Adventures with FPGAs

Create your own microprocessor at home !

Bruno.Levy@inria.fr

inria

RISC-V AMBASSADOR

Google Open Source Peer Bonus

Links

> **My homepage:** *brunolevy.github.io*

> **Learn-fpga project:**
https://github.com/BrunoLevy/learn-fpga/

Designing your own processor, why ?

> **Research**:

Invent and test new architectures

arithmetics, encryption, energy, OS research

Real-scale testbed for formal methods

Designing your own processor, why ?

- **Research**:
  - *Invent and test new architectures*
    - arithmetics, encryption, energy, OS research
  - *Real-scale testbed for formal methods*

- **Curiosity:** you want to have a deeper understanding of how things work

Designing your own processor, why ?

- **Research**:
  - *Invent and test new architectures*
    - arithmetics, encryption, energy, OS research
  - *Real-scale testbed for formal methods*

- **Curiosity:** you want to have a deeper understanding of how things work

- **Opportunity:** you feel it is something that is going to be important in a near future

Designing your own processor, why ?

> **Research**:

Invent and test new architectures

arithmetics, encryption, energy, OS research

Real-scale testbed for formal methods

> **Curiosity**

understar

> **Opportu**

to be imp

What is the most important industrial actor of AI ?

Designing your own processor, why ?

> **Research**:

Invent and test new architectures

arithmetics, encryption, energy, OS research

Real-scale testbed for formal methods

> **Curiosity**

understar

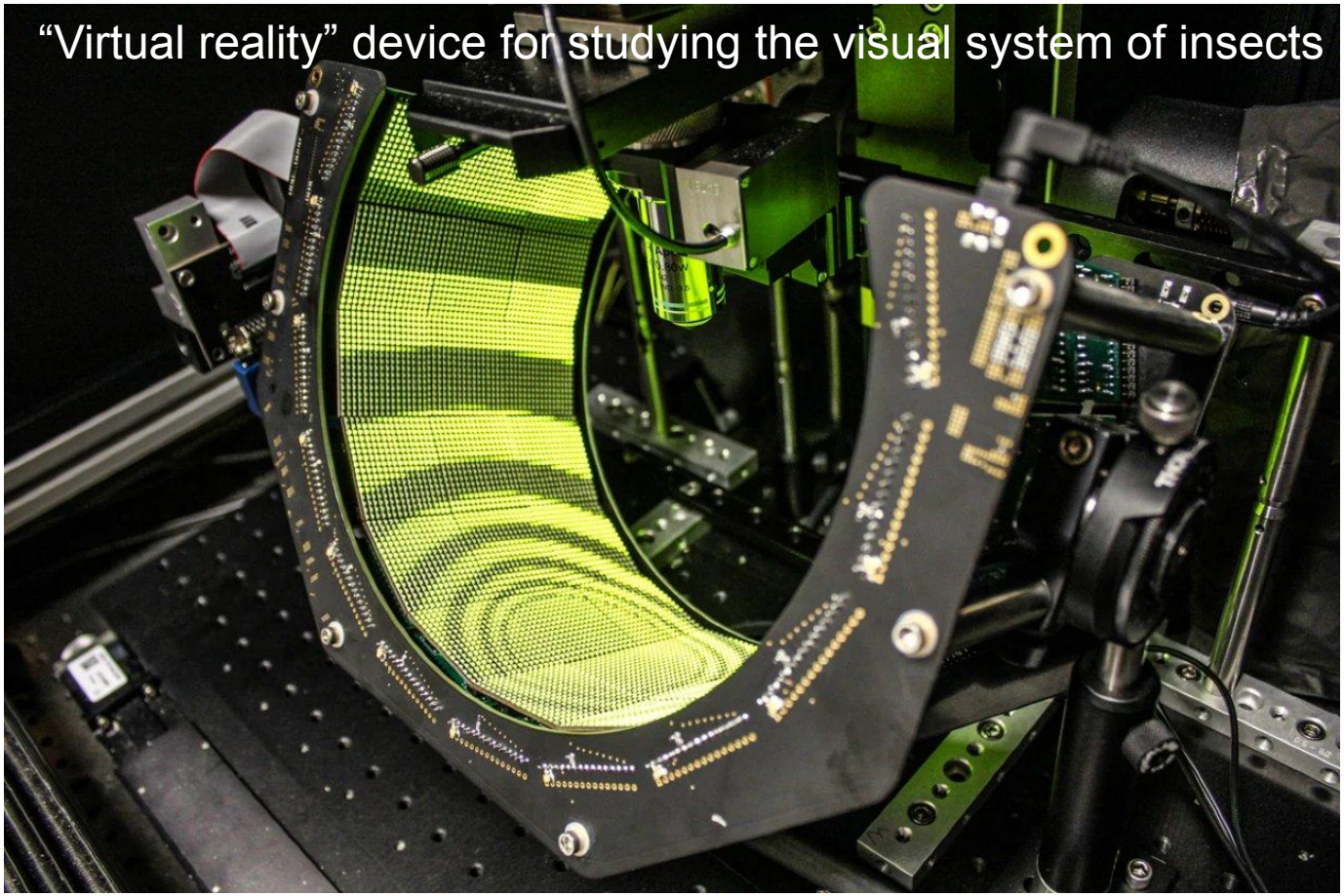
> **Opportu**

to be imp

Nvidia:
1000 billions !!!

Designing your own processor, why ?

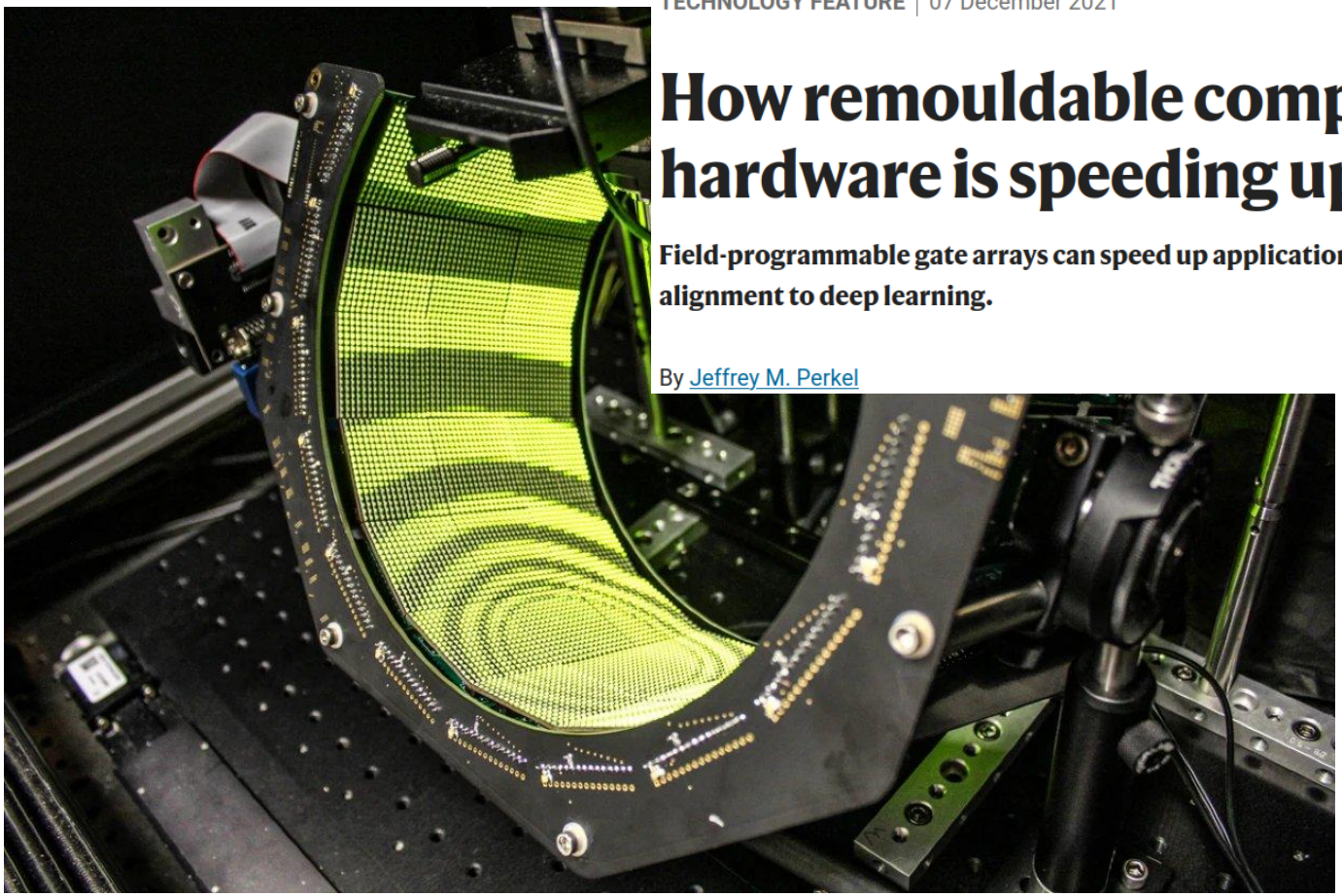
Very special needs



nature

Explore content | About the journal | Publish with us | Subscribe

nature > technology features > article



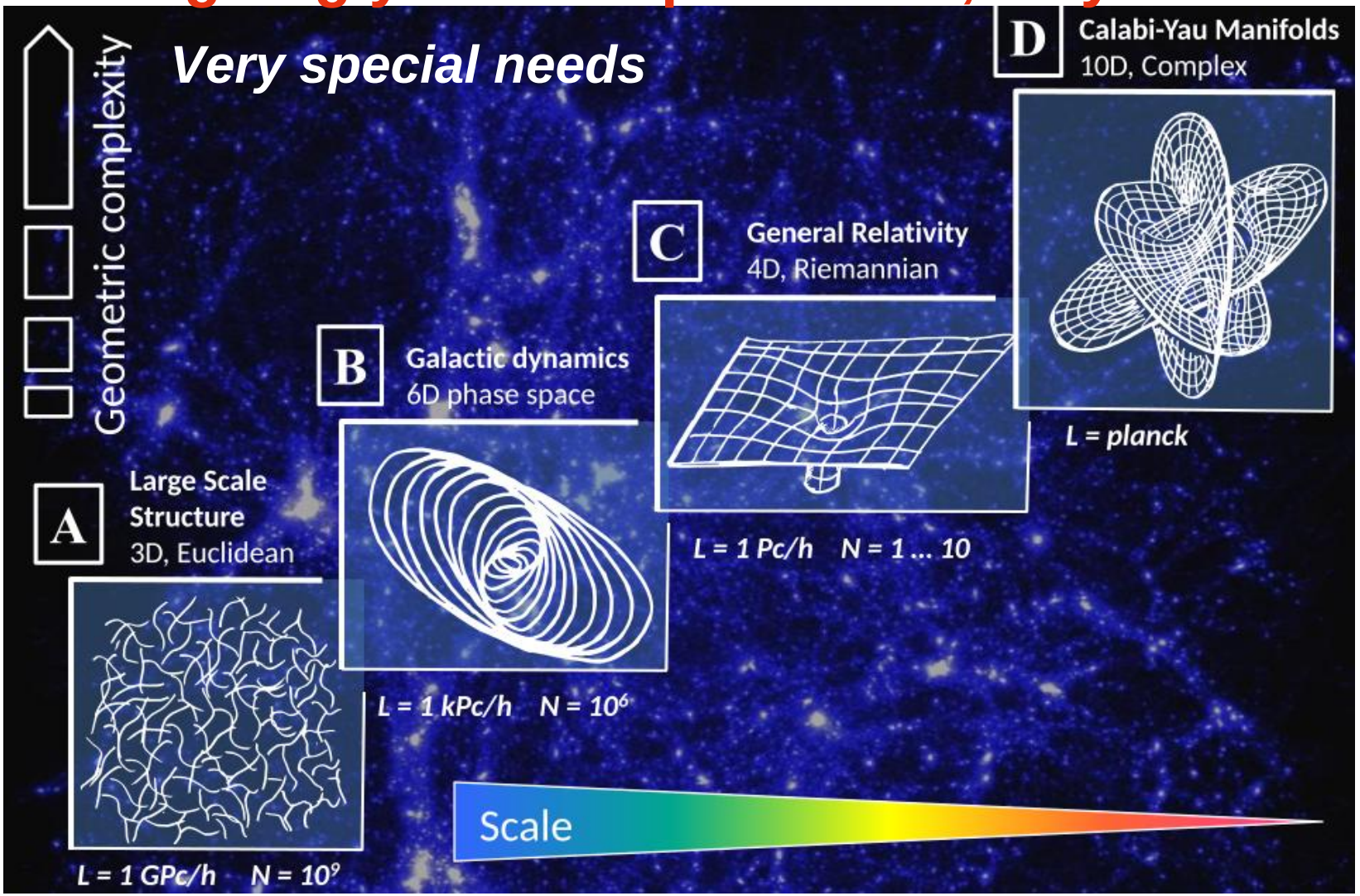
TECHNOLOGY FEATURE | 07 December 2021

How remouldable computer hardware is speeding up science

Field-programmable gate arrays can speed up applications ranging from genomic alignment to deep learning.

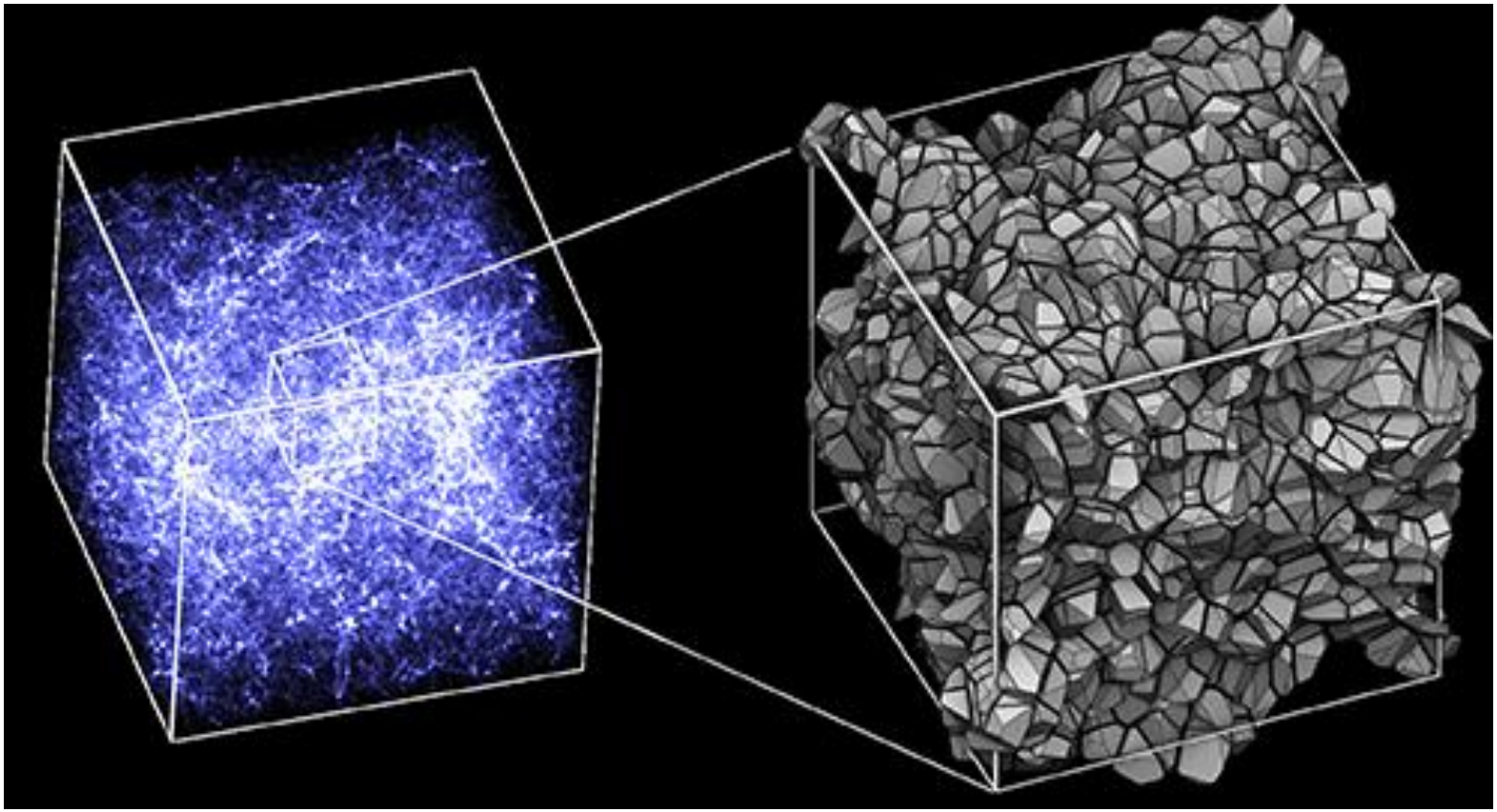
By Jeffrey M. Perkel

Designing your own processor, why ?



Designing your own processor, why ?

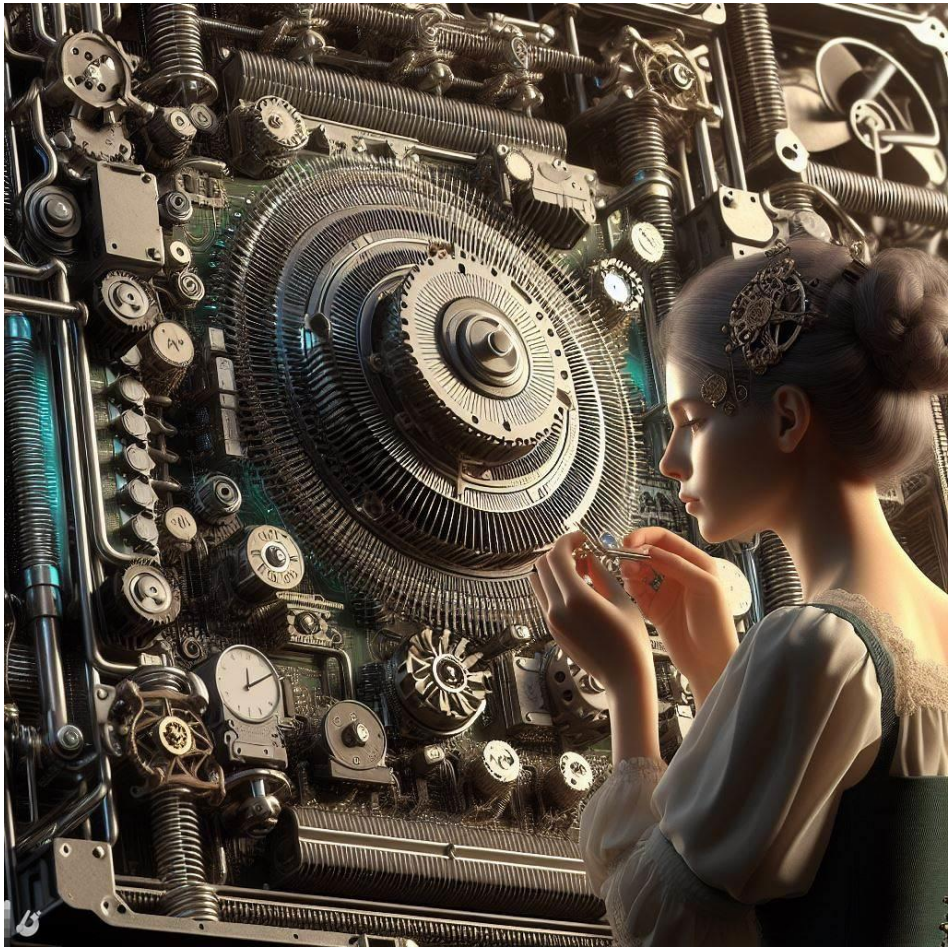
Very special needs: the cosmological computer



Designing your own processor, how ?

> **Invent idealized mechanism, Program it.**

(Babbage, Lovelace)



Designing your own processor, how ?

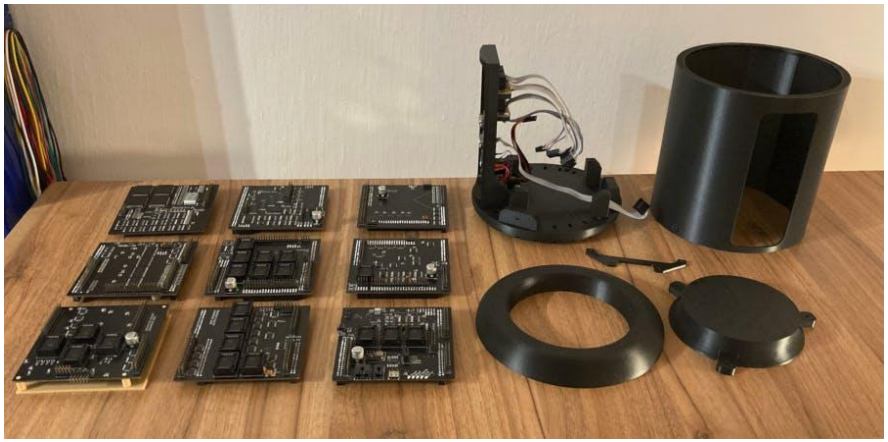
> **Create a processor in Minecraft ?**

Designing your own processor, how ?

> **Create a processor in Minecraft ?**

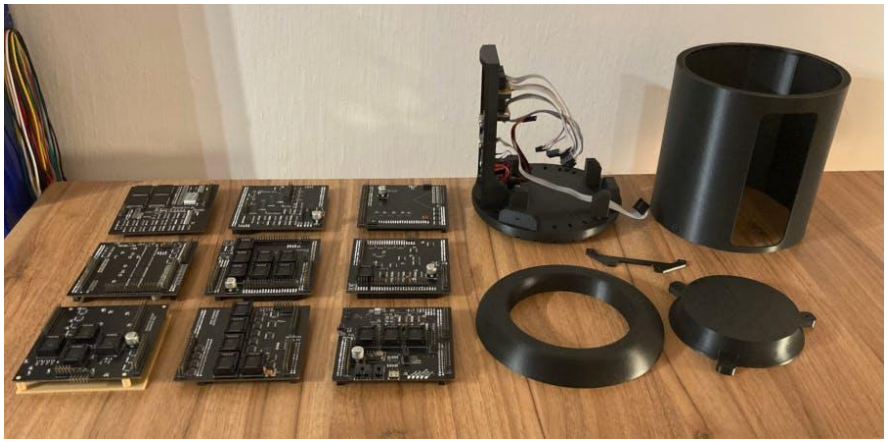
Designing your own processor, how ?

> **Build out of discrete components ?**



Designing your own processor, how ?

> Build out of discrete components ?



PineAppleOne – RISC-V
$$$$, slow, difficult

Designing your own processor, how ?

> **You have access to a foundry** ?

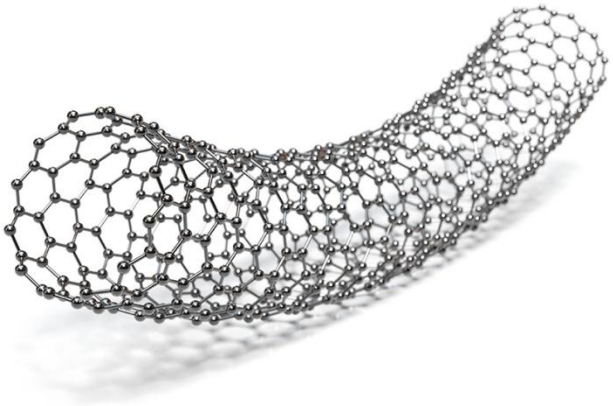
Designing your own processor, how ?

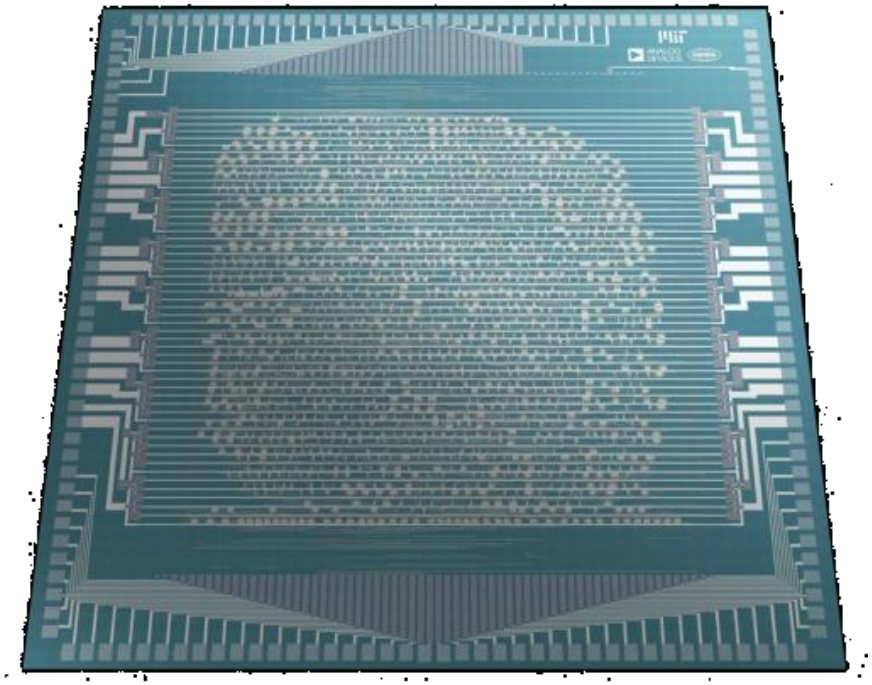
> **You have access to a foundry** ?

Commercial: < 100K unit, forget it
Research: possibilities, using old process (SkyWaters, …), but $$$

Designing your own processor, how ?

> **RiscV with exotic transistors ?**



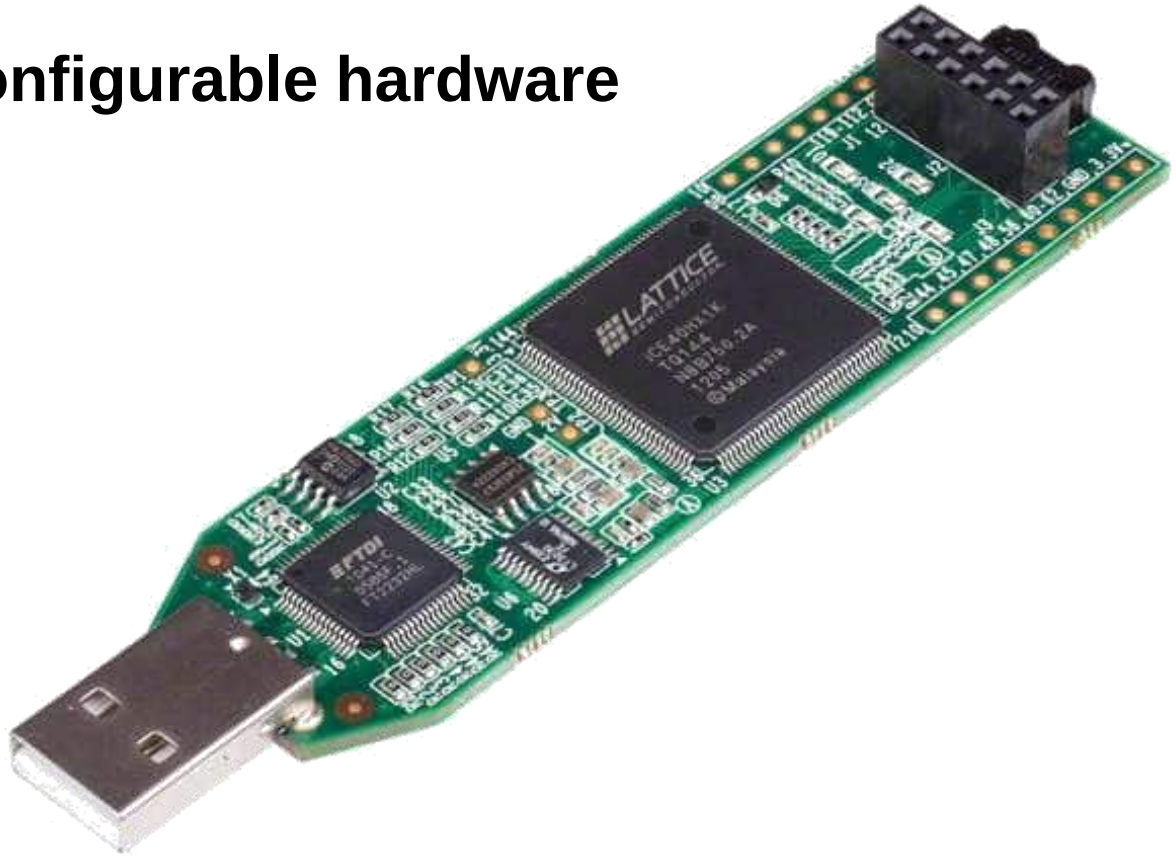


Designing your own processor, how ?

> **Create a quantum computers ?**

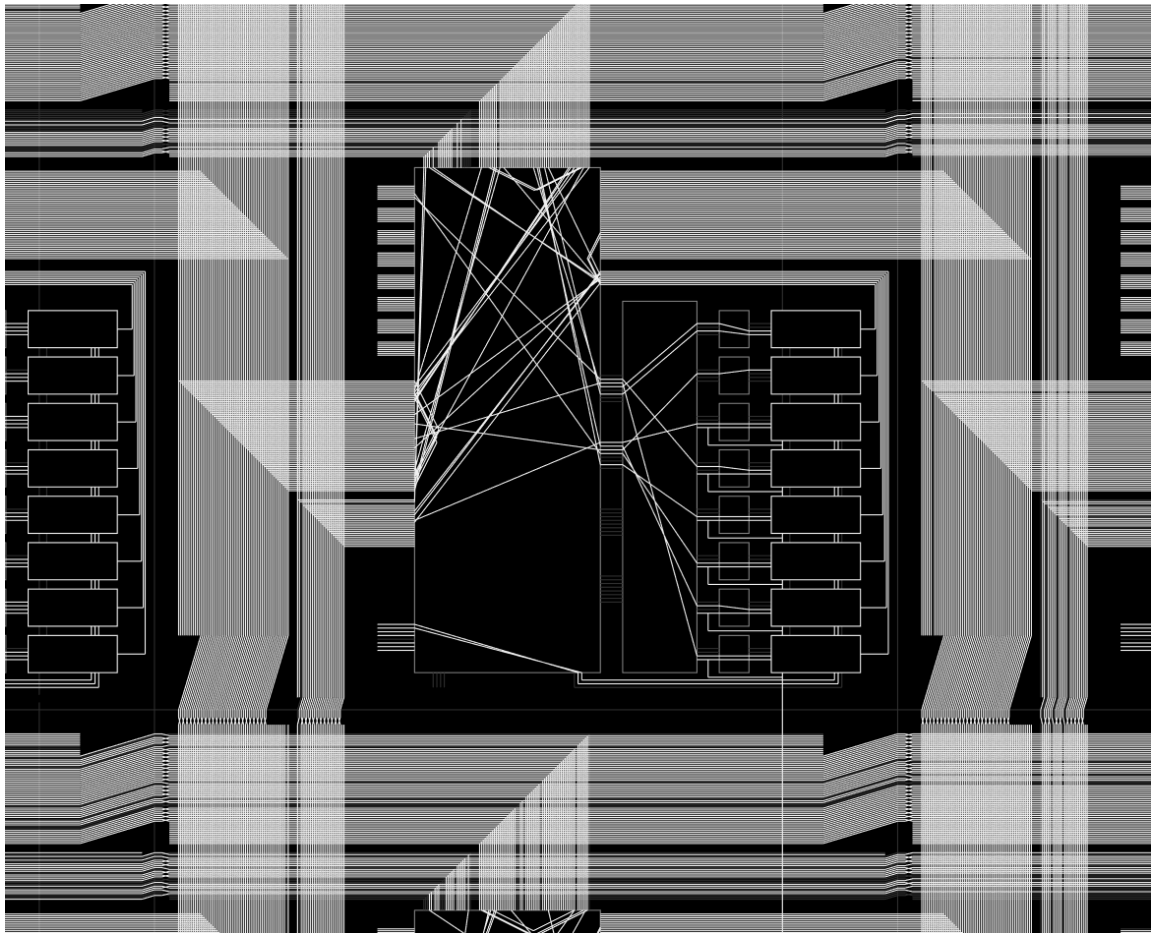
Designing your own processor, how ?

> **FPGAs: configurable hardware**

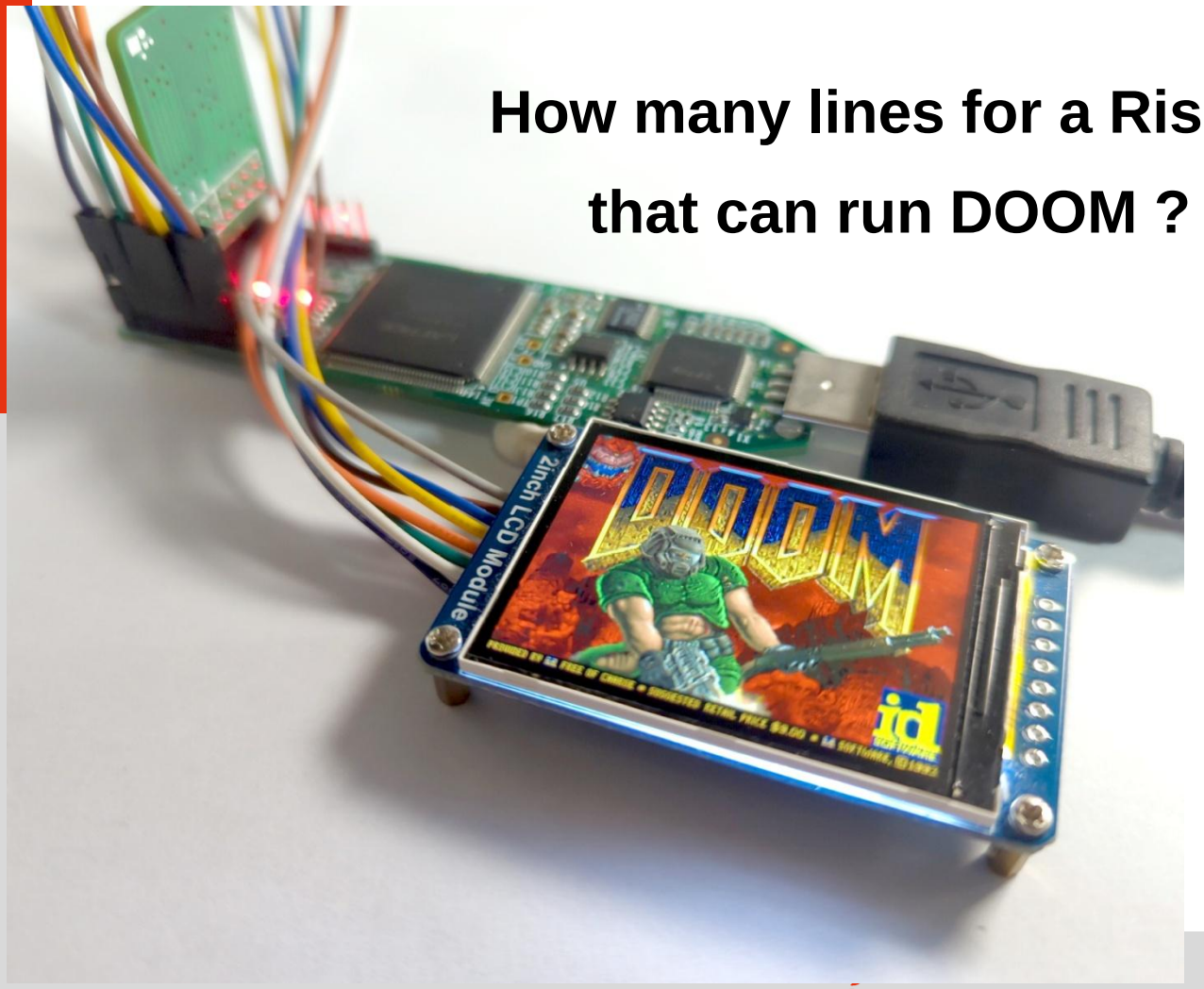


Designing your own processor, how ?

Designing your own processor, how ?



Designing your own processor, how ?



How many lines for a RiscV CPU

that can run DOOM ?

```
module FemtoRV32(
   input             clk,
   output [31:0] mem_addr,
   output [31:0] mem_wdata,
   output [3:0]  mem_wmask,
   input [31:0]  mem_rdata,
   output        mem_rstrb,
   input         mem_rbusy,
   input         mem_wbusy,
   input         reset
);
   parameter RESET_ADDR = 32'h00000000;
   parameter ADDR_WIDTH = 24;
   localparam ADDR_PAD  = {(32-ADDR_WIDTH){1'b0}};
   wire [4:0] rdId = instr[11:7];
   (* onehot *) wire [7:0] funct3Is = 8'b00000001 << instr[14:12];

   wire [31:0] Uimm = {    instr[31],   instr[30:12], {12{1'b0}}};
   wire [31:0] Iimm = {{21{instr[31]}}, instr[30:20]};
   wire [31:0] Simm = {{21{instr[31]}}, instr[30:25],instr[11:7]};
   wire [31:0] Bimm = {{20{instr[31]}}, instr[7],instr[30:25],instr[11:8],1'b0};
   wire [31:0] Jimm = {{12{instr[31]}}, instr[19:12],instr[20],instr[30:21],1'b0};

   wire isLoad    =  (instr[6:2] == 5'b00000); // rd <- mem[rs1+Iimm]
   wire isALUimm  =  (instr[6:2] == 5'b00100); // rd <- rs1 OP Iimm
   wire isAUIPC   =  (instr[6:2] == 5'b00101); // rd <- PC + Uimm
   wire isStore   =  (instr[6:2] == 5'b01000); // mem[rs1+Simm] <- rs2
   wire isALUreg  =  (instr[6:2] == 5'b01100); // rd <- rs1 OP rs2
   wire isLUI     =  (instr[6:2] == 5'b01101); // rd <- Uimm
   wire isBranch  =  (instr[6:2] == 5'b11000); // if(rs1 OP rs2) PC<-PC+Bimm
   wire isJALR    =  (instr[6:2] == 5'b11001); // rd <- PC+4; PC<-rs1+Iimm
   wire isJAL     =  (instr[6:2] == 5'b11011); // rd <- PC+4; PC<-PC+Jimm
   wire isSYSTEM  =  (instr[6:2] == 5'b11100); // rd <- cycles
   wire isALU = isALUimm | isALUreg;

   reg [31:0] rs1;
   reg [31:0] rs2;
   reg [31:0] registerFile [31:0];
   always @(posedge clk) begin
     if (writeBack)
       if (rdId != 0)
         registerFile[rdId] <= writeBackData;
   end

   wire [31:0] aluIn1 = rs1;
   wire [31:0] aluIn2 = isALUreg | isBranch ? rs2 : Iimm;
   reg  [31:0] aluReg;
   reg  [4:0]  aluShamt;
   wire aluBusy = |aluShamt;
   wire aluWr;
   wire [31:0] aluPlus = aluIn1 + aluIn2;
   wire [32:0] aluMinus = {1'b1, ~aluIn2} + {1'b0,aluIn1} + 33'b1;
   wire        LT  = (aluIn1[31] ^ aluIn2[31]) ? aluIn1[31] : aluMinus[32];
   wire        LTU = aluMinus[32];
   wire        EQ  = (aluMinus[31:0] == 0);
   wire [31:0] aluOut =
     (funct3Is[0]  ? instr[30] & instr[5] ? aluMinus[31:0] : aluPlus : 32'b0) |
     (funct3Is[2]  ? {31'b0, LT}                                     : 32'b0) |
     (funct3Is[3]  ? {31'b0, LTU}                                    : 32'b0) |
     (funct3Is[4]  ? aluIn1 ^ aluIn2                                 : 32'b0) |
     (funct3Is[6]  ? aluIn1 | aluIn2                                 : 32'b0) |
     (funct3Is[7]  ? aluIn1 & aluIn2                                 : 32'b0) |
     (funct3IsShift ? aluReg                                         : 32'b0) ;
   wire funct3IsShift = funct3Is[1] | funct3Is[5];
   always @(posedge clk) begin
      if(aluWr) begin
         if (funct3IsShift) begin  // SLL, SRA, SRL
            aluReg <= aluIn1;
            aluShamt <= aluIn2[4:0];
         end
      end
      if (|aluShamt) begin
         aluShamt <= aluShamt - 1;
         aluReg <= funct3Is[1] ? aluReg << 1 :
                   {instr[30] & aluReg[31], aluReg[31:1]};
      end
   end
   wire predicate = funct3Is[0] &  EQ  | funct3Is[1] & !EQ  | funct3Is[4] &  LT  |
                    funct3Is[5] & !LT  | funct3Is[6] &  LTU | funct3Is[7] & !LTU ;

   reg  [ADDR_WIDTH-1:0] PC;
   reg  [31:2] instr;
   wire [ADDR_WIDTH-1:0] PCplus4 = PC + 4;
   wire [ADDR_WIDTH-1:0] PCplusImm = PC + ( instr[3] ? Jimm[ADDR_WIDTH-1:0] :
                                            instr[4] ? Uimm[ADDR_WIDTH-1:0] :
                                                       Bimm[ADDR_WIDTH-1:0] );
   wire [ADDR_WIDTH-1:0] loadstore_addr = rs1[ADDR_WIDTH-1:0] +
                  (instr[5] ? Simm[ADDR_WIDTH-1:0] : Iimm[ADDR_WIDTH-1:0]);
   assign mem_addr = {ADDR_PAD,
                      state[WAIT_INSTR_bit] | state[FETCH_INSTR_bit] ?
                      PC : loadstore_addr};
   wire [31:0] writeBackData  =
      (isSYSTEM            ? cycles                : 32'b0) |
      (isLUI               ? Uimm                  : 32'b0) |
      (isALU               ? aluOut                : 32'b0) |
      (isAUIPC             ? {ADDR_PAD,PCplusImm}  : 32'b0) |
      (isJALR   | isJAL    ? {ADDR_PAD,PCplus4  }  : 32'b0) |
      (isLoad              ? LOAD_data             : 32'b0) ;
   wire mem_byteAccess     = instr[13:12] == 2'b00;
   wire mem_halfwordAccess = instr[13:12] == 2'b01;
   wire LOAD_sign =
        !instr[14] & (mem_byteAccess ? LOAD_byte[7] : LOAD_halfword[15]);
   wire [31:0] LOAD_data =
         mem_byteAccess ? {{24{LOAD_sign}},     LOAD_byte} :
     mem_halfwordAccess ? {{16{LOAD_sign}}, LOAD_halfword} :
                          mem_rdata ;
   wire [15:0] LOAD_halfword =
               loadstore_addr[1] ? mem_rdata[31:16] : mem_rdata[15:0];
   wire  [7:0] LOAD_byte =
               loadstore_addr[0] ? LOAD_halfword[15:8] : LOAD_halfword[7:0];
   assign mem_wdata[ 7: 0] = rs2[7:0];
   assign mem_wdata[15: 8] = loadstore_addr[0] ? rs2[7:0]  : rs2[15: 8];
   assign mem_wdata[23:16] = loadstore_addr[1] ? rs2[7:0]  : rs2[23:16];
   assign mem_wdata[31:24] = loadstore_addr[0] ? rs2[7:0]  :
                             loadstore_addr[1] ? rs2[15:8] : rs2[31:24];
   wire [3:0] STORE_wmask =  mem_byteAccess ?
              (loadstore_addr[1] ?
                    (loadstore_addr[0] ? 4'b1000 : 4'b0100) :
                    (loadstore_addr[0] ? 4'b0010 : 4'b0001) ) :
              mem_halfwordAccess ?
                    (loadstore_addr[1] ? 4'b1100 : 4'b0011) : 4'b1111;
   localparam FETCH_INSTR_bit      = 0;
   localparam WAIT_INSTR_bit       = 1;
   localparam EXECUTE_bit          = 2;
   localparam WAIT_ALU_OR_MEM_bit = 3;
   localparam NB_STATES            = 4;
   localparam FETCH_INSTR      = 1 << FETCH_INSTR_bit;
   localparam WAIT_INSTR       = 1 << WAIT_INSTR_bit;
   localparam EXECUTE          = 1 << EXECUTE_bit;
   localparam WAIT_ALU_OR_MEM = 1 << WAIT_ALU_OR_MEM_bit;

   reg [NB_STATES-1:0] state;
   wire writeBack = ~(isBranch | isStore ) &
                    (state[EXECUTE_bit] | state[WAIT_ALU_OR_MEM_bit]);
   assign mem_rstrb = state[EXECUTE_bit] & isLoad | state[FETCH_INSTR_bit];
   assign mem_wmask = {4{state[EXECUTE_bit] & isStore}} & STORE_wmask;
   assign aluWr = state[EXECUTE_bit] & isALU;

   wire jumpToPCplusImm = isJAL | (isBranch & predicate);
   wire needToWait = isLoad | isStore | isALU & funct3IsShift;


   always @(posedge clk) begin
      if(!reset) begin
         state      <= WAIT_ALU_OR_MEM;
         PC         <= RESET_ADDR[ADDR_WIDTH-1:0];
      end else
      (* parallel_case *)
      case(1'b1)
        state[WAIT_INSTR_bit]: begin
           if(!mem_rbusy) begin
              rs1 <= registerFile[mem_rdata[19:15]];
              rs2 <= registerFile[mem_rdata[24:20]];
              instr <= mem_rdata[31:2];
              state <= EXECUTE;
           end
        end
        state[EXECUTE_bit]: begin
           PC <= isJALR          ? {aluPlus[ADDR_WIDTH-1:1],1'b0} :
                 jumpToPCplusImm ? PCplusImm :
                 PCplus4;
           state <= needToWait ? WAIT_ALU_OR_MEM : FETCH_INSTR;
        end
        state[WAIT_ALU_OR_MEM_bit]: begin
           if(!aluBusy & !mem_rbusy & !mem_wbusy) state <= FETCH_INSTR;
        end
        default: state <= WAIT_INSTR;
      endcase
   end
   reg [31:0]  cycles;
   always @(posedge clk) cycles <= cycles + 1;
endmodule
```

What makes it easy ?

- **RISC-V ISA + software ecosystem (gcc, Linux, …)**

What makes it easy ?

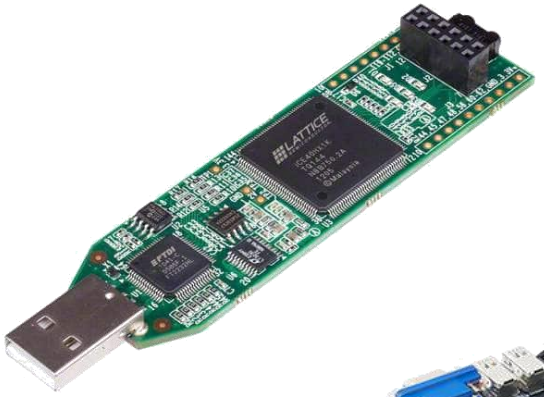
- **RISC-V ISA + software ecosystem (gcc, Linux, …)**
- **Open-source tools** Yosys / NextPNR

What makes it easy ?

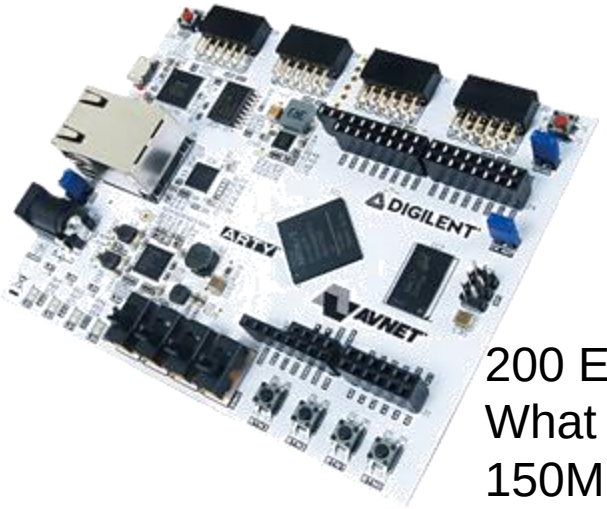
- **RISC-V ISA + software ecosystem (gcc, Linux, …)**
- **Open-source tools** Yosys / NextPNR
- **Cheap FPGAs**

Designing your own processor, how ?

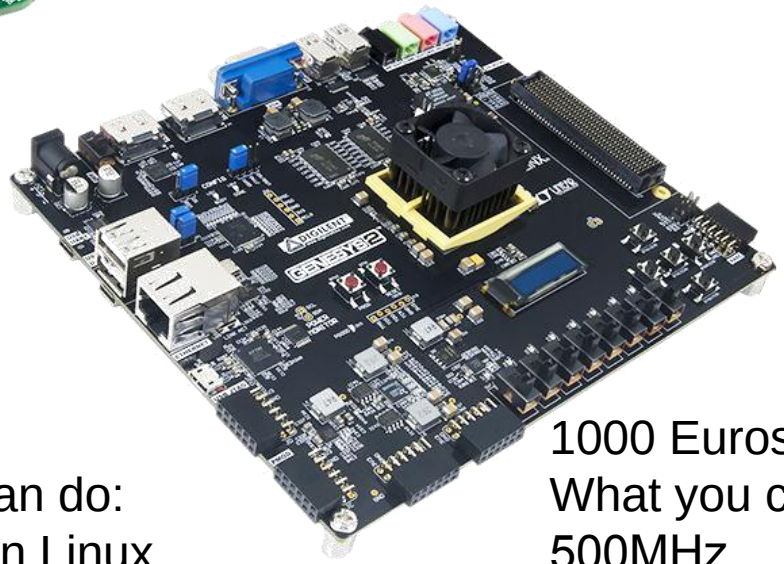
> **FPGAs**



50 Euros
What you can do:
80MHz microcontroller



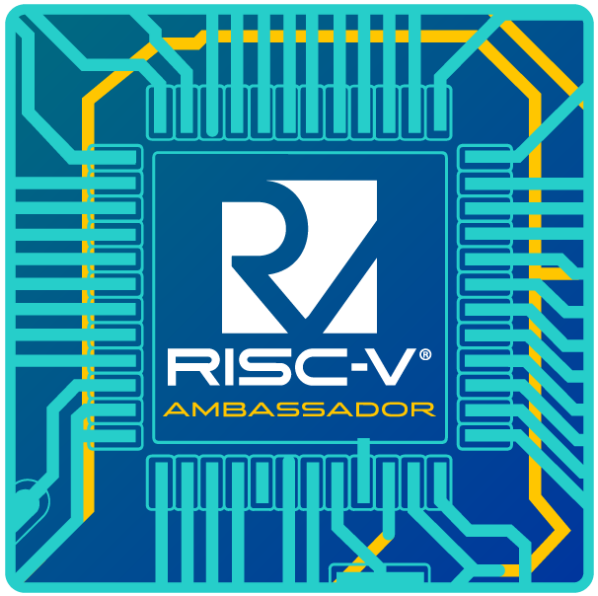
200 Euros
What you can do:
150MHz, run Linux



1000 Euros
What you can do:
500MHz, …

Learn-FPGA project - Vision

https://github.com/BrunoLevy/learn-fpga

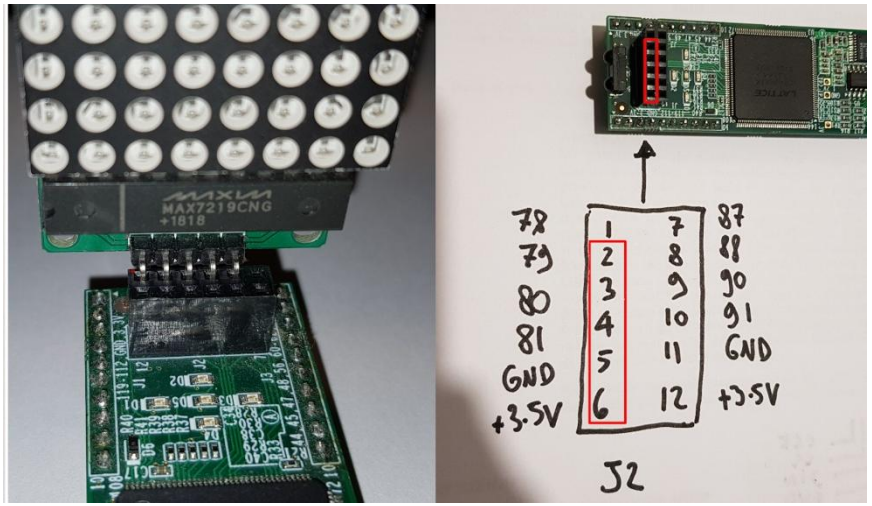


- **It is going to be important (remember, Nvidia !)**
- **People/competence is the critical resource**

Learn-FPGA project

https://github.com/BrunoLevy/learn-fpga

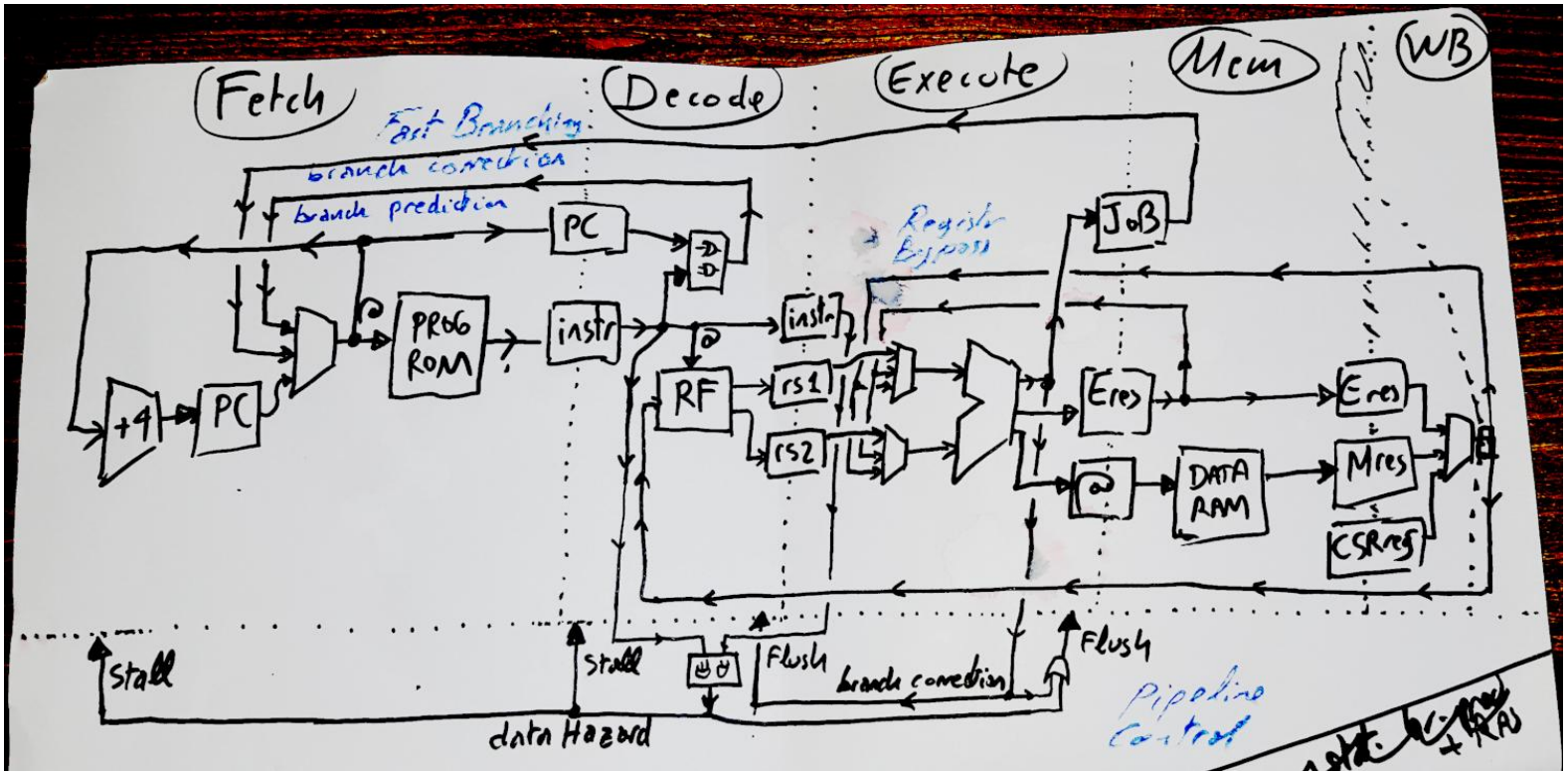
- >**Zero to RISC-V courses, teaching material**
- >**Small HW requirement (50 Euros / student)**
- >**Small Risc-V design (200 lines)**



Learn-FPGA project

https://github.com/BrunoLevy/learn-fpga

>Pipelined design, FPU



Learn-FPGA project

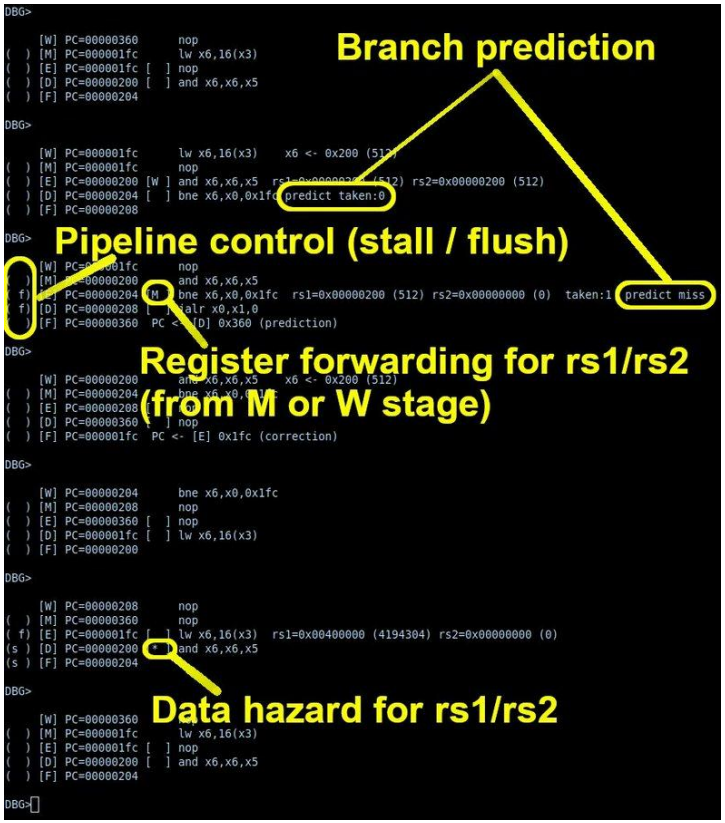
https://github.com/BrunoLevy/TordBoyau

Performance (RV32I) (A35T/Vivado)

branch prediction	CoreMarks/MHz	DMips/MHz	Raystones	LUTs	FFs
none	0.928	1.298	5.665	909	517
static (BTFNT)	1.118	1.488	6.633	938	516
static + RAS	1.147	1.528	6.795	1040	676
gshare	1.124	1.562	7.186	1297	547
gshare + RAS	1.153	1.606	7.375	1388	711

Performance (RV32IM) (A35T/Vivado)

branch prediction	CoreMarks/MHz	DMips/MHz	Raystones	LUTs	FFs
none	2.387	1.341	15.296	1368	681
static (BTFNT)	2.763	1.545	16.097	1363	680
static + RAS	2.790	1.579	16.476	1478	840
gshare	2.837	1.597	17.753	1760	711
gshare + RAS	2.866	1.634	18.215	1801	875



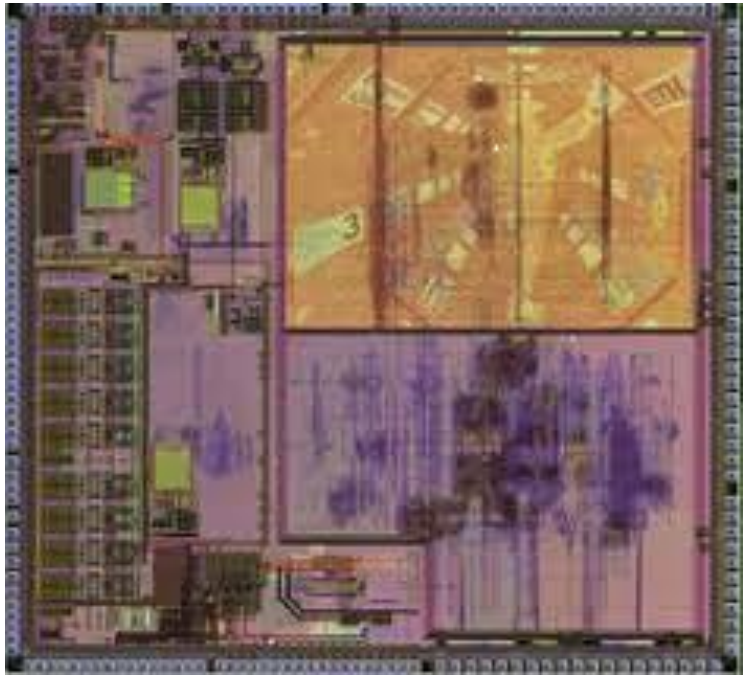
Learn-FPGA project

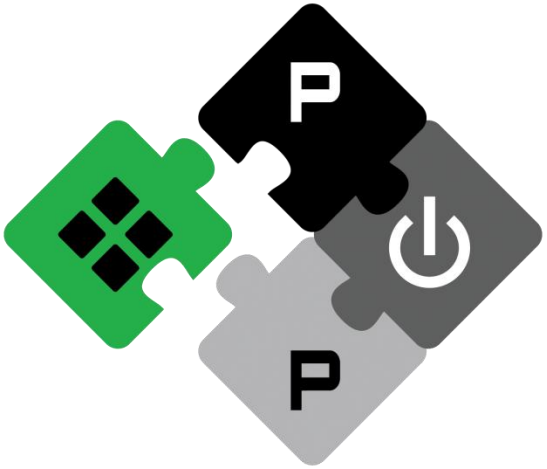
https://github.com/BrunoLevy/learn-fpga

- **To come** (help wanted)
  - Priviledged instructions, running Linux
  - OoO, Tomasulo
  - Port tutorials to other HDLs
  - LiteX integration, LiteOS, porting RiscOS

What’s next ?

> **PULP platform ETH Zurich**





What’s next ?

> **Why don’t we have a French RISC-V design ?**