ویدیو 2

Characteristic of ES

1) Dependability → قابلیت اعتماد

↳ Reliability , Maintainability , Availability , Safety , Security

قابلیت اطمینان، قابلیت نگهداری، دسترسی‌پذیری، ایمنی، امنیت

* مثلاً Pc ها نیازمند dependability نیستن چون اگه یهو down شد و خاموش شد یه بازی قطع شده
فاجعه‌ای رخ نمی‌ده!

احتمال $R(t) = 0.99999$ → DO-178B استاندارد

۱تا۵ قابلیت اطمینان

2) Energy Efficiency : چون مبتنی بر باطری هستش و مثلاً سنسور تشخیص ریزش بهمن که توی جای سختی هستش تعویض سخت و چون صنعت باطری کننده

↳ $\int P dt$

* کاهش ولتاژ منجر به کاهش انرژی می‌شود ← سرعت می‌آید پایین چون فرکانس می‌آید پایین

↳ کم شدن نویز مارجین ← احتمال رخداد ارور بیشتر می‌شد ← $R(t)\downarrow$ ← dependibility کم میشه.

3) Performance ⟷ 4) Real-time Constraints

avg case مهم! ← در معماری کامپیوتر

حدود ۹۰٪ ES ها

↳ For real-time systems, right answers arriving too late are wrong.

* در ES با worst case کار داریم!

سامانه های بی درنگ : سامانه هایی هستند که برای اینکه نتیجه‌ای که تولید می‌کنند صحیح باشد، نه تنها باید محاسباتشان سالم باشد بلکه در زمان درستی هم تولید شوند.

سامانه های بی درنگ :

1) Hard - Real time → miss کردن ددلاین فاجعه

2) Soft Real time → MP3 → فاجعه رخ نمیده ولی باعث کاهش کیفیت می‌شه

3) Firm Real time → هیچ دیتایی نداره

5) weight Efficient , Cost Efficient , Code-size efficient

↳ باطری ها وزن زیادی دارند و در کل خیلی کوچیک هستن

↳ چون تعداد زیادی می‌خوایم

↳ پیچیدگی که

محدودیت Ram ← خود Ram هم چون دسترسی سریع تر به همراه شانس هست با dependability در تناقض هستش.

ویدیو 3

System A	احتمال	زمان
	90%	1 ثانیه
	10%	10 ثانیه

Avg : 1.9 S

System B	احتمال	زمان
	50%	0.5s
	50%	8s

Avg : 4.25

صعد زمانی مهم است.

* پس سیستم A بهتر از سیستم B است
اما اگر ددلاین زمانی ۹ ثانیه داشته باشیم
باید از B استفاده کنیم تا همیشه
درست کار کند.

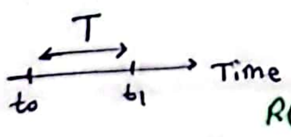
6) Dedicated towards a certain App. خاص منظوره بودن → Minimize resources, Maximize robustness

↳ راه های کلیدی : شبیه سازی کردن + کم کردن ابزارهای دیجیتال + ساده کردن پیاده سازی + خاص منظوره پیاده کردن

↳ جلوگیری از ایراد

7) Dedicated user interface → no mouse, no keyboard or screen.

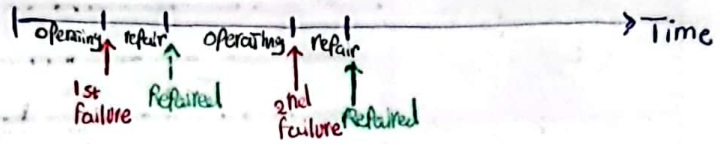
*1) Reliability : The reliability of a system, $R(t)$, is a function of time, defined as the conditional probability that the system will perform correctly throughout the interval $[t_0, t_1]$, given that the system was performing correctly at time t_0



* اگر خیلی repair سخت باشد ← $T = 10$ سال

* برای پرواز ← $R(t) \geq 0.99999$, $T =$ چند ساعت

2) **Availability** : The availability, A(t) is a function of time, defined as the probability that system is operating correctly and is available to perform its function at the instant of time t.



* The availability depends not only on how frequent the system becomes inoperable. but also on how quickly it can be repaired.

3) **maintainability** : M(t), is the probability that a failed system will be restored to an operational state whithin a specified period of time, t.

↳ * The restoration process includes :
1) Locating the problem
2) Physically repairing the system
3) Bringing the system back to its operational conditions.

4) **Safety** : S(t), is the probability that a system will either perform its functions correctly, or will dis continue its function in a manner that does not distrupt the operation of other systems or compromise the safety of any people associated with the system. → Safety is a measure of the fail-safe techinques of a system

5) **Security** : The prevention of
* Unauthorized access of information and/or
* " handling " " " "
supporting the authorized access of information.

* The study of embedded systems is symply a combination of well-known areas such as :
* Dependability
* Real-time systems
* Low-power design
* etc.

یعنی همیشه یک تک این موضوعات رو جدا در نظر گرفت!

inter disciplinary area of study

interplay of Design objectives

Design objectives :
* Fault tolerance (Dependability)
* Energy Efficiency
* Real time
* Cost efficient

این ها ممکن در تناقض باشن

* Fault tolerance requires some types of redundancy and leads to energy consumption.

دیدید 4 Typically ES are reactive systems.

"A reactive system is continual interaction with its enviroment and executes at a pace determined by that enviroment."

* Reactive Systems = Event-Based Systems
* The Traditional paradigms of Programming (i.e. model of computable functions) are inappropriate.
↳ Von neumann Paradigm
↳ Sequential computing

Automata-based Programming paradigm ⟷ reactive systems.

Von-neuman
عدد
Physical
مقداری جمع وضرب وتقسیم
shared memory

Automata
event
logic ترتیبی
مقایسه‌ای

4 مشخص مهم برای دسته‌بندی پارادایم چیست؟
1) ورودی خروجی ها چیست؟
2) کنترل و نگاه ما به زمان به چه صورت است؟
3) Operation ها به چه صورت هستند؟
4) ارتباطات به چه صورت هستند؟

ویدیو 5 Histerssis هیسترسیس
یعنی عملکردی که باعث می‌شه همه‌چی بروز پیدا کنه.
یک بازه‌ای که دما باید تغییراتش زیاد باشه تا event ها عوض بشه!

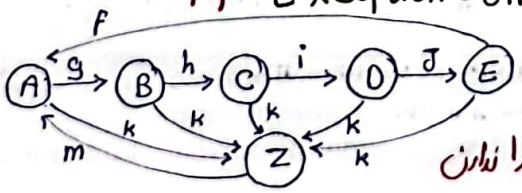
* طراحی سلسله مراتبی
↳ خطایی بیشتر + طراحی راحت‌تر
↳ super state!

* verification → ساده‌است
* Automatic Code Generation

* functional → عملکردی + ورودی خروجی
* non-functional
↳ توان عمومی + سادگی + latency

تعریف Specification: 1. تعریف اول: بیان صورت مسئله طراحی
2. تعریف دوم: ارائه راه حل سطح بالا.

ویژگی زبان های ES specification:
1) Hierarchy → Behavioral: Super-States
↳ Structural:
2) Timing Behavior → Delay
↳ Cause and effect Relationship
3) State-oriented behavior
automata provide a good mechanism for modeling reactive systems
4) Event handling → The reactive nature of ES
Caused by components of the system ← event های داخلی
Caused by environment ← event های خارجی
هم
هم
Hardware/Software Co-Design
5) Support for efficient implementation
6) Support for dependable system design
↳ Uncambigious semantics
↳ Faciliate formal verification
7) Exception-oriented behaviour



state diagram with exception k
این که هر کدوم یه استیت state جدا دارن خیلی خوبه.

* ویژگی های ابتدایی: در ابتدا کسی باید اونرا بنویسه requires human inteligence
نباید به زبان طبیعی باشه باید یک ساختار مشخص داشته باشه.
چون هم باید کامل باشه و هم بدون تناقض
↳ it should be possible to derive implementations from the specification in systematic way.

8) Concurrency 9) Synchronization & communication

10) Presence of Programming elements → Usual Programming languages have Proven to be a convinient means of expressing computations.
State Diagrams do not meet this req.

11) Eexcutability
→ simulation → Design verification

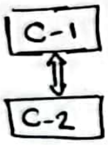
12) Readability and Flexibility → Readable by human
→ small changes of the system → Small changes of the specification.

13) Support for non-standard I/O devices
→ to describe inputs and outputs.

14) Non-functional Properties
Reliability
Size
Power consumption

15) Appropriate model of computation.
← مدل فون نیومن مدل مناسبی نیست.

Models of Computation * C, C++ برای مدل ون نیومن مناسب اند. ویدیو 6
→ Components and an execution model for computations for each component.
→ Communications model for exchange of information between components

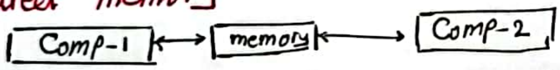


* share recourse → منجر به race
→ wait وایجاد صف → تأخیر
→ dead lock

* Hirarchial

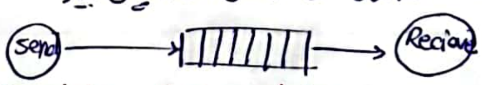
بعد این تبدیل
توصیف پذیری آسون تر! دقت طراحی کمتر می‌شود

models of Communication
→ 1) shared memory



Variables accessible to several components/tasks.
Model mostly restricted to local systems.

2) Message Passing → a) non-blocking/asynchronous message Passing
Sender has not to wait until message has arrived.
→ مشکلی که ممکنه پیش بیاد: buffer overflow



b) Blocking / Synchronous message Passing
Sender will wait untill reciever has recieved message

این مشکل buffer رو نداره
ولی کند تره!

c) Extended rendezvous, remote invocation
* The sender is allowed to continue only after an acknoledgment has been recieved from the recipient.
* The recipient does not have to send this acknowledge immediately after reciving the message but can do some Preliminary checking before actually sending the acknowledge.

State chart
→ is a language
CFSM MoC
(comminucating finite state machine Model of Computation)
Communication → shared memory
Deterministic FSM
← برای افزایش dependability

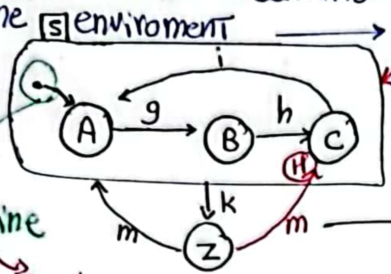
ویدیو 7

OR-SUPER-STATE

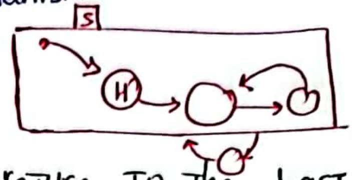
The FSM can only be in one of the sub-states of super state S at any time

Design Modularity طراحی ساخت یافته

There are 2 mechanism to hide the internal structure of super-states from the enviroment
1) Default state Mechanism
2) History mechanism



default state machine
خودش یه استیت نشده ها!



History mechanism → return to the last sub-state that was active before the super-state was left.

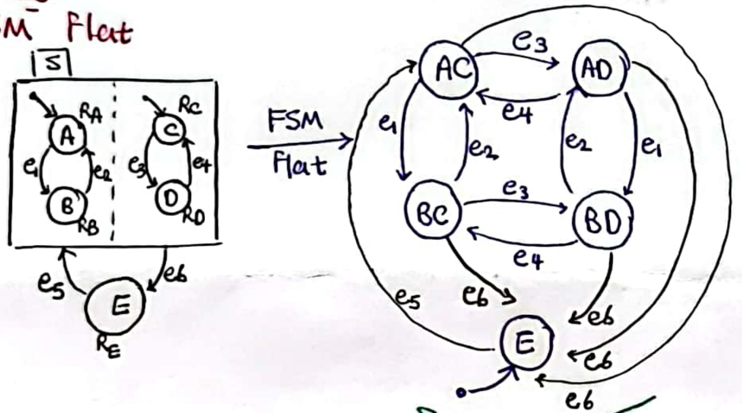
* البته هنوزم default داره برای اولین باری که میاد توی S

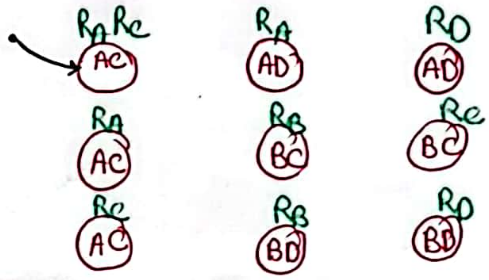
AND-SUPER-STATE

↳ برای concurrency استفاده می شه!

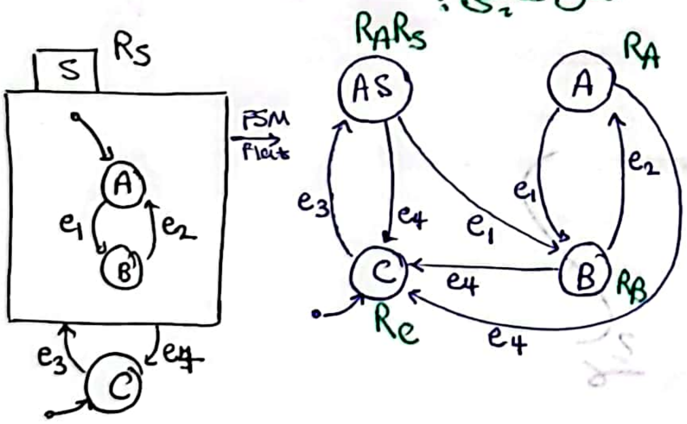
* Mixed mealy and moore are possible
Moore → ریکشن توی state ها (node)
Mealy → ریکشن روی یال ها (arc)

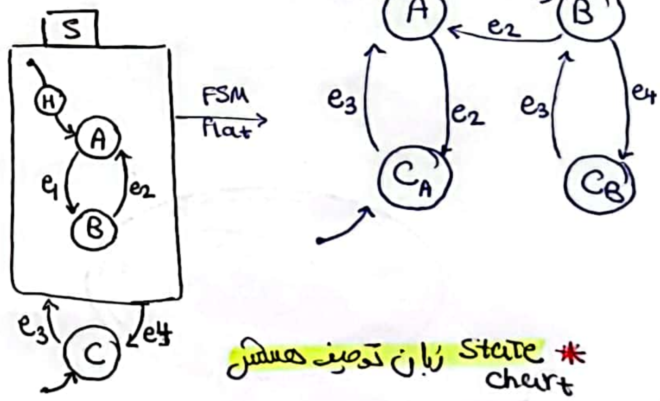
تمرین
FSM Flat





اکشن ها چی؟





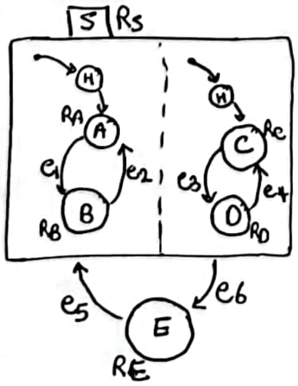
* State chart زبان توصیف همشون
برای پیاده سازی باید تبدیل به FSM شود!

* هم روندی به معنای موازات نیست
و با یک پردازنده هم می توان هم روندی داشت!

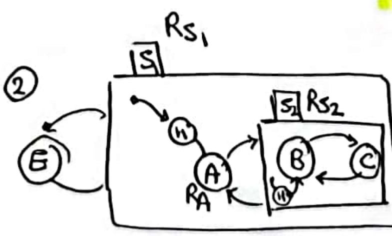
* هم روندی یعنی ترتیب انجام معنی ندارد!

. تمرین

①



②



LProc → یه همچین سوپراستیتی که فقط اسمش توشه یعنی پیاده سازی اش یه جا دیگه است.

* وقتی یه super state تموم می‌شه done می‌ده! **اگر طی روش هیچی نبود بذارید همون done ه!**

* چرا نباید وارد یک استیت درون super state بشیم؟ encapsulation چون مجرا باشن بر پیاده‌سازی super state کاری نداشته باشیم

Timer

توی state chart ، timer داریم چون عموماً سیستم‌های نهفته جزء سیستم‌های بی‌درنگ هستند!

اگر تو 20ms ، اینجا اتفاق افتاد می‌ریم چپ — اگر تو 20ms هیچ اتفاقی نیفته — الان اینجا اگر a بیاد میس شده

* **تایمرها می‌توانند ریکیشن داشته باشند**

همون state باشن هم super state

* استیت های dead ، wait state هستند منتظر واهستیم که یک ورودی بیاید.

Edge Lables

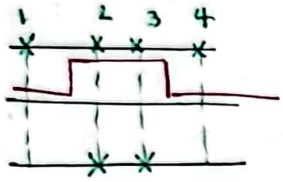
Event [condition] / Reaction

a proper reaction may depend on both the system state and the event that has happend

مثل mealy

On-key / On := 1 بدون شرط

Off-key [Battery = charged] / on := 1 شرط!



e1 رو با v1 فیلتر می‌کنیم. فقط 2,3 رو می‌ده!

Execution of State chart → Atomic Execution

Simulation

VHDL, Verilog ← هم روندی به واسطه بدون Atomic!

Statechart ← مدل قابل اجرا:
← زبان توصیف قابل اجرا

* Shadow and Main Variables
* Simulatain of Concurrency

ps , ns → next states : Shadow

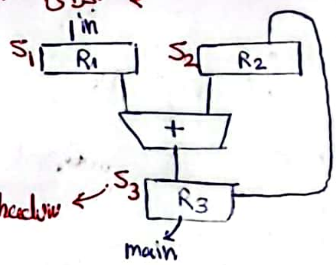
Present state : Main var.

Evaluation Phase

اجرای اگر مثلاً 2 تا داره: 1)

از متغیرهای اصلی استفاده می‌کنیم و می‌ریزیم توی shadow

مثال:



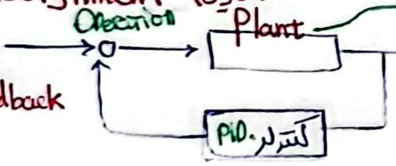

```
while (1) {
  Evaluation Phase
    S1 = in;
    S2 = R3;
    S3 = R1 + R2;
  Assignment Phase
    R1 = S1;
    R2 = S2;
    R3 = S3;
}
```

توی phase ترتیب جابجایی مهم نیست

Assignment phase

2) حالا shadow هارو به main ها می‌ریزیم

Assignment (تمرین)



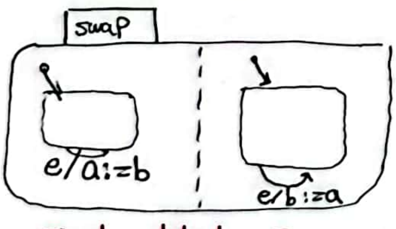
→ Incubator Temprature Control

هم در پارادایم معادلات دیفرانسیل

هم در state chart

* برتری state chart طراحی دیجیتال به معادلات دیفرانسیل آنالوگ چون توی پارادایم معادلات دیفرانسیل خیلی نوسان دارد!

Mutually Dependent Assignments



/a:=1 ; b:=0

① $a_1 = a$
$b_1 = b$

2 $a = b_1$
$b = a_1$

اگر این اجرای atomic نبود

برای اجرای همروند وارد critical section می‌شیم و داستان می‌شد!

Embedded System Hardware

Embedded system hardware is frequently used in a loop ویدیو ۹ "hardware in a loop"

Cyber Physical Systems

عموماً ES ، micro processor هستش

معمولاً پارادایم معادلات دیفرانسیل

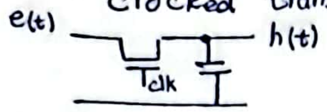
H/D Converter Sensors generate signals → a signal S is a mapping from time domain D_T to a value domain D_V : $S: D_T \rightarrow D_V$ (time → value)

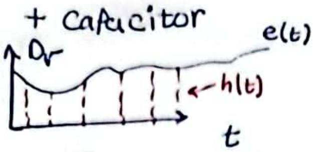
Discretization of time

Digital computers require discrete sequence of Physical values.

Sample and hold circuits → Discrete time Domain

↳ clocked transistor + Capacitor





e(t) is mapping $\mathbb{R} \rightarrow \mathbb{R}$
h(t) is a sequence of values or a mapping $\mathbb{Z} \rightarrow \mathbb{R}$

Discretization of values : A/D converters

Digital Computers require digital form of Physical values.

$S : D_T \rightarrow D_V$ Districte value Domain

information Processing

↳ cps & ES must be efficient : Code size efficient, Run-time efficient, Weight ", Cost ", Energy "

$$E = \int P(t)\, dt$$

* Minimizing Power consumption توسل کیر ← باتری ← * Minimizing energy Consumption

↳ design of the power supply & regulators

dimensioning of interconnect, short tem cooling

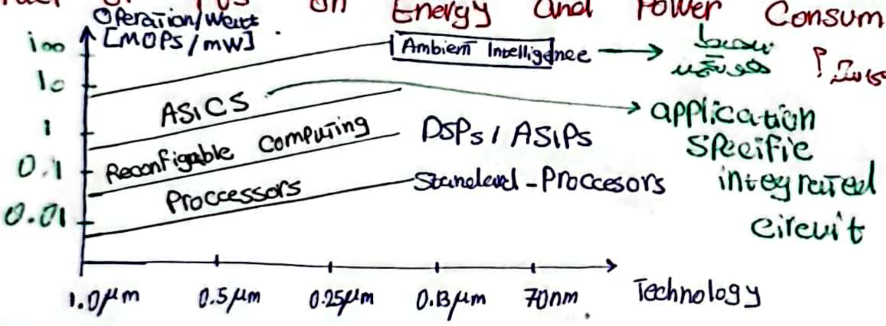
Minimizing energy Consumption:
- restricted availibity of energy
- cooling : high costs, limited space
- thermal effects
- dependability, long lifetime

* هم به پاور توجه کنیم هم انرژی

مثلاً از فن استفاده می‌کنی چون مکانیکی‌ه reliability اش کمتره سایزش بزرگتره

با تکنیک های نرم افزاری می‌توان توان و انرژی را کاهش داد.

Impact of PUs on Energy and Power Consumption



* چرا توان مصرفی با پیشرفت تکنولوژی کمتر می‌شه؟

چون خازن ها کوچکتر می‌شوند.

Embedded Processors vs. PCs → ep do not need to be instruction set compatible with PCs

Efficieny → Energy efficient, Code-size effic, Run-time effic

} تو ec هست، تو PC ها نیست! چون reactive هستند!

MicroProcessor vs. Micro Controller

off-chip | On-chip

90٪ بازار هست!

ASIP, FPGA دقیقاش

قابل پیشبینی است → Cache ندارد

پایپلاین ندارد

} → predictable on run time

1) Lower cost One Part replaces many Parts

2) More reliable Fewer Packages, Fewer interconnects

3) Better Performance System Components are Optimized for their enviroment. Singals Can Stay on The chip

worst-time executation → زو بهتر می‌کنه بلند } بر خاطر پیش بینی پذیری

dark silicon

↳ به علت تراکم تعداد ترانزیستورها از ۱۰۰ درصد ممکن نیست ما بطور همزمان استفاده کرد!

تمرین !! توان مصرفی dram از Sram بالاتره نیاز به رفرش دارد ولی جای خیلی کمتری مصرف می‌کنند.
↙ سرعت بالاتر

* در CPU افزایش سرعت خیلی‌خیلی بیشتر از پیشرفت در صنعت حافظه است.

main memory ← DRAM و Chache ← SRAM
↙ سرعت بالاتر

* در سیستم‌های نهفته از Cache استفاده نمی‌کنیم چون Predictibility avg time رو بهتر می‌کند ما دنبال worst case هستیم. ← dynamic Slack

* S-Ram ها جزء اصلی on-chip باشند.

* با افزایش سایز حافظه هم توان و هم تأخیر خطی افزایش می‌یابد.

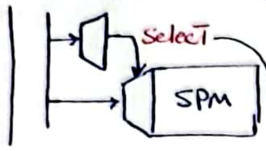
Scratch Pad Memory

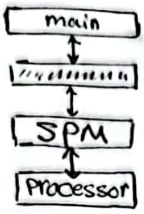
SPM Vs. Cache

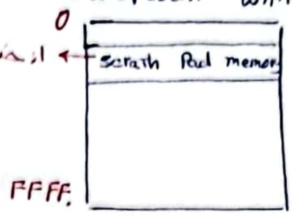
The only difference is that Caches are transparent while SPMs are not.

SPM is a small, physically seperate memory mapped into the address space.

هم سرعت هم توان مصرفی







بصورت نرم افزاری

selection is by an appropriate address decoder.

* چرا انرژی مصرفی Cache از SPM بیشتره؟
Cache ← کنترل یونیت داره
چون Predictability هست!
زاویه برنامه‌نویس عیان ← در آینده می‌دانیم که چه اتفاقی قراره بیفته پس چیزایی که می‌خواهیم رو توش می‌نویسیم

Chache بر اساس سابقه می‌نویسه
SPM بر اساس آینده!

SPM Vs. Cache

more Power efficient → Cache چون Controller hardware دارن.

و از دید برنامه‌نویسی Cache ساده‌تره!

Embedded System Software

↳ Increasing design complexity + Stringent time-to-market requirments
↳ Reuse of components — Reuse requires knowledge from previous design to be made available in the form of intellectual Property. → IP
از صفر طراحی نکنیم.

HW
operating systems
middle ware (communications, databases)

اگر scheduler داشته باشیم OS نمی‌خواهیم ←

↳ OSs Configurability: No overhead for unused functions tolerated, no single OS fits all needs. → Configurability needed.

- Object Orientation → کلاس‌بندی ماژول‌ها → module مجزا
- Aspect Oriented → Aspect های مجزا!
- Conditional → command های خاص
- Advanced compile-time evaluation useful → دیتاهای STATIC
- بازچیکره بندی
- Linker Time optimization: بدون فانکشن‌های اضافی

Dynamic data might be replaced by static data.

* Disks and network handled by Tasks

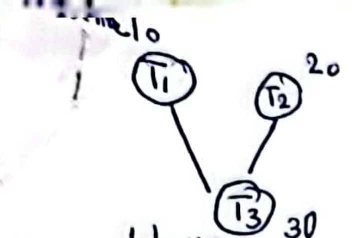
↲ تسک‌ها هندلشون می‌کنن به جای driver ها!

* Protection is optional → همیشه واجب نیست → ES typically designed for a single purpose.
↳ untested programs rarely loaded, SW considered reliable

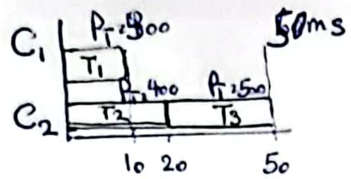
* Interrupts not restricted to OS

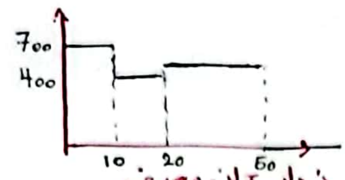
* مشخص RTOS ها — fault telerance هستش!

* Real-time Capability
↳ Predictibility
↳ Managing time



	توان دینامیک	توان استاتیک
T_1	200	100
T_2	300	100
T_3	400	100





نمودار توان مصرفی با DPM

DPM ← خاموش

$P_{avg}\, C_1 = 1/6 \times 300 + 5/6 \times 0$

نزدیم DPM → $P_{S_{DPM}}$ 0

$P_{avg}\, C_2 = 3/6 \times 300 + 2/6 \times 400 \times 1/6$

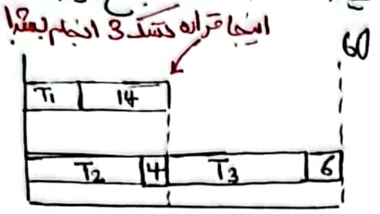
نزدیم DPM → P_S

$P_{avg}\, TOTAL = P_{avg}\, C_1 + P_{avg}\, C_2$

=

* توی کل هسته توان های avg رو باید حساب کنیم برای چند هسته باید جمع کنیم.

DVS :



$P_1 = \frac{10}{24} = 5/12$

$P_2 = \frac{20}{24} = 5/6$ (20+4) $\quad P_3 = \frac{30}{36} = 5/6$ (30+6)

اینکه slack time رو به اندازه execute ها تقسیم می‌کنیم به heuristic میگن

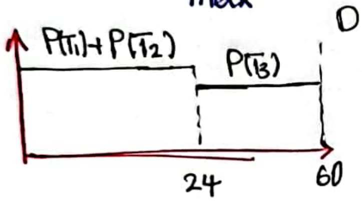
$P_D = \rho^3 P_{D_{max}}$

$P_S = \rho\, P_{S_{max}}$

$\longrightarrow P_{جدید} : \rho^3 P_D + \rho P_S$

تا جایی که می‌شه DVS اول

DVS : توان رو کم می‌کنه ⟶ DVS می‌زنیم بعد DPM



$P_{avg}\, C_1 : \frac{24}{60} \times P(T_1) + \frac{36}{60} \times P_{static}$ $\quad P_{avg}\, C_2 : \frac{24}{60} \times P(T_2) + \frac{36}{60} P(T_3)$

می‌تونیم 0 بذاریم یعنی DPM

$P_{avg}\, TOTAL = P_{avg}\, C_1 + P_{avg}\, C_2$

اگر DVS بزنیم می‌تونیم ρP_S بزنیم یا P_S یعنی ضریب scaling رو اعمال کنیم.

Single Event UpSets (SEU)

ذرات پر انرژی در مدار داریم DVS می‌زنیم ولتاژ کم بشه نویز کم بشه امّا این ذرات می‌توانند باعث خطا بشن!

$\lambda_{SEU} \propto \exp(-Q_{CRIT})$

$Q_{CRIT} = V_{DD} \cdot C_L$

Soft error ها

DVS has a negative impact on SEU rate.

DVS	ABB	DPM
lowering Vdd and F	lowering Vbs	turning off the unused components

برای پرفورمنس Vbs رو کم کنیم برای انرژی static باید Vbs رو زیاد کنیم.

خرابی ها به 4 دسته تقسیم بندی می‌شوند!

1. خرابی به دلیل اشتباه در طراحی
2. اشتباه در پیاده سازی
3. فرسودگی
4. خرابی به دلیل عوامل خارجی

Timming,

خرابی orginal و follow-up ← مشکل اصلی ← چیزی که بوجود می‌آید ←

Safety Critical Embedded System Requirments

Hard deadline — Real-time Computing
High Reliability
Low Power Consumption
} Conflicting objectives

برای قابلیت اطمینان بالا از تحمل‌پذیر اشکال استفاده می‌کنیم به جای یک کامپوننت 3 تا استفاده می‌کنیم ← افزایش توان مصرفی

حالا یه راه دیگه اجرای 2 باره است ← miss deadline

کاهش ولتاژ برای کاهش انرژی و توان مصرفی

ولتاژ نزدیک به ولتاژ ترشلود می‌شه ← noise margin کم میشه خطا زیاد می‌شه

در Cmos، ولتاژ و فرکانس رابطه مستقیم خطی دارند.

Dynamic Power management → The shutdown of idle system components.

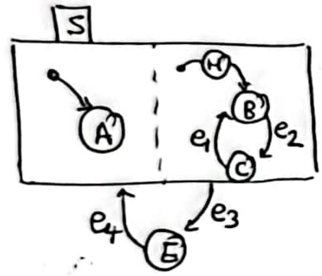
مزیتش generality اش هست.

در SC ها راه اندازی مجدد پرهزینه و زمانبر هست.

کی مصرف؟ وقتی طولانی خاموشه! ← برای خاموش کردن باید STATE رو save کنیم و برای روشن کردن واکشی کنیم. ← DPM طول عمر کامپوننت رو هم بالاتر می‌بره!

* سوال کوئیز: تفاوت ها و شباهت های SPM و کش

ویدیو 14

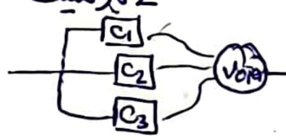


Energy Problem in FT systems → Fault Handling requires redundancy

↳ addition

4 نحوه افزونگی برای کاهش خطا

- → Time — 2 بار پشت هم اجرا کنیم
- → Hardware

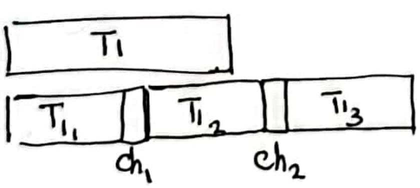


Triple — TMR — module
NMR → Redundancy
n module

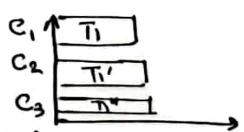
- → Software — Code version Programming
- → Information → Code Hamming و Parity bit

beyond what is needed for normal system operation

حالت checkpoint redundancy



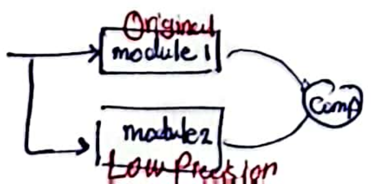
information هم
هم تایم



Software و Hardware

وقتی تایم که آوردیم T1' اضافه کنیم.

duplication with comparison



Fault-tolerance techniques → High reliability

Common aproach to deal with the Permanent Fault

↳ Hardware Redundancy → Increasing the average Peak

Dark Silicon Problem

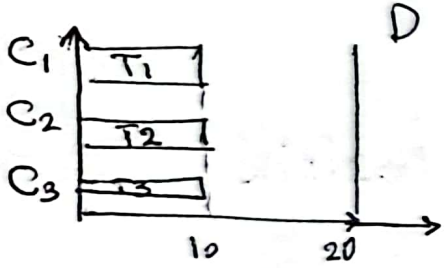
↳ Thermal Design Power (TDP)

↳ Highest Sustainable Power ⟵ راه حل ⟵ Chip's Cooling.

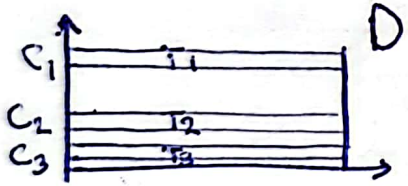
وقتی ترکیم می‌ره بالا نمی‌شه هرزمان استفاده کرد از همه باید ولتاژ فرکانس رو بیاریم پایین.

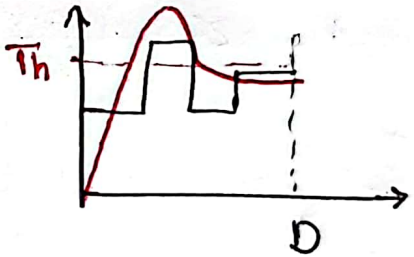
↳ این مشکل در سطح هسته است

↳ توی Embeded مثلاً فن نداریم ⟵ راه حل در امبده راه حل نرم افزاری



کاهش ولتاژ فرکانس





در تیم‌های SC نمی‌توانیم حتی از Th بالاتر بریم.

چون ممکنه تیم زیادت بشه!