

Property Specification Language (PSL)

Kivonat és adaptáció a következő szerzőktől:

- Roderick Bloem (TU Graz)
- Cindy Eisner (IBM Haifa Research Labs)
- Dana Fisman (Weizmann Institute of Science)
- Avigail Orni (IBM Haifa Research Labs)
- Sitvanit Ruah (IBM Haifa Research Labs)



A PSL nyelv



- **Deklaratív nyelv tulajdonságok specifikálására**
 - Tömör, intuitív, temporális követelményeket támogat
- **Kombinálja a következőket:**
 - temporális logika és
 - reguláris kifejezések,
 - mindezeket kényelmes szintaxis kiegészítések mellett
- **Használható:**
 - Specifikáció precíz dokumentálása
 - Modellellenőrzés bemenete
 - On-line monitorok automatikus generálása

A nyelv története

1994

CTL szintaxis kiterjesztése

1995

Reguláris kifejezések hozzáadása

1997

On-line monitorok automatikus generálása

Sugar 1.0

2001

LTL jellegű szemantikai kiterjesztések

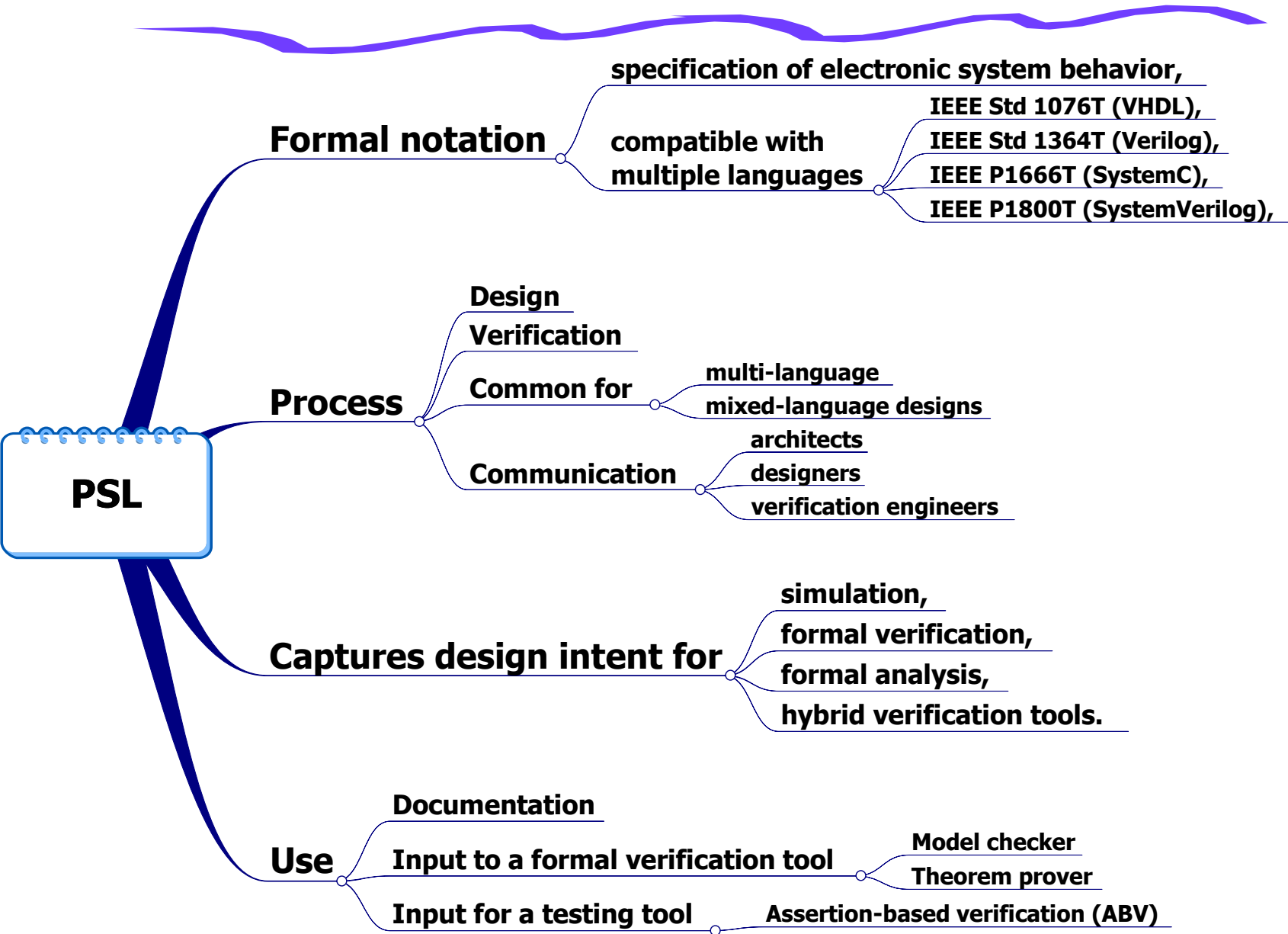
Sugar 2.0

2002

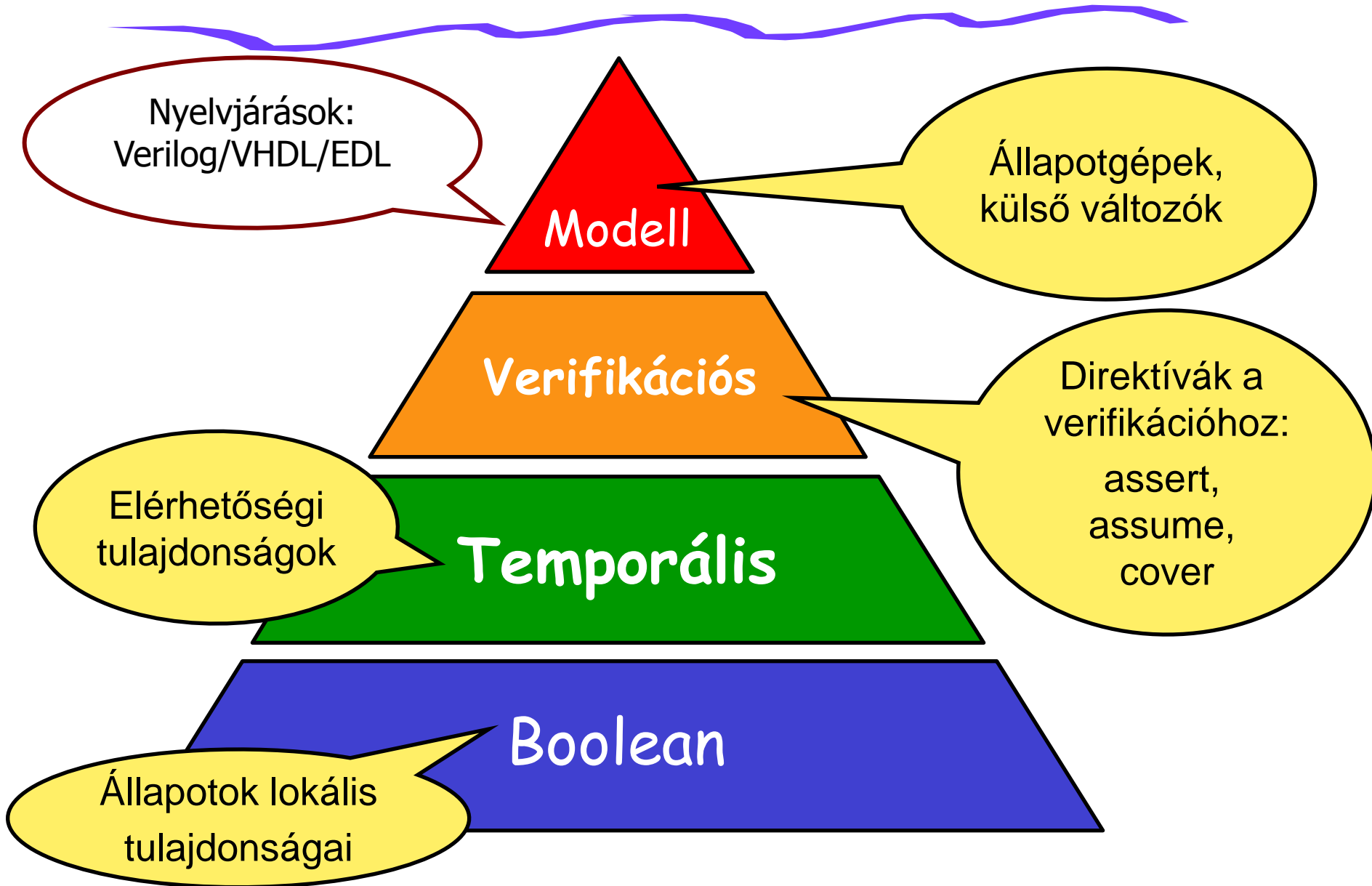
IEEE szabványosítás indul (PSL) az Accellera által

PSL

IEEE Std 1850-2005



A nyelv felépítése



A temporális réteg

Építőelemek (atomok)

Boolean Expressions



Sugar Extended Regular Expressions (SERE)

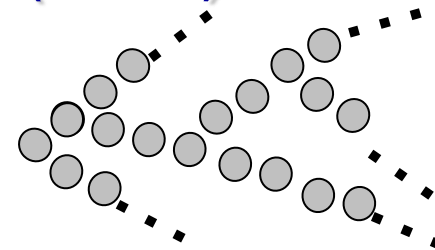


Temporális kifejezések

Sugar Foundation Language



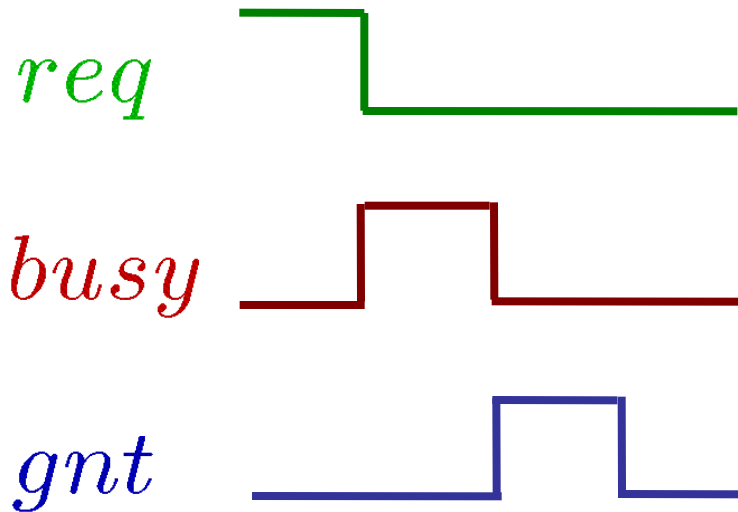
Optional Branching Extension (OBE)



SERE – Példa 1

$\{req; busy; gnt\}$

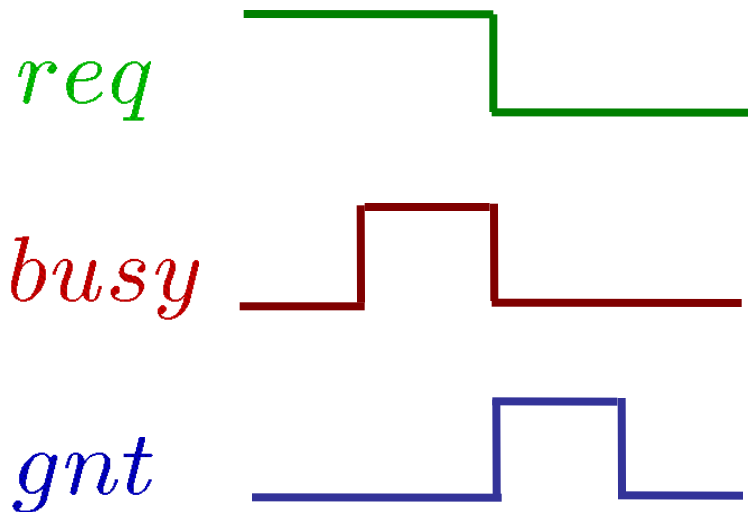
Egy SERE leírja állapotok egy sorozatát (ezek idődiagramokkal is megjeleníthetők)



Ezt a diagramot az adott SERE írja le

SERE – Példa 1

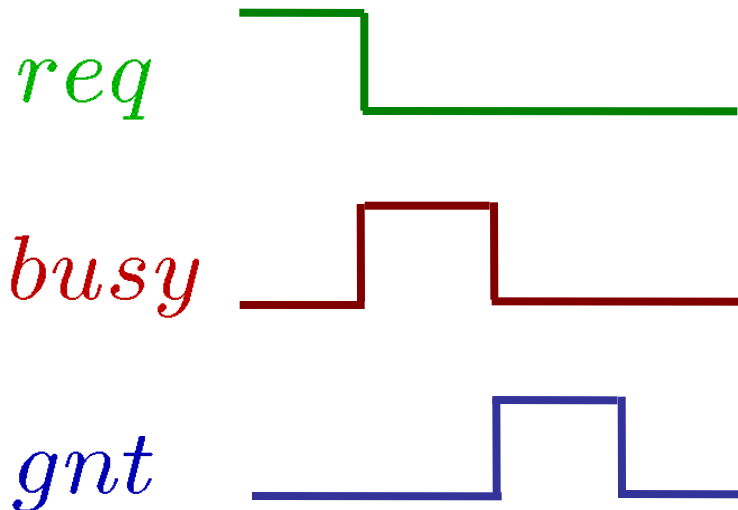
$\{req; busy; gnt\}$



Ezt a diagramot
is az adott SERE
írja le

SERE – Példa 1

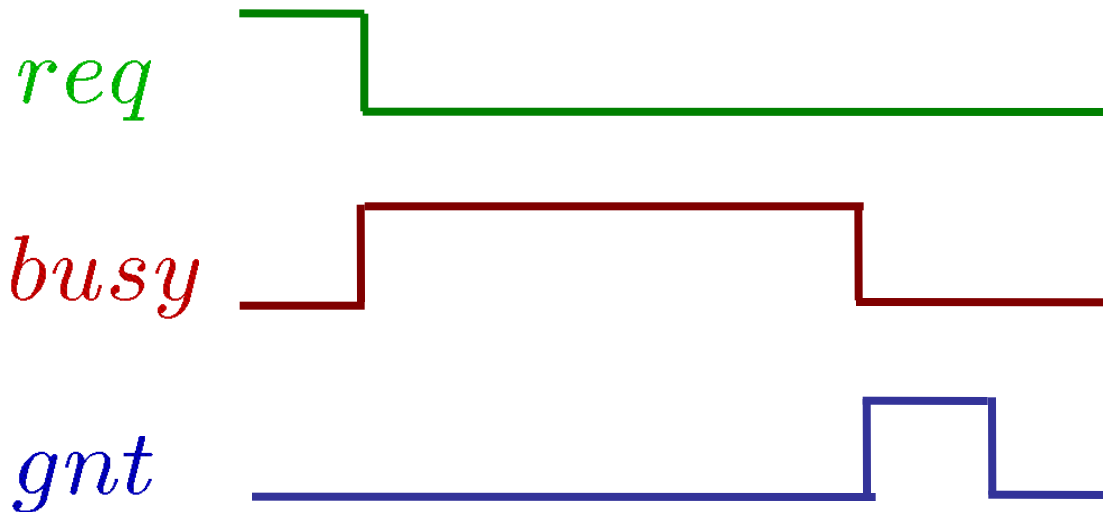
```
{ req & !busy & !gnt;  
  !req & busy & !gnt;  
  !req & !busy & gnt }
```



Csak az első diagram
leírásához módosítani kell a
kifejezést

SERE – Példa 2

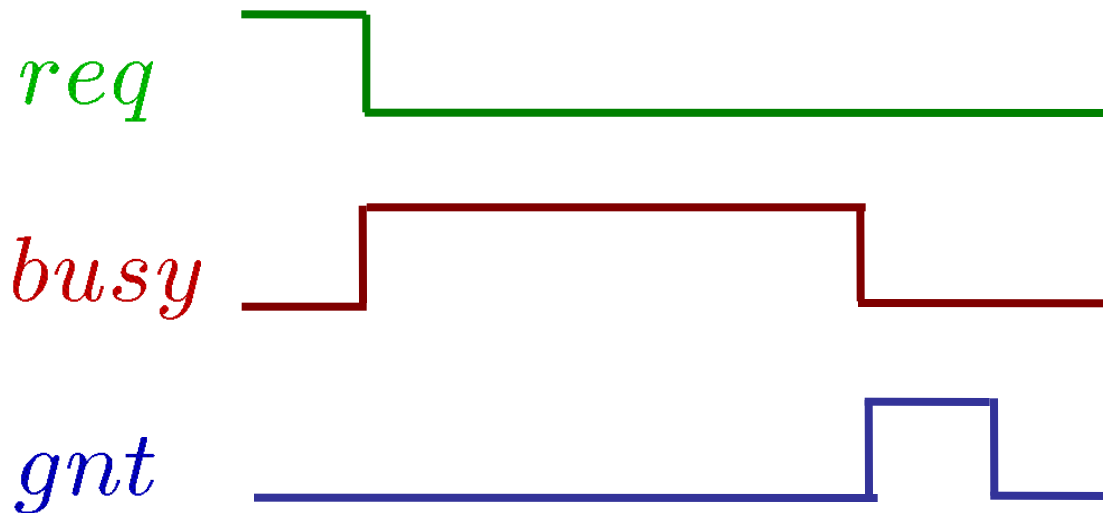
$\{req; busy; busy; busy; busy; gnt\}$



SERE – Példa 2

$\{req; busy[*4]; gnt\}$

A *busy* állapot
4-szer



SERE – Példa 3

$\{req; busy[*3..5]; gnt\}$

A *busy* állapot előfordulása 3..5-ször

$\{req; busy[*]; gnt\}$

A *busy* állapot előfordulása tetszőleges számban

Temporális operátorok

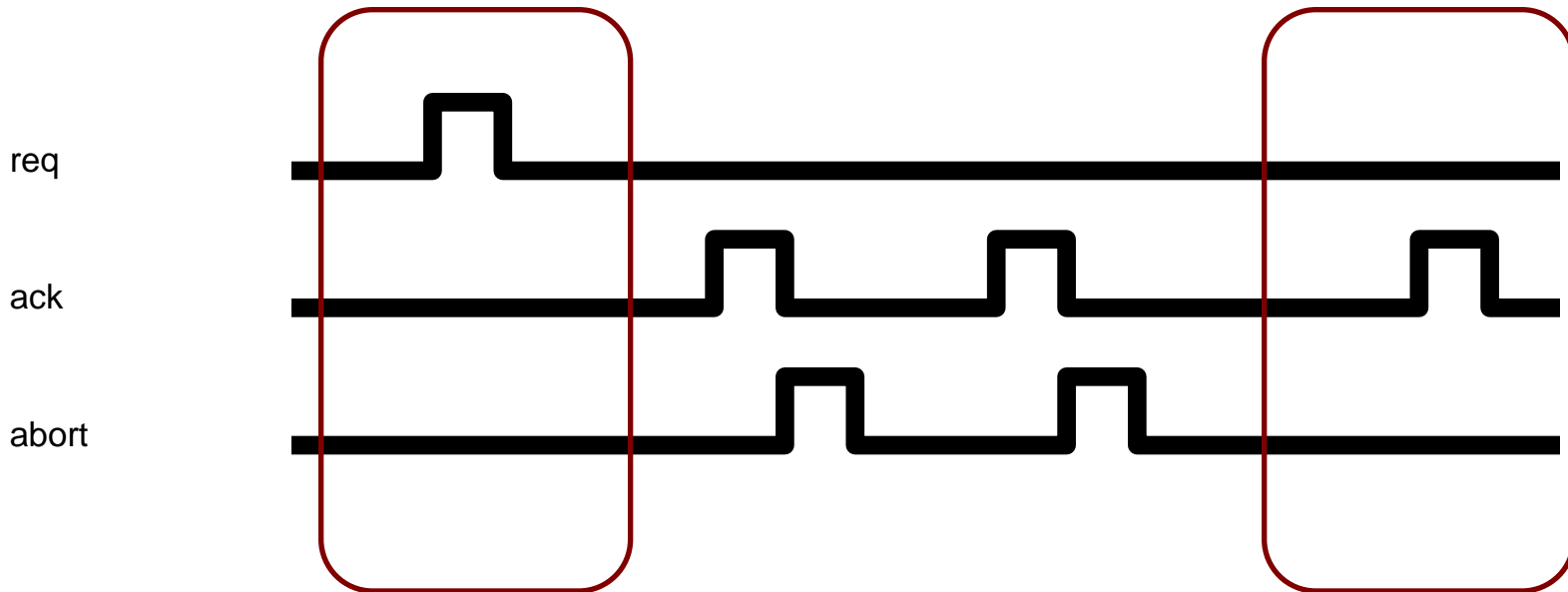


- always
- never
- next
- eventually!
- until (gyenge until)
- until! (erős until)
- until_ (közös bekövetkezés)
- before

Temporális követelmények

- Ha **req** bekövetkezik, akkor valamikor egy **ack** következik, amit nem követ **abort**.

always (req -> eventually! {ack ; !abort})



Temporális követelmények

- `ena` és `enb` soha nem fordulnak elő egyszerre:

never (ena & enb)

- Ha `req` bekövetkezik, akkor 4 állapottal később `ack` is

always (req -> next[4] ack)

- Ha `req` bekövetkezik, akkor valamikor `ack` is

always (req -> eventually! ack)

Összetett tulajdonságok

A szuffix implikáció operátor

$$\{S E R E_A\} \models \{S E R E_B\}$$

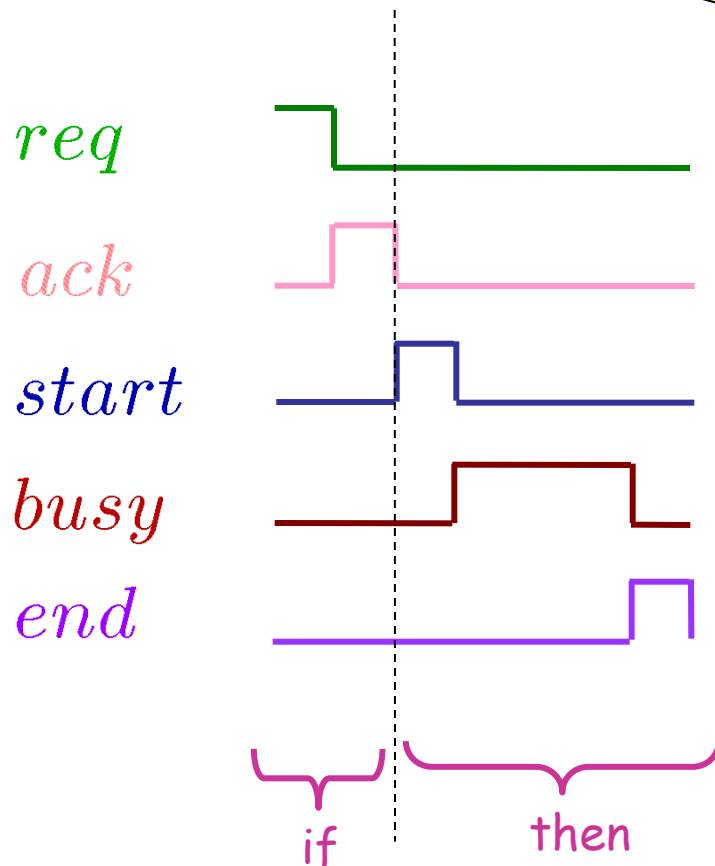
ha most elkezdődik a $S E R E_A$ útvonal
akkor a folytatása $S E R E_B$ lesz

\models esetén az útvonalak nem lapolódhatnak át

\dashv esetén az útvonalak átlapolódhatnak

Összetett tulajdonságok – Példa

$$\{req; ack\} \Rightarrow \{start; busy[*]; end\}$$

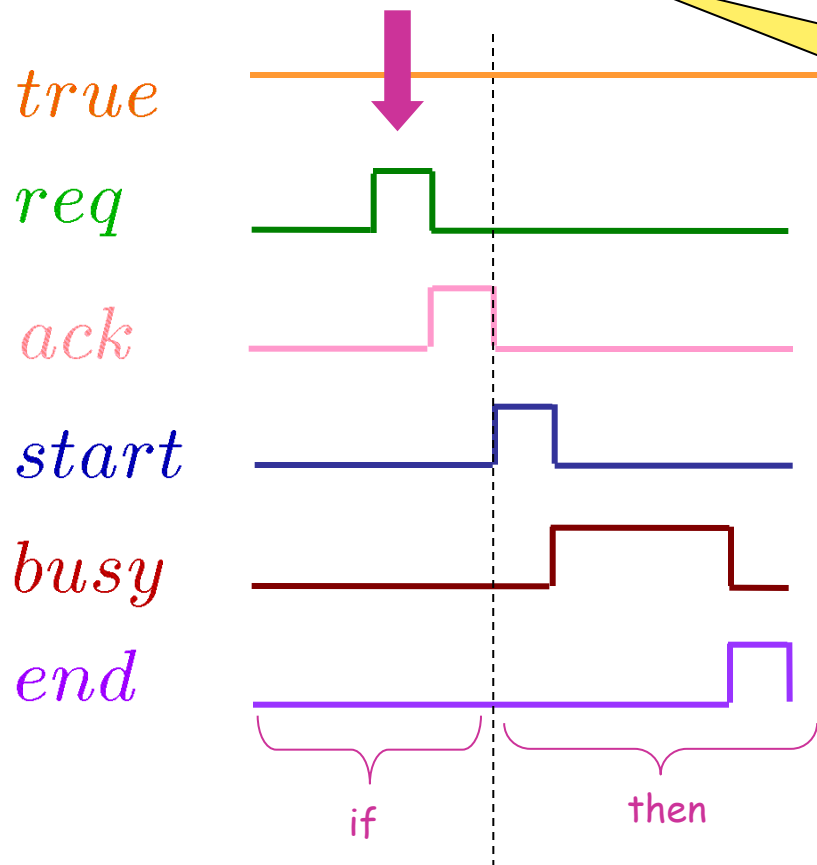


Ez esetben a *req* az első állapot kell legyen

Lehet persze *true*[*] a SERE elején

Összetett tulajdonságok – Példa

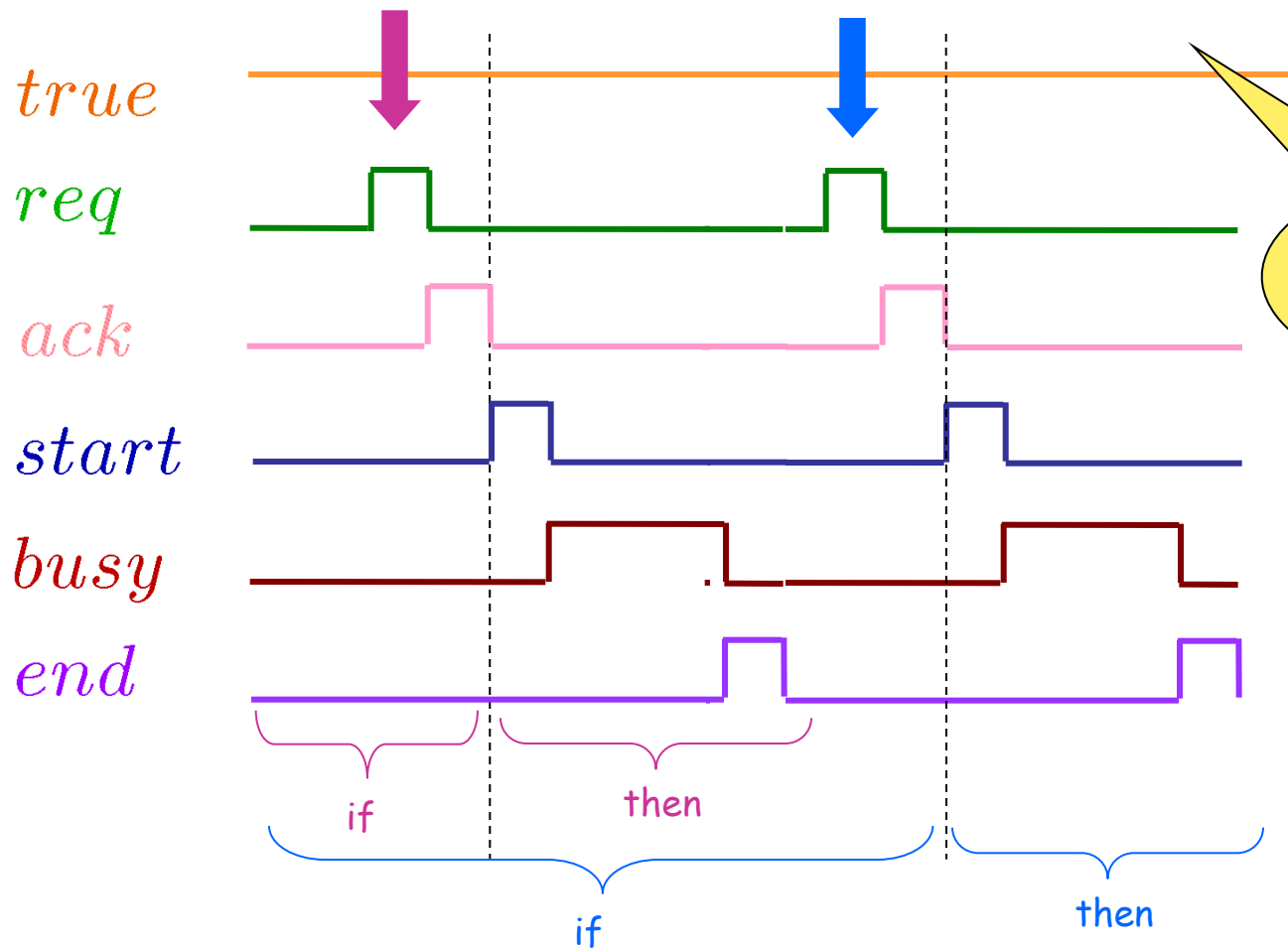
$$\{ \textit{true}[*]; \textit{req}; \textit{ack} \} \Rightarrow \{ \textit{start}; \textit{busy}[*]; \textit{end} \}$$



Tetszőleges helyen kezdődhet a megadott viselkedés

Összetett tulajdonságok – Példa

$$\{ \textit{true}[*]; \textit{req}; \textit{ack} \} \Rightarrow \{ \textit{start}; \textit{busy}[*]; \textit{end} \}$$



Kifejezőképesség



■ Elméletben:

Legalább olyan kifejező, mint a

- ◆ LTL
- ◆ CTL
- ◆ Reguláris kifejezések

■ Gyakorlatban:

Az Accellera által specifikált minden tulajdonság (hardver jelekre) kifejezhető volt

Megvalósítás



- Minden **PSL** tulajdonság redukálható **LTL** vagy **CTL** tulajdonsággá
- **CTL** és **LTL** esetén létezik modellellenőrző algoritmus
- **Szimuláció** esetén csak on-line ellenőrizhető részalmazt tekintenek (ez **LTL** alapú); az ilyen kifejezésekhez **elfogadó állapotgép** határozható meg

A verifikációs réteg



Direktívák:

- `assume <formula>`
- `assert <formula>`
- `cover <SERE>`

Példa:

```
vunit arb{  
    assume always (req1 → next(ack1 before req1));  
    assume always (req2 → next(ack2 before req2));  
    assert always (req1 → eventually! ack1 )  
    assert always (req2 → eventually! ack2 )  
    assert never(req1 & req2)  
}
```

A tulajdonságok strukturálása

Nyelvjárások:
Verilog, VHDL, ...

```
vunit example {
```

```
wire req;  
assign req = read_req | write_req ;
```

**Modeling
layer**

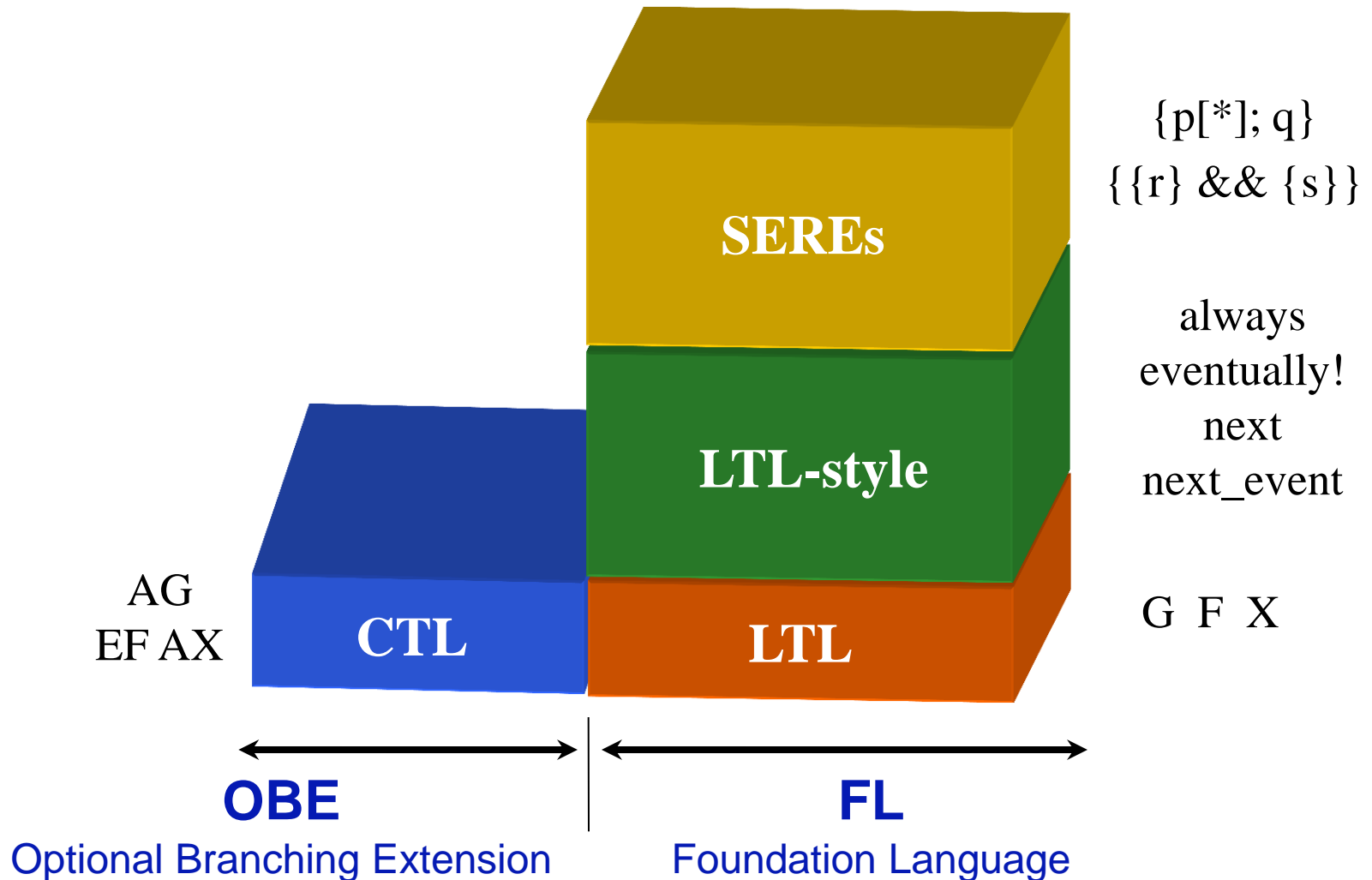
```
assert always (req -> next (ack | abort)) ;
```

**Verification
layer**

**Boolean
layer**

**Temporal
layer**

A temporális réteg (összefoglaló)



Eszközök

- @HDL: @Verifier/@Designer, Formal Property Checker
- Aldec Inc.: Riviera-2005.04, Assertion Checking Integrated in Mixed HDL Simulation Kernel
- Assertive Design: DesignPSL, PSL-based verification tool suite
- Atrenta, Inc.: PeriScope, Predictive Analysis, Assertion-Based Analysis and Functional Verification for RTL designs
- Averant: Solidify, Formal Property Checker
- Avery Design: TestWizard, Assertion Checking Integrated in Testbench Automation Tool
- Axis System: Assertion Processor, Assertion Checking Integrated in Emulation Engine
- Cadence: Incisive Unified Simulator, Simulator with Integrated Static and Dynamic Assertion Checking Capabilities
- Denali Software: MMAV, PureSpec Interface Verification IP
- Dolphin Integration: SMASH, Assertion Checking Packaged with Mixed-Signal Simulator
- Doulos Ltd.: PSL Training Courses
- Esperan: PSL Training Courses
- Esterel Technologies: Esterel Studio, Assertion Checking Integrated in System Modeling Platform
- Fintronics: FinSim, Assertion Checking Integrated in Simulation Engine
- FTL Systems: Auriga, Assertion Checking Integrated in Parallel, Mixed-Signal Simulator
- FTL Systems: Merlin, Asynchronous Behavioral Synthesis (Sugar extended for capturing properties of asynchronous design)
- IBM: RuleBase PE, Formal Property Checker
- IBM: FoCs, Assertion Checker for Simulation/Emulation (vendor independent)
- IBM: Sugar1to2, Translate Sugar 1.0 Assertions to PSL/Sugar
- Interra Systems: Beacon-PSL, PSL Test Suite
- Interra Systems: Cheetah/Jaguar, PSL Analysis/Parsing
- Jasper Design Automation: JasperGold, Verification System Formal Property Checker
- Jeda Technologies, Inc.: JEDAX, Assertion Checking Integrated in HVL
- Mentor Graphics: CheckerWare, Library of Protocol Checkers
- Mentor Graphics: ModelSim 5.8, Assertion Checking Integrated in Simulation Engine
- Nobug Consulting: Specification Compiler Sugar-to-'e' Translator
- Novas: Debussi/Verdi , Assertion-Based Debug Systems
- Real Intent: Verix, Formal Assertion-Based Verification System
- Structured Design Verification: TransactorWizard, Protocol Verification Tool
- Summit Design: Visual Elite,
- Synapticad: TestBencher Pro, Generation of Protocol Checkers
- Synapticad: Transaction Tracker, Specify Transaction Patterns
- Syntest Technologies: TurboCheck-RTL, Design-for-Test Rule Checking
- Temento Systems: Dialite, Dynamic assertion checking
- Tharas Systems: Hammer, Assertion Checking Integrated in Emulation Engine
- TNI/Valiosys: imPROVE-HDL, Formal Property Checker
- Verific Design Automation: PSL Parser/Analyzer
- Verity (Cadence): Specman Elite, Assertion Checking Integrated in Testbench Automation Tool
- Veritable: Verity-Check, Formal Property Checker
- Veritools: Undertow Suite, Integrated Debugging Environment

Néhány IBM felhasználás



■ IBM termékek

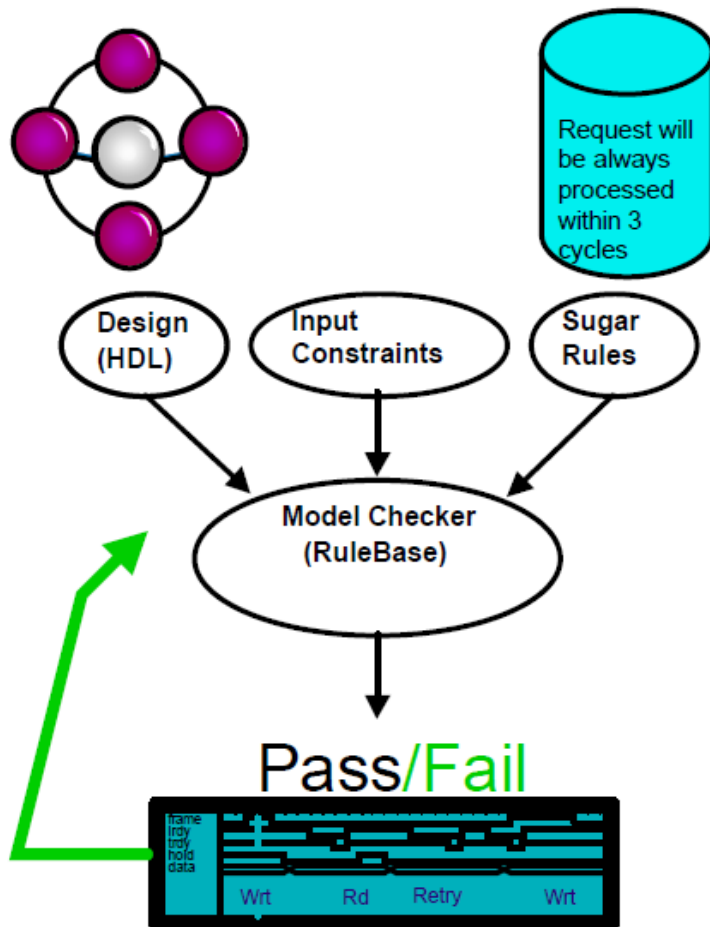
- ◆ Mainframe line (S/390)
- ◆ Mid-range line (AS/400)
- ◆ Workstation line (RS/6000)
- ◆ PC line (Netfinity)
- ◆ Super Computers (ASCI)
- ◆ ASIC/OEM business

■ Külső licenszek

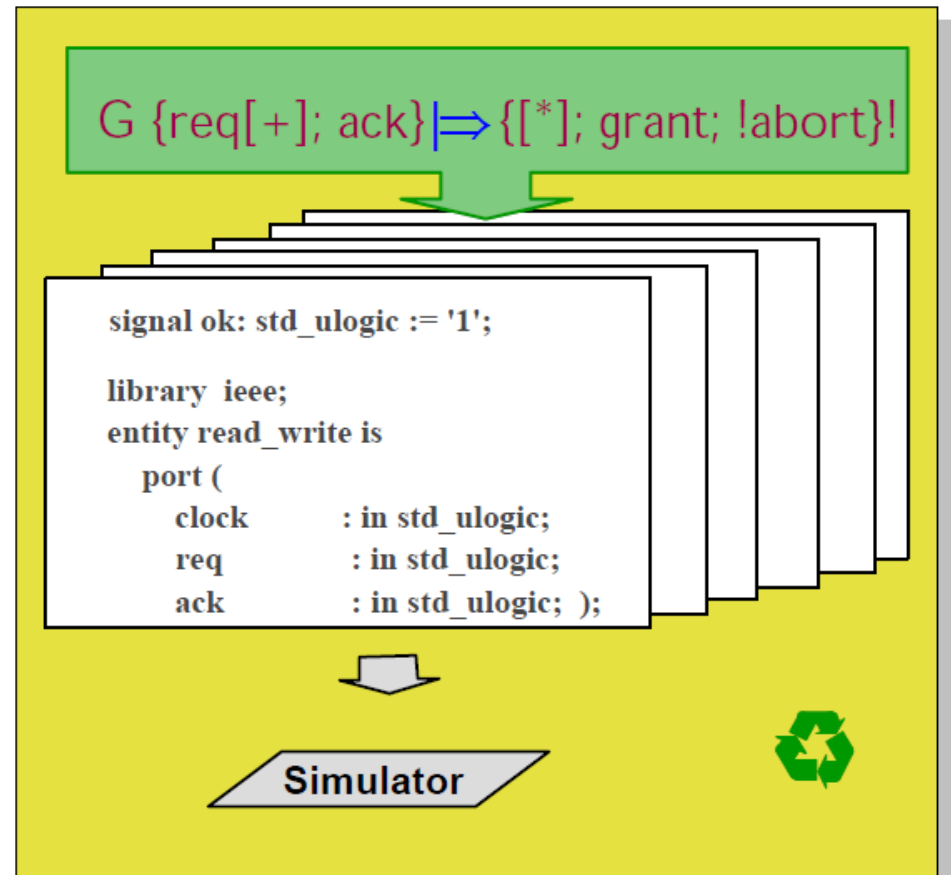
■ Egyetemi programok

Eszközök (IBM Haifa Design Labs)

Modellellenőrző (RuleBase):



Szimulációhoz monitorgenerálás (FoCs):



Eredmények (IBM Haifa Design Labs)

Ethernet kártya vezérlőjének tervezése:

- ✍ A core of about 400,000 gates
- ✍ 40% went through Formal Verification
- ✍ 3 verification engineers out of 10 practice Formal Verification (model checking)
- ✍ Formal Verification found 33% of documented design bugs
- ✍ Areas that went through Formal Verification had no bugs found in simulation
- ✍ Late FV found bugs in areas that were already heavily simulated

Tanulságok (IBM Haifa Design Labs)

Ethernet kártya vezérlőjének tervezése:

- ✍ Candidate modules for formal verification (static checking)
 - Much control, little data
 - Coverage is hard to reach through simulation (e.g. arbiters)
 - Modules that are buried inside the design, far from the simulation interfaces
- ✍ Where do we consider using FoCs (dynamic checking)
 - Protocol checking of chip interfaces and partition interfaces
 - Protocol checking of interfaces between designers
 - Reuse of assumptions and rules from modules verification
 - Coverage is measured for all FoCs properties
- ✍ Where FoCs is not considered
 - Abstract data checking (packet structure, data comparison)