Hardware Verification

Specification and Verification with Higher-Order Logic

Arnd Poetzsch-Heffter
(Slides by Jens Brandt)

Software Technology Group
Fachbereich Informatik
Technische Universität Kaiserslautern

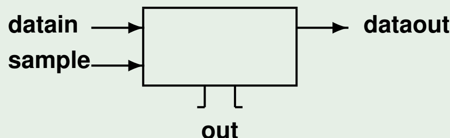
Sommersemester 2008

Outline

1. Introduction
2. Hardware Verification in HOL
   - General Approach
   - Composition
   - Verification
   - Full Adder Example
3. Summary

Why Formal Hardware Verification?

Example (Simple Sampler)



Informal Specification

The input line **datain** accepts a stream of bits, and the output line **dataout** emits the same stream delayed by four cycles. The bus **out** is four bits wide. If the input **sample** is false then the 4-bit word at **out** is the last four bits input at **datain**. Otherwise, the output word is all zeros.

Inadequacies of Informal Specification

Problems

- The informal specification is vague:
  - Does ‘the last four bits input’ include the current bit?
- The informal specification is incomplete:
  - What is the value at **dataout** during the first three cycles?
- The informal specification is unusable:
  - Natural language can’t easily be simulated or compiled!

Outline

1 Introduction

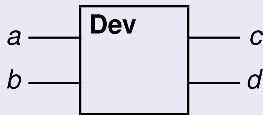
2 Hardware Verification in HOL
- General Approach
- Composition
- Verification
- Full Adder Example

3 Summary

Formal Specification in HOL

General Approach

Consider the following device **Dev**

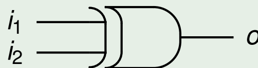


specified by a boolean term **Dev**(a, b, c, d)

$$\text{Dev}(a, b, c, d) = \begin{cases} \text{T} & \text{if } a,\ b,\ c, \text{ and } d \text{ could occur} \\ & \text{simultaneously on the corresponding} \\ & \text{external wires of the device Dev} \\ F & \text{otherwise} \end{cases}$$

Combinational Specification Examples

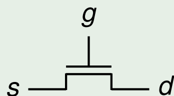
Example (Exclusive-or (XOR) Gate)



$$\vdash \text{Xor}(i_1, i_2, o) = (o = \neg(i_1 = i_2))$$

Xor : *bool* $\times$ *bool* $\times$ *bool* $\rightarrow$ *bool*

Example (Bidirectional Wires)



$$\vdash \text{Ntran}(g, s, d) = (g \Rightarrow (d = s))$$

Ntran : *bool* $\times$ *bool* $\times$ *bool* $\rightarrow$ *bool*

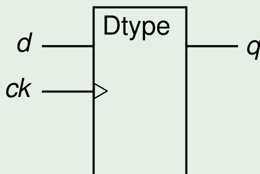
Sequential (Time-Dependent) Devices

Example (Unit delay)

$$\vdash \mathrm{Del}(in, out) = \forall t.\ out(t+1) = in\ t$$

$\mathrm{Del} : (time \rightarrow bool) \times (time \rightarrow bool) \rightarrow bool$

Example (Rising edge triggered DTYPE register)



$\vdash \mathrm{Dtype}(ck, d, q) = \forall t.\, q(t+1) = \texttt{if}\ \mathrm{Rise}\, ck\, t\ \texttt{then}\ d\ t\ \texttt{else}\ q\, t)$

$\vdash \mathrm{Rise}\ ck\ t = \neg ck(t) \wedge ck(t+1)$

$\mathrm{Dtype} : (time \rightarrow bool) \times (time \rightarrow bool) \times (time \rightarrow bool) \rightarrow bool$

Specification of the Sampler

Example (Sampler)

- Identify *time* with 'number of clock cycles' so *time*=*nat*
- We can specify the sampler formally by

$$\forall t{::}nat.\ (dataout(t) = datain(t-4))\ \wedge$$
$$(out(t) = \texttt{if}\ sample(t)$$
$$\texttt{then}\ [F;F;F;F]\ \texttt{else}\ [datain(t-4);\ datain(t-3);\ datain(t-2);\ datain(t-1)])$$

Formal Specification

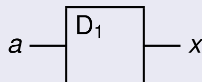
The formal specification is

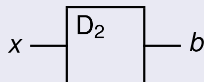
- precise
  'the last four bits input' does not include the current bit.
- complete
  can infer that *dataout* equals $datain(0)$ during the first three cycles (assuming $m < n \Rightarrow (m-n = 0)$)
- usable
  the predicate calculus notation can be processed by machine

...but you need some knowledge to read it

Composing Behaviours

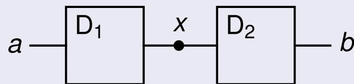
Consider the following two devices





$\mathscr{S}_1[a, x]$ $\mathscr{S}_2[x, b]$

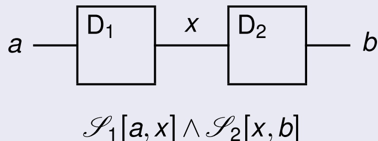
Conjunction ($\wedge$) models connecting components together



$\mathscr{S}_1[a, x] \wedge \mathscr{S}_2[x, b]$

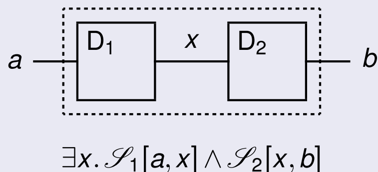
Hiding Internal Structure

Consider the composite device



$$\mathscr{S}_1[a,x] \wedge \mathscr{S}_2[x,b]$$

Existential quantification (∃) ‘hides’ internal wires



$$\exists x.\, \mathscr{S}_1[a,x] \wedge \mathscr{S}_2[x,b]$$

Why does it work?

Hiding Internal Behaviour

- $D_1\ f_1\ (a, x) = (x = f_1 a)$
- $D_2\ f_2\ (x, b) = (b = f_2 x)$

$$
\begin{array}{lll}
 & D\,(f_1, f_2)(a, b) & \\
= & \exists x. D_1\ f_1\ (a, x) \wedge D_2\ f_2\ (x, b) & \\
= & \exists x. (x = f_1\ a) \wedge (b = f_2\ x) & \text{expanding definitions} \\
= & \exists x. (x = f_1\ a) \wedge (b = f_2 (f_1\ a)) & \text{unwinding} \\
= & (\exists x. (x = f_1\ a)) \wedge (b = f_2 (f_1\ a)) & \text{narrowing scope of } \exists \\
= & \mathrm{T} \wedge (b = f_2 (f_1\ a)) & \text{properties of } \exists \\
= & (b = f_2 (f_1\ a)) & \text{property of } \wedge
\end{array}
$$

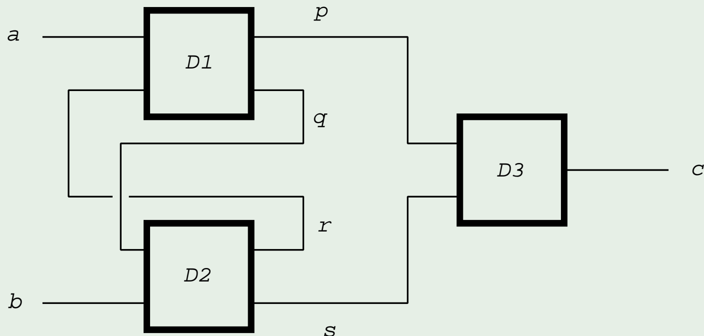
In HOL

Example (Composition in HOL)

```
consts f1 :: "(bool => bool)"
consts f2 :: "(bool => bool)"

fun DevIO :: "(bool => bool) => (bool * bool) => bool" where
  "DevIO f (x,y) = (y = f x)"

constdefs D1 :: "(bool * bool) => bool"
  "D1 == DevIO f1"

constdefs D2 :: "(bool * bool) => bool"
  "D2 == DevIO f2"

fun D :: "(bool * bool) => bool" where
  "D (a,b) = (EX x. D1 (a,x) & D2 (x,b))"

lemma "D (a,b) = (b = f2 (f1 a))"
```

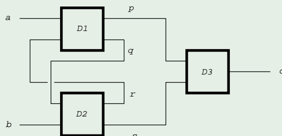
Another example

Example (Composed System)



Another example

Example (Composed System)



consts f1, f3, f4, f5 :: "(bool ∗ bool) => bool"
consts f2 :: "bool => bool"
fun D1 :: "(bool ∗ bool ∗ bool ∗ bool) => bool" **where**
  "D1 (w,x,y,z) = (y = f1 (w,x) & z = f2 w)"
fun D2 :: "(bool ∗ bool ∗ bool ∗ bool) => bool" **where**
  "D2 (w,x,y,z) = (y = f3 (w,x) & z = f4 (w,x))"
fun D3 :: "(bool ∗ bool ∗ bool) => bool" **where**
  "D3 (x,y,z) = (z = f5 (x,y))"
fun D :: "(bool ∗ bool ∗ bool) => bool" **where**
  "D (a,b,c) = (EX p q r s. D1 (a,r,p,q) & D2 (q,b,r,s) & D3 (p,s,c))"
lemma "D (a,b,c) = (c = f5 (f1 (a,(f3 ((f2 a),b))),(f4 ((f2 a),b))))"

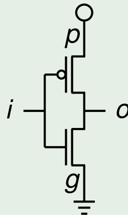
Formulating Correctness

What means correct?

- The strongest formulation is equivalence
  $\vdash \forall v_1 \ldots v_n.\ \mathscr{M}[v_1, \ldots, v_n] = \mathscr{S}[v_1, \ldots, v_n]$
- For *partial* specifications, use implication
  $\vdash \forall v_1 \ldots v_n.\ \vdash \mathscr{M}[v_1, \ldots, v_n] \Rightarrow \mathscr{S}[v_1, \ldots, v_n]$
- In general, the satisfaction relationship in
  $\vdash \mathscr{M}[v_1, \ldots, v_n]$ satisfies $\mathscr{S}[v_1, \ldots, v_n]$
  must be one of abstraction.

A Simple Correctness Proof

Example (CMOS Inverter)



Suppose we wish to verify that $o = \neg i$.

Three Steps

- define a model of the circuit in logic
- formulate the correctness of the circuit
- prove the correctness of the circuit

Formal specifications of CMOS Primitives

Simple switch model

$$\vdash \mathrm{Ptran}(g, s, d) = (\neg g \Rightarrow (d = s))$$

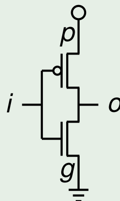
$$\vdash \mathrm{Ntran}(g, s, d) = (g \Rightarrow (d = s))$$

$$\vdash \mathrm{Gnd}\ g = (g = \mathrm{F})$$

$$\vdash \mathrm{Pwr}\ p = (p = \mathrm{T})$$

Design Model and Correctness

Example (CMOS Inverter)



- The design is modelled using composition and hiding:
  $\vdash \mathrm{Inv}(i, o) = \exists g\, p.\, \mathrm{Pwr}\ p \wedge \mathrm{Gnd}\ g \wedge \mathrm{Ntran}(i, g, o) \wedge \mathrm{Ptran}(i, p, o)$
- Correctness is formulated by the equivalence:
  $\vdash \forall i\, o.\, \mathrm{Inv}(i, o) = (o = \neg i)$

The Correctness Proof

Example (CMOS Inverter)

- Definition of Inv:
  $\vdash \mathrm{Inv}(i, o) = \exists g\, p.\, \mathrm{Pwr}\ p \wedge \mathrm{Gnd}\ g \wedge \mathrm{Ntran}(i, g, o) \wedge \mathrm{Ptran}(i, p, o)$
- Expanding with definitions:
  $\vdash \mathrm{Inv}(i, o) = \exists g\, p.\, (p = \mathrm{T}) \wedge (g = \mathrm{F}) \wedge$
  $(i \Rightarrow (o = g)) \wedge (\neg i \Rightarrow (o = p))$
- By simple logical reasoning:
  $\vdash \mathrm{Inv}(i, o) = (i \Rightarrow (o = \mathrm{F})) \wedge (\neg i \Rightarrow (o = \mathrm{T}))$
- Simplifying gives:
  $\vdash \mathrm{Inv}(i, o) = (i \Rightarrow \neg o) \wedge (\neg i \Rightarrow o)$

The Correctness Proof

Example (CMOS Inverter)

$\vdash \text{Inv}(i, o) = (i \Rightarrow \neg o) \wedge (\neg i \Rightarrow o)$

- By the law of the contrapositive:
  $\vdash \text{Inv}(i, o) = (o \Rightarrow \neg i) \wedge (\neg i \Rightarrow o)$
- By the definition of boolean equality:
  $\vdash \text{Inv}(i, o) = (o = \neg i)$
- Generalizing the free variables gives:
  $\vdash \forall i\, o.\, \text{Inv}(i, o) = (o = \neg i)$

In HOL

Example (CMOS Inverter)

```
fun Pwr :: "bool => bool" where "Pwr p = (p = True)"
fun Gnd :: "bool => bool" where "Gnd p = (p = False)"
fun Ntran :: "(bool * bool * bool) => bool" where "Ntran (x,y,z) = (x --> y=z)"
fun Ptran :: "(bool * bool * bool) => bool" where "Ptran (x,y,z) = (~x --> y=z)"

fun Inv :: "(bool * bool) => bool" where
  "Inv (inp,out) = (EX p g. Pwr p & Gnd g & Ntran (inp,g,out) & Ptran(inp,p,out))"

lemma "Inv (inp,out) = (out = (~inp))"
```

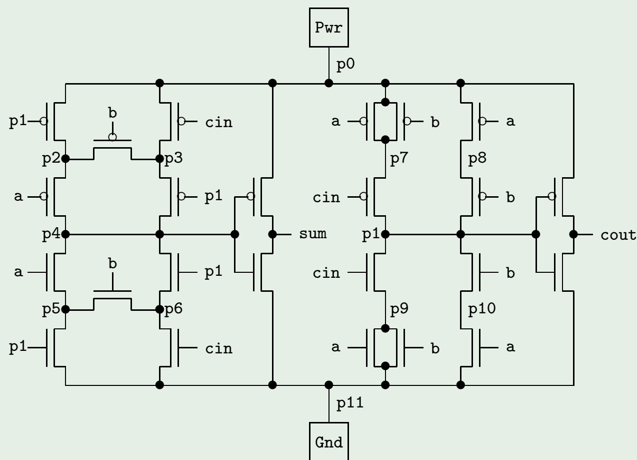
Hardware Verification in HOL

Scope of the Method

- The inverter example is, of course, trivial!
- But the same method has been applied to
  - a commercial CMOS cell library,
  - several (small) complete microprocessors
  - complex signed-binary arithmetic hardware
  - many other systems, large and small
- Features of the approach:
  - the specification language is just logic
    - logic can mimic HDL constructs
  - the rules of reasoning are also pure logic
    - special-purpose derived rules are possible
  - big formal proofs require machine assistance

Bigger example

Example (Full Adder)



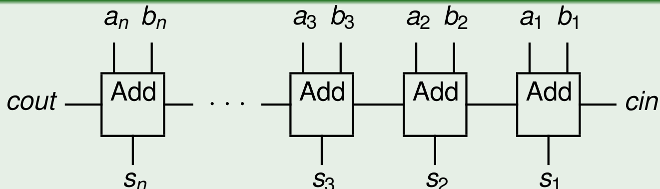
In HOL

Example (Full Adder)

```
fun Add1 :: "(bool ∗ bool ∗ bool ∗ bool ∗ bool) => bool" where
 "Add1(a,b,cin,sum,cout) =
    (EX p0 p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 p11.
      Ptran(p1,p0,p2) & Ptran(cin,p0,p3) & Ptran(b,p2,p3) & Ptran(a,p2,p4) &
      Ptran(p1,p3,p4) & Ntran(a,p4,p5) & Ntran(p1,p4,p6) & Ntran(b,p5,p6) &
      Ntran(p1,p5,p11) & Ntran(cin,p6,p11) & Ptran(a,p0,p7) & Ptran(b,p0,p7) &
      Ptran(a,p0,p8) & Ptran(cin,p7,p1) & Ptran(b,p8,p1) & Ntran(cin,p1,p9) &
      Ntran(b,p1,p10) & Ntran(a,p9,p11) & Ntran(b,p9,p11) & Ntran(a,p10,p11) &
      Pwr(p0) & Ptran(p4,p0,sum) & Ntran(p4,sum,p11) & Gnd(p11) &
      Ptran(p1,p0,cout) & Ntran(p1,cout,p11) )"

constdefs Bv :: "bool => nat"
  "Bv x == if x then 1 else 0"

lemma "Add1(a,b,cin,sum,cout) = (2∗(Bv cout) + Bv sum = Bv a + Bv b + Bv cin)"
```

Ripple-Carry Adder

Example (An n-bit ripple-carry adder)



We wish to prove that:

- $(2^n \times cout) + s = a + b + cin$
- (note that *cout*, *s*, *a*, *b*, *cin* are numbers)
- There are, as usual, three steps ...

Notation for Bits and Words

Bits and Words

- Bits are represented by booleans
  - T represents 1 and F represents 0
- Bv *b* is the number represented by *b*
  - $\mathrm{Bv} : \mathit{bool} \to \mathit{nat}$
  - $\mathrm{Bv}\ b = \texttt{if}\ b\ \texttt{then}\ 1\ \texttt{else}\ 0$
- Words are represented as lists of booleans
  - the head of a list represents the most significant bit
  - example: 1011 represented as [T, F, T, T]

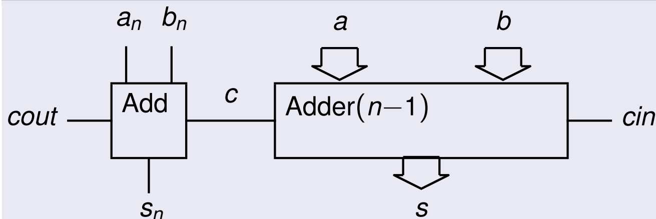
Notation for Bits and Words

Bits and Words

- V w is the natural natber represented in binary by w
  - example: V [T;F;T;T] $=$ 11
  - V : *bool list* $\longrightarrow$ *nat*
  - V w easily defined by primitive recursion on the length of w
- Bit w i selects ith bit of w

Recursive Definition of an n-bit Adder

Primitive Recursive Definition



$\vdash \text{Adder}(\,[\;]\,,[\;]\,,cin,\,[\;]\,,cout) = (cout = cin)$

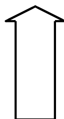
$\vdash \text{Adder}((a_n :: a),(b_n :: b),cin,(s_n :: s),cout) =$
$\quad \exists\, c.\ \text{Add}(a_n, b_n, c, s_n, cout)\ \wedge\ \text{Adder}(a, b, cin, s, c)$

Adder : $\textit{bool list} \times \textit{bool list} \times \textit{bool} \times \textit{bool list} \times \textit{bool} \rightarrow \textit{bool}$

Formulation of Correctness

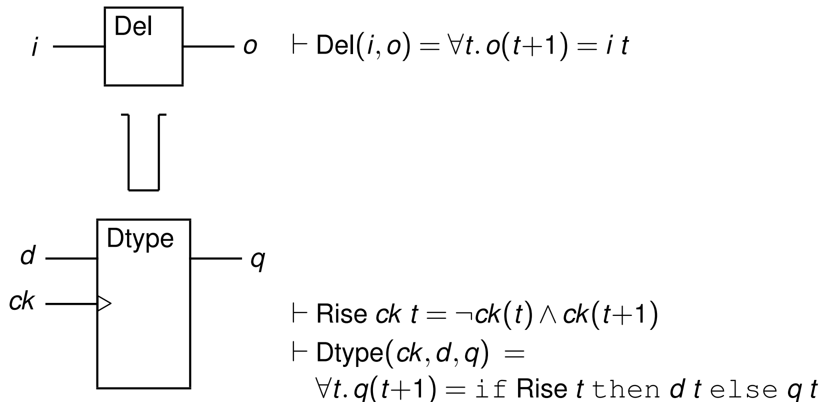
- Specification
  $\vdash \text{AdderSpec}\ n\ (a, b, cin, s, cout) = ((2^n \times cout) + s = a + b + cin)$
- Correctness condition
  $\vdash \forall n\ a\ b\ cin\ s\ cout.$
  $\quad \text{Adder}(a, b, cin, s, cout) \Rightarrow$
  $\quad \text{AdderSpec}(\text{V}\ a, \text{V}\ b, \text{Bv}\ cin, \text{V}\ s, \text{Bv}\ cout)$
- Note the data abstraction

$\text{AdderSpec}(a, b, cin, s, cout)$ numbers



$\text{Adder}(a, b, cin, s, cout)$ words

Temporal Abstraction: example – abstracting to unit delay



$\vdash \mathrm{Del}(i, o) = \forall t.\, o(t+1) = i\ t$

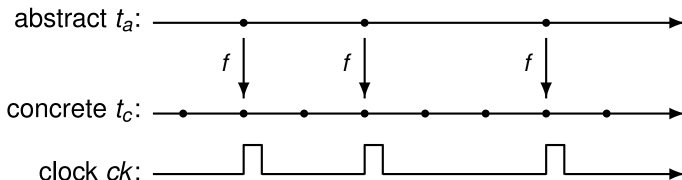
$\vdash \mathrm{Rise}\ ck\ t = \neg ck(t) \wedge ck(t+1)$

$\vdash \mathrm{Dtype}(ck, d, q) =$
$\forall t.\, q(t+1) = \texttt{if}\ \mathrm{Rise}\ t\ \texttt{then}\ d\ t\ \texttt{else}\ q\ t$

- Notions of time involved:
  - coarse grain of time – unit time = 1 clock cycle
  - fine grain of time – unit time $\approx$ 1 gate delay

Formulating Correctness

- A mapping between time-scales:



- Define the temporal abstraction functions:
  $\vdash$ Timeof $\mathscr{P}$ $n =$ *the time t_c such that P is true for the nth time*
  $\vdash s \text{ when } P = s \circ (\text{Timeof } P)$
- Then correctness is stated by:
  $\vdash \forall ck.\, \text{Inf}(\text{Rise } ck) \Rightarrow$
  $\forall d\, q.\, \text{Dtype}(ck, d, q) \Rightarrow$
  $\text{Del}(d \text{ when } (\text{Rise } ck), q \text{ when } (\text{Rise } ck))$
- Note the formal validity condition
  $\vdash \text{Inf } P = \forall t.\, \exists t'.\, t' > t \wedge P\, t'$

Outline

1 Introduction

2 Hardware Verification in HOL
- General Approach
- Composition
- Verification
- Full Adder Example

3 Summary

Summary

Summary

- Specifying behaviour:
  - predicates – $\mathscr{S}[a, b, c, d]$
- Specifying/expressing structure:
  - composition – $\mathscr{S}_1[a, x] \wedge \mathscr{S}_2[x, b]$
  - hiding – $\exists x.\ \mathscr{S}_1[a, x] \wedge \mathscr{S}_2[x, b]$
- Formulating correctness:
  - $\vdash \forall v_1 \ldots v_n.\ \mathscr{M}[v_1, \ldots, v_n] = \mathscr{S}[v_1, \ldots, v_n]$
  - $\vdash \forall v_1 \ldots v_n.\ \mathscr{M}[v_1, \ldots, v_n] \Rightarrow \mathscr{S}[v_1, \ldots, v_n]$
  - $\vdash \forall v_1 \ldots v_n.\ \mathscr{M}[v_1, \ldots, v_n] \Rightarrow \mathscr{S}[\text{Abs } v_1, \ldots, \text{Abs } v_n]$
- Abstraction:
  - data – mappings between data types
  - temporal – mappings between time scales

Alternative Technique

Embedding Semantics of HDLs in HOL

- lets you use the engineer-friendly HDL notation
- formally verify correctness properties
- metatheorems about HDL