A « Correct by Construction » Realistic Digital Circuit

Marc BENVENISTE
Formal Methods
Digital Secure Access Division

Recent Innovations and Applications in B'2009, November 3rd, 2009, TU Eindhoven

Digital Secure Access (DSA) Division

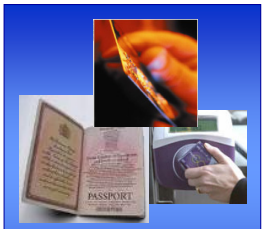
http://www.st.com/stonline/domains/applications/security/index.htm

Smartcard Circuits

Mobile Telecom
TAM 2008 ~ 3 Bu

SIM
USIM
NFC Solutions

Secure Smartcards
TAM 2008 ~ 800 Mu



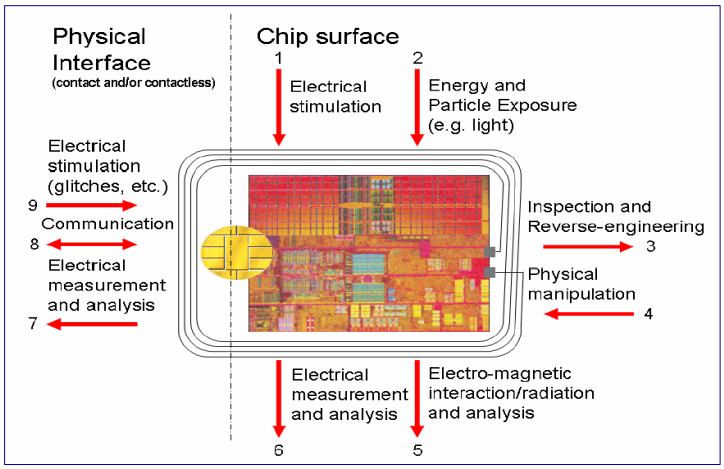
Gov & ID
Banking
Transport

Secure Conditional Access
TAM 2008 ~ 150 Mu

Pay TV
IT & Security

Mid Range Security | High Range Security

Chip Security is NOT (only) Design Correctness...



* BSI-PP-0035 version 1.0
Security IC Platform Protection Profile , by Eurosmart, 2007/06/15

Chip Security (only) BUILDS on Design Correctness

- Accurate & complete documentation
- Exhaustive verification & validation

*

- ... the very least to start working on security

* http://www.hsconsortium.co.uk/workplacesafety.html

Quoting www.scdsource.com*

/Correct-by-construction design flow/

ARM's Bryan Dickman said:

We hope to see a **correct-by-construction design flow** that may also reduce the overall verification burden

We could get there by having designers use the tools from design onwards in the flow to **systematically explore and formally prove their code**

At the moment, we're using formal **rather retrospectively**

* http://www.scdsource.com/article.php?id=341
Mixing Formal and Dynamic Verification, Part1 & Part 2, by Bill Murray, 2009/05/29
(emphasizing is mine)

Correct by Construction Chip Functionality: An advanced R&D feasibility study report

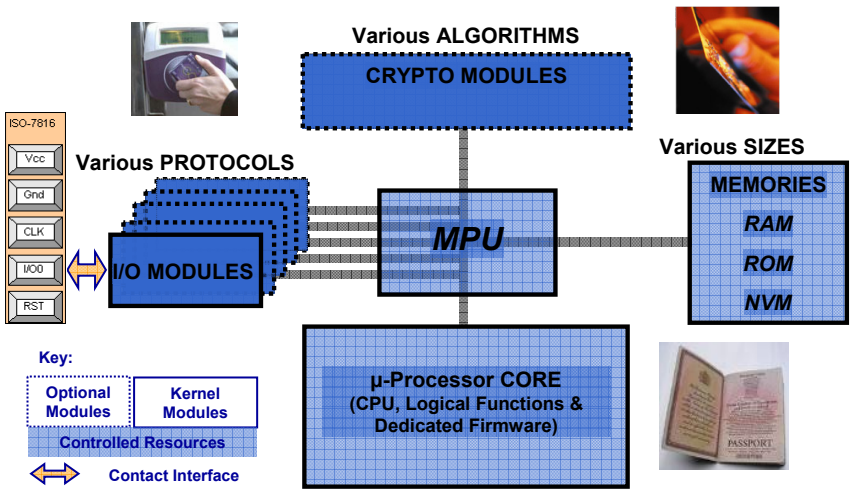
- ST23 Memory Protection Unit at a Glance
- Traditional Design Flow
- Experimental Design Flow
- Main Construction Steps
- Traditional versus Experimental
- Limitations & Ways Forward

Correct by Construction Chip Functionality: An advanced R&D feasibility study report

- **ST23 Memory Protection Unit at a Glance**
- Traditional Design Flow
- Experimental Design Flow
- Main Construction Steps
- Traditional versus Experimental
- Limitations & Ways Forward

MPU: Complete Programmable Access Control

- Memory Protection Unit: All accesses monitored

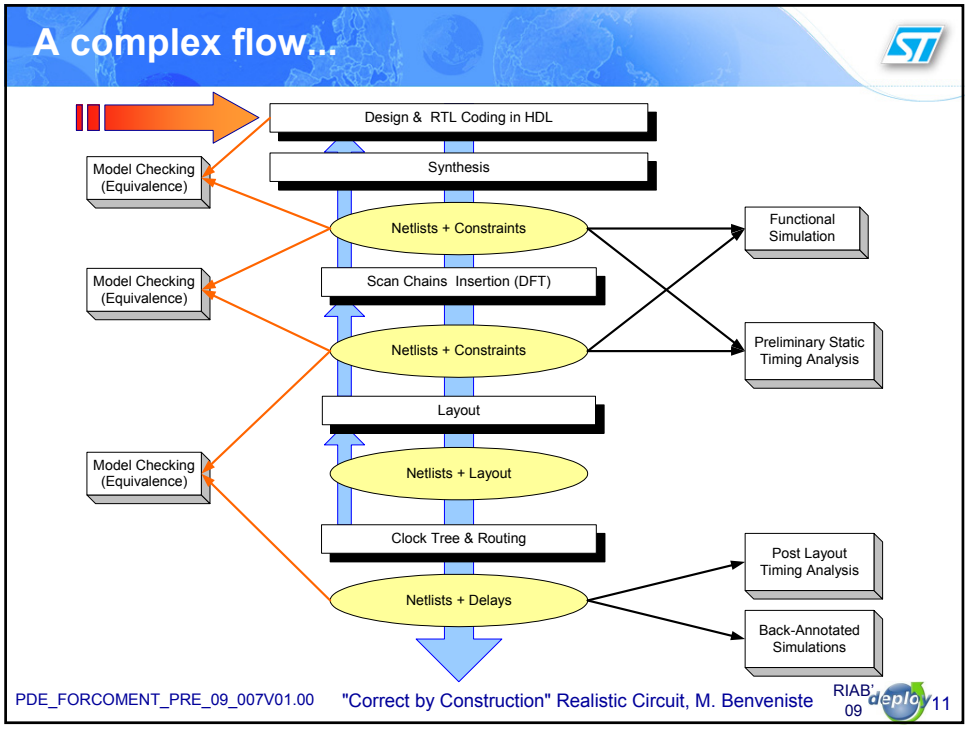


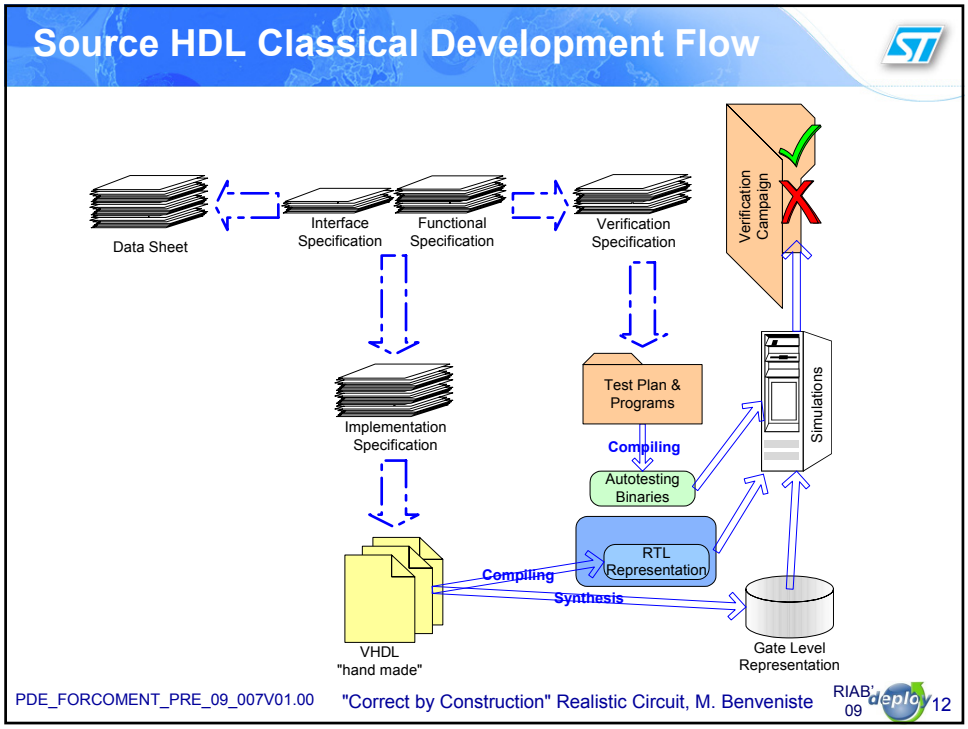
MPU - Overview

- **Segment**: addressable window in memory, **defined by**
  - **Start** / **End** addresses
  - *A clearance level*
  - Access rights (**R**ead, **W**rite, e**X**ecute)
  - Active status (mapped, not mapped)
- **Clearance level**: privilege level of a program
  - A value in [**Supervisor** .. **Public**] (actually [0 .. 3])
  - Each segment is given a clearance level
  - Read & write access to **peripheral registers** are globally **denied/granted** for each clearance level
  - Program **clearance change** control is **enabled/disabled** for each clearance level (call gates)
- Access control principles: code (e**X**ecute) & data (**R**ead/**W**rite) controls
  - A program can enforce clearance change control based on **current** and **previous** program clearance levels
  - All needed **data** for a given program must be located in its own **clearance** (or **Public** data) segments

Correct by Construction Chip Functionality: An advanced R&D feasibility study report

- ST23 Memory Protection Unit at a Glance
- **Traditional Design Flow**
- Experimental Design Flow
- Main Construction Steps
- Traditional versus Experimental
- Limitations & Ways Forward

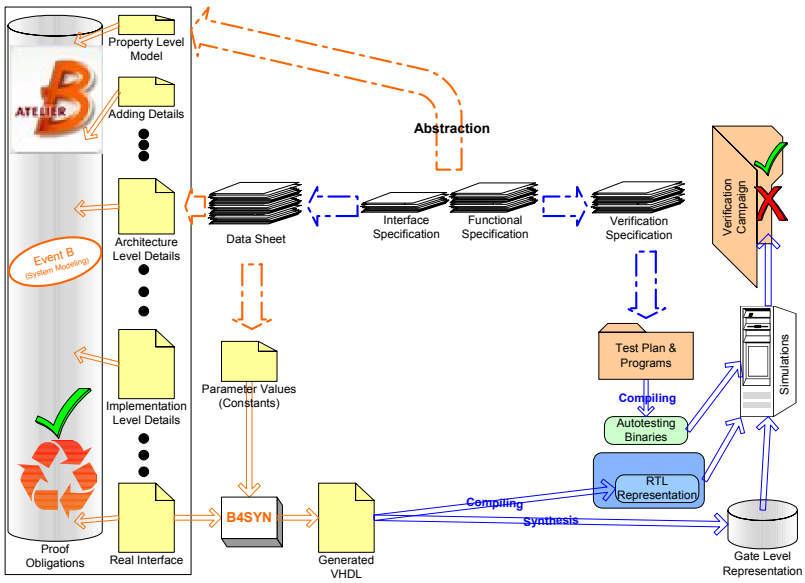

A complex flow...
Design & RTL Coding in HDL
Synthesis
Model Checking (Equivalence)
Netlists + Constraints
Functional Simulation
Model Checking (Equivalence)
Scan Chains Insertion (DFT)
Netlists + Constraints
Preliminary Static Timing Analysis
Layout
Model Checking (Equivalence)
Netlists + Layout
Clock Tree & Routing
Post Layout Timing Analysis
Netlists + Delays
Back-Annotated Simulations
PDE_FORCOMENT_PRE_09_007V01.00
"Correct by Construction" Realistic Circuit, M. Benveniste
RIAB' 09
11


Source HDL Classical Development Flow
Data Sheet
Interface Specification
Functional Specification
Verification Specification
Verification Campaign
Implementation Specification
Test Plan & Programs
Simulations
Compiling
Autotesting Binaries
RTL Representation
Compiling
Synthesis
VHDL "hand made"
Gate Level Representation
PDE_FORCOMENT_PRE_09_007V01.00
"Correct by Construction" Realistic Circuit, M. Benveniste
RIAB' 09
12

Correct by Construction Chip Functionality: An advanced R&D feasibility study report

- ST23 Memory Protection Unit at a Glance
- Traditional Design Flow
- **Experimental Design Flow**
- Main Construction Steps
- Traditional versus Experimental
- Limitations & Ways Forward

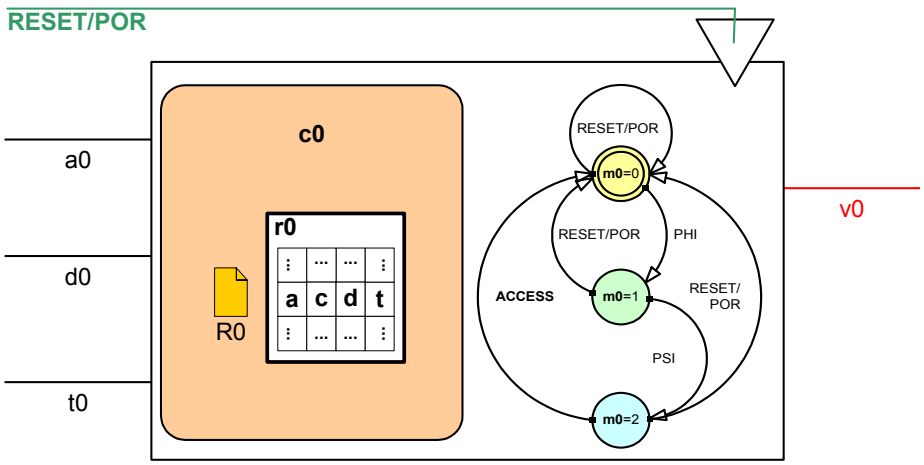
New Proved Source HDL Development Flow



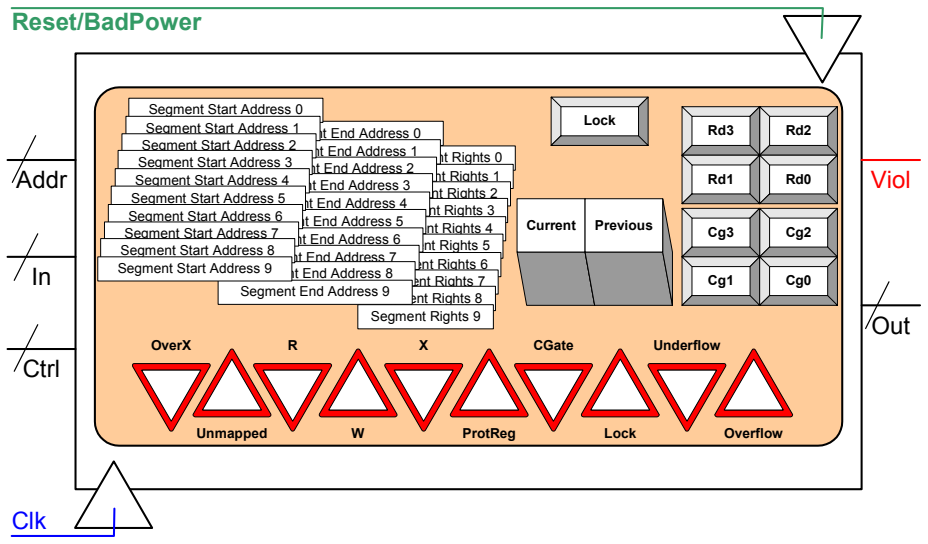
Correct by Construction Chip Functionality: An advanced R&D feasibility study report

- ST23 Memory Protection Unit at a Glance
- Traditional Design Flow
- Experimental Design Flow
- **Main Construction Steps**
- Traditional versus Experimental
- Limitations & Ways Forward

Initial (Simple) Models



Final (Complex) Models



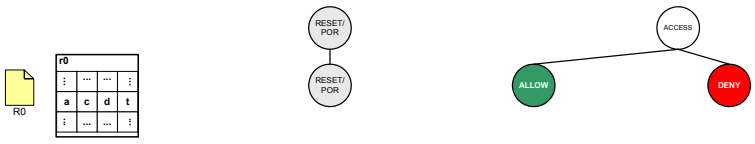
What's an Event B System ?

- An observation contract over a state space
- State space:
  - the set of all valuations of a given set of typed variables
- Observation contract:
  - if the system reaches a point where the invariant holds, then all observable transitions of the system are kept within this invariant subspace
- Invariant:
  - logical condition statement about the system variables
- Observable transition:
  - labeled guarded event fired infinitely often when open

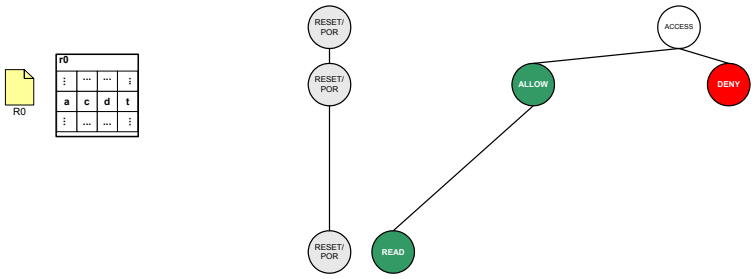
What's an Event B Refinement ?

- A change of observation space preserving bound contracts
- State Space usually expands as the represented system becomes more precisely defined
- Observable transitions can be split or joined according to description needs:
  - stretching observation or folding behaviors, for instance
- If "allow" refines "access", then:
  - "allow" occurs at most as often as "access"
    (guard strengthening)
  - "allow" preserves the expanded invariant
    (everlasting contract)
  - "allow" doesn't do something "access" would not do
    (no contradiction)

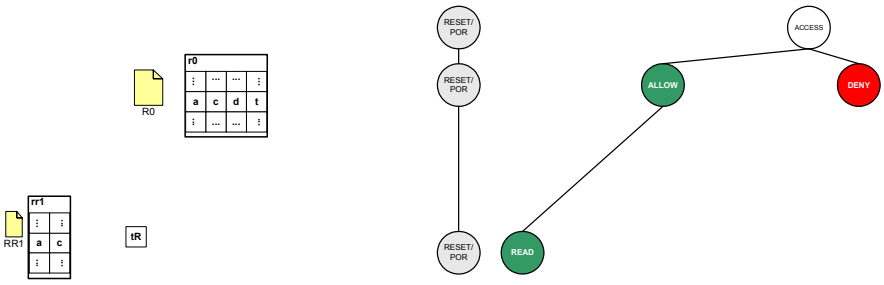
Refining State & Behavior in sync... (1/6)



Refining State & Behavior in sync... (2/6)

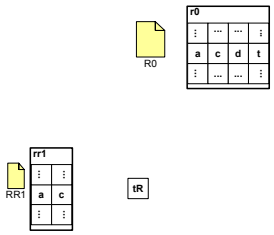


Refining State & Behavior in sync... (3/6)

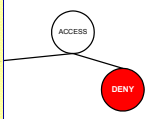


Let's take a closer look at this event...

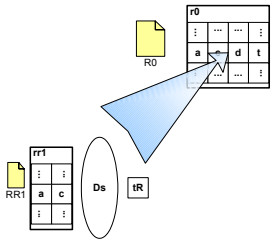
Refining State & Behavior in sync... (3/6)

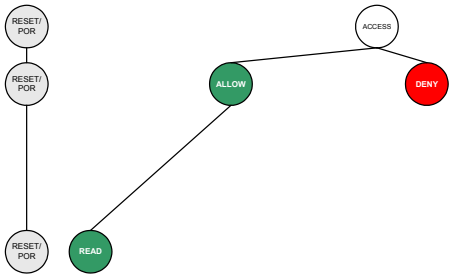



```
REFINEMENT
  mpu02
...
EVENTS
  ...
; read
    ref allow
=
  SELECT
    m0 = 2
  & t0 = TyR
  & a0 |-> c0 : rr1
  THEN
    m0 := 0
  || v0 := FALSE
  END
```

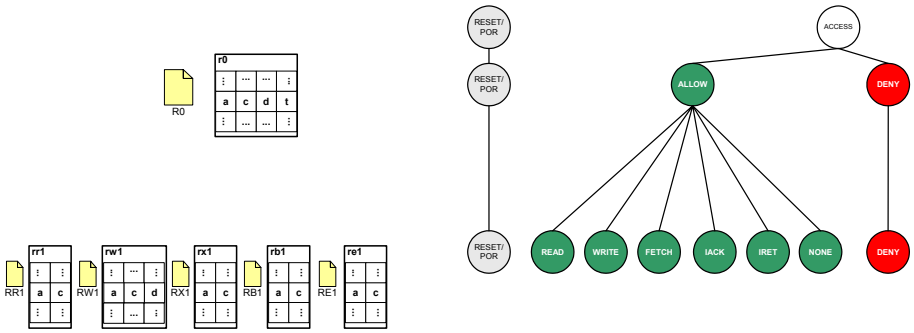


Refining State & Behavior in sync... (4/6)

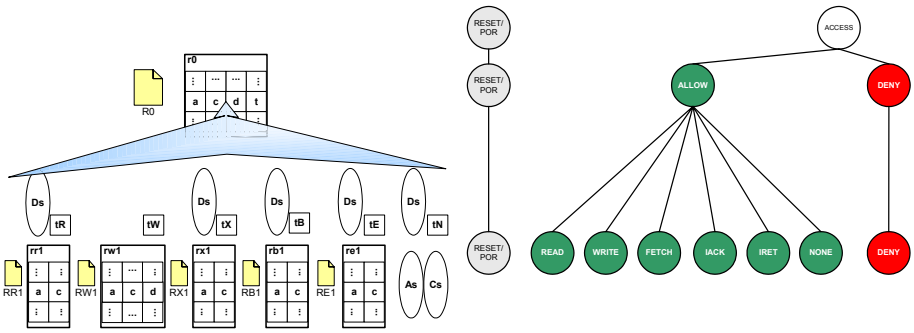




Refining State & Behavior in sync... (5/6)



Refining State & Behavior in sync... (6/6)



Let's Refine

```
SYSTEM
  mpu00
...
EVENTS
  ...
; access =
  SELECT
   m0 = 2
  THEN
   m0 := 0
  END
```

⊑

```
REFINES mpu00
...
; allow ref access =
  SELECT
   m0 = 2
 & a0 |-> c0 |-> d0 |-> t0 : r0
  THEN
   m0, v0 := 0, FALSE
 || c0 :: CLs
 || r0 ::  ADs * CLs * BYs <-> TYs
  END

      ; deny ref access =
        SELECT
         m0 = 2
       & a0 |-> c0 |-> d0 |-> t0 /: r0
        THEN
         m0, v0 := 0, TRUE
        END
```


"allow" Refines "access"

```
SYSTEM
  mpu00
...
EVENTS
  ...
; access =
  SELECT
   m0 = 2
  THEN
   m0 := 0
  END
```

⊑ (⇐)

```
REFINES mpu00
...
; allow ref access =
  SELECT
   m0 = 2
 & a0 |-> c0 |-> d0 |-> t0 : r0
  THEN
   m0, v0 := 0, FALSE
 || c0 :: CLs
 || r0 ::  ADs * CLs * BYs <-> TYs
  END

      ; deny ref access =
        SELECT
         m0 = 2
       & a0 |-> c0 |-> d0 |-> t0 /: r0
        THEN
         m0, v0 := 0, TRUE
        END
```

"deny" Refines "access"

```
SYSTEM
  mpu00
...
EVENTS
  ...
; access =
  SELECT
   m0 = 2
  THEN
   m0 := 0
  END
```

⊑

```
REFINES mpu00
...
; allow ref access =
  SELECT
   m0 = 2
 & a0 |-> c0 |-> d0 |-> t0 : r0
  THEN
   m0, v0 := 0, FALSE
 || c0 :: CLs
 || r0 ::  ADs * CLs * BYs <-> TYs
  END

      ; deny ref access =
        SELECT
         m0 = 2
        & a0 |-> c0 |-> d0 |-> t0 /: r0
        THEN
         m0, v0 := 0, TRUE
        END
```


Refinement is not enough...

```
SYSTEM
  mpu00
...
EVENTS
  ...
; access =
  SELECT
   m0 = 2
  THEN
   m0 := 0
  END
```

⊑

```
REFINES mpu00
...
; allow ref access =
  SELECT
   m0 = 2
 & a0 |-> c0 |-> d0 |-> t0 : r0
  THEN
   m0, v0 := 0, FALSE
 || c0 :: CLs
 || r0 ::  ADs * CLs * BYs <-> TYs
  END

      ; deny ref access =
        SELECT
         m0 = 2
        & a0 |-> c0 |-> d0 |-> t0 /: r0
        THEN
         m0, v0 := 0, TRUE
        END
```

```
  m0 = 2 &
  not( a0 |-> c0 |-> d0 |-> t0: r0 ) &
  m0 = 2 &
  a0 |-> c0 |-> d0 |-> t0: r0 &
"`Check refinement exclusivity - companion model'"
=>
  bfalse
```

...exclusive refinement is still not enough...

```
SYSTEM
  mpu
...
EVENTS
  ...
; access =
  SELECT
   m0 = 2
  THEN
   m0 := 0
  END
```

⊑ ⇐ ⇒

```
REFINES mpu
...
; allow ref access =
  SELECT
   m0 = 2
 & a0 |-> c0 |-> d0 |-> t0 : r0
  THEN
   m0, v0 := 0, FALSE
 || c0 :: CLs
 || r0 ::  ADs * CLs * BYs <-> TYs
  END

      ; deny ref access =
        SELECT
         m0 = 2
        & a0 |-> c0 |-> d0 |-> t0 /: r0
        THEN
         m0, v0 := 0, TRUE
        END
```

```
  m0 = 2 &
  not( m0 = 2 & not( a0|-> c0|-> d0|-> t0: r0 ) ) &
"`Check refinement liveness - companion model'"
=>
  a0 |-> c0 |-> d0 |-> t0: r0
```


...now we can be confident it's all there...

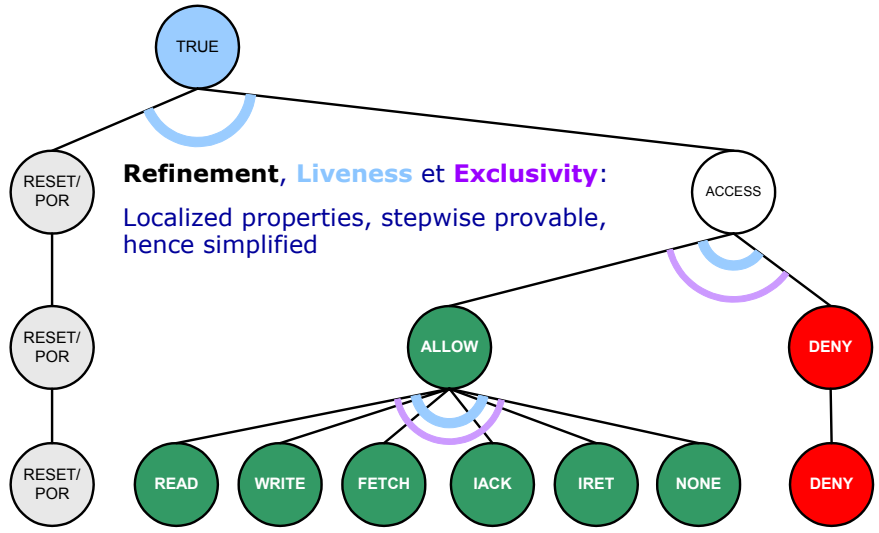
```
SYSTEM
  mpu00
...
EVENTS
  ...
; access =
  SELECT
   m0 = 2
  THEN
   m0 := 0
  END
```

⊑ ⇔

```
REFINES mpu00
...
; allow ref access =
  SELECT
   m0 = 2
 & a0 |-> c0 |-> d0 |-> t0 : r0
  THEN
   m0, v0 := 0, FALSE
 || c0 :: CLs
 || r0 ::  ADs * CLs * BYs <-> TYs
  END

      ; deny ref access =
        SELECT
         m0 = 2
        & a0 |-> c0 |-> d0 |-> t0 /: r0
        THEN
         m0, v0 := 0, TRUE
        END
```

Refine by “squeezing” events



The Refinement Plan (1/3)

Policy
Specification
Implementation

N°	Step	Main Purpose
1	Host_1	Host execution model
2	Complete Policy	Abstract complete access policy
3	Split Access	Rights matrix split for each access type Allow cases split by access type
4	Exceptions	By-passing checks automata Interrupts & reset management
5	Split Deny	Deny cases split by access type Clearances stack introduced (interrupts)
6	Registers_1	Registers access policy (mem. vs. reg.) Supervisor clearance introduced

The Refinement Plan (2/3)

Policy
Specification
Implementation

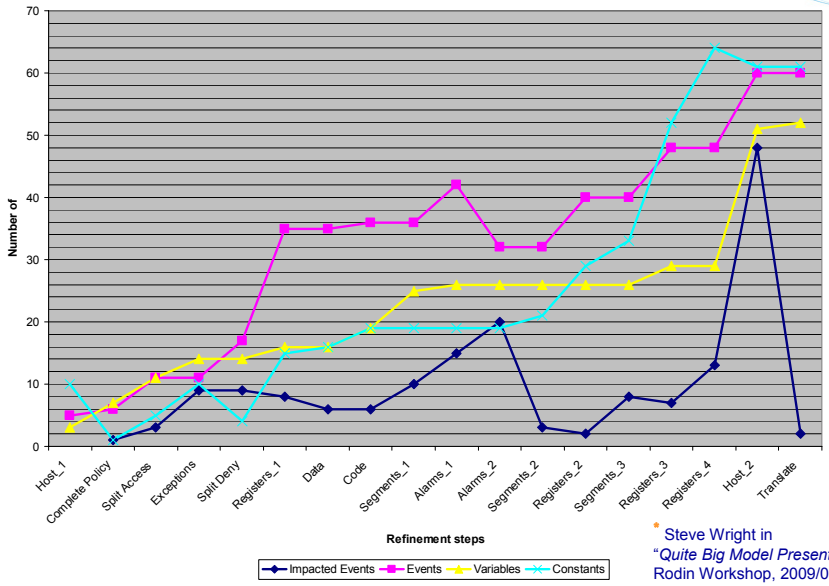
N°	Step	Main Purpose
7	Data	Data access policy Public clearance introduced
8	Code	Code access policy, Call gates introduced
9	Segments_1	Segments as sets of addresses Constructive set expressions defined
10	Alarms_1	Identification of denied accesses Alarm bits introduced
11	Alarms_2	Denied access cases grouped by alarm
12	Segments_2	Segments as address ranges (start & end) Mapped segment bit introduced
13	Registers_2	MPU register access policy Their symbolic addresses introduced

The Refinement Plan (3/3)

Policy
Specification
Implementation

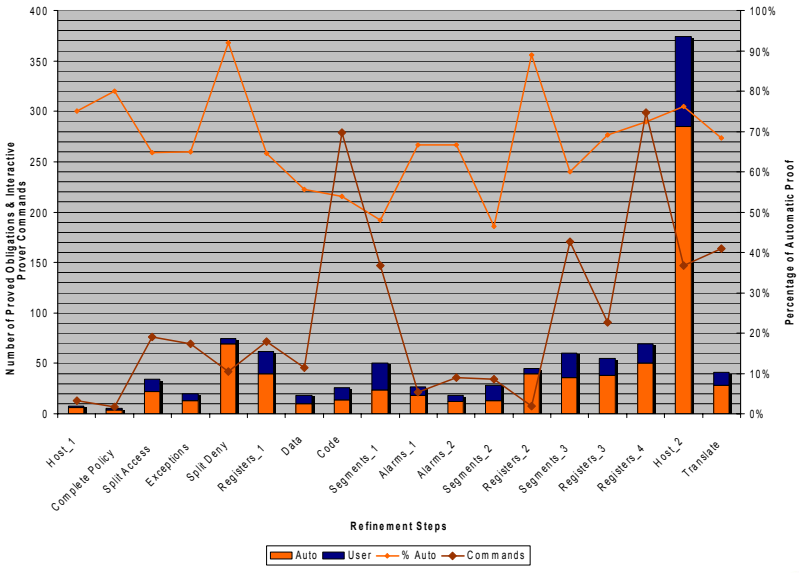
N°	Step	Main Purpose
14	Segments_3	Segments granularity Implementation optimization
15	Registers_3	MPU registers effective read, Output defined Constant read access functions Symbolic addresses are bytes now
16	Registers_4	Effective write of MPU registers Constant write access functions
17	Host_2	Physical interface for meaning (access type) All input cases covered, sampling on “tick”
18	Translate	Expression simplifications for VHDL Prop. not translated, types drive translator

Event & Variables Build up Graph

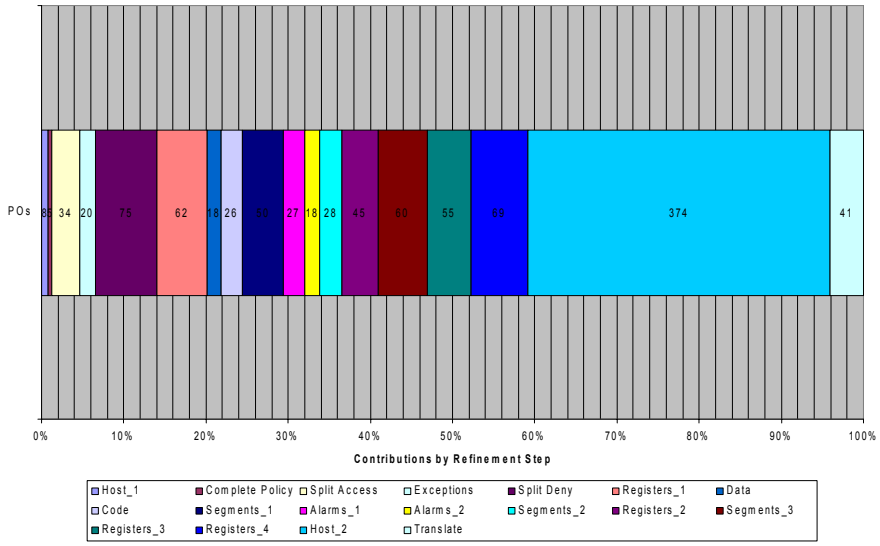


Steve Wright in *"Quite Big Model Presentation"* Rodin Workshop, 2009/07/17

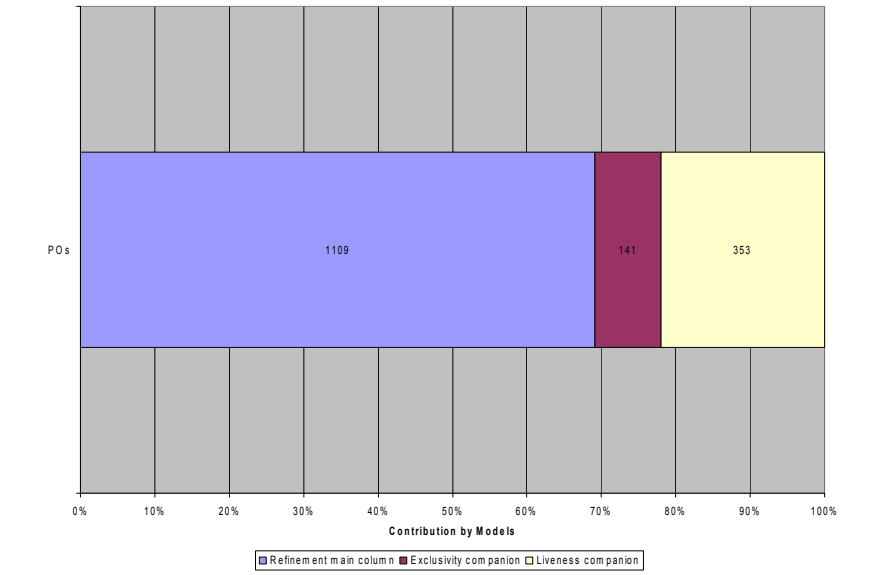
Proof Obligations Build up Graph



Proof Obligation Breakdown Graph (1/2)



Proof Obligation Breakdown Graph (2/2)



Observations on resulting implementation

- Built static structure is quite complex
  ~60 constants, ~50 variables
  Just imagine *if* verification *had* to start at this (RTL) level !
  What would you verify ?
- A lot of individual cases
  56 events
- Although very simple cases
  ~12 lines each
- **Built** with **proved exhaustive** assurance of:
  - Correctness: no contradiction regarding all specified behaviors
  - Determinism: no overlapping specified behaviors
  - Completeness preservation: no development holes

Correct by Construction Chip Functionality: An advanced R&D feasibility study report

- ST23 Memory Protection Unit at a Glance
- Traditional Design Flow
- Experimental Design Flow
- Main Construction Steps
- **Traditional versus Experimental**
- Limitations & Ways Forward

Trying a rough comparison

Criteria	Classical Flow	Experimental Flow
Effort *person.days*	~145	~145 (~200 translator)
Volume *commented source lines*	~3 600 VHDL	~7 500 B ~3 500 VHDL
Proof *number of items*	-	~1 600 obligations ~ 600 with ~4000 cmds
Structure *module number*	~15 tightly linked	4 loosely linked (only 1 generated module)
Simulation *RTL & gate level*	~30 patterns Ok	~30 patterns Ok (better debug signals)
Size *equivalent nand gates*	~5 Kgates	

Correct by Construction Chip Functionality: An advanced R&D feasibility study report

- ST23 Memory Protection Unit at a Glance
- Traditional Design Flow
- Experimental Design Flow
- Main Construction Steps
- Traditional versus Experimental
- **Limitations & Ways Forward**

Benefits, Drawbacks & Future work

- Benefits
  - Zero bug macros become feasible
  - Highly parameterized coding is encouraged
  - Focus can safely turn onto security concerns
- Drawbacks
  - No composition mechanism to assemble units
  - Not yet ready for industry (both method & tool)
- Future work
  - Extend to other case studies
  - Propose language support enhancements

Credits & Acknowledgement

The FORmal CO-developMENT Project

B Models : L. Mussat, Clearsy
B4SYN Translator: T. Lecomte & A. Requet, Clearsy
Screenplay, Direction & Realization : M. Benveniste, STMicroelectronics
Support Design : D. Chomaud, STMicroelectronics
Administration : R. Petri, STMicroelectronics, J.P. Pitzalis, Clearsy
Production : 4th et 5th Conventions PACA Regional Council - STMicroelectronics - Clearsy

Thank You !

Any Question?