FPGAs for low latency audio applications ?

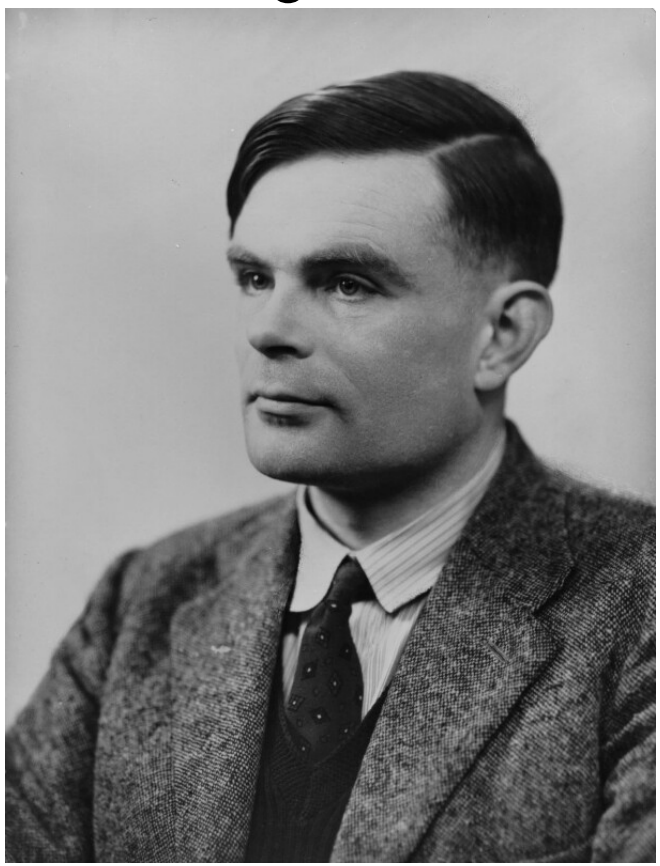
Tanguy Risset, Florent de Dinechin
Adeyemi Gbadamosi, Ousmane Touat, Gero Muller,
Stéphane Letz, Romain Michon, Yann Orlarey,
Alain Darte

Programmable Audio Workshop
(when the program is a circuit)

Introduction

The twin gods of the computing pantheon (A. C. Clarke)

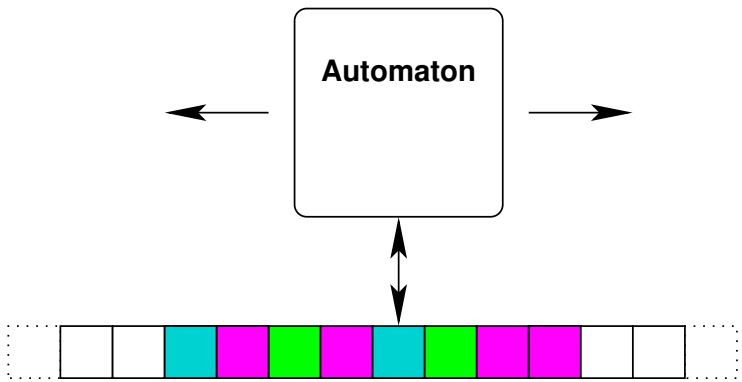
Alan Turing



John von Neumann

The twin gods invented the math and the machine

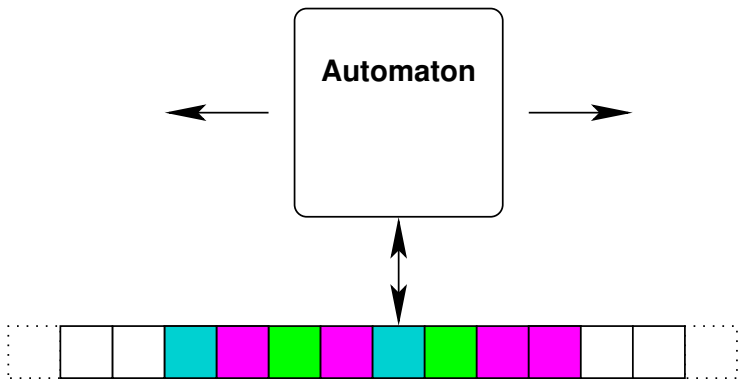
Turing machine



A universal computing object
(on paper)

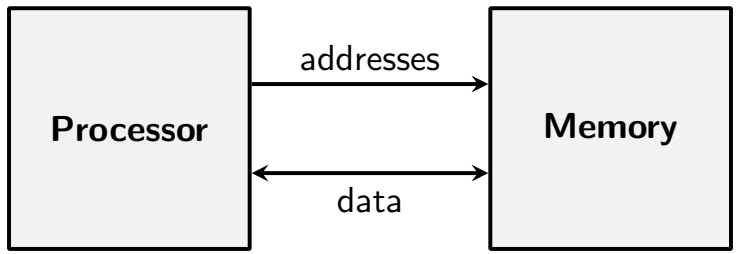
The twin gods invented the math and the machine

Turing machine



A universal computing object (on paper)

von Neumann machine



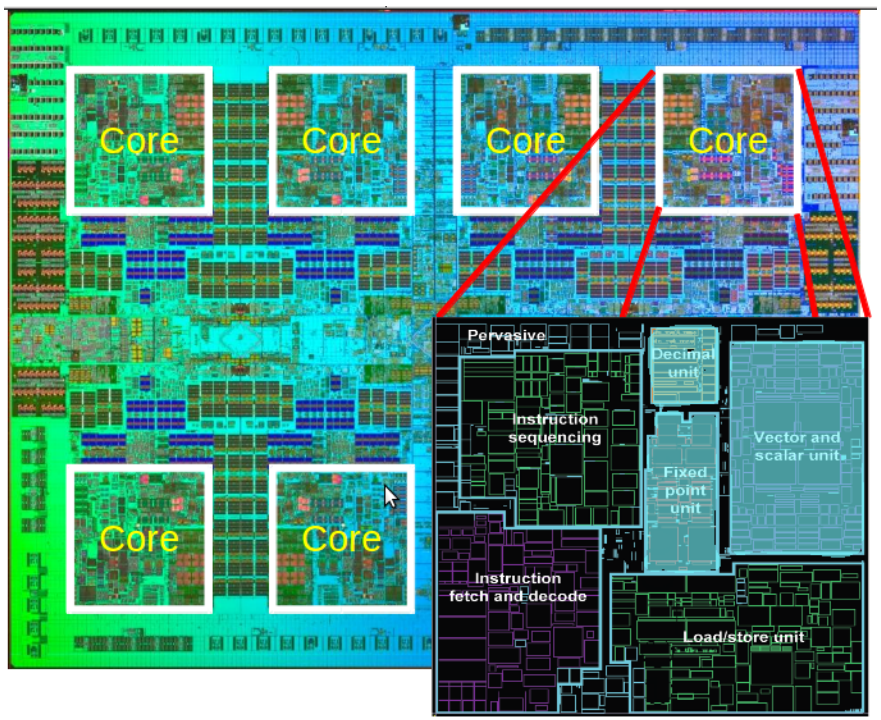
A **practical** universal computer thanks to **random access** to memory

When reality kicks back

Random access in constant (fast) time is impossible in practice...

A law of nature

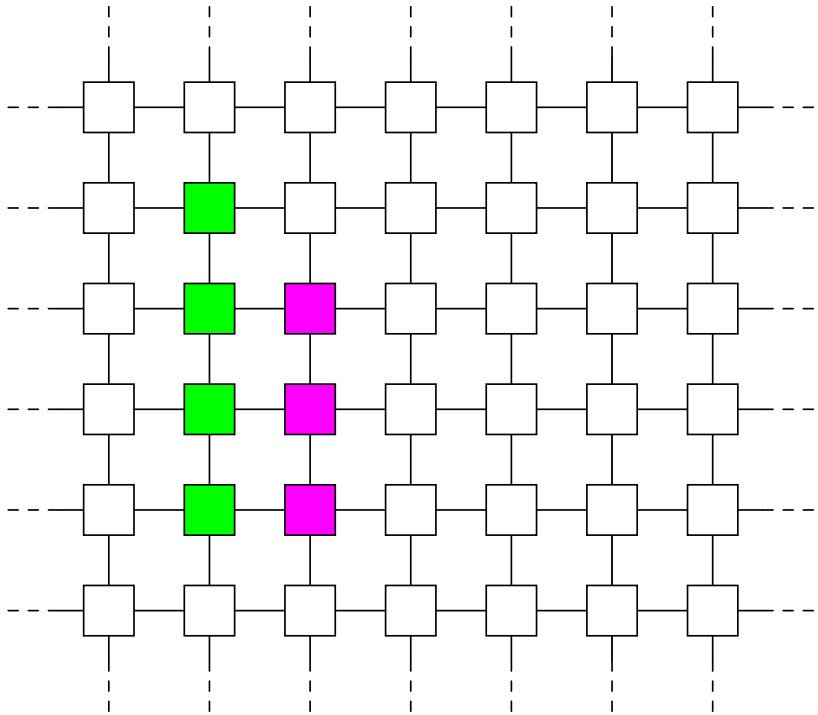
If the memory is infinite, some bits will be physically distant
and will therefore be accessed slowly.



Half of your silicon is there to maintain the illusion of random access...

Meanwhile, von Neumann had a better idea

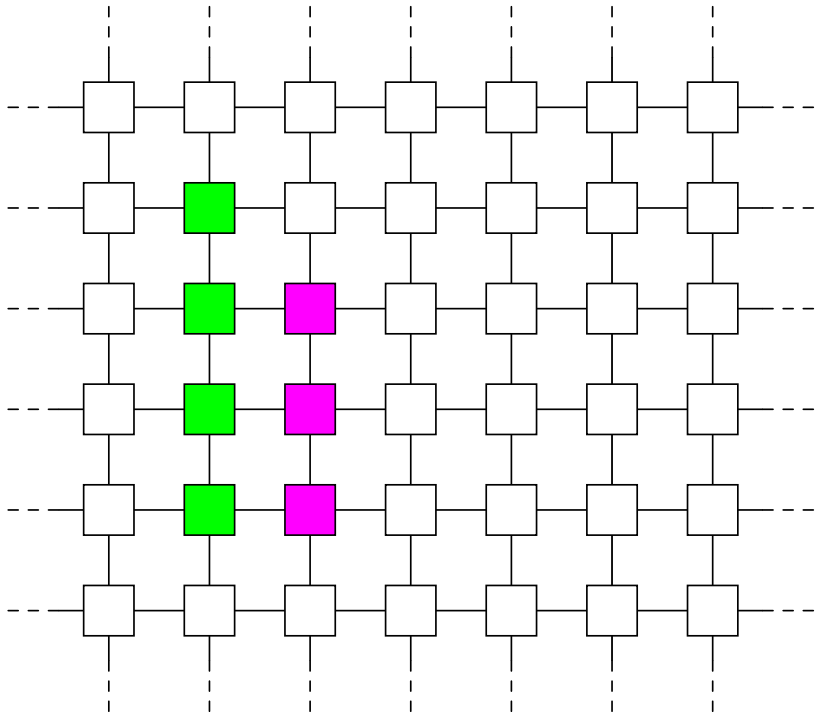
Cellular automata, e.g. Conway's Game of Life



- an infinite number of automata (in 1D or 2D)
- next-neighbour communication

Meanwhile, von Neumann had a better idea

Cellular automata, e.g. Conway's Game of Life



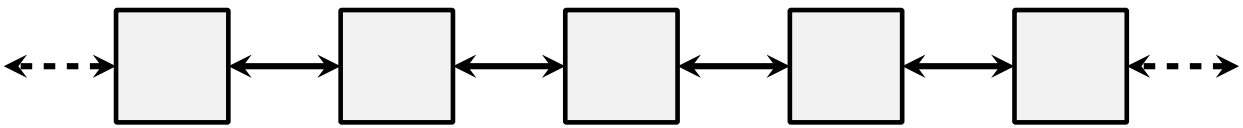
- an infinite number of automata (in 1D or 2D)
- next-neighbour communication

Field-programmable gate arrays (FPGA)

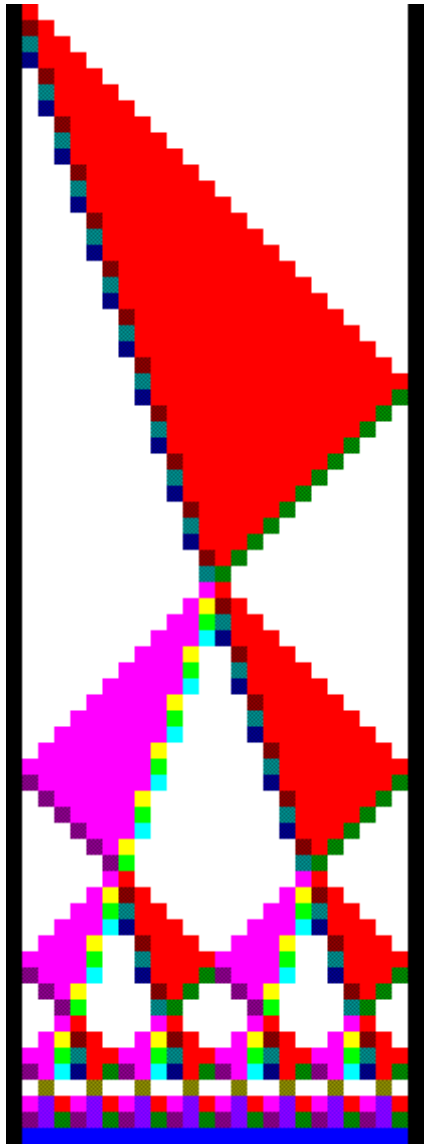
are to cellular automata what your PC is to a Turing machine.

Next neighbour communications suck

Example: the "firing squad problem": how to synchronize n cells?

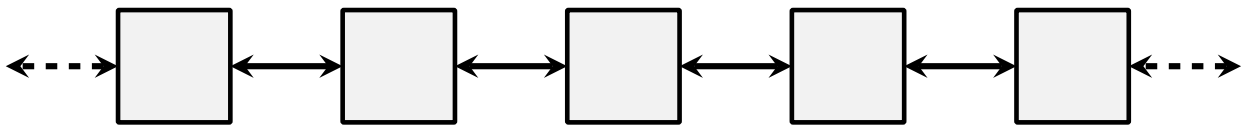


- right: $3n$ steps, using 15 states
- (best known: $2n - 2$ steps, 6 states)



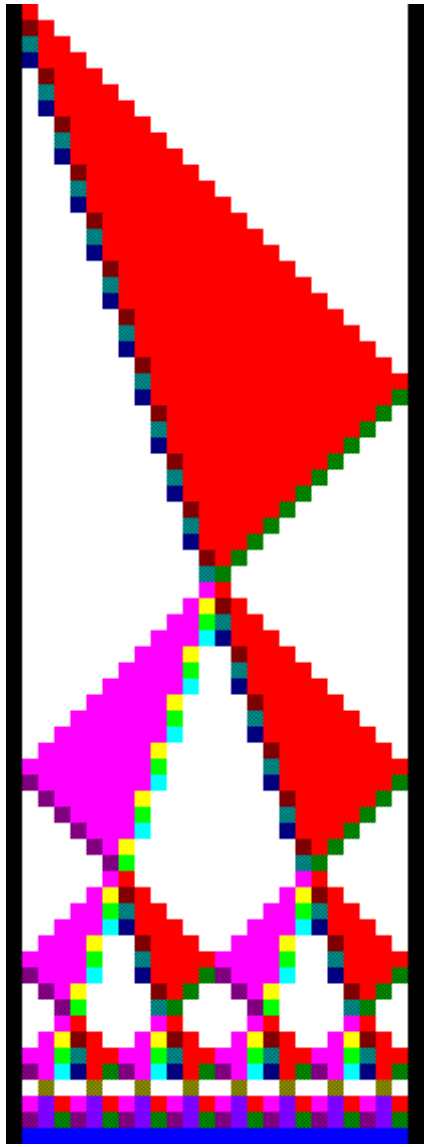
Next neighbour communications suck

Example: the "firing squad problem":
how to synchronize n cells?



- right: $3n$ steps, using 15 states
- (best known: $2n - 2$ steps, 6 states)

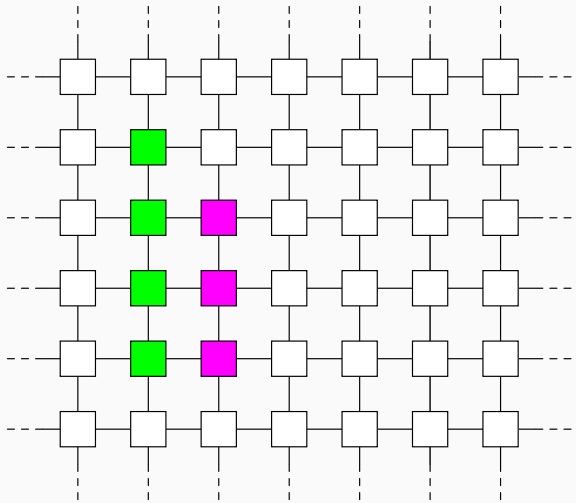
... in real life the captain simply shouts "Fire!"



FPGA architectures

Basic FPGA structure

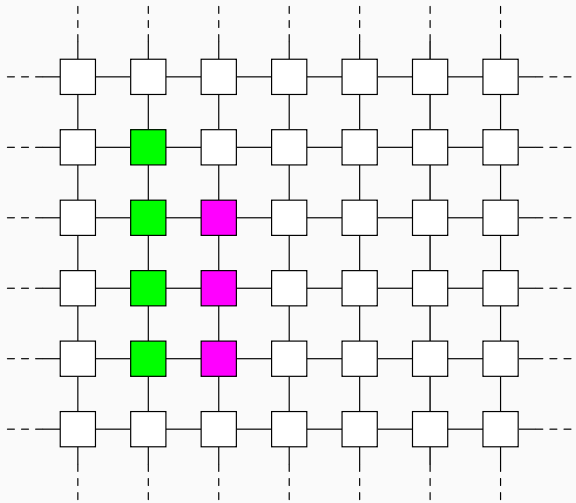
Overview



- A grid of configurable cells

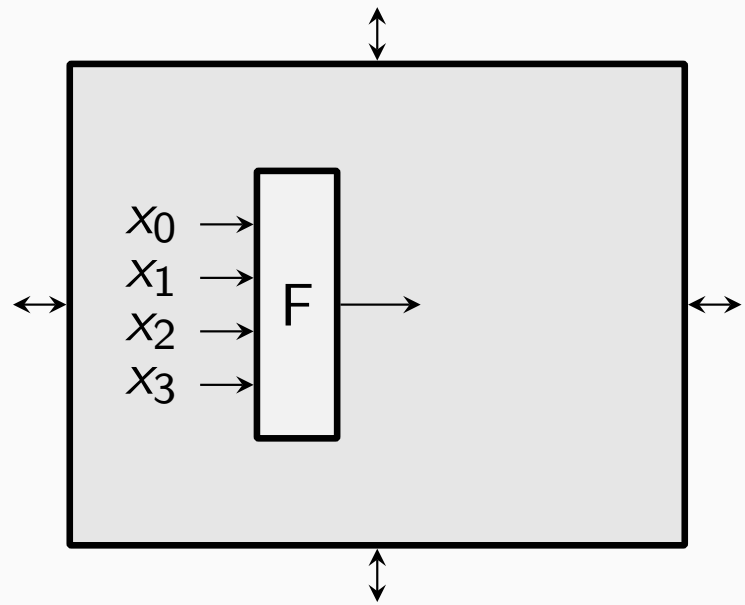
Basic FPGA structure

Overview



- A grid of configurable cells
  - ... to build arbitrary logic

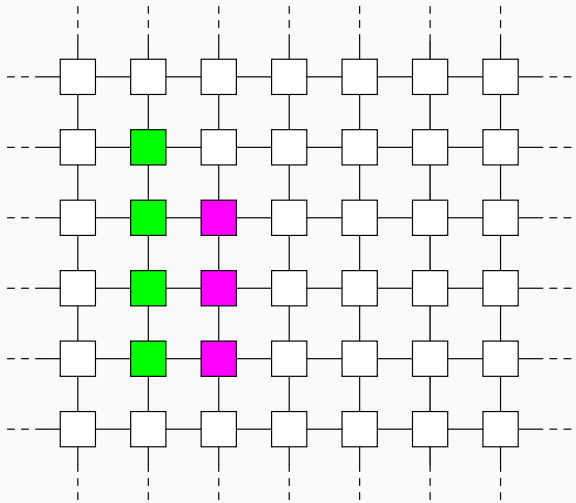
Inside a cell



- A Look-Up Table (LUT) F
  - 4 inputs, one output
  - holds any **truth table**

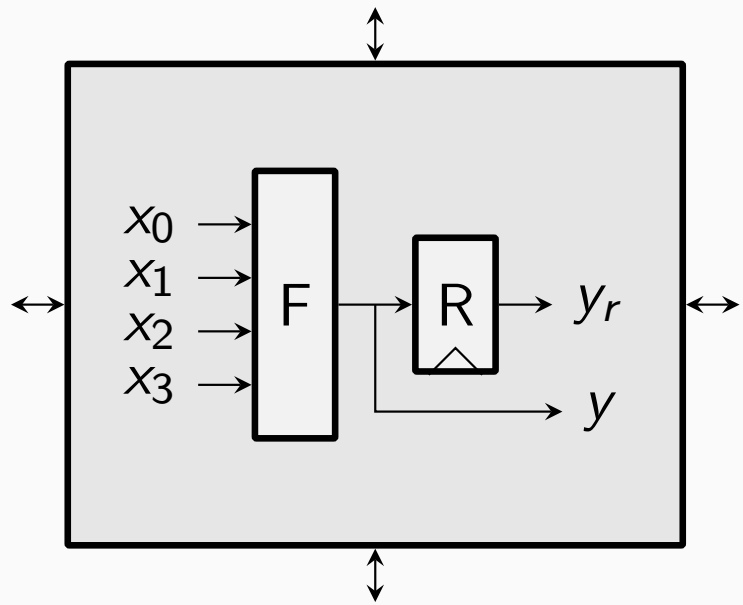
Basic FPGA structure

Overview



- A grid of configurable cells
  - ... to build arbitrary logic
  - ... and sequential circuits

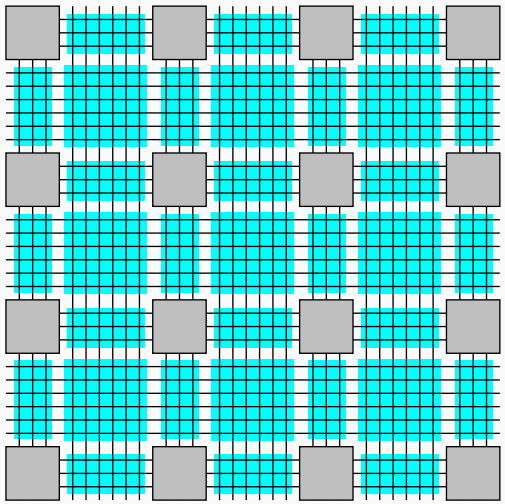
Inside a cell



- A Look-Up Table (LUT) F
  - 4 inputs, one output
  - holds any **truth table**
- 1 bit of run-time memory R

Basic FPGA structure

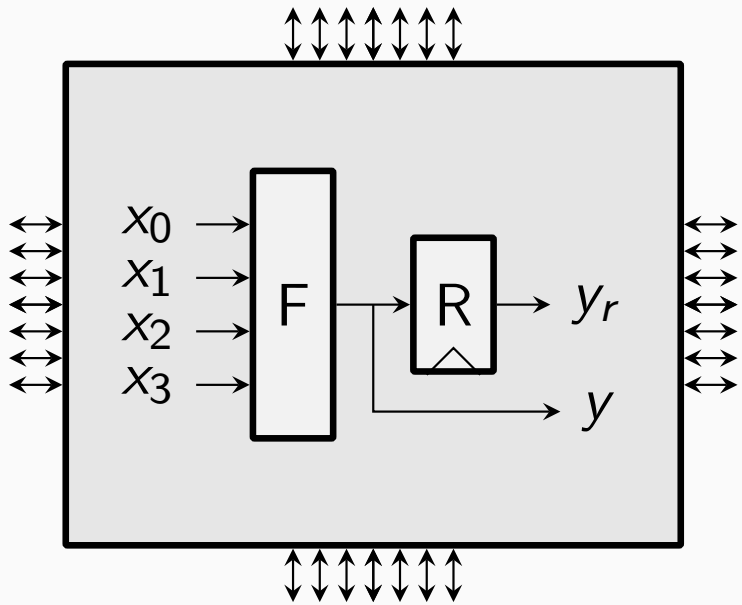
Overview



- A grid of configurable cells
  - ... to build arbitrary logic
  - ... and sequential circuits
- Configurable wiring
  - routing channels
  - switch boxes

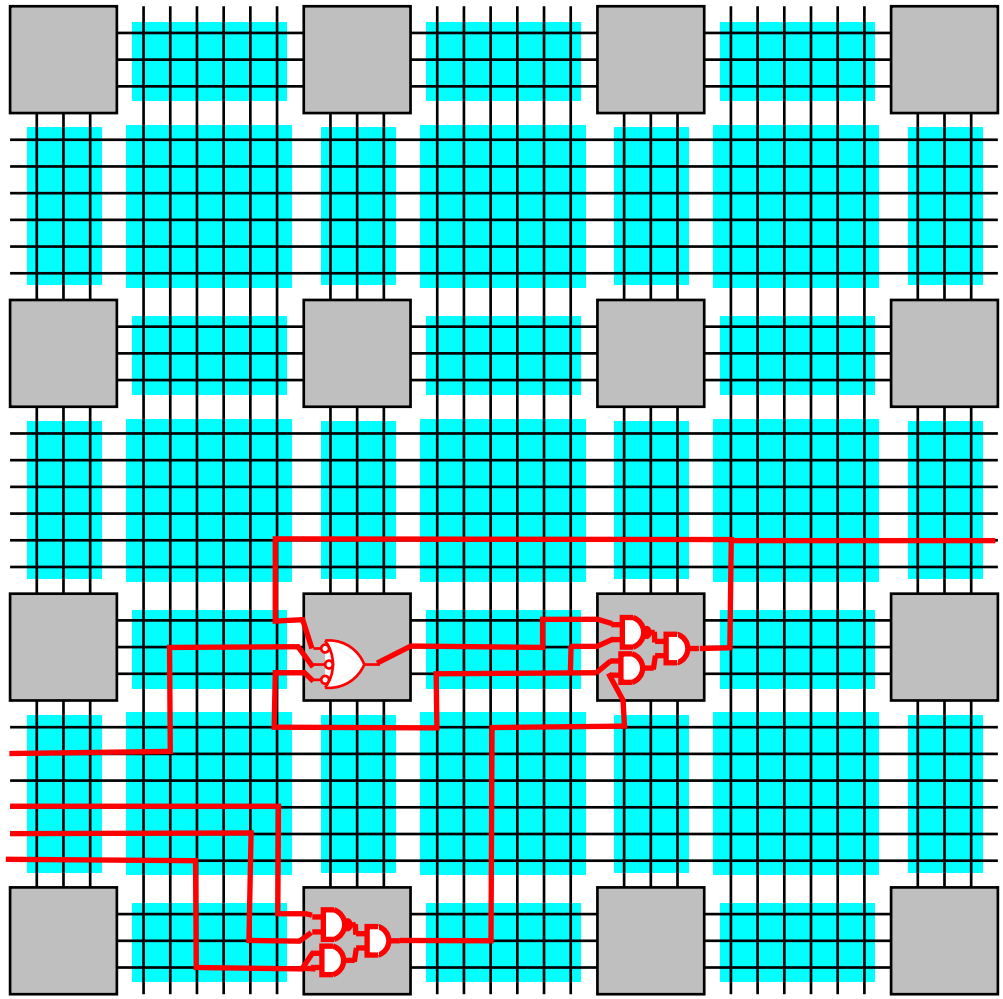
→ **random access** to distant cells

Inside a cell



- A Look-Up Table (LUT) F
  - 4 inputs, one output
  - holds any **truth table**
- 1 bit of run-time memory R

A configured FPGA



Also known as **reconfigurable circuits**

used for **reconfigurable computing**

Two moments in the life of an FPGA

Configuration time (1-1000 ms)

- the LUTs are filled with truth tables
- the state (on/off) of each switch in each switch box is defined

a program == a lot of configuration bits

Two moments in the life of an FPGA

Configuration time (1-1000 ms)

- the LUTs are filled with truth tables
- the state (on/off) of each switch in each switch box is defined

a program == a lot of configuration bits

Run time (forever if needed)

- Data is processed by each LUT according to its truth table
- Data moves from LUT to LUT along the (static) connexions
- The FPGA behaves as a circuit of gates

Two moments in the life of an FPGA

Configuration time (1-1000 ms)

- the LUTs are filled with truth tables
- the state (on/off) of each switch in each switch box is defined

a program == a lot of configuration bits

Run time (forever if needed)

- Data is processed by each LUT according to its truth table
- Data moves from LUT to LUT along the (static) connexions
- The FPGA behaves as a circuit of gates

The programming model of FPGAs is the digital circuit.

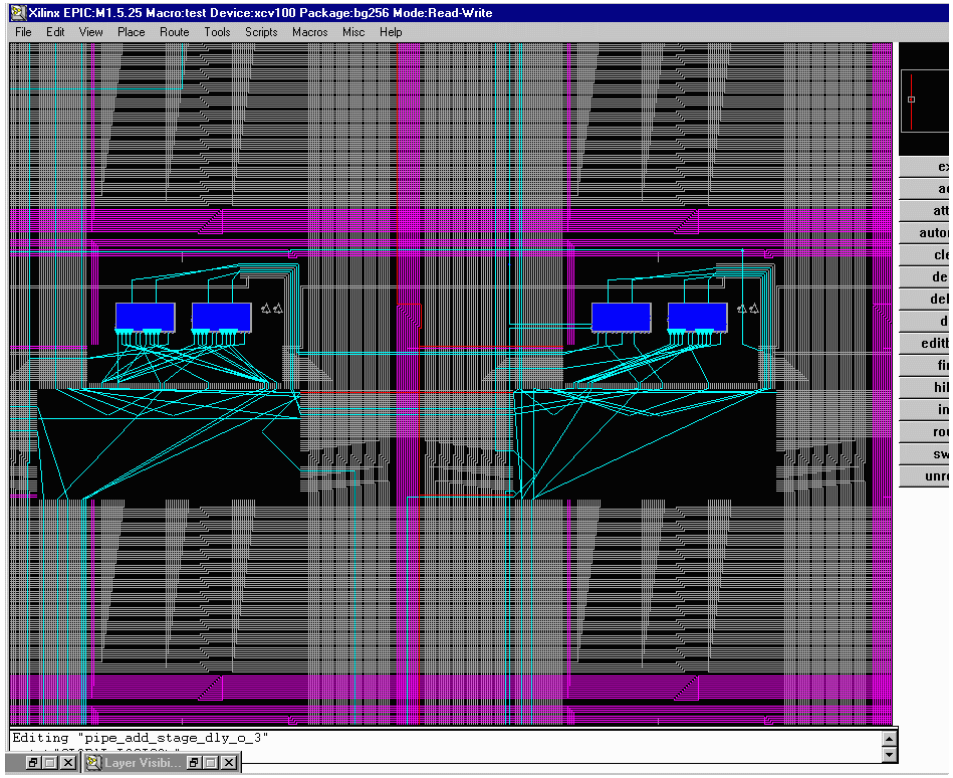
You don't program an FPGA, you configure it (as a circuit).

Performance: It's the routing, stupid

Most FPGA silicon is dedicated to programmable routing.

"Customers buy logic,
but they pay for routing"
(Langhammer, Intel)

a picture from 1999 $\longrightarrow$
(it got much worse since then)

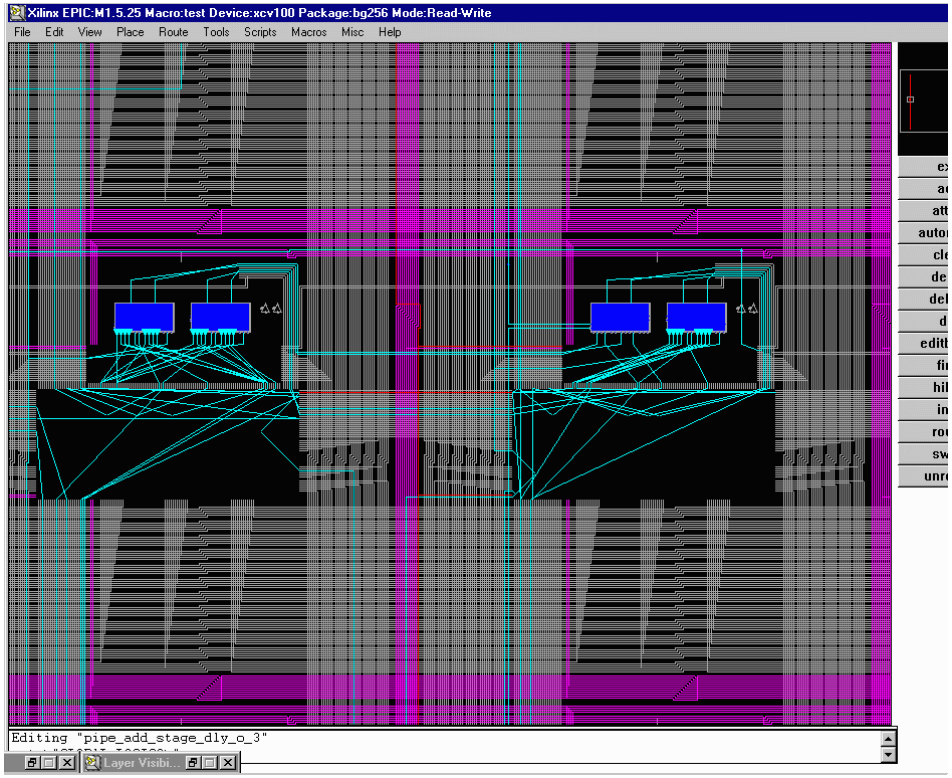


Performance: It's the routing, stupid

Most FPGA silicon is dedicated to programmable routing.

"Customers buy logic,
but they pay for routing"
(Langhammer, Intel)

a picture from 1999 $\longrightarrow$
(it got much worse since then)



- A circuit that would fit in 1 mm^2 of ASIC silicon will only fit in a 50mm^2 FPGA...
- ... and the configured FPGA will run at 1/10th the frequency
  - there are transistors on all the wires!

The FPGA costs 50€, where a 1mm^2 circuit would cost 20,000€...

The curse of Los Angeles

... or, the fatality of traffic jams in an expanding city.

The curse of Los Angeles

... or, the fatality of traffic jams in an expanding city.

More and more space between the buildings is dedicated to roads and parking.

The curse of Los Angeles

... or, the fatality of traffic jams in an expanding city.

More and more space between the buildings is dedicated to roads and parking. How does that scale?

The curse of Los Angeles

... or, the fatality of traffic jams in an expanding city.

More and more space between the buildings is dedicated to roads and parking. How does that scale?
Los Angeles is a proof that the answer is: **very badly**:

2/3 of the area is dedicated to cars (roads + parking lots)

The curse of Los Angeles

... or, the fatality of traffic jams in an expanding city.

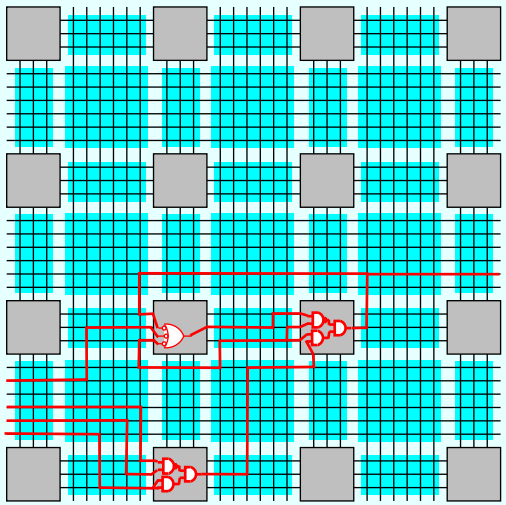
More and more space between the buildings is dedicated to roads and parking. How does that scale?
Los Angeles is a proof that the answer is: **very badly**:

2/3 of the area is dedicated to cars (roads + parking lots)

The circuit variant of this curse is called Rent's law.

Rent's law?

Yet another experimental law



In a circuit of diameter n,
the number of wires crossing a diameter
is proportional to n^r with $1 < r < 2$.

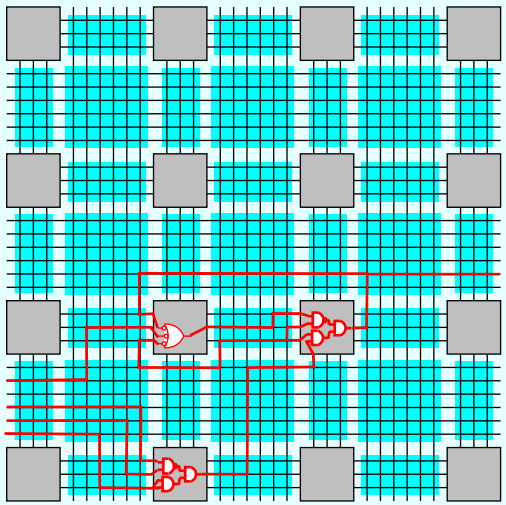
- more than proportional to n, the diameter,
- note quite proportional to the area n^2 of each half-circuit.

The value of r (Rent's exponent) depends of the class of circuit.

FPGAs are designed for worst-case circuits, hence r close to 2...

Rent's law?

Yet another experimental law



In a circuit of diameter n,
the number of wires crossing a diameter
is proportional to n^r with $1 < r < 2$.

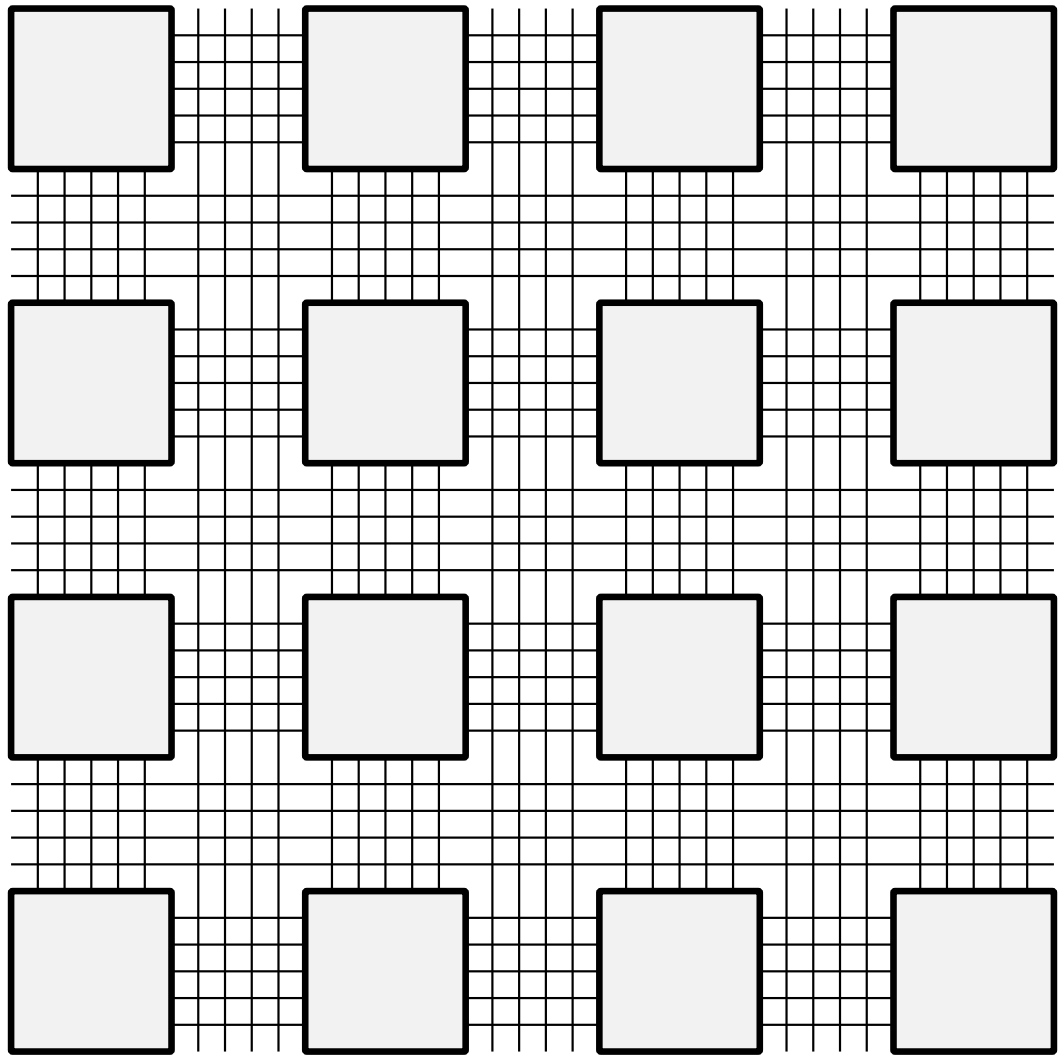
- more than proportional to n, the diameter,
- note quite proportional to the area n^2 of each half-circuit.

The value of r (Rent's exponent) depends of the class of circuit.

FPGAs are designed for worst-case circuits, hence r close to 2...

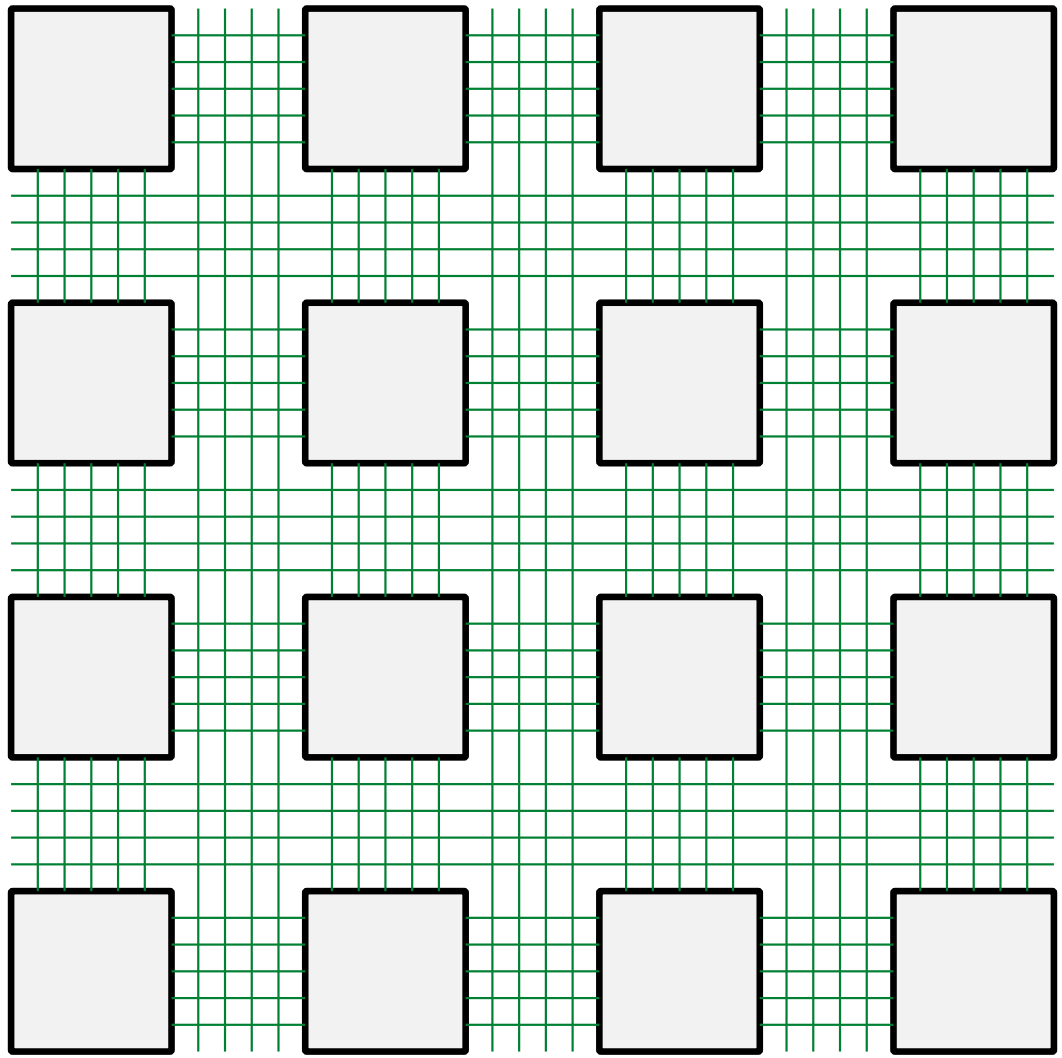
Replace "circuit" with "city", and "wires" with "citizen commuters", and you have the explanation of the Hopeless Universal Trafic Jam in expanding cities.

How many wires per routing channel?



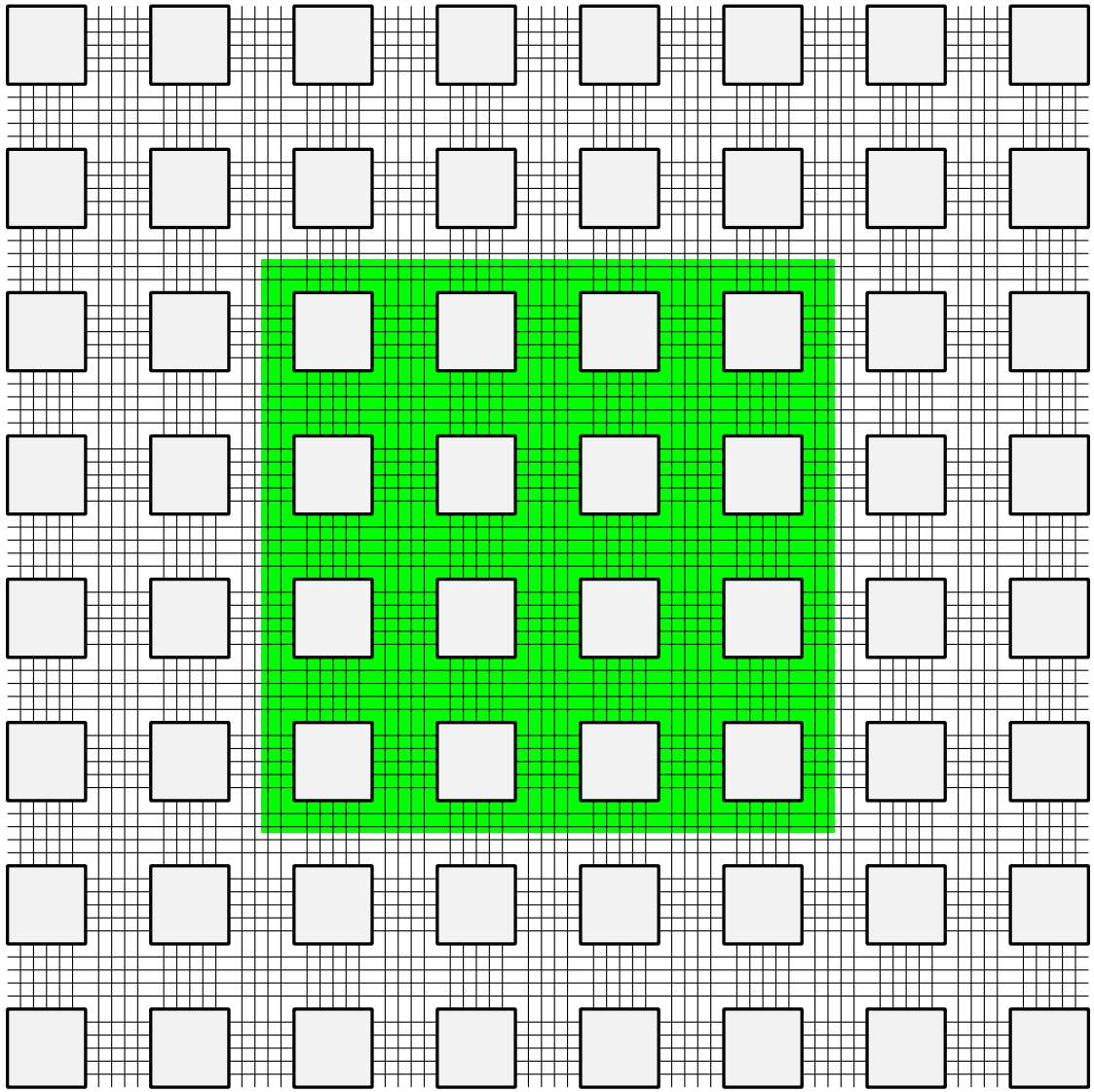
1990

How many wires per routing channel?



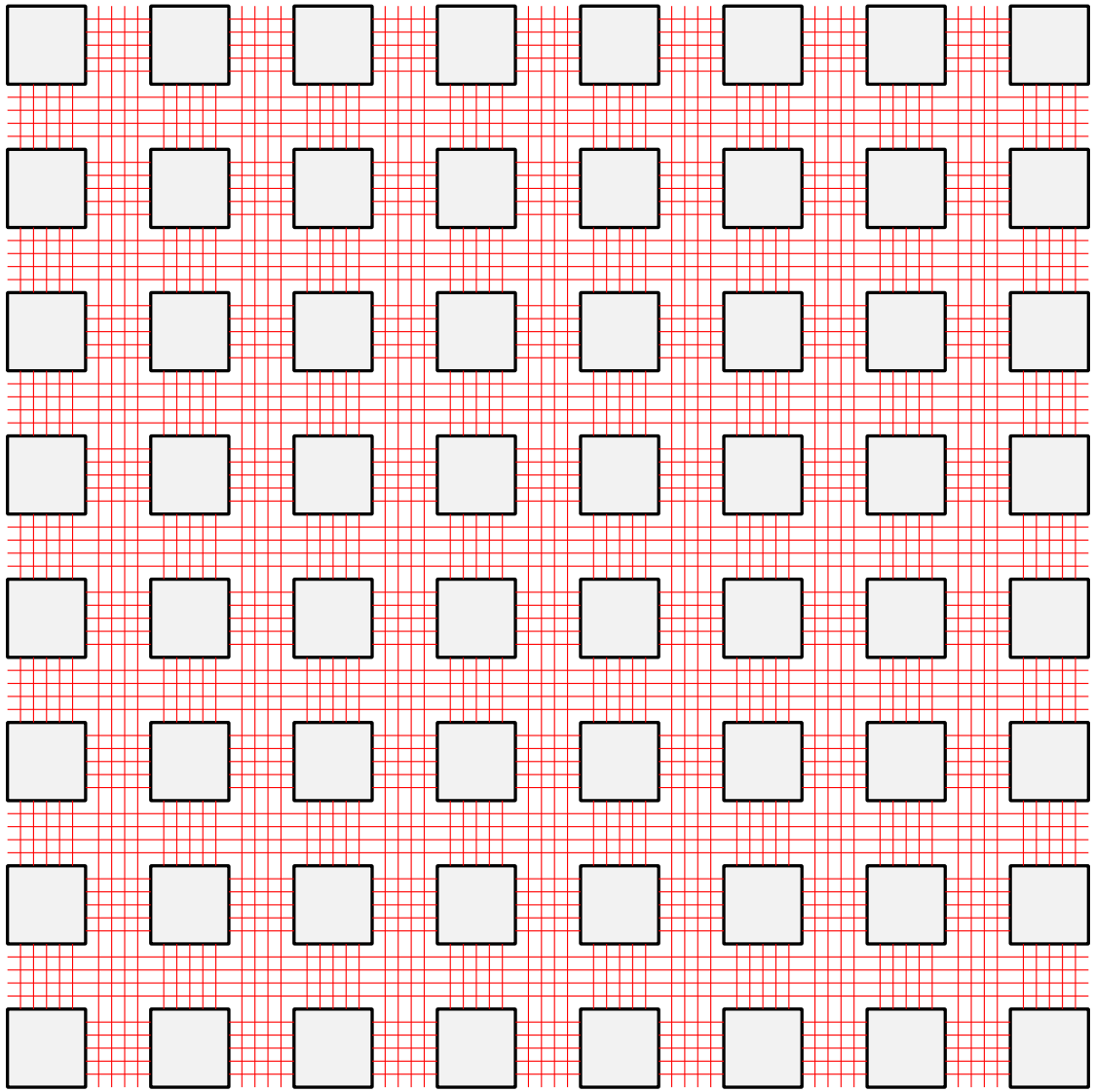
1990

How many wires per routing channel?



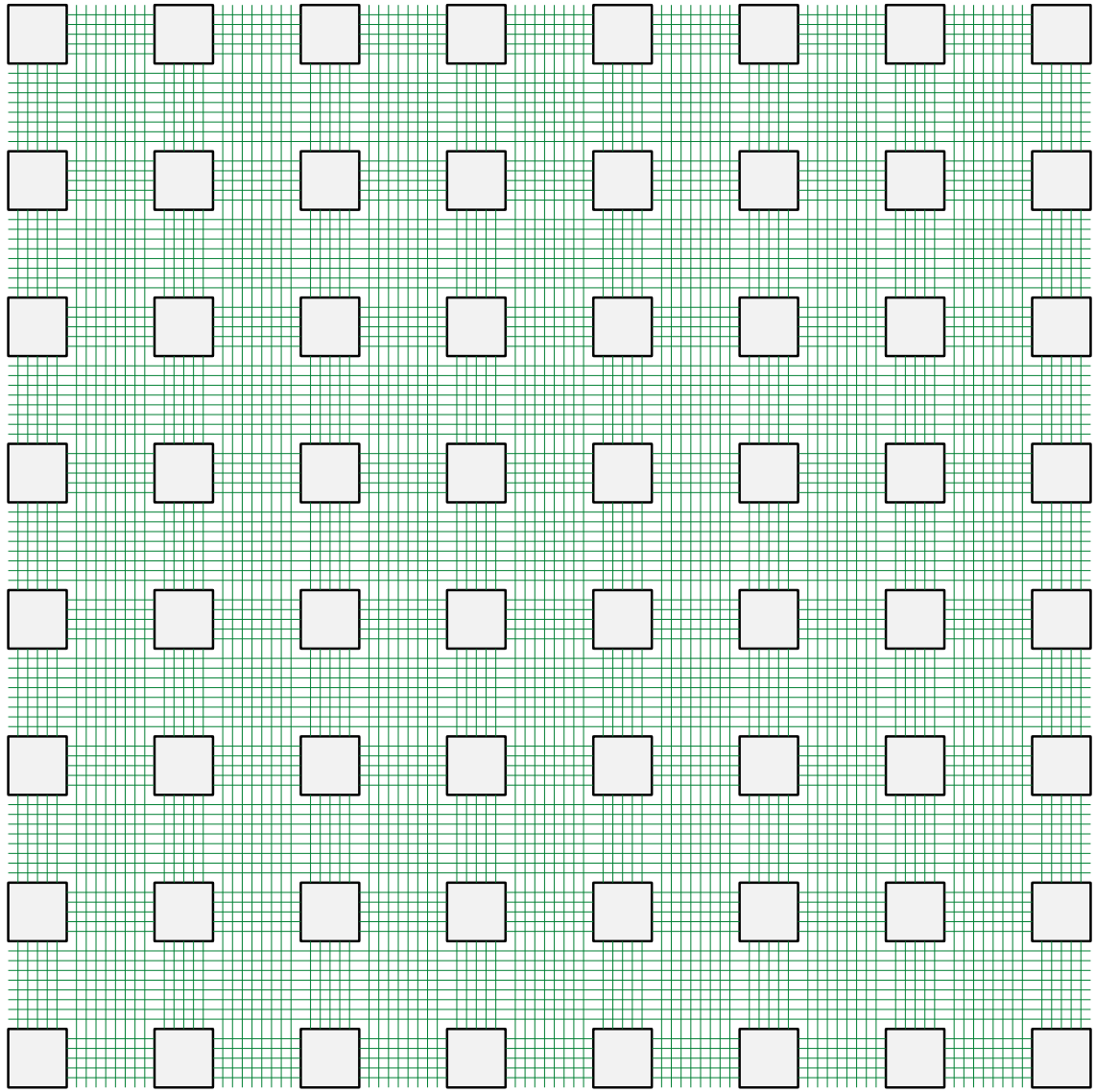
1993 ?

How many wires per routing channel?



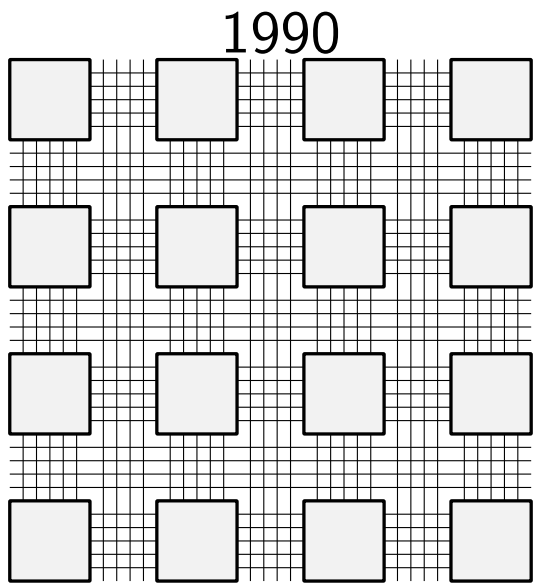
1993 ?

How many wires per routing channel?



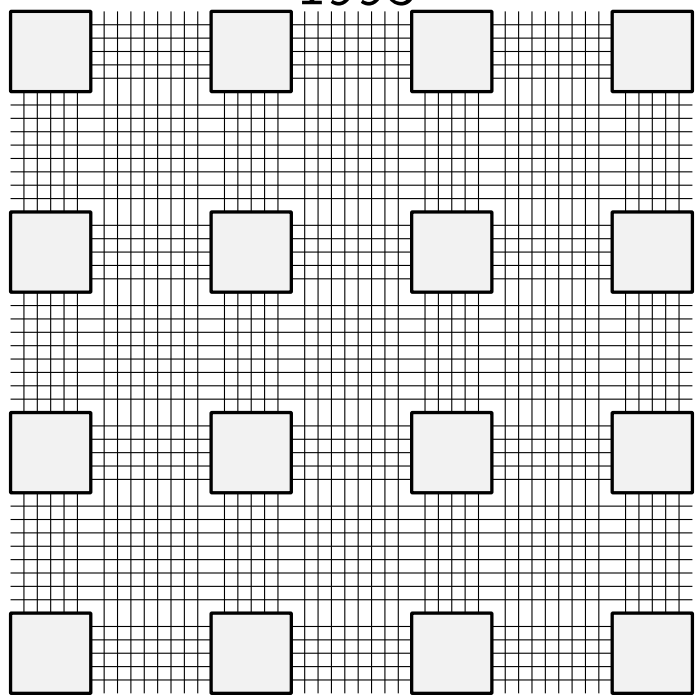
1994

How many wires per routing channel?



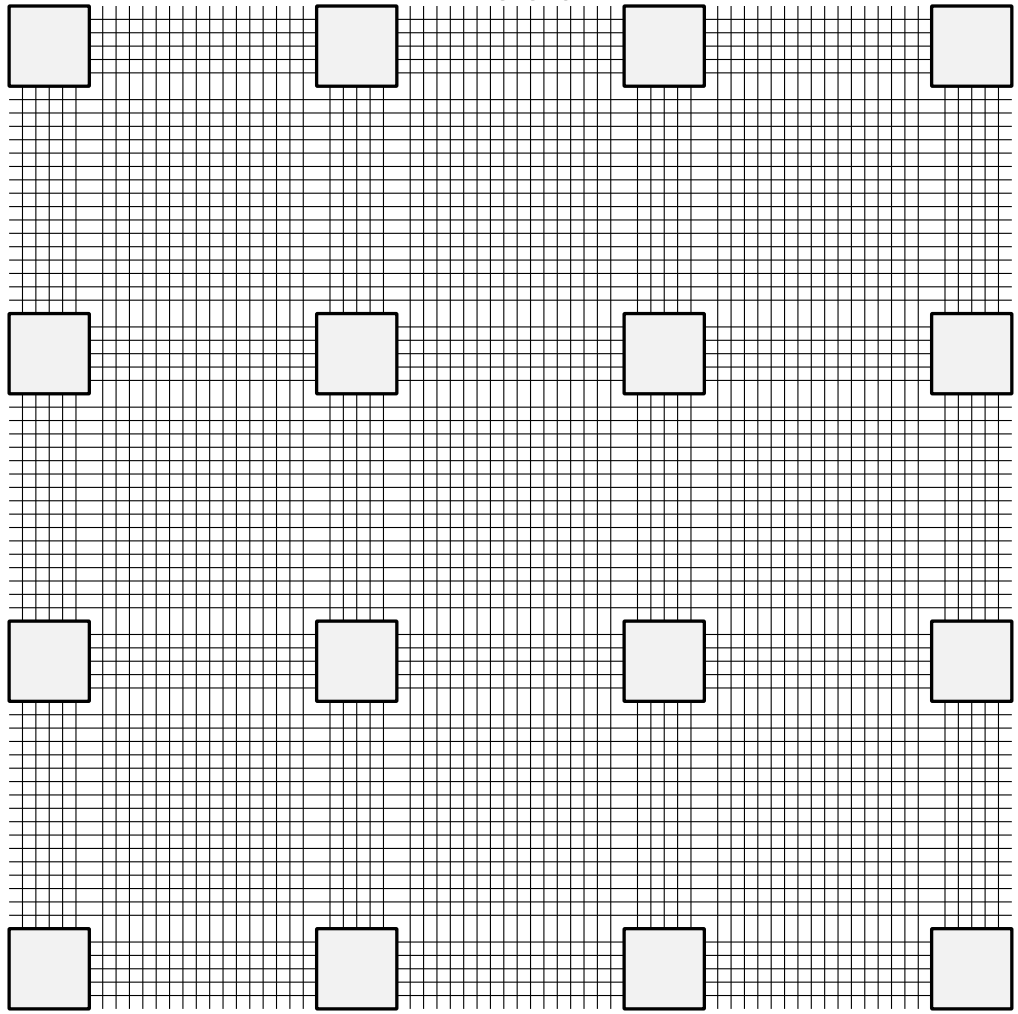
How many wires per routing channel?

1993

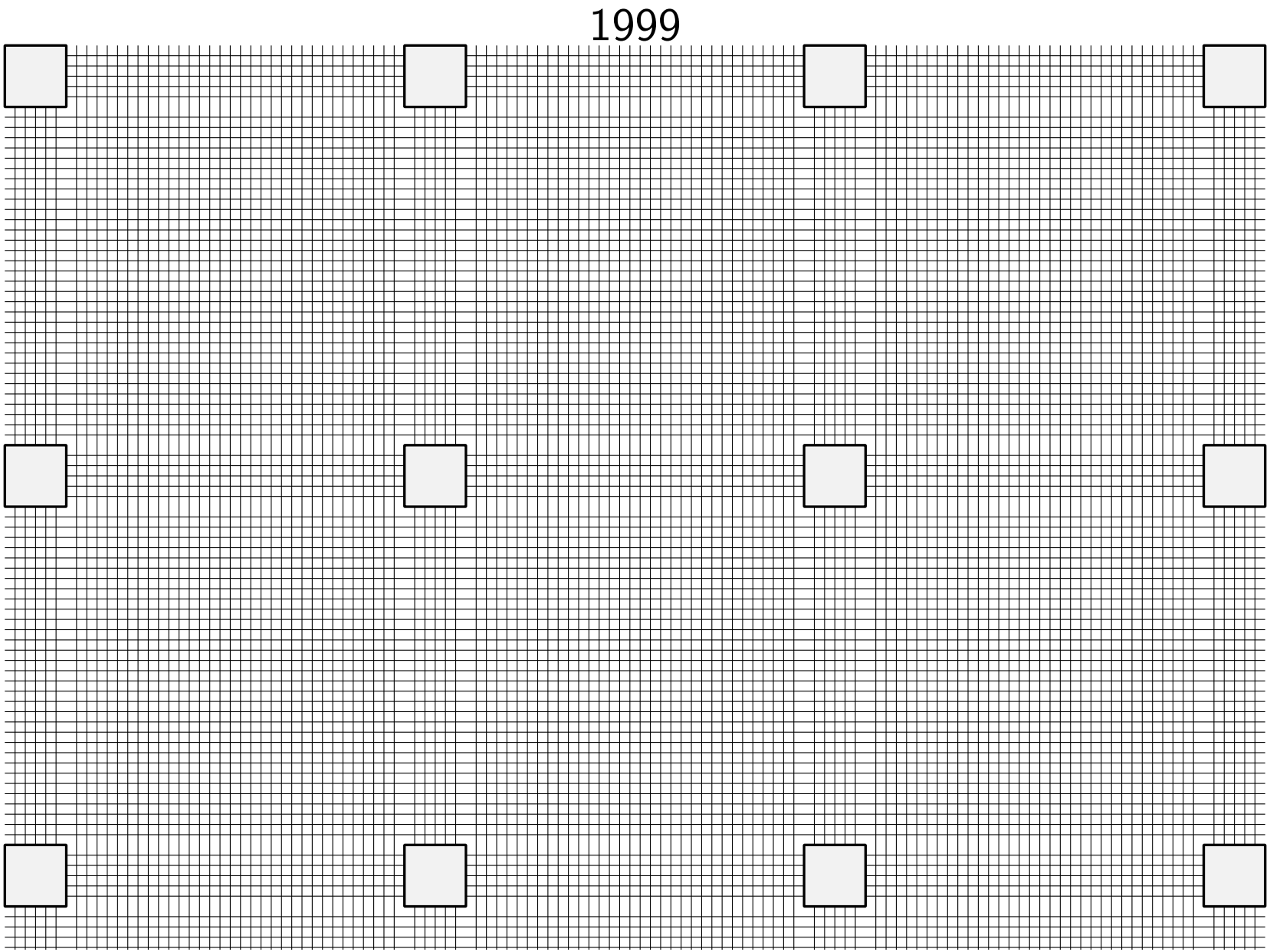


How many wires per routing channel?

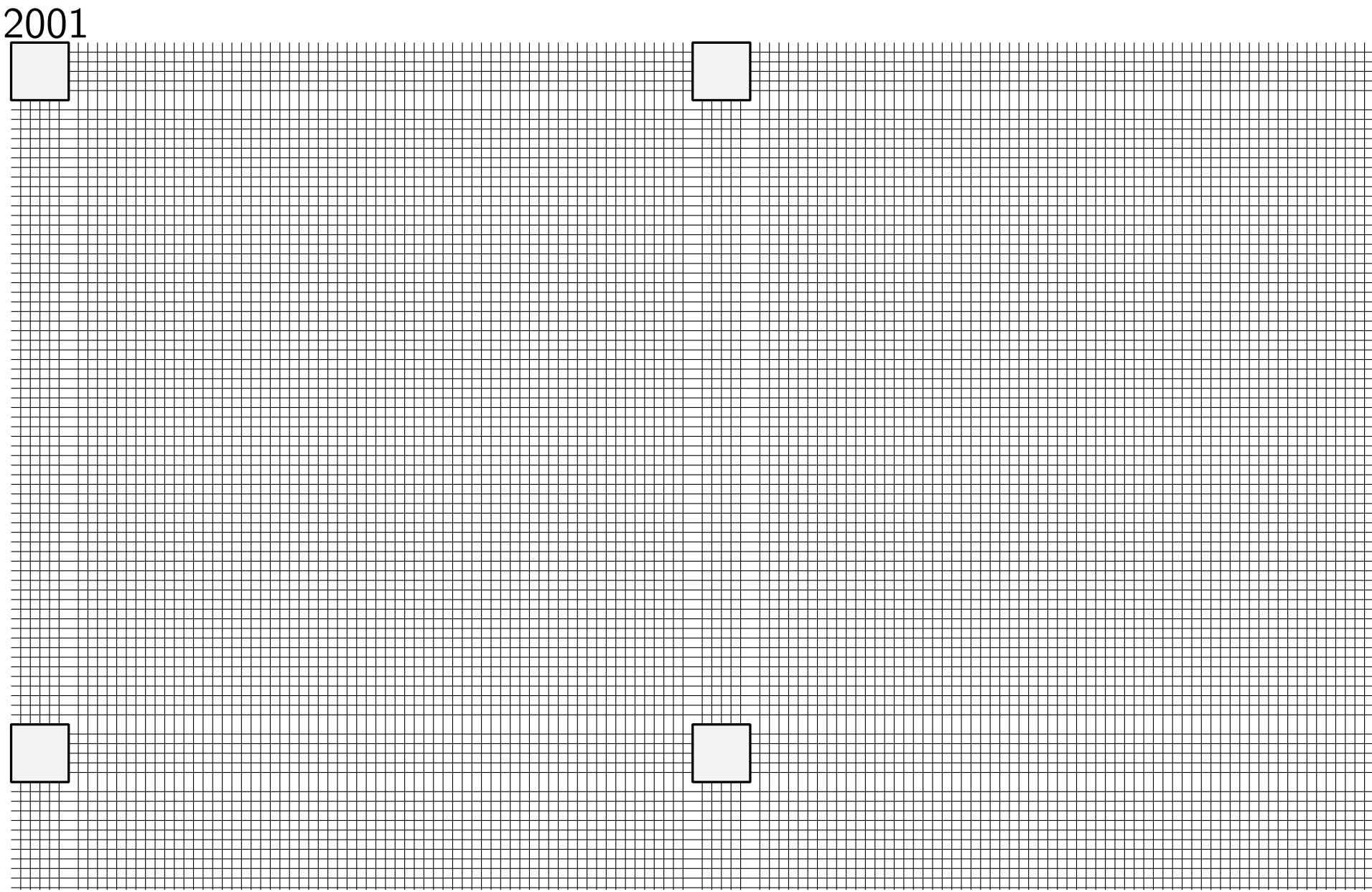
1996



How many wires per routing channel?



How many wires per routing channel?



Solutions

Can we save Los Angeles?

- Build highways of various widths
- Build busses, underground, light rail
- Relocalize the economy
- Use bicycles instead of SUVs

Solutions

Can we save Los Angeles?

- Build highways of various widths
- Build busses, underground, light rail
- Relocalize the economy
- Use bicycles instead of SUVs

Transposed to FPGA:

- heterogeneous routing
- increase compute granularity
- relocalize computations
- compute just right

Solutions

Can we save Los Angeles?

- Build highways of various widths
- Build busses, underground, light rail
- Relocalize the economy
- Use bicycles instead of SUVs

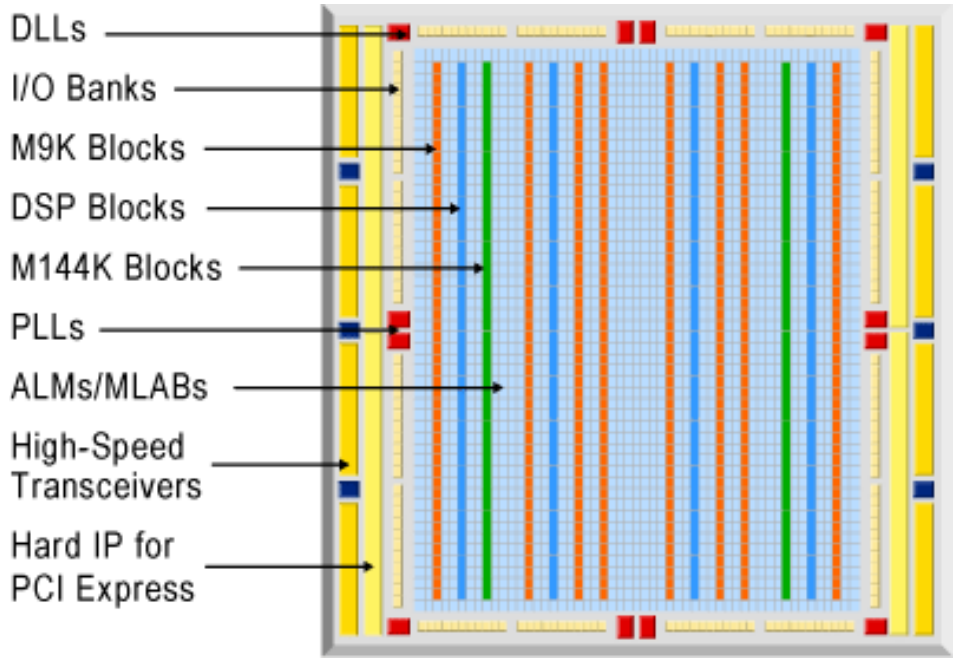
Transposed to FPGA:

- heterogeneous routing
- increase compute granularity
- relocalize computations
- compute just right

Oh, and by the way, did you get the message that *complexity* tools can be applied to circuits?

It was too simple so far, people would complain

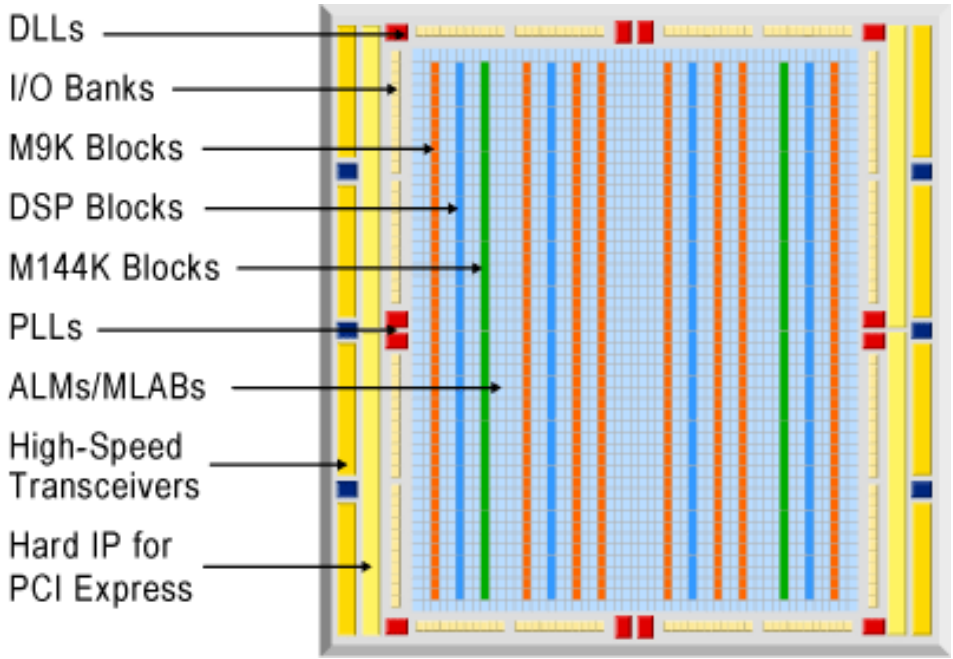
- coarser cells,
  optimized for additions
- many independent clock networks with their PLLs
- many small ($\approx$ 24 bit) hard multipliers ("DSP blocks")
- many small ($\approx$ 10 kBit) memories



the Altera/Intel stratix IV FPGA

It was too simple so far, people would complain

- coarser cells,
  optimized for additions
- many independent clock networks with their PLLs
- many small ($\approx$ 24 bit) hard multipliers ("DSP blocks")
- many small ($\approx$ 10 kBit) memories



the Altera/Intel stratix IV FPGA

Numbers for our cheap-ish ($\approx$ 50€) Xilinx Zynq 7010

2 ARM processor cores @ 667MHz
+ **28k** logic cells + **80** DSP blocks + **60** 36kBit memory block ...
running at **200 MHz**

FPGAs in 2019

- Lots of (coarse) configurable cells
  - tens of millions of equivalent logic gates

FPGAs in 2019

- Lots of (coarse) configurable cells
  - tens of millions of equivalent logic gates
- up to several tens of Mb of embedded memories
  - in small blocks of a few kbits
  - configurable in all sorts of ways
  - providing massive intra-FPGA bandwidth

FPGAs in 2019

- Lots of (coarse) configurable cells
  - tens of millions of equivalent logic gates
- up to several tens of Mb of embedded memories
  - in small blocks of a few kbits
  - configurable in all sorts of ways
  - providing massive intra-FPGA bandwidth
- up to several thousand DSP blocks
  - Multiply-Accumulate unit (typically 18x18+40 bits, **integer**)
  - Also sometimes flexible (may be split into 9x9 multipliers, etc)
  - Capable of 32-bit floating point on Intel

FPGAs in 2019

- Lots of (coarse) configurable cells
  - tens of millions of equivalent logic gates
- up to several tens of Mb of embedded memories
  - in small blocks of a few kbits
  - configurable in all sorts of ways
  - providing massive intra-FPGA bandwidth
- up to several thousand DSP blocks
  - Multiply-Accumulate unit (typically 18x18+40 bits, **integer**)
  - Also sometimes flexible (may be split into 9x9 multipliers, etc)
  - Capable of 32-bit floating point on Intel
- matching clock capabilities (hundreds of clock domains etc)

FPGAs in 2019

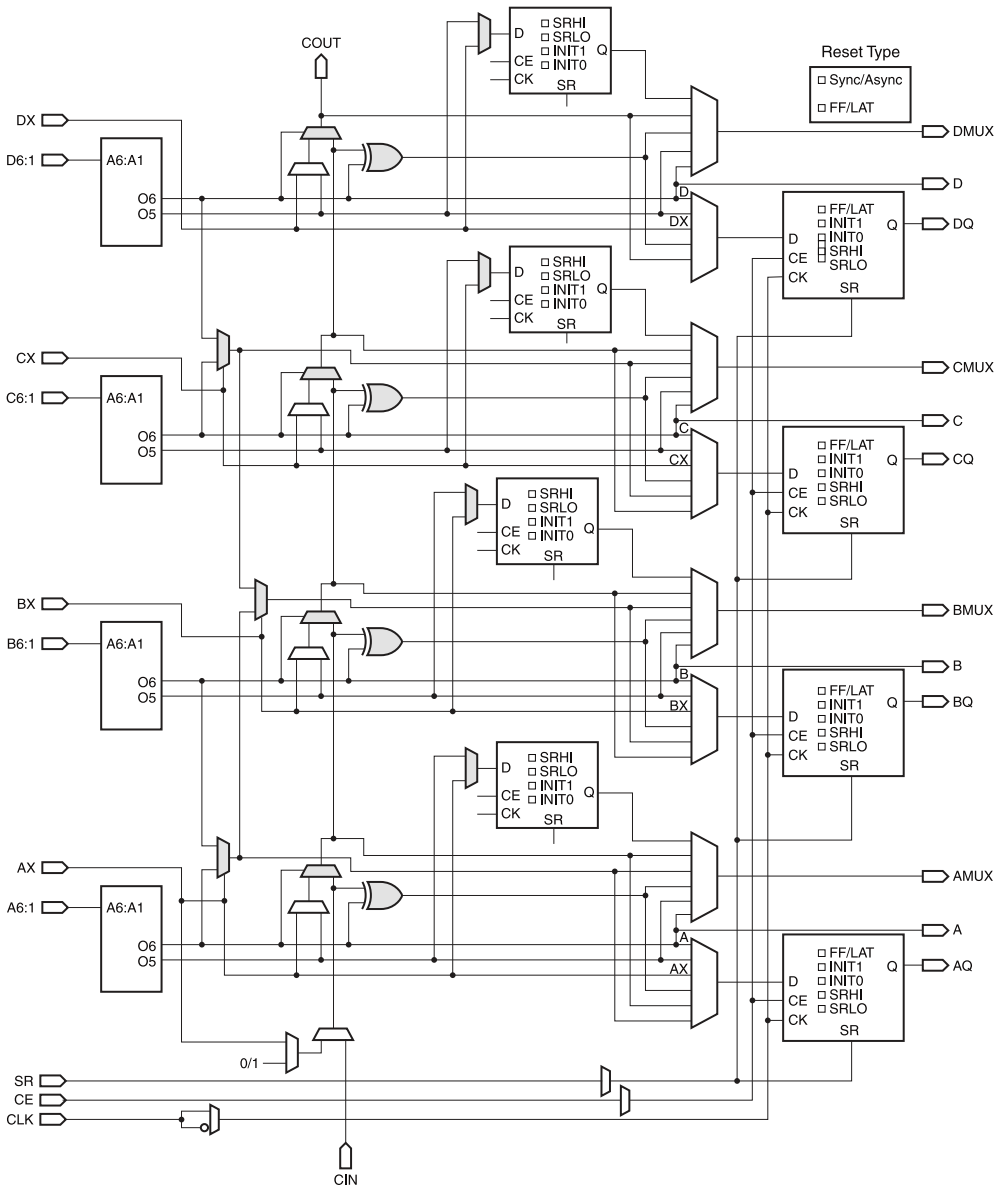
- Lots of (coarse) configurable cells
  - tens of millions of equivalent logic gates
- up to several tens of Mb of embedded memories
  - in small blocks of a few kbits
  - configurable in all sorts of ways
  - providing massive intra-FPGA bandwidth
- up to several thousand DSP blocks
  - Multiply-Accumulate unit (typically 18x18+40 bits, **integer**)
  - Also sometimes flexible (may be split into 9x9 multipliers, etc)
  - Capable of 32-bit floating point on Intel
- matching clock capabilities (hundreds of clock domains etc)
- matching I/O capabilities
  - more than 1000 pins on large FPGAs
  - High-speed configurable I/Os: 1500 pages of documentation

FPGAs in 2019

- Lots of (coarse) configurable cells
  - tens of millions of equivalent logic gates
- up to several tens of Mb of embedded memories
  - in small blocks of a few kbits
  - configurable in all sorts of ways
  - providing massive intra-FPGA bandwidth
- up to several thousand DSP blocks
  - Multiply-Accumulate unit (typically 18x18+40 bits, **integer**)
  - Also sometimes flexible (may be split into 9x9 multipliers, etc)
  - Capable of 32-bit floating point on Intel
- matching clock capabilities (hundreds of clock domains etc)
- matching I/O capabilities
  - more than 1000 pins on large FPGAs
  - High-speed configurable I/Os: 1500 pages of documentation
- Embedded ARM multi-core processors
  - it is unclear what is embedded in what

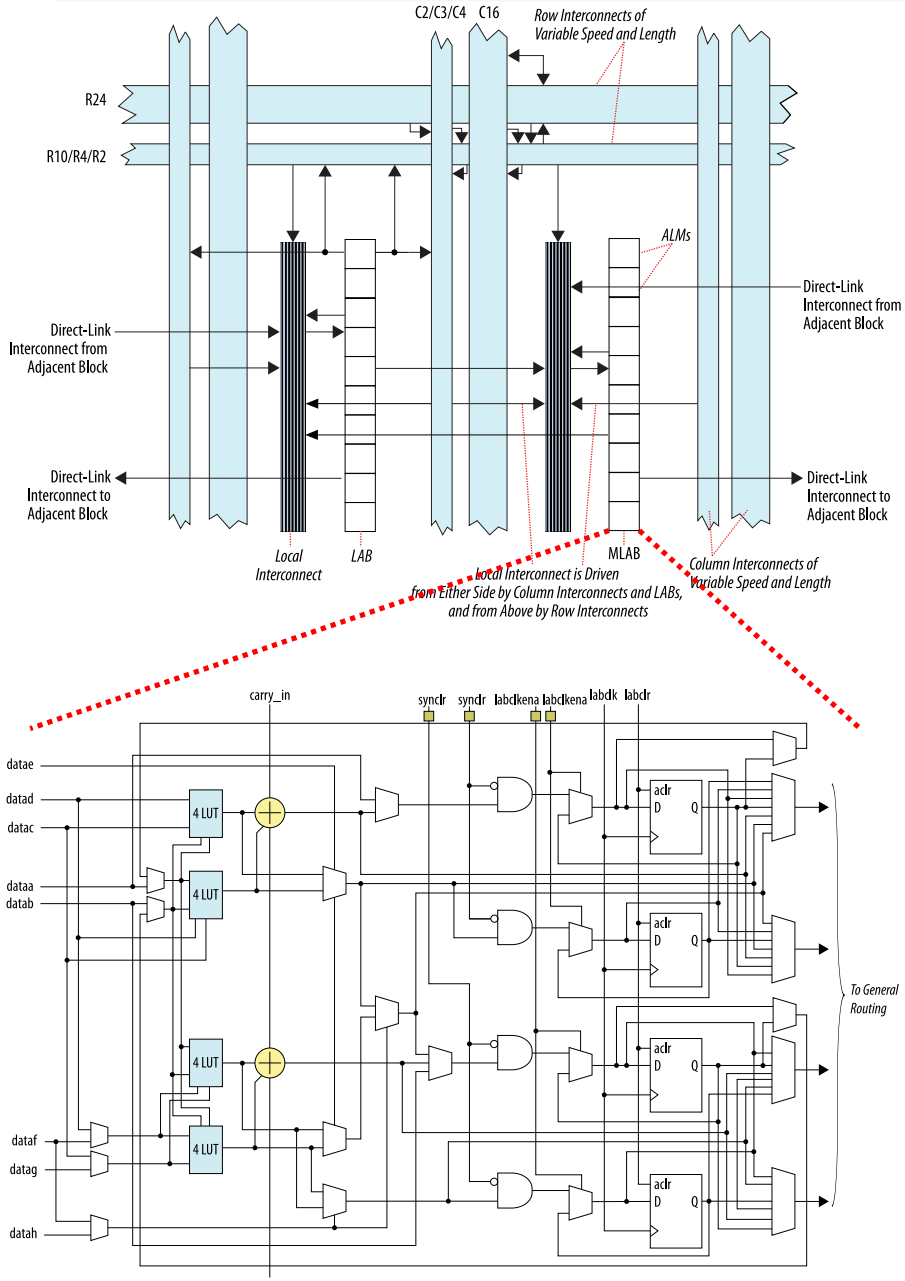
The real (Xilinx) Configurable Logic Block

A "slice" (cut from Xilinx Virtex7 doc):



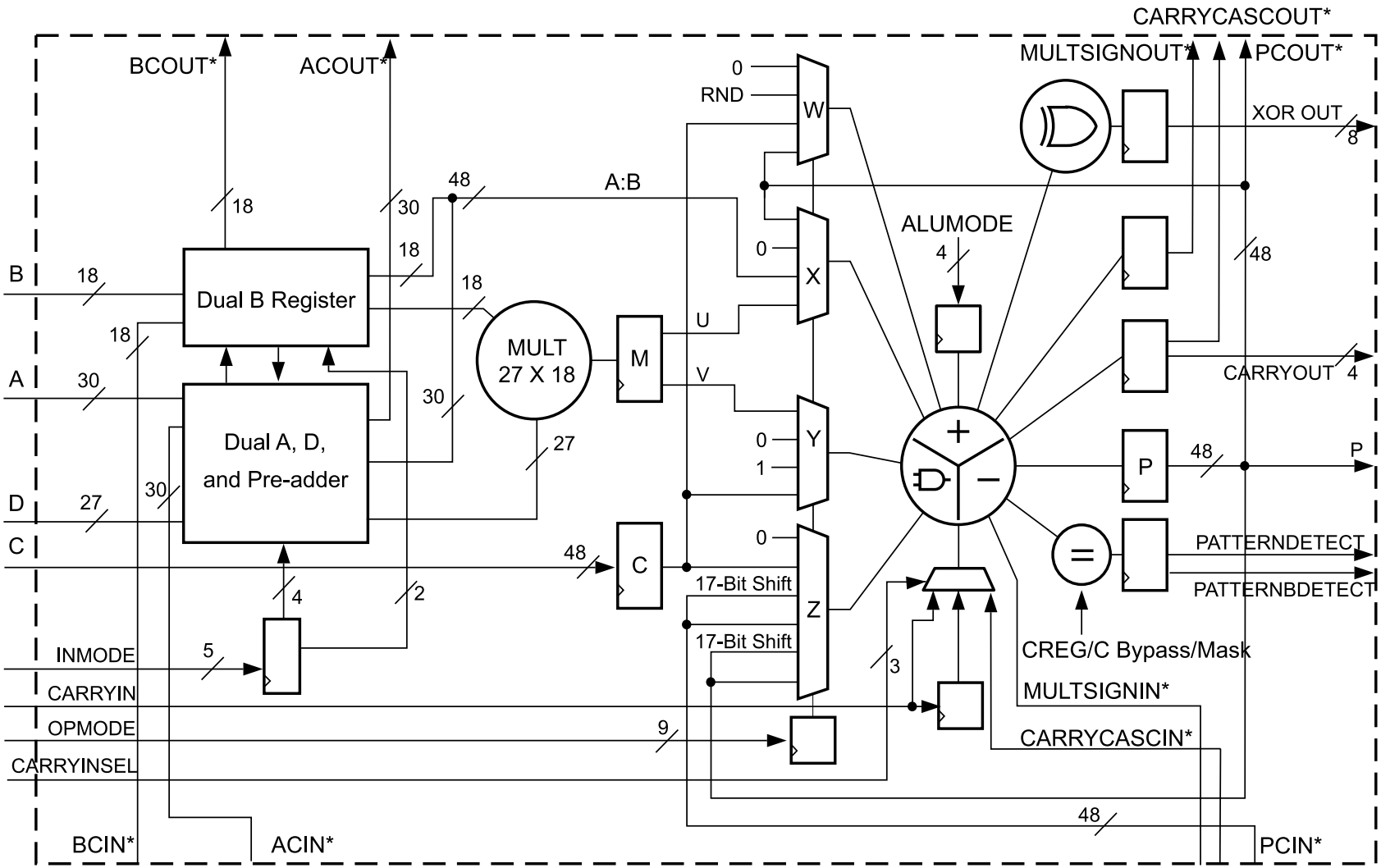
- Granularity increasing
  - 6-input LUTs
    (and counting)
  - 4 LUT/slice
    (and counting)
  - 2 slice/CLB
  - Ratio reg/LUT
    still equal to 1
- All this keeps routing local
- Support of frequent ops
  - addition: **carry logic**
    (skips the slow routing)
  - shift registers: SRL

The real (Intel) Logic Array Block



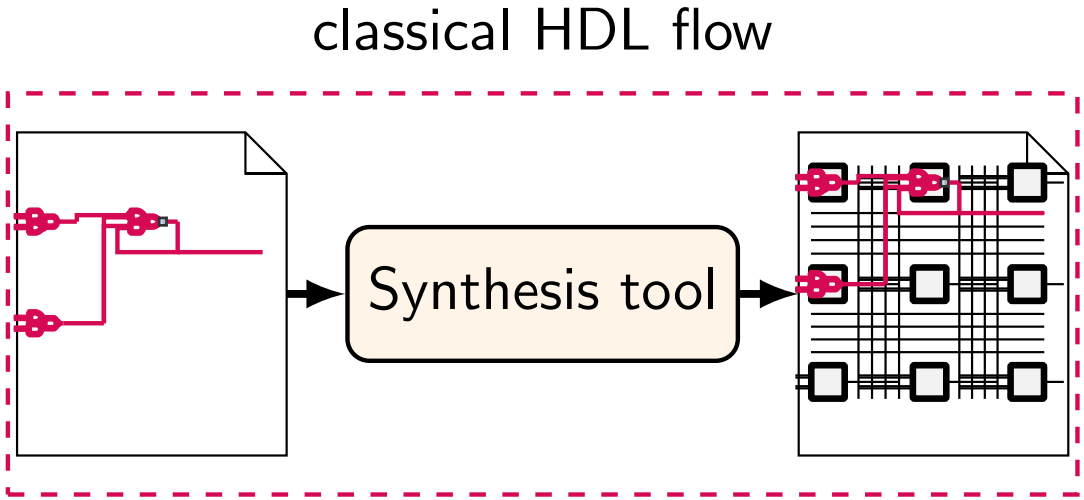
- 4 input/LUT
- 4 LUT/ALM
- 10 ALM/LAB
- ratio LUT/reg still 1
- specific addition logic.

Xilinx configurable DSP block



Programmable?

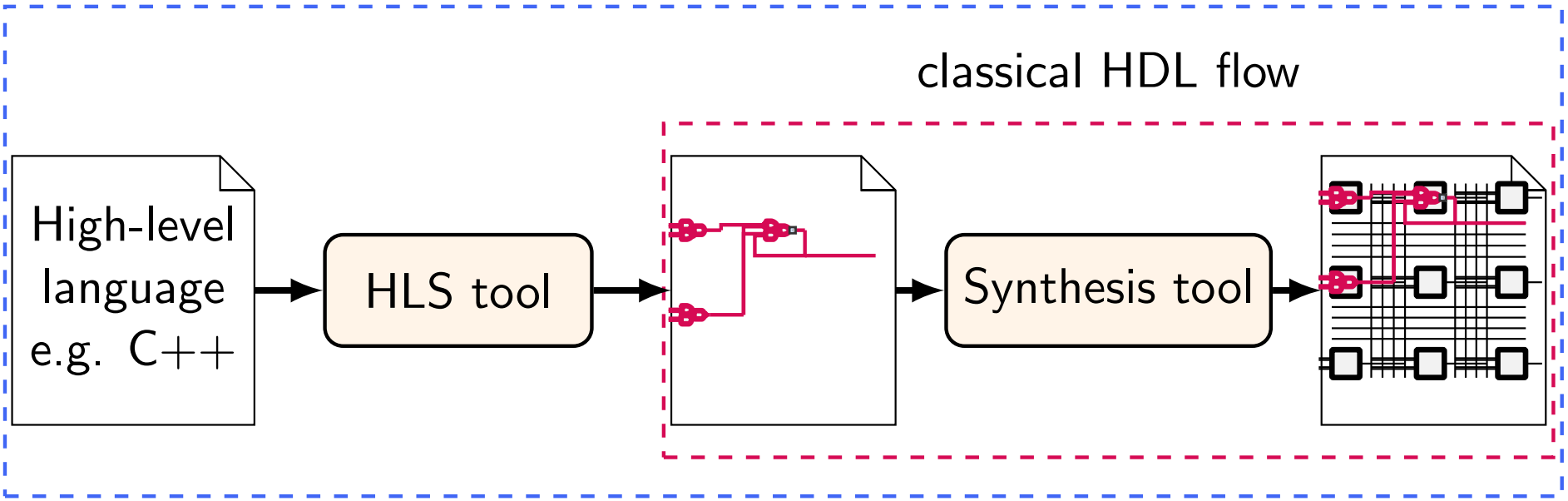
It was too simple, people would complain



- You don't program, you design a circuit
  - with fancy languages such as VHDL or Verilog
  - with compilers called "synthesis tools" that can take hours

It was too simple, people would complain

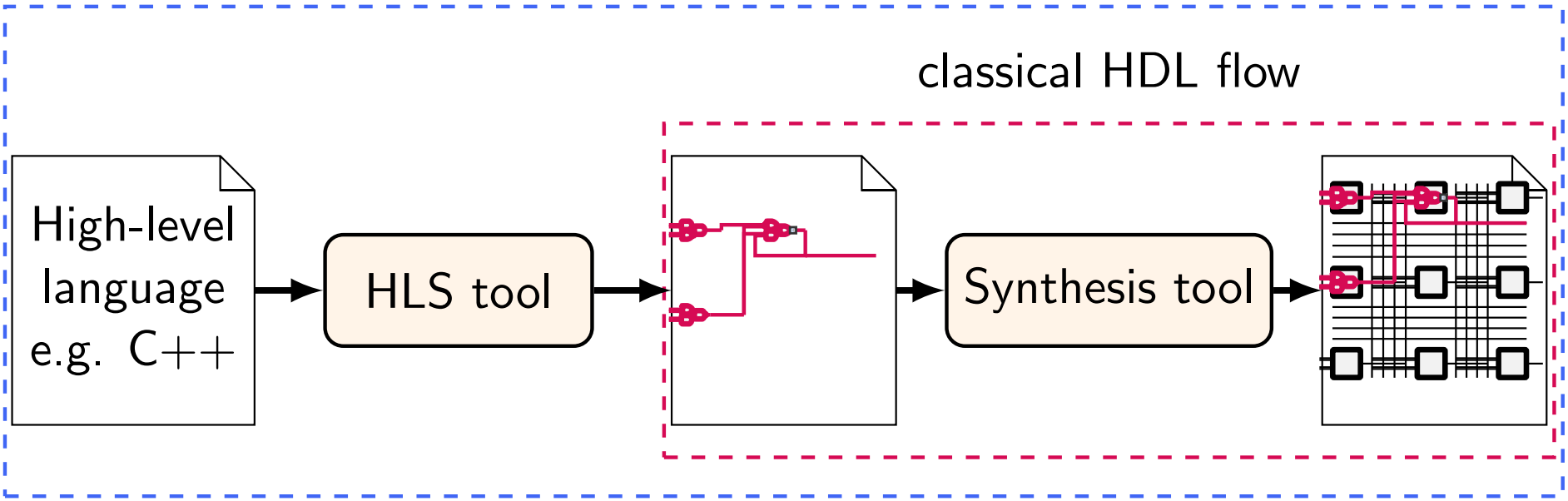
High-level synthesis (HLS) flow



- You don't program, you design a circuit
  - with fancy languages such as VHDL or Verilog
  - with compilers called "synthesis tools" that can take hours
- Since the 2010, FPGA programming in C/C++ for the rest of us
  - but you won't escape the synthesis tools

It was too simple, people would complain

High-level synthesis (HLS) flow



- You don't program, you design a circuit
  - with fancy languages such as VHDL or Verilog
  - with compilers called "synthesis tools" that can take hours
- Since the 2010, FPGA programming in C/C++ for the rest of us
  - but you won't escape the synthesis tools

C++ programming is called "high level" !?!

The VHDL language in two slides (1)

- Entities (= black boxes), ports, instances, signals (= wires)
  - just like when we draw architectures

The VHDL language in two slides (1)

- Entities (= black boxes), ports, instances, signals (= wires)
  - just like when we draw architectures
- Intrinsically parallel
  - in a circuit, all the gates operate in parallel
  - consequence: the order of statements in the code is often irrelevant
  - `A <= B xor C;` means: connect output of `B xor C` to `A`
    (this describes an infinite number of xor operations)

The VHDL language in two slides (1)

- Entities (= black boxes), ports, instances, signals (= wires)
  - just like when we draw architectures
- Intrinsically parallel
  - in a circuit, all the gates operate in parallel
  - consequence: the order of statements in the code is often irrelevant
  - `A <= B xor C;` means: connect output of `B xor C` to `A`
    (this describes an infinite number of xor operations)
- Two approaches to describing circuits (to be used together)
  - structural: boxes connected with wires
    - to be used for divide-and-conquer description of complex circuits
  - behavioural: describes *what the circuit does*, not how it is built
    - to be used to describe the lower-level (smaller) boxes
    - describe semantics, leave to the compiler the technology-dependent plumbing of gates/LUT
  - compiling behaviour into structure is a challenge, forever

Semantic of a circuit? Event-driven simulation

.

- an event (t, s, v) is a transition of signal s to value v at instant t. v may be 0, 1, Z (high impedance), and a few others
- the semantic of a circuit is: how it reacts to events on its inputs.

How to simulate a circuit (event-driven simulator)

- maintain a list of events (t, s), sorted by t:
  next event to happen is first of list
- while(list not empty) {
  remove first event;
  propagate it through the components that have it at input;
  insert the resulting events in the list;
  }

Not completely deterministic if several events happen at the same time.

The VHDL language in two slides (2)

- The semantic of `A <= B xor C;` is:
  each time an event arrives to `B` or to `C`, propagate it through the `xor` to generate an event on `A`
- again, such statements may be written in any order: the order of events is given by the graph of the circuit
- more accurate (when needed): `A <= B xor C after 10 ns;`
- Behaviourial VHDL: describe your circuit as processes that react to events. Such processes may be described in an imperative language.

The big challenge: rising abstraction level

Refining: Incrementally moving from an high-level description to an implementation
Examples of refinements (non exhaustive list)

- Replace a behavioral block with a structural one
- Parallelization: extract parallelism from sequential code
- Timing refinement: schedule everything
- Data refinement: from floating-point to fixed-point
- Placement and routing
- ...

FPGAs for low-latency audio

The Syfala project

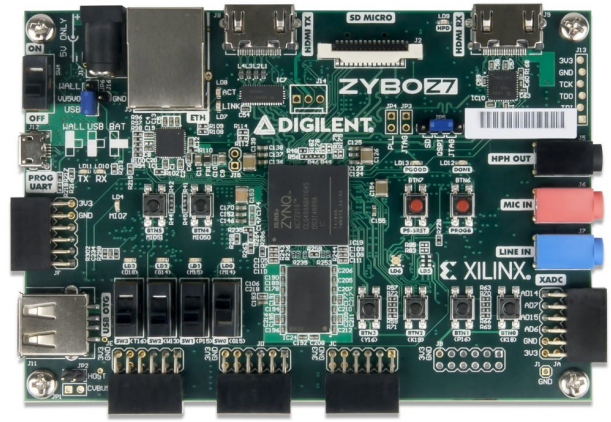
- Motivation: low-latency audio embedded systems
  - musical instruments
  - virtual and augmented audio (ambisonics...)
  - etc.
- Classical processors: enough power but latency too high
  - audio buffers all over the place
- Sound processing on FPGAs?
  - no bufffer, an audio pipeline with sub-sample latency
  - 4000 FPGA cycles in a sample period @ 44kHz
  - and also ample parallel computing power (80 DSP blocks)
  - and also fancy operators (e.g. 1-cycle sine)
  - ... but **extremely complex to program**

The Syfala project

- High level audio description language: Faust (GRAME).
- High Level Synthesis of FPGA bitsreams: VivadoHLS (Xilinx)
- Advanced arithmetics optimization: Flopoco (Socrate)

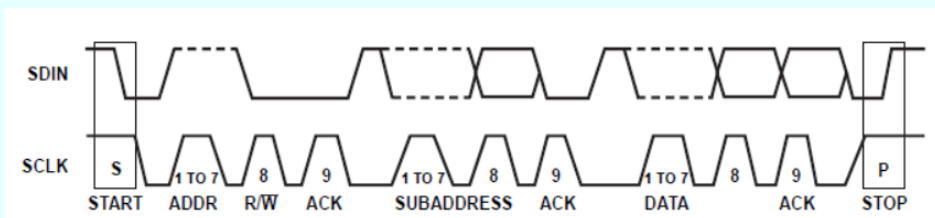
First Syfala prototype on Zybo Z7

- main chip: Xilinx Zynq-7000
  - 2 ARM processor cores
  - FPGA fabric
    - 28k logic cells
    - 80 DSP blocks
    - 60 36kBit memory block
    - ... running at 200 MHz
- Audio chip: SSM 2603
- 1GB DDR RAM
- all sorts of inputs/outputs

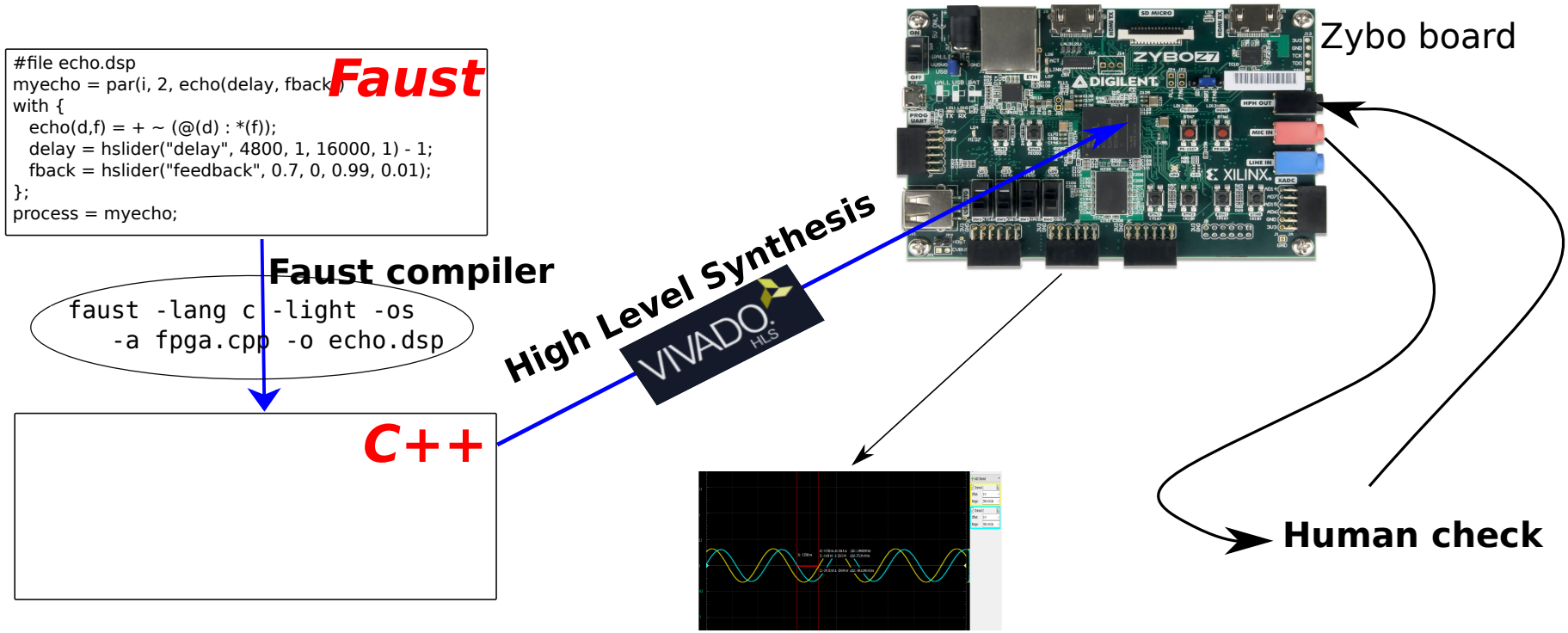


First internship: set up an audio pipeline on the FPGA

- route audio directly from SSM chip to FPGA logic and back
  (not using the ARM audio driver)
- (I2C/I2S protocols)



First latency evaluation: simple echo



- result: 840 μs latency: too much! (target: 100 μs)
  - 800 μs due to SSM chip

$\Rightarrow$ get rid of SSM chip?

- use a lower-latency audio chip?
- use a plain ADC + analog filter? (see e.g. futur3soundz)

Ultimately we want multiple input, multiple outputs anyway

Current state and upcoming challenges

Current state:

- Faust-to-Zybo working and automated
  - credits to Adeyemi, Ousmane, Tanguy, Stéphane, Romain, Yann, Gero and Alain
- ... including RAM access (to store past audio samples)
- but currently inefficient
  - who needs floating-point for audio hardware?

Fun stuff ahead

- fix the sound chip latency issue
- human interface on the ARM controlling the FPGA?
- fine-tune the Faust-generated hardware
- define a fixed-point Faust backend
- audio sample clock versus FPGA clock
- ...

Conclusion

FPGA programming for the rest of us

I want to try to program an FPGA, where do I start?

- Amazon, Microsoft or OVH all offer FPGA resources on the cloud
  - I haven't tried yet.
- Xilinx as well as Intel (intend to) sell entry-level boards that can be programmed in Python
  - I haven't tried yet (but maybe somebody here has).
- Frameworks that limit application domains/expressivity for the sake of ease of use?
  - e.g. assemble pre-compiled block to avoid the huge compilation time
  - Maxeler data-flow compiler (see last talk today)
- Go for HLS, even if it means sacrificing performance?
  - parallel with popular programming languages...
  - don't believe that you get good results without understanding the FPGA architecture
  - don't believe that you get good results without tweaking

The Dinechin theorem

For 20 years, the FPGA community has been waiting for the “killer application”.
(The widely useful application on which the FPGA is so much better)

The Dinechin theorem

For 20 years, the FPGA community has been waiting for the “killer application”.
(The widely useful application on which the FPGA is so much better)

Theorem: we’ll wait forever.

The Dinechin theorem

For 20 years, the FPGA community has been waiting for the "killer application".
(The widely useful application on which the FPGA is so much better)

Theorem: we'll wait forever.

Proof: When such an application pops up,

- either it is indeed widely useful, and next year's Pentium will do it in hardware 10x faster than the FPGA, so it won't be an *FPGA* killer app next year,
- or the FPGA remains competitive next year, but it means that it was not a killer app.

The Dinechin theorem

For 20 years, the FPGA community has been waiting for the "killer application".
(The widely useful application on which the FPGA is so much better)

Theorem: we'll wait forever.

Proof: When such an application pops up,

- either it is indeed widely useful, and next year's Pentium will do it in hardware 10x faster than the FPGA, so it won't be an *FPGA* killer app next year,
- or the FPGA remains competitive next year, but it means that it was not a killer app.

Still, the killer feature of FPGAs is their programmability.

Finest Programmable Granularity Around

- Programmable chips, but programmed in terrible languages

Finest Programmable Granularity Around

- Programmable chips, but programmed in terrible languages
- Efficient for integer addition and integer multiplication

Finest Programmable Granularity Around

- Programmable chips, but programmed in terrible languages
- Efficient for integer addition and integer multiplication
- Efficient for implementing tables

Finest Programmable Granularity Around

- Programmable chips, but programmed in terrible languages
- Efficient for integer addition and integer multiplication
- Efficient for implementing tables
- All this at the granularity of the bit
  - 17-bit adder
  - multiplier of 7-bit numbers by 56-bit numbers
  - table of 2^7 entries of 17 bits each
  - ...

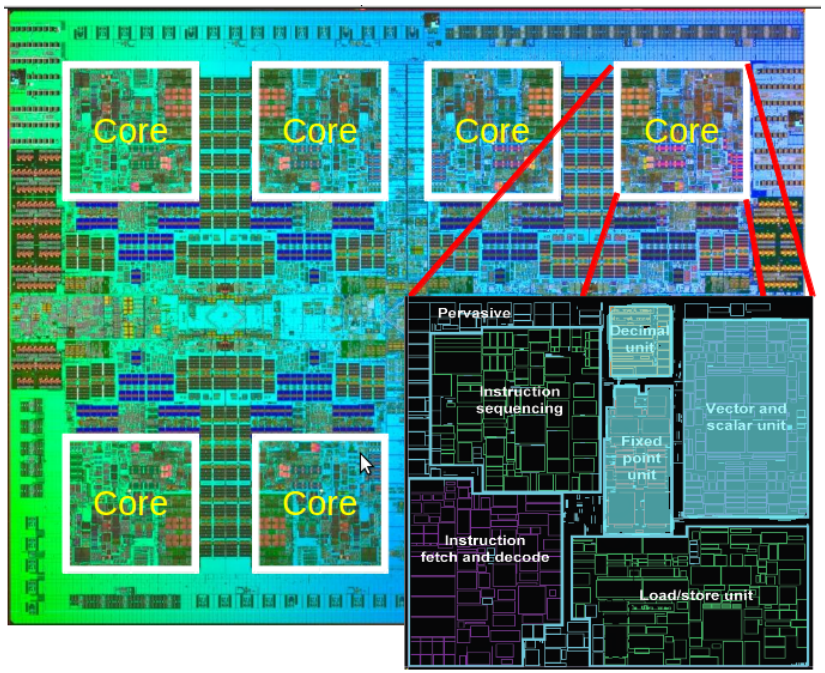
Finest Programmable Granularity Around

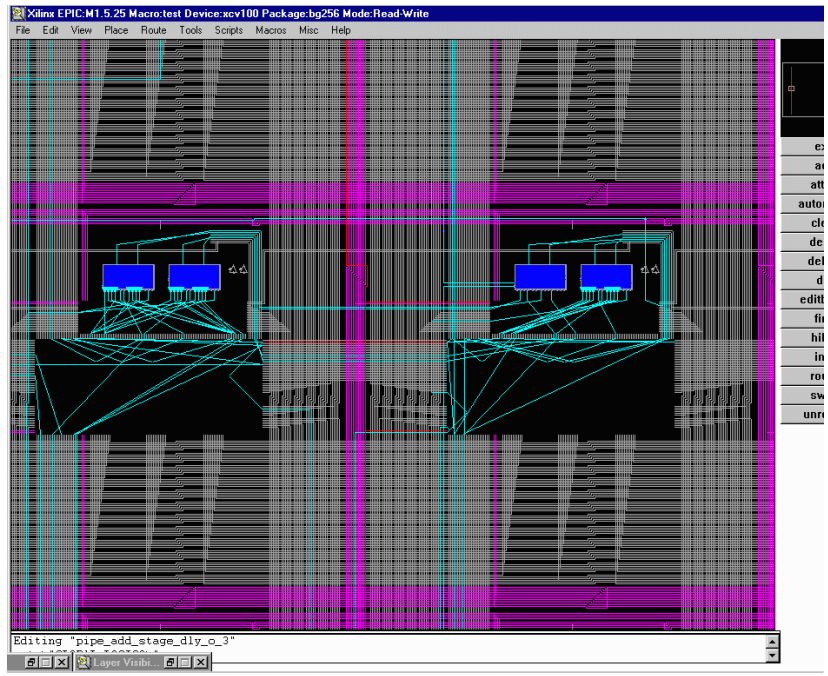
- Programmable chips, but programmed in terrible languages
- Efficient for integer addition and integer multiplication
- Efficient for implementing tables
- All this at the granularity of the bit
  - 17-bit adder
  - multiplier of 7-bit numbers by 56-bit numbers
  - table of 2^7 entries of 17 bits each
  - ...
- And of course glue logic

This is the teaser for my second talk.

Back to the war of the programming models

Here are two ~~programmable chips~~ ways of wasting silicon.





Which is best for (insert your computation here) ?

Thank you for your attention.

Backup slides