This work is partially funded by the German Federal Ministry of Education, Science, Research and Technology (BMBF) in the framework of the Verisoft project under grant 01 IS C38. The responsibility for this article lies with the author(s)

Prototyping a Formal Verification Platform for SoCs

Presented by: Primrose MBANEFO

PhD Student, CPR ST

Infineon Technologies AG

Munich, Germany

Supervisors:

Wolfgang RAAB

CPR ST

Infineon Technologies AG, Munich, Germany

Pierre WODEY

ISIMA/LIMOS laboratory

Université Blaise-Pascal, Clermont-Ferrand, France

New Age – New Ways

- Market tendencies
  - –Multiple functionality devices
- Industry's reply
  - –Entire systems on single chips
- Developer's reply
  - –System Level Design
- Alarm
  - –Costly end-product errors
  - –Mid-product errors also expensive
- Tester's reply
  - –Early error detection: Cannot wait till RTL to test
  - –Simulation: Cost and coverage problems due to system size
  - –Formal verification?

Outline

- Different Forms of Formal Verification

- A Proposition
  - –Theorem Proving
  - –The System Model
  - –Semantics
  - –Translation

- Simple Test Model

Different Forms of Formal Verification

- Formal Verification: Not just a buzz-word

  Mathematical Satisfaction of Specification by Implementation

- Deductive Methods: Theorem Proving
  - –Automated or Interactive
- State-Based Methods
  - –Automata theory (Equivalence checking, Model checking)
  - –Language theory
  - –Symbolic Simulation
  - –Trace conformation
- Algebraic Systems

Outline

- Different Forms of Formal Verification

- A Proposition
  - –Theorem Proving
  - –The System Model
  - –Semantics
  - –Translation

- Simple Test Model

A Proposition

- Baseline: Theorem proving
  - –Automation where possible
- Why?
  - –System building and fine-tuning: painstaking tasks
    - Not push-button completed
  - –Fine planning of verification: respect SoC as unique
  - –Avoids state space explosion problems

Theorem Proving

- Specification and Implementation: Formulae
  - –Implementation ⇔ Specification?
  - –Implementation → Specification?
    - E.G: up & !down ⇔ Counting upwards?
- Automated Theorem Proving
  - –Missing expressivity
- Interactive Theorem Proving
  - –Quite expressive
  - –Human interaction – real brain and master of the tool
  - –Can produce material for automated theorem proving assistants and other automated tools

The System Model

- Specification as formula: No problem
- Implementation as formula: How?
- Translate SystemC to a computer logic... but
- SystemC itself is not where the proofs are
- The SoC is more interesting
  - Extract SoC from SystemC implementation

The System Model

- Step 1. Identify the SoC.

$$System = Computation \cup Communication$$

$$Computation = \{PE\}$$

$$PE = \{Processes\} \cup \{Ports\}$$

$$Process = Behaviour \cup \{Triggers\}$$

$$Communication = \{Channelled\ messages\}$$

$$Channelled\ message = \{port \cup Channel\}$$

Gerstlauer and Gajski's model modified

Semantics

- Step 2. Extract the SoC from the Implementation
  - –Involves: Mapping of implementation to SoC model
- Implementation format: SystemC files
  - –Implies: Understanding of SystemC semantics
- 3 popular forms of semantics descriptions
  - –Operational semantics
  - –Denotational semantics
  - –Axiomatic semantics
- Form chosen: Denotational semantics
  - –More intersted in the “what happens” than in the “how it happens”

Translation

- Translation: Maps SystemC constructs to System model

E.G. sc_module(upDown)

{

sc_in<bool> clk;

SC_CTOR(upDown){

  SC_METHOD(entry); ...}

}

➔ constant upDown ofType PE

➔ upDown is: $\{entry\} \cup \{clk\}$

Outline

- Different Forms of Formal Verification

- A Proposition
  - –Theorem Proving
  - –The System Model
  - –Semantics
  - –Translation
  - –Tools
    - Isabelle/HOL
    - ANTLR

- Simple Test Model

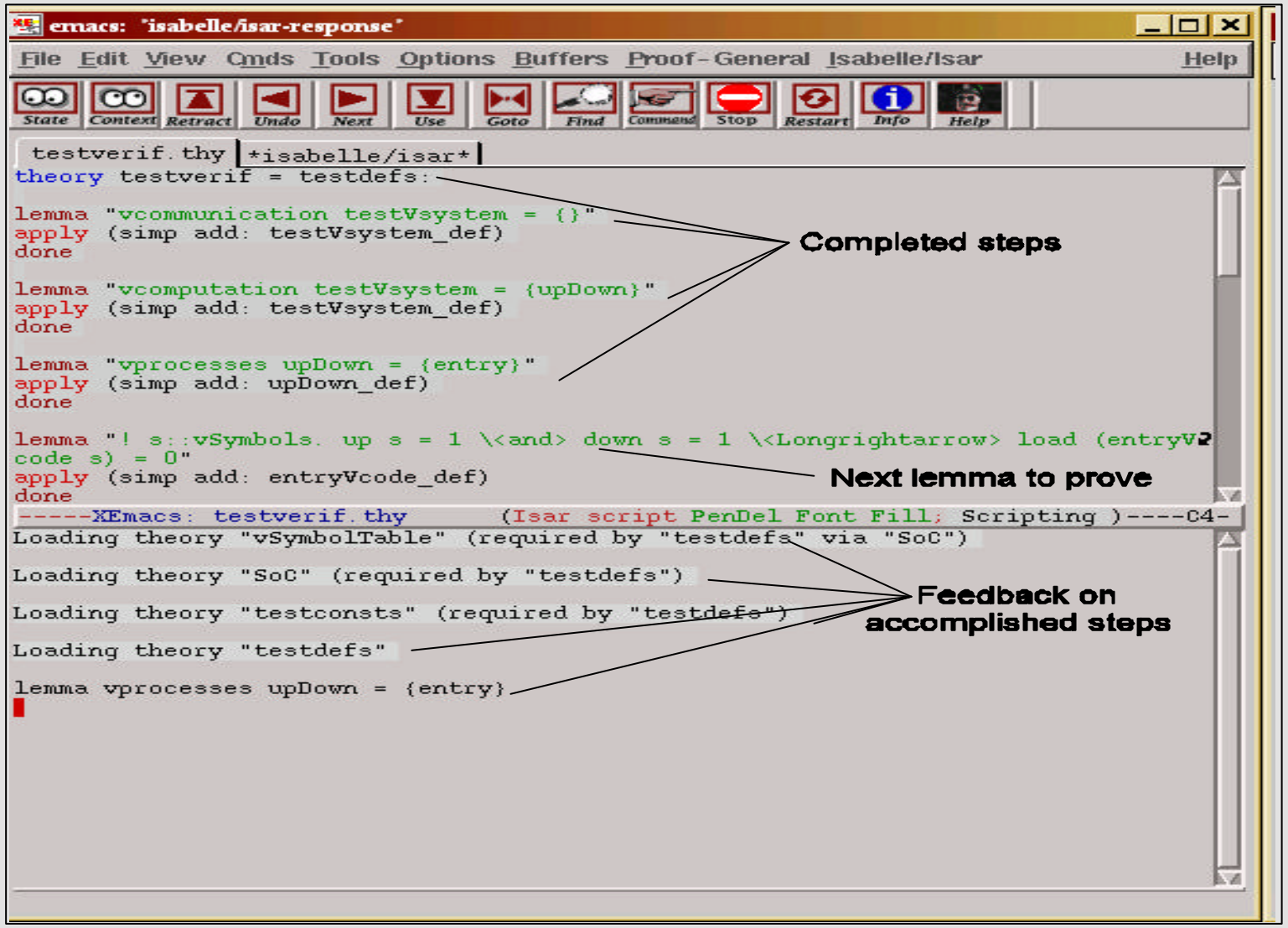
Tools

Isabelle/HOL/Proof General

- Isabelle: Theorem proving assistant by T.U.M and Cambridge
  - –Will not think up the proofs for you
  - –But will simplify what it can simplify
  - –Comes with different logics to reason in
- HOL: Higher Order Logic
  - –The chosen logic for this work
  - –The System Model is described in this logic
- Proof General: Xemacs based GUI by David Aspinal
  - –Simplifies work with Isabelle/HOL
  - –Same project in Eclipse

Tools

Isabelle/HOL/ProofGeneral



Proof General

Tools

ANTLR: ANother Tool for Language Recognition

- Developed by Terence Parr

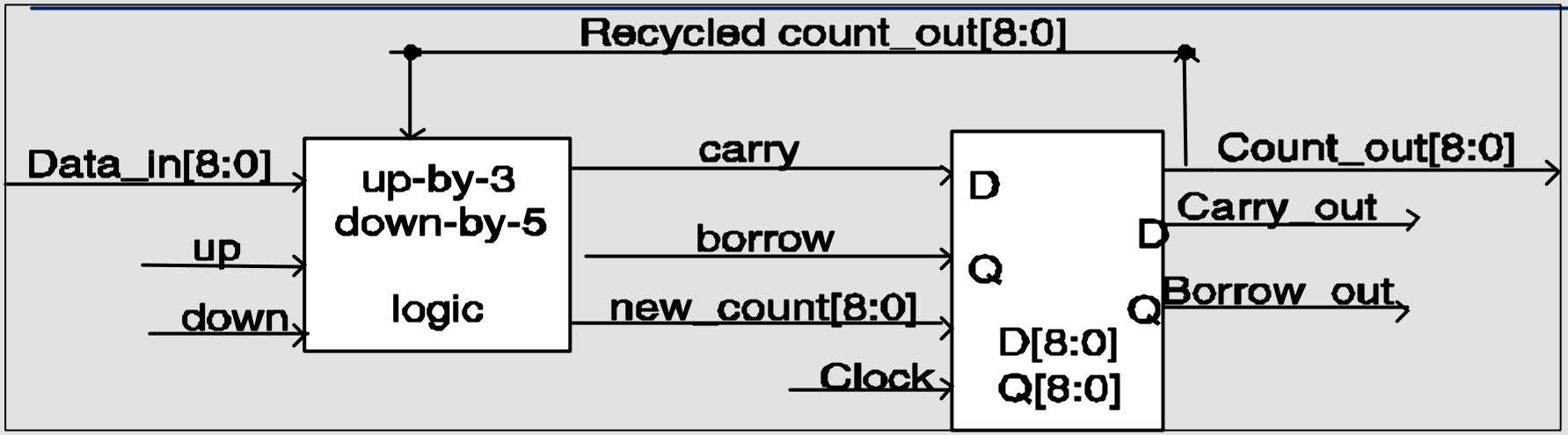
- Simple and Straightforward tools
  - –lexer, parser and tree creation

- “recursive-descent predictive” or “LL(k)” parsing technique

E.G.

```
NAME: (‘a’..’z’|’A’..’Z’)+ NEWLINE;
NEWLINE: ‘\r’’\n’ | ‘\n’;
StartRule: n: NAME {cout << “Name: ”<< n.getText() << endl;};
```

Simple Test Model

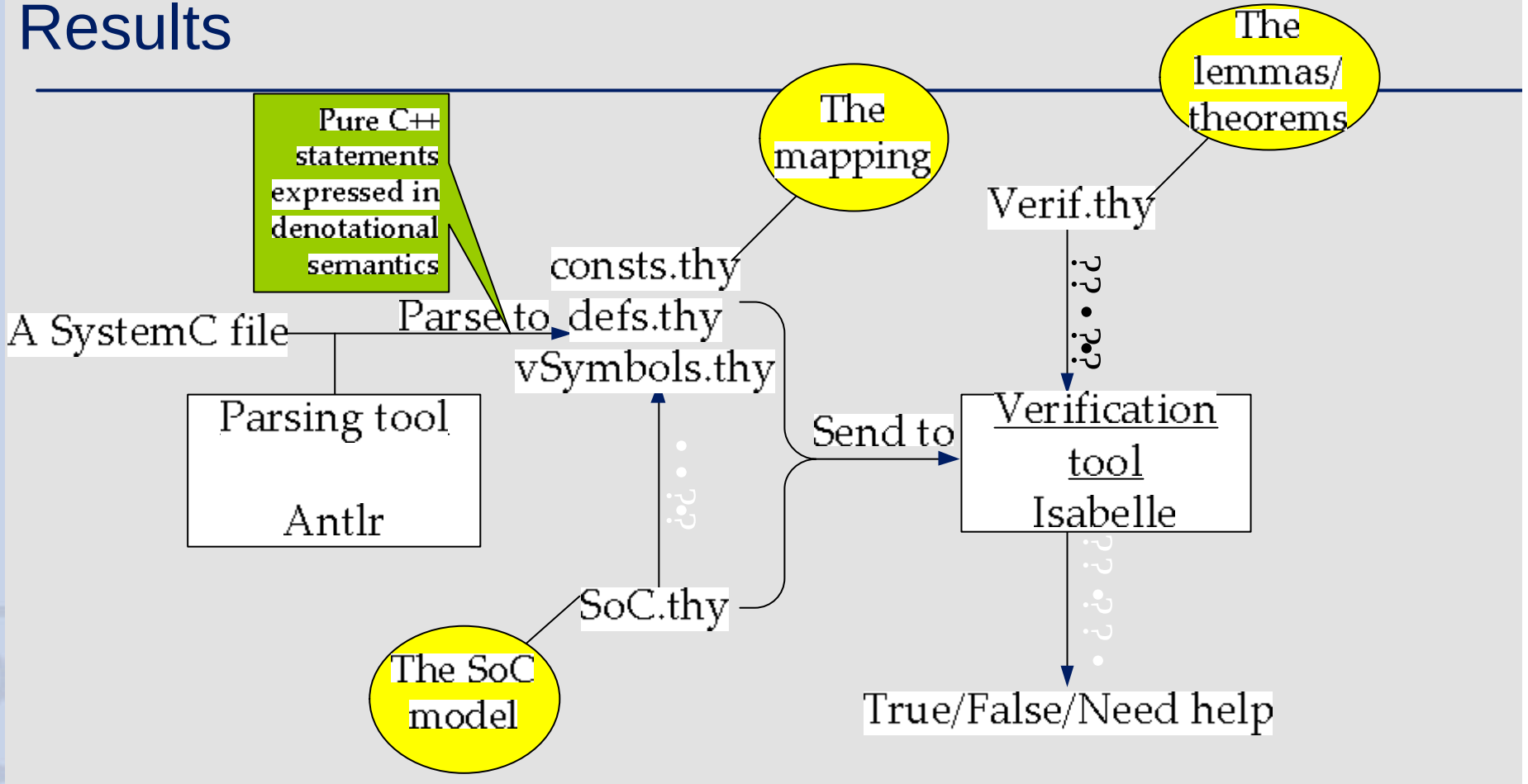


Up-By-3 Down-By-5 counter (no parity)

UP	DOWN	DATA_IN	COUNT_OUT
0	0	Valid	Load DATA_IN
0	1	Don't care	Q-5
1	0	Don't care	Q+3
1	1	Don't care	Q unchanged

Up-By-3 Down-By-5 counter specification

Results



Formal Verification platform

- Lemmas: examples

lemma “[|up s = 1; down s = 1|] → load (entryCode s) = 0"

lemma “[|up s = 0; down s = 0|] → count_out (entryCode s) = data_in (entryCode s)

Conclusion

- Simple Test: Simple lemma, proven by simplification
  - –Lemma, apply simp, done
- Above work accomplished without prior HOL knowledge
  - –Accessible to system engineers for now
- Small subset of SystemC recognised
  - –Risk: SystemC/C++ mapping difficult
- Efficient platform construction needs a whole lot more work
  - –E.g. Implementation detail needed to write proofs. No encapsulation
  - –Some constructs take to long during simplification
- Next step: Verification of a high level description of UMTS Cell search