Hierarchical Concurrent Finite State Machines

- **There are two important mechanisms that reduce the size of an FSM model:**
  1. **Hierarchy**
  2. **Concurrency**

Important

- **Using Hierarchy and concurrency we only reduce the size of the graphical or textual model; the intrinsic complexity - the number of states of the actual system - cannot be reduced.**
- **However, the difficulty of realising the model is drastically reduced.**

بعد أن رأينا كيف يمكن للأنظمة المتزامنة أن تؤدي إلى "انفجار الحالات" في نموذج آلة الحالة المحدودة المسطحة، تقدم لنا هذه الشريحة الحل :**آلات الحالة المحدودة الهرمية والمتزامنة (Hierarchical Concurrent Finite State Machines)**

هذا النموذج، الذي يُعرف غالباً باسم **مخططات الحالة(Statecharts)** ، هو أسلوب قوي لإدارة التعقيد.

الآليات الأساسية

تقدم الشريحة آليتين مهمتين لتقليل حجم وتعقيد **النموذج** الذي نبنيه:

1. **الهرمية (Hierarchy) :**
   - تسمح بأن تحتوي حالة معينة على آلة حالة أخرى أصغر بداخلها. هذا يشبه خاصية "التكبير" أو "zoom-in" على حالة ما.
   - **مثال :**يمكن أن تكون لدينا حالة اسمها "القيادة" في سيارة. بدلاً من إنشاء حالات منفصلة مثل "القيادة في الغيار الأول" و"القيادة في الغيار الثاني"، يمكننا أن نجعل حالة "القيادة" تحتوي بداخلها على آلة حالة أبسط تدير التغيير بين الغيارات {الأول، الثاني، الثالث}.
   - **الفائدة :**هذا يقلل من تكرار الانتقالات وينظم النموذج بشكل منطقي.
2. **التزامن(Concurrency) :**
   - تسمح بتشغيل عدة آلات حالة بشكل متوازٍ ومستقل في نفس الوقت.

- **مثال :** في مثالنا السابق الذي أدى إلى انفجار الحالات، كنا نريد تشغيل "آلة المهمة أ" بالتوازي مع "آلة المهمة ب". بدلاً من دمجهم في آلة واحدة ضخمة، يسمح لنا هذا المفهوم بإبقائهما منفصلتين ومتجاورتين، مع إمكانية التنسيق بينهما عند الحاجة.
- **الفائدة :** هذا يتجنب بشكل مباشر النمو الأسيّ للحالات الذي رأيناه، حيث إن عدد الحالات في النموذج يصبح **مجموع** حالات الآلات المتوازية وليس **حاصل ضربها**.

نقطة مهمة: النموذج مقابل التعقيد الجوهري

- **باستخدام الهرمية والتزامن، نحن فقط نقلل من حجم النموذج الرسومي أو النصي.**
  - أي أن المخطط الذي نرسمه أو الكود الذي نكتبه يصبح أصغر وأبسط وأكثر قابلية للفهم بشكل كبير. فبدلاً من رسم 9 حالات وعشرات الانتقالات، نرسم آلتين صغيرتين لكل منهما حالتان أو ثلاث.
- **التعقيد الجوهري (intrinsic complexity) - وهو العدد الفعلي لحالات النظام الكلية - لا يمكن تقليله.**
  - في مثال المصباحين، سيظل النظام يحتوي على 4 حالات كلية ممكنة (مطفأ/مطفأ، مطفأ/مضاء، إلخ). لا يمكننا تغيير هذه الحقيقة الفيزيائية. الهرمية والتزامن لا تغيران عدد الحالات الفعلية التي يمكن للنظام أن يكون فيها.
- **"ومع ذلك، فإن صعوبة تحقيق النموذج تقل بشكل كبير".**
  - هذا هو المكسب الحقيقي. على الرغم من أن النظام لا يزال معقداً بطبيعته، فإن قدرتنا كمهندسين على **وصف هذا التعقيد وفهمه وإدارته** تصبح أسهل بكثير.
  - **مثال توضيحي :**لنأخذ خريطة العالم. يمكننا محاولة رسم خريطة مسطحة عملاقة للعالم كله (وهو ما يعادل FSM المسطحة). أو يمكننا استخدام أطلس (كتاب خرائط) يحتوي على صفحة لكل قارة (تزامن)، وداخل كل صفحة خريطة مكبرة لكل دولة (هرمية). العالم (التعقيد الجوهري) لم يتغير، لكن الأطلس يجعل التعامل مع هذه المعلومات وفهمها (تحقيق النموذج) أسهل بما لا يقاس.

Hierarchical Concurrent Finite State Machines

Hierarchy

- A single state S can represent an enclosed state machine F:

  Being in state S means that state machine F is active ⇒ the system is in one of the states of the state machine F (*or states*).

Concurrency

- Two or more state machines are viewed as being simultaneously active ⇒ the system is in one of each parallel state machine simultaneously (*and* states).

هذه الشريحة تقدم لنا تعريفاً أكثر دقة للآليتين اللتين تحدثنا عنهما للتو، الهرمية والتزامن، وتعرفنا على مصطلحين مهمين جداً في هذا السياق:" **حالات أو (OR states)** "و" **حالات و(AND states)** "

الهرمية(Hierarchy)

- **"A single state S can represent an enclosed state machine F:" يمكن لحالة واحدة S أن تمثل آلة حالة F مغلقة بداخلها .**
  - هذا هو جوهر الهرمية. الحالة لم تعد مجرد نقطة بسيطة، بل يمكن أن تكون "صندوقاً" أو "حاوية" لآلة حالة كاملة بتفاصيلها الخاصة.
- **"Being in state S means that state machine F is active ⇒ the system is in one of the states of the state machine F (*or* states)." الوجود في الحالة S يعني أن آلة الحالة F نشطة <-- النظام موجود في إحدى حالات آلة الحالة F حالات أو**
  - **ماذا يعني ذلك؟** عندما نقول إن النظام في الحالة الكلية S (super-state) ، فنحن لا نصف حالته بدقة كافية .S هي مجرد عنوان. الوجود في S يعني أن آلة الحالة الداخلية F أصبحت نشطة، وبما أن F يجب أن تكون في إحدى حالاتها الداخلية (لنفترض أنها f1, f2, f3)، فإن الحالة الحقيقية للنظام هي:
    - في الحالة f1 أو في الحالة f2 أو في الحالة f3
  - لهذا السبب، تسمى الحالة الكلية S " **حالة أو(OR-state)** "، لأنها تمثل علاقة "أو" المنطقية بين حالاتها الفرعية.

- **مثال :**إذا كانت حالتي هي "في العمل"، فهذا يعني أنني "في المكتب "أو "في غرفة الاجتماعات "أو "في الكافتيريا". حالة "في العمل" هي OR-state

التزامن(Concurrency)

- **"Two or more state machines are viewed as being simultaneously active" يُنظر إلى آلتين أو أكثر من آلات الحالة على أنها نشطة في نفس الوقت**
  - هذا هو جوهر التزامن. لدينا آلات حالة منفصلة تعمل جنباً إلى جنب كأجزاء مستقلة من النظام.
- **" the system is in one state of each parallel state machine simultaneously (*and* states)." النظام موجود في حالة واحدة من كل آلة حالة متوازية في نفس الوقت (حالات و).**
  - **ماذا يعني ذلك؟** لوصف "الحالة الكلية" للنظام، يجب أن نحدد حالة كل آلة من الآلات المتوازية. الحالة الكلية هي تركيبة من حالات الأجزاء.
  - إذا كان لدينا "آلة المصباح" حالاتها: {مضاء، مطفأ} تعمل بالتوازي مع "آلة المروحة" حالاتها: {تعمل، متوقفة}، فإن الحالة الكلية للنظام هي:
    - (حالة المصباح و حالة المروحة).
  - مثلاً، يمكن أن يكون النظام في الحالة ("المصباح مضاء " و "المروحة تعمل").
  - لهذا السبب، تسمى هذه الحالة الكلية **"حالة و(AND-state) "**، لأنها تمثل علاقة "و" المنطقية بين حالات المكونات المتوازية. وهذا يوضح لنا رياضياً سبب "انفجار الحالات": عدد الحالات الكلية هو حاصل ضرب عدد حالات كل آلة تعمل بالتوازي.

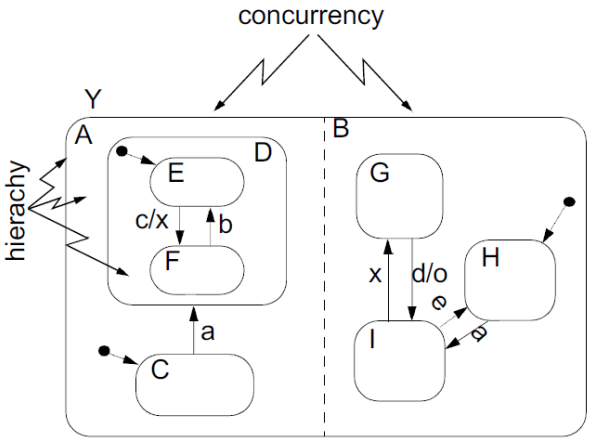
الخلاصة والفرق الجوهري:

- **الهرمية (حالات OR)**: تضيف **العمق** للنموذج. تتعلق بتجريد التفاصيل. نحن في حالة كلية واحدة، مما يعني أننا في **واحدة من** حالاتها الفرعية.
- **التزامن (حالات AND)**: يضيف **العرض** للنموذج. يتعلق بالعمل المتوازي. الحالة الكلية للنظام هي تركيبة من **حالة واحدة من كل** آلة تعمل بالتوازي.

النماذج الحديثة والقوية مثل Statecharts تجمع بين هاتين الفكرتين لتعطينا أداة فعالة جداً لتخفيض التعقيد في تصميم الأنظمة.

Hierarchical Concurrent Finite State Machines

Statecharts **is a graphical language for hierachical concurrent FSMs**

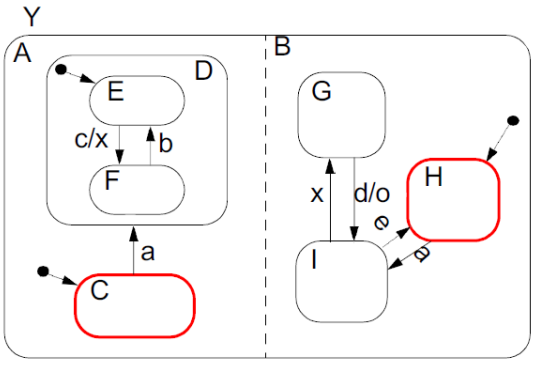


Hierarchical Concurrent Finite State Machines

Statecharts **is a graphical language for hierachical concurrent FSMs**

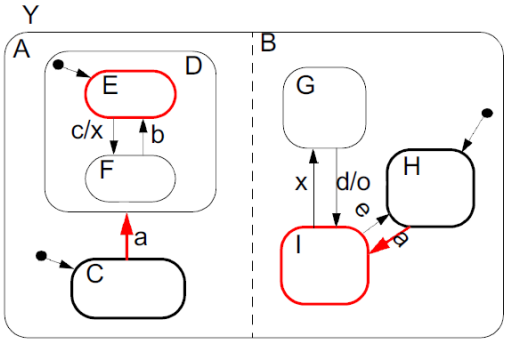
- **System enters state Y ⇒ it will be in both A and B.**
- **A consists of D and C; C is initial state for A.**
  D consists of E and F; E is initial state for D.
- **B consists of G, I, and H; H is initial state for B.**

Entering Y, the system will be simultaneously in C and H;



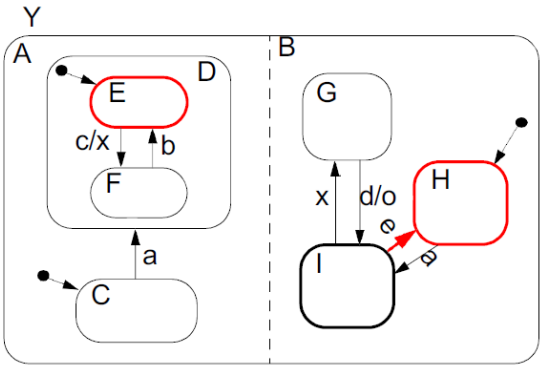
Hierarchical Concurrent Finite State Machines

a



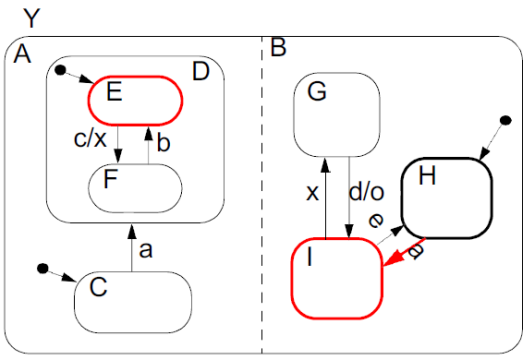
Hierarchical Concurrent Finite State Machines

a e



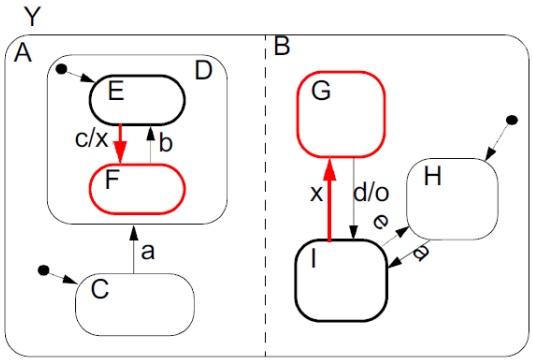
Hierarchical Concurrent Finite State Machines

a e a



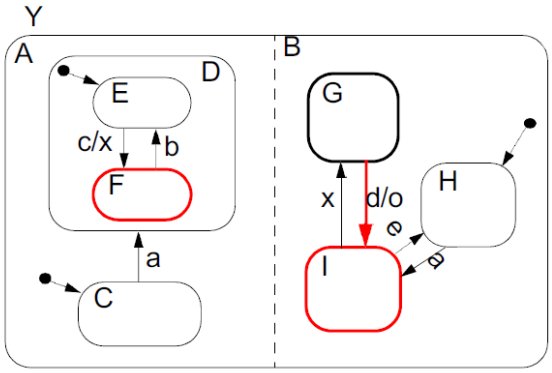
Hierarchical Concurrent Finite State Machines

a e a c



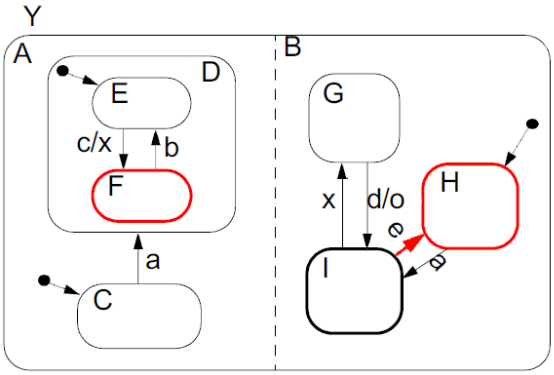
Hierarchical Concurrent Finite State Machines

a e a c d



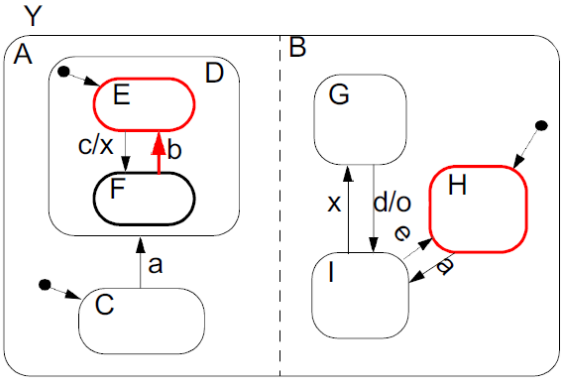
Hierarchical Concurrent Finite State Machines

a e a c d e



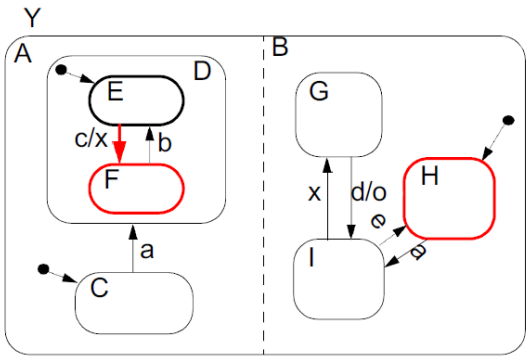
Hierarchical Concurrent Finite State Machines

a e a c d e b



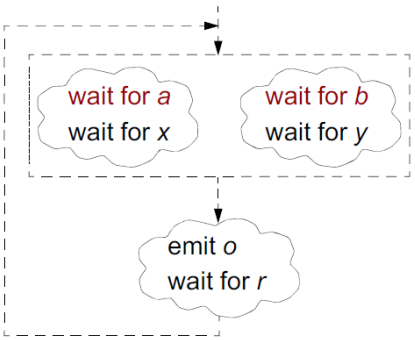
Hierarchical Concurrent Finite State Machines

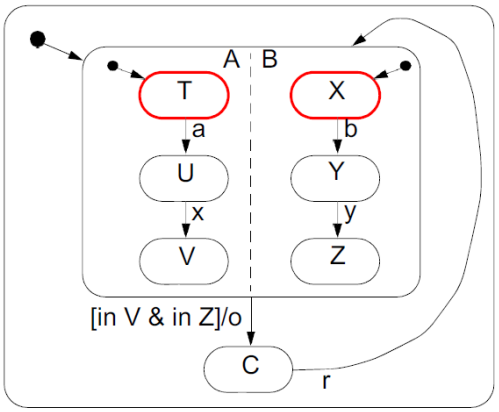
a e a c d e b c



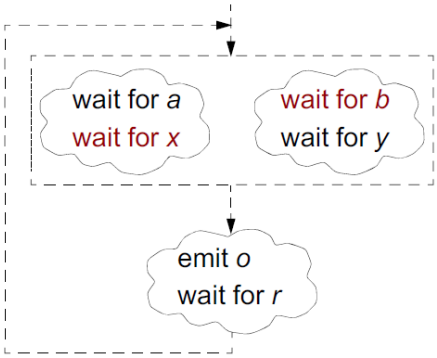
Hierarchical Concurrent Finite State Machines

Our earlier example, now using Statecharts:

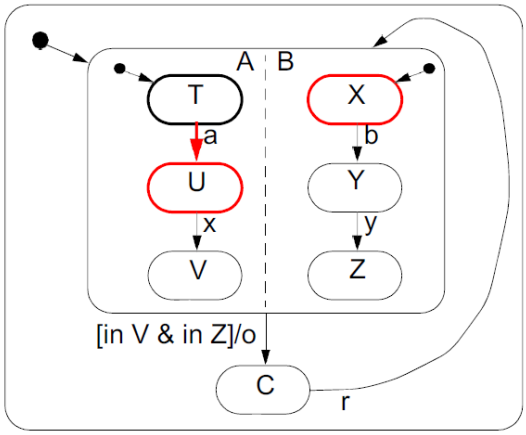




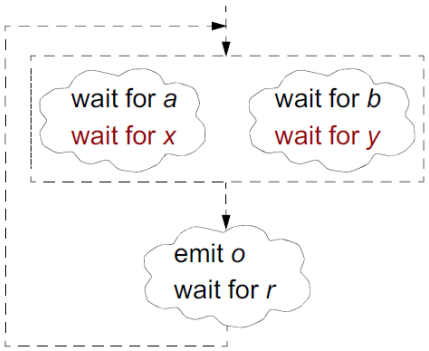
Hierarchical Concurrent Finite State Machines



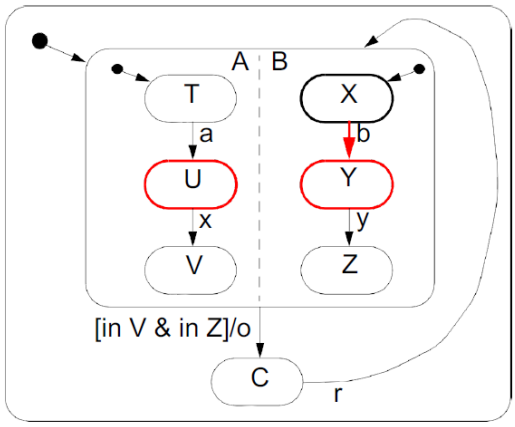
a



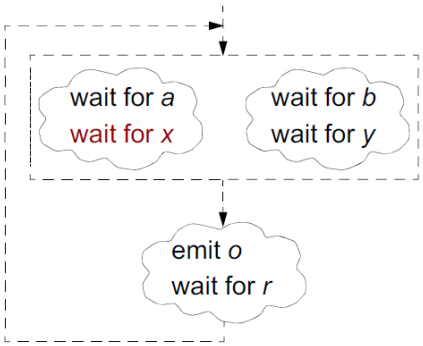
Hierarchical Concurrent Finite State Machines



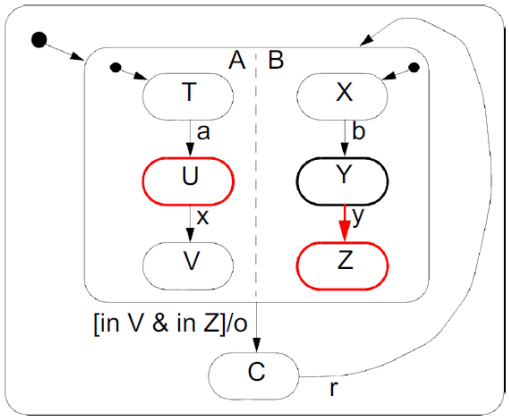
a b



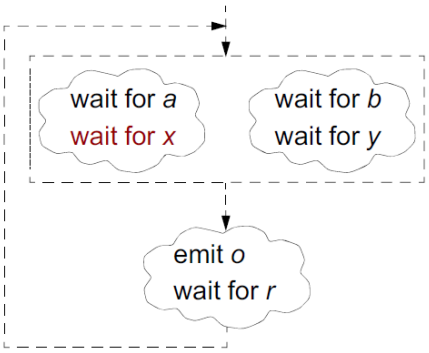
Hierarchical Concurrent Finite State Machines



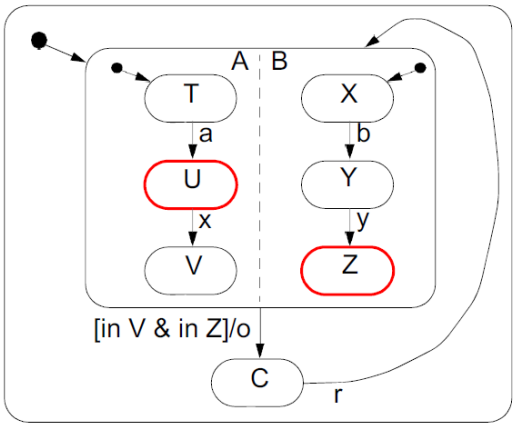
a b y



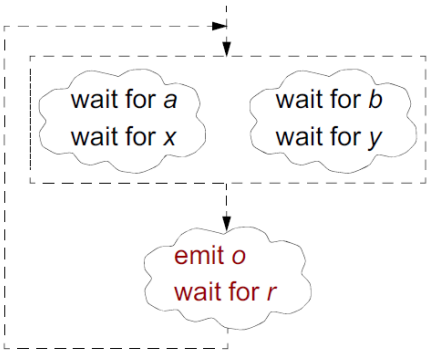
Hierarchical Concurrent Finite State Machines



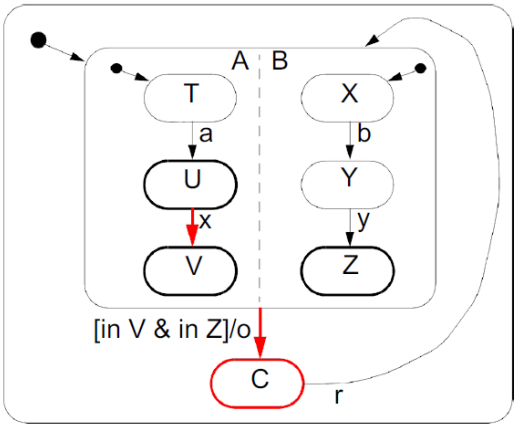
a b y a



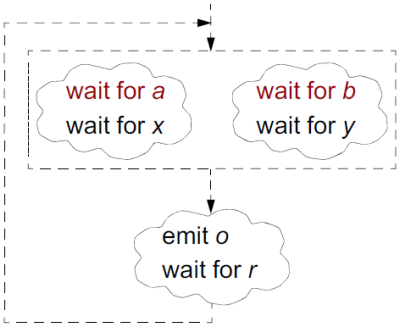
Hierarchical Concurrent Finite State Machines

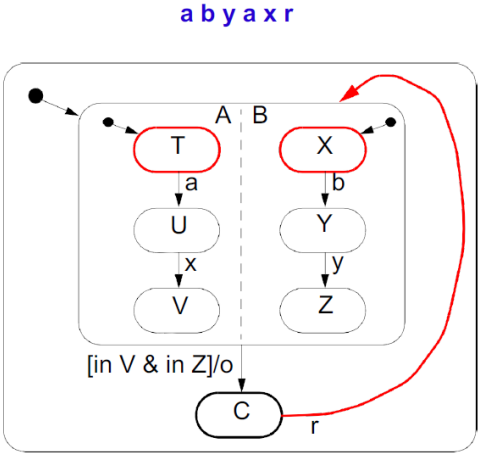


a b y a x



Hierarchical Concurrent Finite State Machines





FSMs: Time and Synchrony

- **(hierarchical concurrent) FSMs are *synchronous models*.**
  - **The synchrony hypothesis:**
    The time is a sequence of instants (clock ticks) between which nothing interesting occurs. In each instant, some events (inputs) occur in the environment, and a reaction (output) is computed *instantly* by the modelled design.
    - **Computation and internal communication (between the FSMs composing the system) take no time (compare to Discrete Event, where components can have arbitrary delays!).**
    - **Events are either simultaneous (occur at the same clock tick) or one strictly precedes the other (as opposed to dataflow and Petri Nets where we only have a partial order of events).**

آلات الحالة الهرمية والمتزامنة (Statecharts) هي **نماذج متزامنة**. هذا التصنيف مبني على افتراض أساسي وقوي يُعرف بـ **"الفرضية المتزامنة"**، والتي تمثل فلسفة النظام في التعامل مع الزمن.

الفرضية المتزامنة:

تُبسط هذه الفرضية مفهوم الزمن بشكل كبير، وتفترض أن النظام يعمل مثل لعبة تعتمد على الأدوار، حيث يسير الزمن على شكل "نبضات" أو "لحظات" منفصلة.

الفرضية تقوم على ركنين أساسيين:

1. **الزمن سلسلة من اللحظات** :الوقت لا يتدفق بشكل مستمر، بل يقفز من لحظة مهمة (أو نبضة ساعة) إلى التي تليها. بين هذه اللحظات، لا يحدث أي شيء ذي أهمية، ويكون النظام مستقراً تماماً.
2. **ردود الفعل فورية** :هذا هو الجزء الأكثر مثالية وقوة. تفترض الفرضية أن النظام، في كل لحظة، يمكنه أن يقرأ كل المدخلات، ويقوم بكل حساباته وانتقالاته الداخلية، ويُنتج كل مخرجاته في **صفر من الوقت** .أي أن ردة فعل النظام على الأحداث تكون **لحظية**.

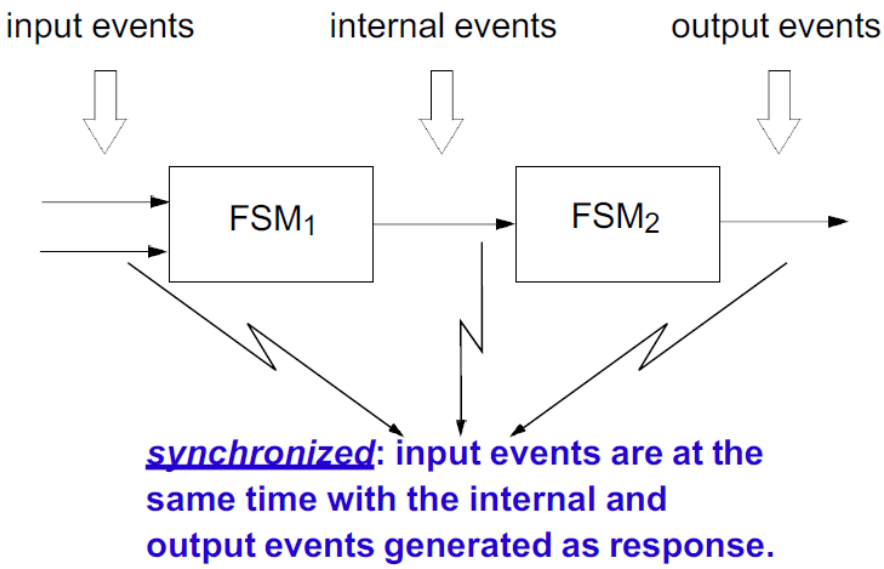
عو اقب هذه الفرضية

هذا الافتراض بأن النظام فائق السرعة له نتيجتان مهمتان تميزان النموذج المتزامن:

- **الحسابات والاتصالات الداخلية لا تستغرق وقتاً**
  - عندما يُرسل جزء من النظام (مثل المكون A في مثالنا) حدثاً داخلياً إلى جزء آخر (المكون B)، نفترض أن هذا الاتصال يحدث فوراً. هذا يختلف عن "محاكاة الأحداث المتقطعة" العامة التي رأيناها في البداية، حيث يمكن أن تكون هناك تأخيرات زمنية اختيارية (مثل after 5ns).
  - **كيف يتحقق ذلك؟** من خلال آلية **دورات دلتا (delta cycles)** التي ناقشناها. فكرة "الوقت الصفري" ليست سحراً، بل هي أن المحاكي يقوم بتنفيذ سلسلة من دورات دلتا الصغيرة جداً *داخل نفس اللحظة الزمنية* لحل كل سلسلة السبب والنتيجة قبل أن يحرك ساعته الرئيسية إلى اللحظة التالية.
- **للأحداث ترتيب كلي(Total Order)**
  - في النموذج المتزامن، بالنسبة لأي حدثين، يمكننا دائماً أن نقول بشكل قاطع: إما أنهما حدثا **في نفس اللحظة تماماً**، أو أن أحدهما **يسبق الآخر تماماً** .لا يوجد أي غموض في الترتيب.
  - هذا يختلف عن نماذج أخرى (مثل شبكات بتري) التي قد يكون لها "ترتيب جزئي"، أي أننا قد نعرف أن المهمة C تتطلب انتهاء A و B، لكن لا يهمنا أيهما انتهى أولاً. النموذج المتزامن أكثر صرامة في تحديد ترتيب الأحداث.

الخلاصة :الفرضية المتزامنة هي **نموذج مثالي (idealization)** قوي جداً، يبسط تصميم الأنظمة التفاعلية بالافتراض أن النظام أسرع بما لا يقاس من البيئة التي يتفاعل معها. هذا يسمح للمهندسين بتصميم منطق صحيح وموثوق، مع ترك تحدي "كيفية تطبيق نظام فائق السرعة على عتاد حقيقي له سرعة محدودة" كمشكلة تنفيذية يتم حلها لاحقاً.

FSMs: Time and Synchrony



هذه الشريحة تقدم لنا تصوراً بصرياً لـ"الفرضية المتزامنة" التي ناقشناها للتو، وتوضح كيفية تدفق الأحداث في نظام متزامن مكون من عدة أجزاء.

الزمن والتزامن في آلات الحالة المحدودة

لنتأمل المخطط البياني:

- لدينا آلتان، **FSM1** و **FSM2**، متصلتان على التوالي. يمكن أن تمثل FSM1 مثلاً جزءاً من النظام يستقبل المدخلات الأولية (مثل قراءة حساس)، بينما تمثل FSM2 جزءاً يتخذ القرار بناءً على ما يصله من FSM1 .
- **أحداث الإدخال (input events)**: هي الأحداث القادمة من البيئة الخارجية وتدخل إلى FSM1
- **الأحداث الداخلية (internal events)**: هي الأحداث التي تنتجها FSM1 وتقوم بتمريرها كمدخلات إلى FSM2 هذه هي "لغة الحوار" بين مكونات النظام الداخلية.
- **أحداث الإخراج: (output events)** هي ردة الفعل النهائية للنظام، والتي تنتجها FSM2 وتؤثر في البيئة الخارجية (مثل تشغيل محرك).

المفهوم الأساسي التزامن(Synchronized) :

النقطة الجوهرية هي العبارة" :**متزامنة: أحداث الإدخال تكون في نفس الوقت مع الأحداث الداخلية والخارجية التي تم توليدها كردة فعل".**

- **ماذا يعني هذا؟** هذا لا يعني أنها تحدث فيزيائياً في نفس اللحظة المستحيلة. بل يعني أنها تحدث جميعها **ضمن نفس "اللحظة الزمنية" أو "نبضة الساعة" المنطقية** للنموذج.
- **هذه هي "الفرضية المتزامنة" في صورة عملية:**
  1. في لحظة زمنية معينة T، تصل **أحداث الإدخال**.
  2. **فوراً** (أي ضمن نفس اللحظة T)، تقوم FSM1 بمعالجتها وتوليد **الأحداث الداخلية**.
  3. **فوراً** (وما زلنا في نفس اللحظة T)، تستقبل FSM2 هذه الأحداث الداخلية وتعالجها لتنتج **أحداث الإخراج** النهائية.
- **كيف هذا ممكن؟** كما ذكرنا، الآلية التي تحقق هذه "الفورية" الوهمية هي **دورات دلتا (delta cycles)** . قد تحدث هذه السلسلة على عدة دورات دلتا، ولكن ساعة المحاكاة الرئيسية تبقى ثابتة عند الزمن T حتى تنتهي سلسلة ردود الفعل بأكملها.

الخلاصة :هذا المخطط يوضح لنا سلسلة السبب والنتيجة في نظام متزامن. على الرغم من أن المكونات مرسومة بشكل متسلسل، فإن الافتراض الأساسي للنموذج هو أن السلسلة الكاملة، من المدخل الخارجي إلى المخرج النهائي، تحدث في **لحظة منطقية واحدة** .هذا يجعل سلوك النظام قابلاً للتنبؤ والتحليل، لأننا لا نحتاج للقلق بشأن الوقت الذي تستغرقه الإشارات للانتقال بين المكونات الداخلية.

FSMs: Time and Synchrony

Question

Is the synchronous model sufficiently realistic to be used in practice?

Answer

For some applications yes!

It is the case when the following assumption is true:

The reaction time of the system (including internal communication) is neglectable compared to the rate of external events.

هذا هو السؤال الجوهري الذي يطرح نفسه بعد كل هذا الحديث عن "الوقت الصفري" و"ردود الفعل الفورية". هل هذا النموذج المتزامن واقعي بما يكفي لنستخدمه في بناء أنظمة حقيقية؟

الجواب، هو **نعم، ولكن ضمن شرط مهم جداً.**

متى يكون النموذج المتزامن و اقعياً؟

النموذج المتزامن يكون مقاربة واقعية ومفيدة عندما يتحقق الافتراض التالي:

"أن يكون زمن ردة فعل النظام (بما في ذلك الاتصالات الداخلية) ضئيلاً جداً ومهملاً، مقارنة بمعدل وصول الأحداث الخارجية".

سوف نشرح بتفصيل أكثر هذه العبارة:

- **زمن ردة فعل النظام** :هو الوقت الحقيقي بالمايكروثانية أو الملي ثانية الذي يستغرقه المعالج لتنفيذ الكود، وتحديث الحالات، وإصدار الأوامر.
- **معدل وصول الأحداث الخارجية** :هو الفاصل الزمني بين كل حدث قادم من البيئة والحدث الذي يليه. مثلاً، قراءة بيانات حساس كل 100 ملي ثانية، أو استجابة لضغط إنسان على زر كل ثانية.

مثال توضيحي: لنتخيل أننا نلعب الشطرنج عبر الإنترنت مع لاعب عالمي.

- **الحدث الخارجي** :هو عندما نقوم نحن بحركتنا.
- **معدل الأحداث** :قد يكون هناك فاصل زمني مقداره 30 ثانية بين كل حركة والأخرى.
- **زمن ردة فعل اللاعب** :قد يفكر ويقوم بحركته في ثانية واحدة فقط.

بالنسبة لنا، تبدو استجابة اللاعب **فورية تقريباً** أو "متزامنة" مع حركتنا. الثانية الواحدة التي استغرقها هي زمن **ضئيل ومُهمل** مقارنة بالـ30 ثانية المتاحة. إذن، "الفرضية المتزامنة" هنا صالحة لوصف هذه اللعبة.

لكن، لو كان هذا اللاعب يلعب ضد كمبيوتر فائق يقوم بحركاته كل نصف ثانية، فإن زمن ردة فعل اللاعب (ثانية واحدة) لن يكون مهملاً بعد الآن، وهنا تنهار الفرضية المتزامنة.

في الأنظمة المدمجة :هذا الشرط غالباً ما يكون متحققاً. يمكن لمتحكم صغري (microcontroller) حديث أن ينفذ ملايين التعليمات في الثانية، وقد يكون زمن ردة فعله في حدود المايكروثانية. بينما الأحداث التي يتفاعل معها (تغير درجة حرارة، ضغط زر، إشارة من محرك) تحدث على مقياس الملي ثانية أو حتى الثواني.

لأن المايكروثانية ضئيلة جداً مقارنة بالملي ثانية، يمكننا بكل أمان أن **نصمم وننمذج** نظامنا وكأنه متزامن، مما يبسط عملية التصميم والتحقق بشكل هائل، مع علمنا التام بأنه في الواقع الفيزيائي ليس فورياً.

Why Do We Like Synchronous Models?

- **A set of communicating, concurrent FSMs behaves exactly like one equivalent FSM.**

Models are deterministic.
It is possible to formally reason about models and to formally check properties of the model. This is important for safety critical applications.

- **It is possible to efficiently synthesise (compile) synchronous models to hardware or software.**

هذه الشريحة تلخص لنا الأسباب الجوهرية التي تجعلنا كمهندسين نحب ونفضل استخدام النماذج المتزامنة، خاصة في تصميم الأنظمة المعقدة والحرجة. سوف نستعرض هذه المزايا.

لماذا نحب النماذج المتزامنة؟

- **"A set of communicating, concurrent FSMs behaves exactly like one equivalent FSM."مجموعة من آلات الحالة المحدودة المتزامنة والمتصلة تتصرف تماماً مثل آلة حالة محدودة واحدة مكافئة.**
  - هذه نقطة نظرية عميقة وقوية جداً. لنتذكر مثال "انفجار الحالات" الذي رأيناه؟ كان لدينا نموذج Statechart النظيف والمكون من جزئين متوازيين، وكان لدينا أيضاً نموذج" FSM المسطح" والمعقد ذو الـ 9 حالات.
  - ما تقوله هذه النقطة هو أن هذين النموذجين، على الرغم من اختلاف شكلهما، يصفان **نفس السلوك تماماً** . النموذج المتزامن المنظم هو مجرد طريقة أكثر ذكاءً وفعالية لوصف نفس آلة الحالة الضخمة المكافئة.
  - هذا يعطينا أفضل ما في النموذجين: نصمم بنموذج منظم وسهل الفهم، مع الحفاظ على الأسس الرياضية القوية لآلات الحالة المحدودة.
- **نتيجة للنقطة السابقة، نحصل على المزايا التالية:**
  - **"Models are deterministic."النماذج حتمية.**
    - بما أن نموذجنا يكافئ آلة حالة محدودة، فإن سلوكه حتمي تماماً. أي أنه عند إعطائه نفس الحالة الأولية ونفس سلسلة المدخلات، سينتج دائماً نفس سلسلة المخرجات ويزور نفس تسلسل الحالات. لا يوجد أي مجال للغموض أو السلوك غير المتوقع.

- **"It is possible to formally reason about models and to formally check properties of the model."**
  من الممكن الاستدلال رسمياً على النماذج والتحقق رسمياً من خصائصها.
  - وهذه هي الفائدة الكبرى. لأن النموذج له أساس رياضي واضح، يمكننا استخدام أدوات حاسوبية قوية **لإثبات** خصائص معينة حوله بشكل رياضي قاطع.
  - "الإثبات" هنا لا يعني مجرد الاختبار والمحاكاة لبعض السيناريوهات، بل هو برهان رياضي يغطي **كل السيناريوهات الممكنة**.
  - يمكننا طرح أسئلة مثل: "هل يمكن أن يفتح باب المصعد أبداً أثناء حركته؟" أو "هل يمكن أن تدخل وحدة التحكم في السيارة في حالة جمود(deadlock) ؟" والحصول على إجابة "نعم" أو "لا" بثقة تامة.
- **"This is important for safety-critical applications."هذا مهم للتطبيقات الحرجة للسلامة.**
  - لهذا السبب، تُستخدم النماذج المتزامنة بكثافة في تصميم الأنظمة التي لا مجال للخطأ فيها، مثل أنظمة التحكم في الطائرات ، الأجهزة الطبية (منظمات ضربات القلب)، أنظمة التحكم في محطات الطاقة النووية، وأنظمة الفرامل في السيارات .

- **"It is possible to efficiently synthesise (compile) synchronous models to hardware or software."**
  من الممكن ترجمة النماذج المتزامنة بكفاءة إلى عتاد أو برمجيات
  - **Synthesis**: يعني التحويل التلقائي للنموذج عالي المستوى (مثل الStatechart) إلى كود نهائي يمكن تنفيذه مباشرة، سواء كان كوداً برمجياً(مثل لغة C) ليعمل على متحكم صغري أو وصفاً عتادياً (مثل VHDL) ليتم تصنيعه كشريحة إلكترونية.
  - الطبيعة الحتمية والمحددة جيداً للنماذج المتزامنة تجعل عملية الترجمة هذه موثوقة وفعالة للغاية، حيث يمكن للمترجم (compiler) أن يولد كود أمثل لأنه يفهم تماماً تبعيات التوقيت في النموذج.

الخلاصة :نحن نحب النماذج المتزامنة لأنها تجمع بين **أناقة التصميم** (من خلال الهرمية والتزامن) و**الصلابة الرياضية** (لأنها حتمية وقابلة للتحقق الرسمي) و**الجدوى العملية** (لأنها قابلة للترجمة إلى أنظمة حقيقية بكفاءة).

Why Do We Not Like Synchronous Models?

- **Reasoning, verification and synthesis based on synchronous models are meaningful and correct only as long as:**
  1. **A completely synchronous implementation of the whole system is possible.**
  2. **We are sure that for the *implemented system* the assumption is true: *The reaction time of the system (including internal communication) is neglectable compared to the rate of external events.***

- **Implementing *large* models as synchronous systems is expensive and often technically impossible.**

النماذج المتزامنة ليست سيئة، لكن قوتها وفوائدها تأتي مع **شروط مهمة** .كل المزايا الرائعة التي ناقشناها، مثل التحقق الرسمي وتوليد الكود، تكون صحيحة وذات معنى فقط طالما أننا نستطيع الالتزام بـ"الفرضية المتزامنة" في النظام الفعلي.

الشروط اللازمة للنجاح

1. **أن يكون التنفيذ المتزامن ممكناً** :التحدي الأول هو فيزيائي. يجب أن نكون قادرين على بناء نظام تعمل فيه كل المكونات بتناغم تام، كما لو أنها تتبع ساعة عالمية واحدة. هذا ممكن على شريحة إلكترونية واحدة، لكنه يصبح صعباً ومكلفاً جداً للأنظمة الموزعة على لوحة دوائر كبيرة، وشبه مستحيل للأنظمة الموزعة جغرافياً (مثل شبكة الكهرباء).
2. **أن يكون زمن ردة الفعل مهملاً** :هذا هو الشرط الأهم. يجب أن يكون النظام الفعلي الذي نبنيه **أسرع بكثير** من البيئة التي يتفاعل معها.
   - **زمن ردة الفعل** :هو الوقت الحقيقي (بالمايكروثانية مثلاً) الذي يستغرقه المعالج لتنفيذ المنطق وإصدار الاستجابة.
   - **معدل الأحداث الخارجية** :هو الفاصل الزمني بين كل حدث قادم من العالم الخارجي والحدث الذي يليه (بالمللي ثانية مثلاً).

إذا لم يكن زمن ردة فعل النظام ضئيلاً ومهملاً، فإن سلوك النظام الحقيقي لن يتطابق مع سلوك النموذج الذي تحققنا من صحته، مما قد يؤدي إلى كوارث.

القيود العملية

- **تنفيذ النماذج الكبيرة بشكل متزامن هو أمر مكلف وغالباً مستحيل تقنياً** .لتحقيق شرط "ردة الفعل المهملة" في نظام كبير ومعقد، نحتاج إلى معالجات وعتاد فائق السرعة (وبالتالي باهظ الثمن). بالنسبة للأنظمة الموزعة جغرافياً، فإن سرعة الضوء تجعل الاتصال الفوري مستحيلاً فيزيائياً، مما يجعل هذه الأنظمة **غير متزامنة** بطبيعتها.

باختصار، النموذج المتزامن هو أداة قوية، وليس حلاً لكل المشاكل. يتألق في الأنظمة المدمجة السريعة والموجودة في مكان واحد، حيث السلامة والموثوقية لها الأولوية القصوى.

Synchronous/Reactive Languages

- **Synchronous/reactive languages describe systems as a set of concurrently executing synchronized activities.**
  - **Communication is through signals.**
  - **Signals are either present or absent at a certain *tick*.**
  - **The presence of a signal is called an event.**

- ***These language are semantically equivalent to the (extended hierarchical concurrent) FSM model !!!***

- ***Esterel* is a well known synchronous/reactive language. Every Esterel model can be compiled to an extended FSM.**

How to implement a synchronous system?

- **In hardware:**
  - **System described as single FSM:**
    - **implementation as a state machine.**
  - **System described as several FSMs:**
    - **several communicating synchronous state machines or**
    - **implement the equivalent single (very large) state machine**

***But if the system is large*:**

- **How do you distribute the clock signal on a large chip, in order to keep synchrony?**
- **If there are several chips, keeping synchrony is even more difficult.**

هذه الشريحة تجيب على سؤالنا العملي، "كيف ننفذ نظاماً متزامناً؟"، وتركز هنا على الخيار الأول وهو **التنفيذ في العتاد(In hardware)**

خيارات التنفيذ في العتاد

عندما نصمم نموذجنا المتزامن، يكون لدينا خياران رئيسيان لتنفيذه مباشرة كدارة إلكترونية:

- **إذا كان النظام موصوفاً كآلة حالة واحدة بسيطة(single FSM) :**
  - يكون التنفيذ مباشراً. يمكننا بناء دارة رقمية باستخدام مكونات مثل القلابات (flip-flops) لتخزين الحالة، ودارات المنطق التوافقي (combinational logic) لتحديد الحالة التالية والمخرجات. هذا تطبيق كلاسيكي في تصميم المنطق الرقمي.
- **إذا كان النظام موصوفاً كعدة آلات حالة متوازية(Statechart) :**
  - **الخيار1 :** بناء عدة دارات متزامنة متصلة. أي أننا نبني دارة منفصلة لكل مكون متوازٍ، وتتواصل هذه الدارات مع بعضها عبر أسلاك، وتتشارك جميعها نفس إشارة الساعة (clock signal) لتبقى متزامنة.
  - **الخيار2 :** تنفيذ الآلة الواحدة المكافئة. يمكننا استخدام أداة برمجية "لتسطيح" نموذجنا المنظم وتحويله إلى آلة الحالة الواحدة الضخمة والمعقدة المكافئة له، ثم نقوم ببناء دارة لتلك الآلة الكبيرة.

لكن إذا كان النظام كبيراً...

هنا يأتي دور الواقع ليفرض تحدياته الفيزيائية على تصميمنا المثالي:

- **على الشريحة الإلكترونية الكبيرة(large chip) :**
  - **التحدي :**كيف توزع إشارة الساعة على شريحة كبيرة مع الحفاظ على التزامن؟ إشارة الساعة هي نبضة كهربائية، وهي تستغرق وقتاً لتنتقل عبر مسافات، حتى لو كانت صغيرة. قد تصل النبضة إلى جزء من الشريحة قبل جزء آخر ببضعة أجزاء من النانوثانية. هذا الفارق الزمني يسمى **انحراف الساعة (clock skew)** . إذا زاد هذا الانحراف، تفقد أجزاء الشريحة تزامنها التام، وقد يفشل النظام.
- **عبر عدة شرائح إلكترونية(several chips) :**
  - **التحدي :**الحفاظ على التزامن يصبح **أكثر صعوبة بكثير** .التأخير الزمني للاتصال بين شريحة وأخرى أكبر بكثير من التأخير داخل الشريحة الواحدة. يصبح من الصعب جداً جعل شريحتين منفصلتين تعملان على نفس "نبضة" الساعة تماماً.

الخلاصة :التنفيذ المباشر في العتاد للنماذج المتزامنة هو أسلوب قوي جداً، خاصة للأنظمة التي يمكن وضعها على شريحة واحدة. لكن مع ازدياد الحجم المادي للنظام، تظهر القيود الفيزيائية (مثل انحراف الساعة وتأخير الاتصالات) التي تجعل الحفاظ على التزامن المثالي أمراً صعباً ومكلفاً، وفي بعض الأحيان مستحيلاً.

How to implement a synchronous system?

- In software:
  - One single FSM or several FSMs:
    Generate a sequential program which *emulates the state machine*.

Problems:

- Large concurrent systems ⇒ state explosion ⇒ very large programs.
- It is practically impossible to implement the software on a large multiprocessor/distributed system (extremely inefficient to keep the global synchrony of such a multiprocessor/distributed software).

بعد أن ناقشنا التنفيذ في العتاد، ننتقل الآن إلى الخيار الآخر والأكثر شيوعاً في كثير من الأحيان، وهو **التنفيذ في البرمجيات (In software)**، أي تشغيل نموذجنا على معالج أو متحكم صغري.

التنفيذ في البرمجيات

- **الفكرة الأساسية :**سواء كان نموذجنا آلة حالة واحدة بسيطة أو مجموعة معقدة من الآلات المتوازية(Statechart) ، فإن طريقة التنفيذ الأساسية هي **توليد برنامج تسلسلي يقوم بمحاكاة (emulates) سلوك آلة الحالة.**
- **كيف يعمل هذا البرنامج؟**
  - غالباً ما يتم بناؤه على شكل حلقة تحكم لا نهائية (infinite loop) . في كل دورة من هذه الحلقة (والتي تمثل "نبضة ساعة" واحدة في عالمنا المتزامن):
    1. يقرأ البرنامج كل المدخلات الحالية من البيئة.
    2. يحسب الحالة التالية والمخرجات بناءً على الحالة الحالية والمدخلات الجديدة(هنا يتم تطبيق منطق الـ FSM).
    3. يقوم بتحديث متغيرات الحالة وإصدار المخرجات.
    4. ثم ينتظر "النبضة" التالية ليعيد الدورة.
  - هذه الحلقة الواحدة تضمن أن كل شيء يحدث بترتيب محدد، مما يحقق "وهم" التزامن والفورية الذي يتطلبه نموذجنا.

المشاكل والتحديات

ولكن، كما هو الحال مع العتاد، يواجه التنفيذ البرمجي تحديات كبيرة عندما يكبر النظام:

- **مشكلة 1: انفجار الحالات يؤدي إلى برامج ضخمة.**
  - إذا حاولنا تحويل نموذج متزامن معقد إلى آلة حالة واحدة مكافئة، فإن البرنامج الناتج عن هذا التحويل قد يصبح ضخماً جداً. لنتخيل برنامجاً يحتوي على بنية switch بها آلاف الحالات (case) يصبح هذا الكود صعب الفهم، صعب التنقيح، وغير فعال.
- **مشكلة 2: صعوبة التنفيذ على الأنظمة الموزعة أو متعددة المعالجات.**
  - ماذا لو حاولنا تشغيل برنامجنا على عدة معالجات أو حواسيب مختلفة لتسريع الأداء؟ هنا نواجه مشكلة شبه مستحيلة.
  - الحفاظ على التزامن التام بين برامج تعمل على معالجات مختلفة هو أمر **غير فعال للغاية**. التكلفة الزمنية للتواصل والتنسيق بين المعالجات للتأكد من أنها جميعاً في نفس "النبضة" المنطقية وأنها عالجت كل أحداثها قبل الانتقال للنبضة التالية، تكون عالية جداً وتلغي فائدة استخدام عدة معالجات من الأساس.

الخلاصة: التنفيذ البرمجي باستخدام حلقة تحكم تسلسلية هو أسلوب شائع وفعال جداً لتطبيق النماذج المتزامنة، خاصة في الأنظمة المدمجة التي تعمل على متحكم صغري واحد. لكن، محدوديته تظهر مع الأنظمة الكبيرة جداً، حيث يؤدي انفجار الحالات إلى كود معقد، وتصبح فكرة التوزيع على عدة معالجات غير عملية بسبب صعوبة الحفاظ على التزامن.

How to implement a synchronous system?

- **If the model is impossible (or very difficult and expensive) to implement, there is no use that it is elegant, simple, and can be formally verified.**
  ***We get a correct verified model but we cannot implement it correctly*!**

Synchronous models are very good for relatively small systems implemented in hardware or software.

- **For larger systems *we have to give up the assumption of global synchrony*.**

متى نستخدم النماذج المتزامنة؟

هذه الشريحة تلخص لنا المعضلة التي نواجهها كمهندسين:

- **"إذا كان النموذج مستحيلاً (أو صعباً ومكلفاً جداً) في التنفيذ، فلا فائدة من كونه أنيقاً وبسيطاً ويمكن التحقق من صحته رسمياً".**

- **المعضلة:** ما الفائدة من امتلاك "مخطط بناء" مثالي ومثبت رياضياً (وهو نموذجنا المتزامن)، إذا كنا لا نستطيع بناء المبنى الفعلي (وهو التنفيذ) ليتطابق مع هذا المخطط تماماً؟
- **"نحصل على نموذج صحيح ومُتحقَّق منه، لكننا لا نستطيع تنفيذه بشكل صحيح "**
  - هذه هي نقطة الفشل الحاسمة. إذا كان العتاد أو البرنامج الذي نبنيه لا يستطيع أن يكون سريعاً بما يكفي ليحقق "وهم الفورية"، فإن سلوك النظام الحقيقي سيبدأ بالاختلاف عن سلوك النموذج. وعندها، كل ضمانات السلامة التي أثبتناها على النموذج تصبح بلا قيمة في المنتج الفعلي.

- **النتيجة المنطقية:**
- **"النماذج المتزامنة جيدة جداً للأنظمة الصغيرة نسبياً، سواء تم تنفيذها في العتاد أو البرمجيات".**
  - "صغيرة نسبياً" هنا لا تعني بالضرورة أنها بسيطة، بل تعني أنها **موجودة في مكان واحد** (على شريحة واحدة، أو لوحة إلكترونية واحدة، أو تعمل على معالج واحد)، وأنها **سريعة بما يكفي** لتحقيق الفرضية المتزامنة.
  - هذا يشمل غالبية الأنظمة المدمجة، وحدات التحكم في السيارات، الأجهزة الإلكترونية الاستهلاكية، وغيرها الكثير.

- **"بالنسبة للأنظمة الأكبر، علينا أن نتخلى عن افتراض التزامن العالمي".**
  - عندما نتعامل مع أنظمة كبيرة جداً أو موزعة جغرافياً (مثل شبكات الإنترنت، شبكات الطاقة الكهربائية، أو أنظمة التحكم الموزعة في مصنع ضخم)، يصبح من المستحيل فيزيائياً الحفاظ على تزامن عالمي ودقيق.
  - في هذه الحالة، يجب أن ننتقل إلى نماذج تصميم أخرى، أهمها **النماذج غير المتزامنة (Asynchronous Models)** . في هذه النماذج، لا نفترض وجود ساعة عالمية، بل نتعامل بشكل صريح مع تأخيرات الاتصال، ونستخدم آليات مثل طوابير الرسائل وبروتوكولات التنسيق.

الخلاصة النهائية :النموذج المتزامن هو أداة تصميم رائعة وقوية توفر الموثوقية والأمان.. نستخدم النموذج المتزامن عندما يكون افتراض "الفورية" واقعياً، ونتجه إلى النماذج غير المتزامنة عندما تفرض علينا حقيقة الفيزياء والأنظمة الموزعة ذلك.

GLOBALLY ASYNCHRONOUS LOCALLY SYNCHRONOUS SYSTEMS

1. Globally Asynchronous Locally Synchronous Systems

2. Globally Asynchronous Locally Synchronous System Models

قلنا أن النموذج المتزامن رائع للأنظمة الصغيرة، ولكن علينا التخلي عن فكرة "التزامن العالمي" في الأنظمة الكبيرة والموزعة. السؤال الآن هو: "ماذا نفعل بدلاً من ذلك؟"

الجواب هو نموذج هجين وقوي جداً يجمع أفضل ما في الحالتين، **" الأنظمة غير المتزامنة عالمياً والمتزامنة محلياً(GALS) "**

ما هو نموذج GALS؟

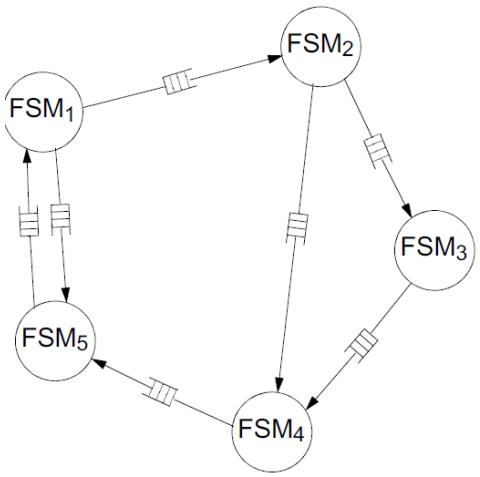
سوف نشرح فكرته الأساسية:

- **متزامن محلياً (Locally Synchronous) :**
  - هذا الجزء يعني أننا سنقوم ببناء **المكونات الفردية** أو النظم الفرعية باستخدام النماذج المتزامنة التي تعلمناها للتو (مثل Statecharts). داخل كل "جزيرة" من هذه الجزر المتزامنة، يكون كل شيء مثالياً: ردود الفعل فورية، السلوك حتمي، ويمكن التحقق من صحته رسمياً. حيث نحتفظ بكل مزايا العالم المتزامن على المستوى المحلي.
- **غير متزامن عالمياً (Globally Asynchronous) :**
  - هذا الجزء يعني أن **التواصل بين هذه الجزر المتزامنة** لا يفترض وجود ساعة عالمية مشتركة. نحن نعترف بأن إرسال رسالة من مكون إلى آخر يستغرق وقتاً، وأنه ليس فورياً.
  - **مثال توضيحي :**لنفترض لدينا شركة عالمية لها مكاتب في طوكيو ونيويورك.
    - **محلياً**، كل مكتب (مثل مكتب طوكيو) يعمل بجدول زمني داخلي صارم ومتزامن. الاجتماعات الداخلية في وقتها والتواصل سريع جداً.
    - **عالمياً**، التواصل بين مكتبي طوكيو ونيويورك غير متزامن. هم يستخدمون البريد الإلكتروني والمكالمات المجدولة، ويجب أن يأخذوا في الحسبان فروق التوقيت وتأخير الاتصالات.

باختصار، نهج GALS هو حل عملي وذكي يتيح لنا استخدام قوة وصلابة التصميم المتزامن للمكونات المعقدة، مع الاعتراف بواقعية الاتصالات غير المتزامنة في الأنظمة الكبيرة.

GALS Systems

Globally asynchronous and locally synchronous (GALS) models:



- **Each FSM individually behaves like a synchronous systems ⇒ reacts instantaneously on a set of available inputs and generates output.**
- **The global system is asynchronous ⇒ communication time is finite and non-zero; reaction time of each FSM, as viewed by other FSMs is finite and non-zero.**
- **With global asynchrony, buffering of signals could be needed.**

تفكيك نموذج GALS

يوضح المخطط البياني الفكرة الأساسية: لدينا مجموعة من آلات الحالة المحدودة (FSMs) التي تمثل "جزرنا المتزامنة"، وهي تتواصل مع بعضها البعض.

1- متزامن محلياً (Locally Synchronous)

- **"Each FSM individually behaves like a synchronous system" كل آلة حالة تتصرف بشكل فردي كنظام متزامن**
- هذا يعني أن كل دارة من هذه الدارات(FSM1, FSM2,....) هي عالم مثالي بحد ذاتها. بداخلها، تنطبق "الفرضية المتزامنة" تماماً: تستجيب للمدخلات وتولد المخرجات بشكل **فوري** (في صفر من الوقت المنطقي)، وسلوكها حتمي ويمكن التحقق منه.

2- غير متزامن عالمياً (Globally Asynchronous)

- **"The global system is asynchronous" النظام الكلي غير متزامن**
- هنا نعود إلى العالم الحقيقي. عندما تتواصل هذه "الجزر" مع بعضها، فإن التواصل ليس فورياً.
- **"communication time is finite and non-zero"وقت الاتصال محدود وغير صفري** :(عندما ترسل FSM1 رسالة إلى FSM2، فإن هذه الرسالة تستغرق وقتاً حقيقياً للوصول. من وجهة نظر FSM2، فإن ردة فعل FSM1 ليست فورية، بل تصلها بعد هذا التأخير.

3- الحاجة إلى التخزين المؤقت(Buffering)

- **"With global asynchrony, buffering of signals could be needed." مع عدم التزامن العالمي، قد تكون هناك حاجة إلى تخزين مؤقت للإشارات .**
- هذه هي النتيجة المباشرة والعملية لكون الاتصالات غير فورية.
- لنفترض أن FSM1 سريعة جداً وأرسلت ثلاث رسائل متتالية إلى FSM2، لكن FSM2 كانت مشغولة بمعالجة الرسالة الأولى. بدون آلية لتخزين الرسائل القادمة، ستضيع الرسالتان الثانية والثالثة.
- **الحل :هو التخزين المؤقت (Buffering)** . الرموز الصغيرة التي تشبه الطابور (queue) على الأسهم في المخطط تمثل "صناديق بريد" أو "مخازن مؤقتة" تحفظ الرسائل أو الأحداث المرسلة من آلة حتى تصبح الآلة المستقبلة جاهزة لمعالجتها. هذا يضمن عدم فقدان أي معلومات بسبب تأخير الاتصالات أو اختلاف سرعات المعالجة بين المكونات.

الخلاصة : نموذج GALS هو عبارة عن شبكة من "الجزر" المتزامنة والمثالية، تتواصل فيما بينها عبر قنوات اتصال "غير متزامنة" وواقعية، مع استخدام مخازن مؤقتة لضمان موثوقية هذا الاتصال.

GALS Systems

- **With a GALS model, the set of FSMs is not any more equivalent with a single FSM (as was the case for the synchronous model).**

Several nice features are gone:

  - **With synchronous FSMs we had the nice theoretical background and the possibility of formal verification of the whole system. *Not the case with GALS*.**
  - **Every implementation of a synchronous FSM model is guaranteed to be functionally equivalent to the initial model and behave exactly and deterministically like the model (in the case we are able to produce an implementation!). *Not the case with GALS*.**

بعد أن رأينا أن نموذج GALS هو حل عملي للأنظمة الكبيرة، سوف نوضح "الثمن" الذي ندفعه أو "المقايضة" التي نقوم بها عند التخلي عن التزامن العالمي.

ما الذي نخسره مع نموذج GALS ؟

- **"With a GALS model, the set of FSMs is not any more equivalent with a single FSM" في نموذجGALS ، لم تعد مجموعة آلات الحالة مكافئة لآلة حالة واحدة بسيطة**
  - لنتذكر أنه في النموذج المتزامن، قلنا أن نموذج Statechart يكافئ تماماً آلة FSM واحدة ضخمة وحتمية.

- هذا **لم يعد صحيحاً** في GALS لماذا؟ لأن الاتصال غير المتزامن وإمكانية وجود تأخيرات غير محددة بين المكونات يضيف سلوكاً **غير حتمي (non-deterministic)** على مستوى النظام ككل. لم نعد نستطيع التنبؤ 100% بالترتيب الذي ستصل به الرسائل، وبالتالي لا يمكن تمثيل كل هذا السلوك بآلة حالة واحدة بسيطة وحتمية.

- **"Several nice features are gone:" العديد من الميزات الرائعة قد ذهبت**

  - **1- فقدان إمكانية التحقق الرسمي الشامل:**

    - في النموذج المتزامن، كان بإمكاننا التحقق رياضياً من سلامة النظام **بأكمله** .أما فيGALS ، يصبح هذا صعباً للغاية.

    - يمكننا التحقق من كل "جزيرة متزامنة" على حدة، ولكن إثبات أن **التفاعل** بين هذه الجزر سيكون آمناً دائماً تحت كل الظروف والتأخيرات الممكنة هو تحدٍ كبير جداً.

  - **2- فقدان التكافؤ المضمون بين النموذج والتنفيذ:**

    - في النموذج المتزامن، إذا تمكنا من بناء النظام بشكل يلتزم بالفرضية، فإننا نضمن أن سلوك المنتج النهائي سيتطابق تماماً مع سلوك النموذج الذي تحققنا منه.

    - فيGALS ، يعتمد السلوك الفعلي للنظام على التوقيت الحقيقي للاتصالات. قد يعمل النظام بشكل مختلف قليلاً في كل مرة يتم تشغيله فيها اعتماداً على ازدحام الشبكة أو عوامل أخرى، مما يؤدي إلى تداخل مختلف للأحداث. هذا يعني أن الرابط بين النموذج النظري والتنفيذ الفعلي أصبح أضعف.

الخلاصة: تحصل مقايضة هندسية (Engineering Trade-off)

- **النموذج المتزامن الصرف** :يمنحنا نظرية حتمية تامة، وضمانات قوية، لكنه غالباً ما يكون مستحيل التنفيذ للأنظمة الكبيرة.
- **نموذج GALS** : يمنحنا نموذجاً **عملياً وقابلاً للتنفيذ** للأنظمة الكبيرة والموزعة، لكننا نضحي في المقابل بالحتمية العالمية وسهولة التحقق الرسمي من النظام بأكمله.

وهذه هي طبيعة الهندسة: اختيار الأداة المناسبة للمهمة، مع فهم كامل للمقايضات التي ينطوي عليها كل خيار .

GALS Systems

- **The GALS model is not deterministic, in the sense that its behavior depends on the amount of time taken for a certain communication or transition.**

Two different implementations of the same GALS model can behave differently.

هذه الشريحة تلخص "الثمن" النهائي الذي ندفعه عندما نختار هذا النهج العملي.

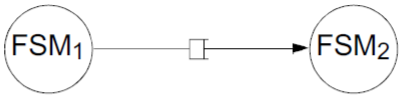
اللا حتمية: السمة الأساسية لأنظمة GALS

- **"The GALS model is not deterministic..."نموذج GALS ليس حتمياً**
  - عكس النموذج المتزامن الصرف الذي يضمن نفس المخرجات لنفس المدخلات دائماً، فإن نموذج GALS **غير حتمي (non-deterministic)**.
  - **ما معنى "غير حتمي" في هذا السياق؟** ": سلوكه يعتمد على مقدار الوقت الذي يستغرقه اتصال أو انتقال معين".
  - **مثال للتوضيح:**
    - لنتخيل أن المكون FSM1 يرسل رسالة "ابدأ" إلى FSM3
    - وفي نفس الوقت تقريباً، يرسل المكون FSM2 رسالة "توقف" إلى FSM3
    - أي رسالة ستصل أولاً إلى FSM3؟ الجواب **غير معروف مسبقاً**.
    - إذا وصلت رسالة "ابدأ" أولاً، سيقوم النظام بعمل شيء. وإذا وصلت رسالة "توقف" أولاً، سيقوم بعمل شيء آخر مختلف تماماً. سلوك النظام يعتمد على التأخيرات الزمنية غير المتوقعة في الاتصالات، وهذا هو جوهر اللا حتمية.
- **"Two different implementations of the same GALS model can behave differently." تطبيقان مختلفان لنفس نموذج GALS يمكن أن يتصرفا بشكل مختلف.**
  - هذه هي النتيجة العملية الخطيرة للا حتمية.
  - لنفترض أننا صممنا نظام GALS ، ثم قمنا ببنائه:
    - **النسخة الأولى** :باستخدام معالج من شركة Intel وشبكة من شركة Cisco .
    - **النسخة الثانية** :باستخدام نفس التصميم المنطقي تماماً، ولكن بمعالج من شركة AMD وشبكة من شركة Juniper .
  - بما أن الخصائص الزمنية للعتاد مختلفة، فإن تأخيرات الاتصال وسرعات المعالجة ستكون مختلفة. هذا قد يؤدي إلى أن تصل الرسائل بترتيب مختلف في النسخة الثانية عما كانت عليه في النسخة الأولى.
  - **النتيجة** :قد نجد أن النسخة الأولى تعمل بشكل سليم، بينما النسخة الثانية تظهر سلوكاً خاطئاً في بعض الحالات النادرة، على الرغم من أن كلاهما تطبيق لنفس التصميم.

الخلاصة النهائية : نختار نموذج GALS لأنه يسمح لنا ببناء أنظمة كبيرة ومعقدة بشكل عملي. لكننا نقوم بذلك على حساب التخلي عن ضمان الحتمية العالمية. هذا يعني أن على المهندس الذي يصمم هذه الأنظمة أن يكون واعياً جداً لهذه اللاحتمية، وأن يصمم بروتوكولات

تواصل قوية (مثل آليات المصافحة Handshaking) تضمن أن النظام يعمل بشكل صحيح وآمن **بغض النظر** عن الترتيب الدقيق الذي تصل به الرسائل.

GALS Systems



- **A GALS model: FSM_1 and FSM_2 communicate through a single-slot buffer.**
- **FSM_1 outputs a signal (writes into the buffer) every 2 ms (we neglect communication time).**
  1. **If the reaction time of FSM_2 is 6ms, every third signal from FSM_1 will be reacted on.**
  2. **If we have a faster implementation of FSM_2, with reaction time 2ms, every signal from FSM_1 will be captured.**

هذه الشريحة GALS تقدم لنا مثالاً رقمياً بسيطاً يوضح العواقب العملية للتواصل غير المتزامن.

مثال على نظام GALS المنتج والمستهلك :

لنتفرض النظام الموضح في المخطط:

- لدينا مكون منتج FSM1 ومكون مستهلك FSM2
- يتواصلان عبر **مخزن مؤقت ذي خانة واحدة (single-slot buffer)** . هذا يعني أن قناة الاتصال بينهما لا يمكنها أن تحمل سوى **رسالة واحدة فقط** في كل مرة.

شروط التجربة:

- FSM1 (المنتج) يضع إشارة جديدة في المخزن المؤقت كل **2 ملي ثانية** بانتظام.
- FSM2 (المستهلك) يحتاج إلى بعض الوقت لمعالجة الإشارة بعد أخذها من المخزن.

الآن، سنرى سيناريوهين مختلفين بناءً على سرعة المستهلك FSM2 :

السيناريو الأول: المستهلك بطيء

- زمن ردة فعل FSM2 هو **6 ملي ثانية** .أي أنه يحتاج 6 ملي ثانية كاملة لمعالجة كل إشارة يستلمها.

ماذا سيحدث؟ لنتتبع الزمن:

- **عند الزمن ms0 :** FSM1 تضع الإشارة رقم 1 في المخزن. المخزن الآن ممتلئ. تبدأ FSM2 في معالجتها، وستنتهي عند الزمن ms6
- **عند الزمن ms2 :** FSM1 تحاول وضع الإشارة رقم 2. لكنها تجد أن المخزن **ممتلئ** (لأن FSM2 لا تزال مشغولة بالإشارة الأولى). لا يمكن وضع الإشارة الجديدة، وبالتالي **تضيع الإشارة رقم 2**.
- **عند الزمن ms4 :** FSM1 تحاول وضع الإشارة رقم 3. لكن المخزن **لا يزال ممتلئاً** . **تضيع الإشارة رقم 3** أيضاً.
- **عند الزمن ms6 :**
  - تنهي FSM2 عملها على الإشارة رقم 1 وتفرغ المخزن.
  - **في نفس هذه اللحظة**، تحاول FSM1 وضع الإشارة رقم 4. تجد المخزن فارغاً، فتضع الإشارة فيه بنجاح.
  - تبدأ FSM2 في معالجة الإشارة رقم 4، وستستمر الدورة على هذا النحو.

النتيجة : لأن المستهلك أبطأ بثلاث مرات من المنتج، فإنه لا ينجح إلا في التقاط **إشارة واحدة من كل ثلاث إشارات** (S1, S4, S7, ...). هذا يوضح كيف يمكن أن يؤدي عدم تطابق السرعات في نظام GALS إلى فقدان البيانات.

السيناريو الثاني: المستهلك سريع

- زمن ردة فعل FSM2 هو **2ملي ثانية** .إنه سريع تماماً مثل المنتج.

ماذا سيحدث؟ لنتتبع الزمن:

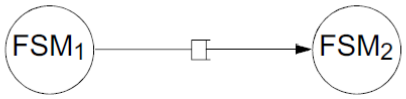
- **عند الزمن ms0 :** FSM1 تضع الإشارة رقم 1 في المخزن. تبدأ FSM2 في معالجتها، وستنتهي عند الزمن ms.2
- **عند الزمن ms 2 :** تنهي FSM2 عملها وتفرغ المخزن. في نفس اللحظة، تكون FSM1 جاهزة بالإشارة رقم 2 وتجد المخزن فارغاً، فتضعها فيه.
- **عند الزمن ms 4 :**تتكرر العملية بشكل مثالي.

النتيجة :بما أن سرعة المستهلك تتوافق مع سرعة المنتج، يتم تفريغ المخزن المؤقت في الوقت المناسب تماماً لوصول الإشارة الجديدة .**لا يتم فقدان أي إشارة**، والنظام يعمل بشكل موثوق.

هذا المثال يوضح أن صحة وموثوقية الاتصال في أنظمة GALS تعتمد بشكل حاسم على **السرعات النسبية للمكونات وحجم المخازن المؤقتة** بينها.

GALS Systems

- **Each individual FSM can still be verified and formal methods can be used.**
- **However, individual correctness of each FSM does not guarantee the correctness of the whole system. The system behaves correctly only if, in addition, certain assumptions regarding the timing of components and of communications are satisfied.**



- **Each FSM can be functionally verified individually.**
- **The global system will be correct (no signal is lost) if FSM_2 has a reaction time which is smaller than the production rate of FSM_1.**
- **Estimation and simulation can be used in order to verify that a certain implementation (like FSM_1 as software on a certain μprocessor, and FSM_2 as an ASIC) satisfies this assumption.**

بعد أن عرفنا أن أنظمةGALS غير حتمية وقد تفقد البيانات، السؤال هو: كيف يمكننا إذاً أن نثق بها؟

التحقق من صحة أنظمة GALS

- **"Each individual FSM can still be verified and formal methods can be used." لا يزال من الممكن التحقق من كل آلة حالة فردية، ويمكن استخدام الأساليب الرسمية.**
  - بما أن كل "جزيرة (FSM1, FSM2) "هي نظام متزامن بحد ذاتها، فلا يزال بإمكاننا استخدام أدوات التحقق الرسمي القوية لإثبات صحة المنطق الداخلي لكل مكون على حدة. يمكننا أن نثبت رياضياً أن FSM1 تقوم بوظيفتها الداخلية بشكل صحيح، وأن FSM2 كذلك.
- **"However, individual correctness of each FSM does not guarantee the correctness of the whole system." الصحة الفردية لكل آلة لا تضمن صحة النظام بأكمله.**
  - **التحدي:** أن يكون كل مكون مثالياً لا يعني أن تفاعلهم معاً سيكون مثالياً. صحة النظام ككل تعتمد على **افتراضات إضافية تتعلق بالتوقيت** بين هذه المكونات.

مثال عملي توضيحي

- **يمكننا التحقق وظيفياً من كل آلة على حدة .** حيث يمكن أن نثبت أن FSM1 تنتج الإشارات الصحيحة، وأن FSM2 تعالجها بشكل صحيح عندما تستلمها.

- **النظام الكلي سيكون صحيحاً (لن تضيع أي إشارة) فقط إذا تحقق شرط التوقيت** .في مثالنا السابق، كان الشرط هو: أن يكون **زمن ردة فعل المستهلك (FSM2) أصغر من معدل إنتاج المنتج(FSM1)**
- **"Estimation and simulation can be used in order to verify..."**
  - بما أننا لا نستطيع "إثبات" صحة التوقيت للنظام الكلي بشكل رسمي، فإننا نلجأ إلى أساليب هندسية عملية للتحقق من أن افتراض التوقيت هذا صحيح في التنفيذ الفعلي:
    - **التقدير (Estimation)** :يقوم المهندسون بتحليل التنفيذ. مثلاً، إذا كانت FSM1 ستُنفذ كبرنامج على معالج معين، يمكن تقدير أسوأ حالة لزمن تنفيذها. وإذا كانت FSM2 ستُنفذ كدارة عتادية مخصصة (ASIC)، يمكن حساب زمن ردة فعلها بدقة.
    - **المحاكاة (Simulation)** : نقوم بتشغيل محاكاة للنظام بأكمله باستخدام هذه الأزمنة المقدرة، ونعرضه لظروف تشغيل مختلفة، لنتأكد من أن شرط التوقيت (المستهلك أسرع من المنتج) لا يتم خرقه، وأنه لا يتم فقدان أي بيانات.

هذه الاستراتيجية الهجينة (تحقق رسمي محلياً، ومحاكاة وتقدير عالمياً) هي النهج العملي الذي يسمح لنا ببناء أنظمة كبيرة ومعقدة وموثوقة باستخدام نموذج GALS

GALS System Models

- **A GALS system is modelled as a *network of FSMs*:**
  - **Each FSM has a *locally synchronous* behavior: it executes a transition by producing a single output reaction based on a single, snap-shot input assignment in zero time.**
  - **A System has a globally asynchronous behavior: each FSM reads inputs, executes a transition, and produces outputs in a finite amount of time as seen by the rest of the system.**

نماذج أنظمة GALS

هذه الشريحة تقدم لنا كيفية بناء "نماذج" لأنظمة GALS

- **"A GALS system is modelled as a network of FSMs"يتم نمذجة نظام GALS كشبكة من آلات الحالة المحدودة**
  - الفكرة الأساسية هي أننا ننظر للنظام كشبكة من المكونات (أو الجزر)، حيث كل مكون هو آلة حالة محدودة.

السلوك المزدوج للنظام

النموذج له وجهان أو رؤيتان للزمن، وهذا هو جوهر قوة GALS

1- **السلوك المتزامن محلياً (Locally Synchronous)**

- **داخلياً**، كل آلة حالة (FSM) تتصرف كنظام متزامن مثالي.
- هي تأخذ **"لقطة فورية "**لجميع مدخلاتها في لحظة معينة، وبناءً عليها، تنتج ردة فعل (مخرج) في **"صفر" من الوقت المنطقي**.
- هذا يعني أن كل جزيرة، عند النظر إليها بمعزل عن غيرها، هي عالم مثالي، حتمي، ويمكن التحقق منه.

2- **السلوك غير المتزامن عالمياً (Globally Asynchronous)**

- **خارجياً**، أو من وجهة نظر بقية مكونات النظام، فإن هذه العملية ليست فورية على الإطلاق.
- الوقت الذي تستغرقه الآلة لقراءة مدخلاتها، وتنفيذ انتقالها، وإصدار مخرجاتها هو **وقت حقيقي محدود وغير صفري**.
- **مثال توضيحي :** ردة فعل الإنسان عند لمس شيء ساخن.
  - **داخلياً (محلياً) :**بالنسبة لجهازنا العصبي، قد تبدو الاستجابة "فورية" وتلقائية.
  - **خارجياً :**بالنسبة لشخص يراقبنا، هناك وقت حقيقي وقابل للقياس بين لحظة اللمس ولحظة سحب يدنا.

الخلاصة :نموذج GALS يجمع بذكاء بين رؤيتين: رؤية داخلية مثالية ومتزامنة لكل مكون (مما يسهل التصميم)، ورؤية خارجية واقعية وغير متزامنة للتفاعلات بين المكونات (مما يعكس حقيقة العالم المادي).

GALS System Models

- **FSMs communicate through signals.**
  - **A signal, in general, carries an event and associated data.**
  - **A signal is communicated between two FSMs via a connection that has an associated input buffer.**
  - **A sender can communicate a signal to several receivers; each receiver buffers the signal in its own input buffer (of a certain size) associated to the connection.**
  - **Communication is asynchronous and has undefined (finite) delays. Each input buffer stores the most recently received events and values.**
  - **In general, the transmitter sends without waiting for the receiver; nothing prevents the transmitter from sending a new event before the last one was consumed and, thus, potentially, overwriting it.**

آليات التواصل في نماذجGALS

- **"FSMs communicate through signals." تتواصل آلات الحالة المحدودة عبر الإشارات.**

- "الإشارة" هي الرسالة أو المعلومة التي يتم تبادلها بين المكونات.

- **"A signal, in general, carries an event and associated data." تحمل الإشارة، بشكل عام، حدثاً وبيانات مرتبطة به.**
  - الإشارة ليست مجرد نبضة فارغة. هي حزمة تحتوي على شيئين:
    1. **حدث (Event)** : إشعار بوقوع شيء ما (مثلاً، "وصلت سيارة جديدة").
    2. **بيانات (Data)** : معلومات إضافية مرتبطة بالحدث (مثلاً، رقم لوحة السيارة).

- **"A signal is communicated... via a connection that has an associated input buffer." يتم توصيل الإشارة عبر اتصال له مخزن مؤقت للمدخلات.**
  - كما ذكرنا، الاتصال بين المكونات ليس فورياً. لذلك، يوجد عند كل مستقبل "صندوق بريد" أو **مخزن مؤقت (buffer)** يستقبل الإشارات ويحتفظ بها حتى يكون المستقبل جاهزاً لمعالجتها.

- **"A sender can communicate a signal to several receivers..."**
  - يمكن لمكون واحد أن يرسل نفس الإشارة إلى عدة مكونات أخرى في نفس الوقت (بث). كل مستقبل لديه مخزنه المؤقت الخاص به لاستلام نسخته من الإشارة.

- **"Communication is asynchronous and has undefined (finite) delays." الاتصال غير متزامن وله تأخيرات غير محددة (لكنها محدودة).**
  - هذا هو جوهر اللاتزامن. لا نعرف بالضبط كم من الوقت ستستغرق الإشارة لتصل، لكننا نعرف أنها ستصل في النهاية. وظيفة المخزن المؤقت هي التعامل مع هذا التأخير غير المحدد.

- **"In general, the transmitter sends without waiting for the receiver..." بشكل عام، المرسل يرسل دون انتظار المستقبل**
  - المرسل (transmitter) لا ينتظر ليتأكد من أن المستقبل قد استلم الرسالة أو أنه مستعد لها. هو فقط "يرمي" الرسالة في قناة الاتصال.
  - **"...nothing prevents the transmitter from sending a new event before the last one was consumed and, thus, potentially, overwriting it...." لا شيء يمنع المرسل من إرسال حدث جديد قبل استهلاك الحدث الأخير، وبالتالي، من المحتمل أن يقوم بالكتابة فوقه.**
  - هذه هي المخاطرة الكبرى في هذا النموذج. إذا كان المخزن المؤقت ممتلئاً (لأن المستقبل بطيء)، وقام المرسل بإرسال رسالة جديدة، فقد يتم "الكتابة فوق" الرسالة القديمة التي لم تُقرأ بعد، وبالتالي تضيع تلك المعلومة. هذا بالضبط ما رأيناه في المثال الرقمي للمستهلك البطيء.

الخلاصة :نموذج الاتصال في GALS هو نموذج واقعي. هو يقر بأن الاتصالات تستغرق وقتاً وأن المكونات قد تعمل بسرعات مختلفة. هذا يضع على عاتق المصمم مسؤولية التعامل مع هذه التحديات، مثل تصميم مخازن مؤقتة بحجم كافٍ أو استخدام بروتوكولات تضمن عدم فقدان البيانات الهامة.

GALS System Models

- **A FSM reacts when at least one event is available on any of its inputs; in this case the FSM**
  - **reads and consumes the available input signal(s);**
  - **identifies the matching transition and performs the corresponding state transition with the associated action set;**
  - **writes the outputs associated to the transition.**
- **The reaction takes a certain, finite, amount of time.**

  After executing a transition, the FSM will be ready to react to new inputs.

  Question: When? Immediately, just after it finished the current transition?
  Answer: Not necessarily!

  When a certain FSM is ready to check inputs and react, depends on the, execution platform, the execution times, periods, and the scheduling policy used at implementation.

هذه الشريحة تجيب على سؤالين :**ماذا تفعل** آلة الحالة المحدودة (FSM) عندما تتلقى حدثاً؟ و **متى** تتمكن من القيام بذلك مرة أخرى؟

دورة حياة ردة الفعل لآلة الحالة(FSM)

الجزء الأول من الشريحة يصف الخطوات المنطقية التي تقوم بها أي آلة حالة (أو "جزيرة متزامنة") عندما "تستيقظ" لتعالج حدثاً ما:

1. **تقرأ وتستهلك الإشارة :**أولاً، تقوم بقراءة الحدث المتوفر في "صندوق بريدها" (المخزن المؤقت)، وتستهلكه حتى لا تتم معالجته مرة أخرى.
2. **تحدد وتنفذ الانتقال :**ثانياً، بناءً على حالتها الحالية والحدث الذي قرأته، تبحث عن الانتقال المناسب في منطقها وتقوم بتنفيذه. هذا يشمل الانتقال إلى حالة جديدة وتنفيذ أي إجراءات داخلية (مثل تحديث متغير).
3. **تكتب المخرجات :**أخيراً، تقوم بتوليد أي مخرجات مرتبطة بهذا الانتقال وإرسالها إلى المكونات الأخرى.

هذه الدورة الكاملة، كما ذكرنا، تستغرق **وقتاً حقيقياً محدوداً وغير صفري**.

السؤال الأهم: متى تكون الآلة جاهزة مرة أخرى؟

بعد أن تنهي آلة الحالة دورتها وتصبح جاهزة لمعالجة مدخلات جديدة **" هل تكون جاهزة للعمل فوراً بعد انتهاء انتقالها الحالي؟"**

والجواب هو :**ليس بالضرورة.**

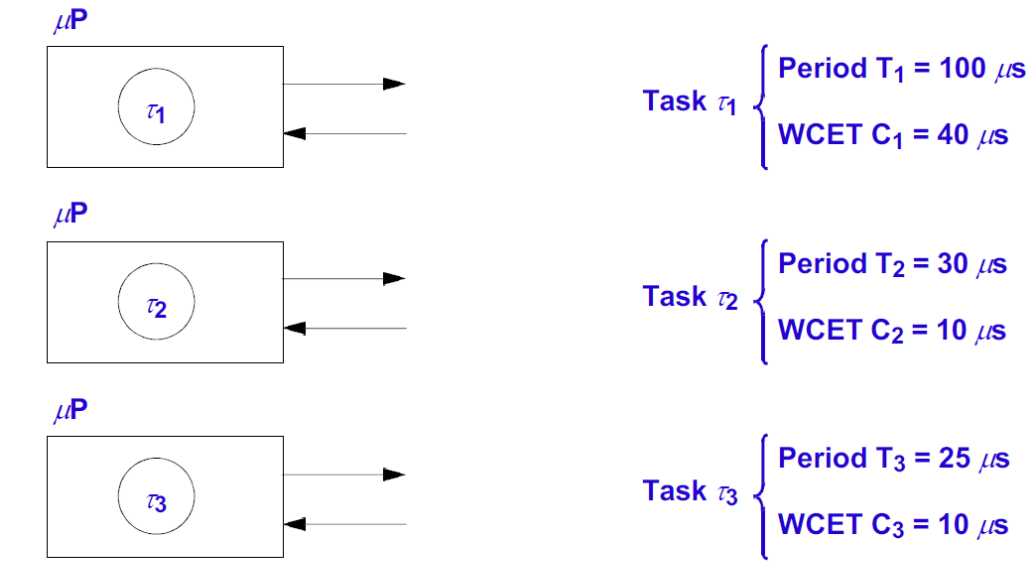
لماذا؟ لأن جاهزية آلة الحالة لتفحص مدخلاتها والتفاعل معها لا تعتمد عليها وحدها. في نظام برمجي حقيقي، هناك "مدير" أعلى ينسق العمل، وهذا هو **منظم المهام (Scheduler)** الخاص بنظام التشغيل .الأمر يعتمد على عدة عوامل خارجية:

- **منصة التنفيذ (Execution Platform)** : ما هي سرعة المعالج الذي يعمل عليه البرنامج؟
- **أزمنة وفترات التنفيذ (Execution Times/Periods)** : كم من الوقت تستغرق كل مهمة؟ وهل هي مجدولة لتعمل بشكل دوري (مثلاً، كل 10 مللي ثانية)؟
- **سياسة الجدولة (Scheduling Policy)** : كيف يقرر المنظم توزيع وقت المعالج على المهام المختلفة؟ هل يعطي الأولوية لمهام معينة على حساب أخرى؟

مثال توضيحي :لنتخيل أن آلة الحالة هي موظف في مكتب، والمعالج هو "المكتب" الذي يعمل عليه. حتى لو أنهى الموظف مهمته الحالية، لا يمكنه البدء في مهمة جديدة إلا إذا سمح له "مدير المكتب" (المنظم) باستخدام "المكتب" (المعالج). قد يكون المدير قرر إعطاء المكتب لموظف آخر لديه مهمة أكثر إلحاحاً.

الخلاصة :في نظام GALS برمجي، لا يكفي أن تكون المهمة جاهزة، بل يجب أن يمنحها **منظم المهام** فرصة للعمل على المعالج. وهذا يضيف طبقة أخرى من التحليل واللا حتمية التي يجب على مهندس النظم أخذها في الحسبان.

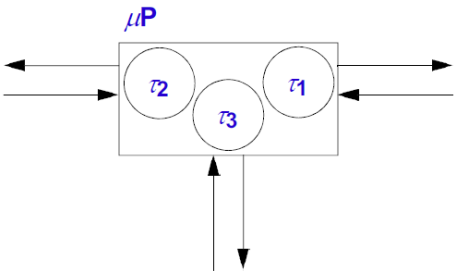
GALS System Models



Each task implements an FSM (in software).

Works! No problem!

GALS System Models



Task τ_1 { Period T_1 = 100 μs ; WCET C_1 = 40 μs }

Task τ_2 { Period T_2 = 30 μs ; WCET C_2 = 10 μs }

Task τ_3 { Period T_3 = 25 μs ; WCET C_3 = 10 μs }

Does this work?
Can each of the tasks work at the required rate (period)?

- τ_1 needs to run for 40 μs every 100 μs: 40% of CPU
- τ_2 needs to run for 10 μs every 30 μs: 33% of CPU
- τ_3 needs to run for 10 μs every 25 μs: 40% of CPU

Total: 113%
This will not work!

If the total utilisation is not larger than 100% it is possible to implements the tasks!

هذا المثال الأخير يجمع كل ما تعلمناه و يطرح السؤال الهندسي الأهم: هل تصميمنا قابل للتنفيذ في العالم الحقيقي؟

المشكلة: مهام متعددة، معالج واحد

لدينا ثلاث مهام (τ_1، τ_2، τ_3) كل منها تنفذ آلة حالة (FSM) معينة. في عالم مثالي، يمكننا تخصيص معالج منفصل لكل مهمة، وكما تشير الشريحة الأولى، "الأمر سينجح لا توجد مشكلة". كل مهمة ستعمل بحرية وتلبي مواعيدها النهائية بسهولة.

لكن في الواقع، التكلفة والموارد تفرض علينا تشغيل كل هذه المهام على **معالج واحد فقط**. وهنا يبدأ التحدي الحقيقي الذي تطرحه الشريحة الرئيسية:

هل سينجح هذا؟ هل يمكن لكل مهمة أن تعمل بالمعدل المطلوب منها؟

التحليل: حساب عبء العمل

لكي نجيب على هذا السؤال، يجب أن نحلل "عبء العمل" أو **"استخدام المعالج (CPU Utilization)"** لكل مهمة.

- **المعطيات** :لكل مهمة خاصيتان:

1. **الفترة (Period T)** : هي المعدل الذي **يجب** أن تنفذ فيه المهمة T_1 = 100 µs. تعني أن المهمة الأولى يجب أن تنفذ مرة كل 100 مايكروثانية. هذا هو موعدها النهائي(deadline)

2. **أسوأ زمن للتنفيذ (WCET C)** :هو أقصى وقت تحتاجه المهمة من المعالج لإنجاز عملها عندما يتم تشغيلها.

- **الحسابات:**
  - **المهمة τ_1 :** تحتاج 40 مايكروثانية من وقت المعالج كل 100 مايكروثانية.
    - نسبة الاستخدام 0.40 = 100 / 40 :أي **40%** من طاقة المعالج.
  - **المهمة τ_2:** تحتاج 10 مايكروثانية كل 30 مايكروثانية.
    - نسبة الاستخدام 0.33 ≈ 30 / 10 :أي **33%** من طاقة المعالج.
  - **المهمة τ_3:** تحتاج 10 مايكروثانية كل 25 مايكروثانية.
    - نسبة الاستخدام 0.40 = 25 / 10 :أي **40%** من طاقة المعالج.
- **النتيجة النهائية:**
  - عندما نجمع عبء العمل الكلي على المعالج الواحد، نجد أن:
    - **المجموع = %40 + %33 + %40 = %113**
  - من المستحيل فيزيائياً أن يعمل المعالج بنسبة 113% من طاقته.
  - **إذاً، هذا التصميم بهذه المواصفات لن ينجح .(This will not work!)** واحدة على الأقل من هذه المهام (أو جميعها) ستفوت مواعيدها النهائية، مما قد يؤدي إلى فشل النظام.

قاعدة الجدولة الأساسية

الشريحة الثانية تقدم لنا القانون الذهبي في أنظمة الوقت الحقيقي، وهو شرط أساسي وبديهي **" إذا لم يكن الاستخدام الكلي للمعالج أكبر من 100%، فمن الممكن تنفيذ المهام"**

هذا يعني أنه لكي يكون هناك **أمل** في نجاح النظام، يجب أن يكون مجموع استخدام المعالج أقل من أو يساوي 100%. إذا تجاوز 100%، فالفشل مضمون. أما إذا كان أقل، فالنجاح **ممكن**، ولكنه ليس مضموناً، حيث يعتمد بعد ذلك على كفاءة "منظم المهام (scheduler) " الذي يوزع وقت المعالج على المهام المختلفة.