Saclay 2022

Computing with FPGAs ?

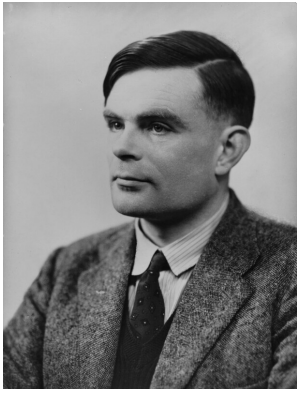
Florent de Dinechin

Outline

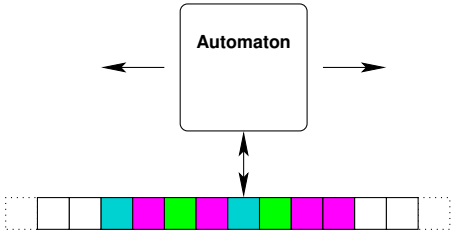
Introduction: the war of the programming models

Alan Turing and John von Neumann

... *the twin gods of the computing pantheon* (A. C. Clarke in 2001: A Space Odyssey)

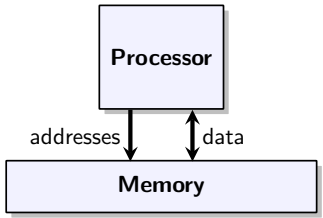


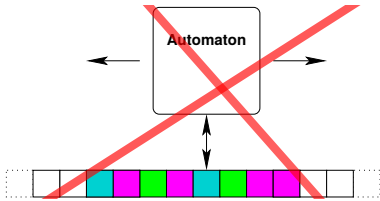
The Turing Machine: a good idea, but a commercial failure



- Defines universality
- Finite automaton infinite memory
- local access to the memory

The von Neumann Machine that ruled

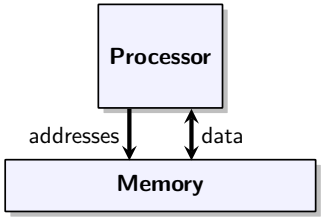


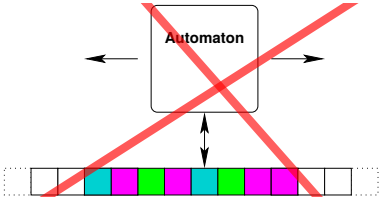


- Universal as well
- Same finite automaton (processor), same infinite[1] memory
- Turing-killer feature: random access to the memory

[1] From there on, infinite means: some power of two, e.g. 2^{32} so large that I can't count that far.

The von Neumann Machine that ruled





- Universal as well
- Same finite automaton (processor), same infinite[1] memory
- Turing-killer feature: random access to the memory
  - so much more efficient (no tape rewinding)
  - so much easier to exploit (program here, data there, etc)

[1] From there on, infinite means: some power of two, e.g. 2^{32} so large that I can't count that far.

When reality kicks back

A law of nature

You can't move data
faster than the speed of light

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

When reality kicks back

A law of nature

You can't move data
faster than the speed of light
If the memory is infinite,
some bits will be physically distant
and will therefore be accessed slowly.

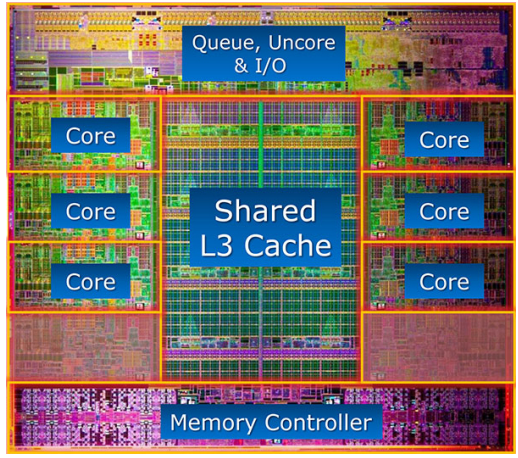
Random access in constant (fast) time
is but a dream
and the von Neumann model
is but a model...

When reality kicks back

A law of nature

You can't move data
faster than the speed of light
If the memory is infinite,
some bits will be physically distant
and will therefore be accessed slowly.

Random access in constant (fast) time
is but a dream
and the von Neumann model
is but a model...

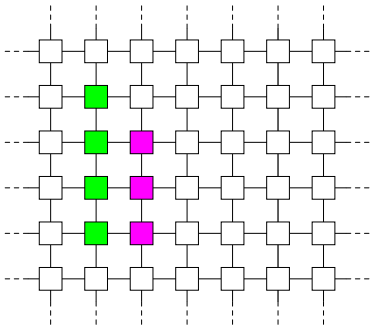


One half of your processor is there to keep this dream alive!

Meanwhile, von Neumann had a better idea

Cellular automaton: a spatial/parallel version of Turing machine.

(most famous instance: Conway's **Game of Life**)



- an infinite number of automata
- all working in parallel
- with next-neighbour (local) communications
- univeral all the same

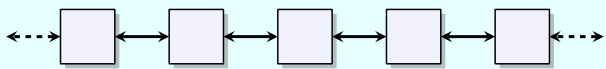
(youtube "game of life in game of life")

Let us build this! (with a little help of Moore's law)

Yes but... next-neighbour communications suck!

The firing squad synchronization problem

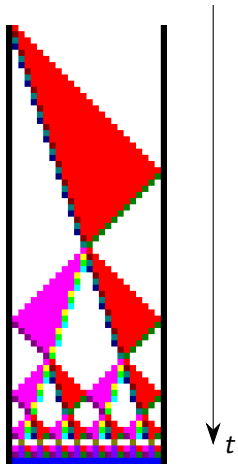
Synchronize n cells, using next-neighbour communication only



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

Jacques Mazoyer.
A six-state minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, 50(2):183–238, 1987.

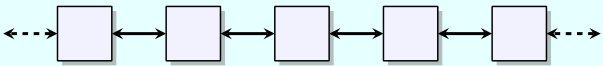
2D version: youtube "game of life in game of life"



Yes but... next-neighbour communications suck!

The firing squad synchronization problem

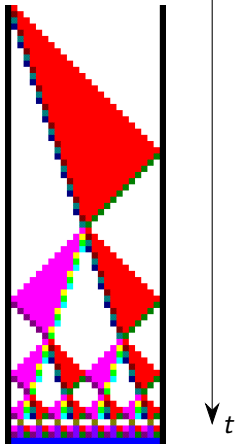
Synchronize n cells, using next-neighbour communication only



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

Jacques Mazoyer.
A six-state minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, 50(2):183–238, 1987.

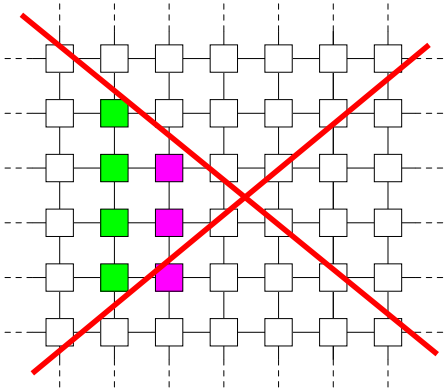
2D version: youtube "game of life in game of life"

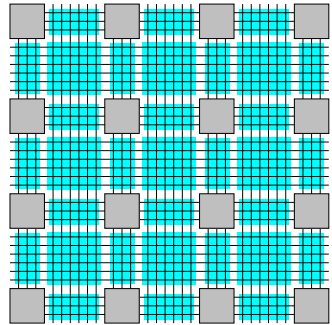


... in real life the captain simply shouts "Fire!" (a global communication)

Field-Programmable Gate Arrays

FPGAs are to 2D cellula automata what von Neuman machines are to Turing machines:
something useful in practice.

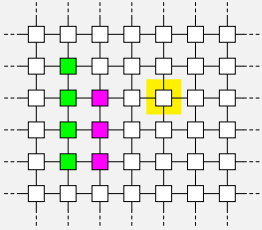




FPGA architectures

Basic FPGA structure

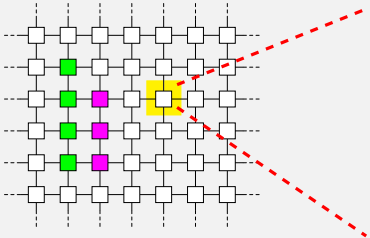
Overview



- Configurable logic cell
  - ... all what you need to build an automaton

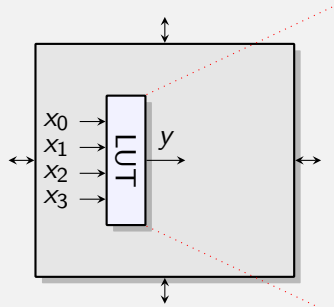
Basic FPGA structure

Overview



- Configurable logic cell
  - ... all what you need to build an automaton

Inside a cell



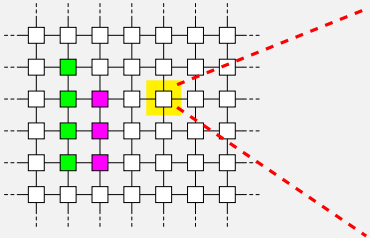
x_0	x_1	x_2	x_3	y
0	0	0	0	0
0	0	0	0	1
0	0	0	1	1
0	0	0	1	1
0	0	1	0	0
0	0	1	0	0
0	0	1	1	0
0	0	1	1	1
1	0	0	0	1
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	1
1	0	1	0	0
1	0	1	1	0
1	0	1	1	1

- A Look-Up Table (LUT)
  - 4 inputs (for some value of 4),
  - 1 output

... may be filled with any truth table

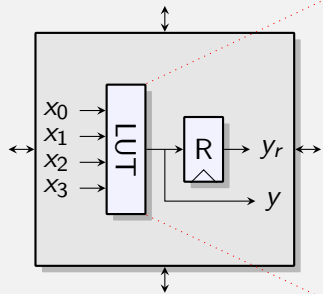
Basic FPGA structure

Overview



- Configurable logic cell
  - ... all what you need to build an automaton

Inside a cell



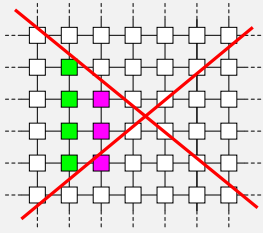
x_0	x_1	x_2	x_3	y
0	0	0	0	0
0	0	0	0	1
0	0	0	1	1
0	0	0	1	1
0	0	1	0	0
0	0	1	0	0
0	0	1	1	0
0	0	1	1	1
1	0	0	0	1
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	1
1	0	1	0	0
1	0	1	1	0
1	0	1	1	1

- A Look-Up Table (LUT)
  - 4 inputs (for some value of 4),
  - 1 output

  ... may be filled with any truth table
- 1 bit of run-time memory R

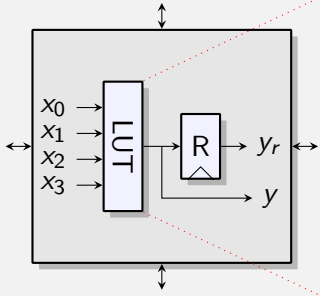
Basic FPGA structure

Overview



- Configurable logic cell
  - ... all what you need to build an automaton
- Random access to distant cells

Inside a cell



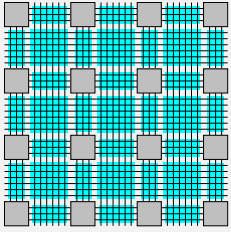
x_0	x_1	x_2	x_3	y
0	0	0	0	0
0	0	0	0	1
0	0	0	1	1
0	0	0	1	1
0	0	1	0	0
0	0	1	0	0
0	0	1	1	0
0	0	1	1	1
1	0	0	0	1
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	1
1	0	1	0	0
1	0	1	1	0
1	0	1	1	1

- A Look-Up Table (LUT)
  - 4 inputs (for some value of 4),
  - 1 output

  ... may be filled with any truth table
- 1 bit of run-time memory R

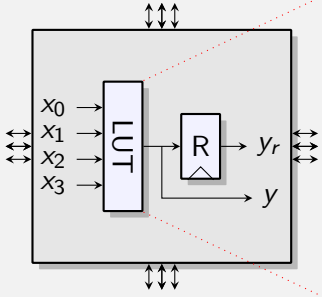
Basic FPGA structure

Overview



- Configurable logic cell
  - ... all what you need to build an automaton
- Random access to distant cells
  - routing channels
  - configurable switch boxes

Inside a cell

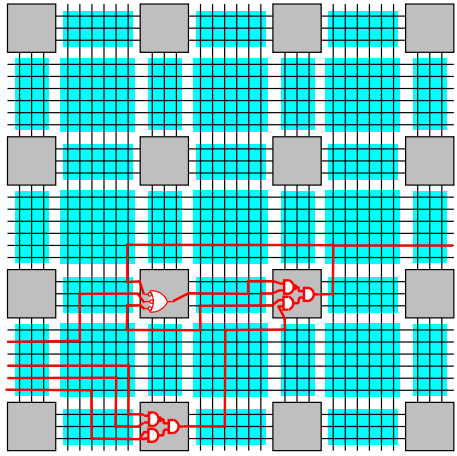


x_0	x_1	x_2	x_3	y
0	0	0	0	0
0	0	0	0	1
0	0	0	1	1
0	0	0	1	1
0	0	1	0	0
0	0	1	0	0
0	0	1	1	0
0	0	1	1	1
1	0	0	0	1
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	1
1	0	1	0	0
1	0	1	1	0
1	0	1	1	1

- A Look-Up Table (LUT)
  - 4 inputs (for some value of 4),
  - 1 output

  ... may be filled with 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 (many ms)

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

an FPGA program == a lot of configuration bits

Two moments in the life of an FPGA

Configuration time (many ms)

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

an FPGA 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 (many ms)

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

an FPGA program == a lot of configuration bits

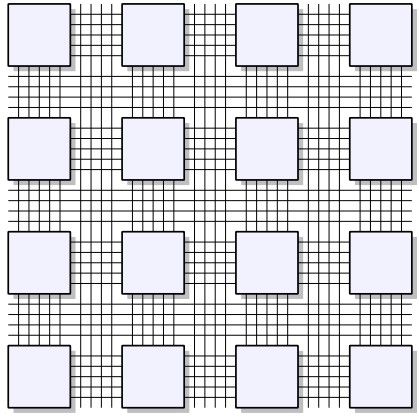
Run time (forever if needed)

- 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

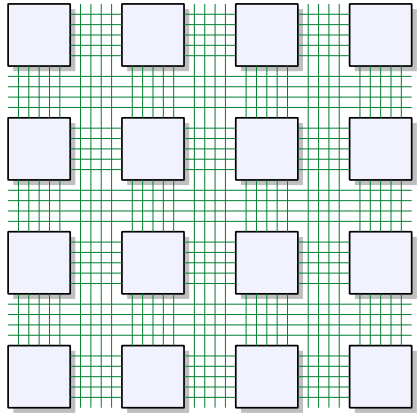
When reality kicks back

How many wires do we need per routing channel for random access to distant cells?
1990:



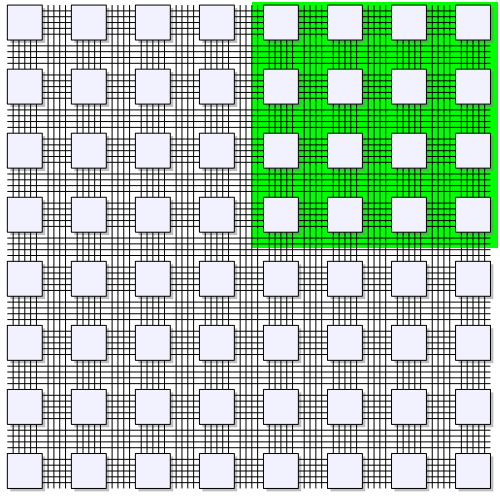
When reality kicks back

How many wires do we need per routing channel for random access to distant cells?
1990:



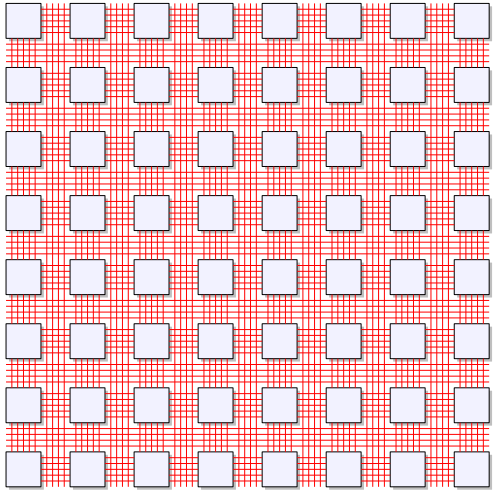
When reality kicks back

How many wires do we need per routing channel for random access to distant cells?
1993:



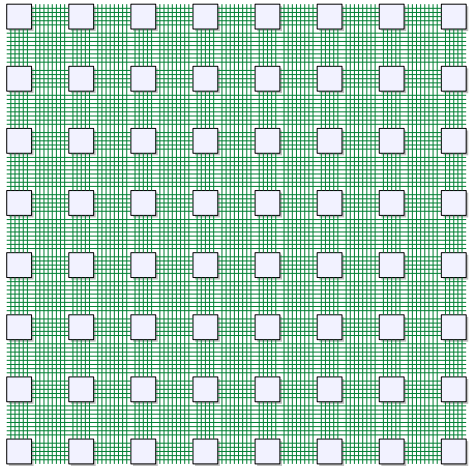
When reality kicks back

How many wires do we need per routing channel for random access to distant cells?
1993:



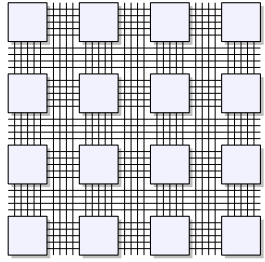
When reality kicks back

How many wires do we need per routing channel for random access to distant cells?
1994:



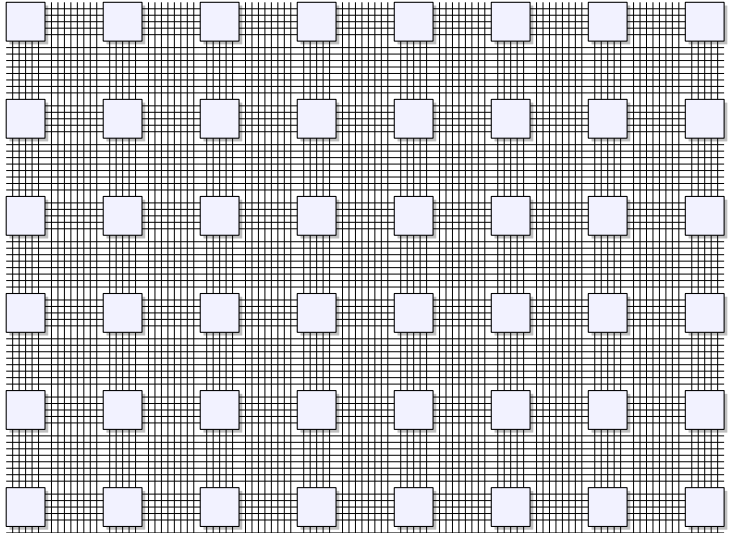
When reality kicks back

1990:



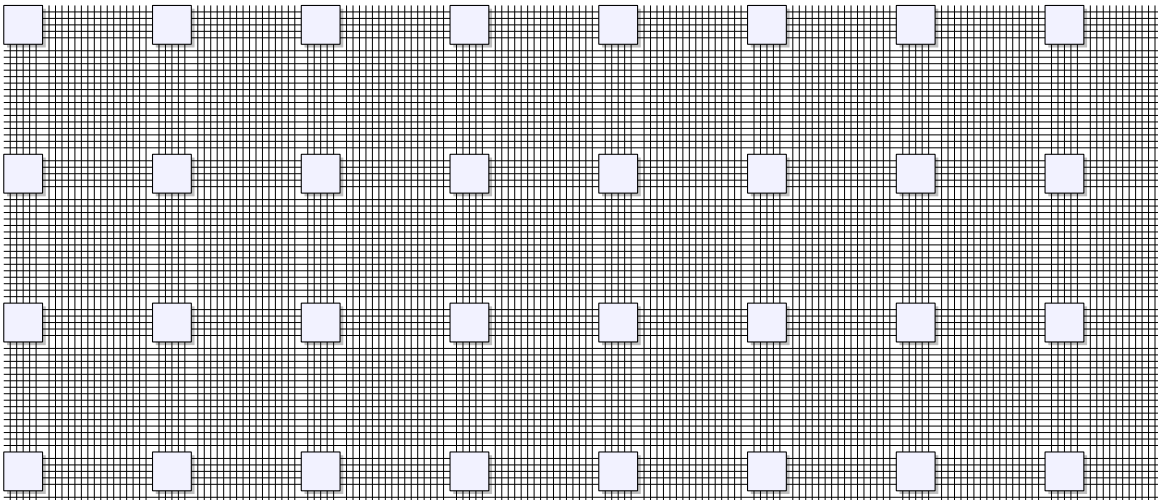
When reality kicks back

1994:



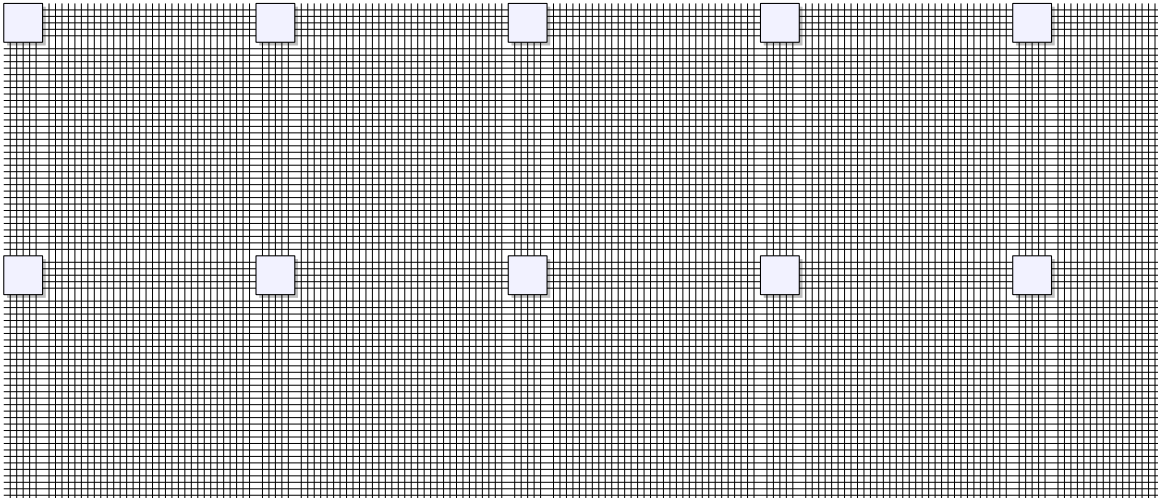
When reality kicks back

1996:



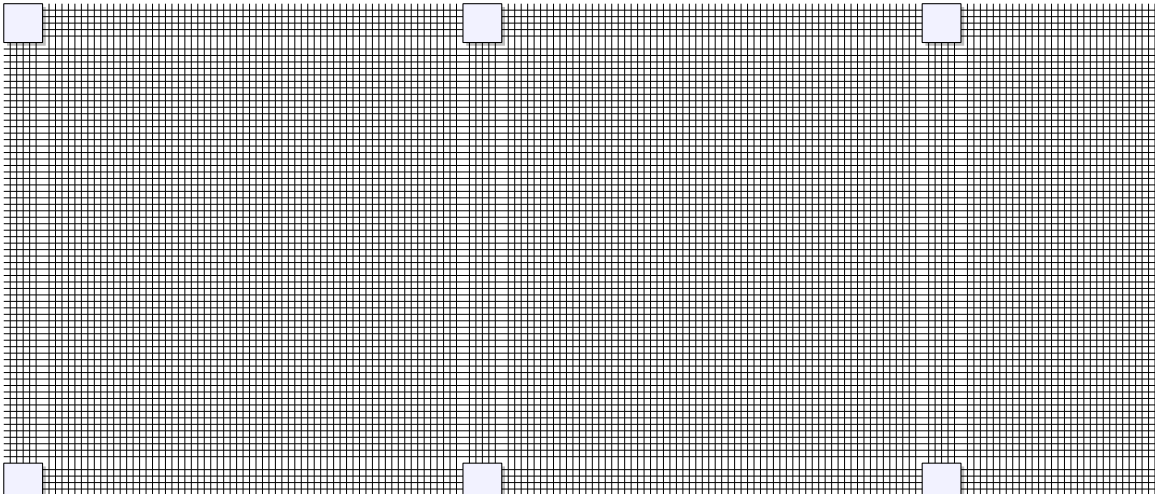
When reality kicks back

1999:



“Customers buy logic, but they pay for routing” (M. Langhammer)

2001 (after which I quit counting):



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.

- you plan a city, with roads sized to avoid traffic jams at the time

The curse of Los Angeles

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

- you plan a city, with roads sized to avoid traffic jams at the time
- then the city grows, and more people want to use the roads in the old center

The curse of Los Angeles

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

- you plan a city, with roads sized to avoid traffic jams at the time
- then the city grows, and more people want to use the roads in the old center
- so they get traffic-jammed.

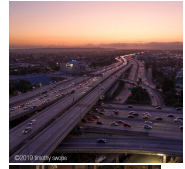
The curse of Los Angeles

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

- you plan a city, with roads sized to avoid traffic jams at the time
- then the city grows, and more people want to use the roads in the old center
- so they get traffic-jammed.

Two examples of this process at work:

- **Los Angeles**
  - since the 30s: one century of car-centered progress
  - ever-wider highways built in place of housing (see Roger Rabit)
  - now: 2/3 of the area dedicated to cars (roads + parking lots)
- **Lyon**
  - in the 70s: planned destruction of a Renaissance area to make space for a highway
  - conservatism, opposition to progress $\longrightarrow$ project cancellation
  - now: a car-free area, and UNESCO world heritage



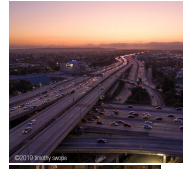
The curse of Los Angeles

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

- you plan a city, with roads sized to avoid traffic jams at the time
- then the city grows, and more people want to use the roads in the old center
- so they get traffic-jammed.

Two examples of this process at work:

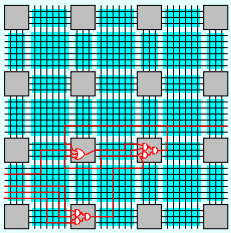
- **Los Angeles**
  - since the 30s: one century of car-centered progress
  - ever-wider highways built in place of housing (see Roger Rabit)
  - now: 2/3 of the area dedicated to cars (roads + parking lots)
- **Lyon**
  - in the 70s: planned destruction of a Renaissance area to make space for a highway
  - conservatism, opposition to progress $\longrightarrow$ project cancellation
  - now: a car-free area, and UNESCO world heritage



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

Rent's law

Yet another experimental law



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

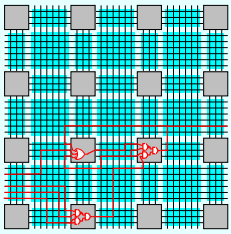
- more than proportional to n, the diameter,
- note quite proportional to the area n^2 of each half-circuit.

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 (gates),
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...

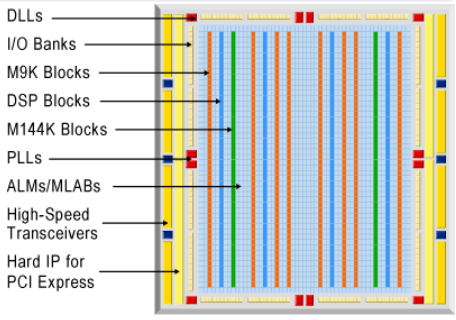
Our city planners should take crash courses in complexity theory.

Solutions

In cities:	**Transposed to FPGA:**
Build highways of various widths	heterogeneous routing
Build busses, metro, tramway	increase compute granularity
Relocalize the economy	relocalize computations
And for you, the user:	
Use bicycles instead of SUVs	compute just right

Current FPGAs (with all these solutions)

- coarser cells, optimized for additions (up to 2,000,000 6-input LUT)
- small (24 bit) multipliers ("DSP blocks") (up to 3,000)
- small ($\approx$ 10 kBit) memories (up to 2,000)
- many independent clock networks & PLLs
- flexible input/outputs
- ...



the Altera/Intel stratix IV FPGA

... and still, the price of routing

- 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 of the ASIC
  - there are transistors on all the wires!

How **not** to do scientific computing on FPGAs

- In 2007, an Intel team proudly demonstrates
  their 1994 flagship processor in a single FPGA
- and it runs at 25MHz (1/3rd of the 1994 frequency).
- It boots to Linux (terminal only) in 10mn

It will probably *not* accelerate your scientific code.

How **not** to do scientific computing on FPGAs

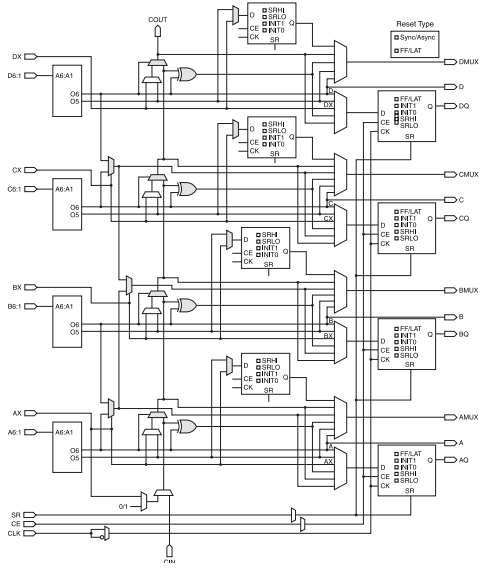
- In 2007, an Intel team proudly demonstrates
  their 1994 flagship processor in a single FPGA
- and it runs at 25MHz (1/3rd of the 1994 frequency).
- It boots to Linux (terminal only) in 10mn

It will probably *not* accelerate your scientific code.

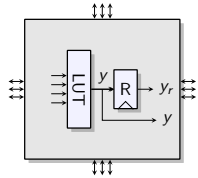
... jokes aside, this is a really nice paper.

Shih-Lien L. Lu, Peter Yiannacouras, Taeweon Suh, Rolf Kassa, Michael Konow.
An FPGA-Based Pentium® in a Complete Desktop System
In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2007.

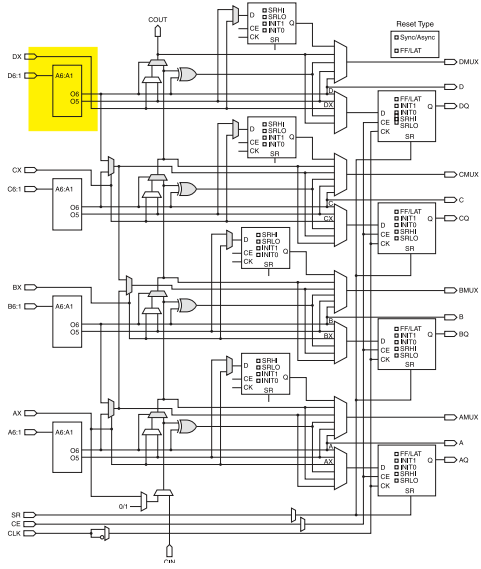
WRKB: The real ~~Xilinx~~ AMD Configurable Logic Block



A “slice” (Virtex7)

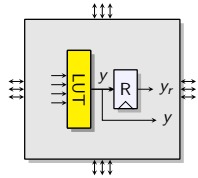


WRKB: The real ~~Xilinx~~ AMD Configurable Logic Block

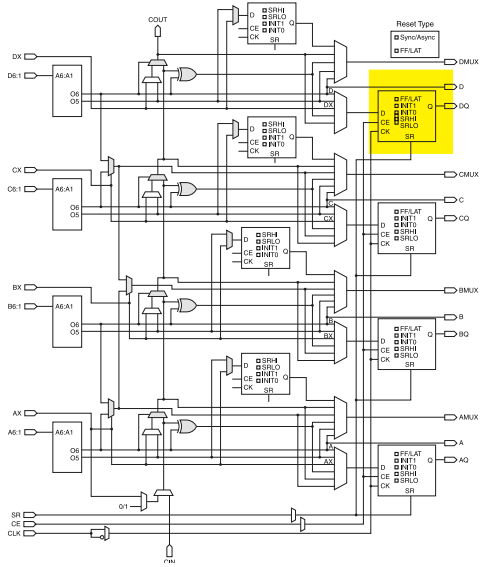


A “slice” (Virtex7)

- See the LUT?

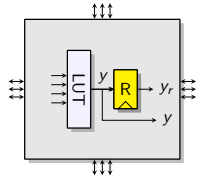


WRKB: The real ~~Xilinx~~ AMD Configurable Logic Block

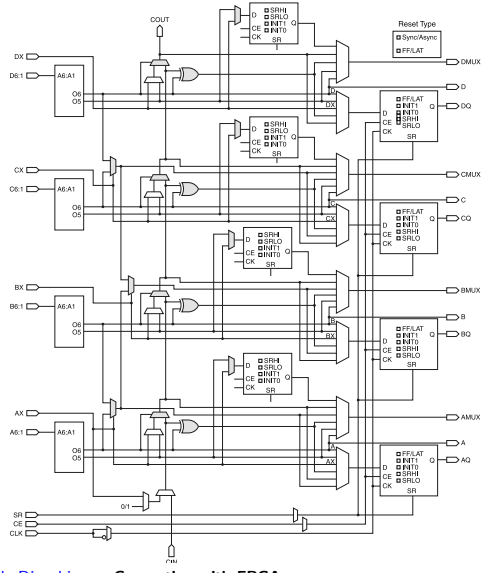


A “slice” (Virtex7)

- See the LUT?
- ... the register?

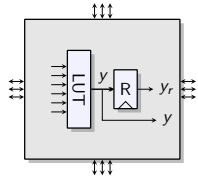


WRKB: The real ~~Xilinx~~ AMD Configurable Logic Block



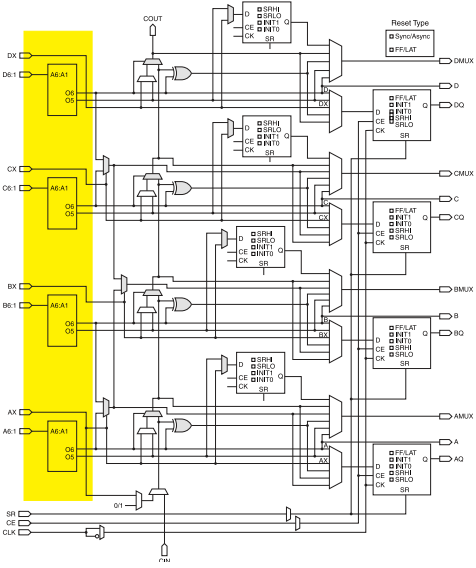
A "slice" (Virtex7)

- See the LUT?
- ... the register?



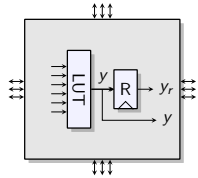
- Granularity increasing
  - 6-input LUTs (and counting)

WRKB: The real ~~Xilinx~~ AMD Configurable Logic Block



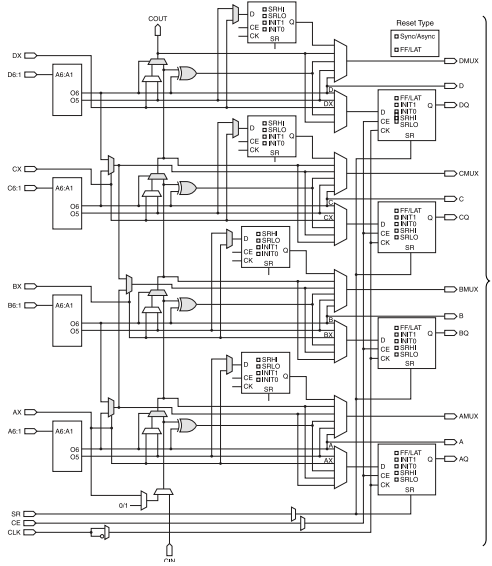
A "slice" (Virtex7)

- See the LUT?
- ... the register?



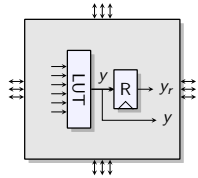
- Granularity increasing
  - 6-input LUTs (and counting)
  - 4 LUT/slice (and counting)

WRKB: The real ~~Xilinx~~ AMD Configurable Logic Block



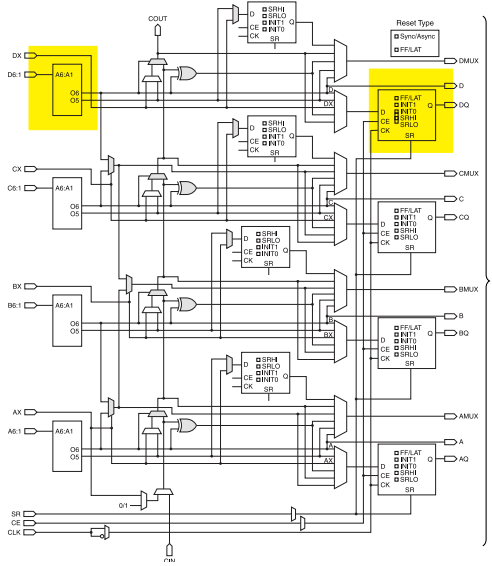
A "slice" (Virtex7)

- See the LUT?
- ... the register?



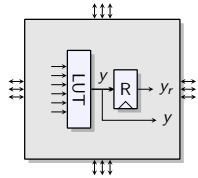
- Granularity increasing
  - 6-input LUTs (and counting)
  - 4 LUT/slice (and counting)
  - 2 slices/CLB

WRKB: The real ~~Xilinx~~ AMD Configurable Logic Block



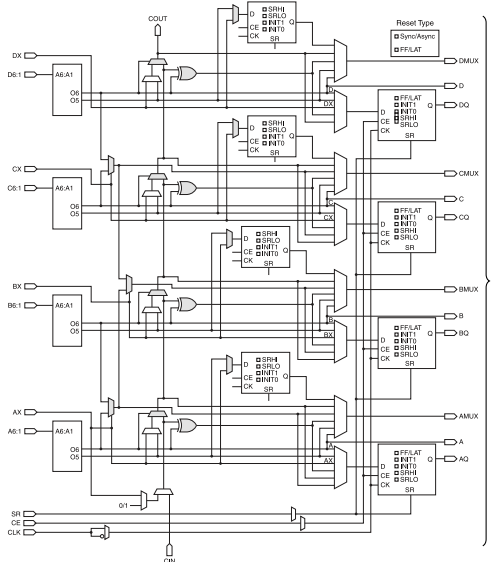
A "slice" (Virtex7)

- See the LUT?
- ... the register?



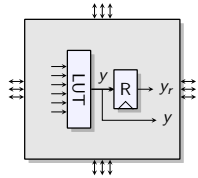
- Granularity increasing
  - 6-input LUTs (and counting)
  - 4 LUT/slice (and counting)
  - 2 slices/CLB
  - Ratio reg/LUT still equal to 1

WRKB: The real ~~Xilinx~~ AMD Configurable Logic Block



A "slice" (Virtex7)

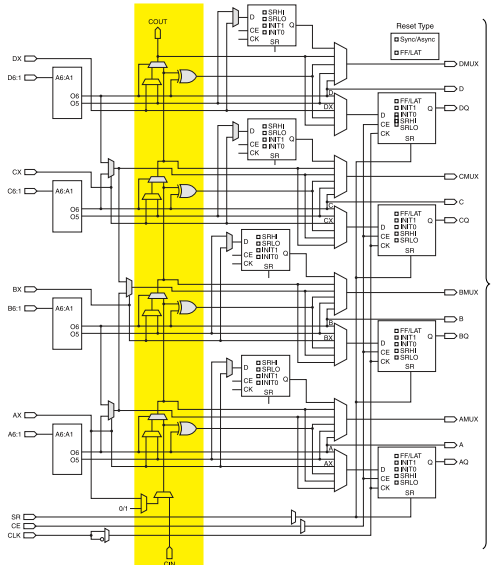
- See the LUT?
- ... the register?



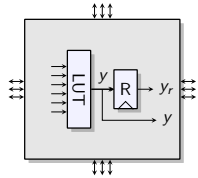
- Granularity increasing
  - 6-input LUTs (and counting)
  - 4 LUT/slice (and counting)
  - 2 slices/CLB
  - Ratio reg/LUT still equal to 1

All this keeps routing local

WRKB: The real ~~Xilinx~~ AMD Configurable Logic Block



A "slice" (Virtex7)

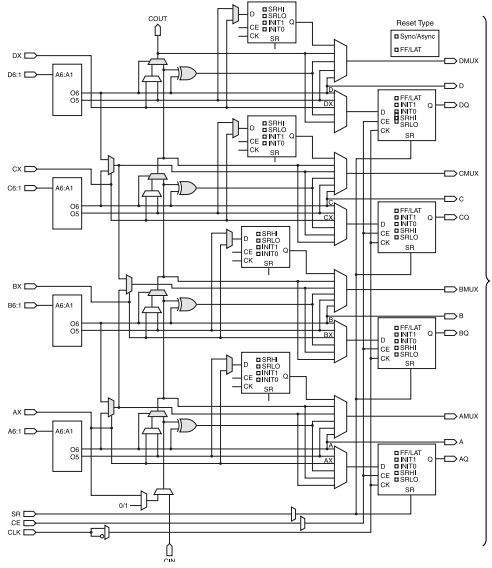


- See the LUT?
- ... the register?
- Granularity increasing
  - 6-input LUTs (and counting)
  - 4 LUT/slice (and counting)
  - 2 slices/CLB
  - Ratio reg/LUT still equal to 1

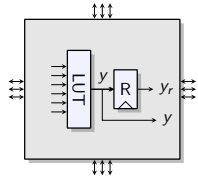
  All this keeps routing local
- Support of frequent ops
  - addition: **carry logic**

    (skips the slow routing)

WRKB: The real ~~Xilinx~~ AMD Configurable Logic Block



A "slice" (Virtex7)

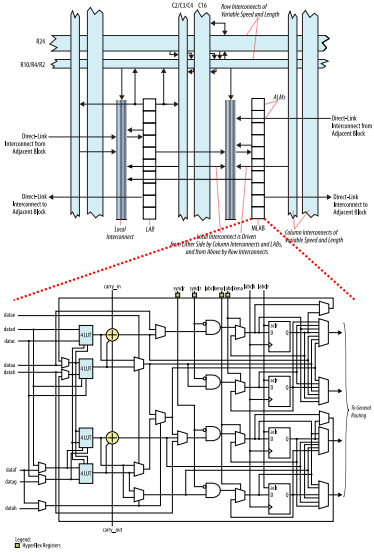


- See the LUT?
- ... the register?

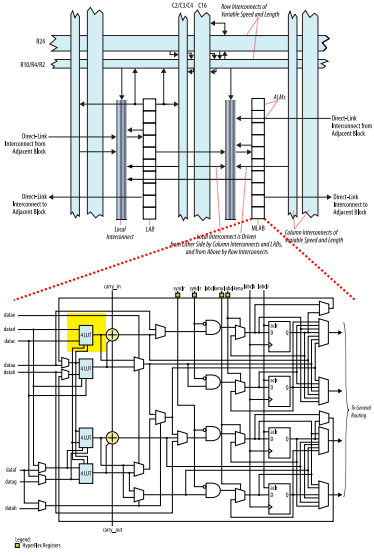
- Granularity increasing
  - 6-input LUTs (and counting)
  - 4 LUT/slice (and counting)
  - 2 slices/CLB
  - Ratio reg/LUT still equal to 1

  All this keeps routing local
- Support of frequent ops
  - addition: **carry logic** (skips the slow routing)
  - shift registers (SRL)
  - etc...

WRKB: The real ~~Altera~~ Intel Logic Array Block

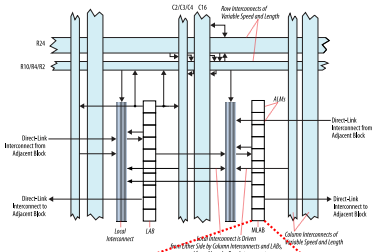


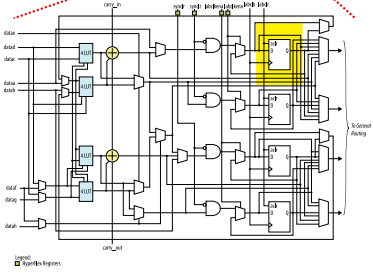
WRKB: The real ~~Altera~~ Intel Logic Array Block



- You still see the LUTs (4 inputs/LUT)

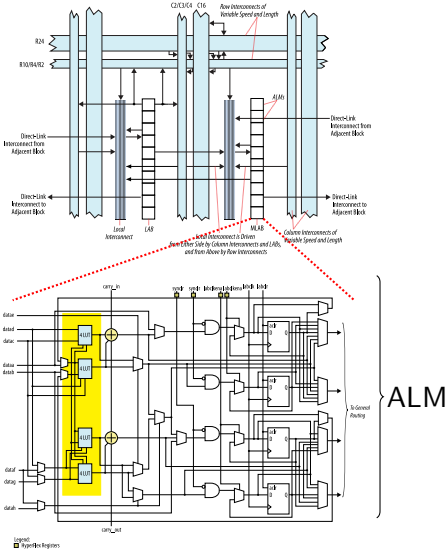
WRKB: The real ~~Altera~~ Intel Logic Array Block





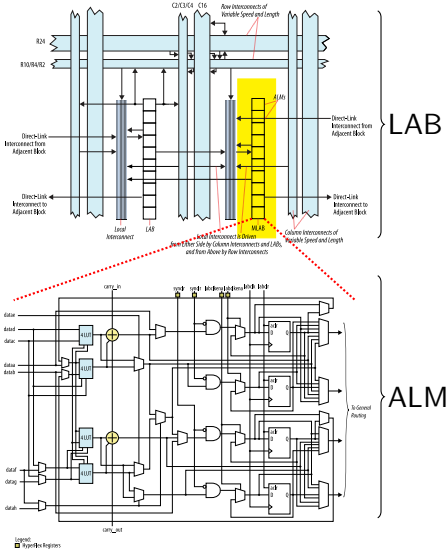
- You still see the LUTs (4 inputs/LUT)
- and the registers

WRKB: The real ~~Altera~~ Intel Logic Array Block



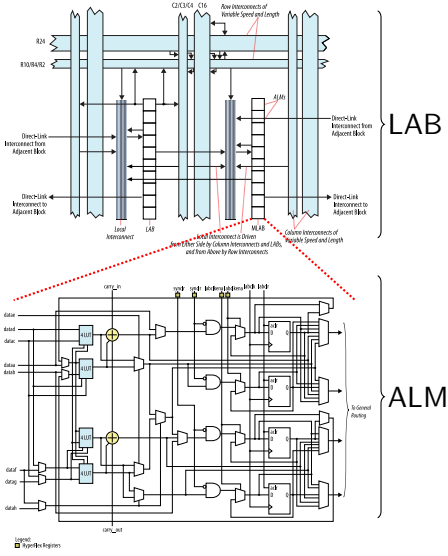
- You still see the LUTs (4 inputs/LUT)
- and the registers
- Granularity increasing
  - 4 LUT/ALM (adaptive logic module)

WRKB: The real ~~Altera~~ Intel Logic Array Block



- You still see the LUTs (4 inputs/LUT)
- and the registers
- Granularity increasing
  - 4 LUT/ALM (adaptive logic module)
  - 10 ALM/LAB (logic array block)

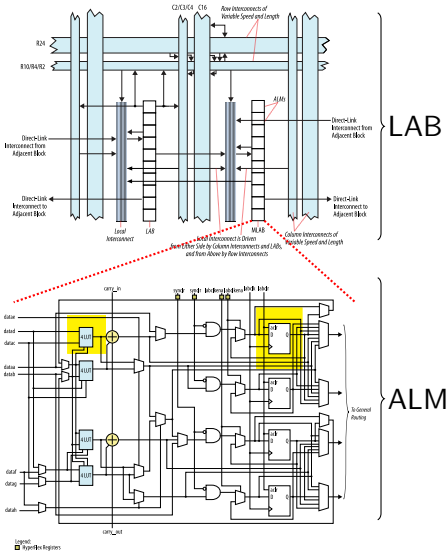
WRKB: The real ~~Altera~~ Intel Logic Array Block



- You still see the LUTs (4 inputs/LUT)
- and the registers
- Granularity increasing
  - 4 LUT/ALM (adaptive logic module)
  - 10 ALM/LAB (logic array block)

(and here you see the routing!)

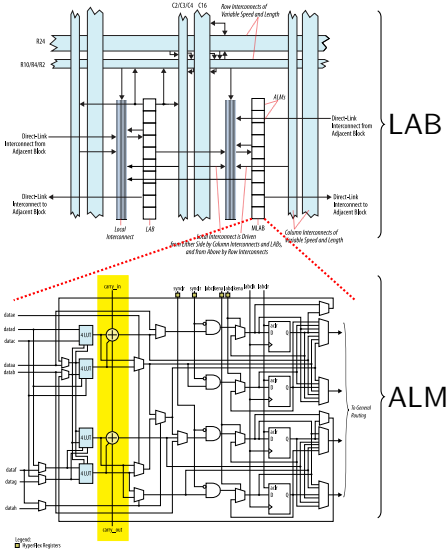
WRKB: The real ~~Altera~~ Intel Logic Array Block



- You still see the LUTs (4 inputs/LUT)
- and the registers
- Granularity increasing
  - 4 LUT/ALM (adaptive logic module)
  - 10 ALM/LAB (logic array block)

  (and here you see the routing!)
- ratio LUT/reg still 1

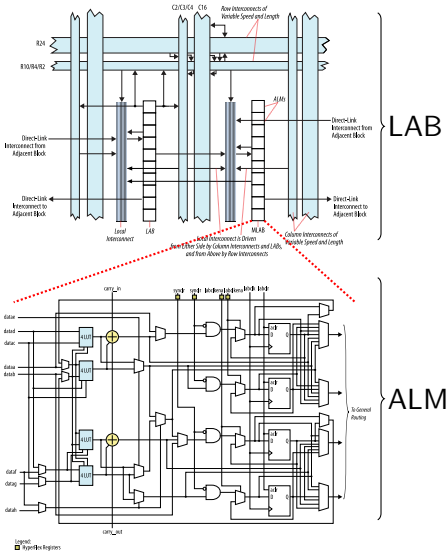
WRKB: The real ~~Altera~~ Intel Logic Array Block



- You still see the LUTs (4 inputs/LUT)
- and the registers
- Granularity increasing
  - 4 LUT/ALM (adaptive logic module)
  - 10 ALM/LAB (logic array block)

  (and here you see the routing!)
- ratio LUT/reg still 1
- specific addition logic.

WRKB: The real ~~Altera~~ Intel Logic Array Block

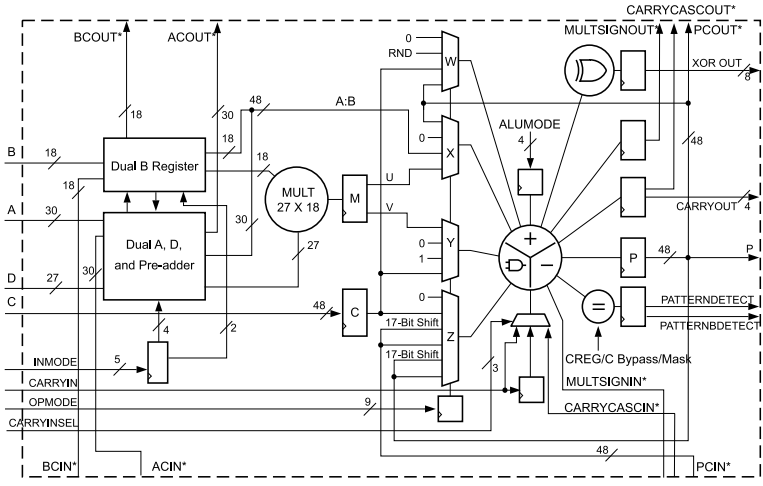


- You still see the LUTs (4 inputs/LUT)
- and the registers
- Granularity increasing
  - 4 LUT/ALM (adaptive logic module)
  - 10 ALM/LAB (logic array block)

  (and here you see the routing!)
- ratio LUT/reg still 1
- specific addition logic.

Two teams solving the same problems with mutual patent avoidance

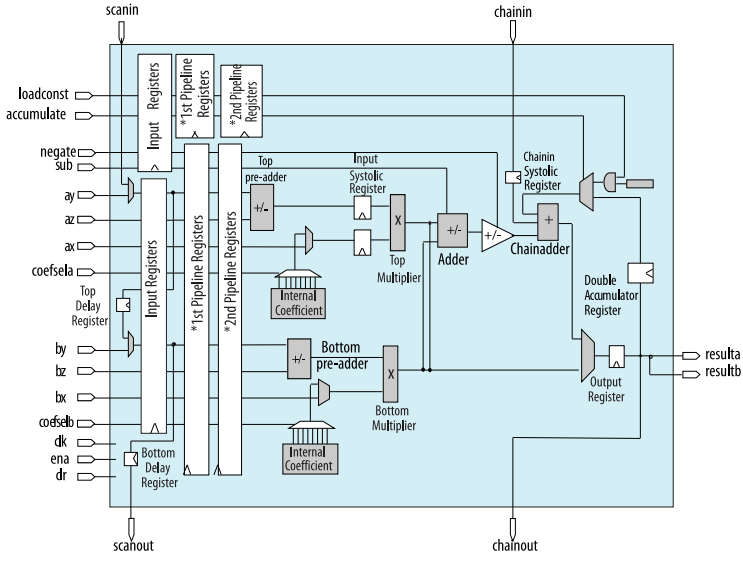
The ~~Xilinx~~ AMD configurable DSP block



A multiplier with pre-adders and post-adders (for complex mult, symmetric FIR filters, etc.)

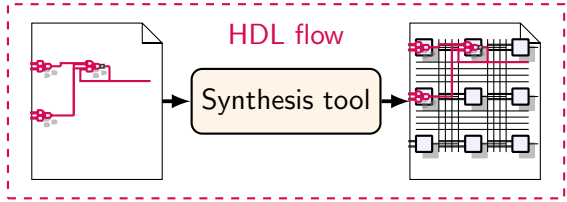
The ~~Altera~~ Intel variable-precision DSP block

same comment
(with 2 multipliers)



Programming FPGAs

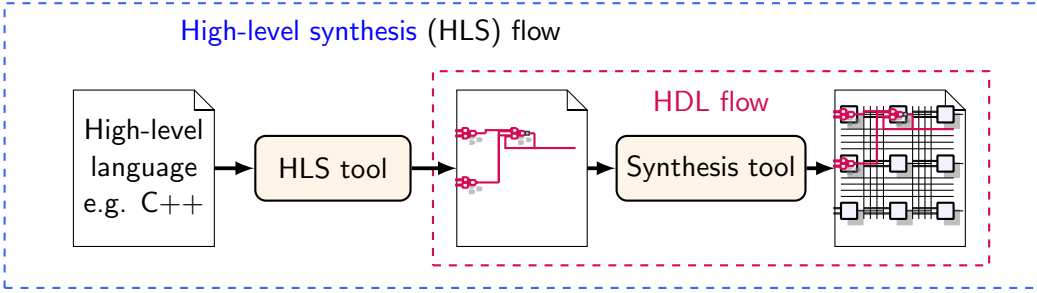
Real Programmers needed



- You don't program, you design a circuit
  - in fancy Hardware Description Languages (HDL) such as VHDL or Verilog
  - with compilers called "synthesis tools" that can take hours:

1-week compilation time for a large FPGA...

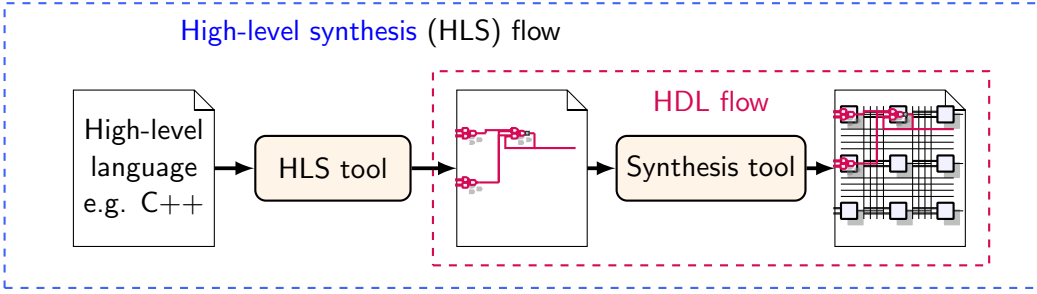
Real Programmers needed



- You don't program, you design a circuit
  - in fancy Hardware Description Languages (HDL) such as VHDL or Verilog
  - with compilers called "synthesis tools" that can take hours:

    1-week compilation time for a large FPGA...
- Since the 2010, FPGA programming in C/C++ for the rest of us
  - higher productivity, but... no escaping the synthesis tools backend.

Real Programmers needed



- You don't program, you design a circuit
  - in fancy Hardware Description Languages (HDL) such as VHDL or Verilog
  - with compilers called "synthesis tools" that can take hours:

    1-week compilation time for a large FPGA...
- Since the 2010, FPGA programming in C/C++ for the rest of us
  - higher productivity, but... no escaping the synthesis tools backend.

Haha, C++ programming is called "high level"...

If you have ever touched Cuda or OpenCL, you know what comes next

From a state of the art matrix-matrix product:

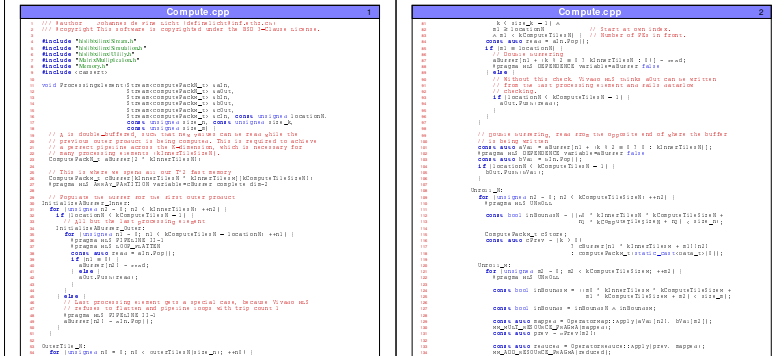
Johannes de Fine Licht, Grzegorz Kwasniewski, and Torsten Hoefler.
Flexible communication avoiding matrix multiplication on FPGA with high-level synthesis.
In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 244–254, 2020.

```
1 for (i = 0; i < M; i++)
2     for (j = 0; j < N; j++)
3         for (k = 0; k < K; k++)
4             C[i, j] = C[i, j] + A[i, k]*B[k, j];
```

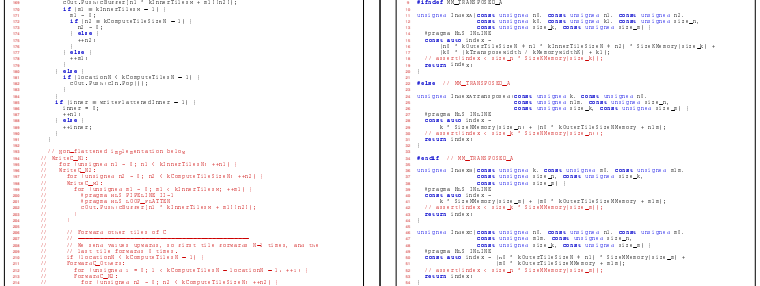
Listing 1: Classical MMM algorithm.

```
1  // Memory tiles m
2  for (i_m = 1; i_m ≤ n; i_m = i_m + x_tot)
3    for (j_m = 1; j_m ≤ m; j_m = j_m + y_tot)
4      for (k = 1; k ≤ k; k = k + 1) // Full dimension k
5  // [Sequential] Block tiles b in memory tile
6        for (i_b = i_m; i_b ≤ i_m + x_tot; i_m = i_m + x_t x_c x_p)
7          for (j_b = j_m; j_b ≤ j_m + y_tot; j_m = j_m + y_t y_p y_c)
8  // [Sequential] Compute tiles t in block tile
9            for (i_t = i_b; i_t ≤ i_b + x_t x_p x_c; i_b = i_b + x_c x_p)
10             for (j_t = y_b; j_t ≤ j_b + y_t y_p y_c; j_t = j_t + y_c y_p)
11 // [Parallel] Processing elements p in compute tile
12               forall (i_p = i_t; i_p ≤ i_t + x_p x_c; i_t = i_t + x_c)
13                 forall (j_p = j_t; j_p ≤ j_t + y_p y_c; j_p = j_p + y_c)
14 // [Parallel] Compute units c in processing element
15                   forall (i_c = i_p; i_c ≤ i_p + x_c; i_c = i_c + 1)
16                     forall (j_c = j_p; j_c ≤ j_p + y_c; j_c = j_c+1)
17                       C(i_c, j_c) = C(i_c, j_c) + A(i_c, k) · B(k, j_c)
```

Listing 2: Pseudocode of the tiled MMM algorithm.



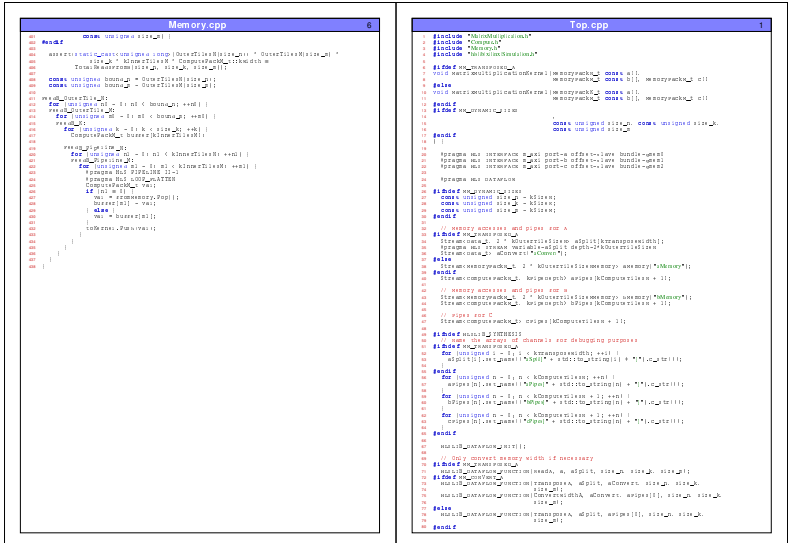
Actual listing (1)



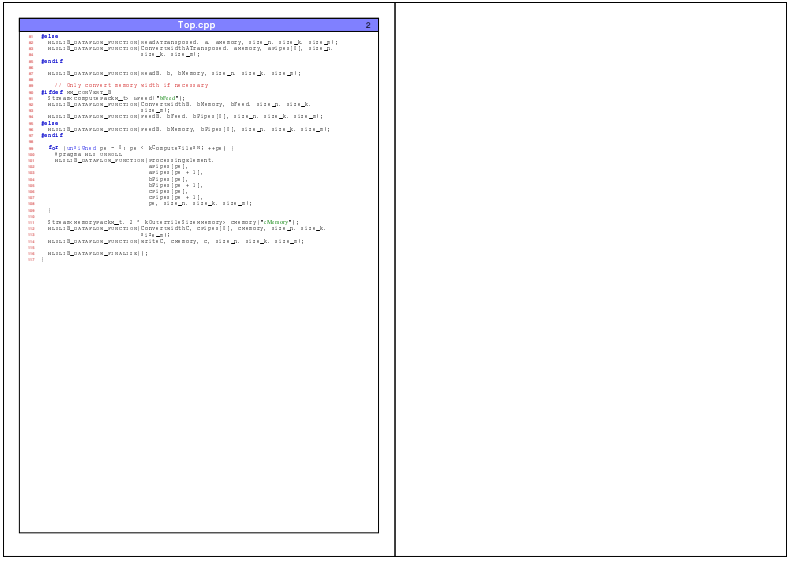
Actual listing (2)

Actual listing (3)

Actual listing (4)



Actual listing (5)



Actual listing (6)

Linpack performance of a large FPGA

For 32-bit floating-point:

- 409 GFlop/s (10 GFlop/J),
- (other works report 800 GFlop/s)
- The processor of my laptop: about 100 GFlop/s (3 GFlop/J)
- Top-end GPUs are above 1000 GFlop/s peak

Linpack performance of a large FPGA

For 32-bit floating-point:

- 409 GFlop/s (10 GFlop/J),
- (other works report 800 GFlop/s)
- The processor of my laptop: about 100 GFlop/s (3 GFlop/J)
- Top-end GPUs are above 1000 GFlop/s peak

Not bad considering that the FPGA runs below 500MHz, but...
you are a bit disappointed, aren't you ?

If you lose according to a metric, change the metric.

Ideed, processors and GPUs are engineered to be good at FP32 and FP64 Linpack Flops.

If you lose according to a metric, change the metric.

Ideed, processors and GPUs are engineered to be good at FP32 and FP64 Linpack Flops.

Peak marketing lies for double-precision floating-point exponential

(Intel MKL vector libm, versus FloPoCo FPExp)

- Processor core: 20 cycles / DPExp @ 4GHz: **200 MDPExp/s**
- FPExp in FPGA: 1 DPExp/cycle @ 400MHz: **400 MDPExp/s**
- Chip vs chip: 20 cores/processor vs 200 FPExp/FPGA: **FPGA 20 times better**
- Power consumption improvement even higher
- Single precision data also better

Florent de Dinechin and Bogdan Pasca.
Floating-point exponential functions for DSP-enabled FPGAs.
In *Field Programmable Technologies*, 2010.

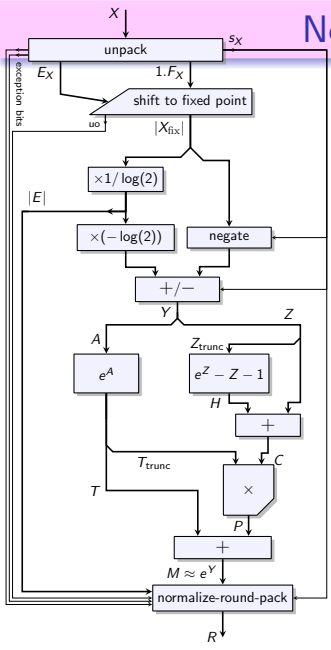
Dura Amdahl lex, sed lex

SPICE Model Evaluation

Models	Instruction Distribution					
	Add	Mult.	Div.	Sqrt.	Exp.	Log
bjt	22	30	17	0	2	0
diode	7	5	4	0	1	2
hbt	112	57	51	0	23	18
jfet	13	31	2	0	2	0
mos1	24	36	7	1	0	0
vbic	36	43	18	1	10	4

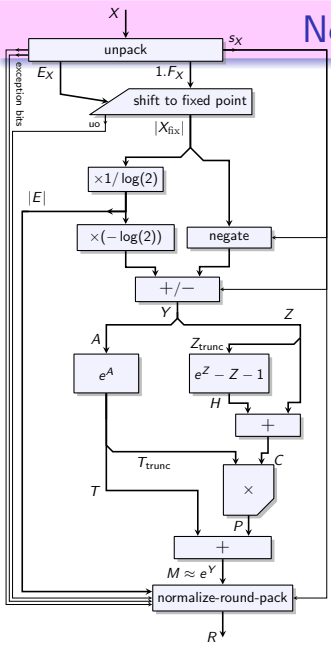
Nachiket Kapre and Andre DeHon
Accelerating SPICE Model-Evaluation using FPGAs.
In *Field-Programmable Custom Computing Machines*, 2009.

Not your processor's exponential



Why is it 2x faster when it should be 10x slower ?

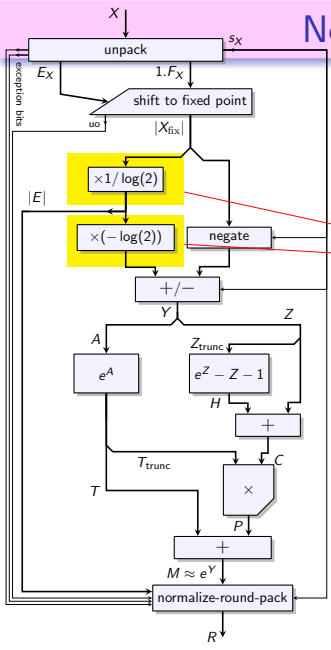
Not your processor's exponential



Why is it 2x faster when it should be 10x slower ?

- data-flow architecture (no von Neuman tax)

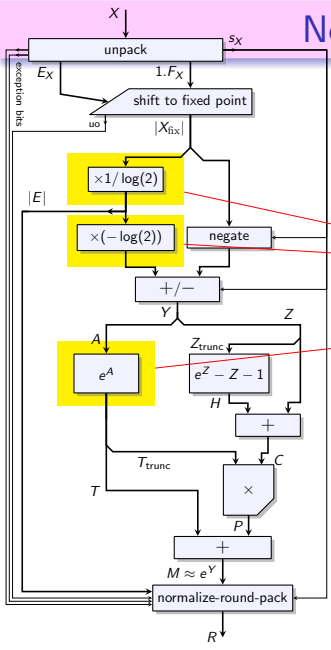
Not your processor's exponential



Why is it 2x faster when it should be 10x slower ?

- data-flow architecture (no von Neuman tax)
- application-specific constant multipliers

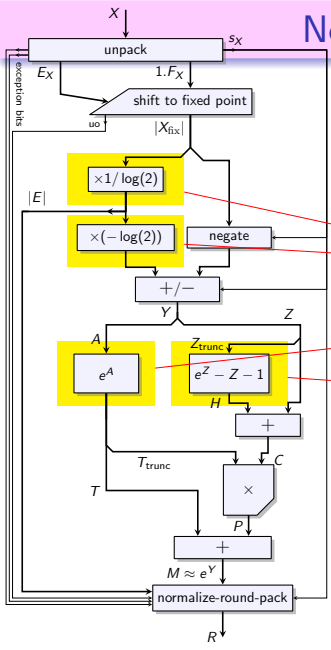
Not your processor's exponential



Why is it 2x faster when it should be 10x slower ?

- data-flow architecture (no von Neuman tax)
- application-specific constant multipliers
- tables of pre-computed values

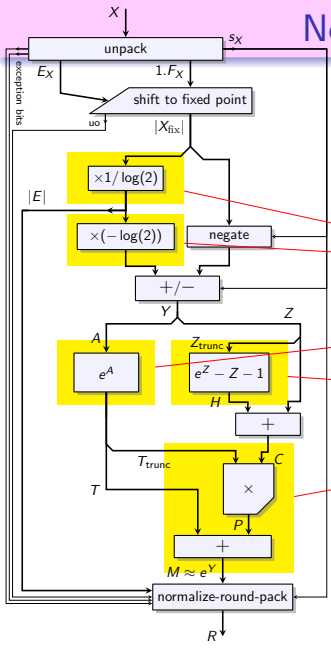
Not your processor's exponential



Why is it 2x faster when it should be 10x slower ?

- data-flow architecture (no von Neuman tax)
- application-specific constant multipliers
- tables of pre-computed values
- arbitrary function evaluator

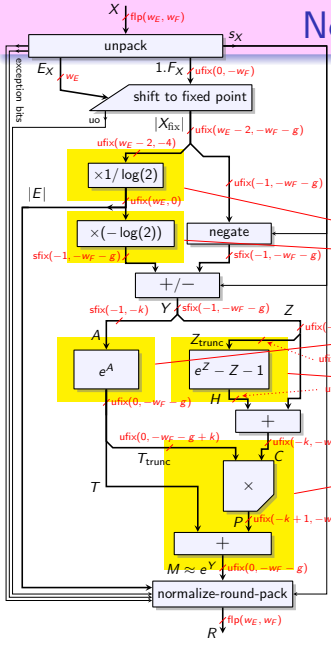
Not your processor's exponential



Why is it 2x faster when it should be 10x slower ?

- data-flow architecture (no von Neuman tax)
- application-specific constant multipliers
- tables of pre-computed values
- arbitrary function evaluator
- matching DSP blocks

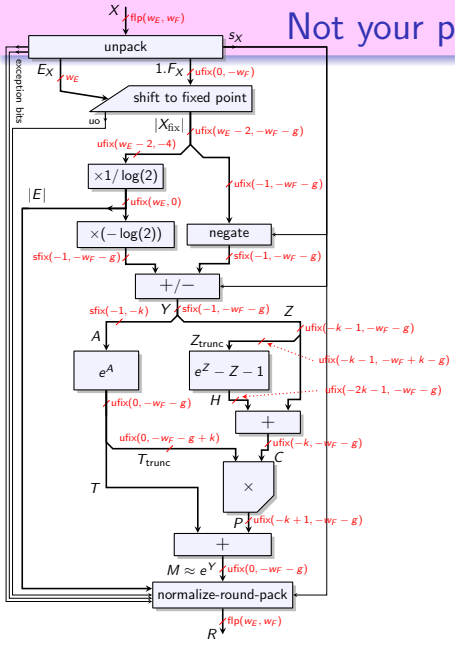
Not your processor's exponential



Why is it 2x faster when it should be 10x slower ?

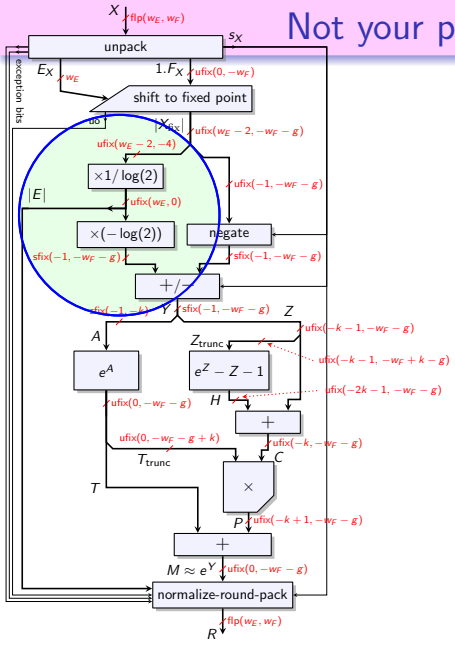
- data-flow architecture (no von Neuman tax)
- application-specific constant multipliers
- tables of pre-computed values
- arbitrary function evaluator
- matching DSP blocks
- only compute useful bits

Not your processor's multipliers, either



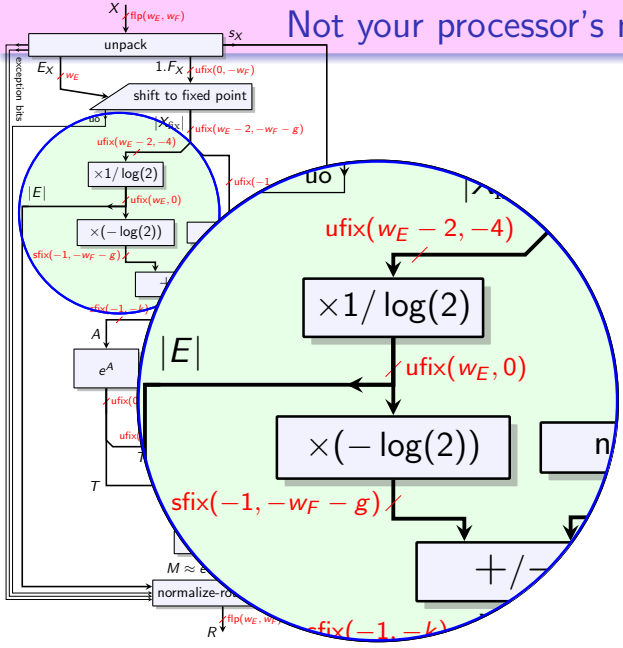
- only compute useful bits

Not your processor's multipliers, either



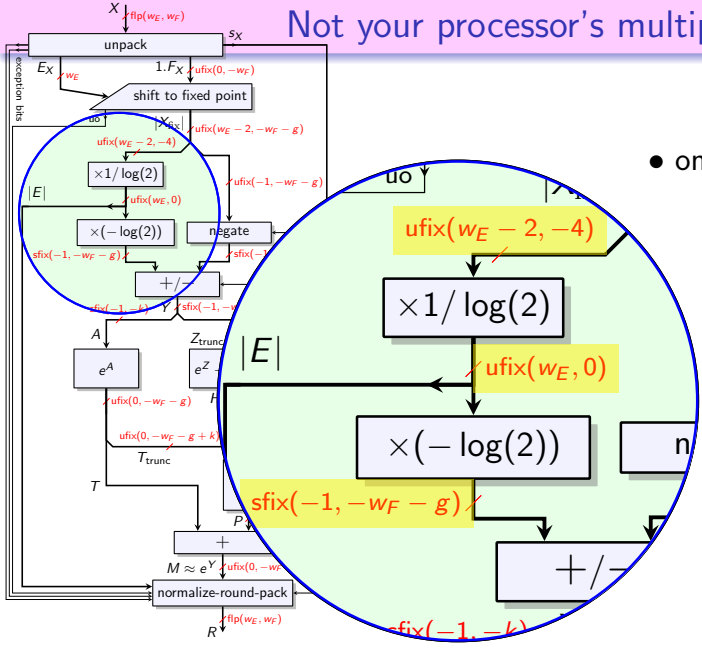
- only compute useful bits

Not your processor's multipliers, either



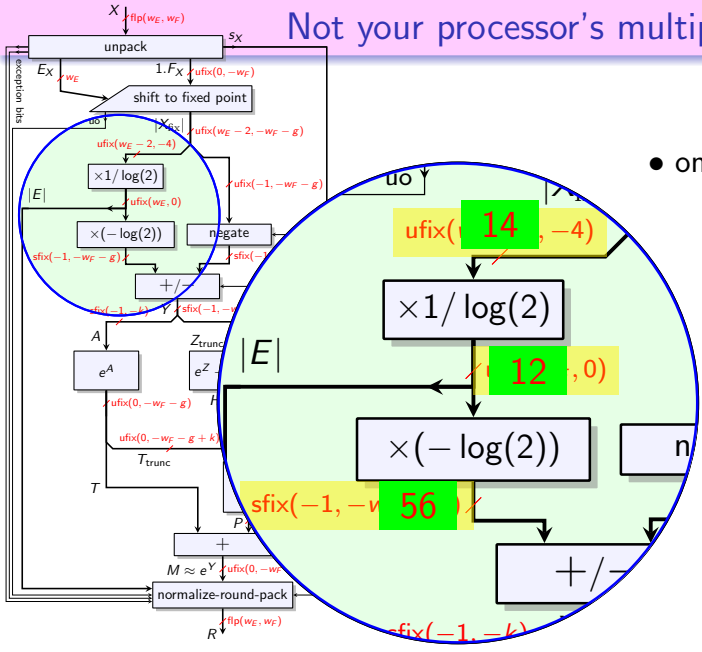
- only compute useful bits

Not your processor's multipliers, either



- only compute useful bits
  - depending on
    exponent size w_E
    and mantissa size w_F

Not your processor's multipliers, either



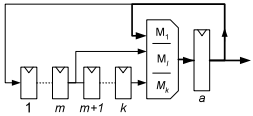
- only compute useful bits
  - depending on
    exponent size $w_E = 11$
    and mantissa size $w_F = 52$
  - compare to FP64
    12 bits: too small
    56 bits: too large

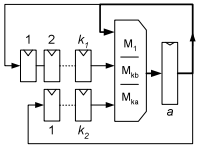
A few FPGA success stories

- A snapshot from Xilinx' HPC page (when it was still Xilinx):

- You find the same keywords on Intel FPGA's pages
- and on Maxeler Technologies' (a company providing computation acceleration service)
- among others ...

Monte Carlo simulation





- uniform random bits are cheap as chips on FPGAs
  - LFSRs are a CPU thing
  - generalize them to several parallel shift registers
  - several random bits in parallel from a single state
  - high-quality randomness if you get the math right

David B. Thomas and Wayne Luk.
FPGA-Optimised High-Quality Uniform Random Number Generators
In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2008.

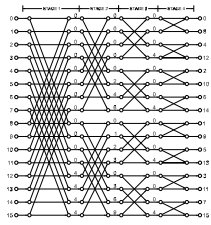
- FPGAs also much better at non-uniform (Gaussian etc) than CPUs or GPUs

David B. Thomas, Lee Howes, and Wayne Luk.
A Comparison of CPUs, GPUs, FPGAs, and Massively Parallel Processor Arrays for Random Number Generation
In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2009.

All this was developped for antiscientific computing (high-speed trading), but still...

Signal processing

Mario Garrido, Konrad Möller, and Martin Kumm.
World's Fastest FFT Architectures: Breaking the Barrier of 100 GS/s.
IEEE Transactions on Circuits and Systems I, 66(4):1507–1516, 2019.



- Fully unrolled FFT (up to 256 points)
  - i.e. inputting 256 complex values per cycle, at 500 MHz
  - well above 10 TOp/s if you count all additions and multiplications
- 16-bit in/out, wider datapath inside
- Look, Ma: no multiplier !
  - each multiplier expanded as an adder graph (and optimally so)
  - ... leaving the 1800 DSP blocks free for other things.
- about 1/5th of LUT + registers of the target device (Virtex UltraScale 190)

As previously, a good start is not to imitate the processor solution.

Artificial intelligence and machine learning

So many papers these days, just pick one extreme:

Adrien Prost-Boucle, Alban Bourge, and Frédéric Pétrot.
High-efficiency convolutional ternary neural networks with custom adder trees and weight compression.
ACM Transactions on Reconfigurable Technologies and Systems, 11(3), dec 2018.

- Ternary logic: weight and activations $\in \{-1, 0, 1\}$
- Specific network retraining
- (and a few more layers to reach comparable accuracy)
- All the weights fit in the FPGA RAM blocks
- Embedded HPC on small FPGAs

Graph-based big data analytics

TigerGraph claims 100× acceleration of Cosine Similarity in one FPGA+HBM, compared to a dual 24-core Xeon server.

- memory-intensive kernels on GBs of data
- the expensive cache hierarchy of the processor is no good here

Conclusion

FPGA programming for the rest of us

OK, I'm convinced, I want to try to program an FPGA, where do I start?

- Amazon, Microsoft, OVH,... all offer FPGA resources on the cloud
- ... sometimes with pre-packaged FPGA-ware.
- Intel and Xilinx sell accelerator cards
- also many entry-level boards to play with
- HLS definitely more welcoming than Hardware Description Languages
  - don't believe that you'll get good results without understanding the FPGA architecture
  - don't believe that you'll get good results without tweaking

A quote from the random generation paper:

The surprising result is that
each platform requires a different approach to random number generation

A quote from the random generation paper:

The ~~surprising~~ result is that
each platform requires a different approach to random number generation

I hope you appreciate now that it is not surprising.

A quote from the random generation paper:
$\forall X$ The ~~surprising~~ result is that
each platform requires a different approach to X

I hope you appreciate now that it is not surprising.

... and I invite you to suspect that this is true also for your computation

When do we get an FPGA in every PC ?

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)

When do we get an FPGA in every PC ?

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)

Dinechin's Theorem: we'll wait forever.

When do we get an FPGA in every PC ?

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)

Dinechin's Theorem: we'll wait forever.

Proof: Assume that such an application pops up.

- either it is indeed widely useful
  - then next year's processor 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 then it means that it was not a killer app.

When do we get an FPGA in every PC ?

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)

Dinechin's Theorem: we'll wait forever.

Proof: Assume that such an application pops up.

- either it is indeed widely useful
  - then next year's processor 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 then it means that it was not a killer app.

The killer feature of FPGAs is their fine-grain programmability.

When do we get an FPGA in every PC ?

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)

Dinechin's Theorem: we'll wait forever.

Proof: Assume that such an application pops up.

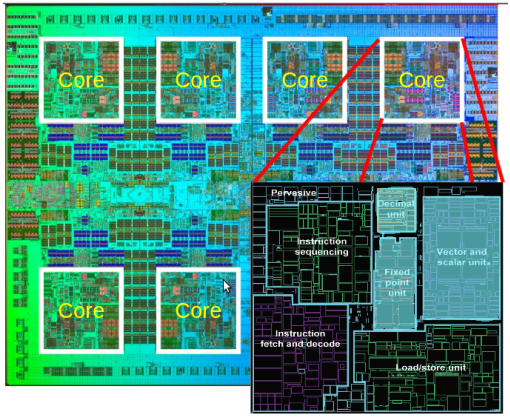
- either it is indeed widely useful
  - then next year's processor will do it in hardware 10x faster than the FPGA,
  - so it won't be an *FPGA* killer app next year...
- or the FPGA remains competitive next year
  - but then it means that it was not a killer app.

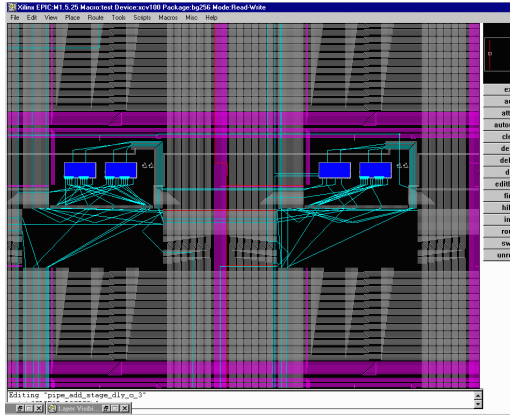
The killer feature of FPGAs is their fine-grain programmability.

Constraints in classical programming become degrees of freedom in FPGAs

Back to the war of the programming models

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





Which is best for (insert your computation here) ?