Intensive one week course in

Hardware Description Languages for Synchronous Hardware Design

Introduction

Gordon J. Pace

September 2000

Why hardware design and description languages?

Pretty much the same as to 'Why software design and high-level languages?', but ...

- hardware is less easy to test cheaply
- fixing bugs in hardware is more expensive

But isn't hardware boring?

- Isn't assembly programming boring?
- Configurable hardware starting to make hardware programming a necessity
- Much more varied compilation strategies (eg different technologies, different applications)

The plan for the course (I)

Monday The 'boring' stuff. Introduction to hardware, basic hardware components and low level hardware design.

Tuesday Verilog. A hardware description language designed to help simulate the hardware and compare to algorithmic specifications.

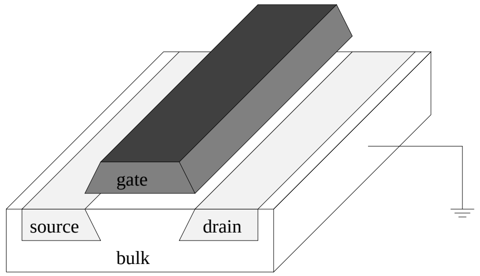
Wednesday Lava. Another hardware description language built upon Haskell, aimed at designing and verifying hardware within a functional programming language.

The plan for the course (II)

Thursday Compiling languages to hardware. The compilation of a small imperative programming language to hardware and how it can be extended.

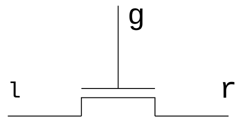
Friday Combining HDLs. How different languages can be combined together to allow multi-language algorithms to be compiled to hardware.

CMOS Transistors: The physical picture

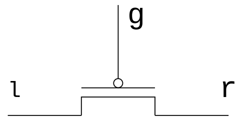


CMOS Transistors: An informal account

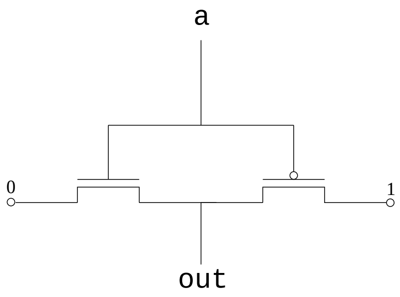
n-channel transistor If the gate carries `1` then the switch opens and `0` can be carried across the source and drain.



p-channel transistor If the gate carries `0` then the switch opens and `1` can be carried across the source and drain.

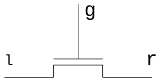


CMOS Transistors: An inverter



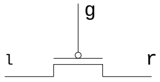
CMOS Transistors: Logical description

n-channel transistor



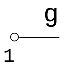
$$\textit{n-trans}(g,l,r) \stackrel{\text{def}}{=} (\delta g \wedge g) \implies (\delta l \wedge \neg l \;=\; \delta r \wedge \neg r)$$

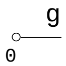
p-channel transistor



$$\textit{p-trans}(g,l,r) \stackrel{\text{def}}{=} (\delta g \wedge \neg g) \implies (\delta l \wedge l \;=\; \delta r \wedge r)$$

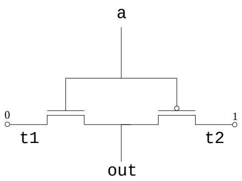
Power and ground





$$\textit{power}(g) \stackrel{\text{def}}{=} \delta g \wedge g \qquad\qquad \textit{ground}(g) \stackrel{\text{def}}{=} \delta g \wedge \neg g$$

CMOS Transistors: An inverter

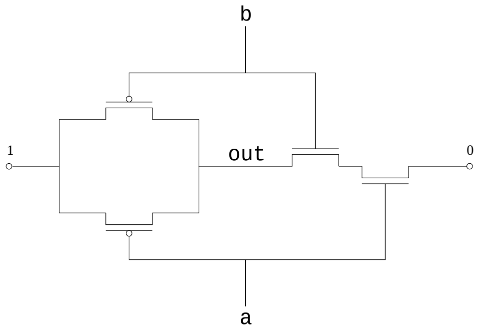


$$
\begin{aligned}
\texttt{impl} \stackrel{\text{def}}{=}\; & \textit{ground}(t1) \wedge \textit{power}(t2)\\
\wedge\; & \textit{p-trans}(a, \textit{out}, t1)\\
\wedge\; & \textit{n-trans}(a, \textit{out}, t2)
\end{aligned}
$$

$$\texttt{spec} \stackrel{\text{def}}{=} \delta a \implies (\delta \textit{out} \wedge a = \neg \textit{out})$$

The proof requirement: $\texttt{spec} \implies \texttt{impl}$

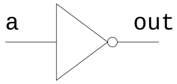
CMOS Transistors: A nand gate



a	b	out
0	0	1
0	1	1
1	0	1
1	1	0

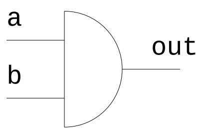
Gates (I)

Inverter:



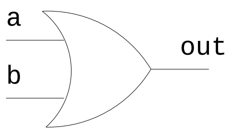
$\textit{out} = \neg a$

And:



$\textit{out} = a \wedge b$

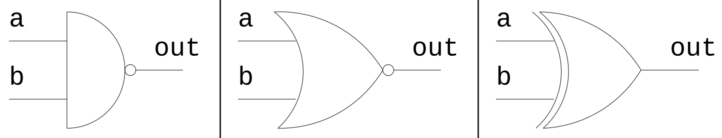
Or:



$\textit{out} = a \vee b$

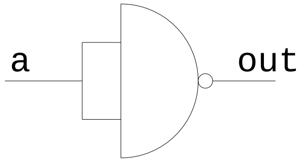
Gates (II)

Nand: | **Nor:** | **Xor:**



$out = \neg(a \wedge b)$ | $out = \neg(a \vee b)$ | $out = \neg(a \Leftrightarrow b)$

A wasteful inverter

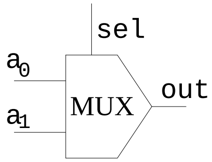


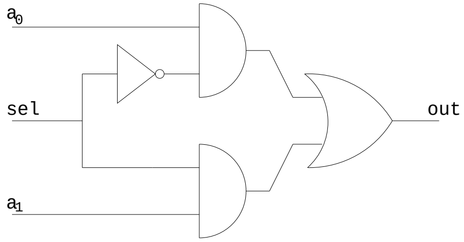
Proof of correctness:

$$\begin{aligned} out &= \neg(a \wedge a) \\ &= \neg a \end{aligned}$$

A multiplexer

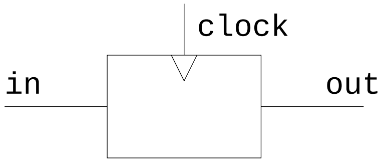
When `sel` is 0, output $\texttt{a}_0$, otherwise output $\texttt{a}_1$:





Circuits with memory: Delay

When there is a *rising edge* on `clock`, start outputing the value of `in` as it was before the rising edge.



`in`	*mem*	`out`	*mem'*
0	0	0	0
0	1	1	0
1	0	0	1
1	1	1	1

Synchronous vs asynchronous circuits

In synchronous circuits, a global clock is fed to *all* memory elements.

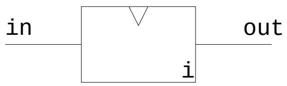
Pros:

- Much, much easier to reason about and test
- More reliable

Cons:

- Higher power consumption
- Clock skew in large circuits
- Clock has to tick at the speed of the slowest sub-circuit

Delay revisited



Signals are no longer *Boolean*:

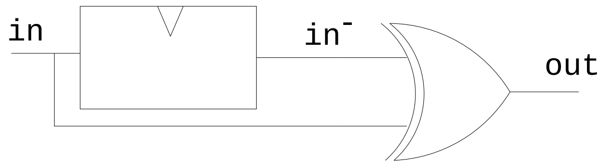
$$\text{Signal} \stackrel{\text{def}}{=} \mathit{Time} \longrightarrow \mathit{Boolean}$$

$$\mathit{out}(t+1) = \mathit{in}(t)$$

But what about at time 0?

Edge detector

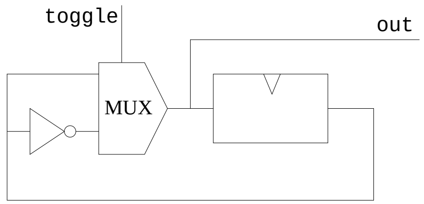
Requirement: Outputs 1 in clock cycles when the input has just changed:



Specification: $\mathit{out}(t+1) \Leftrightarrow (\mathit{in}(t) \neq \mathit{in}(t+1))$

Toggle

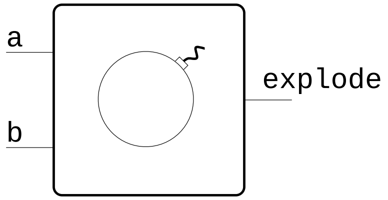
Requirement: Toggle the output whenever the `toggle` input is true.



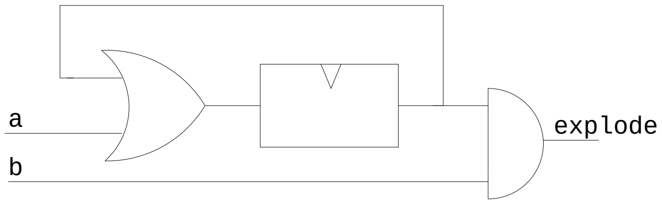
`toggle`	*mem*	`out`	*mem′*
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

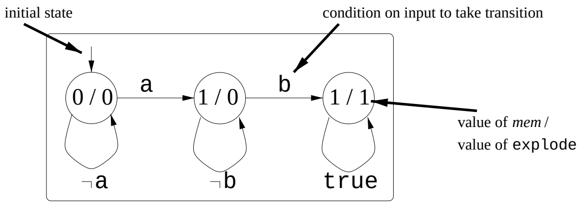
A safety critical system :)

Requirement: If `a` is true for some time, and then (later) `b` becomes true, explode.



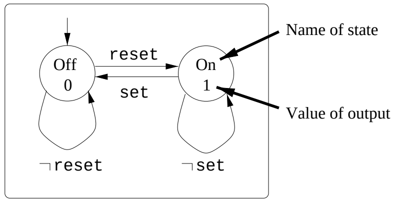
An implementation



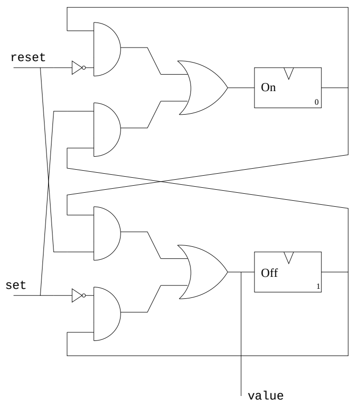


Set-reset

Requirement: Design a circuit with two inputs `set` and `reset` which starts outputing 0. Upon receiving 1 on `set`, the circuit starts outputing 1, and similarly upon receiving 1 on `reset`, the circuit starts outputing 0.



An implementation



Gate delays

- The previous circuit can be 'optimised' to use only one delay.
- Reducing the number of latches tends to increase the number of gates between the input/delay outputs and delay inputs.
- Gates take time to stabilise their output and thus longer chains of gates require a *slower clock*.

Conclusions

Circuits can be expressed at a number of different levels of abstraction:

- transistors
- gates and delays
- finite state machines

This course will attempt to answer the question:

How can high-level languages be used to design and describe hardware?