NUMERO: 2323A ANNO: 2018

A P P U N T I

STUDENTE: Caldera Filippo

MATERIA: Model Based Software Design - Teoria - Materiale d'Esame - Esercizi - Prof. Massimo Violante

MBSD

MODEL BASED

SOFTWARE DESIGN

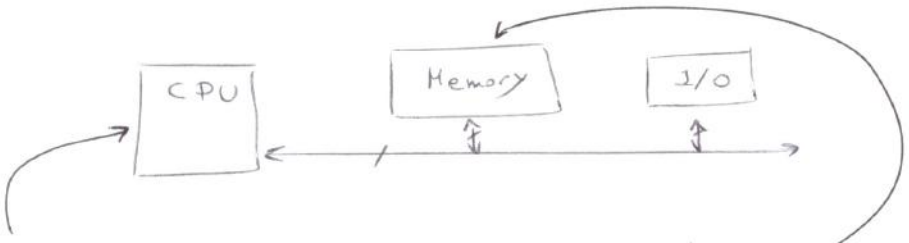
- M. VIOLANTE -

FILIPPO
CALDERA

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 3 di 92

BASIC COMPUTER ARCHITECTURE

Easiest way to represent a computer is:



- **CPU** executing all-time-long the code present in the **Memory**
- **Memory** place where code to be executed is stored could be VOLATILE or NO-VOLATILE
- **I/O** Since we're talking about real-world application CPU must know what's outside the computer I/O Provides the ability to "interface" CPU with sensors and actuator.

Most critical moments are powering-on and powering-off.

@ Powering on CPU needs to knows where is the first instruction.

⇓

There is a special spot RESET VECTOR = place in memory where 1st instruction is located

Of course 1st instruction MUST BE VALID

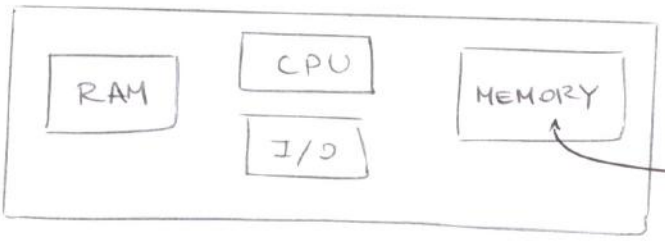
also exploited for powering off

In case power goes down 1st instruction will be executed @ power on anyhow.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 5 di 92

ARCHITECTURES

Sow basic blocks are:



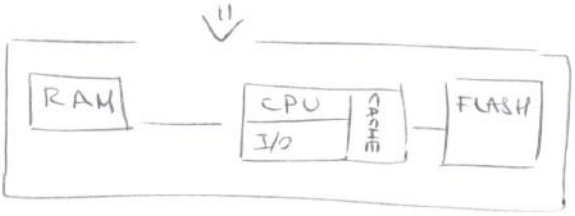
2 generic ways of implementations

System on Chip (SoC)

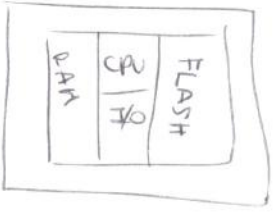
⚠ BOARD ≠ Chip

Microcontroller (μC)

Blocks separated but located in the same board

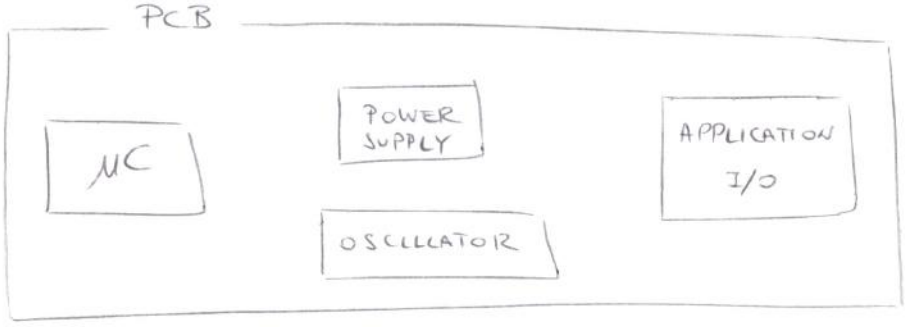


All 4 blocks integrated in the same Si chip



used in k64 Board

In mechatronic system usually we have



Some drivers got killed because the most instinctive driver's responses didn't work.

↳ must be considered even in panic/critical case

Ex.
to stop VA was just five to

- Turn vehicle off
- Going neutral gear

} Ok, but this is not instinctive

⇓

In panic situation that option won't work.

Some causes

- No mirroring critical variables
- Stack overflow
- Watchdog set in wrong way

} All these cases are not a sign of not testing enough but, since the high complexity of the software is impossible to be absolutely sure of a complete debugging

ISO 26262

=

International Standard for Automotive

→ applies safety-related road vehicle electronic & electrical

Risk evaluated on customer risk by identifying Automotive Safety Integrity Level (ASIL) allocated with each undesired effect

based on FUNCTIONAL SAFETY

≠

SAFETY

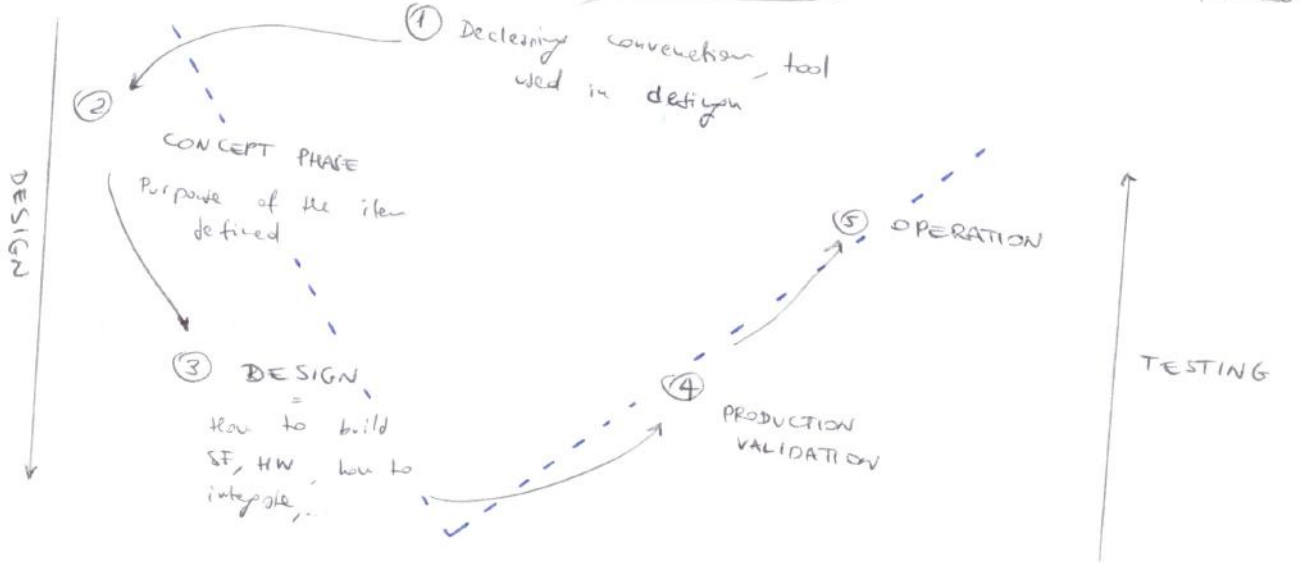
FUNCTIONAL SAFETY = taking care no matter what is gone wrong.

↓

Safety not guaranteed by strongly interested only different methods.

dealing with what happens when safety responsible system doesn't work.

During development of a product the V model process is followed



SOME CONCEPT

- ITEM = what you want to built, following ISO 26262
- HARM = physical injury / damage to people

  ⇓

  Don't care about item

  JUST PEOPLE HEALTH
- SEVERITY = how bad injuries would be
- FAILURE = item no more able to perform correctly a certain task

  ↳ always happen ⇒ MUST BE READY TO DEAL WITH THEM
- RISK = $P(H) \cdot S$ = combination of probability of an harm and its severity

  ⇓

  If $P(H) \downarrow$, $S \uparrow$ ⇒ Risk is low
- CONTROLLABILITY = if anything goes wrong it's possible to control a task with the failed item
- EXPOSURE = How a failure can affect the ability to perform a correct operation.
- Automotive Safety Integrity Level (ASIL)

  Just a correction how critical the item is

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 13 di 92

03/17/2017

When design a product in automotive is needed to define

- GOAL = what item is intended to do
  - ⇓
  - Understanding item's elements and theirs connections and interactions
  - ⇓
  - Need to know
    - Item's element
    - Interaction = how, where data flow come from
    - Fuctionality provided
    - " Required to achieve the defined task.

During design is ALSO VERY IMPORTANT the

HAZARD ANALYSIS AND RISK ASSESSMENT

=

understanding what would happen if an item gets broke and their effects and risk

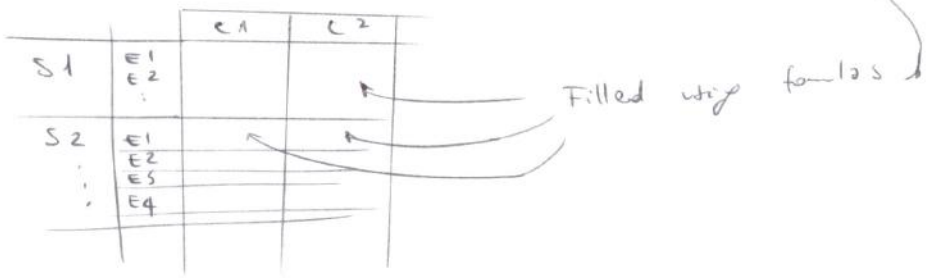
- Lot of Brainstorming needed ⇐ Not defined by a convention to follow step by step
- Very Expensive (But needed)

ASIL can be defined as following.

$S + E + C =$

- $\leq 6 \Rightarrow$ ASIL = Q.M (ok)
- $= 7 \Rightarrow$ ASIL = A
- $= 8 \Rightarrow$ ASIL = B
- $= 9 \Rightarrow$ ASIL = C
- $= 10 \Rightarrow$ ASIL = D (IMPORTANT)

rappresented in a matrix



EX.

S3 = High Severity; E1 = Low P; C1 = High controllability $\Rightarrow$ Q.M = Low Risk

⇓

No improvment needed

=

Not wholly spending money to fix issue

EX

S2 = Severe Severity; E4 = High P; C3 = Low controllability $\Rightarrow$ C = Risky

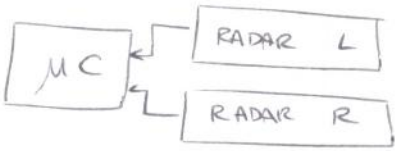
⇓

Improvment needed

EXAMPLE

CTA = system helping exiting from parking lot whether if there are cars/bikes inbound

based on 2-radar μC ⇒



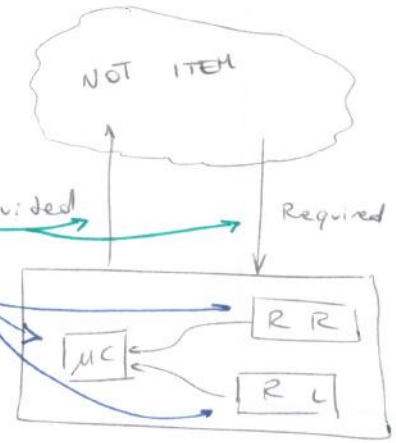
STEP 1: ITEM DEFINITION

STEP 1.1: ITEM'S ELEMENTS

- μC
- RADAR L
- RADAR R

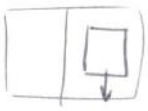
STEP 1.2: INTERACTION

- Buses to send data

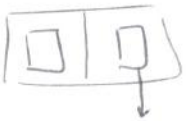


STEP 2: OPERATIONAL SITUATIONS

OS1: Free spot Car incoming



OS2: Spot taken Car Incoming



OS3: bike incoming Free spot

OS4: bike incoming Spot taken

code for only left-inbound

(right inbound would be the same)

STEP 4: HAZARD ANALYSIS & RISK ASSESMENT

OS1 — HZ1, HZ2, HZ3 — SEC
OS2 — HZ1 — SEC
⋮

For each OS must be considered all hazards.
For each hazards must be considered all the Severity, Exposure, Controllability

STEP 4.1: SEVERITY

↓ depends on the O.S.

must be considered all persons (including: cyclist, riders, pedestrian) ⇒ NOT DRIVER ONLY BUT ALL PEOPLE

STEP 4.2: EXPOSURE

If $P < 0.1\%$ ⇒ VERY LOW P ⇒ E1
If $P < 1\%$ ⇒ LOW P ⇒ E2
If $P < 10\%$ ⇒ MEDIUM P ⇒ E3
If $P \geq 10\%$ ⇒ HIGH P ⇒ E4

Given by
↓ PERCENTAGE OF OPERATING TIME
↓ FREQUENCY OF OCCURRENCE = how often a situation appears

STEP 4.3: CONTROLLABILITY

↓ estimation of the P[] driver/endangered people are able to gain control of the event to avoid harm

considering average driver ⇒ Assuming person with driver licence for driver.

STEP 5 : ASIL MATRIX

ASIL given by the sum of S, E, C.

So...

OS 1

H1 S+E+C = 6 ⇒ QM

H2 S+E+C = 5 ⇒ QM

H3 S+E+C = 6 ⇒ QM

OS 2

H1 4+2+1 = 7 ⇒ A

H2 4+2+0 = 6 ⇒ QM

H3 1+4+2 = 7 ⇒ A

OS 3

H1 2+4+1 = 7 ⇒ A

H2 0+4+1 = 6 ⇒ QM

H3 2+4+1 = 7 ⇒ A

OS 4

H1 2+4+2 = 8 ⇒ B

H2 0+4+2 = 6 ⇒ QM

H3 2+4+2 = 8 ⇒ B

Design is aimed to reduce the highest ASIL

(in this case max(ASIL) = B)

MODEL-BASED SOFTWARE DESIGN

03/22/2017

REAL WORLD ≠ MODEL = reality approximation

=

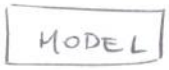
Abstraction

simplified or partial representation of reality

full of details

- Too difficult represent all details
- Some details are not of interesting

Idea:



all details different systems

Just:

- Relevant details

easier understanding

MBSD Concepts

When study a model to be implemented we focous on system's T.F.

⇓

No need how value would will get

⇓

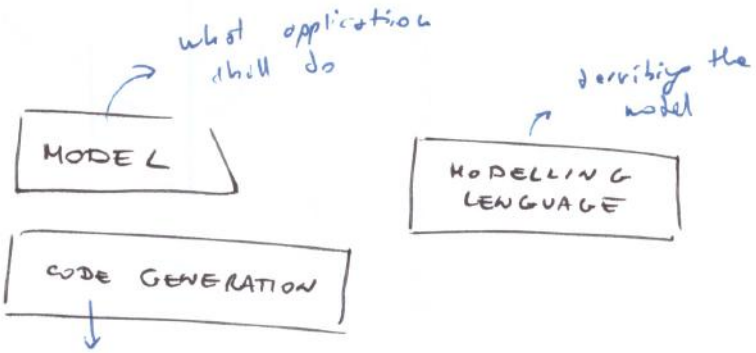
MODEL IS TECHNOLOGICALLY INDIPENDENT

ABSTRACTION

⇓

allows to develop software design and architect in parallel

MBSD Architecture



System functionality is described utily platform-indipendent model

MBSD USING SIMULINK

04/05/2017

In simulink model is defined by I/O and Integration step

Integration step defines when evaluate the next step values

Integration step = How fast model is updated

To simulate models simulink use the following approach:

1. Compute y
   Compute next state
   Compute Δ
2. Update time $t = t_0 + \Delta$

Solver Solution

Solvers types

Fixed Step

Δ is fixed in all execution time

⇒ t_sim is fixed

MUST PAY ATTENTION DUE ALIASING

Used in the course

Variable Step

Δ is adaptive to system dynamics

t_simulation can change

Good trade of between t_sim and precision

Continuous Solver

T_sim is divided in ΔT where system is evaluated

Good Accuracy

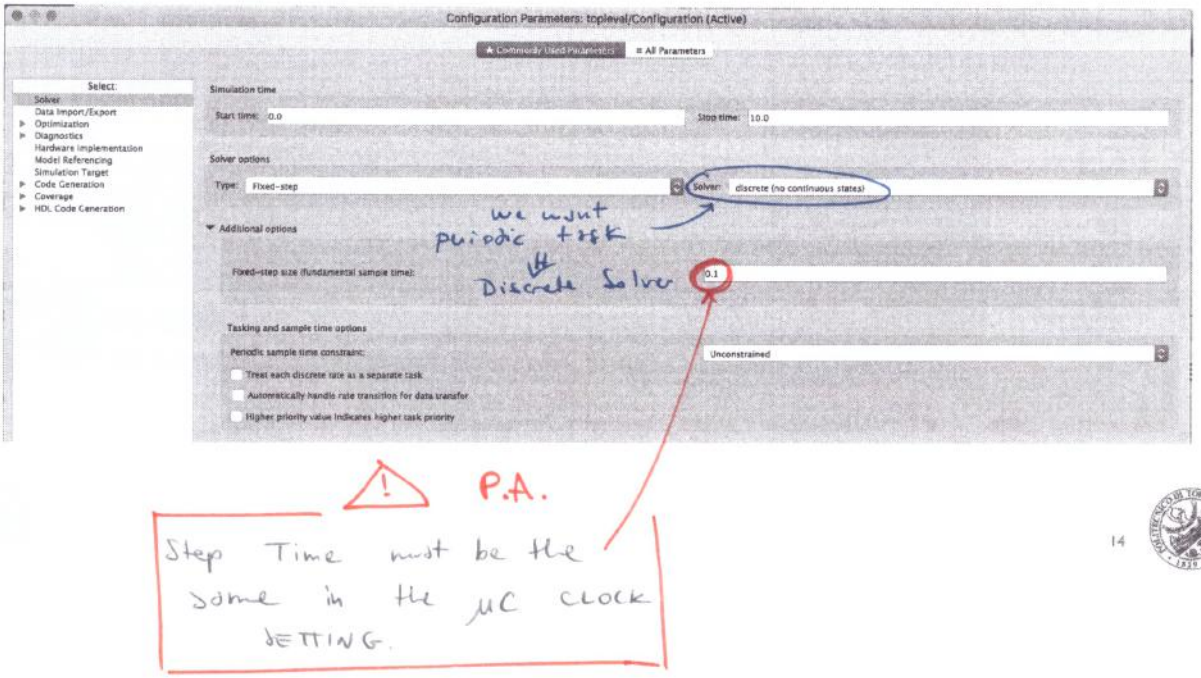
Discrete Solver

Simulation takes place in t_START, t_STOP only

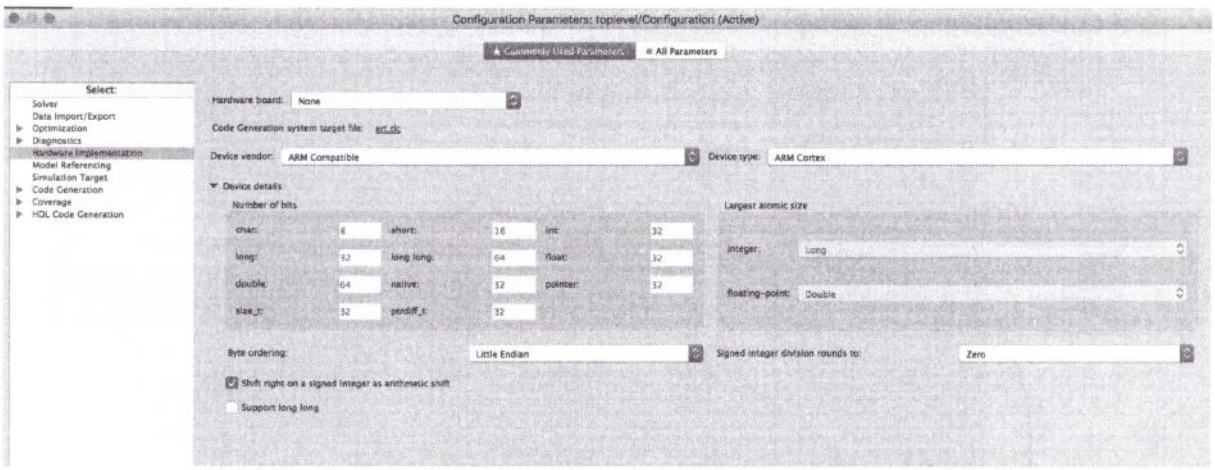
Low Accuracy

First step to do when starting a new project is to set-up the Simulink Solver

Solver configuration

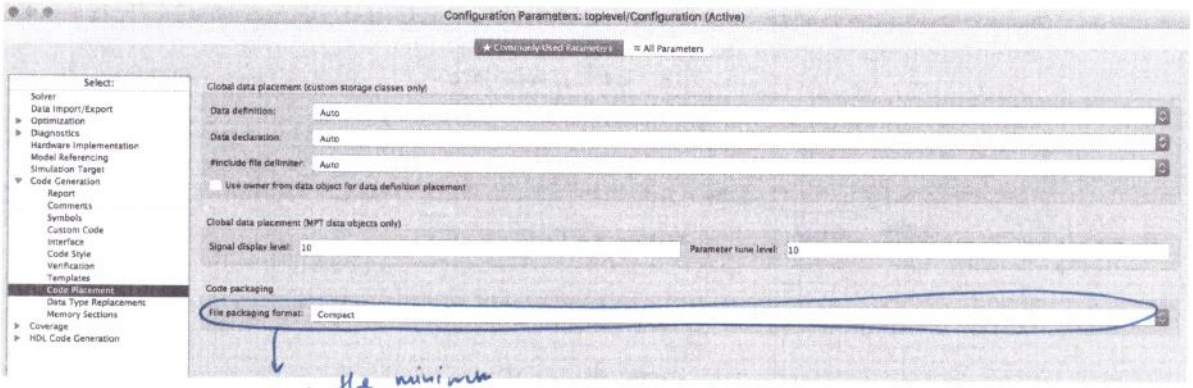


Hardware implementation



© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 31 di 92

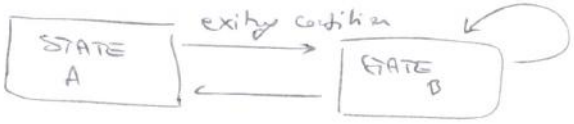
Code generation – code placement



18

Then, we use the STATE DESIGN to write our model

Model behavior is just like a state machine ASM



State Actions

en : Action to be done when entering the State

du : Action to be done when in the State

ex : Action to be done before leaving the State

04/21/201

SOFTWARE DESIGN GUIDELINES

aiming to make model more readable by people only ⇒ No faster model or more efficient simulation but BETTER UNDERSTANDING

GOOD PRACTICES

- Show eqs used in the model
- If there are specifications, document them.
- Use subsystems = little system with certain function ⇒ Entire model would be tidy and easily understandable
  - Specify IN/OUT with significant names
  - Show measures used
  - Show equation subsystem performs
- Naming conventions
  - No start or end with underscore

04/26/2017

How to represent Controller (Discrete Time) interfaced with Plant (Continious Time) ?

SOL 1 : Turn Plant in Discrete Time with small steps

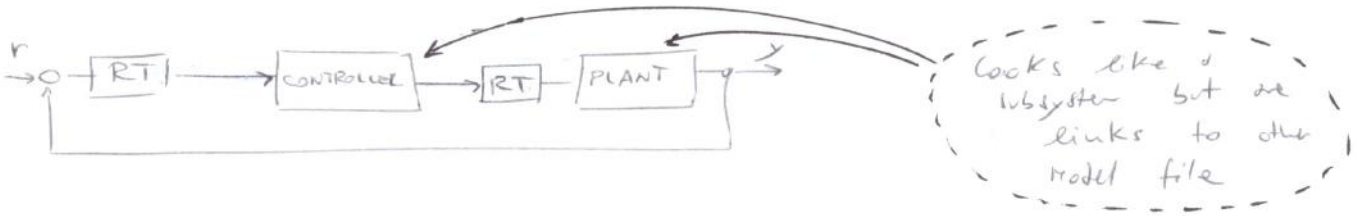
SOL 2 : Mixed Model ⇒ Controller simulated in Discrete
Plant simulated in Continious

Possible using Simulink Project!

SIMULINK PROJECT contains

- Controller Model File with $T_s = 1\ ms$ by default but can be changed ⚠
- Plant Model File

Harnesses = Controller + Plant



SOFTWARE TESTING

05/03/2017

In all application item is designed to provide functional safety even in hazardous events.

Item fails can be due

- HW Failure
- SW faults (bugs)

In both cases item must remain safe, nomatter what's the situation.

Achieved applying certain methods and countermeasures.

Countermeasures list for HW fault

- **RANGE CHECKS OF I/O DATA**

  Reading the values and compare if fix with given range. If not fix then error occurred
  ↓
  given by designer item

  Sensor are reading value of item's dynamics which is well-known by designer
  ↓↓
  Possible to implement a "critical view" and recognize no-sense values

- **PLAUSIBILITY CHECK**

  Compare different values, supposed to be very different.
  If they match then an error occours

- **DETECTION FOR DATA ERROR**

  Memory will deteriorate during time.
  To avoid time-changing data critical values are stored in different variables.
  Before use a critical value it's compared with its clone if there mismatch the error occured.

- **EXTERNAL MONITORING FACILITY**

  During software development of function's time is measured in different conditions.
  ↓↓
  execution time is know
  ↓↓
  function execution is measured
  if it's taking too long or too early
  an error occurred

  watchdogs used

Countermeasures for SW "faults"

Software is not affected by age issues.

Then software is "always" correct, no breaking software

⇓

Software infections are always the same and there are no changes

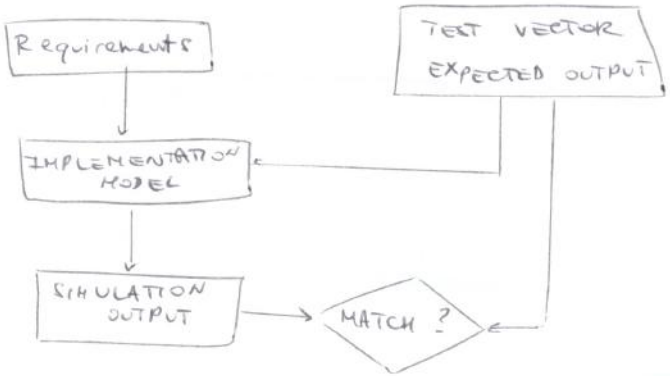
SW Fails = SW Bugs

Software is always the same but can be wrong

Good debugging needed
Model-Based + Code Generation needed

There is software testing = analyzing software item to detect differences between existing and required condition

flow-chart



NOTE:

Later bug is discovered much money are spent to fix it on the items

EX.

Cost bug in Design = 5

Cost bug in Delivery = 150

If values match ⇒ software ok

If values mismatch ⇒ Bugs in software

⇓

Debugging and Test Again

UNIT TESTING

Aiming to confirm the correct operation of each units

Based on tables specifing methods and testing settings

UNIT TESTING METHODS

(b) **REQUIREMENTS-BASED TEST** =

understanding which is the minimum amount of input to provide in order to obtain a good testing

ASIL

A	B	C	D
++	++	++	++

divided in

→ **Equivalence class partitioning** =

input domain divided in disjoint sub-domains called equivalence-classes

Test generated by selecting one input in each chosen

2 partitioning type

→ unidimensional partitioning = test one input at time other input neglected

→ multidimensional partitioning = partition considering cartesian product between inputs

Usually bugs occours at boundary

⇓

For better check it good to provide boundry value as input

⇓

Boundary value analysis =

selecting input at the boundary of each equivalence class

Ex. $99 \leq x \leq 999$ ⇒

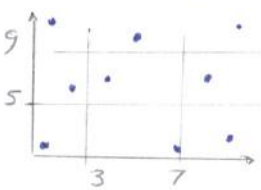
Testing for

$x = 98$, $x = 99$, $x = 100$

$x = 998$, $x = 999$, $x = 1000$

Ex. $3 \leq x \leq 7$, $5 \leq y \leq 9$

multidimensional



→ unidimensional

$x < 3$ $y = -$

$3 < x < 7$ $y = -$

$x > 7$ $y = -$

$x = -$ $y < 5$

$x = -$ $5 \leq y \leq 9$

$x = -$ $y > 9$

05/10/2017

(16) INTERFACE TEST ⟶ ASIL

A	B	C	D
++	++	++	++

testing if inputs are correct and if not verify that SW doesn't crash and works correctly anyway.

⇓

Checking SW behavior with different format → both correct ones and wrong ones.

Allows detect possible cases where output is not initialized

Ex.

```
int z;
if ((x>0) && (y>0)) {
    z = x;
}
return z;
```

If if branch not executed

⇓

z not initialized

⇓

z would be a random value

(1d) RESURCE USAGE TEST → ASIL

A	B	C	D
+	+	+	++

= testing CPU / memory usage by unit

requires

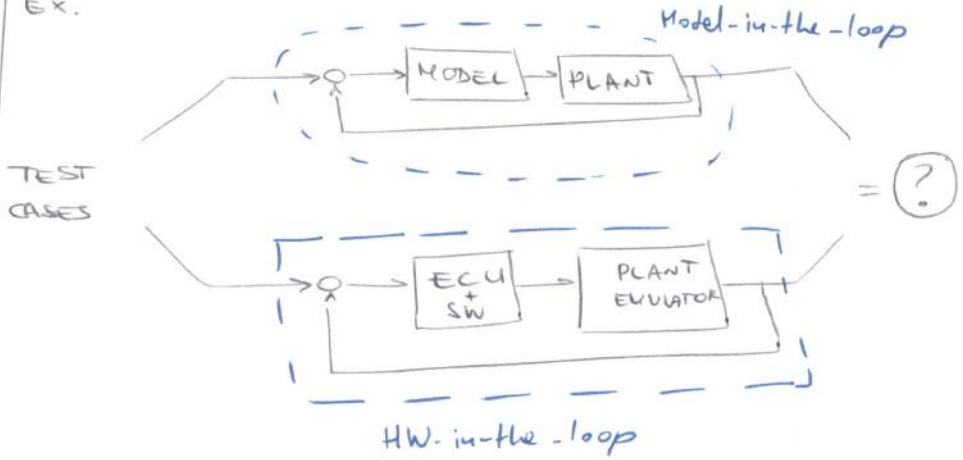
SW running on used CPU testing different outputs

(1e) BACK-TO-BACK COMPARISON TEST → ASIL

A	B	C	D
+	+	++	++

= testing outputs for models and outputs for SW by model-based design and verify that both of them return the same outputs.

EX.



Should be performed at each X-in-the-loop

SYSTEM TESTING

= SW put in the real application and real situations, and tested if behavior is the wanted/designed one

Method List

1a) **Hardware-in-the-loop** → ASIL A B C D; + + ++ ++

= testing the real hardware with a plant emulator

physical Test is:

real ECU that will be deployed to the item

Adapts the electrical values in order to present the same V, I, R that ECU will encounte in real-world.

EMULATOR running a real time plant model

1b) **ECU Network Test** → ASIL A B C D; ++ ++ ++ ++

= testing connection between ECU and sensors/actuators

The electronic system is on the body car and it is tested

Real HV bat can is just "filled" with electronics parts only

Test takes place in 1/6 where a mission is planned and a driver perform a drive on a simulator-like environment

SAFE

Electronics part (ECU + sensor + actuator + network)

1c) **TES vehicle**

= vehicle is tested on the road ⇒ All item/units are in place and tested in very first on-ground test

Human Pilot is used

Most risky part

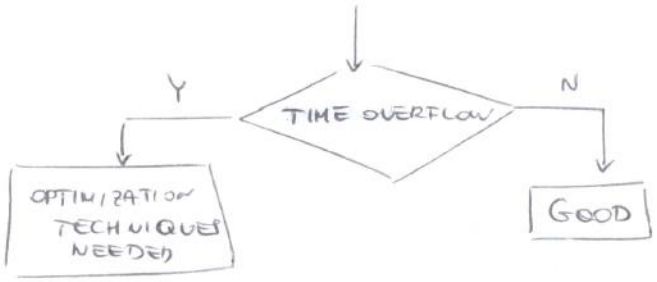
In this test all the issues should be known and already discovered since human boeing is envolved in these tests

If deadline is missed ⇒ simulation produced is no more valid

↓

Likely to loose plant control

After code generation, SW is tested



OPTIMIZATION TECHNIQUES = strategies to make SW faster

different

→ USE FASTER HW ⇒ use µP with fclk higher

↓

Not always possible due cost increase

BETTER USE OF RESOURCES

① Flash/RAM

② Interrupts

③ Cache memory vs Multiple Cores

④ HW accelerators

⑤ Fixed vs Floating Point arithmetic

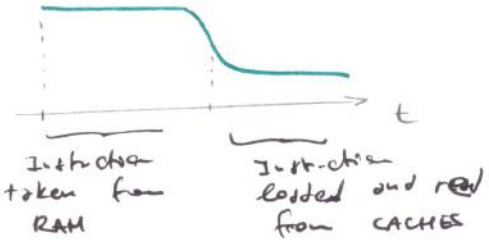
(3) CACHE MEMORY vs MULTIPLE CORES

In µP we have

Storing the program caches load with instruction from the RAM and the pass them to cores to be executed

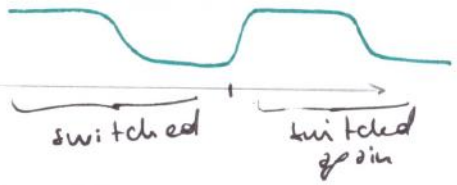
Usually caches monitors the most used instruction to speed up the system

caches faster the RAM



In multi-cores systems is frequently used the CORE switching = CORES change tasks

Slowing execution time ← Time spent moving instruction ← Cache's instruction must be moved in the CORES' CACHE



⇒ Time benefits are very limited with core switching

NO CORE SWITCHING ⇔ Better assign tasks to a specific core

↓

Caches need to be loaded once

↓

Time benefits are significant

EXAM MATERIAL

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 57 di 92

Class	S0	S1	S2	S3
Description	No injuries	Light and moderate injuries	Severe and life-threatening injuries (survival probable)	Life-threatening injuries (survival uncertain), fatal injuries

Class	C0	C1	C2	C3
Description	Controllable in general	Simply controllable	Normally controllable	Difficult to control or uncontrollable

Class	E0	E1	E2	E3	E4
Description	Incredible	Very low probability	Low probability	Medium probability	High probability

		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 59 di 92

STEP 4: REQUIRED FUNCTIONALITIES = what's needed to make the item works correctly

To perform a call connection required ⟹ • Phone network connectivity } Functionalities from the environment

Locate the crash site ⟹ • GPS signal from GPS ECU
• Airbag Deployed signal from Airbag ECU } Functionalities from other items

STEP 5: HAZARDS ANALYSIS = how components' fails can compromise the item's functionality

Write down all the item's elements and imagine what can go wrong with those specific elements

• µC ⟶ µC Failure ⇒ No layer able to run SW ⇓ No functionality wanted } HZ1

• CAN Network ⟶ CAN Failure ⇒ No data from Airbag and GPS ⇓ No wanted functionality } HZ1

• Modem
⟶ No calls when needed ⇒ No wanted functionality } HZ1
⟶ Calling when NOT needed ⇒ Not needed operation } HZ2

• Crash Detector ⟶ No crash detected ⇒ No wanted functionality } HZ1

STEP 7: ASIL TABLE

	HZ 1	HZ 2
OS 1	S = 3 E = 1 C = 3	
OS 2	S = 3 E = 0 ← (*1) E=0, difficult to be involved in a crash and having Airbag fault C = 3	
OS 3	S = 2 E = 1 C = 3	
OS 4	S = 2 E = 0 ← (*1) C = 3	
OS 5	S = 2 E = 1 C = 3	
OS 6	S = 3 E = 0 ← (*1) C = 3	
OS 7	S = 1 E = 1 C = 3	
OS 8	S = 2 E = 0 ← (*1) C = 3	
OS 9	HZ 1, OS 9 NOT CONSISTENT	S = 0 E = 4 C = 3

HAZARD NOT CONSISTENT WITH THE OPERATIONAL SITUATIONS

E=4, high probability to have a safe operation

No crash at all is highly probable, luckily!

Model based software design
Examples of exam questions

1. Describe the V-shaped development flow;
2. Describe the model-in-the-loop simulation concept;
3. Describe the software-in-the-loop simulation concept;
4. Describe the processor-in-the-loop simulation concept;
5. Describe the hardware-in-the-loop simulation concept;
6. Describe the difference between algorithm export and full-executable export;
7. Describe the difference between integration time and sampling time;
8. Describe the difference between continuous time solver and discrete time solver;
9. Describe which is the most appropriate solver for modeling for code generation;
10. Assuming a model is converted into the step function S() starting from a solver with integration time set to 15 ms, which of the following assertions makes the software behaving exactly as the model?
    a. The sw platform shall execute S() at least every 15 ms;
    b. The sw platform shall execute S() exactly every 15 ms;
    c. The sw platform shall execute S() at most every 15 ms;
11. Assuming a model is configured with integration time equal to 20 ms, and that the resulting step function S() requires 25 ms for running on the selected hardware platform. Can the obtained code behave exactly as the model it derives from?
12. Assuming a model is configured with integration time equal to 20 ms, and that the resulting step function S() requires 15 ms for running on the selected hardware platform. Can the obtained code behave exactly as the model it derives from?

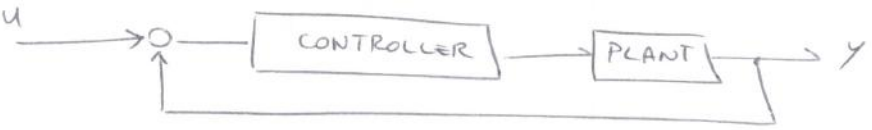
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 65 di 92

(2)

Model-in-the-loop simulation is a subpart of the V-shaped development flow.

It involves the first three steps: System Requirements, System Design, and Software Design.

In this part no hardware is involved (it is intended to study the plant + controller and their behaviors, with the highest precision possible in the simulation software (as Simulink)



These blocks are just a modelization of the designed items.

Output y considered applying different inputs u.

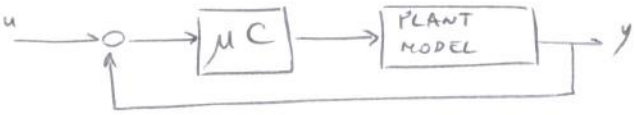
Since these blocks are models, this part is platform indipendent.

(No HW-SW considerations have been made yet)

④

Processor-in-the-loop is part of the V-shaped development flow, involving Software Integration and HW/SF Integration parts.

In this simulation a simulated plant interacts with the real µC, running the generated code.



In such way, it's possible to test the SW and HW combined.

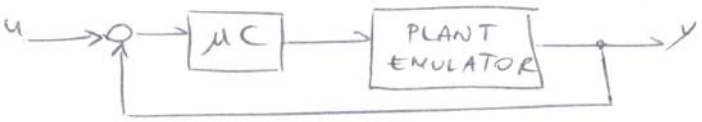
Plant is still a modelized entity.

---

⑤

Hardware-in-the-loop is the last simulation present in the V-shaped development flow.

In this simulation controller is formed by µC running the control code and the Plant Emulator: a very precise and accurate machine able to behave like a real-world plant.



In this simulation both blocks are considered real item so, there is no model involved anymore.
This is the closest simulation to reality possible.

Plant Emulator is very expensive to use then Hardware in the loop simulations are performed only when designers are sure about their project.

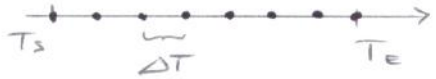
Hardware in the loop should be intended as a final check, just to be sure the item works as expected.

(8)

Continious Time - T_simulation is divided in ΔT steps where the program evaluates the system

⇓

Good Accuracy



Discrete Solver = Simulation takes place in t_{START} and t_{END} only

⇓

Faster but less accurate

---

(9)

Most appropriate solver for code-generation is the DISCRETE SOLVER because the ECU, μC or SoC on the deployment hasn't high resources to run a continious time code.

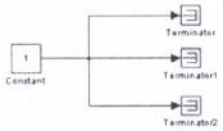
Also, by definition, ECU samples I/O with ADC/DAC discretizing the signals.

---

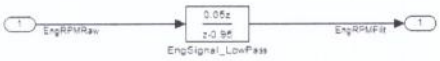
(10)

Model based software design
Examples of exam questions

1. Describe the purpose of modeling rules such as MAAB;
2. Draw the Simulink model for the following equation $F = m \times a$, where F is the output, and m and a are the inputs;
3. Draw the Simulink model for the following equation $a = \frac{dv}{dt}$, where a is the output, and v is the input;
4. Is the following Simulink schematic consistent with the MAAB modeling guidelines? Why yes/no?



5. Is the following Simulink schematic consistent with the MAAB modeling guidelines? Why yes/no?



6. Describe why code optimization may be needed after code generation from a Simulink model;
7. Describe at least one code-optimization technique;
8. Describe why moving code to RAM from Flash can improve code execution speed;
9. Describe how interrupts can affect code execution speed;
10. Describe how cache memory can affect code execution speed;
11. Describe in which case fixed-point scaling can be useful to improve code execution speed;
12. Describe which can be the drawback of using fixed-point scaling;
13. Assuming we need to multiply 2.25 and 6.5 on a CPU which does not have floating point hardware, which would be the best scaling factor between 2 and 4 as far as representation accuracy is concerned?
14. Describe at least one method to measure code execution time;
15. Describe the code instrumentation method to measure code execution time, and illustrate its limitations;
16. Describe the purpose of the static code analysis techniques;
17. Illustrate the potential programming problem related to implicit cast usage;
18. Illustrate the potential programming problem when using recursive functions;
19. Describe the potential programming problem when not using coding rules;
20. Given the following code fragment, is there any programming problem which can be detected using static code analysis? If any, how the programming problem would be classified, error or warning?

```
1:  uint8_T alfa;
2:
3:  void foo( uint16_T *result )
4:  {
5:    *result = 25;
6:  }
7:
8:  main( void )
9:  {
10:   foo( &alfa );
11: }
```

bits mismatch

Program should be classified as an error

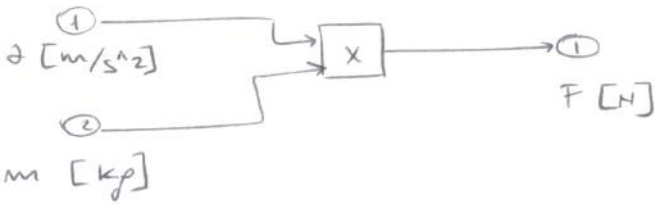
(1)

MAAB purpose is providing guidelines in order to make the design/project's schematics understandable by other people not directly involved in the project development

It is NOT a way to make project more efficient or faster in the execution, it is just a set of guidelines intended to make the project understandable by almost everyone

---

(2)



---

(3)

---

(4)

Nope. Not consistent due to units of measurements missing, triple termination of output (useless redundancy) No significant names.

---

(5)

Nope. Missing units of measurements.

3 - CACHE MEMORY IN Multiple Cores

Assign to a task a fixed core, in this way the latency time for cache-2-cache is reduced.
No need to unload and unload a cache.

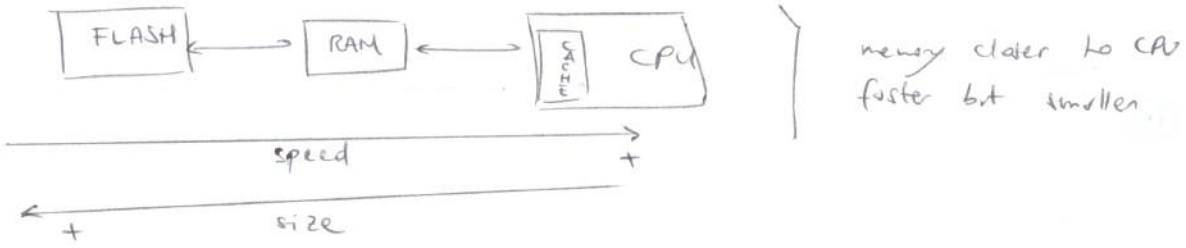
4 - HW ACCELERATORS

Use a simpler model, less precise but still valid, then the execution time is lower due to less instruction to run

---

(8)

In a computer architecture, memories are distributed and they have different functions



Flash memory is where code is stored and in the scheme it is the biggest one but slowest one.
When a function has to be executed is loaded to RAM from Flash, this action takes time due to low speed of Flash.
Instead, if a function is stored already in RAM then it is ready to be executed, no time wasted in loading it to RAM.

(13)

CASE SCALING 2

$2.25 \cdot 2 = 4.5$
$6.5 \cdot 2 = 13$ $\Rightarrow$ 4, 13

$S = 4 \times 13 = 52$

$R = \frac{52}{2 \cdot 2} = 13$

CASE SCALING 4

$2.25 \cdot 4 = 9$
$6.5 \cdot 4 = 26$

$S = 26 \cdot 9 = 234$

$R = \frac{234}{4 \cdot 4} = 14.625$

Since

$2.25 \cdot 6.5 = 14.625$ $\Rightarrow$ Best solution is SCALING with 4

---

(14)

A way to measure the execution time is running a program embedded in the O.S. equiped with a tool-box taking times of the running process.

Can be, however, imprecise due to multiple tasks running.

---

(15)

Modify the code running on the target in order to produce additional signals notifing when the execution has started and done.

Just few line of code are added to reach this goal

An example is using a GPIO pin. Here how:

```
set GPIO pin to 0
set GPIO pin to 1
    Running Code
set GPIO pin to 0
```

$\Rightarrow$ A square wave is generated if looking the pin on an oscilloscope

However the GPIO is not precise due to pins activation / configuration time.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 79 di 92

(18)

When recursion is involved the function results are stored in the stack memory, a part of the RAM.
If the recursion is too long it possible that the stack gets filled and, for keeping recursion going, the new values are going to be saved outside the stack, where other variables are saved. This leads to a data loss.
Usually when stack overflow occurs the processes are stuck completely.

---

(19)

If coding rules are not considered there is an high ~~pro~~ risk of making conceptual mistakes.
These errors are not recognized by IDE because the code language's syntax is correct but, maybe, the code is interpreted in a different way than the one thought by the designer.

---

(20)

Take a look on the questions' list

25. Compute a test that allows to perform the interface test for the following function:

```
00: int foo( int a, int b )
01: {
02:   int w, q, z;
03:
04:   w = a*K1;
05:   q = w*K2;
06:   if( b > 0 )
07:           z = a+q*K3;
08:
09:   return( z );
10: }
```

26. Describe fault injection test;
27. Describe resource usage test;
28. Describe back-to-back test;

⑤

When a model based software is obtained before the deployment and during the development all its functions are time measured: measuring the executing time.

The maximum execution time is known by testing and is possible to implement watchdogs.

Watchdogs monitor the function's execution time and, if a function is taking too much time to terminate then is likely that an error occurred.

---

⑥

In this tecnique µP provide to a watchdog the informations about the taken branches of the program.

Watchdog compare the branches taken with the SW structure and verify if both are consistent with expected and actual execution time.

---

⑦

Diverse Software design exploits redundancy.

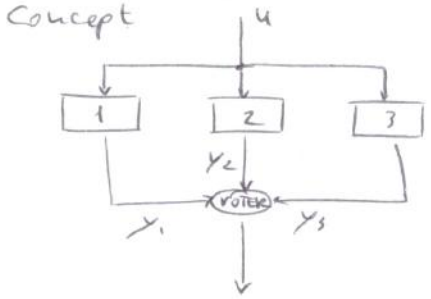
Very same task provided by different systems (different SW, different HW and different development teams).

These systems are design to achive the very same task.

All system are executed and the results are compared

Results match ⇒ ok

Results mismatch ⇒ Error occured



(11)

Unit testing is focused on apply stimuli to a single unit and check if the outputs are the expected ones.
Its aim is to confirm the correct operation of the Unit Under Test.

Integration Testing is a procedure to check if the units are "communicating" to each others in the designed way.

System Testing is when all the units are connected to the very system that would be actually deployed.
Here the entire system's behavior is checked!

(12)

This procedure is used to select the input to provide in the test phase.
Input domain is divided in equivalence-classes and the input provided for testing is one for each classes.
There are 2 partitioning type

- Multi dimensional = considering products between inputs
- Unidimensional = test one input at time, other input neglected

(13)

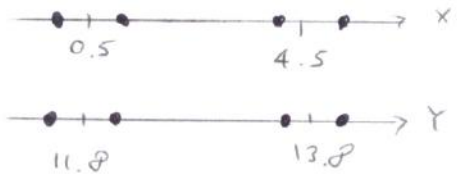
Inputs are selected at the class boundaries.
This is performed because often bugs occur at the boundary.

(14)

Unidimensional Partitioning = in case of multiple inputs only one input is tested at a time

Multidimensional Partitioning = partition considering products between inputs of different classes.

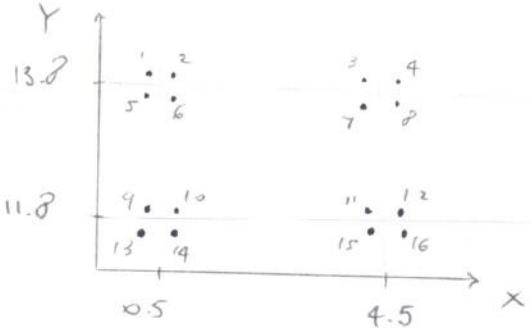
(17)



$x = 0.49$
$x = 0.51$
$x = 4.4$
$x = 4.6$
} $Y = -$

$Y = 11.7$
$Y = 11.9$
$Y = 13.7$
$Y = 17.9$
} $X = -$

(18)



1 $x = 0.49$ $Y = 13.9$
2 $x = 0.51$ $Y = 13.9$
5 $x = 0.49$ $Y = 13.7$
6 $x = 0.51$ $Y = 13.7$

9 $x = 0.49$ $Y = 11.9$
10 $x = 0.51$ $Y = 11.7$
13 $x = 0.49$ $Y = 11.9$
14 $x = 0.51$ $Y = 11.7$

3 $x = 4.4$ $Y = 13.9$
4 $x = 4.6$ $Y = 13.9$
5 $x = 4.4$ $Y = 13.7$
7 $x = 4.6$ $Y = 13.7$

11 $x = 4.4$ $Y = 11.9$
12 $x = 4.6$ $Y = 11.7$
15 $x = 4.4$ $Y = 11.9$
16 $x = 4.6$ $Y = 11.7$

(19)

Given a program to be tested a set of input has to be provided.

Statement coverage is the percentage of statement executed when the test use a certain input set.

(25)

b need to $b > 0$ otherwise z is not initialized.

$\Downarrow$

$b = 0,\ b = 1$

---

(26) Fault injection is a tecnique where the unit has to deal with a faulty in the pocess and the behavior is studied.

It's a way to emulate a faulty scenario and evaluate what kind of faults and failure can podice in order to study some error handling strategies.

In case of an unit with some redundancy then also redundancy is tested in order to guarantee a correct function even with faulty parts.

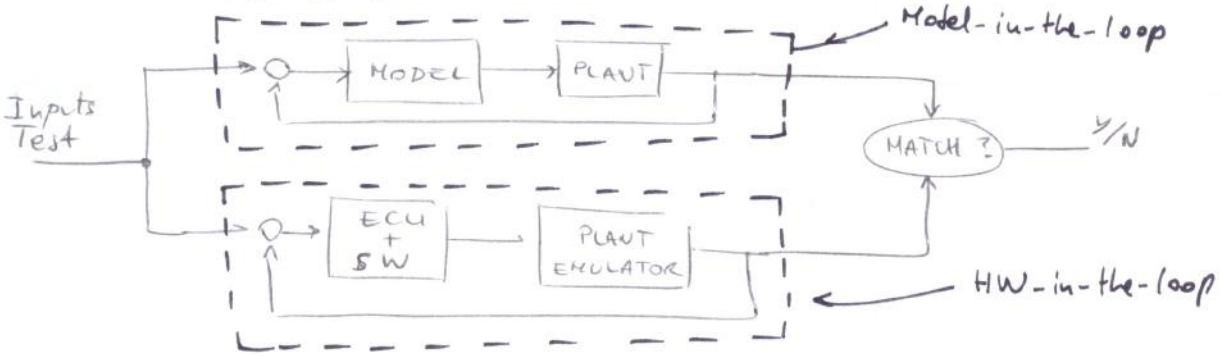
---

(27) In this test SW runs on the designed HW in order to study how much memory/resurces/execution time the SW takes in the application HW. All possible outputs are tested

---

(28) In this test the model-in-the-loop and hardware-in-the-loop are run in parallel and the outputs are compared to check the consistency between real item and intended one.

The block is:



Since Model-in-the-loop is the very first simulation and HW-in-the loop is the last one, if the outputs match then the HW + SW in the item are a very good approximation of the wanted control system, designed using models and simulation programs.