Interactive Formal Verification 12: Modelling Hardware

Tjark Weber (Slides: Lawrence C Paulson) Computer Laboratory University of Cambridge

• Define *mathematical abstractions* of the objects of interest (systems, hardware, protocols,...).

- Define *mathematical abstractions* of the objects of interest (systems, hardware, protocols,...).
- Whenever possible, use *definitions* not axioms!

- Define mathematical abstractions of the objects of interest (systems, hardware, protocols,...).
- Whenever possible, use *definitions* not axioms!
- Ensure that the abstractions capture enough detail.
 - Unrealistic models have unrealistic properties.
 - Inconsistent models will satisfy *all* properties.

- Define mathematical abstractions of the objects of interest (systems, hardware, protocols,...).
- Whenever possible, use *definitions* not axioms!
- Ensure that the abstractions capture enough detail.
 - Unrealistic models have unrealistic properties.
 - Inconsistent models will satisfy *all* properties.

All models involving the real world are *approximate*!

• Pioneered by M. J. C. Gordon and his students, using successive versions of the HOL system.

- Pioneered by M. J. C. Gordon and his students, using successive versions of the HOL system.
- Used to model substantial hardware designs, including the ARM6 processor.

- Pioneered by M. J. C. Gordon and his students, using successive versions of the HOL system.
- Used to model substantial hardware designs, including the ARM6 processor.
- Works *hierarchically* from arithmetic units and memories right down to flip-flops and transistors.

- Pioneered by M. J. C. Gordon and his students, using successive versions of the HOL system.
- Used to model substantial hardware designs, including the ARM6 processor.
- Works *hierarchically* from arithmetic units and memories right down to flip-flops and transistors.
- Crucially uses higher-order logic, modelling signals as boolean-valued functions over time.

Devices as Relations



The relation describes the possible combinations of values on the ports.

Values could be bits, words, signals (functions from time to bits), etc





two devices modelled by two formulas





 $\mathbf{S}_1[a,x]$



two devices modelled by two formulas

 D_2 *x* D_1 **-** b *a* — $\mathbf{S}_1[a,x] \wedge \mathbf{S}_2[x,b]$

the connected ports have the same value





 $\mathbf{S}_1[a, x]$

 $\mathbf{S}_2[x,b]$

two devices modelled by two formulas



the connected ports have the same value



the connected ports have some value

• The *implementation* of a device in terms of other devices can be expressed by composition.

- The *implementation* of a device in terms of other devices can be expressed by composition.
- The specification of the device's intended behaviour can be given by an abstract formula.

- The *implementation* of a device in terms of other devices can be expressed by composition.
- The specification of the device's intended behaviour can be given by an abstract formula.
- Sometimes the implementation and specification can be proved equivalent: Imp⇔Spec.

- The *implementation* of a device in terms of other devices can be expressed by composition.
- The specification of the device's intended behaviour can be given by an abstract formula.
- Sometimes the implementation and specification can be proved equivalent: Imp⇔Spec.
- The property $Imp \Rightarrow Spec$ ensures that every possible behaviour of the Imp is permitted by Spec.

- The *implementation* of a device in terms of other devices can be expressed by composition.
- The specification of the device's intended behaviour can be given by an abstract formula.
- Sometimes the implementation and specification can be proved equivalent: Imp⇔Spec.
- The property Imp⇒Spec ensures that every possible behaviour of the Imp is permitted by Spec.
 Impossible implementations satisfy all specifications!

The Switch Model of CMOS

$$g \rightarrow d \qquad \mathsf{Ptran}(g, s, d) = (\neg g \Rightarrow (d = s))$$

$$g \rightarrow d \qquad \mathsf{Ntran}(g, s, d) = (g \Rightarrow (d = s))$$

$$g \rightarrow d \qquad \mathsf{Gnd} \ g = (g = \mathsf{F})$$

$$p \rightarrow \mathsf{Pwr} \ p = (p = \mathsf{T})$$

The Switch Model of CMOS



Pwr
$$p = (p = \mathbf{T})$$

subsection{* Specification of CMOS primitives *}

text{* P and N transistors *} definition "Ptran = ($\lambda(g,a,b)$. (~g $\rightarrow a = b$))" definition "Ntran = ($\lambda(g,a,b)$. (g $\rightarrow a = b$))"

text{* Power and Ground*}
definition "Pwr p = (p = True)"
definition "Gnd p = (p = False)"

Full Adder: Specification



 $2 \times cout + sum = a + b + cin$

Full Adder: Specification



$2 \times cout + sum = a + b + cin$

Full Adder: Implementation



Full Adder in Isabelle

$\odot \odot \odot$		Adder.thy		\bigcirc
တ္ ဏ 🗶 ┥	🕨 🗶 🛏 🖀 🔎 🕦	ver 😄 😌 🚏		
<pre>text{* 1-bit 0</pre>	MOS full adder imple	mentation *}		ĥ
definition "Ad	$d1Imp = (\lambda(a,b,cin,s))$	um,cout).		
	∃p0 p1 p2 p3 p4 p5 p	6 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) 	^	U
	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)))"	
text{* Verific	ation of CMOS full a	dder *}		
Lemma Add1Corr	ect:			
"Add1Imp(a,	b,cin,sum,cout) = Ad	d1Spec(a,b,cin,sum,co	ut)"	1
by (simp add:	Pwr_def Gnd_def Ntra	n_def Ptran_def Add1S	pec_def	Ă
	Add1Imp_def bit_val_	def ex_bool_eq)		Ψ.
-u-:**- Adder.	thy 27% L53	(Isar Utoks Abbrev; S	cripting)	
				11.

Full Adder in Isabelle

		er 🚽 👽 👷		
text{* 1-D1	t CMOS TULL adder impleme	ntation *}		
definition	$Add1Tmp = \Omega(a,b,cin,sum)$	cout).		
	∃ng n1 n2 n3 n4 n5 n6	p7 p8 p9 p10 p11.		
	$Ptran(p1,p0,p2)$ \land	Ptran(cin.p0.p3)	٨	
	$Ptran(b,p2,p3)$ ^	Ptran(a,p2,p4)	٨	
	$Ptran(p1,p3,p4) \land$	Ntran(a, p4, p5)	٨	
	Ntran(p1,p4,p6) ^	Ntran(b,p5,p6)	٨	
	Ntran(p1,p5,p11) ^	Ntran(cin,p6,p11)	٨	
	Ptran(a,p0,p7) ^	Ptran(b,p0,p7)	Λ	
	Ptran(a,p0,p8) ^	Ptran(cin,p7,p1)	Λ	
	Ptran(b,p8,p1) ^	Ntran(cin,p1,p9)	٨	
	Ntran(b,p1,p10) ^	Ntran(a,p9,p11)	٨	
	Ntran(b,p9,p11) ^	Ntran(a,p10,p11)	Λ	
	Pwr(p0) ^	Ptran(p4,p0,sum)	Λ	
	Ntran(p4,sum,p11) ^	Gnd(p11)	∧	
	Ptran(p1,p0,cout) ^	Ntran(p1,cout,p11))"	
tout [* Voni	fication of CMOS full add	on *]		
lomma Add1C	TICATION OF CMUS TULL dad	er *}		
"Add1Tmp	(a h cin sum cout) - Add1	Spec(a h cin sum co	u+)"	
by (simp ad	d: Pwr def Gnd def Ntran	def Ptran def Add1S	nec def	
by (Stinp uu	Add11mp def bit val de	f ex bool ea)	pec_uer	
u-:**- Add	er.thy 27% L53 (1	sar Utoks Abbrev: S	cripting)	





 $(2^n \times cout) + s = a + b + cin$

• Cascading several full adders yields an *n*-bit adder.



- Cascading several full adders yields an *n*-bit adder.
- The implementation is expressed recursively.



- Cascading several full adders yields an *n*-bit adder.
- The implementation is expressed recursively.
- The specification is obvious mathematics.

Adder Specification

Adder Specification

$$(2^n \times cout) + s = a + b + cin$$

values of n-bit words

Adder Specification



text{* Unsigned number denoted by bitstring f(n-1)...f(0) *}

fun bits_val where
 "bits_val f 0 = 0"
I "bits_val f (Suc n) = 2^n * bit_val(f n) + bits_val f n"

text{* Specification of an n-bit adder *}

definition

```
"AdderSpec n = (λ(a, b, cin, sum, cout).
    2^n * bit_val cout + bits_val sum n =
    bits_val a n + bits_val b n + bit_val cin)"
```
Adder Specification



text{* Unsigned number denoted by bitstring f(n-1)...f(0) *}

fun bits_val where
 "bits_val f 0 = 0"
I "bits_val f (Suc n) = 2^n * bit_val(f n) + bits_val f n"

text{* Specification of an n-bit adder *}

definition

"AdderSpec n = (λ(a, b, cin, sum, cout).
 2^n * bit_val cout + bits_val sum n =
 bits_val a n + bits_val b n + bit_val cin)"

Adder Specification



Adder Specification







text{* Implementation of an n-bit ripple-carry adder*}

fun AdderImp where

"AdderImp 0 (a, b, cin, sum, cout) = (cout = cin)"
I "AdderImp (Suc n) (a, b, cin, sum, cout) =
 (∃c. AdderImp n (a, b, cin, sum, c) ^
 Add1Imp (a n, b n, c, sum n, cout))"



text{* Implementation of an n-bit ripple-carry adder*}

fun AdderImp where

"AdderImp 0 (a, b, cin, sum, cout) = (cout = cin)"
I "AdderImp (Suc n) (a, b, cin, sum, cout) =
 (∃c. AdderImp n (a, b, cin, sum, c) ^
 Add1Imp (a n, b n, c, sum n, cout))"





Partial Correctness Proof

```
000
                                     Adder.thy
00 00 I 🔺 🕨 Y 🛏 🖀 🔎 🕦 🐷 🤤 🤣 🚏
lemma AdderCorrect:
     "AdderImp n (a, b, cin, sum, cout) \Rightarrow AdderSpec n (a, b, cin, sum, cout)"
proof (induct n arbitrary: cout)
  case 0 thus ?case
    by (simp add: AdderSpec_def)
next
  case (Suc n)
then obtain c
    where AddS: "AdderSpec n (a, b, cin, sum, c)"
    and Add1: "Add1Imp (a n, b n, c, sum n, cout)"
    by (auto intro: Suc)
                                  (Isar Utoks Abbrev; Scripting )-
-u-:--- Adder.thy
                       53% L85
this:
  AdderImp n (a, b, cin, sum, ?cout) \Rightarrow AdderSpec n (a, b, cin, sum, ?cout)
  AdderImp (Suc n) (a, b, cin, sum, cout)
goal (1 subgoal):
 1. An cout.
       [Acout.
           AdderImp n (a, b, cin, sum, cout) \Rightarrow
           AdderSpec n (a, b, cin, sum, cout);
        AdderImp (Suc n) (a, b, cin, sum, cout)
       \Rightarrow AdderSpec (Suc n) (a, b, cin, sum, cout)
-u-:%%- *aoals*
                        5% L4 (Isar Proofstate Utoks Abbrev;)------
```

Partial Correctness Proof

000	Adder.thy	\supset
00 CO 🔳	🔺 🕨 🗶 H 🖀 🔎 🚯 🕼 😂 🥸 🚏	
lemma Adde "Adde proof (ind case 0 t by (sin next	rCorrect: rImp n (a, b, cin, sum, cout) ⇒ AdderSpec n (a, b, cin, sum, cout)" uct n arbitrary: cout) hus ?case mp add: AdderSpec_def)	0
 case (Survey) then obtended where and and by (aurvey) -u-: Ad 	c n) ain c AddS: "AdderSpec n (a, b, cin, sum, c)" Add1: "Add1Imp (a n, b n, c, sum n, cout)" to intro: Suc) der.thy 53% L85 (Isar Utoks Abbrev: Scripting)) 4
this: AdderImp AdderImp	<pre>n (a, b, cin, sum, ?cout) ⇒ AdderSpec n (a, b, cin, sum, ?cout) (Suc n) (a, b, cin, sum, cout)</pre>	0
goal (1 su 1. ∧n cou [∧c	bgoal): assumptions cout. AddenTmp p (a, b, cip, sum, cout) \Rightarrow	
Ad ⇒ -u-:%%- *g	AdderSpec n (a, b, cin, sum, cout); derImp (Suc n) (a, b, cin, sum, cout)] AdderSpec (Suc n) (a, b, cin, sum, cout) oals* 5% L4 (Isar Proofstate Utoks Abbrev;)	↓ ▼
		11.

Partial Correctness Proof

000	Adder.thy	\bigcirc
00 00	▼ ◀ ▶ Y H 🆀 🔎 🗊 🐖 🖨 🦻	
lemma A proof (case by	dderCorrect: dderImp n (a, b, cin, sum, cout) ⇒ AdderSpec n (a, b, cin, sum, cout)" induct n arbitrary: cout) Ø thus ?case (simp add: AdderSpec_def)	(
next case then whe	(Suc n) obtain c re AddS: "AdderSpec n (a, b, cin, sum, c)"	0
and	Add1: "Add1Imp (a n, b n, c, sum n, cout)"	4
by	(auto intro: Suc) Adder thy 53% 180 (Isar Utoks Abbrev: Scripting December 2010)	Ť.
have (/ ?t	<pre>\c. [AdderSpec n (a, b, cin, sum, c); Add1Imp (a n, b n, c, sum n, cout)] ⇒ ?thesis) ⇒ hesis</pre>	
-u-:%%-	<pre>*response* All L4 (Isar Messages Utoks Abbrev;)</pre>	
-		111

000	Adder.thy	\bigcirc
\odot \odot	፲ ◀ ▶ ፻ ⊨ 🏰 🔎 🕕 🐖 🖨 🤣 🚏	
lemma A proof (case by next case then whe and by -u-: have (/	AdderCorrect: AdderImp n (a, b, cin, sum, cout) ⇒ AdderSpec n (a, b, cin, sum, cout)" (induct n arbitrary: cout) 0 thus ?case (simp add: AdderSpec_def) (Suc n) obtain c ere AddS: "AdderSpec n (a, b, cin, sum, c)" 1 Add1: "AddIImp (a n, b n, c, sum n, cout)" (auto intro: Suc) Adder.thy 53% L&0 (Isar Utoks Abbrev; Scripting) \c. [AdderSpec n (a, b, cin, sum, c); AddIImp (a n, b n, c, sum n, cout)] ⇒ ?thesis) ⇒ thesis	
-u-:%%-	<pre>*response* All L4 (Isar Messages Utoks Abbrev;)</pre>	

A Tiresome Calculation

000	Adder.thy	\bigcirc
00 00 I 🔺 🕨 I	। 🖀 🔎 🚯 🕼 🤤 🤣 🚏 👘	
<pre>where AddS: "Adder and Add1: "Add11 by (auto intro: Su have "bit_val (sum n (bit_val (sum by (simp add: alge also have " = (bi (2 using Add1 by (sim finally show "AdderS by (simp add: Adder by (simp add: Adder u-: Adder.thy calculation: bit_val (sum n) * 2 (bit val c + (bit val</pre>	<pre>Spec n (a, b, cin, sum, c)" np (a n, b n, c, sum n, cout)") * (2 ^ n) + bit_val cout * (1) + (bit_val cout * 2)) * (2 0ra_simps) c_val c + (bit_val (a n) + bit ^ n)" 0 add: Add1Correct Add1Spec_de 0ec (Suc n) (a, b, cin, sum, conspec_def algebra_simps) 57% L96 (Isar Utoks Abbrev ^ n + bit_val cout * (2 * 2 ^ (a n) + bit val (b n))) * 2</pre>	<pre>2 * 2 ^ n) = ^ n)" _val (b n))) * f) out)" using AddS r; Scripting)</pre>
-u-:%%- *response*	All L3 (Isar Messages Uto	ks Abbrev;)
tool-bar next		1.

A Tiresome Calculation

<pre> where AddS: "AdderSpec n (a, b, cin, sum, c)" and Add1: "Add1Imp (a n, b n, c, sum n, cout)" by (auto intro: Suc) have "bit_val (sum n) * (2 ^ n) + bit_val cout * (2 * 2 ^ n) = </pre>	000	Adder.thy	\bigcirc	
<pre>where AddS: "AdderSpec n (a, b, cin, sum, c)" and Add1: "Add1Imp (a n, b n, c, sum n, cout)" by (auto intro: Suc) have "bit_val (sum n) * (2 ^ n) + bit_val cout * (2 * 2 ^ n) =</pre>	သ က 🗶 🔺 🕨	- 🗴 🛏 🖀 🔎 🕦 🕼 🗢 🗢 🚏 🔜 🔜		
<pre>by (simp add: algebra_simps) also have " = (bit_val c + (bit_val (a n) + bit_val (b n))) *</pre>	where AddS: and Add1: by (auto int have "bit_val (bit_val by (simp add also have "	<pre>"AdderSpec n (a, b, cin, sum, c)" "Add1Imp (a n, b n, c, sum n, cout)" ro: Suc) (sum n) * (2 ^ n) + bit_val cout * (2 * 2 ^ n) = (sum n) + (bit_val cout * 2)) * (2 ^ n)" : algebra_simps) = (bit_val c + (bit_val (a n) + bit_val (b n))) * (2 ^ n)"</pre>	ranging the term	าร
 using Add1 by (simp add: Add1Correct Add1Spec_def) finally show "AdderSpec (Suc n) (a, b, cin, sum, cout)" using AddS by (simp add: AdderSpec_def algebra_simps) -u-: Adder.thy 57% L96 (Isar Utoks Abbrev; Scripting)	<pre>finally show " by (simp add -u-: Adder.th</pre>	y (simp add: Add1Correct Add1Spec_def) AdderSpec (Suc n) (a, b, cin, sum, cout)" using AddS 1: AdderSpec_def algebra_simps) 1y 57% L96 (Isar Utoks Abbrev; Scripting)	5	
<pre>calculation: bit_val (sum n) * 2 ^ n + bit_val cout * (2 * 2 ^ n) = (bit_val c + (bit_val (a n) + bit_val (b n))) * 2 ^ n</pre>	calculation: bit_val (sum n (bit_val c + () * 2 ^ n + bit_val cout * (2 * 2 ^ n) = bit_val (a n) + bit_val (b n))) * 2 ^ n		
-u-:%%- *response* All L3 (Isar Messages Utoks Abbrev;)	-u-:%%- *respons tool-bar next	e* All L3 (Isar Messages Utoks Abbrev;)		

A Tiresome Calculation

The Finished Proof

```
000
                                     Adder.thy
                                                                                \bigcirc
00 00 I 🔺 🕨 Y 🛏 🖀 🔎 🗊 🐷 🤤 🤣 🚏
text{* Partial correctness of ripple-carry adder for all n by induction *}
lemma AdderCorrect:
     "AdderImp n (a, b, cin, sum, cout) \implies AdderSpec n (a, b, cin, sum, cout)"
proof (induct n arbitrary: cout)
  case 0 thus ?case
    by (simp add: AdderSpec_def)
next
  case (Suc n)
  then obtain c
    where AddS: "AdderSpec n (a, b, cin, sum, c)"
    and Add1: "Add1Imp (a n, b n, c, sum n, cout)"
    by (auto intro: Suc)
  have "bit_val (sum n) * (2 ^ n) + bit_val cout * (2 * 2 ^ n) =
        (bit_val (sum n) + (bit_val cout * 2)) * (2 ^ n)"
    by (simp add: algebra_simps)
  also have "... = (bit_val c + (bit_val (a n) + bit_val (b n))) *
                   (2 ^ n)"
    using Add1 by (simp add: Add1Correct Add1Spec_def)
  finally show "AdderSpec (Suc n) (a, b, cin, sum, cout)" using AddS
    by (simp add: AdderSpec_def algebra_simps)
aed
-u-:--- Adder.thy
                       51% L78
                                  (Isar Utoks Abbrev; Scripting )------
```

The Finished Proof

Proving Equivalence

Proving Equivalence

Proving Equivalence

000		Adder.thy		\bigcirc
$\infty \infty$	⊼ ◀ ► ⊻ ⊬ (🎍 📣 🚯 📨 🤤 🤣 🎽		
lemma b by (ind	its_val_less: "bits uct n, auto simp ad	:_val f n < 2^n" ld: bit_val_def)		
lemma A	dderSpec_Suc: AdderSpec (Suc n) ((∃c. AdderSpec n (d	a, b, cin, sum, cout) , b, cin, sum, c) & A) = Add1Spec (a n, b n, c, sum n,	cout P
using b by (sim	its_val_less [of a p add: AdderSpec_de	n] bits_val_less [of ef Add1Spec_def ex_boo	b n] bits_val_less [of sum nj ol_eq bit_val_def)	ב ב
-u-:	Adder.thy 855	6 L139 (Isar Utoks /	Abbrev; Scripting)	
proof (using t bits_ bits_ bits_	prove): step 1 his: val a n < 2 ^ n val b n < 2 ^ n val sum n < 2 ^ n			
goal (1 1. Add (∃c	<pre>subgoal): erSpec (Suc n) (a, . AdderSpec n (a, b Add1Spec (a n, b</pre>	<pre>b, cin, sum, cout) = o, cin, sum, c) ^ n, c, sum n, cout))</pre>		
-u-:%%-	*goals* 19	6 L2 (Isar Proofs	tate Utoks Abbrev;)	

000	Adder.thy	0
00 00	▲ ▶ 포 ⊨ 🆀 🔎 🚯 🐖 🖨 🦻	
lemma by by (ind lemma A))" using b by (sin	<pre>bits_val_less: "bits_val f n < 2^n" duct n, auto simp add: bit_val_def) AdderSpec_Suc: "AdderSpec (Suc n) (a, b, cin, sum, cout) = (3c. AdderSpec n (a, b, cin, sum, c) & Add1Spec (a n, b n, c, sum n, cout a bits_val_less [of a n] bits_val_less [of b n] bits_val_less [of sum n] mp add: AdderSpec_def Add1Spec_def ex_bool_eq bit_val_def)</pre>	
-u-: proof (using t bits_ bits_ bits_	Adder.thy 85% L139 (Isar Utoks Abbrev; Scripting) (prove): step 1 this: _val a n < 2 ^ n _val b n < 2 ^ n _val sum n < 2 ^ n	
goal (1 1. Ada (∃a	<pre>1 subgoal): derSpec (Suc n) (a, b, cin, sum, cout) = c. AdderSpec n (a, b, cin, sum, c) ^ Add1Spec (a n, b n, c, sum n, cout)) *goals* 1% L2 (Isar Proofstate Utoks Abbrev;)</pre>	

The Opposite Implication

$\odot \odot \odot$	Adder.thy	\supset
00 00	▼ ◀ ▶ Y ⊨ 🏰 🔎 🕕 🐖 🖨 🤣 🚏	
lemma A "A apply (apply (apply (dderCorrect2: dderSpec n (a, b, cin, sum, cout) ⇒ AdderImp n (a, b, cin, sum, cout)" induct n arbitrary: cout) simp add: AdderSpec_def) auto simp add: AdderSpec_Suc Add1Correct)	(
done	Adden thy 91% 1148 (Tean Utoks Abbrev: Scripting)	
lemma Adder Add Add	Correct2: erSpec ?n (?a, ?b, ?cin, ?sum, ?cout) ⇒ erImp ?n (?a, ?b, ?cin, ?sum, ?cout)	
-u-:%%-	<pre>*response* All L4 (Isar Messages Utoks Abbrev;)</pre>	

The Opposite Implication

$\bigcirc \bigcirc \bigcirc \bigcirc$	Adder.thy	\supset
တတ	▼ ◀ ▶ ⊻ ⋈ ≟ ∞ ① ☞ ≎ ♡ 🚏	
lemma A "/ apply (apply (apply (dderCorrect2: dderSpec n (a, b, cin, sum, cout) ⇒ AdderImp n (a, b, cin, sum, cout)" induct n arbitrary: cout) simp add: AdderSpec_def) auto simp add: AdderSpec_Suc Add1Correct)	0
-u-:**-	Adder.thy 91% L148 (Isar Utoks Abbrev; Scripting)) + +
lemma Adder Ada Ada	Correct2: erSpec ?n (?a, ?b, ?cin, ?sum, ?cout) ⇒ erImp ?n (?a, ?b, ?cin, ?sum, ?cout)	0
	The implementation and specification are equivalent!	
-u-:%%-	<pre>*response* All L4 (Isar Messages Utoks Abbrev;)</pre>	

• thm [of a b c]

replaces variables by terms from left to right

- thm [of a b c]
 replaces variables by terms from left to right
- thm [where x=a]
 replaces the variable x by the term a

- thm [of a b c]
 replaces variables by terms from left to right
- thm [where x=a]
 replaces the variable x by the term a
- thm [OF thm₁ thm₂ thm₃]
 discharges premises from left to right

- thm [of a b c]
 replaces variables by terms from left to right
- thm [where x=a]
 replaces the variable x by the term a
- thm [OF thm₁ thm₂ thm₃]
 discharges premises from left to right
- *thm* [simplified]
 applies the simplifier to *thm*

- thm [of a b c]
 replaces variables by terms from left to right
- thm [where x=a]
 replaces the variable x by the term a
- thm [OF thm₁ thm₂ thm₃]
 discharges premises from left to right
- *thm* [simplified] applies the simplifier to *thm*
- *thm* [*attr*₁, *attr*₂, *attr*₃] applying multiple attributes

The End

You know my methods. Apply them! Sherlock Holmes