محاضرة1 د.فادي متوج

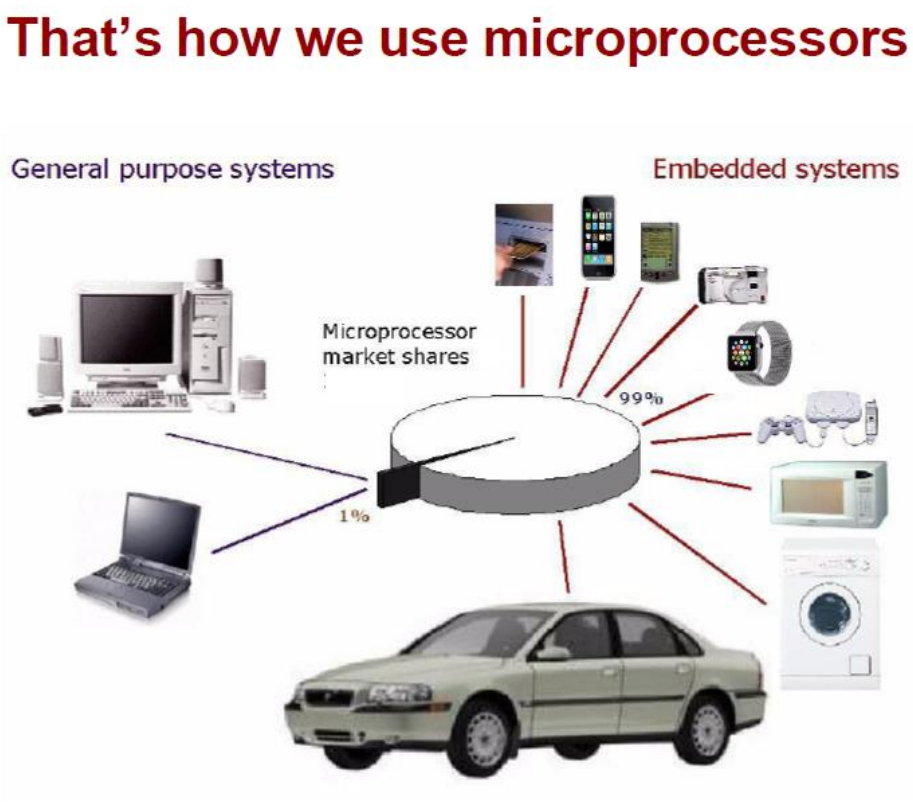
EMBEDDED SYSTEMS AND THEIR DESIGN

1. What is an Embedded System
2. Characteristics of Embedded Applications
3. The Traditional Design Flow
4. An Example
5. A New Design Flow
6. The System Level
7. Course Topics

محاور المحاضرة

1. ما هو النظام المدمج:
   - سنبدأ بالتعريف الأساسي. ما الذي يجعل نظاماً ما "مدمجاً"؟ سنكتشف أنه أكثر من مجرد حاسوب داخل جهاز، بل هو تكامل عميق بين الحوسبة والعالم الفيزيائي المحيط به.
2. خصائص تطبيقات الأنظمة المدمجة:
   - هنا سنتعرف على ما يجعل هذه الأنظمة فريدة ومليئة بالتحديات. سنتحدث عن القيود الصارمة التي تعمل تحتها، مثل: متطلبات الزمن الحقيقي، استهلاك الطاقة المحدود، التكلفة، والموثوقية العالية.
3. مسار التصميم التقليدي:
   - سننظر إلى الطريقة الكلاسيكية التي اتبعها المهندسون لتصميم هذه الأنظمة، حيث يتم تصميم العتاد (Hardware) والبرمجيات (Software) بشكل منفصل تقريباً.
4. لتوضيح الأفكار، سنأخذ مثالاً عملياً ونرى كيف يتم تصميمه باستخدام المسار التقليدي، وسنكتشف نقاط الضعف في هذه الطريقة.

5. مسار تصميم جديد:
   - بعد أن نرى قصور الطريقة التقليدية، سنقدم نهجاً أحدث وأكثر فعالية، يُعرف بـ التصميم القائم على النماذج (Model-Based Design) أو التصميم المشترك للعتاد والبرمجيات، وهو الأسلوب المعتمد للتعامل مع الأنظمة المعقدة اليوم.
6. المستوى النظمي:
   - سنناقش أهمية النظر إلى النظام بأكمله كوحدة واحدة—عتاد وبرمجيات معاً—من مستوى تجريدي عالٍ لاتخاذ قرارات تصميم أفضل في وقت مبكر.
7. مواضيع المقرر:
   - في النهاية، سنربط كل هذه المفاهيم بمواضيع مقررنا بشكل عام



عندما يسمع معظم الناس كلمة "معالج صغري(microprocessor) "، فإن أول ما يتبادر إلى أذهانهم هو المعالج القوي من شركة Intel أو AMD الموجود داخل أجهزة الكمبيوتر المكتبية والمحمولة. لكن هذه النظرة، رغم شيوعها، لا تمثل سوى قمة جبل الجليد.

أين تذهب كل المعالجات؟

المخطط البياني يقسم سوق المعالجات الصغرية إلى فئتين:

- **أنظمة الأغراض العامة (General purpose systems) :**
  - هذه هي أجهزة الكمبيوتر التي نعرفها جميعاً، مثل الكمبيوتر المكتبي واللابتوب. هي مصممة لتشغيل مجموعة متنوعة من البرامج التي يختارها المستخدم (متصفح ويب، معالج نصوص، ألعاب، برامج تصميم).
  - هذه الأنظمة، بكل قوتها وأهميتها، لا تمثل سوى **1%** فقط من سوق المعالجات الصغرية.
- **الأنظمة المدمجة (Embedded systems) :**
  - هذه هي الأنظمة التي يكون فيها المعالج "مدمجاً" أو "مدفوناً" كجزء من جهاز أكبر لأداء **وظيفة محددة ومخصصة** .في معظم الأحيان، لا يدرك المستخدم حتى وجود حاسوب قوي بداخلها.
  - هذه الأنظمة تستهلك **99%** من جميع المعالجات الصغرية المصنعة في العالم. إنها موجودة في كل مكان حولنا.

أمثلة من حياتنا اليومية

- **السيارة :**السيارة الحديثة ليست مجرد آلة ميكانيكية، بل هي شبكة معقدة من الحواسيب. قد تحتوي على أكثر من 100 معالج صغري مدمج يتحكم في كل شيء، من المحرك والمكابح (وهي أنظمة حرجة للسلامة) إلى نظام الترفيه والنوافذ الكهربائية.
- **الهاتف الذكي :**هو نظام مدمج فائق القوة.
- حتى أبسط الأجهزة في منزلنا :**المايكروويف، الغسالة، الكاميرا الرقمية، ساعتنا الذكية** ...كلها تحتوي على عقل إلكتروني مخصص، وهو نظام مدمج.

ما علاقة هذا بهندسة الروبوتات؟

الروبوت هو في جوهره مجموعة من الأنظمة المدمجة المتطورة التي تعمل معاً:

- نظام مدمج للتحكم في المحركات.

- نظام مدمج لمعالجة بيانات الحساسات.
- نظام مدمج للملاحة وتحديد المسار.
- نظام مدمج للتواصل.

مقررنا هذا هو بوابة لفهم كيفية تصميم هذه الأنظمة المدمجة.

What is an Embedded System?

There are several definitions around!

- **Some highlight what it is (not) used for:**

  "An embedded system is any sort of device which includes a programmable component but itself is not intended to be a general purpose computer."

- **Some focus on what it is built from:**

  "An embedded system is a collection of programmable parts surrounded by ASICs and other standard components, that interact continuously with an environment through sensors and actuators."

ما هو النظام المدمج؟

لا يوجد تعريف واحد متفق عليه عالمياً، بل هناك طرق مختلفة للنظر إلى المفهوم. سوف نستعرض التعريفين التاليين لأنهما يكملان بعضهما البعض.

التعريف الأول: حسب الوظيفة

"النظام المدمج هو أي نوع من الأجهزة يحتوي على مكون قابل للبرمجة، ولكنه في حد ذاته ليس مصمماً ليكون حاسوباً للأغراض العامة".

هذا التعريف يركز على **الغرض** من النظام.

- **"يحتوي على مكون قابل للبرمجة"**: هذا يعني أن لديه "عقل" إلكتروني، مثل معالج صغري أو متحكم صغري. إنه ليس مجرد جهاز كهربائي بسيط.
- **"ليس حاسوباً للأغراض العامة"** : هذه هي النقطة المفصلية. على عكس الحاسوب المحمول الذي يمكننا تثبيت أي برنامج عليه، فإن النظام المدمج مصمم لأداء **مهمة واحدة ومحددة** أو مجموعة صغيرة جداً من المهام. لا يمكننا تثبيت متصفح ويب على المايكروويف مثلاً. وظيفته "مدمجة" وثابتة.

التعريف الثاني: حسب البنية (مما يتكون)

"النظام المدمج هو مجموعة من الأجزاء القابلة للبرمجة محاطة بـ ASICs ومكونات قياسية أخرى، تتفاعل باستمرار مع البيئة من خلال الحساسات والمشغلات".

هذا التعريف يركز على **الهيكل المادي** للنظام وكيفية تفاعله.

- النظام المدمج ليس مجرد معالج صغري، بل هو مزيج من "العقل" (الجزء القابل للبرمجة) وقطع **عتاد متخصصة** تسمى **ASICs** .
  - **ASIC** هي اختصار لـ "دارة متكاملة محددة التطبيق(Application-Specific Integrated Circuit) " . وهي شريحة إلكترونية مصممة خصيصاً لأداء مهمة واحدة بكفاءة فائقة (مثل معالجة إشارة صوتية أو صورة)، بدلاً من استخدام المعالج العام لكل شيء.
- النظام المدمج ليس معزولاً، بل هو الجسر الذي يربط عالم الحوسبة الرقمي بالعالم الفيزيائي.
  - **الحساسات** :هي "حواس" النظام. تجمع المعلومات من العالم الفيزيائي (حرارة، ضوء، مسافة، صورة).
  - **المشغلات** :هي "عضلات" النظام. تؤثر في العالم الفيزيائي (محركات، أضواء، مكبرات صوت).

الخلاصة :إذا دمجنا التعريفين، نحصل على صورة كاملة: النظام المدمج هو نظام حاسوبي متخصص، يمزج بين البرمجة والعتاد المخصص، لأداء وظيفة محددة، ويتفاعل باستمرار مع العالم الحقيقي عبر الحساسات والمشغلات.

هذا التعريف يصف الروبوت بشكل مثالي.

What is an Embedded System?

Some of the main characteristics:

- Dedicated (not *general purpose*)
- Contains a programmable component
- Interacts (continuously) with the environment

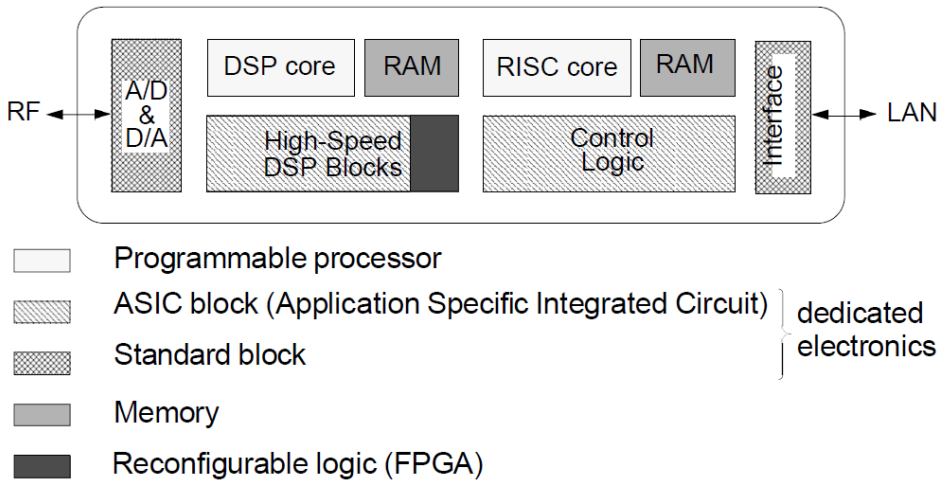
الخصائص الرئيسية للنظام المدمج

1. مخصص(Dedicated) :
   - النظام المدمج يُصمم لأداء وظيفة واحدة محددة أو نطاق ضيق جداً من الوظائف.
   - الغرض من النظام يكون ثابتاً ومحدداً منذ لحظة تصميمه. هو ليس "للأغراض العامة" مثل الحاسوب الشخصي الذي يمكن أن يفعل أي شيء تقريباً.
   - مثال :نظام منع انغلاق المكابح (ABS) في السيارة هو نظام مدمج مخصص لمهمة واحدة فقط: منع العجلات من الانغلاق أثناء الفرملة الطارئة. لا يفعل أي شيء آخر، ويجب أن يؤدي هذه المهمة بشكل مثالي.
2. يحتوي على مكون قابل للبرمجة (Contains a programmable component) :
   - هذا ما يعطي النظام المدمج "ذكاءه". إنه ليس مجرد دارة كهربائية بسيطة، بل يحتوي على مكون يمكن برمجته، مثل معالج صغري (microprocessor) أو متحكم صغري (microcontroller) .
   - هذه القابلية للبرمجة تسمح لنا بتنفيذ منطق تحكم معقد، اتخاذ قرارات، ومعالجة الإشارات، وهو أمر مستحيل بالعتاد البسيط وحده.
3. يتفاعل باستمرار مع البيئة(Interacts continuously with the environment) :
   - الأنظمة المدمجة ليست معزولة في عالمها الرقمي، بل هي متصلة بشكل دائم بالعالم الفيزيائي.
   - إنها في حلقة مستمرة من الإحساس و المعالجة و التأثير:
     - تستخدم الحساسات لترى أو تشعر بحالة العالم الحقيقي (حرارة، سرعة، ضوء، ضغط).
     - يقوم المكون القابل للبرمجة بمعالجة هذه المعلومات واتخاذ قرار.
     - تستخدم المشغلات (actuators) لتؤثر في العالم الحقيقي (تدوير محرك، إضاءة مصباح، إغلاق صمام).

الخلاصة :إذاً، يمكننا الآن تعريف النظام المدمج بأنه: جهاز مخصص وذكي (لأنه قابل للبرمجة)، يعيش في تفاعل مستمر مع البيئة الفيزيائية. هذه الخصائص الثلاث هي التي تولد كل التحديات والمتعة في تصميم هذه الأنظمة.

Two Typical Implementation Architectures

Telecommunication System on Chip



يوضح الشكل أحد الأنظمة المدمجة المعقدة والحديثة، لنرى كيف يبدو من الداخل:

هذا المخطط يمثل بنية نموذجية لما يسمى "نظام على شريحة(System on Chip - SoC) "، وهو نظام إلكتروني كامل بكل مكوناته، تم تصغيره ودمجه على شريحة سيليكون واحدة. هذا هو قلب هاتفنا الذكي، ساعتنا الذكية، والعديد من الأجهزة المتقدمة الأخرى.

مكونات النظام على شريحة(SoC)

سوف نحلل "المكونات" المختلفة في هذا النظام، مستعينين بدليل الألوان في أسفل الشكل. النظام ليس مجرد معالج واحد، بل هو فريق من المتخصصين يعملون معاً.

- المعالجات القابلة للبرمجة (Programmable processor) باللون الأبيض:
  - هذه هي المكونات المرنة في النظام، والتي يمكننا تغيير وظيفتها عن طريق البرمجة.
  - RISC core : هو معالج للأغراض العامة (مثل معالجات ARM الصغيرة) .إنه المدير العام في النظام، وهو بارع في اتخاذ القرارات، تشغيل نظام تشغيل صغير، وإدارة بقية المكونات.
  - DSP core : هذا ليس مديراً عاماً، بل هو "عبقري رياضيات". إنه معالج إشارة رقمية (Digital Signal Processor)، وهو مصمم خصيصاً لأداء العمليات الحسابية المعقدة بسرعة هائلة، مثل تحليل الإشارات الراديوية أو معالجة الصوت والصورة.
- دارات ASIC :
  - هذه هي "الأدوات المتخصصة ASIC ". هي دارة إلكترونية تم تصميمها من الصفر لأداء مهمة واحدة فقط، ولكنها تؤديها بسرعة خيالية وبأقل استهلاك ممكن للطاقة. هي تفتقر للمرونة تماماً.

- A/D & D/A :(محول من تمثيلي إلى رقمي والعكس) وظيفته الوحيدة هي الترجمة بين عالم الإشارات الراديوية التماثلية وعالم البيانات الرقمي
- High-Speed DSP Blocks: هي دارات تقوم بعمليات حسابية متكررة أسرع حتى من معالجDSP

- المنطق القابل لإعادة التشكيل:- (Reconfigurable logic - FPGA)
  - FPGA هي شريحة من العتاد (hardware) يمكننا إعادة برمجة داراتها الداخلية حتى بعد تصنيعها.
  - هي تجمع بين سرعة العتاد وبعض من مرونة البرمجيات، وهي مثالية لتنفيذ البروتوكولات أو الخوارزميات التي قد تتغير أو تحتاج إلى تحديث في المستقبل.
- الذاكرة (Memory) :
  - كل معالج قابل للبرمجة يحتاج إلى "مساحة عمل" خاصة به، وهي ذاكرة الوصول العشوائي(RAM) .

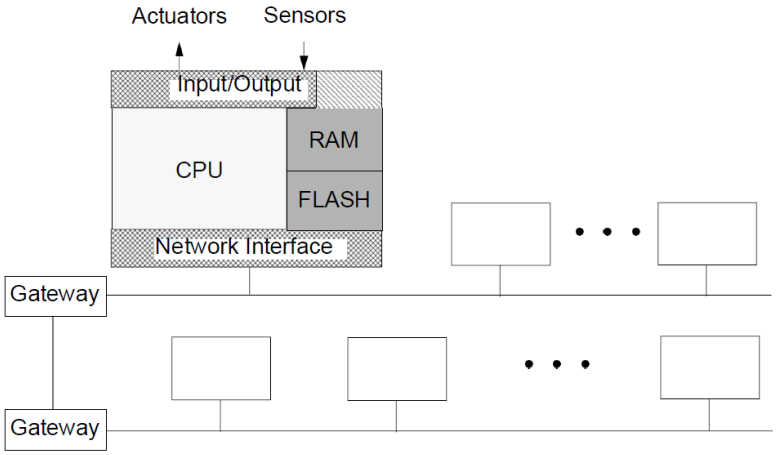
لماذا كل هذا التعقيد؟ لماذا لا نستخدم معالجاً واحداً عملاقاً وفائق السرعة بدلاً من هذا المزيج المعقد؟ الجواب يكمن في المقايضة الهندسية بين المرونة، الأداء، وكفاءة استهلاك الطاقة.

- المعالجات القابلة للبرمجة :تعطينا مرونة عالية.
- دارات ASIC : تعطينا أعلى أداء وأقل استهلاك للطاقة لمهمة محددة.
- دارات FPGA : تعطينا حلاً وسطاً بين الاثنين.

بدمج هذه المكونات المختلفة، يستطيع المهندسون بناء نظام محسن للغاية لأداء مهمته المخصصة (في هذا المثال، الاتصالات) بأفضل طريقة ممكنة. وهذا يوضح أن تصميم الأنظمة المدمجة الحديثة هو فن اختيار المزيج الصحيح من العتاد والبرمجيات.

Two Typical Implementation Architectures

Distributed Embedded System (automotive application)



تعرض لنا هذه الشريحة البنية النموذجية الثانية، والتي هي على النقيض تماماً :النظام المدمج الموزع (Distributed Embedded System)

والتطبيق الكلاسيكي لهذا النموذج، هو السيارات الحديثة .السيارة اليوم ليست حاسوباً واحداً، بل هي شبكة حاسوبية على عجلات".

تحليل بنية النظام الموزع

سوف نفكك هذا المخطط لنفهم كيف يعمل:

- وحدة التحكم الإلكترونية (ECU - The "Brain" Node)
  - هذا الصندوق يمثل وحدة تحكم إلكترونية(Electronic Control Unit - ECU) . يمكن أن نفكر فيها كحاسوب صغير ومتين، مصمم لأداء وظيفة محددة جداً داخل السيارة.
  - مكوناتها:
    - CPU : المعالج الرئيسي أو "العقل" الخاص بها.
    - RAM : ذاكرة العمل المؤقتة التي تستخدمها أثناء التشغيل.
    - FLASH :الذاكرة الدائمة أو "القرص الصلب" الخاص بها. البرنامج الذي يتحكم في المحرك أو المكابح يتم تخزينه هنا بشكل دائم.
    - Input/Output :هذا هو اتصالها بالعالم الفيزيائي. لوحدة تحكم المحرك، الحساسات قد تكون حساسات حرارة وأكسجين، والمشغلات قد تكون بخاخات الوقود وشمعات الإشعال.
    - Network Interface : هذا الذي يسمح لها بالتحدث مع بقية أجزاء السيارة.
- الشبكة(The Network) :
  - الآن، لننظر إلى الصورة الأكبر. السيارة الحديثة تحتوي على عشرات من وحدات التحكم هذه، بل إن بعض السيارات الحديثة تحتوي على ما يزيد عن 100 وحدة ECU
  - يمثلها المخطط بالصناديق الفارغة المتصلة بخط. هذا الخط يمثل ناقل شبكة السيارة (vehicle network bus)، وأشهر مثال عليه هو بروتوكول CAN
  - لدينا وحدة للمحرك، وأخرى للمكابح، وثالثة للوسائد الهوائية، ورابعة لنظام الترفيه، وهكذا. كل منها هو حاسوب منفصل وموزع في مكانه المناسب في السيارة.
- البوابة (Gateway) :

- لماذا يوجد خطان للشبكة و"بوابات" بينهما؟ لأنه ليست كل المعلومات بنفس درجة الأهمية.
- قد يكون لدينا شبكة عالية السرعة وحرجة للمحرك والمكابح والسلامة.
- وقد يكون لدينا شبكة أخرى أبطأ وأقل أهمية للراديو ومكيف الهواء.
- البوابة (Gateway) هي وحدة تحكم خاصة تعمل "كمترجم" أو جسر، تسمح لهذه الشبكات المختلفة بتبادل المعلومات الضرورية فيما بينها بشكل آمن.

المقارنة معSoC :

- النظام على شريحة (SoC) :كل شيء على شريحة واحدة. مدمج جداً وسريع للاتصالات الداخلية.
- النظام الموزع :عدة شرائح موزعة فيزيائياً. هذا التصميم رائع من أجل الموثوقية والنمطية .إذا تعطلت وحدة التحكم الخاصة بالراديو، فإن مكابح السيارة ستظل تعمل لأنها حاسوب منفصل. كما أنه منطقي في نظام كبير كالسيارة أن تكون وحدة التحكم والحساسات الخاصة بالمكابح قريبة من العجلات، وليس متصلة بسلك طويل إلى حاسوب مركزي واحد في مقدمة السيارة.

الخلاصة :البنية الموزعة هي المعيار للأنظمة الكبيرة والمنتشرة فيزيائياً مثل السيارات والطائرات والمصانع. فهم كيفية تصميم وإدارة هذه الشبكات من الأنظمة المدمجة هو مهارة مهمة لمهندس الأنظمة المدمجة.

The Software Component

Software running on the programmable processors:

- Application tasks
- Real-Time Operating System
- I/O drivers, Network protocols, Middleware

بعد أن رأينا البنى الهندسية للعتاد (SoC و النظام الموزع)، نركز هنا على النصف الآخر والمكمل للمعادلة، وهو **المكون البرمجي(The Software Component)** .

البرنامج الذي يعمل على نظام مدمج ليس مجرد كتلة واحدة من الكود، بل هو عادةً عبارة عن طبقات منظمة، لكل منها وظيفة محددة.

طبقات البرمجيات في نظام مدمج

- **مهام التطبيق(Application tasks)**
  - هذه هي الطبقة العليا، وهي تمثل **"ماذا "**يفعله النظام. إنها الكود الذي نكتبه لتحقيق الوظيفة الفريدة للنظام.
  - **مثال :**في روبوت، مهام التطبيق هي خوارزمية الملاحة، كود التعرف على الأشياء، ومنطق التحكم في المحركات. هذه هي البرامج التي تجعل هذا الروبوت مختلفاً عن أي نظام مدمج آخر.
- **نظام التشغيل في الزمن الحقيقي(Real-Time Operating System - RTOS) :**
  - هذه هي الطبقة الوسطى، وهي تمثل **"كيف ومتى "**يتم تنفيذ المهام. نظام التشغيل هذا هو "مدير الموارد" أو "شرطي المرور" للنظام.
  - وظيفته الأساسية هي **جدولة المهام(scheduling)** ، أي تحديد أي مهمة من مهام التطبيق يجب أن تعمل على المعالج في كل لحظة، لضمان أن كل المهام تلتزم بمواعيدها النهائية الحرجة.
  - هو يختلف عن أنظمة التشغيل العادية (مثل Windows) بأنه مصمم ليكون **قابلاً للتنبؤ**. الوقت الذي يستغرقه للانتقال بين المهام مضمون، وهو أمر حيوي في الأنظمة التي قد يؤدي فيها أي تأخير إلى فشل كارثي.
- **برامج تشغيل الإدخال/الإخراج، بروتوكولات الشبكة، والبرمجيات الوسيطة:**
  - هذه هي الطبقة السفلى، وهي تربط الطبقات العليا بالعتاد والعالم الخارجي.
  - **برامج التشغيل (I/O drivers)** : هي برامج صغيرة متخصصة "تتحدث" مباشرة مع قطعة عتاد معينة (مثل حساس حرارة أو شاشة عرض). هي تخفي تفاصيل العتاد المعقدة عن طبقة التطبيق.
  - **بروتوكولات الشبكة (Network protocols)** : إذا كان النظام موزعاً (مثل السيارة)، فهذه هي البرامج التي تنفذ قواعد الاتصال (مثل بروتوكول CAN) لتسمح لوحدات التحكم المختلفة بفهم بعضها البعض.
  - **البرمجيات الوسيطة (Middleware)** : هي طبقة تقع بين نظام التشغيل والتطبيق، وتقدم خدمات مشتركة. من أشهر الأمثلة هو **نظام تشغيل الروبوت ROS** ، الذي يقدم مكتبات وخدمات جاهزة للتواصل بين حساسات الروبوت ومحركاته.

باختصار، المكون البرمجي هو حزمة متكاملة من الطبقات تعمل معاً بتناغم: برامج التشغيل في الأسفل، نظام التشغيل في الزمن الحقيقي في المنتصف لإدارة الزمن، ومهام التطبيق في الأعلى لتحقيق الهدف النهائي للنظام.

Characteristics of Embedded Applications

What makes them special?

- **Like with "ordinary" applications, functionality and user interfaces are often *very complex*.**

But, in addition to this:

- **Time constraints**
- **Power constraints**
- **Cost constraints**
- **Safety**
- **Time to market**

بعد أن عرفنا النظام المدمج، نسأل هنا سؤالاً مهماً: ما الذي يجعل هذه التطبيقات خاصة أو فريدة من نوعها؟

خصائص تطبيقات الأنظمة المدمجة

أولاً، من المهم أن نعرف أن "مدمج" لا تعني "بسيط". فمثلها مثل تطبيقات الحاسوب العادية، يمكن أن تكون وظائفها وواجهات المستخدم الخاصة بها **معقدة للغاية** كما في نظام الملاحة والترفيه في سيارة حديثة مثلاً.

ولكن، بالإضافة إلى هذا التعقيد، يواجه مهندس النظم المدمجة مجموعة من القيود القاسية التي لا يقلق بشأنها مطور برامج الحاسوب العادي. هذه القيود هي :

- **قيود الزمن (Time constraints)** : الكثير من الأنظمة المدمجة هي **أنظمة تعمل في الزمن الحقيقي**. هذا يعني أن الحصول على الإجابة الصحيحة ليس كافياً، بل يجب الحصول عليها **في الزمن الصحيح**.
  - **مثال :** في نظام الوسادة الهوائية في السيارة، يجب أن يصل أمر "الفتح" خلال أجزاء من الثانية بعد استشعار الحادث. وصول الأمر متأخراً لا يقل سوءاً عن عدم وصوله إطلاقاً.

- **قيود الطاقة (Power constraints)** : العديد من الأنظمة المدمجة تعمل على بطاريات أو لديها ميزانية طاقة محدودة جداً لتجنب الحرارة.
    - **مثال :**جهاز تنظيم ضربات القلب الطبي أو ساعتك الذكية يجب أن يتم تصميمها لتستهلك أقل قدر ممكن من الطاقة، لتعمل لأشهر أو سنوات على بطارية صغيرة.
- **قيود التكلفة (Cost constraints)** : غالباً ما تكون الأنظمة المدمجة جزءاً من منتجات استهلاكية تُصنع بالملايين. توفير بضعة سنتات من تكلفة كل وحدة يعني توفير الملايين للشركة.
    - **مثال :**قد يكون الفرق بين متحكم صغري بدولار واحد وآخر بدولارين هو الفرق بين نجاح المنتج وفشله في السوق.
- **السلامة (Safety)** : العديد من هذه الأنظمة **حرجة للسلامة**، أي أن أي عطل فيها قد يؤدي إلى إصابة أو وفاة.
    - **مثال :**أنظمة التحكم في الطائرات ، أنظمة المكابح في السيارات، وأجهزة العلاج الإشعاعي. تصميم هذه الأنظمة يتطلب إجراءات صارمة جداً لضمان أنها آمنة تماماً.
- **وقت الوصول إلى السوق(Time to market) :** سوق الإلكترونيات شديد التنافس. الوصول إلى السوق أولاً بمنتج جديد قد يكون العامل الحاسم للنجاح التجاري. هذا يضع ضغطاً هائلاً على المهندسين لتصميم أنظمة معقدة في جداول زمنية ضيقة جداً.

الخلاصة :وظيفة مهندس النظم المدمجة هي عملية موازنة مستمرة بين إضافة وظائف معقدة وتلبية هذه القيود القاسية.

Time constraints

- **Embedded systems have to perform in real-time: if data is not ready by a certain deadline, the system fails to perform correctly.**
    - ***Hard deadline*: failure to meet leads to major hazards.**
    - ***Soft deadline*: failure to meet is tolerated but affects quality of service.**

قيود الزمن في الأنظمة المدمجة

- **يجب على الأنظمة المدمجة أن تعمل في الزمن الحقيقي.**

- هذه هي الفكرة الأساسية. في حاسوبنا المكتبي، إذا استغرقت صفحة ويب ثانية إضافية للتحميل، فهذا مزعج ولكنه ليس فشلاً. في الأنظمة المدمجة، **الصحة لا تعتمد فقط على صحة النتيجة، بل على توقيت وصولها**.
- القاعدة الذهبية هنا هي :**الإجابة الصحيحة التي تأتي متأخرة هي إجابة خاطئة.**
- إذا لم تكن البيانات جاهزة بحلول موعد نهائي معين(deadline) ، فإن النظام **يفشل في أداء وظيفته بشكل صحيح**.

أنواع المواعيد النهائية(Deadlines)

نميز بين نوعين من المواعيد النهائية:

- **الموعد النهائي الصارم(Hard deadline) :**
  - الفشل في تلبية هذا الموعد النهائي يؤدي إلى **فشل كارثي** للنظام، أو "مخاطر جسيمة".
  - **أمثلة:**
    - **نظام الوسادة الهوائية** :يجب أن يصدر أمر الفتح خلال أجزاء من الثانية بعد الحادث.
    - **الروبوت الجراحي** :أمر "توقف" يجب أن ينفذ فوراً قبل أن يؤذي المريض.
    - **نظام التحكم في الطائرة** :يجب أن تتم تعديلات أسطح التحكم بسرعة فائقة للحفاظ على استقرار الطائرة.
- **الموعد النهائي المرن(Soft deadline) :**
  - الفشل في تلبية هذا الموعد النهائي ليس كارثياً، ولكنه **يؤثر على جودة الخدمة** أو تجربة المستخدم. النظام لا يزال يعمل، ولكن بشكل أسوأ.
  - **أمثلة:**
    - نظام الترفيه في السيارة يستغرق ثانية إضافية للاستجابة للمس.
    - كاميرا رقمية تستغرق وقتاً أطول قليلاً لمعالجة وحفظ الصورة.

الخلاصة :من أولى مهام مهندس الأنظمة المدمجة عند تحليل أي مشكلة هو تحديد المواعيد النهائية وتصنيفها. هذا القرار سيحدد كل شيء آخر في التصميم، من اختيار المعالج ونظام التشغيل إلى استراتيجية الاختبار والتحقق.

Power constraints

- There are several reasons why low power/energy consumption is required:
  - Cost aspects:
    High energy consumption ⇒ large electricity bill
    expensive power supply
    expensive cooling system
  - Reliability
    High power consumption ⇒ high temperature that affects life time
  - Battery life
    High energy consumption ⇒ short battery life time
  - Environmental impact

قيود الطاقة :

بعد أن تحدثنا عن قيود الزمن، ننتقل الآن إلى القيد الثاني الهام الذي يميز مجالنا :**قيود الطاقة**.

عندما نصمم حاسوباً قوياً للألعاب، يمكننا ببساطة استخدام مزود طاقة أكبر ومراوح أكثر. في عالم الأنظمة المدمجة، نحن لا نملك هذه الرفاهية. الطاقة ليست مجرد مورد، بل هي قيد أساسي يؤثر على كل قرار في التصميم. هنا سوف نشرح الأسباب الرئيسية لذلك.

أسباب أهمية استهلاك الطاقة المنخفض

- **جوانب التكلفة:**
  - **للمستخدم** :استهلاك طاقة عالٍ يعني فاتورة كهرباء أكبر، خاصة للأجهزة التي تعمل باستمرار (مثل أجهزة الراوتر أو أنظمة المنزل الذكي).
  - **للمصنع** :وهذا هو الأهم لنا كمصممين، استهلاك طاقة أعلى يتطلب مكونات أغلى ثمناً. نحتاج إلى دارة تزويد طاقة أكبر وأقوى، والأهم من ذلك، أن الطاقة العالية تولد حرارة عالية، مما يتطلب نظام تبريد مكلف (مثل مشتت حراري أو مروحة). كل هذا يضاف إلى تكلفة المنتج النهائية.
- **الموثوقية :**

- **الحرارة هي عدو الإلكترونيات** .هذه قاعدة أساسية. استهلاك طاقة عالٍ يولد حرارة عالية، والحرارة المرتفعة تقلل بشكل كبير من العمر الافتراضي للمكونات الإلكترونية. كلما عملت الشريحة الإلكترونية في درجة حرارة أعلى، تدهورت بشكل أسرع. هذا أمر حيوي للأنظمة التي يجب أن تعمل لسنوات في بيئات قاسية، مثل وحدات التحكم في محرك السيارة.

- **عمر البطارية:**
  - هذا هو السبب الأكثر وضوحاً للأجهزة المحمولة. استهلاك طاقة عالٍ يعني ببساطة عمر بطارية أقصر..
  - هذا هو التحدي التصميمي الأكبر للهواتف الذكية، الساعات الذكية، الأجهزة الطبية المزروعة، وأي جهاز متنقل. كل جزء من الملي أمبير يمكننا توفيره في تصميمنا يترجم مباشرة إلى منتج أفضل وأكثر تنافسية.

- **الأثر البيئي :**
  - أخيراً، هناك الصورة الأكبر. هناك المليارات من هذه الأجهزة التي تعمل حول العالم. تقليل استهلاك الطاقة لكل جهاز، حتى بكمية صغيرة، يؤدي إلى توفير هائل في استهلاك الطاقة العالمي وتقليل البصمة الكربونية.

الخلاصة :الطاقة ليست مجرد مسألة عمر بطارية. إنها قيد أساسي يؤثر مباشرة على **تكلفة** النظام، **موثوقيته**، **عمره الافتراضي**، و**أثره البيئي**.

Cost constraints

- **Embedded systems are very often mass products in highly competitive markets and have to be shipped at a low cost.**

  What we are interested in:

  - **Manufacturing cost**
  - **Design cost**
  - **Material cost (Bill of Material)**
  - **Warranty cost**

بعد الحديث عن قيود الزمن والطاقة، نصل الآن إلى القيد الثالث الكبير الذي يحكم عملنا كمهندسي أنظمة مدمجة : **قيود التكلفة(Cost constraints) .**

في عالم المنتجات التجارية، التصميم العبقري تقنياً ولكنه باهظ الثمن هو تصميم فاشل.

تحليل التكاليف في الأنظمة المدمجة

إن الأنظمة المدمجة غالباً ما تكون منتجات تصنع بكميات ضخمة في أسواق شديدة التنافس، مما يجعل التكلفة المنخفضة عاملاً حاسماً للنجاح. كمهندسين، نحن نهتم بتحليل عدة أنواع من التكاليف:

- **تكلفة التصنيع:**
  - هذه هي تكلفة تجميع المنتج في المصنع. تشمل تشغيل الآلات، أجور العمال، واختبار كل وحدة للتأكد من أنها تعمل بشكل صحيح.
- **تكلفة التصميم :**
  - هذه هي التكلفة التي تدفعها الشركة **مرة واحدة** قبل بدء الإنتاج. تشمل رواتب فريق المهندسين (عتاد وبرمجيات)، تكلفة رخص برامج التصميم والمحاكاة، وتكلفة بناء النماذج الأولية. تُعرف أيضاً بـ "التكاليف الهندسية غير المتكررة.
- **تكلفة المواد :**
  - هذه هي التكلفة الأكثر شهرة وأهمية في المنتجات ذات الإنتاج الضخم .**قائمة المواد** هي فاتورة مفصلة بتكلفة كل مكون إلكتروني على اللوحة: المعالج، شرائح الذاكرة، المقاومات، المكثفات، اللوحة المطبوعة نفسها، إلخ.
  - **مثال :**توفير سنت واحد في مقاومة قد يوفر مليون دولار عند إنتاج 100 مليون وحدة من جهاز ما.
- **تكلفة الضمان :**
  - هذه هي "تكلفة الفشل". إذا كان تصميمنا يستخدم مكونات رخيصة وغير موثوقة، فقد يتعطل عدد كبير من الأجهزة بعد بيعها. تكلفة إصلاح واستبدال هذه الأجهزة وهي تحت الضمان يمكن أن تكون كارثية وتدمر أرباح الشركة بالكامل.
  - هذا يوضح أن هناك علاقة مباشرة بين جودة التصميم والتكلفة النهائية.

الخلاصة :وظيفتنا كمهندسين ليست فقط جعل النظام يعمل، بل جعله يعمل ضمن ميزانية صارمة. هذا يعني القيام بمقايضات مستمرة بين هذه التكاليف المختلفة. فمثلاً، قد يكون من الحكمة إنفاق المزيد على التصميم (للتأكد من الموثوقية) لتقليل تكاليف الضمان المكلفة لاحقاً.

Safety

- **Embedded systems are often used in life critical applications: avionics, automotive electronics, nuclear plants, medical applications, military applications, etc.**
  - ***Reliability* and *safety* are major requirements.**
    In order to guarantee safety during design:
    - **Formal verification: mathematics-based methods to verify certain properties of the designed system.**
    - **Automatic synthesis:certain design steps are automatically performed by design tools.**

بعد أن ناقشنا قيود الزمن والطاقة والتكلفة، نصل الآن إلى القيد الأكثر أهمية وخطورة على الإطلاق في الكثير من الأنظمة المدمجة: **السلامة**

السلامة في الأنظمة المدمجة

- **غالباً ما تُستخدم الأنظمة المدمجة في تطبيقات حرجة للحياة.**
  - نحن لا نصمم ألعاباً فقط، بل نصمم أنظمة قد يكون الخطأ فيها مميتاً.
  - **أمثلة:**

- **إلكترونيات الطيران:** مثل نظام الطيار الآلي أو نظام التحكم في أسطح توجيه الطائرة.
- **إلكترونيات السيارات:** مثل نظام منع انغلاق المكابح(ABS) ، الوسائد الهوائية، أو نظام القيادة الذاتية.
- **المحطات النووية:** الأنظمة التي تتحكم في قلب المفاعل.
- **التطبيقات الطبية:** مثل منظمات ضربات القلب، مضخات الأنسولين، وأجهزة التنفس الاصطناعي.

كيف نضمن السلامة؟

عندما تكون المخاطر عالية بهذا الشكل، لا يمكننا الاعتماد على الاختبار التقليدي وحده. الاختبار يمكنه أن يظهر وجود الأخطاء، ولكنه لا يستطيع إثبات عدم وجودها. لذا، نلجأ إلى أساليب هندسية أكثر صرامة لضمان السلامة أثناء التصميم:

- **التحقق الرسمي(Formal verification) :**
  - هذه ليست مجرد عملية اختبار، بل هي عملية **إثبات رياضي** .نستخدم أساليب منطقية ورياضية صارمة **لنثبت** أن تصميم نظامنا يلتزم بخصائص سلامة معينة **في جميع الظروف الممكنة**.
  - **مثال :**بدلاً من اختبار المكابح في 100 سيناريو مختلف (اختبار)، نقوم بإثبات رياضي أن "نظام المكابح لن يفشل أبداً في الاستجابة" (تحقق رسمي).
- **التوليف التلقائي(Automatic synthesis) :**
  - هذه هي عملية استخدام أدوات تصميم حاسوبية لتوليد (أو "ترجمة") التصميم التفصيلي (سواء كان كوداً برمجياً أو مخططاً عتادياً) مباشرة من نموذج عالي المستوى.
  - **كيف يساعد هذا في السلامة؟** إنه يقلل بشكل كبير من **الخطأ البشري** .المهندس البشري قد يرتكب أخطاءً صغيرة عند كتابة آلاف الأسطر من الكود أو رسم الدارات المعقدة. الأداة التلقائية، إذا كانت مصممة ومختبرة جيداً، تقوم بعملية التحويل هذه بشكل متسق وخالٍ من الأخطاء.

الخلاصة :في الأنظمة الحرجة للسلامة، لا يكفي أن نعتقد أن النظام آمن. يجب أن نستخدم أساليب هندسية متقدمة مثل التحقق الرسمي والتوليف التلقائي لنبني حجة سلامة قوية ومقنعة.

Short time to market

- In highly competitive markets it is critical to catch the *market window*: a short delay with the product on the market can have catastrophic financial consequences (even if the quality of the product is excellent).
  - Design time has to be reduced!
    - Good design methodologies.
    - Efficient design tools.
    - Reuse of previously designed and verified (hardw&softw) blocks.
    - *Good designers who understand both software and hardware*!

بعد أن غطينا القيود التقنية والهندسية، نأتي الآن إلى قيد حاسم تفرضه علينا حقيقة عالم الأعمال والمنافسة :**قصر وقت الوصول إلى السوق .**

أهمية سرعة الوصول إلى السوق

- **في الأسواق شديدة التنافس، من الأهمية بمكان اللحاق بـ"نافذة السوق"**
  - **"نافذة السوق"** هي فترة زمنية محددة وحرجة يمكن خلالها إطلاق منتج لتحقيق أقصى قدر من المبيعات والحصة السوقية.
  - في عالم التكنولوجيا، تأخير إطلاق منتجنا لبضعة أشهر فقط قد يعني أن منافسنا قد سيطر على السوق بالكامل، مما يؤدي إلى عواقب مالية وخيمة.

كيف نقلل من وقت التصميم؟

للوصول إلى السوق في الوقت المناسب، يجب علينا تقليل الوقت الذي نستغرقه في تصميم وتطوير المنتج. هناك أربع طرق رئيسية لتحقيق ذلك:

1. **منهجيات تصميم جيدة :**
   - هذا يعني اتباع استراتيجية تصميم ذكية ومنظمة. على سبيل المثال، استخدام **"التصميم القائم على النماذج "**الذي سنتعلمه لاحقاً في المحاضرات القادمة أفضل بكثير من الطرق التقليدية، لأنه يساعدنا على اكتشاف الأخطاء في وقت مبكر جداً من عملية التصميم، مما يوفر وقتاً هائلاً لاحقاً.
2. **أدوات تصميم فعالة:**
   - الاعتماد على برامج وأدوات قوية تسرّع عمل المهندس، مثل: المحاكيات المتقدمة، أدوات التحقق الرسمي، والمترجمات (compilers) التي تقوم بتوليد الكود تلقائياً من النماذج.
3. **إعادة استخدام الكتل المصممة والمحققة مسبقاً:**
   - هذا مبدأ أساسي في الهندسة الحديثة ويُعرف بـ **إعادة استخدام الملكية الفكرية .**
   - **الفكرة :**لماذا تخترع العجلة من جديد؟ إذا كان فريق آخر قد صمم بالفعل واختبر جيداً مكوناً للتعامل مع اتصالات USB ، أو مكتبة برمجية لخوارزمية معينة، فيجب علينا إعادة استخدامها. هذا يوفر كمية هائلة من الوقت ويقلل من المخاطر. الأمر يشبه البناء بمكعبات الليغو الجاهزة بدلاً من صنع كل مكعب من الصفر.
4. **مصممون جيدون يفهمون البرمجيات والعتاد معاً:**

- أخيراً، العنصر البشري. في الماضي، كان فريق العتاد وفريق البرمجيات يعملان بشكل منفصل. هذا الأسلوب بطيء وغير فعال.
- مهندس النظم المدمجة الحديث يجب أن يفهم كلا الجانبين. يجب أن نعرف كيف سيعمل برنامجنا على العتاد المختار، وأن نكون قادرين على اتخاذ قرارات المقايضة بين تنفيذ وظيفة ما في العتاد (للسرعة) أو في البرمجيات (للمرونة). هذا ما يسمى **التصميم المشترك للعتاد والبرمجيات (Hardware/Software Co-design)**.

الخلاصة: النجاح في سوق اليوم لا يعتمد فقط على الجودة التقنية، بل على إيصال منتج موثوق وفعال من حيث التكلفة إلى السوق في الوقت المناسب. وهذا يتطلب مزيجاً من المنهجيات الذكية، والأدوات القوية، وإعادة الاستخدام الفعالة، والمهندسين متعددي التخصصات.

Why is Design of Embedded Systems Difficult?

- High Complexity
- Strong time&power constraints
- Low cost
- Short time to market
- Safety critical systems

In order to achieve these requirements, systems have to be highly optimized.

Both hardware and software aspects have to be considered simultaneously!

بعد أن استعرضنا كل خصائص وقيود الأنظمة المدمجة، سوف نجيب على السؤال: **لماذا تصميم الأنظمة المدمجة صعب؟**

خلاصة تحديات التصميم

نعيد التذكير بقائمة التحديات التي يواجهها المهندس، والتي ناقشناها بالتفصيل:

- **التعقيد العالي**: الأنظمة تؤدي وظائف معقدة.
- **قيود الزمن والطاقة**: يجب أن تكون سريعة وتستهلك طاقة قليلة.
- **التكلفة المنخفضة**: يجب أن تكون رخيصة للإنتاج بكميات كبيرة.

- **وقت الوصول القصير للسوق** :يجب تصميمها بسرعة.
- **أنظمة حرجة للسلامة** :يجب أن تكون آمنة وموثوقة تماماً.

لكي نحقق كل هذه المتطلبات المتضاربة، يجب أن يكون تصميمنا **محسّناً ومثالياً (highly optimized)** إلى أقصى درجة.

الحل: التصميم المشترك

وهنا نصل إلى أهم استنتاج في محاضرتنا:

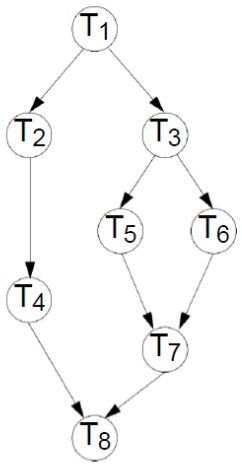
"يجب النظر في جو انب العتاد (Hardware) والبرمجيات (Software) في نفس الوقت"

هذا المبدأ هو حجر الزاوية في تصميم الأنظمة المدمجة الحديثة، ويُعرف باسم **التصميم المشترك للعتاد والبرمجيات (Hardware/Software Co-design)**.

- **الطريقة التقليدية (الخاطئة)** :كان فريق العتاد يصمم شريحة إلكترونية، ثم يرميها إلى فريق البرمجيات ليبرمجها. هذه الطريقة بطيئة وتؤدي إلى أنظمة غير مثالية.
- **الطريقة الحديثة (الصحيحة)** :يعمل فريق العتاد وفريق البرمجيات معاً منذ اليوم الأول.
  - **مثال توضيحي** :لنفكر في تصميم سيارة فورمولا 1. لا يقوم فريق بتصميم المحرك بمعزل عن فريق تصميم هيكل السيارة. بل يعملان معاً، لأن شكل المحرك يؤثر على الديناميكا الهوائية، ومتطلبات التبريد لهيكل السيارة تؤثر على تصميم المحرك.
  - بنفس الطريقة، يجب أن نقرر معاً: هل من الأفضل تنفيذ خوارزمية معينة في **البرنامج** (مرن وسهل التعديل ولكنه أبطأ)، أم تصميم قطعة **عتاد** مخصصة (ASIC) للقيام بها (فائقة السرعة والكفاءة ولكنها غير مرنة)؟
  - **مثال** :المعالجات في الهواتف الذكية اليوم تحتوي على وحدات عتاد مخصصة لتسريع تطبيقات الذكاء الاصطناعي (مثل معالجة الصور). قرار تصميم هذه الوحدات المخصصة هو قرار "تصميم مشترك"، حيث تم تحديد أن تنفيذ هذه المهام برمجياً فقط لن يكون كافياً.

الخلاصة :تصميم الأنظمة المدمجة صعب لأنه يتطلب موازنة دقيقة بين العديد من القيود المتضاربة. والمفتاح لتحقيق هذا التوازن هو عدم الفصل بين العالمين، بل النظر إلى العتاد والبرمجيات كوحدة واحدة متكاملة واتخاذ القرارات التصميمية لهما معاً.

An Example



The system to be implemented is modelled as a *task graph*:

- a node represents a *task* (a unit of functionality activated as response to a certain input and which generates a certain output).
- an edge represents a precedence constraint and data dependency between two tasks.

Period: 42 time units

- The task graph is activated every 42 time units ⇒ an activation has to terminate in time less than 42.

Cost limit: 8

- The total cost of the implemented system has to be less than 8.

ستقدم هنا مثالاً عملياً يوضح كيف نبدأ عملية تصميم نظام معقد.

بعد أن تحدثنا عن ضرورة النظر إلى العتاد والبرمجيات معاً، نحتاج إلى طريقة لوصف وظائف النظام بشكل مجرد قبل أن نقرر أين سيتم تنفيذ كل وظيفة (هل في العتاد أم في البرمجيات؟). هذه الطريقة هي **مخطط المهام(Task Graph)** .

شرح المثال: نظام مبني على مخطط المهام

- **يتم نمذجة النظام المراد تنفيذه كمخطط مهام.**
  - **العقدة (node) تمثل مهمة(task)** : كل دائرة في الرسم(T_1، T_2، إلخ) هي مهمة أو وظيفة يجب إنجازها. مثلاً، T_1 قد تكون "قراءة بيانات من حساس"، و T_2قد تكون "فلترة البيانات".
  - **الحافة (edge) تمثل قيود الأسبقية** :السهم من T_1إلى T_2 يعني أن **المهمة T_1 يجب أن تنتهي بالكامل قبل أن تبدأ المهمة T_2** لأن T_2 تعتمد على نتيجة أو بيانات من T_1
  - **مسارات متوازية** :نلاحظ من الشكل كيف أن T_2 و T_3يمكن أن يعملا بالتوازي بعد انتهاء T_1 لكن المهمة T_4 لا يمكن أن تبدأ إلا بعد انتهاء T_2 والمهمة T_7 لا تبدأ إلا بعد انتهاء T_3و T_6

قيود ومتطلبات النظام

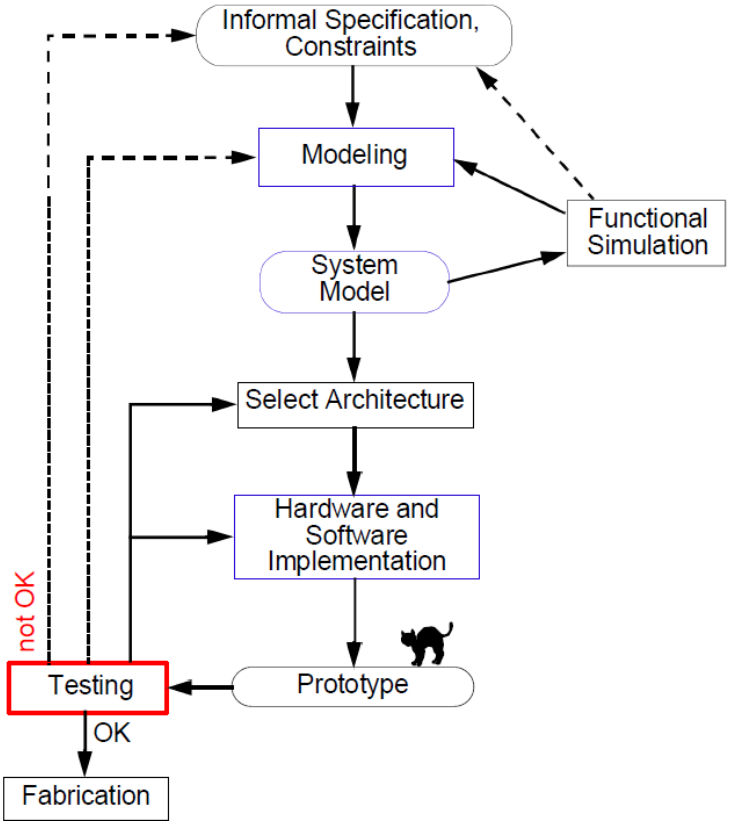
- **الفترة (Period) :42 وحدة زمنية**

- هذا هو **الموعد النهائي الكلي (global deadline)** للنظام. يجب أن يتم تنشيط مخطط المهام هذا وتنفيذ كل شيء فيه، من بداية T_1 إلى نهاية T_8، في زمن **أقل من 42 وحدة زمنية** .إذا استغرق النظام وقتاً أطول، فإنه يعتبر فاشلاً. هذا هو قيد الزمن الحقيقي للنظام.

- **حد التكلفة(Cost limit) :8**

- هذا هو **قيد الميزانية**" .التكلفة" هنا يمكن أن تكون شيئاً من عدة أشياء: التكلفة المادية بالدولار، أو مساحة الشريحة الإلكترونية المستخدمة، أو استهلاك الطاقة. مهما كان تعريفها، يجب أن يكون تصميمنا النهائي بحيث لا تتجاوز تكلفته الإجمالية 8 وحدات.

الخلاصة :مهمتنا كمهندسين الآن هي استكشاف طرق مختلفة لتنفيذ هذه "الوصفة" (مخطط المهام) مع الالتزام الصارم بقيود الزمن والتكلفة). الشرائح التالية ستوضح لنا الخيارات المتاحة لتنفيذ كل مهمة (مثلاً، على معالجات مختلفة أو عتاد مخصص) وكيفية اختيار المزيج الأمثل.



Traditional Design Flow

1. **Start from some informal specification of functionality and a set of constraints**
2. **Generate a more formal model of the functionality, based on some modeling concept. Such model is our task graph**
3. **Simulate the model in order to check the functionality. If needed make adjustments.**
4. **Choose an architecture (µprocessor, buses, etc.) such that cost limits are satisfied and, you hope, time and power constraints are fulfilled.**
5. **Build a prototype and implement the system.**
6. **Verify the system: neither time nor power constraints are satisfied!!!**

هنا نشرح الطريقة التي كانت تصمم بها الأنظمة المدمجة في الماضي، وهي **مسار التصميم التقليدي (Traditional Design Flow)**. من المهم جداً أن نفهم هذه الطريقة القديمة لنقدر قيمة الأساليب الحديثة التي سنتعلمها لاحقاً.

خطوات مسار التصميم التقليدي

سوف نتتبع الخطوات كما هي موضحة في المخطط:

1. **البداية (Informal Specification) :**نبدأ من فكرة عامة أو مواصفات غير رسمية من العميل (مثلاً، "أريد جهازاً يقيس نبضات القلب").
2. **النمذجة(Modeling)** : يقوم المهندسون بتحويل هذه الفكرة إلى **نموذج رسمي** يصف وظائف النظام بدقة، مثل "مخطط المهام (Task Graph) "الذي رأيناه في المثال السابق. هذا النموذج يركز فقط على **ماذا** سيفعل النظام.
3. **المحاكاة الوظيفية (Functional Simulation)** : نقوم بتشغيل محاكاة للنموذج لنتأكد من أن المنطق صحيح (مثلاً، هل المعادلة التي تحسب نبضات القلب صحيحة؟). في هذه المرحلة، نحن **لا نهتم أبداً بالعتاد أو بالزمن**.
4. **اختيار البنية (Select Architecture)** : **بشكل منفصل تماماً**، يقوم فريق آخر باختيار المكونات المادية (المعالج، الذاكرة، إلخ) بناءً على قيود التكلفة حيث يكون لديهم **أمل** فقط في أن تكون هذه المكونات سريعة وقوية بما يكفي لتحقيق قيود الزمن والطاقة. إنه أشبه بالتخمين القائم على الخبرة.
5. **التنفيذ وبناء النموذج الأولي (Implementation & Prototype)** : هذه هي المرحلة الطويلة والمكلفة. يقوم فريق العتاد ببناء الدارات الإلكترونية، ويقوم فريق البرمجيات بكتابة كل الكود ليعمل على هذا العتاد المحدد .
6. **الاختبار (Testing)** : أخيراً، يتم دمج العتاد والبرمجيات معاً واختبار النموذج الأولي لأول مرة.

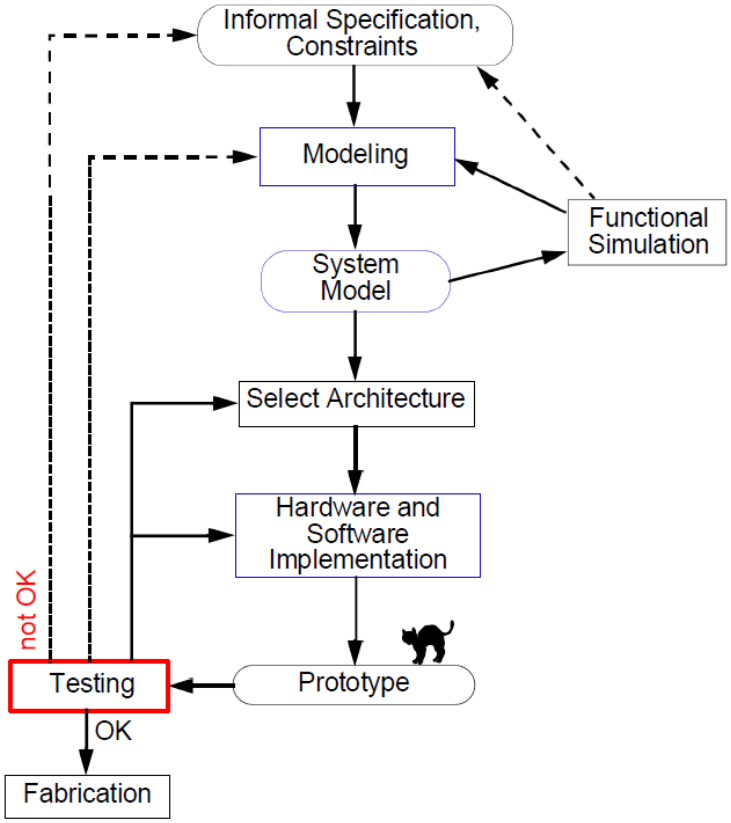
العيب في هذه الطريقة

"لم يتم تحقيق قيود الزمن ولا قيود الطاقة"

إذا فشل الاختبار (النظام بطيء جداً أو يستهلك طاقة أكثر من اللازم)، يجب علينا العودة إلى الوراء. المشكلة هي أننا اكتشفنا هذا الفشل في **مرحلة متأخرة جداً**، بعد أن تم إنفاق الكثير من الوقت والمال.

المشكلة الجوهرية :تم اتخاذ قرارات العتاد وقرارات البرمجيات بشكل منفصل، ولم يتم تقييم الأداء إلا في النهاية. هذا الأسلوب يشبه بناء منزل كامل، وبعد الانتهاء من السقف، نقوم بفحص أساسات المنزل لنرى إن كانت قوية بما يكفي.

الخلاصة :مسار التصميم التقليدي بطيء، مكلف، وعالي المخاطر، وهو غير مناسب للأنظمة المعقدة اليوم. وهذا يمهد الطريق لحاجتنا الماسة إلى "مسار تصميم جديد" يسمح لنا بتقييم الأداء واتخاذ القرارات الصحيحة في وقت مبكر جداً من العملية.



Traditional Design Flow

Now you are in great trouble: you have spent a lot of time and money and nothing works!

- **Go back to 4, choose a new architecture and start a new implementation.**
- **Or negotiate with the customer on the constraints.**

ماذا يحدث عندما تفشل الاختبارات؟

بعد شهور من العمل الشاق وإنفاق الكثير من المال على تصميم العتاد والبرمجيات وبناء النموذج الأولي، نكتشف في مرحلة الاختبار النهائية أن النظام بطيء جداً أو يستهلك طاقة أكثر من اللازم.

أمام الفريق الهندسي الآن خياران، وكلاهما سيء:

1. **العودة إلى الخطوة 4، اختيار بنية جديدة، والبدء بتنفيذ جديد.**
   - هذا يعني عملياً التخلص من جزء كبير من العمل الذي تم إنجازه. قد يعني هذا اختيار معالج أسرع (وأغلى ثمناً)، مما يجبر فريق العتاد على إعادة تصميم لوحة الدارات الإلكترونية. وسيتعين على فريق البرمجيات نقل كل الكود إلى المعالج الجديد وإعادة كتابة أجزاء كبيرة منه.
   - هذا الخيار يؤدي إلى تأخير كارثي في موعد إطلاق المنتج، وزيادة هائلة في التكلفة.
2. **أو التفاوض مع العميل بشأن القيود.**
   - هذا هو الخيار الذي يعني الاعتراف بالفشل. يذهب مدير المشروع إلى العميل ويقول: "نعلم أنك أردت أن تعمل البطارية لمدة 10 ساعات، لكن أفضل ما استطعنا تحقيقه هو 6 ساعات فقط. هل تقبل بذلك؟"

- هذا يضر بسمعة الشركة، وقد يؤدي إلى خسارة العقد أو إطلاق منتج ضعيف لا يستطيع المنافسة في السوق.

التكلفة الحقيقية للأخطاء المتأخرة

أظهرت الدراسات في هندسة البرمجيات حقيقة صادمة :**تكلفة إصلاح خطأ برمجي بعد إطلاق المنتج قد تكون أغلى بما يصل إلى 100 مرة من تكلفة إصلاحه في مرحلة التصميم الأولية.**

الخلاصة :مسار التصميم التقليدي خطير لأنه يؤخر اكتشاف "أخطاء الأداء" (مثل البطء أو استهلاك الطاقة) إلى النهاية، عندما يكون إصلاحها هو الأكثر تكلفة وتدميراً للمشروع. وهذا يثبت لنا بشكل قاطع حاجتنا إلى مسار تصميم جديد وحديث.

The Traditional Design Flow

- **The consequences:**
  - **Delays in the design process**
    - **Increased design cost**
    - **Delays in time to market ⇒ missed market window**
  - **High cost of failed prototypes**
  - **Bad design decisions taken under time pressure**
    - **Low quality, high cost products**

عواقب مسار التصميم التقليدي

عندما نكتشف أن النموذج الأولي لا يلبي متطلبات الأداء أو الطاقة في نهاية العملية، ندخل في سلسلة من العواقب المدمرة:

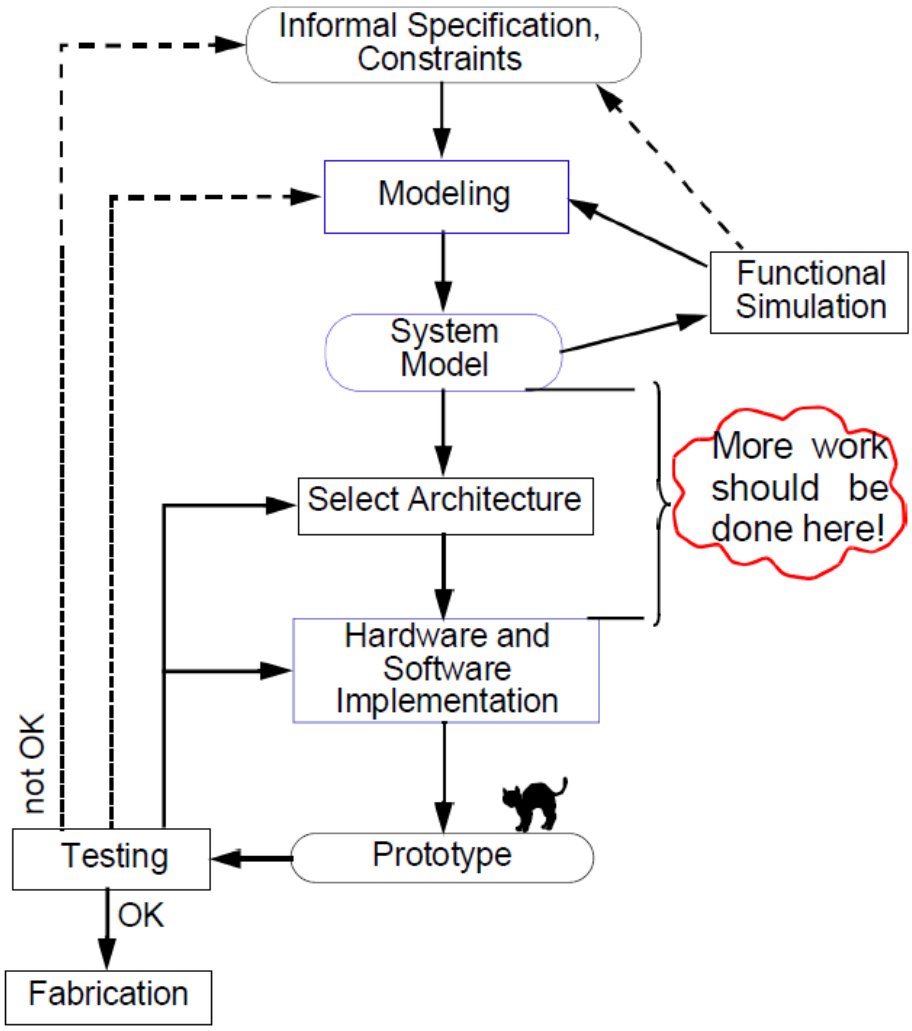
- **التأخير في عملية التصميم**
  - إعادة تصميم أجزاء من النظام تعني **تأخيراً كبيراً** في الجدول الزمني للمشروع.
  - هذا التأخير يؤدي مباشرة إلى **زيادة تكلفة التصميم**، حيث يعمل المهندسون لفترة أطول.

- والأهم من ذلك، أن التأخير في إطلاق المنتج يؤدي إلى **تفويت "نافذة السوق "**، مما يعني فشلاً مالياً للمنتج حتى لو كان عالي الجودة.

- **التكلفة العالية للنماذج الأولية الفاشلة**
  - النموذج الأولي الذي تم بناؤه وأثبت فشله هو الآن قطعة خردة إلكترونية باهظة الثمن.
  - لقد تم إهدار المال على المكونات، تصنيع لوحات الدارات، ووقت العمالة في التجميع. كل هذه التكاليف يجب أن تُدفع مرة أخرى لبناء نموذج أولي جديد بعد إعادة التصميم.

- **قرارات تصميم سيئة تحت ضغط الوقت**
  - هذه هي النتيجة البشرية للتأخير. عندما يدرك الفريق أنه متأخر جداً عن الموعد النهائي، يبدأ الذعر.
  - تحت هذا الضغط، يتوقف المهندسون عن البحث عن الحل **الأفضل** ويبدؤون في البحث عن الحل **الأسرع** أو الحل المؤقت.
  - هذه الحلول السريعة غالباً ما تكون سيئة الهندسة، غير مختبرة جيداً، وتؤدي إلى ظهور أخطاء جديدة.
  - **النتيجة النهائية :**منتج **منخفض الجودة** (بسبب الحلول المتعجلة) و**مرتفع التكلفة** (بسبب إعادة العمل والتأخير).

الخلاصة :مسار التصميم التقليدي ليس فقط غير فعال، بل هو وصفة شبه مؤكدة لفشل المشاريع المعقدة. إنه يخلق أزمة في نهاية المشروع تدفع المهندسين لاتخاذ قرارات سيئة، مما يؤدي إلى إنتاج منتجات سيئة بتكلفة عالية.



أين يكمن الخلل في المسار التقليدي؟

المخطط الذي أمامنا هو نفسه المخطط التقليدي، ولكن مع إضافة ملاحظة تقول: **"يجب بذل المزيد من الجهد هنا"**

هذه الملاحظة تشير إلى الفجوة الكبيرة بين مرحلة "نمذجة النظام (System Model) "ومرحلة "تنفيذ العتاد والبرمجيات". في المسار التقليدي، كانت هذه الفجوة تُملأ بالتخمين والأمل، حيث كنا نختار بنية العتاد ونأمل أن تكون كافية، ثم نبدأ في التنفيذ.

ما هو "المزيد من الجهد" المطلوب؟

"المزيد من الجهد" هنا لا يعني العمل لساعات أطول، بل يعني العمل **بشكل أذكى** عن طريق إضافة خطوات تحليل وتقييم **مبكرة** في هذه المرحلة. هذا الجهد الإضافي في البداية يوفر أضعافاً مضاعفة من الوقت والمال في النهاية. ويشمل هذا الجهد:

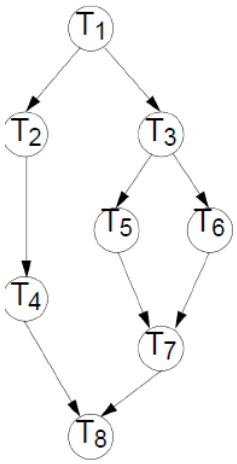
1. **التقديروالتحليل المبكر** :بدلاً من انتظار النموذج الأولي النهائي، يجب أن نستخدم أدوات لتقدير أداء واستهلاك الطاقة للبرنامج على مختلف المعالجات المقترحة.

2. **استكشاف فضاء التصميم (Design Space Exploration) :** يجب ألا نلتزم ببنية واحدة منذ البداية. علينا استكشاف ومقارنة عدة بدائل:

   - ماذا لو استخدمنا معالجاً أسرع ولكنه أغلى؟
   - ماذا لو نفذنا وظيفة حرجة في العتاد المخصص (ASIC) بدلاً من البرمجيات؟
   - ما هي المقايضة بين التكلفة والأداء لكل خيار؟

3. **محاكاة الأداء على مستوى النظام** :نحتاج إلى نوع جديد من المحاكاة. ليس فقط محاكاة وظيفية (تتأكد من أن المنطق صحيح)، بل محاكاة أداء تحاكي نموذج البرنامج وهو يعمل على نموذج للعتاد المختار. هذا يعطينا توقعاً مبكراً جداً لما إذا كنا سنلبي قيود الزمن والطاقة أم لا.

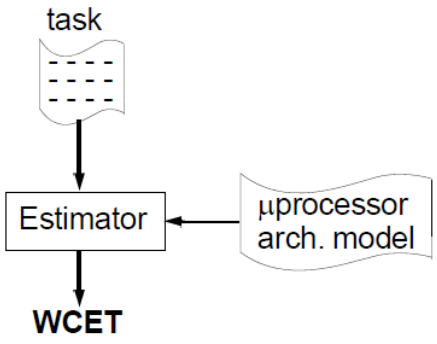
الخلاصة :الفكرة الأساسية هي **نقل مرحلة الاختباروالتحقق من الأداء من نهاية المشروع إلى بدايته** .هذا "الجهد الإضافي" المبذول في التحليل واستكشاف البدائل هو جوهر "مسار التصميم الجديد" الذي سنتعرف عليه، وهو الذي يمنعنا من الوقوع في العواقب التي يسببها المسار التقليدي.



Example

- We have the system model (task graph) which has been validated by simulation.
- We decide on a certain μprocessor μp1, with cost 6.
- For each task the worst case execution time (WCET) when run on μp1 is *estimated*.

Task	WCET
T_1	4
T_2	6
T_3	4
T_4	7
T_5	8
T_6	12
T_7	7
T_8	10



يوضح هذه المثال كيف نبدأ بتحويل النموذج المجرد إلى شيء أقرب إلى التنفيذ الحقيقي.

بعد أن قمنا بتعريف مخطط المهام (Task Graph) ووضعنا قيودنا (الفترة الزمنية والتكلفة)، تأتي الآن الخطوات الهندسية الأولى لحل المشكلة:

- نتأكد أولاً من أن مخطط المهام صحيح. إذا أعطيناه مدخلات معينة، فإنه ينتج المخرجات الصحيحة.
- نختار أول خيار معماري لنا، وهو معالج واحد اسمه µp1. هذه هي أول خطوة في "استكشاف فضاء التصميم".
- أولاً، نفحص قيد التكلفة: تكلفة المعالج هي 6، والميزانية القصوى المسموح بها هي 8. بما أن 8 > 6، إذن هذا الخيار مقبول من ناحية التكلفة.
- **السؤال الآن :**هل هو مقبول من ناحية الأداء؟
- . بدلاً من أن "نأمل" أن يكون المعالج سريعاً بما يكفي، نقوم بعملية **تقدير(Estimation)** . هذا هو جوهر التحليل المبكر
- **أسوأ حالة لزمن التنفيذ :(WCET)** هي أقصى مدة زمنية يمكن أن تستغرقها مهمة معينة لتكتمل على هذا المعالج المحدد.
- **كيف يتم ذلك؟** نستخدم أداة برمجية متخصصة تسمى **(Estimator)** . هذه الأداة تأخذ وصف المهمة (الكود الخاص بها) ونموذجاً لبنية المعالج µp1، وتقوم بتحليل معقد لتحديد أطول مسار ممكن للتنفيذ.

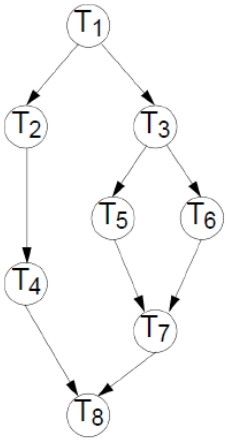
يعرض لنا الجدول نتائج هذا التقدير:

- المهمة T_1ستستغرق 4 وحدات زمنية على الأكثر.
- المهمة T_2ستستغرق 6 وحدات زمنية.
- المهمة T_6هي الأبطأ، حيث ستستغرق 12 وحدة زمنية.

لقد نجحنا في أول اختبار: التكلفة مقبولة .(8 > 6) ولدينا الآن كل المعلومات اللازمة للإجابة على سؤال الأداء :لدينا مخطط يوضح **الاعتماديات** بين المهام، وجدول يوضح **الوقت** الذي تستغرقه كل مهمة.

السؤال الذي يجب أن نجيب عليه الآن هو :بالنظر إلى هذه الأزمنة وهذه الاعتماديات، هل يمكن إنجاز مخطط المهام بأكمله (من T_1إلى T_8) في زمن أقل من الموعد النهائي وهو **42 وحدة زمنية**؟

Example



Task	WCET
T_1	2
T_2	3
T_3	2
T_4	3
T_5	4
T_6	6
T_7	3
T_8	5

We look after a µprocessor which is fast enough: µp2

For each task the WCET, when run on µp2, is estimated.

Using the architecture with µprocessor µp2, after generating a schedule, we got a solution with:

- Execution time: 28 < 42
- Cost: 15 > 8

We have to try with another architecture!

بعد أن فشلت محاولتنا الأولى مع المعالج µp1 (لأنه كان بطيئاً جداً)، ننتقل الآن إلى الخطوة المنطقية التالية في استكشاف التصاميم الممكنة.

المحاولة الثانية: استخدام معالج أسرع

- بما أن المشكلة كانت في الأداء، فإن الحل البديهي هو استخدام معالج أقوى وأسرع. لنطلق عليه اسم µp2.
- كما فعلنا سابقاً، نعيد عملية تقدير زمن التنفيذ لكل مهمة، ولكن هذه المرة على المعالج الجديد.
- نلاحظ في الجدول كيف أن الأزمنة أصبحت أقل بكثير. فمثلاً، المهمة T_1 أصبحت تستغرق وحدتي زمن فقط بدلاً من 4، والمهمة T_6(التي كانت الأبطأ) انخفض زمنها من 12 إلى 6. هذا المعالج الجديد أسرع بشكل واضح.

تحليل النتائج

بعد تحليل أداء مخطط المهام باستخدام المعالج الجديد µp2، حصلنا على النتائج التالية:

- **زمن التنفيذ: 28 < 42**

- تمكن النظام من إنهاء كل مهامه في 28 وحدة زمنية فقط، وهذا أقل بكثير من الموعد النهائي المحدد وهو 42. من ناحية الأداء والسرعة، هذا الحل ممتاز.

- **التكلفة:8 < 15**

- **النتيجة: فشل كارثي** 💣

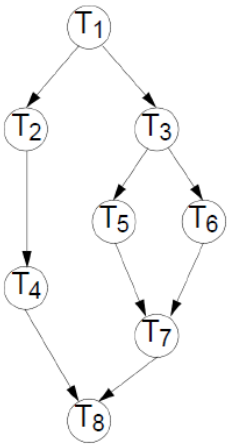
- تكلفة هذا الحل (التي تشمل تكلفة المعالج الأقوى) هي 15 وحدة، بينما ميزانيتنا القصوى هي 8 فقط. هذا الحل يتجاوز الميزانية بشكل كبير.

علينا أن نجرب بنية أخرى.

لقد وقعنا الآن في **مقايضة هندسية** كلاسيكية:

- **الحل الأول (µp1)** : كان رخيصاً بما فيه الكفاية، ولكنه بطيء جداً.
- **الحل الثاني (µp2)** :كان سريعاً بما فيه الكفاية، ولكنه باهظ الثمن.

لا يوجد حل بسيط ومباشر يحقق كلا الشرطين. هذا هو جوهر عمل مهندس النظم المدمجة. لا نيأس هنا، بل نبدأ في التفكير بحلول أكثر إبداعاً.

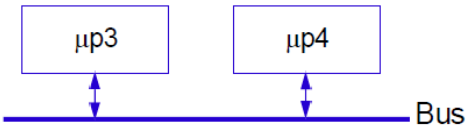


Task	WCET	
	µp3	µp4
T_1	5	6
T_2	7	9
T_3	5	6
T_4	8	10
T_5	10	11
T_6	17	21
T_7	10	14
T_8	15	19

Example

We have to look for a multiprocessor solution

- **In order to meet cost constraints try 2 cheap (and slow) µps:**
  - **µp3: cost 3**
  - **µp4: cost 2**
  - **interconnection bus: cost 1**



For each task the WCET, when run on µp3 and µp4, is estimated.

بعد أن رأينا أن حلول المعالج الواحد البسيطة قد فشلت (إما لكونها بطيئة جداً أو باهظة الثمن)، نصل الآن إلى الحل الهندسي الأكثر إبداعاً وواقعية. وهذه المرة باستخدام **بنية متعددة المعالجات.**

المحاولة الثالثة: الحل متعدد المعالجات

- الفكرة هنا هي: بدلاً من الاعتماد على معالج واحد قوي وباهظ الثمن، ماذا لو استخدمنا معالجين **أرخص ثمناً (و أبطأ)** ونقوم بتوزيع عبء العمل بينهما؟ هذا هو جوهر التصميم المشترك للعتاد والبرمجيات.

- **البنية المقترحة:**
  - معالجان رخيصان: µp3 بتكلفة 3 و µp4 بتكلفة 2
  - ناقل للاتصال بينهما (interconnection bus) بتكلفة 1.

التحليل الأولي: التكلفة والأداء

قبل أن نغوص في تفاصيل الأداء، يجب أن نجيب على السؤال الأول :**هل هذا الحل يلتزم بميزانيتنا؟**

- **التكلفة الإجمالية = تكلفة µp3 +تكلفة µp4 +تكلفة الناقل**
- **التكلفة الإجمالية = 3 + 2 + 1 = 6**
- بما أن ميزانيتنا القصوى هي 8، فإن هذا الحل **ناجح من ناحية التكلفة** لأن (6 < 8)

الآن ننتقل إلى **تحدي الأداء:**

- الجدول الجديد يوضح لنا أداء كل مهمة على كل من المعالجين.
- نلاحظ أن المعالجين ليسا متطابقين . فمثلاً، المهمة T_6 أسرع بكثير على µp3(زمن 17) مقارنة بـ µp4 (زمن 21).

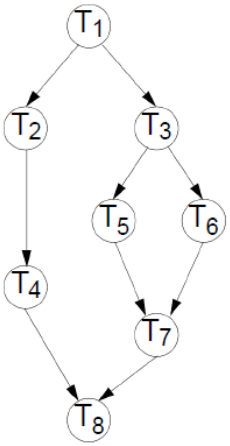
التحدي الهندسي الجديد

لقد نجحنا في اختبار التكلفة، لكننا خلقنا لأنفسنا مشكلة جديدة ومعقدة، وهي من أشهر المسائل في تصميم النظم المدمجة وتعرف بـ**"مسألة تخصيص المهام(Task Mapping/Allocation Problem) ".**

السؤال الآن هو :**كيف نوزع هذه المهام الثماني على المعالجين المتاحين(µp3 و µp4) لنحصل على أسرع أداء ممكن، مع احترام جميع الاعتماديات في مخطط المهام، وهل سيكون هذا الأداء الإجمالي أقل من التوقيت النهائي وهو 42 وحدة زمنية؟**

هذه مسألة استمثال (optimization) معقدة. فمثلاً، هل نضع T_1 و T_2 على نفس المعالج لنتجنب تأخير الاتصال عبر الناقل؟ أم نضعهما على معالجين مختلفين ليستفيدا من التوازي؟

هذا هو المستوى المتقدم من التصميم على مستوى النظام، حيث نقوم باستكشاف المقايضات بين توزيع المهام، الأداء، والتكلفة.



Task	WCET	
	µp3	µp4
T_1	5	6
T_2	7	9
T_3	5	6
T_4	8	10
T_5	10	11
T_6	17	21
T_7	10	14
T_8	15	19

Example

Now we have to *map* the tasks to processors:

µp3: T_1, T_3, T_5, T_6, T_7, T_8.
µp4: T_2, T_4.

If communicating tasks are mapped to different processors, they have to communicate over the bus.

Communication time has to be estimated; it depends on the amount of bits transferred between the tasks and on the speed of the bus.

Estimated communication times:

$C_{1\text{-}2}$: 1
$C_{4\text{-}8}$: 1

. بعد أن اخترنا المكونات وتأكدنا أنها تلبي قيد التكلفة، تأتي الآن المهمة البرمجية الصعبة.

تخصيص المهام وحساب تكلفة الاتصال

- **التخصيص (Mapping):** هي عملية اتخاذ القرار بشأن أي مهمة ستعمل على أي معالج. هذا قرار حاسم يؤثر على أداء النظام بأكمله.
- الشريحة تعرض لنا قراراً اتخذه فريق التصميم:
  - **المعالج µp3 :** سيقوم بتنفيذ معظم المهام.(T1, T3, T5, T6, T7, T8)
  - **المعالج µp4 :** سيقوم بتنفيذ مهمتين فقط.(T2, T4)

- **إذا تم تخصيص مهام متصلة لمعالجات مختلفة، فعليها التواصل عبر الناقل.** عندما تحتاج مهمة على معالج µp3 لإرسال نتيجتها إلى مهمة على µp4، فإن هذه البيانات يجب أن تنتقل عبر الناقل (Bus)، وهذا الانتقال **ليس مجانياً، بل يستغرق وقتاً**.

- **تقدير أزمنة الاتصال :**

  - تماماً كما قدرنا زمن تنفيذ المهام، يجب أن نقدر زمن الاتصال عبر الناقل. الشريحة تعطينا تقديرات للحالات التي يحدث فيها هذا الاتصال بناءً على التخصيص الذي اخترناه:

    - **C_{1-2}: 1**: بما أن T_1 على µp3 و T_2 على µp4، فإن إرسال البيانات من T_1 إلى T_2 سيستغرق **وحدة زمنية واحدة**.

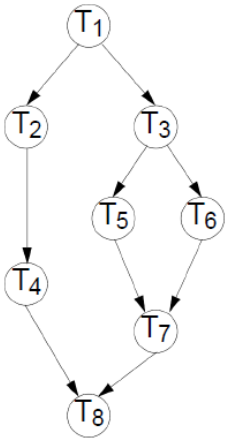
    - **C_{4-8}: 1**: بما أن T_4 على µp4 و T_8 على µp3، فإن إرسال البيانات من T_4 إلى T_8 سيستغرق أيضاً **وحدة زمنية واحدة**.

الخلاصة والخطوة النهائية :لقد أصبحت كل المعلومات الضرورية متوفرة لدينا:

1. الاعتماديات بين المهام (من المخطط).
2. زمن تنفيذ كل مهمة على معالجها المخصص (من الجدول).
3. زمن الاتصال الإضافي لأي بيانات تعبر الناقل.

المهمة الأخيرة الآن هي أخذ كل هذه المعلومات، ووضع **جدول زمني (schedule)** دقيق لتنفيذ هذه المهام، وحساب زمن التنفيذ الكلي النهائي. هذا سيجيب على سؤالنا: هل هذا الحل متعدد المعالجات، بهذا التوزيع المحدد، سيلبي التوقيتالنهائي وهو 42 وحدة زمنية؟

Example

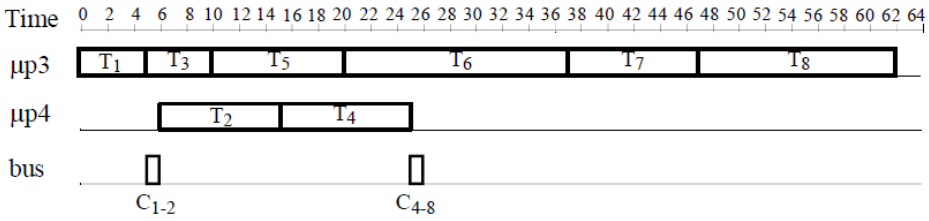


Task	WCET	
	µp3	µp4
T_1	5	6
T_2	7	9
T_3	5	6
T_4	8	10
T_5	10	11
T_6	17	21
T_7	10	14
T_8	15	19

µp3: T_1, T_3, T_5, T_6, T_7, T_8.
µp4: T_2, T_4.

Estimated communication times:
C_{1-2}: 1
C_{4-8}: 1

We generate a schedule:



We have exceeded the allowed execution time (42)!

المحاولة الثالثة: النتيجة النهائية للحل متعدد المعالجات

بعد أن اخترنا بنية المعالجين µp3 و µp4 وخصصنا المهام لكل منهما، وقدرنا أزمنة الاتصال عبر الناقل، سنقوم الآن بإنشاء **جدول زمني (schedule)** لتنفيذ كل المهام.

هذا المخطط الزمني يوضح لنا أي مهمة تعمل على أي معالج في كل لحظة:

- **الصف µp3** :يوضح المهام التي ينفذها المعالج الأول عبر الزمن.
- **الصف µp4** :يوضح المهام التي ينفذها المعالج الثاني.
- **الصف bus** :يوضح متى يتم استخدام الناقل لنقل البيانات بين المعالجين.

إذا تتبعنا المخطط، نرى كيف أن T_1 تبدأ عند الزمن صفر على µp3. وبعد انتهائها، يتم استخدام الناقل (C_{1-2}) لنقل نتيجتها إلى µp4 لكي تبدأ المهمة T_2 وفي نفس الوقت، يبدأ µp3 في تنفيذ T_3، وهكذا.

الحكم النهائي

- من خلال تتبع أطول مسار زمني في هذا المخطط (المسار الحرج)، والذي يأخذ في الاعتبار أزمنة التنفيذ وأزمنة الاتصال والاعتماديات، نصل إلى النتيجة النهائية.

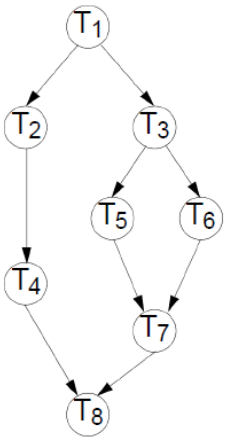
- **لقد تجاوزنا زمن التنفيذ المسموح به(42)**
    - كما يوضح المخطط، المهمة الأخيرة T8 تنتهي عند زمن يقارب **62 وحدة زمنية**.
    - الموعد النهائي الصارم الذي كان علينا الالتزام به هو **42 وحدة زمنية**.
    - بما أن 42 < 62، فإن هذا الحل أيضاً **قد فشل**.

الخلاصة النهائية للمثال

1. جربنا حلاً **رخيصاً وبطيئاً** (µp1)، ففشل في اختبار السرعة.
2. جربنا حلاً **سريعاً ومكلفاً** (µp2)، ففشل في اختبار الميزانية.
3. جربنا حلاً **إبداعياً متعدد المعالجات ورخيصاً** (µp3+µp4)، ولكنه فشل أيضاً في اختبار السرعة بعد التحليل الدقيق.

هذا يوضح أن **تصميم النظم المدمجة على المستوى النظمي هو عملية معقدة من الاستكشاف والمقايضة**. الحل الصحيح قد لا يكون واضحاً على الفور. قد يتطلب الحل الناجح تجربة توزيع مختلف للمهام، أو استخدام ناقل أسرع، أو ربما الحل الأمثل هو بنية هجينة تستخدم المعالج الرخيص مع مسرع عتادي مخصص للمهام البطيئة. هذه هي طبيعة عملنا كمهندسي نظم مدمجة: البحث عن الحل الأمثل الذي يوازن بين كل القيود المتضاربة.

Example



Task	WCET	
	μp3	μp4
T_1	5	6
T_2	7	9
T_3	5	6
T_4	8	10
T_5	10	11
T_6	17	21
T_7	10	14
T_8	15	19

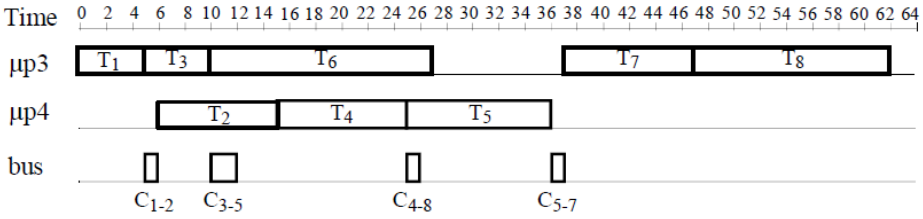
Try a new mapping; T_5 to μp4, in order to increase parallelism.

Two new communications are introduced, with estimated times:

$C_{3\text{-}5}$: 2

$C_{5\text{-}7}$: 1

We generate a schedule:



The execution time is still 62, as before!

المحاولة الرابعة: زيادة التوازي

- **الفكرة**: في المحاولة السابقة، كان المعالج μp3 يقوم بمعظم العمل، بينما μp4 كان أقل انشغالاً. لننقل إحدى المهام الطويلة، وهي T5، من المعالج المزدحم μp3 إلى المعالج الأقل ازدحاماً μp4. الهدف هو **زيادة التوازي** و**تحسين** توزيع المهام.
- **التكلفة الخفية**: لكن، كما تعلمنا، لا يوجد قرار تصميم بدون مقايضة. نقل المهمة T5 خلق لنا **تكاليف اتصال جديدة** عبر الناقل، لأن المهام التي تتواصل معها (T3 و T7) موجودة على المعالج الآخر:
  - الاتصال من T3 إلى T5 يكلف وحدتي زمن ($C_{3\text{-}5}$: 2)
  - الاتصال من T5 إلى T7 يكلف وحدة زمن واحدة ($C_{5\text{-}7}$: 1)

تحليل النتيجة النهائية

بعد إنشاء جدول زمني جديد يأخذ في الحسبان هذا التوزيع الجديد وتكاليف الاتصال الإضافية، نصل إلى النتيجة النهائية:

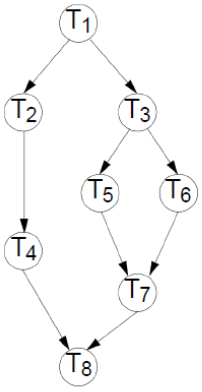
زمن التنفيذ لا يزال 62، كما كان من قبل.

لماذا لم ينجح هذا الحل ؟

على الرغم من أننا قمنا بتحسين توزيع الحمل على المعالجات، فإن **تكلفة الاتصال الإضافية** التي أحدثناها أكلت كل المكاسب التي كنا نأمل في تحقيقها من التوازي.

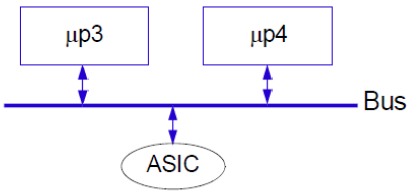
هذا درس بليغ في تصميم النظم الموزعة :**الاتصال ليس مجانياً** .أحياناً، يكون من الأسرع إبقاء مهمتين على نفس المعالج (حتى لو كان أبطأ) لتجنب التأخير الزمني الناتج عن إرسال البيانات بينهما، بدلاً من وضعهما على معالجين منفصلين.

الخلاصة النهائية للمثال :هذا المثال الطويل يوضح أن تصميم النظم على المستوى النظمي هو عملية استمثال (optimization)معقدة. لا توجد إجابات سهلة. التصميم الناجح يتطلب تحليلاً دقيقاً لأزمنة التنفيذ، تكاليف الاتصال، واعتماديات المهام للعثور على الحل الأمثل الذي يلبي كل القيود. هذا هو فن وعلم التصميم المشترك للأنظمة المدمجة.



Task	WCET	
	µp3	µp4
T_1	5	6
T_2	7	9
T_3	5	6
T_4	8	10
T_5	10	11
T_6	17	21
T_7	10	14
T_8	15	19

Example



- **Possible solutions:**
  - **Change µproc. µp3 with faster one ⇒ cost limits exceed**
  - **Implement part of the functionality in hardware as an ASIC**
- **New architecture**

  Cost of ASIC: 1
- **Mapping**

  µp3: T_1, T_3, T_6, T_7.

  µp4: T_2, T_4, T_5.

  ASIC: T_8 with estimated WCET= 3
  - **New communication, with estimated time: $C_{7\text{-}8}$: 1**

المحاولة الخامسة والأخيرة: الحل الهجين (الناجح)

بعد أن فشلت كل محاولاتنا السابقة (المعالج الرخيص كان بطيئاً، والمعالج السريع كان مكلفاً، والمعالجان معاً كانا أيضاً بطيئين بسبب الاتصالات)، نصل إلى الحل الذي يوضح جوهر التصميم المشترك للعتاد والبرمجيات.

- **الحل المقترح:**
  - بدلاً من محاولة حل كل شيء بالمعالجات (البرمجيات)، نقوم بتنفيذ الجزء الأكثر تعقيداً واستهلاكاً للوقت من النظام في **عتاد مخصص** وهو (**ASIC دارة متكاملة محددة التطبيق**).

تحليل البنية الجديدة

- **البنية الهندسية(New architecture) :**
  - أصبح لدينا الآن نظام هجين يتكون من: المعالجين الرخيصين µp3 و µp4، وبجانبهما قطعة عتاد جديدة هي ASIC، وجميعهم متصلون عبر الناقل(Bus) .
- **تحليل التكلفة:**
  - تكلفة µp3 = 3
  - تكلفة µp4 = 2
  - تكلفة ASIC = 1
  - تكلفة الناقل = 1
  - **التكلفة الإجمالية = 3 + 2 + 1 + 1 = 7.**
  - بما أن ميزانيتنا القصوى هي 8، فإن هذا الحل **ناجح من ناحية التكلفة .**
- **تخصيص المهام والأداء(Mapping) :**
  - تم توزيع معظم المهام على المعالجين الرخيصين.
  - لكن المهمة الأخيرة والحرجة T_8، التي كانت تسبب تأخيراً كبيراً، تم تخصيصها للـ ASIC
  - نلاحظ الأداء الفائق: زمن تنفيذ T_8على الـ ASIC هو **3 وحدات زمنية فقط** مقارنة بـ 15 على µp3 هذا تسريع هائل.
  - بالطبع، هذا خلق لنا تكلفة اتصال جديدة(C_{7-8}: 1) ، لأن T_7على µp3 تحتاج لإرسال نتيجتها إلى T_8 على الـ ASIC

الخلاصة النهائية :التخفيض الهائل في زمن تنفيذ المهمة T_8(من 15 إلى 3) سيعوض بسهولة تكلفة الاتصال الإضافية (1 وحدة زمنية)، وسيؤدي إلى زمن تنفيذ كلي أقل بكثير من الموعد النهائي وهو 42.

هذا هو **التصميم المشترك للعتاد والبرمجيات** في أفضل صورة. لقد قمنا بتحليل النظام، حددنا المهمة الحرجة (T_8) وقمنا بتصميم حل هجين يضع المهام العامة على معالجات برمجية رخيصة، ويسرّع المهمة الحرجة باستخدام عتاد مخصص. لقد حققنا هدفي **الأداء والتكلفة** معاً، وهو ما فشلت فيه كل الحلول السابقة.

Example

What did we achieve?

- We have selected an architecture.
- We have mapped tasks to the processors and ASIC.
- We have elaborated a a schedule.

Extremely important!!!
Nothing has been built yet.
All decisions are based on simulation and estimation.

- Now we can go and do the software and hardware implementation, with a high degree of confidence that we get a correct prototype.

ما الذي حققناه؟

- **لقد اخترنا بنية هندسية (architecture)** : وهي البنية الهجينة التي تجمع بين المعالجات الرخيصة والعتاد المخصص(ASIC).
- **لقد خصصنا المهام (mapped tasks)** : قررنا أي مهمة ستعمل على أي معالج وأي مهمة ستعمل على الـ(ASIC).
- **لقد وضعنا جدولاً زمنياً (elaborated a schedule)** : وهو الجدول الزمني (الضمني) الناجح الذي يلبي كل القيود.

النقطة الأهم على الإطلاق:

- **لم يتم بناء أي شيء بعد.**
- **كل القرارات مبنية على المحاكاة والتقدير.**

أهم إنجاز حققناه هو أننا وصلنا إلى تصميم ناجح ومثالي **دون أن نبني قطعة عتاد واحدة أو نكتب سطراً واحداً من الكود النهائي** .كل عملنا كان عبارة عن نمذجة، ومحاكاة، وتقدير للأداء والتكلفة في مرحلة مبكرة جداً.

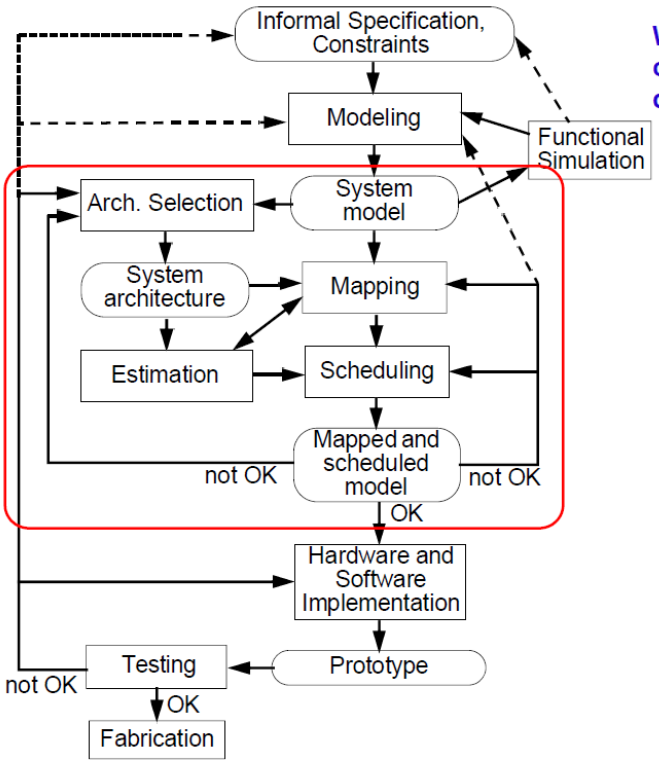
الخطوة التالية: التنفيذ بثقة

- **الآن يمكننا الذهاب وتنفيذ البرمجيات والعتاد، وبدرجة عالية من الثقة بأننا سنحصل على نموذج أولي صحيح.**

وهذا هو الفرق الجوهري بين هذا المسار والمسار التقليدي.

- في المسار **التقليدي**، كنا نصل إلى مرحلة بناء النموذج الأولي ونحن "نأمل" أن يعمل.
- في هذا المسار **الحديث**، نصل إلى هذه المرحلة ونحن "نعرف" بدرجة عالية من اليقين أنه سيعمل، لأننا قمنا بحل كل مشاكل الأداء والتكلفة والتصميم على الورق (أو في المحاكي) أولاً.

لقد استثمرنا الجهد في التحليل المبكر، وسنجني الآن ثمار هذا الجهد: عملية تنفيذ أسرع، وأرخص، وأكثر موثوقية.



What is the essential difference compared to the "traditional" design flow?

- **The inner loop which is performed before the hardware/ software implementation.**

  This loop is performed several times as part of the *design space exploration*. Different architectures, mappings and schedules are explored, before the actual implementation and prototyping.

- **We get highly optimized good quality solutions in short time. We have a good chance that the outer loop, including prototyping, is not repeated.**

سنعرض هنا **مسار التصميم الحديث والذكي** و الذي يقدم لنا الحل لكل المشاكل التي رأيناها في مسار التصميم التقليدي.

ما هو الفرق الجوهري عن المسار التقليدي؟

الفرق الأساسي والجوهري، هو وجود **حلقة داخلية (inner loop)** يتم تنفيذها بشكل متكرر **قبل** البدء في مرحلة التنفيذ وبناء النموذج الأولي المكلفة.

هذه الحلقة هي قلب عملية التصميم الحديثة، وتسمى (**استكشاف فضاء التصميم**.**Design Space Exploration**)

كيف تعمل هذه الحلقة الذكية؟

بدلاً من اختيار العتاد ثم الأمل في أن يعمل، نقوم بالخطوات التالية بشكل متكرر:

1. **نختار بنية مقترحة (Arch. Selection)** :مثلاً، نختار المعالج الرخيص µp1
2. **نقدّر الأداء (Estimation)** :نحسب زمن تنفيذ كل مهمة على هذا المعالج.
3. **نخصص المهام (Mapping)** : نقرر أي مهمة ستعمل على أي مكون.
4. **نضع جدولاً زمنياً (Scheduling)** : ننشئ خطة تنفيذ.
5. **نحصل على نموذج مجدول (Mapped and scheduled model)** : هذا بمثابة **"نموذج أولي افتراضي "** .إنه نموذج رياضي يصف كيف سيتصرف برنامجنا على العتاد الذي اخترناه.
6. **التحقق المبكر** :نقوم بمحاكاة هذا النموذج الافتراضي لنرى هل يلبي قيود الزمن والطاقة.
   - إذا كانت النتيجة **"not OK"**(فشل)، فإننا لا نخسر شيئاً. نحن ببساطة نعود ونغير قرارنا (نجرب تخصيصاً مختلفاً للمهام، أو نختار بنية مختلفة تماماً) ونعيد هذه الحلقة السريعة والرخيصة.
   - نكرر هذه العملية عدة مرات، مستكشفين خيارات مختلفة، حتى نصل إلى حل تكون نتيجته **"OK"**

الفوائد الهائلة للمسار الحديث

- **نحصل على حلول عالية الجودة ومثالية في وقت قصير.**
  - لأننا نستطيع تجربة عشرات البدائل افتراضياً، يمكننا الوصول إلى أفضل حل ممكن يوازن بين التكلفة والأداء والطاقة.
- **لدينا فرصة جيدة جداً أن الحلقة الخارجية لن تتكرر.**
  - هذه هي الفائدة الأكبر. عندما نخرج أخيراً من "الحلقة الذكية" وننتقل إلى مرحلة بناء النموذج الأولي الفعلي، نكون قد قمنا بكل التحليلات اللازمة مسبقاً.
  - هذا يعطينا **ثقة عالية جداً** بأن النموذج الأولي الحقيقي سينجح في الاختبارات من المرة الأولى، مما يوفر كمية هائلة من الوقت والمال ويجنبنا العواقب التي يسببها المسار التقليدي.

The Design Flow

- **Formal verification**
  - **It is impossible to do an exhaustive verification by simulation! Especially for safety critical systems formal verification is needed.**

- **Hardware/Software codesign**
  - **During the mapping/scheduling step we also decide what is going to be executed on a programmable processor (software) and what is going into hardware (ASIC, FPGA).**
  - **During the implementation phase, hardware and software components have to be developed in a coordinated way, keeping care of their consistency (hardware/software cosimulation)**

هنا نسلط الضوء على مبدأين أساسيين في مسار التصميم الحديث، وهما يمثلان خلاصة خبرات المهندسين في بناء أنظمة معقدة وموثوقة.

التحقق الرسمي(Formal Verification)

- **من المستحيل إجراء تحقق شامل باستخدام المحاكاة.**
  - هذه حقيقة مهمة جداً .**المحاكاة والاختبار**، مهما كانا مكثفين، لا يمكنهما تغطية كل السيناريوهات والحالات الممكنة في نظام معقد. الاختبار يمكنه أن يثبت وجود الأخطاء، ولكنه **لا يستطيع إثبات عدم وجودها**.
  - **مثال توضيحي** :اختبار جسر عن طريق قيادة بضع مئات من السيارات عليه لا يثبت أنه لن ينهار تحت ظرف نادر جداً.
  - لذلك، **للأنظمة الحرجة للسلامة**، نحن بحاجة إلى **التحقق الرسمي** .وهو استخدام أساليب رياضية ومنطقية صارمة **لإثبات** أن تصميمنا يلتزم بخصائص السلامة المطلوبة في **كل الحالات الممكنة**، تماماً مثلما يثبت عالم الرياضيات نظرية ما.

التصميم المشترك للعتاد والبرمجيات(Hardware/Software Co-design)

هذا هو المبدأ الأساسي الذي يسمح لنا ببناء أنظمة مثالية (optimized) وهو يتكون من مرحلتين:

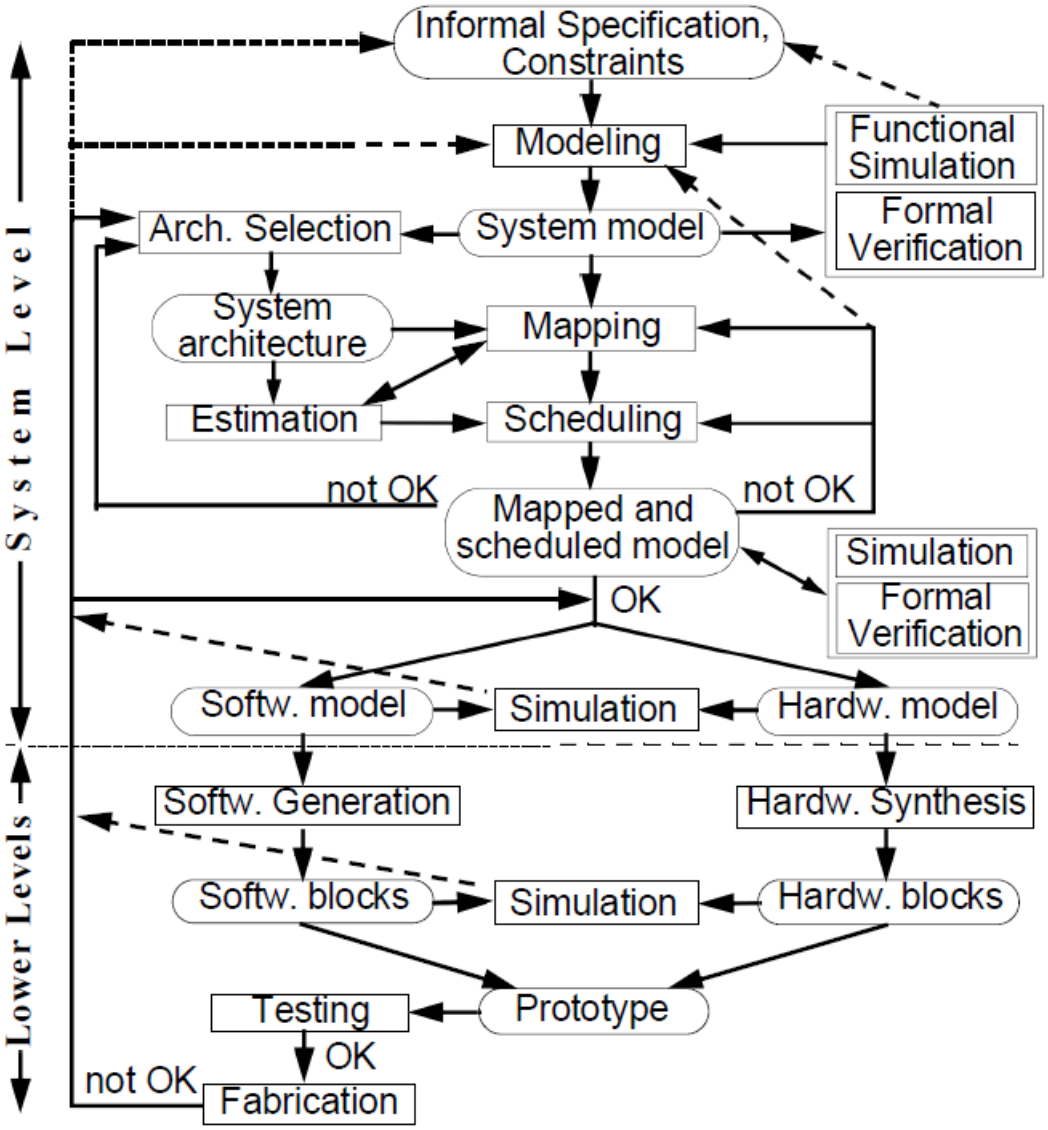
1. **مرحلة التقسيم(Partitioning)**

- **خلال خطوة تخصيص وجدولة المهام، نقرر أيضاً ما الذي سيتم تنفيذه على معالج قابل للبرمجة (برمجيات) وما الذي سيتحول إلى عتاد.**
- هذه هي الخطوة الاستراتيجية التي رأيناها في مثالنا الأخير. نقوم بتحليل النظام ونقرر: ما هي الوظائف التي تحتاج إلى مرونة ويمكن تنفيذها كبرنامج؟ وما هي الوظائف التي تحتاج إلى سرعة قصوى وكفاءة طاقة ويجب تحويلها إلى عتاد مخصص مثل **ASIC**؟

2. **مرحلة التنفيذ المنسق:(Coordinated Implementation)**

- **يجب تطوير مكونات العتاد والبرمجيات بطريقة منسقة.**
- قرار التقسيم ليس كافياً. يجب أن يعمل فريق العتاد وفريق البرمجيات معاً بشكل وثيق طوال عملية التطوير.
- لضمان أن المكونين سيعملان معاً بسلاسة، نستخدم تقنية تسمى **المحاكاة المشتركة (hardware/software co-simulation)**. في هذه التقنية، نقوم بتشغيل محاكاة للعتاد الذي يتم تصميمه مع البرنامج الفعلي الذي يتم كتابته في نفس الوقت، مما يسمح لنا باكتشاف وتصحيح أي أخطاء في التكامل بينهما في وقت مبكر جداً.

الخلاصة :مسار التصميم الحديث يعتمد على هذين العمودين :**التحقق الرسمي** لنضمن أن تصميمنا آمن، و**التصميم المشترك** لنضمن أن تنفيذنا لهذا التصميم هو الأمثل والأكثر كفاءة.



هذه المخطط يمثل الخريطة الكاملة والمفصلة لمسار التصميم الحديث. هي تجمع كل المفاهيم التي ناقشناها—من النمذجة المجردة إلى التنفيذ الفعلي—في صورة واحدة متكاملة.

1- المستوى النظمي - (System Level) الجزء العلوي

هذا هو ملعب المهندسين المعماريين للنظام. هنا يتم اتخاذ كل القرارات الاستراتيجية الكبرى **قبل** كتابة أي كود نهائي أو بناء أي قطعة عتاد.

- **الحلقة الداخلية (Inner Loop)** : هذا هو ما تعلمناه عن مسار التصميم الجديد. نبدأ بنموذج للنظام (System model)ثم ندخل في حلقة **استكشاف فضاء التصميم**:

1. نختار بنية مقترحة(Arch. Selection)
2. نقدر أداءها(Estimation)
3. نخصص المهام عليها(Mapping)
4. نضع جدولاً زمنياً(Scheduling)

- **التحقق المبكر** :في هذه المرحلة، نستخدم **المحاكاة (Simulation)** و**التحقق الرسمي Formal) (Verification**على نماذج مجردة لنتأكد من أن تصميمنا المقترح يلبي كل القيود. إذا لم يكن كذلك (not OK)، نعيد الكرة بتغيير قراراتنا.

- **النتيجة** :عندما نخرج من هذه الحلقة بنتيجة OK، يكون لدينا "مخطط بناء" شبه مؤكد النجاح.

2- المستويات الدنيا - (Lower Levels) الجزء السفلي

بعد الحصول على موافقة من المستوى النظمي، ننتقل إلى مرحلة التنفيذ الفعلي. نلاحظ كيف ينقسم المسار هنا إلى طريقين متوازيين:

- **مسار البرمجيات(Software) :**
  - يتم أخذ نموذج البرمجيات المجرد (Softw. model) واستخدامه **لتوليد الكود البرمجي الفعلي** (Softw. Generation)، والذي ينتج عنه المكونات البرمجية النهائية (Softw. blocks)

- **مسار العتاد(Hardware) :**
  - يتم أخذ نموذج العتاد المجرد (Hardw. model) وإدخاله في أدوات **التوليف (Synthesis)** لتوليد الوصف التفصيلي للعتاد (مثل كودVHDL)، والذي ينتج عنه المكونات العتادية النهائية (Hardw. blocks)

مرحلة التكامل النهائية

أخيراً، يجب أن يلتقي هذان المساران:

1. **المحاكاة المشتركة (Co-simulation)** يتم تشغيل محاكاة تجمع بين المكونات البرمجية والعتادية معاً للتحقق من تكاملهما قبل بناء أي شيء مادي.

2. **النموذج الأولي والاختبار (Prototype & Testing)** : يتم دمج المكونات في نموذج أولي حقيقي واختباره.

3. **التصنيع (Fabrication)** : إذا نجح كل شيء، ينتقل المنتج إلى التصنيع بكميات كبيرة.

الخلاصة :هذا المخطط يوضح لنا أن التصميم الحديث هو عملية منظمة تبدأ من الأعلى إلى الأسفل. نستثمر وقتاً وجهداً كبيراً في **المستوى النظمي** في التحليل والمحاكاة المجردة لنضمن اتخاذ القرارات الصحيحة، مما يقلل بشكل هائل من المخاطر والتكاليف في **المستويات الدنيا** المكلفة الخاصة بالتنفيذ الفعلي.