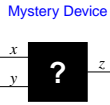


## Hardware Verification in HOL

The verification process:

1. Describe the specification.
2. Describe the implementation.
3. Prove that the implementation meets the specification.

How do we describe hardware?



Observations

x	y	z
on	on	off
on	off	off
off	off	on

## An Observational Model of Hardware

We need to describe hardware in the HOL logic:

- Wires can have the value on or off.  
We model them with boolean variables.
- Devices constrain the values that can be observed on the attached wires.  
We model these with predicates on wires.

Some example device definitions:

**A Not Gate:**

$$\text{NOT } xy \stackrel{\text{def}}{=} (y = \neg x)$$

**Connection to Power:**

$$\text{PWR } x \stackrel{\text{def}}{=} (x = \text{T})$$

**A Nand Gate:**

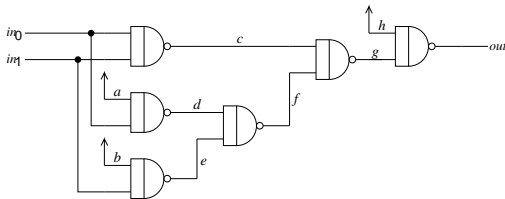
$$\text{NAND } xyz \stackrel{\text{def}}{=} (z = \neg(x \wedge y))$$

**Connection to Ground:**

$$\text{GND } x \stackrel{\text{def}}{=} (z = \text{F})$$

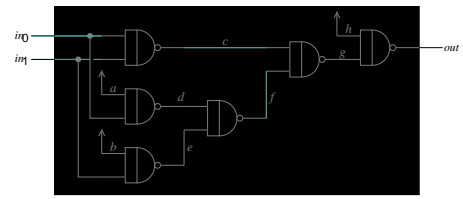
**Note:** These are relations, not functions. The NOT predicate does not state that  $x$  is the input and  $y$  is the output. This cannot be observed.

## Modeling Circuits



$$\begin{aligned} \text{HALF\_ADDER\_IMP } in_0 in_1 out &\stackrel{\text{def}}{=} \\ &\wedge \text{ NAND } in_0 in_1 c \wedge \text{PWR } a \\ &\wedge \text{ NAND } a in_1 d \wedge \text{PWR } b \\ &\wedge \text{ NAND } b in_1 e \wedge \text{NAND } d e f \\ &\wedge \text{ NAND } c f g \wedge \text{PWR } h \\ &\wedge \text{ NAND } h g out \end{aligned}$$

## Hiding Internal Connections



$$\begin{aligned} \text{HALF\_ADDER\_IMP } in_0 in_1 out &\stackrel{\text{def}}{=} \\ &\exists abcdefgh. \\ &\text{ NAND } in_0 in_1 c \wedge \text{PWR } a \\ &\wedge \text{ NAND } a in_1 d \wedge \text{PWR } b \\ &\wedge \text{ NAND } b in_1 e \wedge \text{NAND } d e f \\ &\wedge \text{ NAND } c f g \wedge \text{PWR } h \\ &\wedge \text{ NAND } h g out \end{aligned}$$

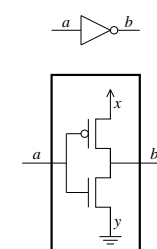
## A HOL Theory of CMOS

Power	Ground	N-Type Transistor	P-Type Transistor

```

val PWR_DEF = new_definition
  ("PWR_DEF", Term `PWR a = (a = T)`);
val GND_DEF = new_definition
  ("GND_DEF", Term `GND a = (a = F)`);
val NTRAN_DEF = new_definition
  ("NTRAN_DEF",
   Term `NTRAN g (a:B) b = (g ⊃ (a = b))`);
val PTRAN_DEF = new_definition
  ("PTRAN_DEF",
   Term `PTRAN g' (a:B) b = (¬g' ⊃ (a = b))`);
  
```

## A CMOS Not Gate



```

val NOT_DEF = new_definition
  ("NOT_DEF",
   Term `NOT a b = (b = ¬a)`);
val NOT_IMP_DEF = new_definition
  ("NOT_IMP_DEF",
   Term
    `NOT_IMP a b = ∃x y. PWR x
      ∧ GND y
      ∧ PTRAN a x b
      ∧ NTRAN a y b`);
  
```

What relationship should hold between NOT\_IMP a b and NOT a b?

## An Observational Notion of Correctness

We could prove that the CMOS Not is equivalent to its specification:

$$\forall a b. \text{NOT\_IMP } a b = \text{NOT } a b$$

In general, an implementation might be 'better' than its specification.

Circuit B is at least as good as circuit A

(is a correct implementation of circuit A)

if every possible observation of B is also a possible observation of A.

In the case of the Not gate, this means proving

$$\forall a b. \text{NOT\_IMP } a b \supset \text{NOT } a b$$

## The Proof of the Not Gate

```

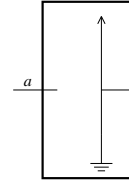
- g `∀a b. NOT_IMP a b ⊃ NOT a b`
> val it =
  Proof manager status: 1 proof.
  1. Incomplete:
    Initial goal:
      `∀a b. NOT_IMP a b ⊃ NOT a b`
    : GoalstackPure.goalstack
- e (REWRITE_TAC [NOT_IMP_DEF, NOT_DEF]);
OK..
1 subgoal:
> val it =
  `∀a b.
  (∃x y. PWR x ∧ GND y ∧ PTRAN a x b ∧ NTRAN a y b) ⊃
  (b = ¬a)`
  : GoalstackPure.goalstack
- e (REWRITE_TAC [PWR_DEF, GND_DEF, PTRAN_DEF, NTRAN_DEF]);
OK..
1 subgoal:
> val it =
  `∀a b.
  (∃x y. x ∧ ¬y ∧ (¬a ⊃ (x = b)) ∧ (a ⊃ (y = b))) ⊃
  (b = ¬a)`
  : GoalstackPure.goalstack
  
```

### The Not Proof Continued

```
- e (mesonLib.MESON_TAC []);
OK.
Meson search level: .....
Goal proved.
⊢ ∀ a b.
  (∃ x y. x ∧ ¬y ∧ (¬a ⊃ (x = b)) ∧ (a ⊃ (y = b))) ⊃ (b = ¬a)
Goal proved.
⊢ ∀ a b.
  (∃ x y. PWR x ∧ GND y ∧ PTRAN a x b ∧ NTRAN a y b) ⊃ (b = ¬a)
> val it =
  Initial goal proved.
  ⊢ ∀ a b. NOT_IMP a b ⊃ NOT a b
  : GoalstackPure.goalstack

- fun CMOS_TAC thms =
  (REWRITE_TAC
   (thms=[PWR_DEF, GND_DEF, PTRAN_DEF, NTRAN_DEF])) THEN
  (mesonLib.MESON_TAC []);
> val CMOS_TAC = fn : Thm.thm list -> tactic
- prove (Term `∀ a b. NOT_IMP a b ⊃ NOT a b`,
  CMOS_TAC [NOT_DEF, NOT_IMP_DEF]);
Meson search level: .....
> val it = ⊢ ∀ a b. NOT_IMP a b ⊃ NOT a b : Thm.thm
```

### Another CMOS Not Gate?

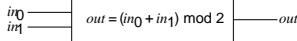


```
val NOT_IMP2_DEF = new_definition
  ("NOT_IMP2_DEF",
   Term
    `NOT_IMP2 a b = PWR b ∧ GND b`);
- prove (Term `∀ a b. NOT_IMP2 a b ⊃ NOT a b`,
  CMOS_TAC [NOT_DEF, NOT_IMP2_DEF]);
Meson search level: .....
> val it = ⊢ ∀ a b. NOT_IMP2 a b ⊃ NOT a b
  : Thm.thm
```

This device is not feasible. We also need to prove that for every input there is an acceptable output.

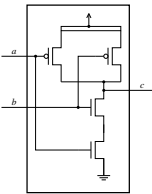
∀ a. ∃ b. NOT\_IMP2 a b

### Verifying the Half-Adder



```
val BVAL_DEF = new_definition ("BVAL_DEF", Term `
  BVAL a = (if a then 1 else 0)`);
val HALF_ADDER_DEF = new_definition ("HALF_ADDER_DEF", Term `
  HALF_ADDER in0 in1 out =
    ((BVAL out) = ((BVAL in0) + (BVAL in1)) MOD 2)`);

val NAND_IMP_DEF = new_definition (...);
val HALF_ADDER_IMP_DEF =
  new_definition
  ("HALF_ADDER_IMP_DEF",
   Term `
    HALF_ADDER_IMP in0 in1 out =
      (∃ a b c d e f g h.
       NAND_IMP in0 in1 c ∧ PWR a ∧
       NAND_IMP a in1 d ∧ PWR b ∧
       NAND_IMP b in1 e ∧ NAND_IMP d e f ∧
       NAND_IMP c f g ∧ PWR h ∧
       NAND_IMP h g out)`);
```

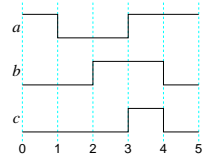


### A Model of Timed Hardware

The value on a wire can change over time. We will now model wires as functions from time (represented by a natural number) to boolean.

Example: A timed And gate:

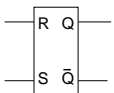
$$\text{AND } abc \stackrel{\text{def}}{=} \forall t. ct = at \wedge bt$$



Note: Since wires are now predicates, and devices predicates on the wires attached, we are now using the higher-order facilities of the logic.

### Example: The RS Flip-Flop

The flip-flop is a circuit with memory.



- If S is held high and R low for two time units, then by the third time unit Q will be high and Q-bar low.
- and visa versa
- If S and R are low and one of Q and Q-bar is low and the other high, then Q and Q-bar are unchanged in the next time unit.

```
stable x dt  $\stackrel{\text{def}}{=} \forall n. (n < d) \supset (x(t+n) = xt)$ 
RSrsq  $\stackrel{\text{def}}{=} \forall t. ((st \neq rt) \wedge \text{stable } s 2t \wedge \text{stable } 2t \supset$ 
  (q(t+2) = st) ∧ (q̄(t+2) = rt) ∧
  (¬(st) ∧ ¬(rt) ∧ (qt ≠ q̄t) ⊃
   (q(t+1) = qt) ∧ (q̄(t+1) = q̄t))
```

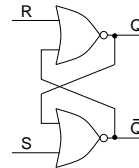
Note: This says nothing about initial and intermediate values of Q and Q-bar.

### The Flip-Flop Implementation

The implementation depends on the fact that gates take time to switch.

This Nor gate has a delay:

$$\text{NOR } abc \stackrel{\text{def}}{=} \forall t. c(t+1) = \neg(at \vee bt)$$



$$\text{RS\_IMP } rsq \stackrel{\text{def}}{=} \text{NOR } r \bar{q} \bar{q} \wedge \text{NOR } s q \bar{q}$$

Note: The overly precise specification of timing means we can prove properties about when R and S are high that real RS flip-flops don't have.

### Recursion and Replication

Recursion and the higher-order features can describe replicated hardware. Why? To describe n-bit wide data-paths and circuits.

- A family of wires (a bus) is a function from n to the nth wire in the family.

$$\begin{matrix} \text{Untimed} & \text{Timed} \\ N \rightarrow B & N \rightarrow N \rightarrow B \end{matrix}$$

- Example specification, n-bit wide negation:

$$\text{nNOT } nab \stackrel{\text{def}}{=} \forall m. (m < n) \supset \text{NOT } (am)(bm)$$

- Example implementation, n-bit wide negation:

$$\text{nNOT\_IMP } nab \stackrel{\text{def}}{=} \text{REPLICATE } n (\lambda m. \text{NOT\_IMP } (am)(bm))$$

### Untimed Replicate

$$\begin{aligned} \text{REPLICATE } 0 p &\stackrel{\text{def}}{=} T \\ \text{REPLICATE } (n+1) p &\stackrel{\text{def}}{=} P \text{ REPLICATE } n P \\ P n \wedge \text{REPLICATE } n P \end{aligned}$$

### Timed Replicate

$$\begin{aligned} \text{REPLICATE } 0 p t &\stackrel{\text{def}}{=} T \\ \text{REPLICATE } (n+1) p t &\stackrel{\text{def}}{=} P n t \wedge \text{REPLICATE } n P t \end{aligned}$$

Example:

$$\begin{aligned} \text{nNOT\_IMP } 3 ab &= \text{REPLICATE } 3 (\lambda m. \text{NOT\_IMP } (am)(bm)) \\ &= \text{NOT\_IMP } (a2)(b2) \wedge \text{NOT\_IMP } (a1)(b1) \wedge \text{NOT\_IMP } (a0)(b0) \wedge T \end{aligned}$$

