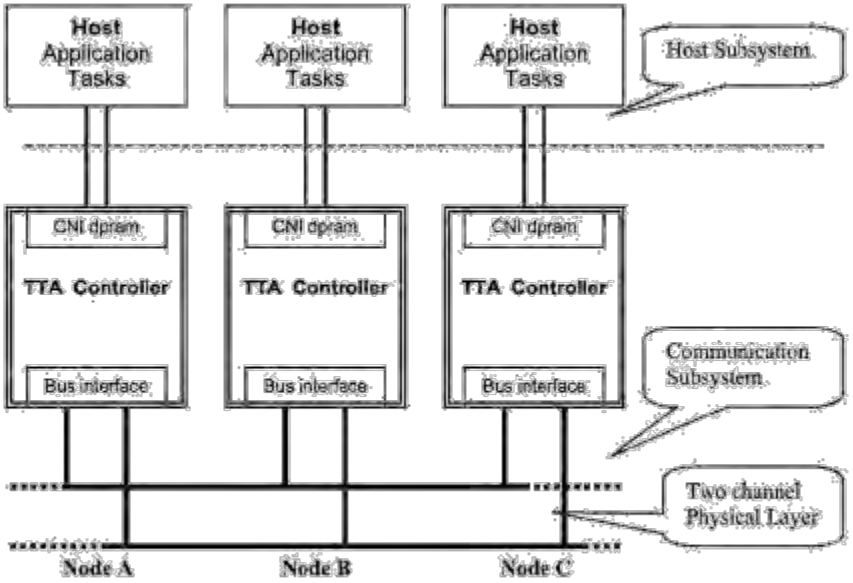


Figure 1: TTA Cluster Architecture

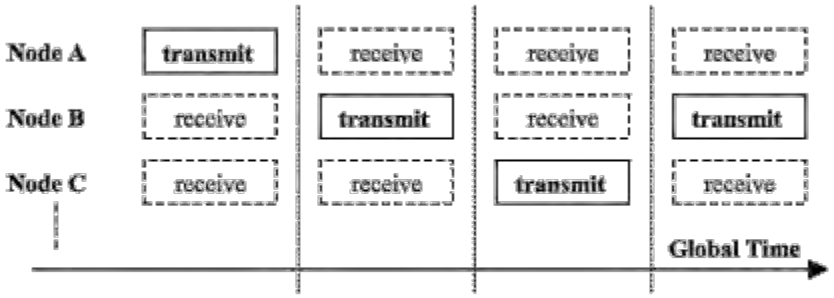


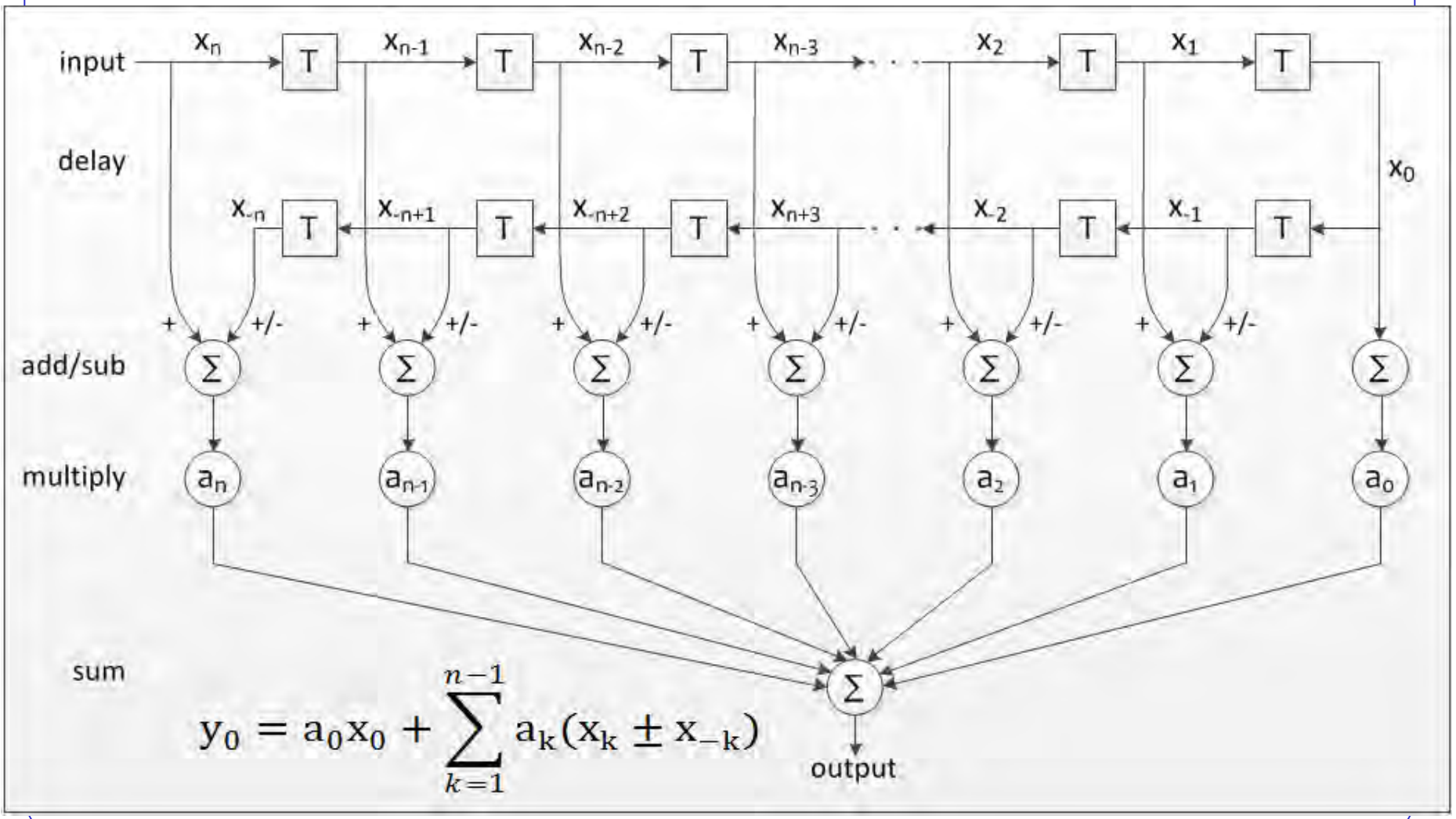
Figure 2: TTA Bus Access Schema

Reliable ?

- Don’t believe anything I say
- You may do anything
- As long as you *know* that it works, and *why* it works
- *Assuming* that your (or worse: *their*) design works, is dangerous
- Assume you (and them) probably made mistakes
- Don’t trust yourself, use Reviews and Inspections
- Assume that you don’t know everything
- Know how to find it out

Digital signal processing principle

Digital Signal Processor - IIR or FIR filter



Possible exercise

- Nano-satellite
- Powered only by solar panels (no battery)
- Two processors having to work together

Seminar Program 7 ~ 11 June 2010 - every morning 9:00 - 12:30

Monday: General introduction Reliable Embedded Systems

- What causes failure and what can we do about it

Tuesday: Requirements and Design

- What are we supposed to accomplish
- How to select the best way to do that
- How to document for better understanding

Wednesday: Planning

- How to make sure we'll be ready on time
- You will learn how to accomplish much more in less time

Thursday: Testing, Reviews and Inspections

- Learning to find our mistakes early and never make them again

Friday Master Class: actually organizing our project

- Using what we learnt to set up our project for success