



# Smalltalk Debug Lives in the Matrix

Loïc Lagadec, Damien Picard  
[Loic.lagadec@univ-brest.fr](mailto:Loic.lagadec@univ-brest.fr)



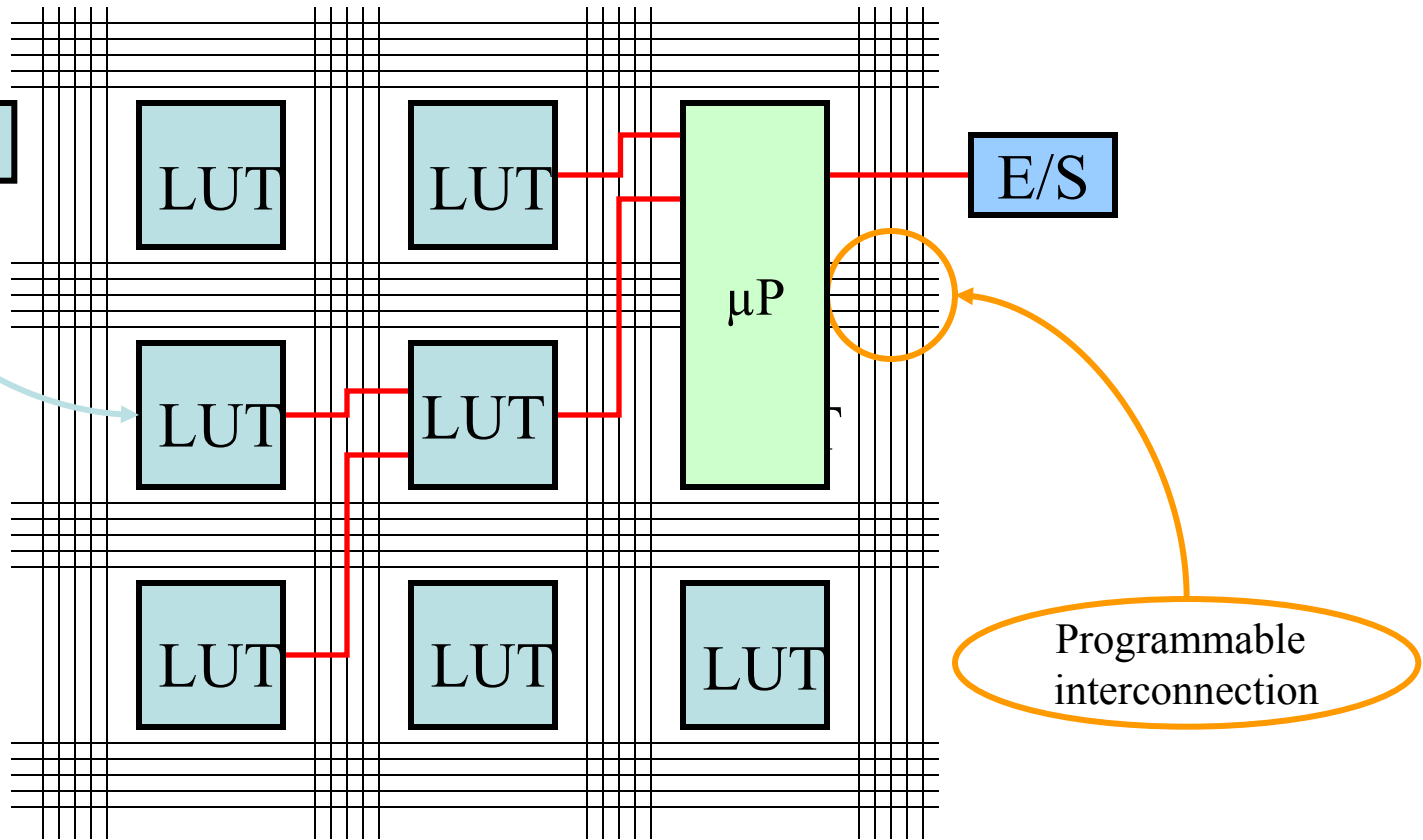
# What Matrix?



“Flexible” hardware  
Time to market



Hard to program  
Hard to debug



# What Matrix?

---



“Flexible” hardware  
Time to market

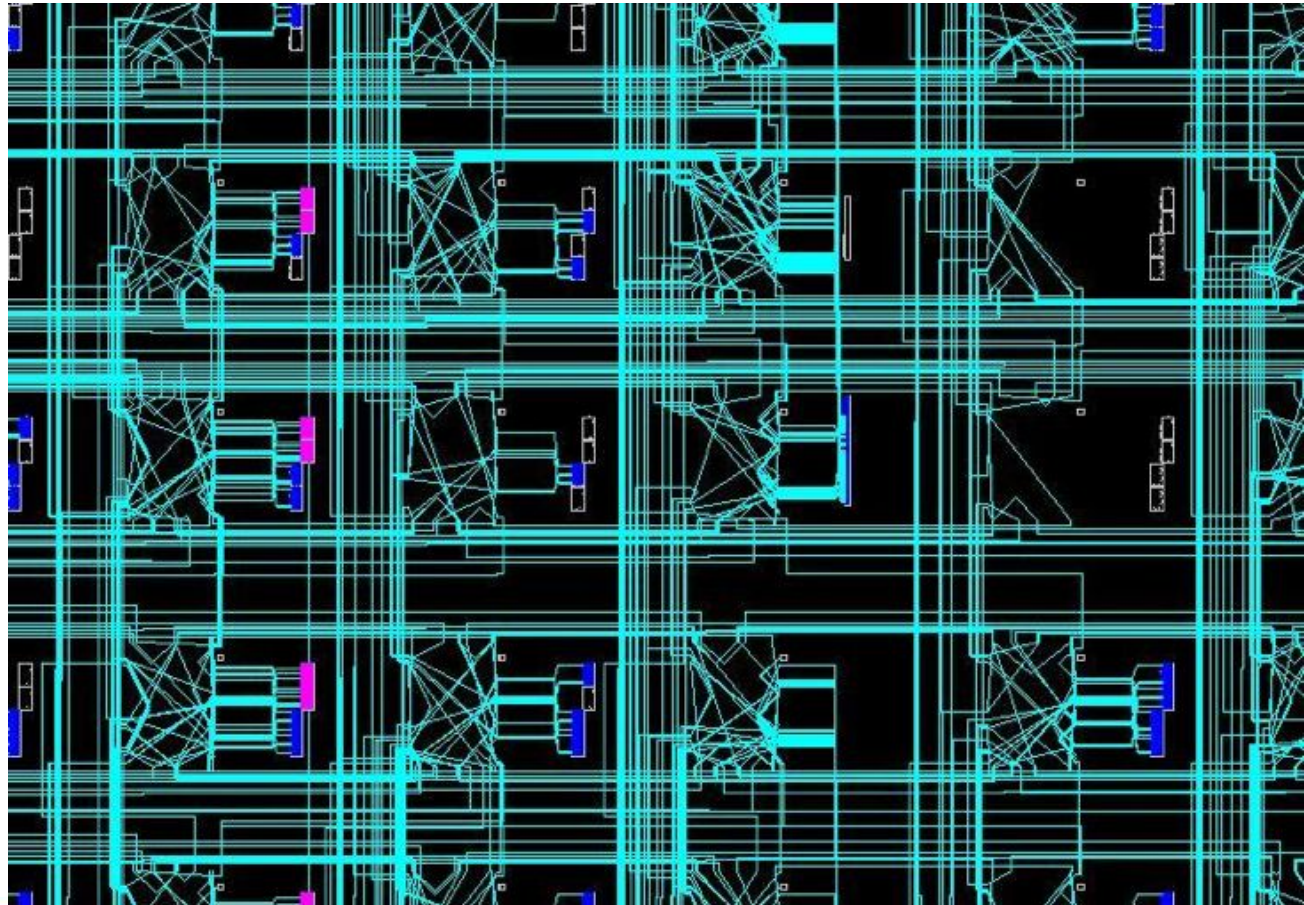


Hard to program  
Hard to debug

Specific languages  
Specific tools

Performances still  
requires manual  
tuning

EE skills required



# State of the art debugging

## Simplifying Xilinx and Altera FPGA Debug



Huge challenge

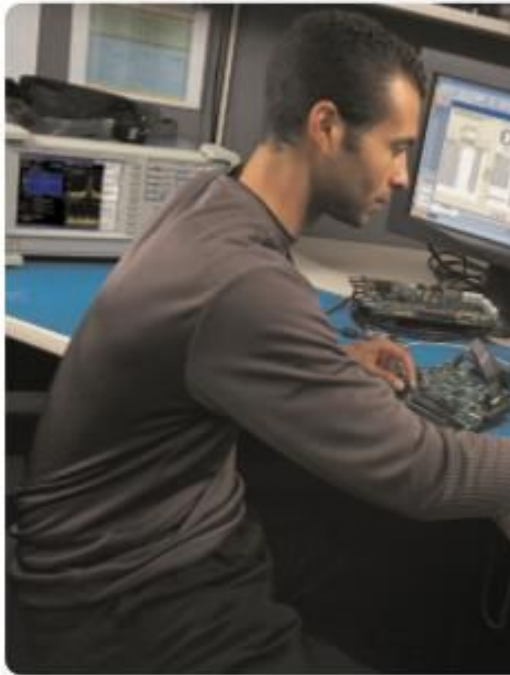
### Debug Your FPGA Design At Full Speed

Solutions such as FPGAView™ enable you to instantly move probe points within your Xilinx and Altera FPGAs without the need to recompile your design. Plus the ability to correlate internal FPGA signal activity to board-level signals can make the difference between hitting your schedule and missing your time-to-market window.

# Touching the void



Simplifying Xilinx and Altera



**Debug Your FPGA Design At Full Speed**  
Solutions such as FPGAView™ enable you to install debuggers on FPGAs without the need to recompile your design. This activity to board-level signals can make the difference in your time-to-market window.



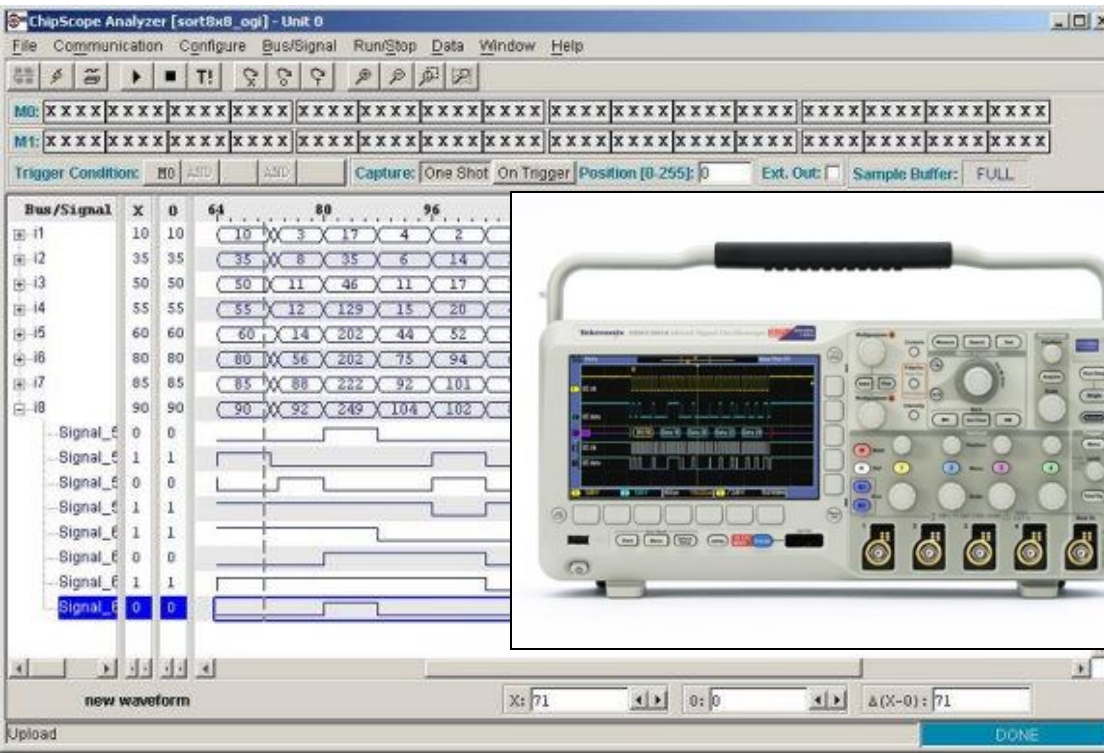
Huge challenge

Meet in the middle



# Debug silver bullet

- Observability
- Controlability
- Abstract analysis
- Fast changes

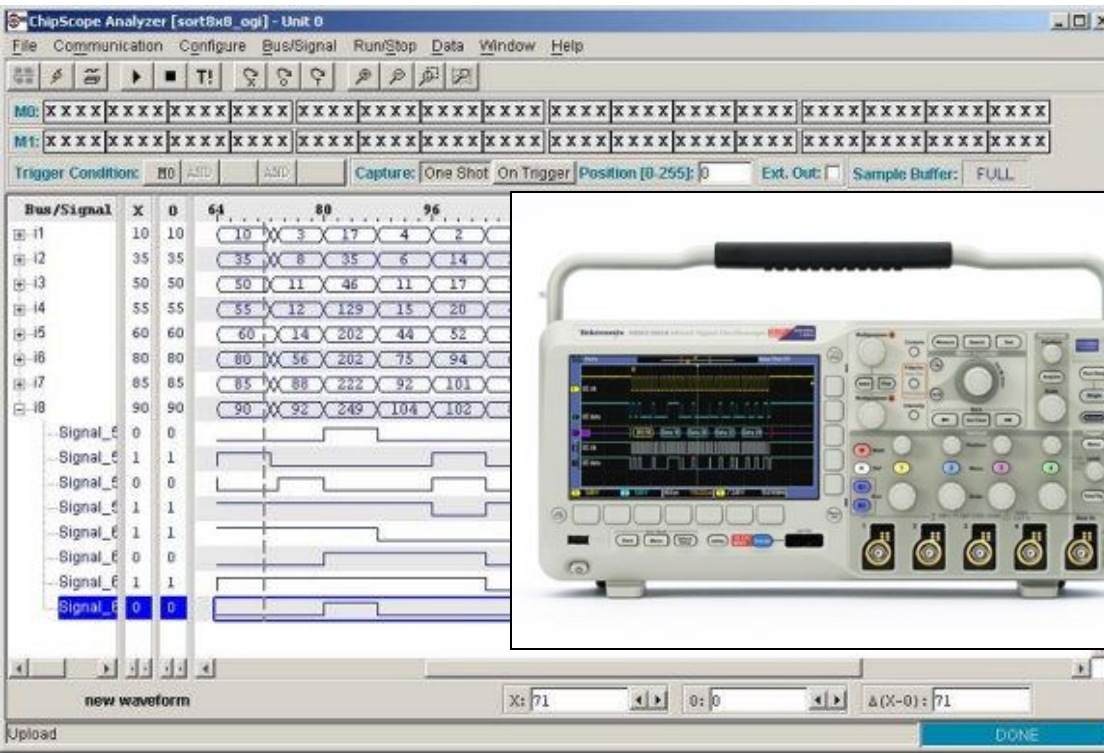
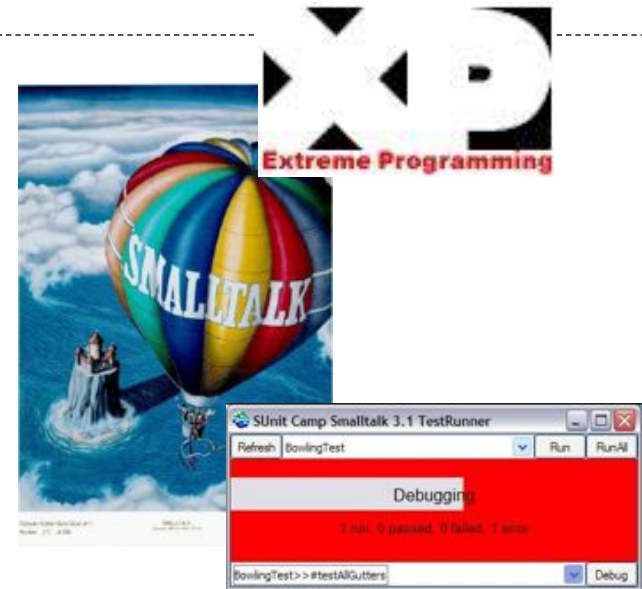


- Time to market
- Multiple runs
- Long cycles
- Simulation is not enough



# Debug silver bullet

- Observability
- Controlability
- Abstract analysis
- Fast changes



- Time to market
- Multiple runs
- Long time to debug
- Single point of failure is not

**New challenges :  
Threat or opportunity?**



# Operate at-speed

---

Keep your speed-up alive



Multiple runs prohibit any over time penalty



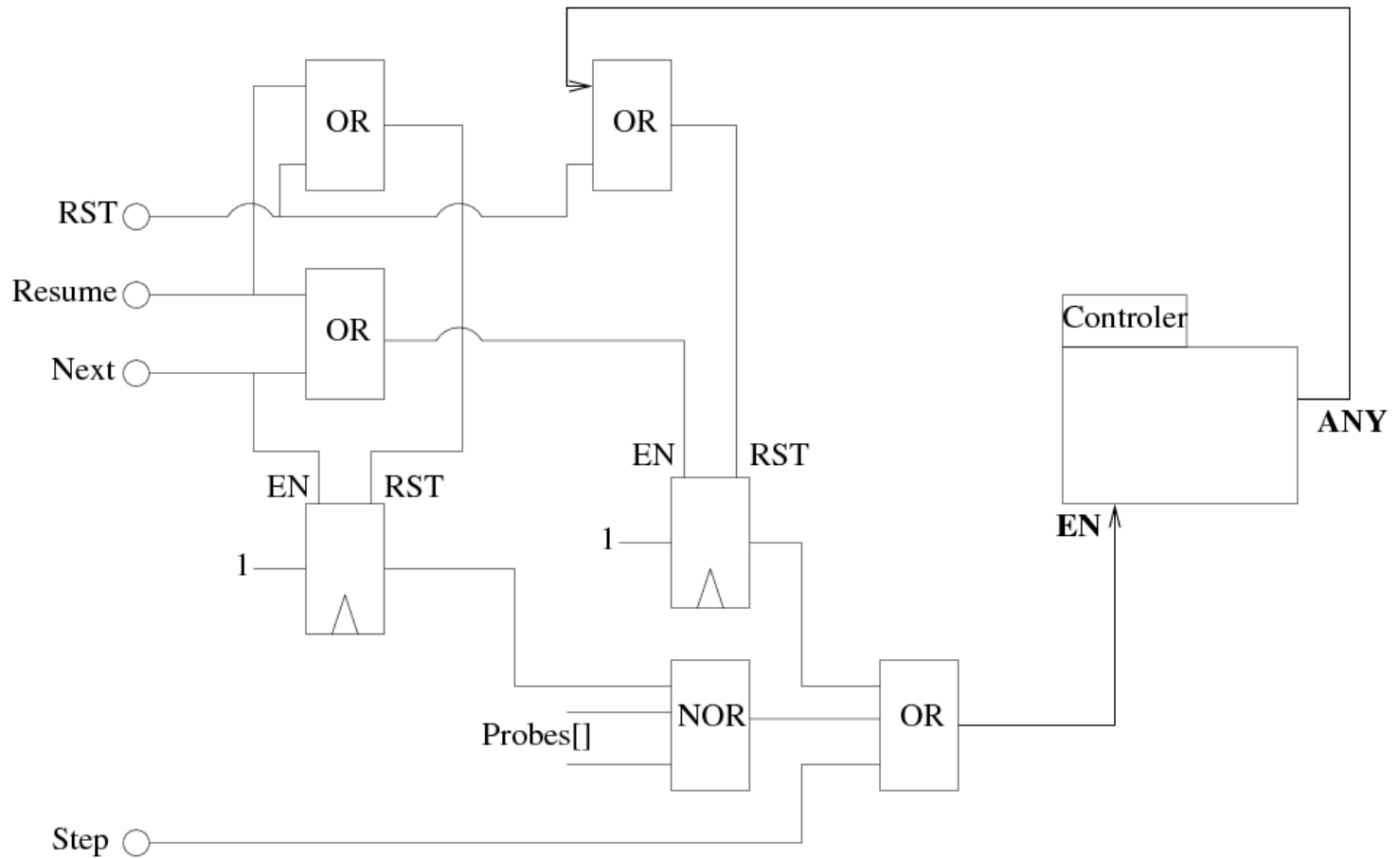
# Operate in-situ

---

Because only HW brings speed-up

Debug requires extra circuitry

# Debugging facilities are circuits



# Debugging facilities are circuits

Break in UIBiniou>>openCDFG at 32

Stack Method Edit Execute Correct

RST  
 Resume  
 Next  
 Step

```

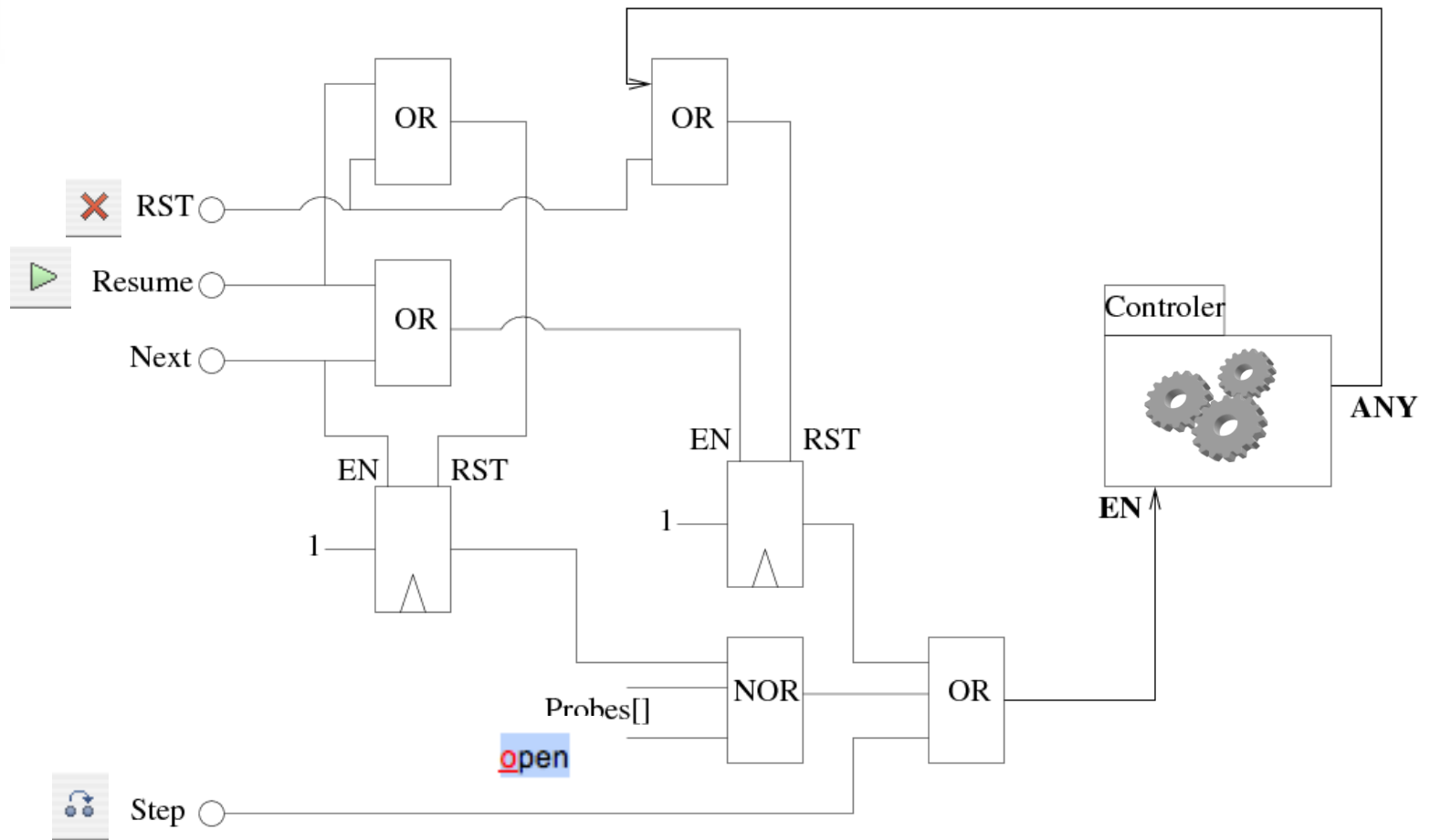
UIBiniou>>openCDFG
MenuBarButtonController>>dispatchMenuSymbol:
MenuBarButtonController>>dispatchMenuSelection:
optimized [] in MenuBarButtonController>>menuBarActionFor
BlockClosure>>ensure:
MenuBarButtonController>>menuBarActionForMenu:
MenuBarButtonController>>menuBarAction
MenuBarButtonController>>redButtonPressedEvent:
RedButtonPressedEvent>>dispatchTo:
MenuBarButtonController(Controller)>>handleEvent:
EventDispatcher>>dispatch.to:
EventDispatcher>>dispatchEvent:
  
```

self cdfgSynthesis **open**

self depends builder uiSession eventHandlers breakpointsList typeList

ANY

# Debugging facilities are circuits



# Observe and monitor

---



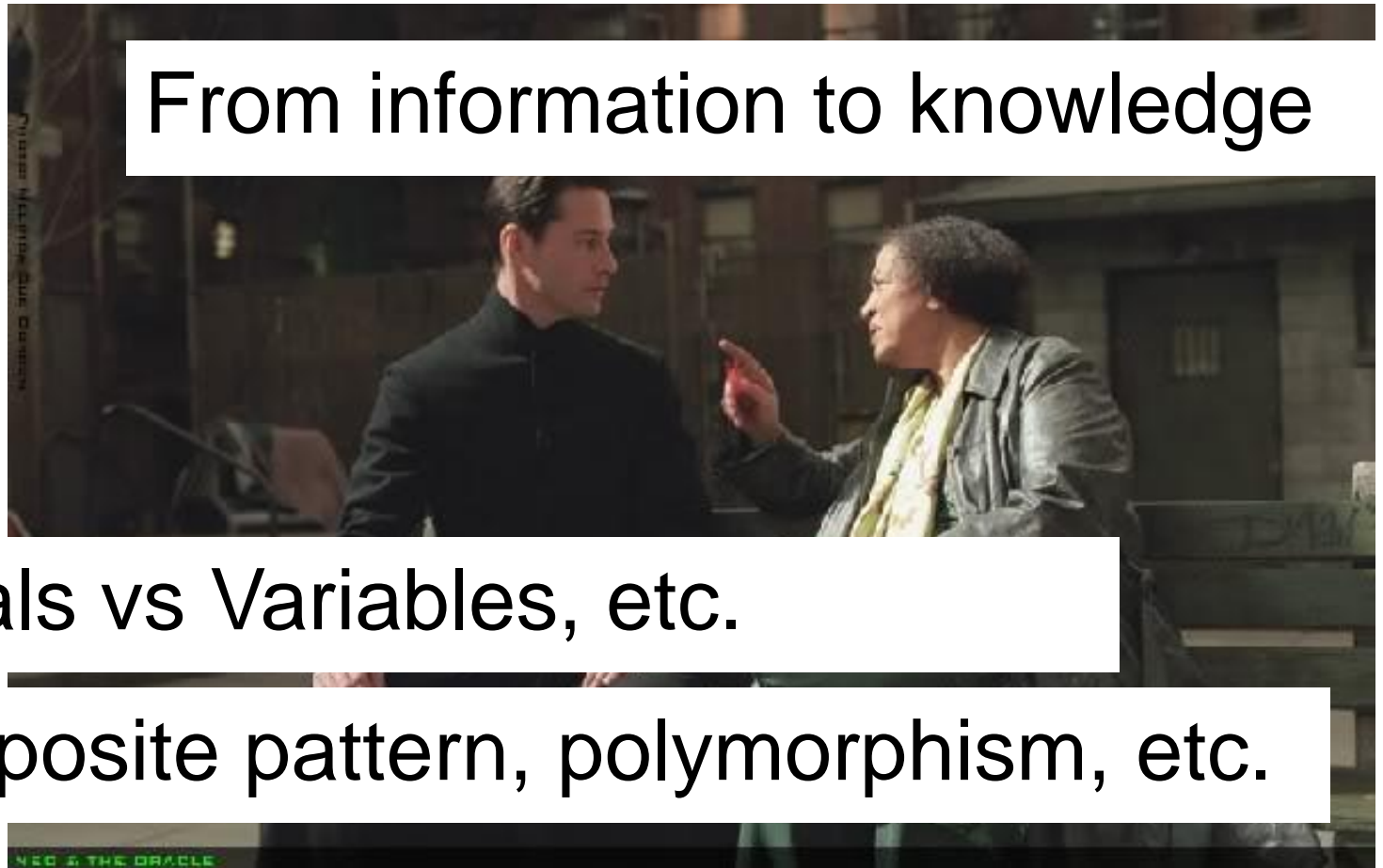
Take decision

You cannot watch everything



# Abstract analysis: semantic needed

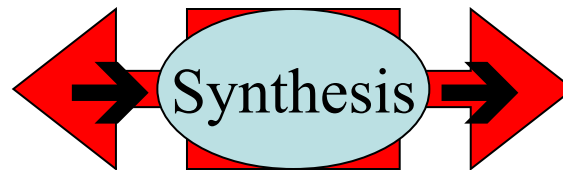
---



# From a technical point of view



Software abstraction



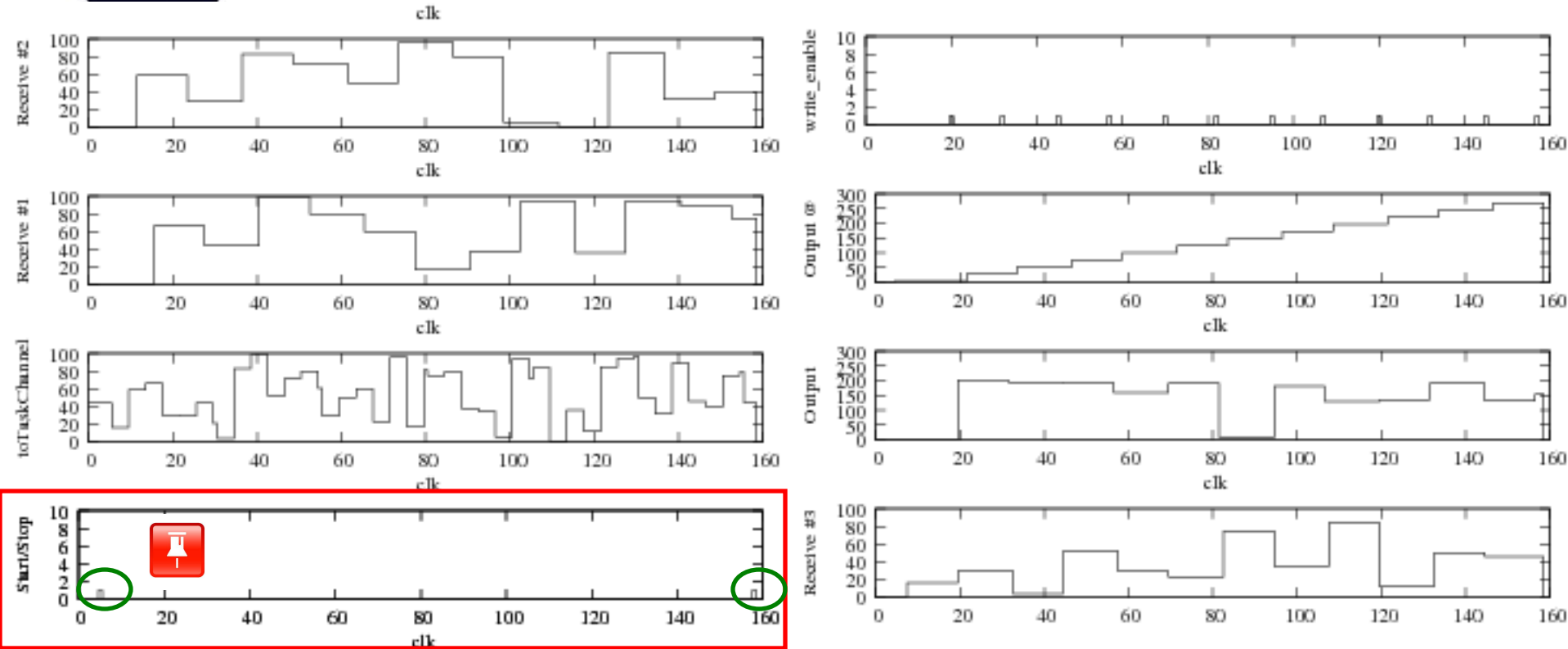
Implementation in hardware

From D. Picard's ESUG 2009 talk



# Full observability isn't scalable

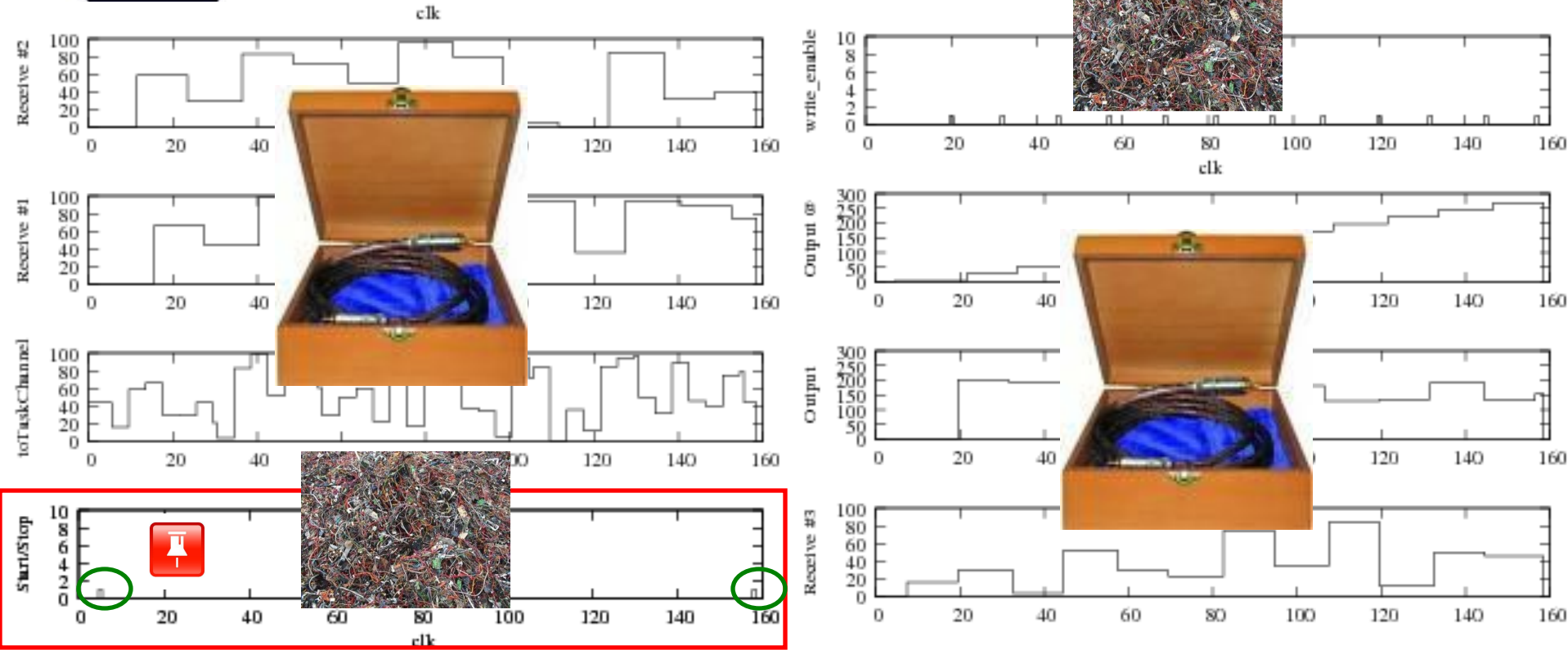
Focus on POI





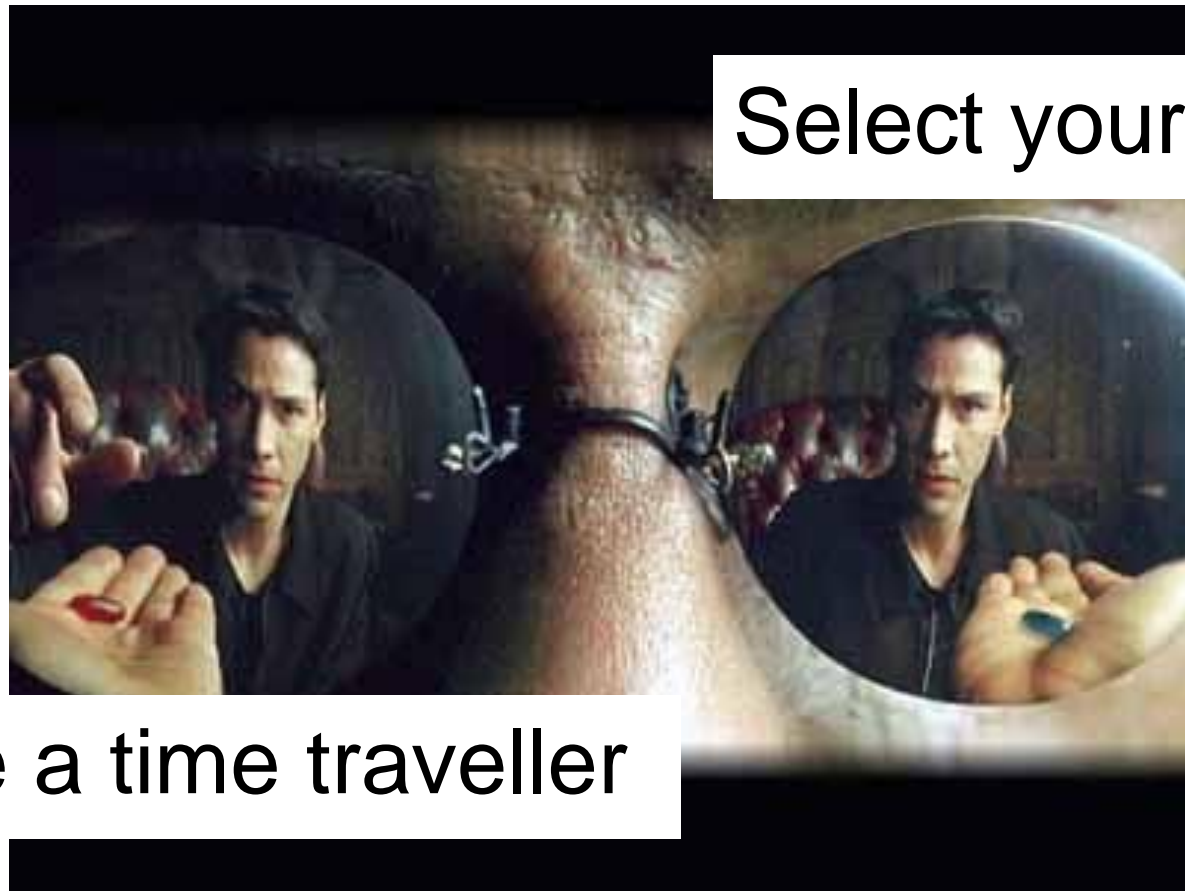
# Full observability isn't scalable

Focus on POI



# Take control

---



Select your



Become a time traveller



# Smalltalk debugger

---

- Just fit approach
- Run code and catch exception
- Code hot replacement, variable update, etc.
- Step on or resume execution
- Possible rollback

- Multiple runs
- Breakpoints update, earlier exception
- Same conceptual behavior

**Become a time traveller**

# Dodge bullets

---



Once the time has stopped

... just operate !

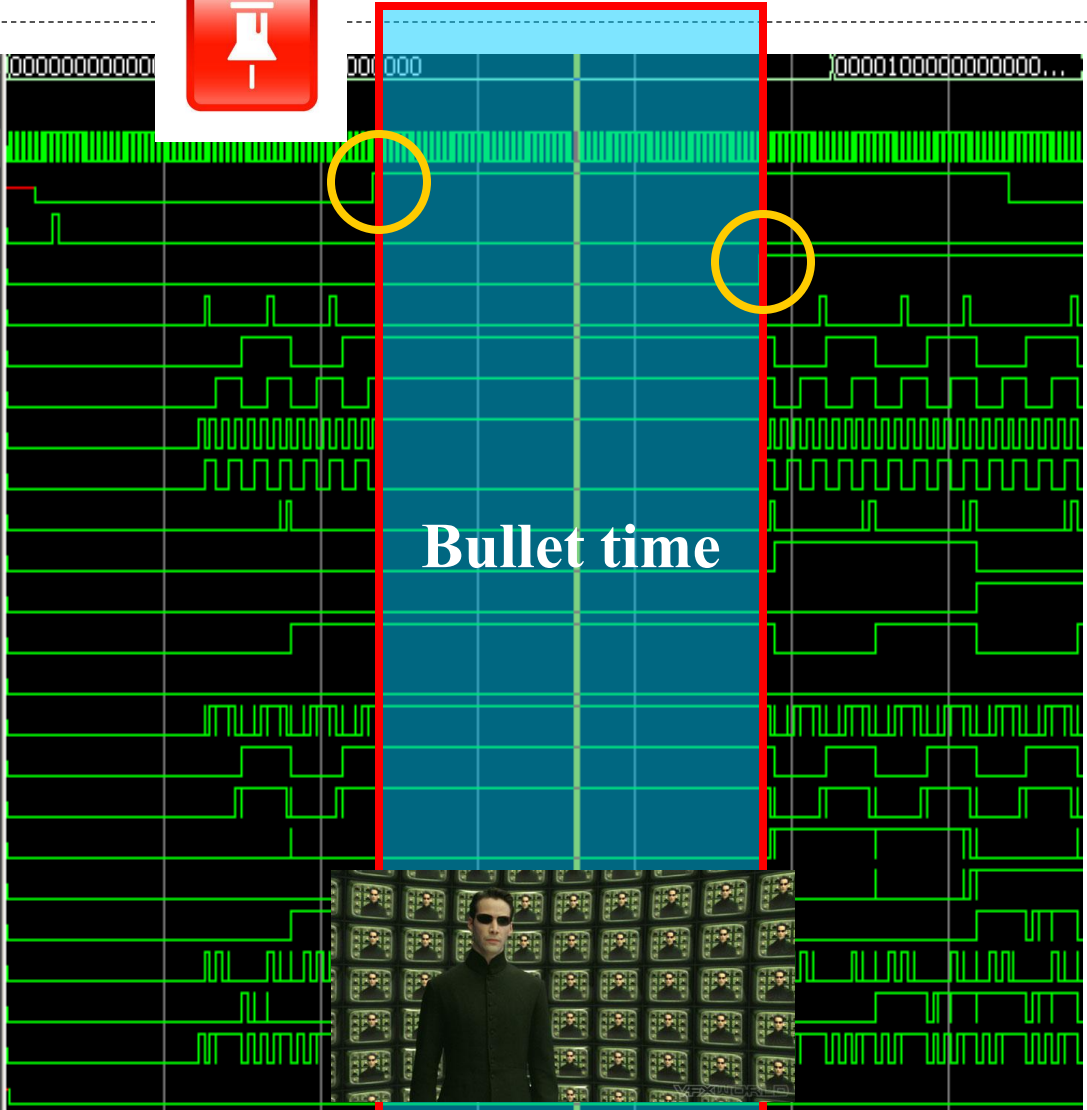


# Bullet time explained



```

+ /testbench/UUT/NUT/WDATA3          00000000000000000000
- CtrlDebug
+ /testbench/UUT/NUT/CLK              St0
+ /testbench/UUT/NUT/PROBE2           St1
+ /testbench/UUT/NUT/START_TASK       St0
+ /testbench/UUT/NUT/RESTART          St0
+ /testbench/UUT/NUT/READ0            St0
+ /testbench/UUT/NUT/\multiple_tg1_2/sig... St1
+ /testbench/UUT/NUT/\multiple_tg1_2/sig... St1
+ /testbench/UUT/NUT/\multiple_tg1_2/sig... St1
+ /testbench/UUT/NUT/\multiple_tg1_2/sig... St0
+ /testbench/UUT/NUT/\multiple_tg1_2/co... St0
+ /testbench/UUT/NUT/\multiple_tg1_2/sig... St0
+ /testbench/UUT/NUT/\multiple_tg1_2/sig... St0
+ /testbench/UUT/NUT/\multiple_tg1_2/sig... St1
+ /testbench/UUT/NUT/\multiple_tg1_2/co... St0
+ /testbench/UUT/NUT/\multiple_tg1_2/sig... St1
+ /testbench/UUT/NUT/\multiple_tg1_2/sig... St1
+ /testbench/UUT/NUT/\multiple_tg1_2/sig... St1
+ /testbench/UUT/NUT/\multiple_tg1_2/sig... St0
+ /testbench/UUT/NUT/\multiple_tg1_2/sig... St0
+ /testbench/UUT/NUT/\multiple_tg1_2/en... St1
+ /testbench/UUT/NUT/\multiple_tg1_2/en... St0
+ /testbench/UUT/NUT/\multiple_tg1_2/en... St0
+ /testbench/UUT/NUT/\multiple_tg1_2/en... St1
+ /testbench/UUT/NUT/DONE             St0
  
```

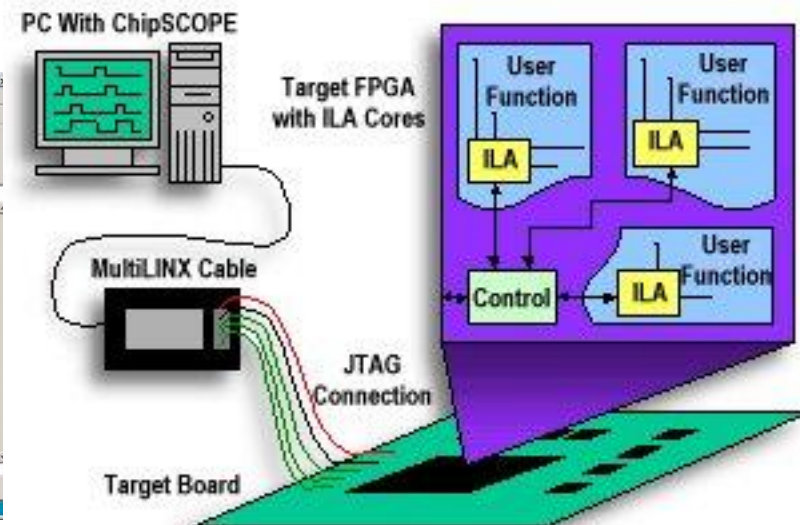
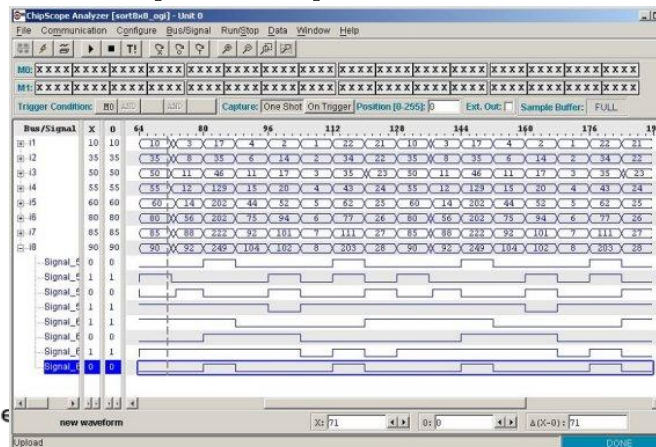


**Bullet time**



# Conditional probes

- Conditional probes offer the controlability that lacks in commercial tools
- Observability can be gained at the cost of adding some variable look-up wires
- ... But also using vendor's tool such as Chipscope



# Red Pill

Decomposition

Probes

Application

Application  
(Generated)

Probes  
inspector

Inserted probe

Control panel

Active probes

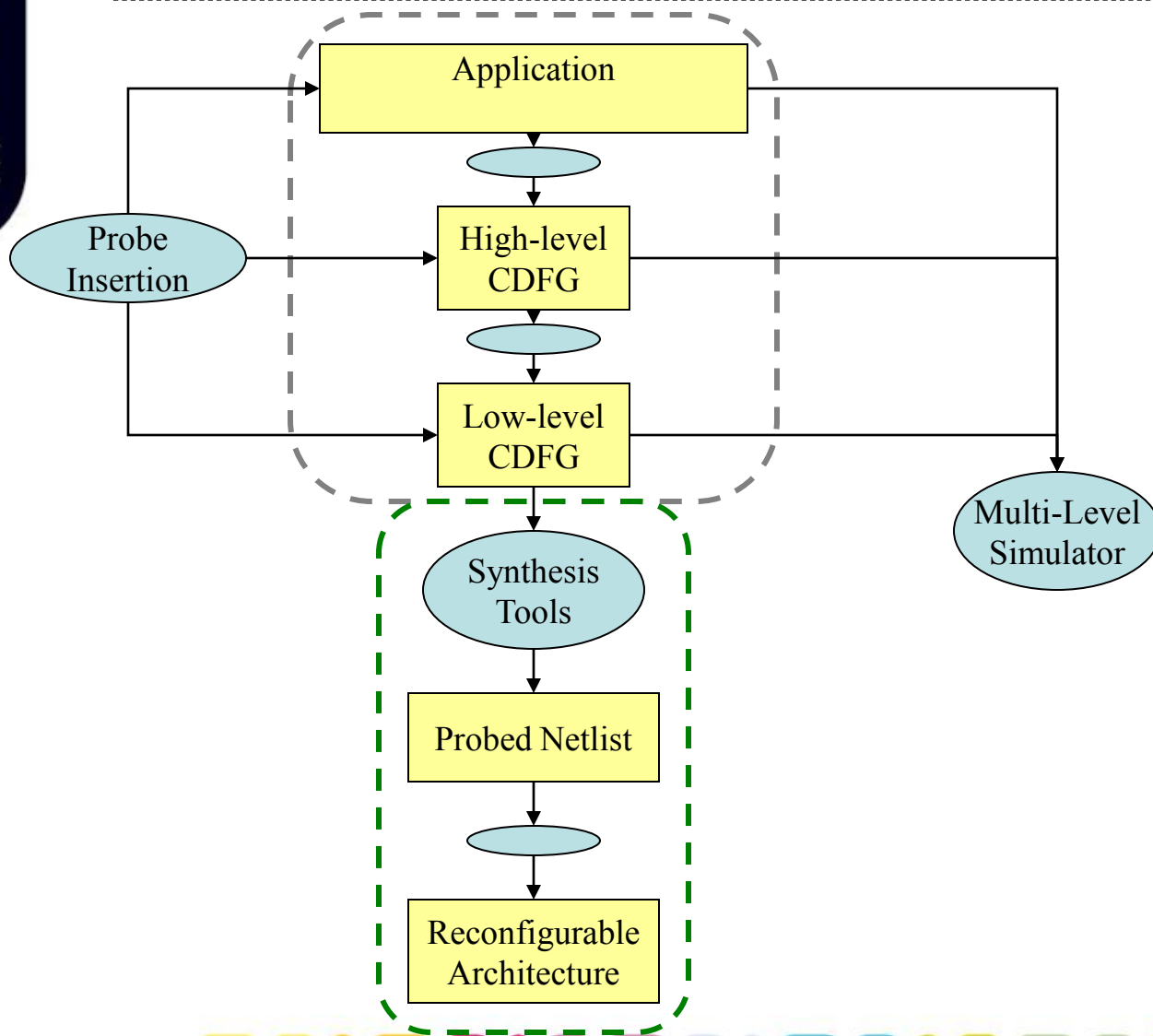
The screenshot shows the Red Pill GUI with the following components:

- Menu:** Compile, Synthesize, Simulate
- Tasks Panel:** Tasks, Connectors, Topology, UnrolledCode
- Code Editor:**

```
process fiab(chanin char cmf, chanout char fib)
{
  char t_1_6;
  char t_1_4;
  t_1_2;
  t_1_5;
  char t_1_1;

  cmf ? t_1_1;
  cmf ? t_1_2;
  cmf ? t_1_3;
}
```
- Probes Inspector:** t\_1\_1, t\_1\_2, t\_1\_3, t\_1\_4, t\_1\_5, t\_1\_6
- Control Panel:** Play, Stop, Add, Search, Refresh icons; Clock: 39
- Output:** test

# RedPill Flow extends Biniou (HLS)



Biniou



# Abstract analysis

---

Encapsulate circuit modules as smalltalk blocks

- Enables soft and hard objects to communicate
- Delivers the power of Sunit to hardware

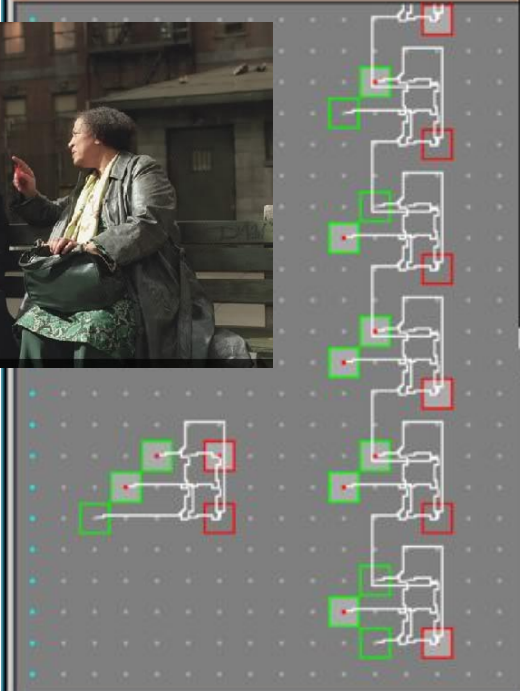
# Abstract analysis (example)



File Modules Routing Device Expe

xc6216PC84-2 selection size : cell 1@10

Converting



Converting

a Hardware Context

```

block
module
position
-- status --
status
interactive
reactive
-- inputs --
in0
in1
-- outputs --
out3

" Hardware Execution"
self value: 15 value:22*2 ["#(37)"]

"Software Execution"

| block |
block := [a :b| Array with:( a + b \
128) ].

block value: 15 value:22 "#(37)"
    
```

a Hardware Context

```

block
module
position
-- status --
status
interactive
reactive
-- inputs --
in0
in1
in2
-- outputs --
out4

" Hardware Execution"
self value: 0 value:1 value:1 "#(2)"

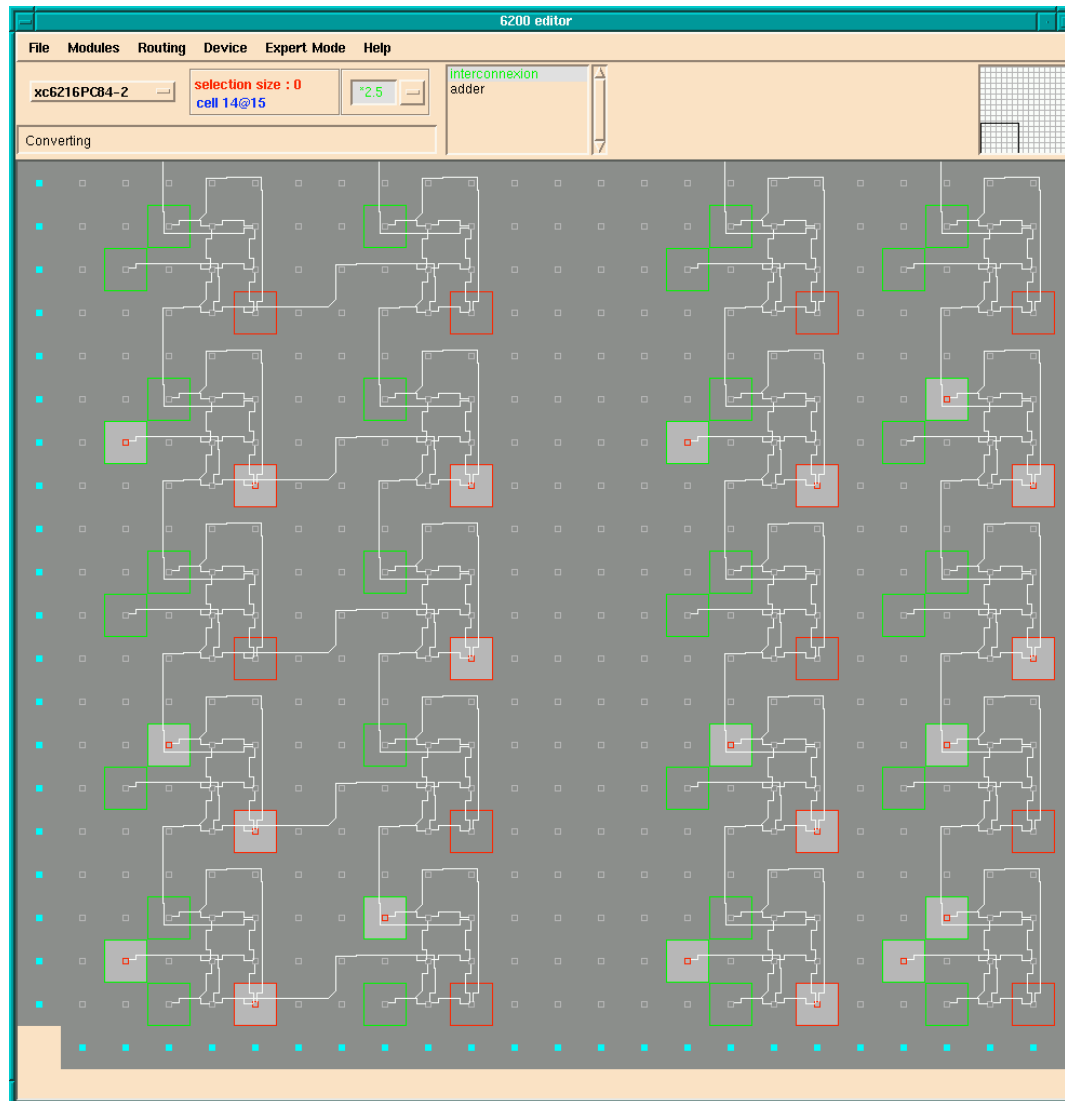
"Software Execution"

| block [
block := [x :y :z| c s]
c := x + y + z > 1 ifTrue:[1]
ifFalse:[0].
s := x + y + z \2.
^ Array with:( c*2 +s)].

block value: 0 value:1 value:1 "
#(2)"
    
```



# Characterization tests & SUnit



# How Many matrix?



The screenshot displays the Architecture Designer software interface. The main window has a menu bar with "File", "Operations", "Options", and "Help". Below the menu bar, there are several panels:

- Title:** A text field containing "LPPGACell2" and buttons for "New" and "Delete".
- Options:** A section with a "Category" dropdown menu set to "lppga".
- Contents:** A code editor showing a menu structure. The menu items are:
  - edition
  - ( )
  - Architecture
  - Command
  - New Window
  - Done
 The corresponding actions are:
  - copy
  - cut
  - paste
  - format
  - file
  - find
  - find and replace
  - again
  - undo
  - hardCopy
- Building:** A row of buttons: "compile", "internal", "tree view", "build", "produce", "UGI", "open".

Overlaid on the main window are three smaller windows:

- OPHEN Editor:** Shows a circuit diagram with various components and connections.
- LPPGArray Editor:** Shows a grid of small square components arranged in a pattern.
- OPHEN Level Editor:** Shows a grid of small square components arranged in a pattern, similar to the LPPGArray Editor but with a different layout.

# Reboot the Matrix



The collage illustrates the transition from a high-level virtual architecture to a detailed physical implementation. The 'Virtual layer' shows a grid of orange blocks, while the 'Physical Layer' shows a more complex grid of green blocks. The 'Architecture Designer' software interface shows a menu with options like 'New Window', 'find', 'find and replace again', 'undo', and 'hardCopy'. The 'Building' section includes buttons for 'compile', 'internal', 'tree view', and 'build'. The central image shows a 'MicroBlaze' and 'CGRA' highlighted in red on a black matrix background. Other screenshots show a circuit diagram with 'lppga' category and a grid of components.

# Will you take the red pill?

---

- HLS (Biniou) offers a path from HL languages to circuits
- Vendors tools offer observability
- Red Pill offers controlability
- Object encapsulation offers abstract analysis and polymorphism.

Smalltalk debug definitively lives in the Matrix

# Thank you for your attention

