Non-traditional architectures and the future

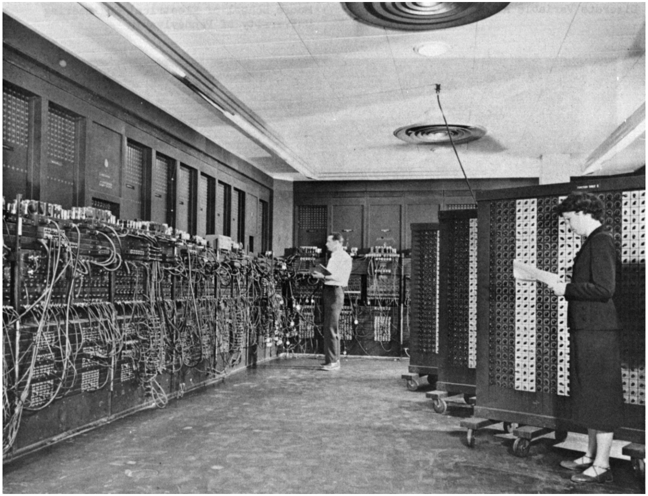
Saulius Gražulis

Vilnius, 2021

Vilnius University, Faculty of Mathematics and Informatics
Institute of Informatics

ENIAC

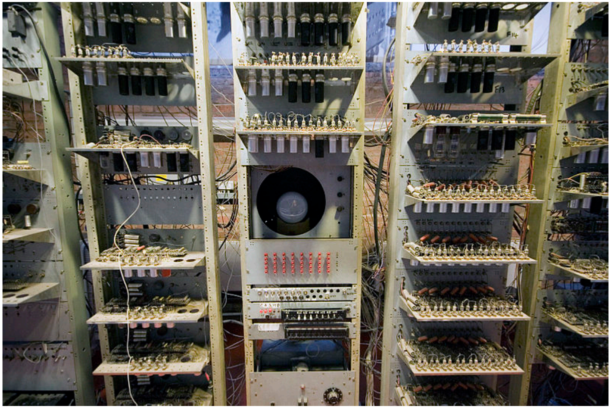
ENIAC – had to be rewired...



U.S. Army Photo, Public Domain

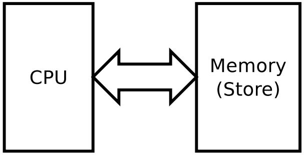
Harvard/von Neumann stored program computers

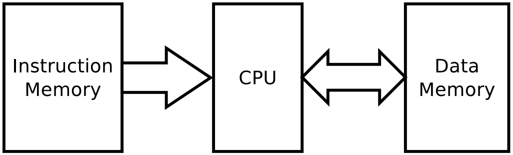
SSEM „Manchester Baby“ – first (?) stored program vacuum tube computer...



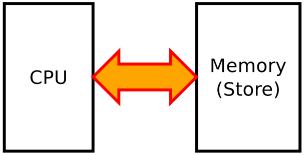
By Parrot of Doom, CC BY-SA 3.0, via Wikimedia Commons

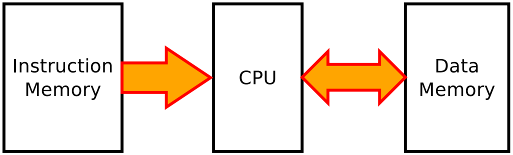
Von Neumann architecture bottleneck



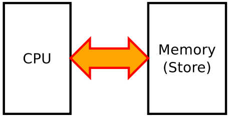


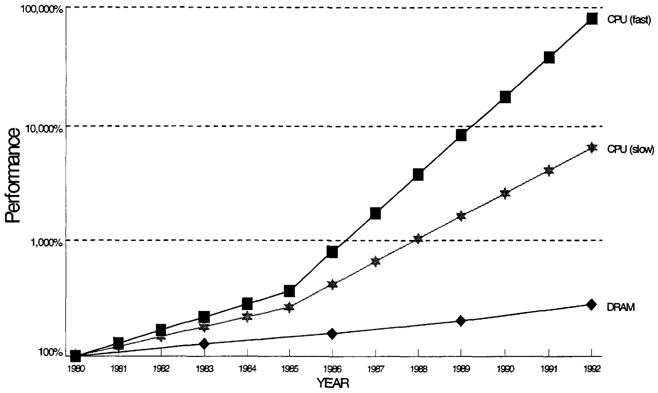
Von Neumann architecture bottleneck





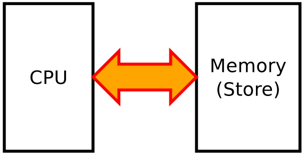
Von Neumann architecture bottleneck





(Groves 1995)

Von Neumann architecture bottleneck

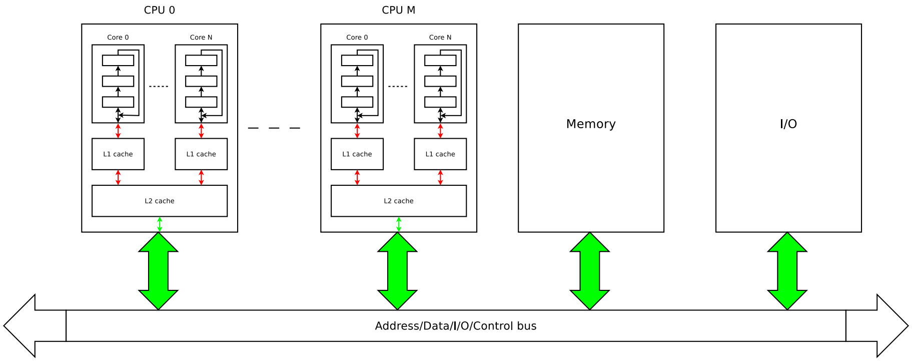


In its simplest form a von Neumann computer has three parts: a central processing unit (or CPU), a store, and a connecting tube that can transmit a single word between the CPU and the store (and send an address to the store). I propose to call this tube **the von Neumann bottleneck**.

John Backus, 1977 ACM Turing Award Lecture (Backus 1978)

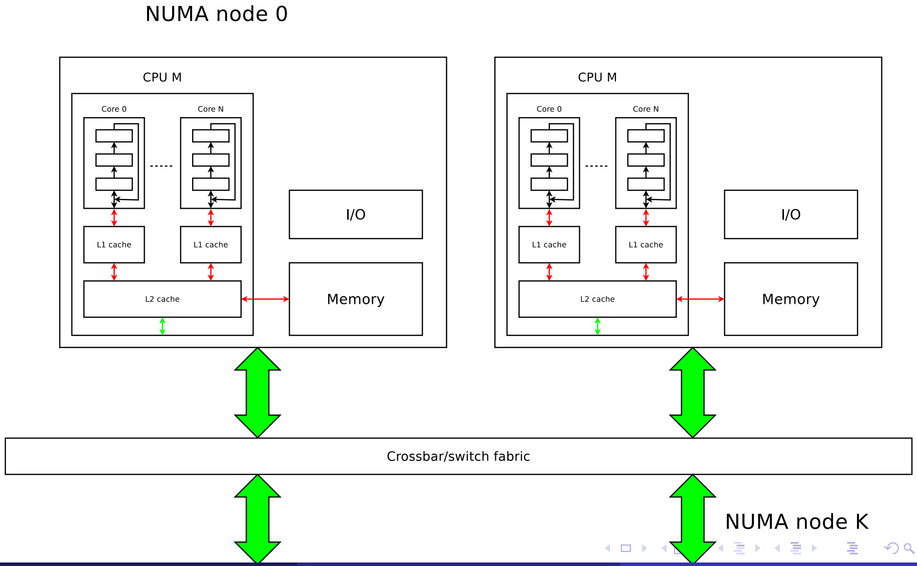
UMA

UMA: Uniform Memory Access (Groves 1995)



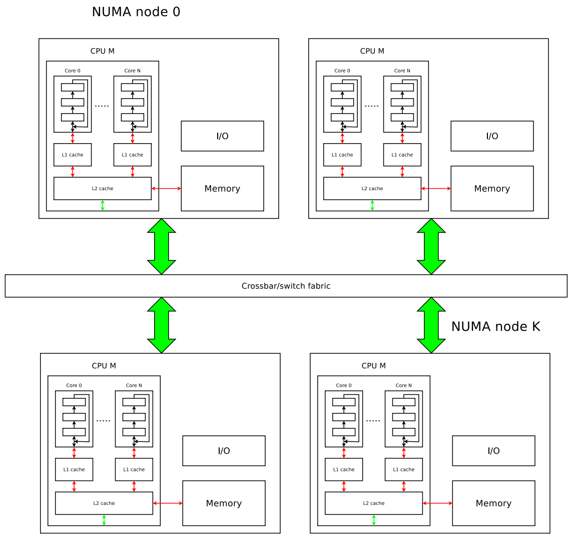
UMA

NUMA: Non-Uniform Memory Access (Groves 1995)



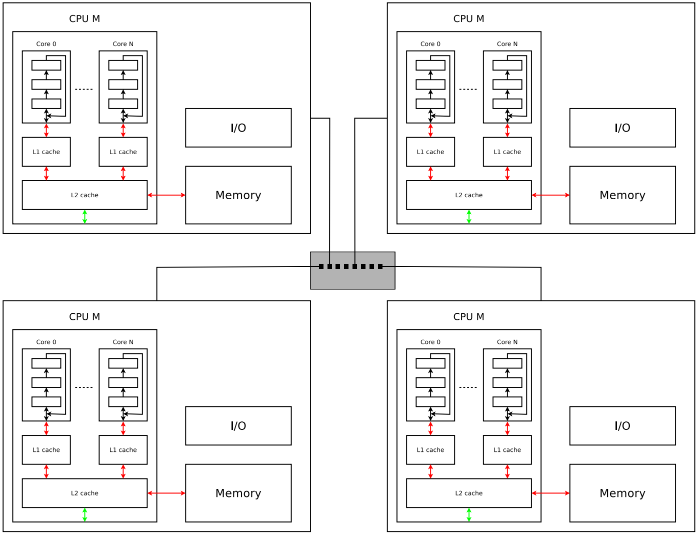
UMA

NUMA: Non-Uniform Memory Access (Groves 1995)



NORMA

NORMA: No Remote Memory Access (Groves 1995)



Programming parallel machines

- For NORMA/MUMA/UMA:
  MPI: Message Passing Interface (https://www.open-mpi.org/)
- For NUMA/UMA:
  OpenMP: Open Multi-Processing API (https://www.openmp.org/)

OpenMP example:

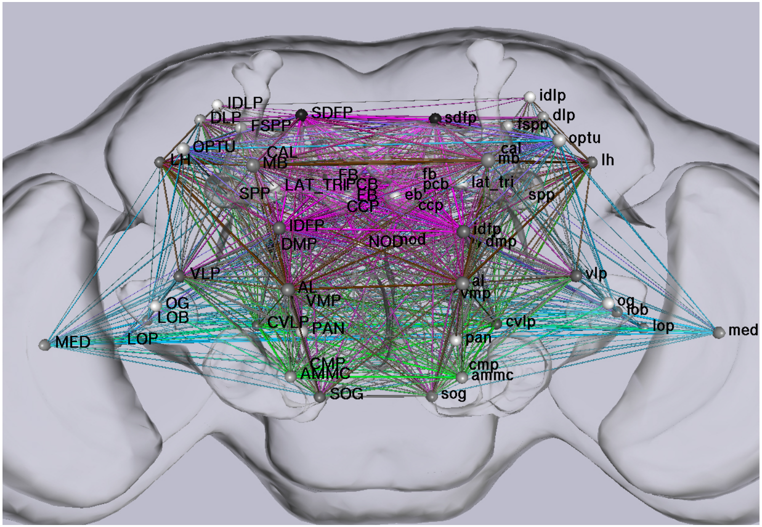
```c
#include <stdio.h>
#define N 100000000LL
int main(int argc, char *argv[]) {
    static long long a[N];
    long long i;

    #pragma omp parallel for
    for (i = 0; i < N; i++)
        a[i] = 2 * i;

    printf( "%lld\n", a[N-1LL] );
    return 0;
}
```

```
cc \
    -fopenmp \
    -Wall \
    -O3 \
    -fomit-frame-pointer \
    -funroll-loops \
    -o loop \
loop.c
```

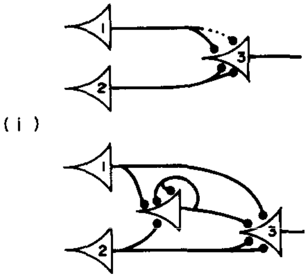
Brain wiring

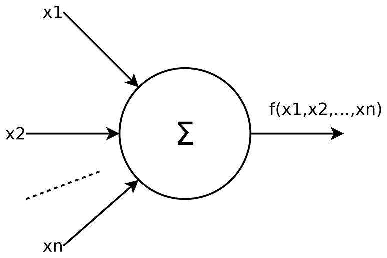


http://www.flycircuit.tw (Chiang et al. 2011)

Neuron models

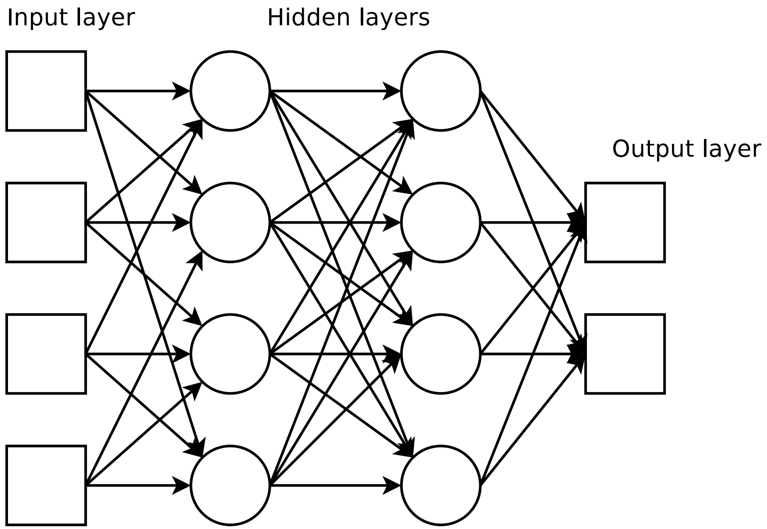
McCulloch–Pitts neuron (McCulloch et al. 1943; Alom et al. 2018):



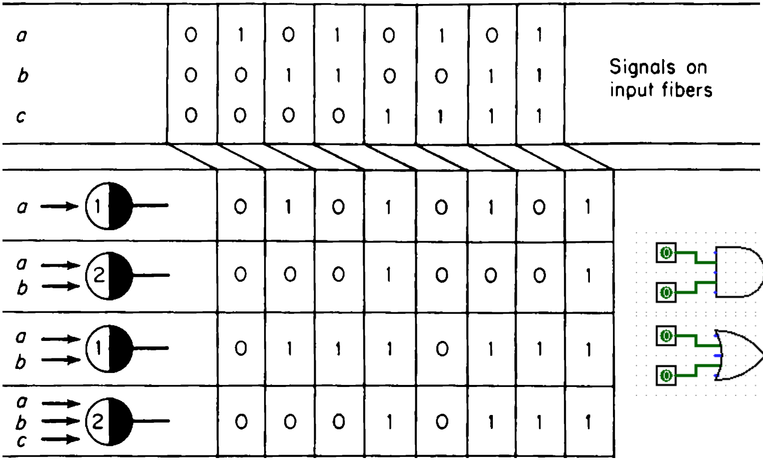


$$f(x_1, x_2, \ldots, x_n) = \varphi\left(b + \sum_{i=1}^{n} w_i x_i\right)$$

Neural Networks

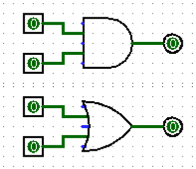


Neurons as logic gates



a	0	1	0	1	0	1	0	1	Signals on input fibers
b	0	0	1	1	0	0	1	1	
c	0	0	0	0	1	1	1	1	

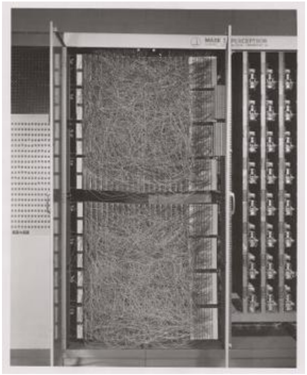
a	1	0	1	0	1	0	1	0	1
a, b	2	0	0	0	1	0	0	0	1
a, b	1	0	1	1	1	0	1	1	1
a, b, c	2	0	0	0	1	0	1	1	1



(Minsky 1967)

Perceptron

Frank Rosenblatt's Perceptron (Rosenblatt 1957):



Deep learning ANNs (Alom et al. 2018):

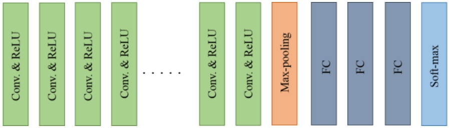


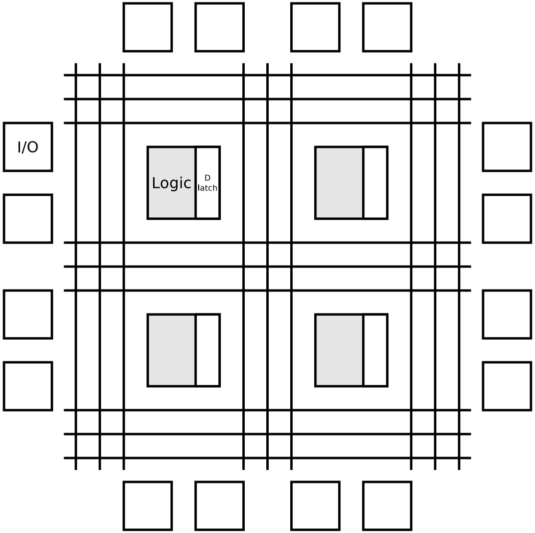
Fig. 15. Basic building block of VGG network: Convolution (Conv) and FC for fully connected layers

Single layer

Multilayer

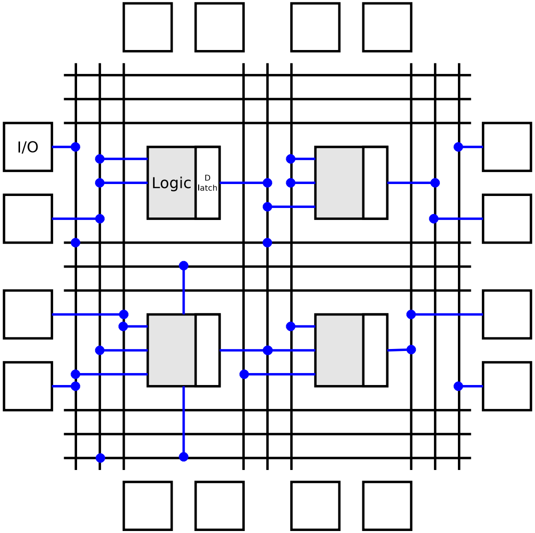
Single layer can not do XOR: (Minsky and Papert 1969)

FPGA



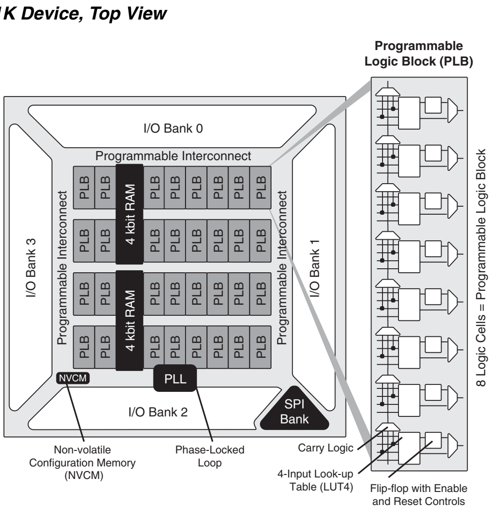
Adapted from (Brown et al. 2000)

FPGA



Adapted from (Brown et al. 2000)

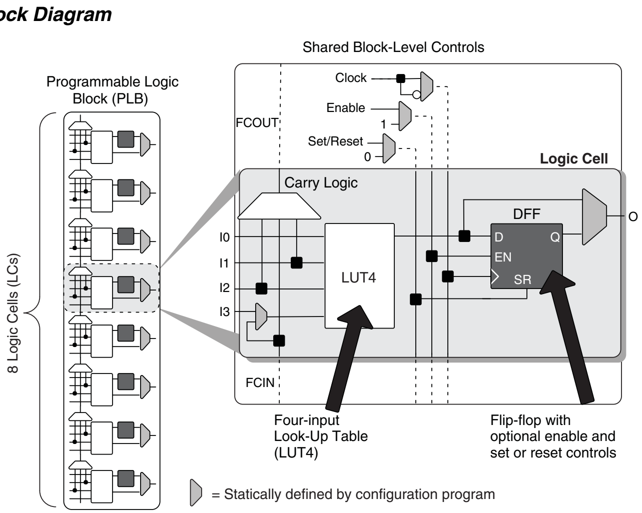
Lattice Semiconductor FPGA



iCE40LP/HX1K Device, Top View
Programmable Logic Block (PLB)
I/O Bank 0
Programmable Interconnect
I/O Bank 3
Programmable Interconnect
PLB
4 kbit RAM
4 kbit RAM
Programmable Interconnect
I/O Bank 1
NVCM
PLL
I/O Bank 2
SPI Bank
8 Logic Cells = Programmable Logic Block
Non-volatile Configuration Memory (NVCM)
Phase-Locked Loop
Carry Logic
4-Input Look-up Table (LUT4)
Flip-flop with Enable and Reset Controls

(Lattice Semiconductor 2017)

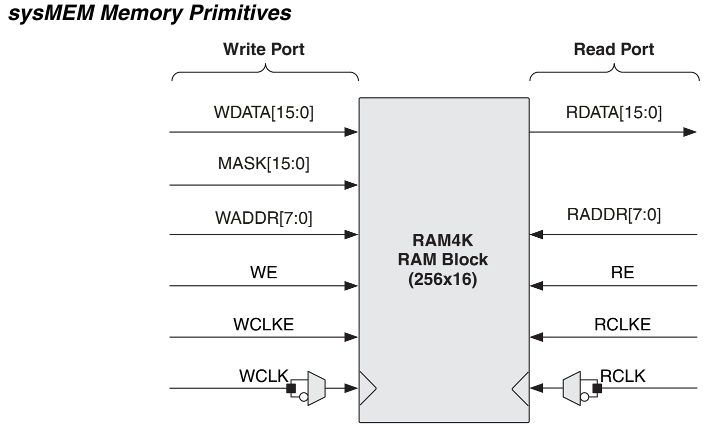
Lattice Semiconductor FPGA



PLB Block Diagram
Shared Block-Level Controls
Programmable Logic Block (PLB)
8 Logic Cells (LCs)
Clock
Enable
1
Set/Reset
0
FCOUT
Logic Cell
Carry Logic
DFF
D
Q
EN
SR
O
I0
I1
I2
I3
LUT4
FCIN
Four-input Look-Up Table (LUT4)
Flip-flop with optional enable and set or reset controls
= Statically defined by configuration program

(Lattice Semiconductor 2017)

Lattice Semiconductor FPGA



(Lattice Semiconductor 2017)

FPGA ANN

This reference design implements Convolutional Neural Network (CNN) based human face identification on Lattice's low power ECP5 FPGA using an image sensor.

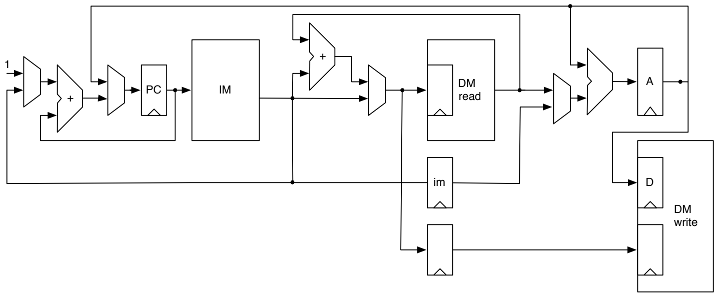
Lattice Semiconductor Reference Designs

Features:

- VGG8 like – 8x (Convolution, Batch Normalisation) + 4x Pooling + 1 fully connected CNN
- Runs at 2 frames per second with 90 x 90 RGB Input
- Total ECP5 power consumption of 850mW

CPU in FPGA

CPUs can be implemented in FPGA:



(Caska et al. 2011; Schoeberl 2011)

Hardware description languages

- Verilog (https://en.wikipedia.org/wiki/Verilog)
- VHDL (https://en.wikipedia.org/wiki/VHDL)
- Chissel (https://www.chisel-lang.org/)

Project stages/system capabilities

1. Describe
2. Simulate
3. Verify
4. Synthesise (for FPGA or Silicon foundry)

Verilog example

```verilog
module rng (
            input  clk,
            output LED1,
            output LED2,
            output LED3,
            output LED4,
            output LED5
            );
   localparam BITS = 5;
   localparam LOG2DELAY = 22;
   reg [BITS+LOG2DELAY-1:0] counter = 0;
   reg                      ready = 0;
   reg [31:0]               rng;
   always@(posedge clk)
     counter <= counter + 1;
   always@(posedge counter[LOG2DELAY-2])
     if( ready )
       begin
          rng <= ({rng[0],(rng >> 1)})^(rng | {(rng << 1),rng[31]});
       end
     else
       begin
          rng = 32'h00010000;
          ready = 1;
       end
   assign {LED1, LED2, LED3, LED4, LED5} = rng[11:7];
endmodule
```

https://github.com/RGD2/icestorm_example

Verilog example

```
module rng (
            input  clk,
            output LED1,
            output LED2,
            output LED3,
            output LED4,
            output LED5
            );
   localparam BITS = 5;
   localparam LOG2DELAY = 22;
   reg [BITS+LOG2DELAY-1:0] counter = 0;
   reg                      ready = 0;
   reg [31:0]               rng;
   always@(posedge clk)
     counter <= counter + 1;
   always@(posedge counter[LOG2DELAY-2])
     if( ready )
       begin
          rng <= ({rng[0],(rng >> 1)})^(rng | {(rng << 1),rng[31]});
       end
     else
       begin
          rng = 32'h00010000;
          ready = 1;
       end
   assign {LED1, LED2, LED3, LED4, LED5} = rng[11:7];
endmodule
```

```
saulius@tasmanijos-velnias verilog/ $ make -n upload
yosys -p "read_verilog␣rng.v;␣synth_ice40␣-blif␣rng.blif"
arachne-pnr -d 1k -p rng.pcf -o rng.txt rng.blif
icepack rng.txt rng.bin
iceprog rng.bin
```

https://github.com/RGD2/icestorm_example

Verilog example

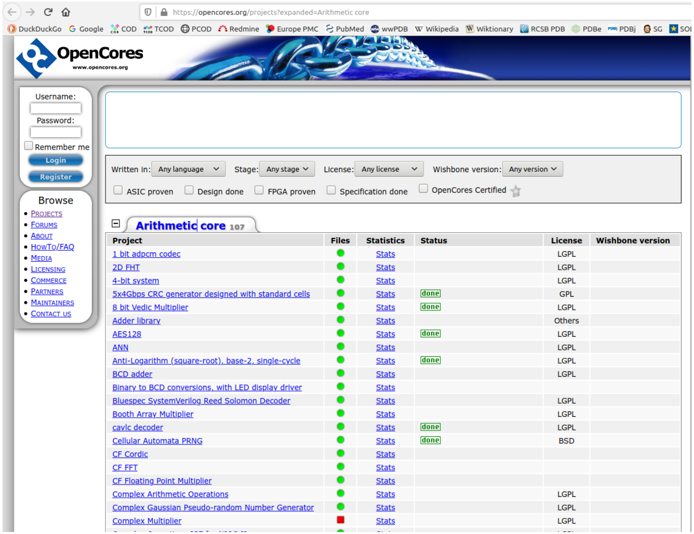
```
module rng (
            input  clk,
            output LED1,
            output LED2,
            output LED3,
            output LED4,
            output LED5
            );
   localparam BITS = 5;
   localparam LOG2DELAY = 22;
   reg [BITS+LOG2DELAY-1:0] counter =
   reg                      ready = (
   reg [31:0]               rng;
   always@(posedge clk)
     counter <= counter + 1;
   always@(posedge counter[LOG2DELAY-2])
     if( ready )
       begin
          rng <= ({rng[0],(rng >> 1)})^(rng | {(rng << 1),rng[31]});
       end
     else
       begin
          rng = 32'h00010000;
          ready = 1;
       end
   assign {LED1, LED2, LED3, LED4, LED5} = rng[11:7];
endmodule
```

```
saulius@tasmanijos-velnias verilog/ $ make simulate
iverilog simulate.v
./a.out
Begin Simulation
At time                    0, LEDS = x x x x x
At time              2097151, LEDS = 0 0 0 0 0
At time             23068671, LEDS = 1 0 0 0 0
At time             27262975, LEDS = 0 1 0 0 0
At time             31457279, LEDS = 1 1 1 0 0
At time             35651583, LEDS = 0 0 0 1 0
^C** VVP Stop(0) **
** Flushing output streams.
** Current simulation time is 39064597 ticks.
> finish
```

https://github.com/RGD2/icestorm_example

Open Cores

https://opencores.org/



https://opencores.org/projects?expanded=Arithmetic core

Username:

Password:

Remember me

Login

Register

Browse

- Projects
- Forums
- About
- HowTo/FAQ
- Media
- Licensing
- Commerce
- Partners
- Maintainers
- Contact us

Written in: Any language Stage: Any stage License: Any license Wishbone version: Any version

ASIC proven Design done FPGA proven Specification done OpenCores Certified

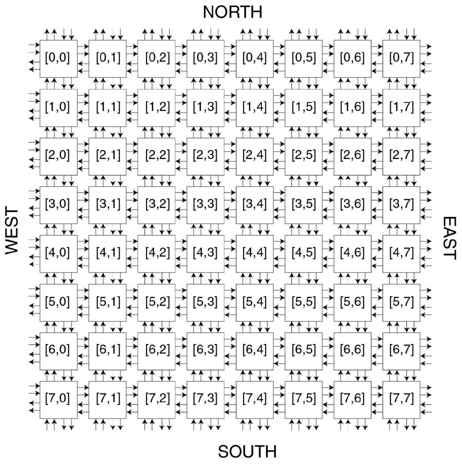
Arithmetic core 107

Project	Files	Statistics	Status	License	Wishbone version
1 bit adpcm codec	●	Stats		LGPL	
2D FHT	●	Stats		LGPL	
4-bit system	●	Stats		LGPL	
5x4Gbps CRC generator designed with standard cells	●	Stats	done	GPL	
8 bit Vedic Multiplier	●	Stats	done	LGPL	
Adder library	●	Stats		Others	
AES128	●	Stats	done	LGPL	
ANN	●	Stats		LGPL	
Anti-Logarithm (square-root), base-2, single-cycle	●	Stats	done	LGPL	
BCD adder	●	Stats		LGPL	
Binary to BCD conversions, with LED display driver	●	Stats			
Bluespec SystemVerilog Reed Solomon Decoder	●	Stats		LGPL	
Booth Array Multiplier	●	Stats		LGPL	
cavlc decoder	●	Stats	done	LGPL	
Cellular Automata PRNG	●	Stats	done	BSD	
CF Cordic	●	Stats			
CF FFT	●	Stats			
CF Floating Point Multiplier	●	Stats			
Complex Arithmetic Operations	●	Stats		LGPL	
Complex Gaussian Pseudo-random Number Generator	●	Stats		LGPL	
Complex Multiplier	🟥	Stats		LGPL	

FPGA for bioinformatics

- Hall, A. Short-Read DNA Sequence Alignment with Custom Designed FPGA-based Hardware (Hall 2010);
- FPGA based molecular dynamics: (Khan et al. 2012; Yang et al. 2019; Waidyasooriya et al. 2016).

Cell Matrix



https://cellmatrix.com/entryway/entryway/branchAbout.html

https://www.cellmatrix.com

Cell Matrix

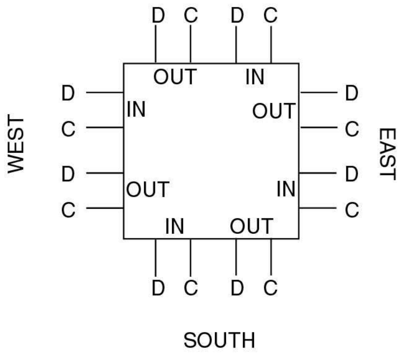


FIGURE 3 - A SINGLE CELL MATRIX CELL

https://cellmatrix.com/entryway/entryway/branchAbout.html

https://www.cellmatrix.com

Other possibilities...

- Cellular automata (e.g. J. H. Conway's "Life"); Turing complete!
- DNA data storage
- DNA computing

References I

Alom, Md Zahangir et al. (Mar. 3, 2018). “The history began from AlexNet: a comprehensive survey on deep learning approaches”. In: *arXiv*, pp. 1–39. arXiv: 1803.01164 [cs.CV]. URL: https://arxiv.org/abs/1803.01164v1.

Backus, John (1978). “Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs”. In: *Communications of the ACM*. Vol. 21. 8. Association of Computing Machinery, pp. 613–641. URL: https://dl.acm.org/doi/10.1145/1283920.1283933.

Brown, Stephen et al. (2000). *Architecture of FPGAs and CPLDs: A Tutorial*. Tech. rep. Department of Electrical and Computer Engineering, University of Toronto. URL: http://www.eecg.toronto.edu/~jayar/pubs/brown/survey.pdf.

Caska, James et al. (2011). “Java dust: how small can embedded Java be?” In: *Proceedings of the 9th International Workshop on Java Technologies for Real-Time and Embedded Systems - JTRES ’11*. ACM Press, pp. 1–5. DOI: 10.1145/2043910.2043931. URL: https://www.jopdesign.com/doc/lerosjvm.pdf.

Chiang, Ann-Shyn et al. (Jan. 2011). “Three-dimensional reconstruction of brain-wide wiring networks in Drosophila at single-cell resolution”. In: *Current Biology* 21.1, pp. 1–11. DOI: 10.1016/j.cub.2010.11.056.

Groves, R. (1995). “Brief history of computer architecture evolution and future trends”. en. In: *18th CERN School of Computing*. CERN, pp. 147–159. DOI: 10.5170/CERN-1995-005.147. URL: https://cds.cern.ch/record/399391/files/p147.pdf.

References II

Hall, Adam (Nov. 2010). “Short-Read DNA Sequence Alignment with Custom Designed FPGA-based Hardware”. MA thesis. THE FACULTY OF GRADUATE STUDIES (Bioinformatics) THE UNIVERSITY OF BRITISH COLUMBIA (Vancouver). URL: https://open.library.ubc.ca/cIRcle/collections/ubctheses/24/items/1.0071441.

Khan, M. A. et al. (2012). *FPGA-accelerated molecular dynamics*. URL: http://www.bu.edu/caadlab/Khan13.pdf.

Lattice Semiconductor (2017). *iCE40^{TM} LP/HX family data sheet*. Tech. rep. Lattice Semiconductor. URL: http://www.latticesemi.com/~/media/LatticeSemi/Documents/DataSheets/iCE/iCE40LPHXFamilyDataSheet.pdf.

McCulloch, Warren S. et al. (Dec. 1943). “A logical calculus of the ideas immanent in nervous activity”. In: *The Bulletin of Mathematical Biophysics* 5.4, pp. 115–133. DOI: 10.1007/bf02478259. URL: https://www.cs.cmu.edu/~./epxing/Class/10715/reading/McCulloch.and.Pitts.pdf.

Minsky, Marvin Lee (1967). *Computation: finite andinfinite machines*. Prentice-Hall.

Minsky, Marvin Lee and Seymour Papert (1969). *Perceptrons: an introduction to computational geometry*. MIT Press.

Rosenblatt, Frank (1957). *The Perceptron: a perceiving and recognising automaton*. Tech. rep. Cornaell Aeronautical Laboratory, Inc., pp. 1–33. URL: https://blogs.umass.edu/brain-wars/files/2016/03/rosenblatt-1957.pdf.

References III

Schoeberl, Martin (2011). *Leros: a tiny microcontroller for FPGAs*. URL: `https://www.jopdesign.com/doc/leros.pdf`.

Waidyasooriya, Hasitha Muthumala et al. (June 2016). “Architecture of an FPGA accelerator for molecular dynamics simulation using OpenCL”. In: *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*. IEEE, p. 7550743. DOI: `10.1109/icis.2016.7550743`.

Yang, Chen et al. (May 14, 2019). “Fully integrated on-FPGA molecular dynamics simulations”. In: *ArXiv*, p. 190505359. arXiv: `http://arxiv.org/abs/1905.05359v1 [cs.DC]`. URL: `https://arxiv.org/pdf/1905.05359.pdf`.