FPGAs

To read more...

This day's papers:

Brown and Rose, "Architecture of FPGAs and CPLDs: A Tutorial". (no review required)

Putnam et al, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services"

reconfigurable hardware

'normal' processor	reconfig. HW
stream of instructions	set of wirings
fetch 1+ instruction/cycle	milliseconds+ to reconfigure
lots of control logic	lots of routing
fixed, fast functional units	flexible, slower functional units

the accelerator concept

second processor specialized for particular computation

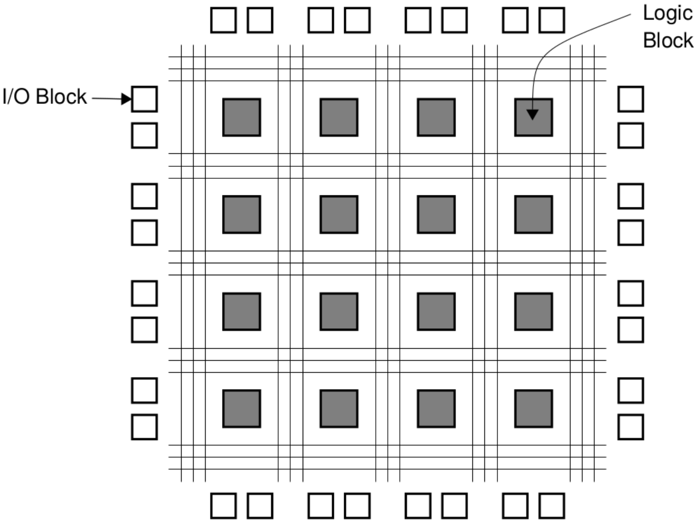
examples:

GPUs — vector computations

FPGAs — ???

custom chips — ??? (next week)

FPGA structure



Brown and Rose, Figure 2

FPGA programs: RTL

e.g.: Verilog

determines wiring between gates, registers, memories

everything happens in parallel every cycle

manually specify what's in registers, etc.

same languages used to design real processors

RTL example

```
module counter(clock,reset,value);
  input clock;
  input reset;
  output value;

  reg [32:0] count;

  always @ (posedge reset or posedge clock)
    if (reset)
      begin
        count <= 0;
      end
    else
      begin
        count <= count + 1'b1;
      end

  assign value = count;
endmodule
```

A note about HW programming

not intuitive

attempts at easier interfaces:

"schematic capture" — draw circuit diagram
- common, doesn't seem great at scale

higher-level tools, e.g., Chisel (Berkeley research project)
- compile to RTL; used at scale

automatic translation of C-like language (C to gates)
- Very mixed reputation — very hard compilers problem
- But see Aladdin paper

FPGA design pipeline

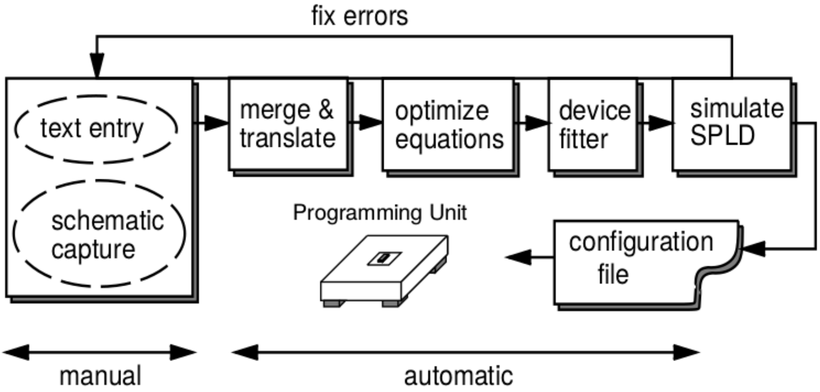


Figure 7 - *CAD Design Flow for SPLDs.*

Brown and Rose, Figure 7

FPGA: place and route

RTL compiles to “gate list”

needs to turn into what components in the FPGA to connect

not straightforward; hours+ to compute if FPGA nearly full

effects performance — longer wires/more switches

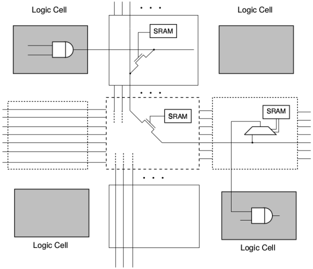
Programmable switches: example

Example switch: transistor + SRAM cell

(SRAM cell ≈ 1-bit register)

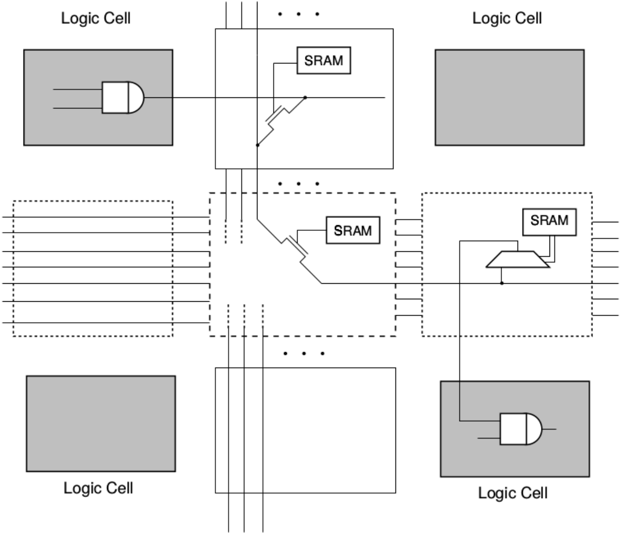
SRAM cell continously outputs stored value

can be written by seperate circuit (not shown)



Brown and Rose, Figure 5

Programmable switches: example



Brown and Rose, Figure 5

FPGA routing example

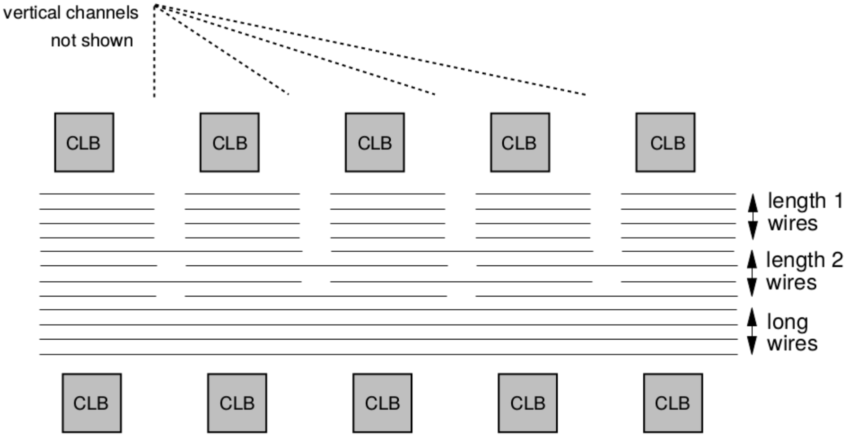
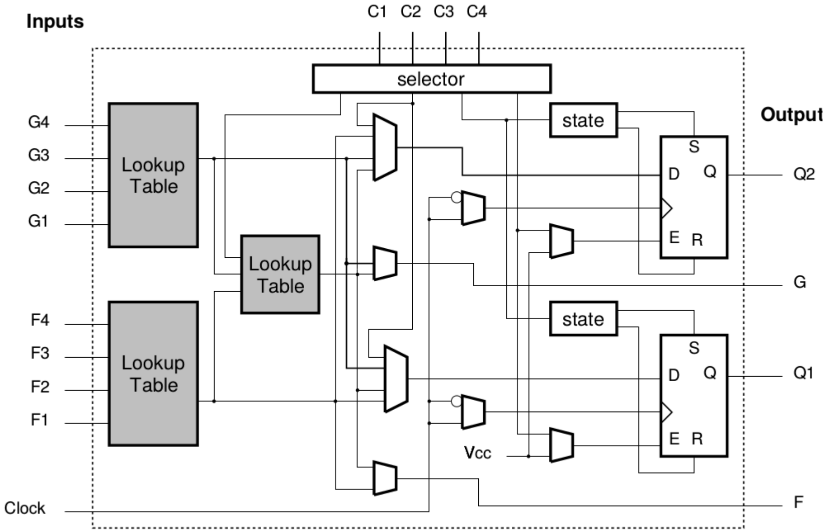


Figure 19 - *Xilinx XC4000 Wire Segments.*

FPGA logic block example (1)



FPGA logic block example (2)

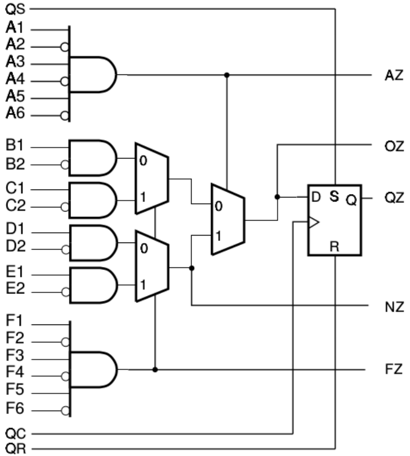


Figure 28 - *Quicklogic (Cypress) Logic Cell.*

FPGA configuration

what to do for every switch

just loading values into memory that controls switch

FPGA efficiency

most transistors perform routing, not computation

much longer signal paths than in CPUs

- slower clock rates for same task

development tool usefulness/quality is not great

FPGA: more complex logic

many FPGAs include specialized fixed functionality

- RAM
- adders, multipliers
- floating point units
- common DSP computations
- full embedded-class CPU cores
- …

could implement these using fully programmable logic

- but slower/bigger

review comments

what are FPGAs good for anyways?

versus/combined with GPUs/CPUs?

other large-scale deployments?

programmability?

Catapult challenges

datacenter logistics
- cost (only 10% more???)
- power density (cooling, power distribution)
- physical space

programs across multiple FPGAs
- needs fast FPGA-to-FPGA communication
- centralized allocation

failure handling

The Shell

23% of FPGA (configurable) area:

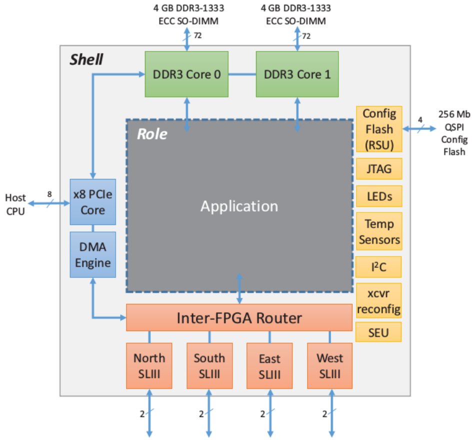


Figure 3: Components of the Shell Architecture.

CPU to FPGA transfers

$10\ \mu$s for 16 KB — approx 15 GB/s

(about maximum PCIe 3.0 transfer rate)

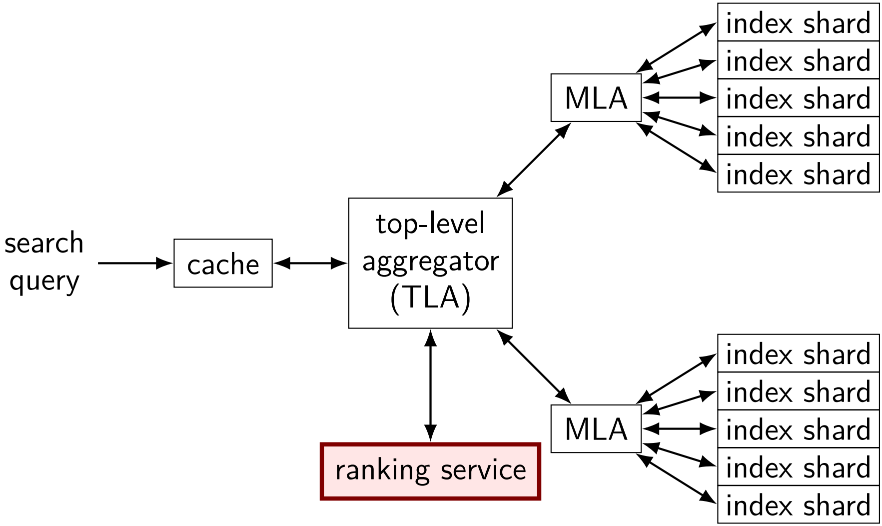
Catapult roles

hand-coded Verilog (RTL language)

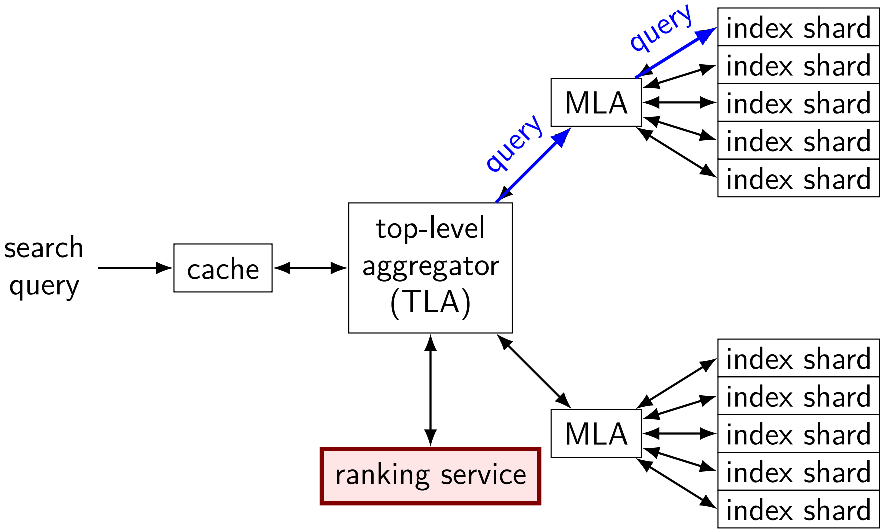
hand partitioned across FPGAs?

precise duplication of existing software

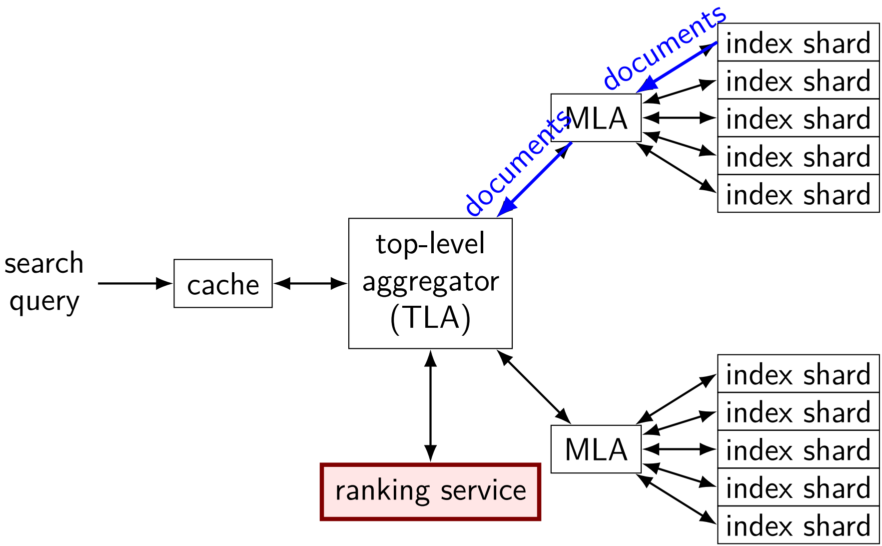
Search engine architecture



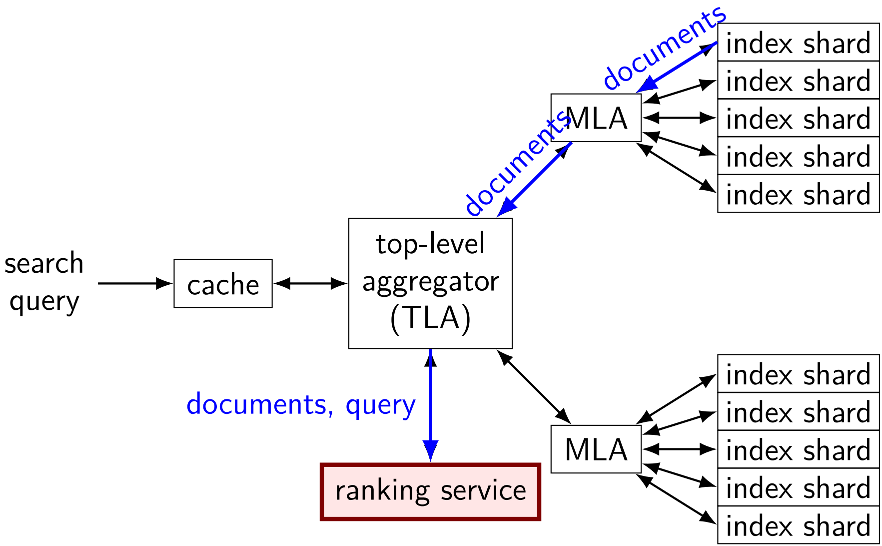
Search engine architecture



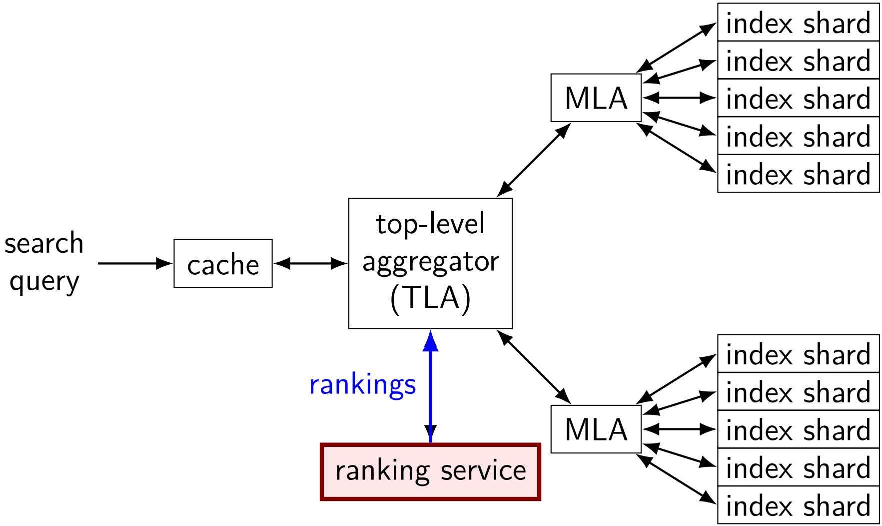
Search engine architecture



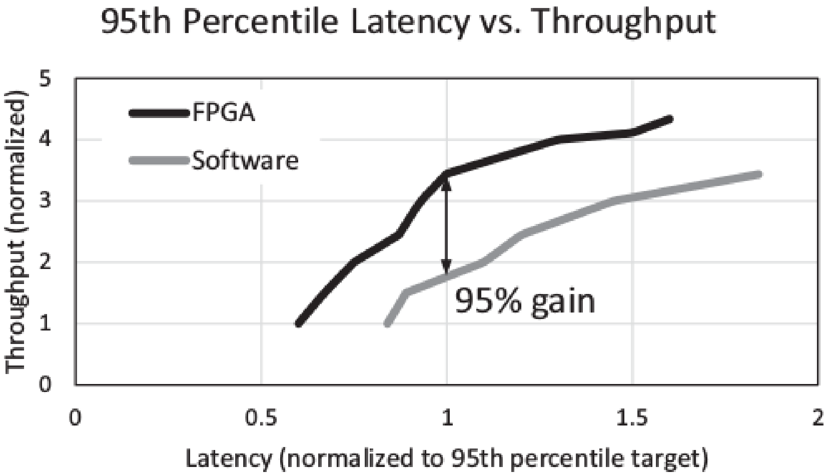
Search engine architecture



Search engine architecture



Overall Motivation



FPGA operation

recieve: document, some features via shared memory

output: score

each FPGA runs a macropipeline stage — 8 μs (1600 clock cycles)

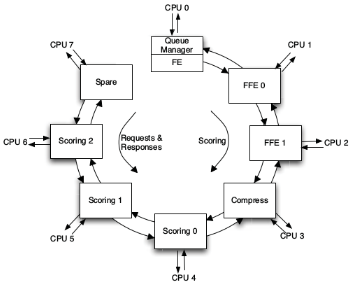


Figure 5: Mapping of ranking roles to FPGAs on the reconfigurable fabric.

Queue Manager

“model reload”

can only store one model at a time — takes 250 μs to load from external RAM

on FPGA memories: approx. 40MB capacity (distributed)

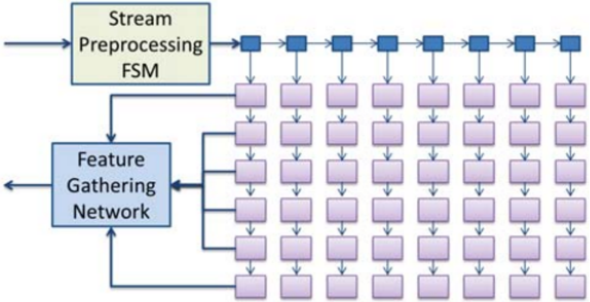
trick: proess queries for same model together

Feature Extraction FSMs

parallel finite-state machines

essentially regexes compiled to gates?

fully pipelined



Feature Extraction FSMs

Feature Expressions

speialized mathematical expressions

custom multithreaded processor

model determines what the expressions are

mostly integer — small FPGA area — but some FP

split across multiple FPGAs

threads priority-scheduled

“Complex” logic area

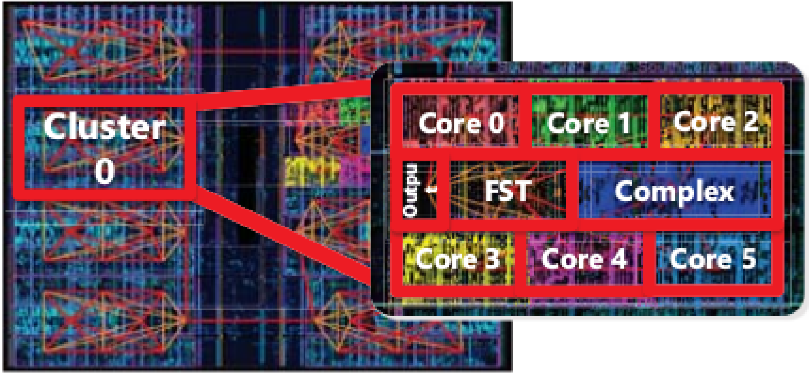


Figure 7: FFE Placed-and-Routed on FPGA.

What are FPGAs good for?

bit-twiddling (lots of simple CPU instrs.)?

inherently parallel programs?

perhaps even if different operations — hard for GPUs

low-latency I/O interface and processing?

prototyping CPUs, GPUs

What are FPGAs bad at?

floating point, other ‘big’ arithmetic operations

- purpose-built, denser ALUs just win

caching lots of data?

- ... but sometimes dedicated SRAM blocks

being easy to program well

- programming FPGAs $\approx$ processor design!

FPGAs versus GPUs

both good at doing massively parallel computations

FPGAs better at exploiting multiple instruction parallelism?

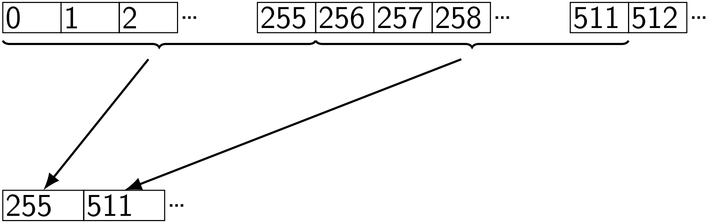
FPGAs can be lower latency for simple operations

FPGAs much worse at floating point/non-small-integer calculations?

Interlude: Homework 3

Homework 3 supplied kernel

what does the supplied kernel do?



Exam topics

Memory hierarchy — caches, TLBs

Pipelining, instruction scheduling, VLIW

Multiple issue/out-of-order:

- register renaming and reservation stations
- reorder buffers and branch prediction
- hardware multithreading

Multicore shared memory:

- cache coherency protocols/networks
- relaxed memory models and sequential consistency
- synchronization: spin locks, transaction memory, etc.

Vector machines, GPUs, other accelerators

Next time: Custom ASICs

higher dev cost/higher efficiency

two papers:

- one on: automating design of custom ASIC accelerators (Aladdin)
- another: a case study using that (Minerva)

all these things probably apply to FPGA stuff

Preview: Minerva

Deep Neural Networks — machine learning models

accelerating evaluating DNNs (making predictions from a pre-trained model)

mathematical tradeoffs (remove “unimportant” things from model)

architectural tradeoffs

Previre: Aladdin

Tool (used by Minerva) for quickly evaluating accelerator designs

Produces fast estimates

Complements existing high-level synthesis ("C to gates"-like) tools