Operating Systems and C

Fall 2022

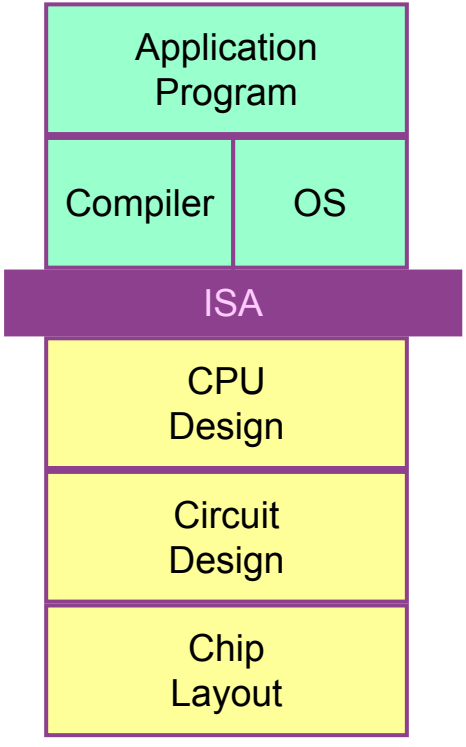
4. Interpreter

Outline

1. Instruction Set Architecture (ISA)
2. Logic Design
3. Moore’s Law and Dennard Scaling

Instruction Set Architecture

- Assembly Language View
  - Processor state
    - Registers, memory, …
  - Instructions
    - `addq`, `pushq`, `ret`, …
    - How instructions are encoded as bytes
- Layer of Abstraction
  - Above: how to program machine
    - Processor executes instructions in a sequence
  - Below: what needs to be built
    - Use variety of tricks to make it run fast
    - E.g., execute multiple instructions simultaneously



ARM ISA

Instruction Set Architecture

Overview

An *Instruction Set Architecture* (ISA) is part of the abstract model of a computer. It defines how software controls the processor.

The Arm ISA allows you to write software and firmware that conforms to the Arm specifications. This mean that, if your software or firmware conforms to the specifications, any Arm-based processor will execute it in the same way.

This guide introduces the A64 instruction set, used in the 64-bit Armv8-A architecture, also known as AArch64.

We will not cover every single instruction in this guide. All instructions are detailed in the *Arm Architecture Reference Manual* (Arm ARM). Instead, we will introduce the format of the instructions, the different types of instruction, and how code written in assembler can interact with compiler-generated code.

At the end of this guide, you can check your knowledge. You will have learned about the main classes of instructions, the syntax of data-processing instructions, and how the use of **w** and **x** registers affects instructions. The key outcome that we hope you will learn from this guide is to be able to explain how generated assembler code maps to C statements, when given a C program and the compiler output for it. Finally, this guide will show you how to write a function in assembler that can be called from C.

Why you should care about the ISA

As developers, you may not need to write directly in assembler in our day-to-day role. However, assembler is still important in some areas, such as the first stage boot software or some low-level kernel activities.

Even if you are not writing assembly code directly, understanding what the instruction set can do, and how the compiler makes use of those instructions, can help you to write more efficient code. It can also help you to understand the output of the compiler. This can be useful when debugging.

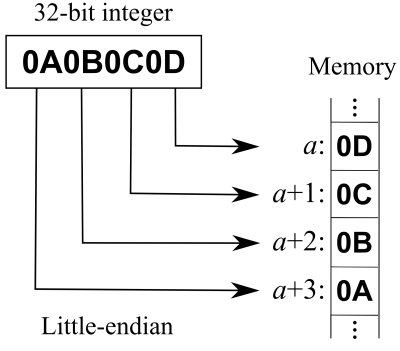
Instruction sets in the Armv8-A

Armv8-A supports three instruction sets: A32, T32 and A64.

Intel x86-64 & Extensions

https://en.wikichip.org/wiki/x86

Little endian
variable length
instruction set



	Extension	First μarch	Description
1978	Real	8086	Original Real mode
1982	Protected	80286	Protected mode
1985	SMM	80386	System Management Mode
1989	FPU	80486	Incorporates the x87 FPU into the same die
1996	MMX	P5	First implementation for SIMD instructions
1998	3DNow!	K6-2	SIMD extension for manipulating single-precision floating point
1999	SSE	P5	Streaming SIMD Extensions, SIMD for single-precision floating point
1999	E3DNow!	K7	Extended 3DNow! (New DSP instructions + some MMX instructions)
1999	Professional 3DNow!		A name given by AMD for E3DNow! + SSE
1999	EMMX	P6	Extended MMX; an extension to MMX
2001	SSE2	P6	Attempt to replace the original MMX instructions; wider XMM registers offer better performance
2004	SSE3	Core	Streaming SIMD Extensions 3; new instructions to operate on multiple values in the same register
2006	SSSE3	Core	Supplemental Streaming SIMD Extensions 3; additional instructions for working with packed integers
2007	SSE4.1	Penryn	Streaming SIMD Extensions 4.1; initial set of instructions for manipulating media data
2007	SSE4a	K10	4 SIMD instructions, not related to 4.1/4.2
2007	ABM	K10	Advanced Bit Manipulation; bit counting instructions
2008	SSE4.2	Nehalem	Streaming SIMD Extensions 4.2; second set of instructions for manipulating media data
~~2007~~	~~SSE5~~		Proposed by AMD in 2007 but was never implemented
2008	SSE4		Streaming SIMD Extensions 4; Another name for SSE4.1 + SSE4.2
2010	CLMUL	Westmere	Carry-less multiplication of two registers
2011	AVX	Sandy Bridge	Advanced Vector Extensions; introduces 256-bits integer operations
2012	F16C	Ivy Bridge	Extension for performing half-precision <-> single-precision conversions
2011	XOP	Bulldozer	eXtended Operations; various vector operations
2011	FMA4	Bulldozer	4-operands fused multiply-add
2011	LWP	Bulldozer	Lightweight Profiling
2011	SMX	Nehalem	Safer Mode Extensions; instructions needed to facilitate trust decisions (Part of Intel's Trusted Execution Technology)
2011	AES	Westmere	Instructions for optimizing AES operations
2012	TBM	Piledriver	Trailing Bit Manipulation; bit manipulation instructions designed to be compiler intrinsics
2013	BMI1	Jaguar	Bit Manipulation Instruction Set 1; introduces a number of bit manipulation instructions
2013	BMI2	Haswell	Bit Manipulation Instruction Set 2; introduces additional bit manipulation instructions
2013	FMA3	Haswell	3-operands fused multiply-add
2013	TSX	Haswell	Transactional Synchronization Extensions; adds transactional memory support
2013	AVX2	Haswell	Advanced Vector Extensions; additional instructions
2014	ADX	Broadwell	Multi-Precision Add-Carry Instruction extension
2014	RdRand	Broadwell	Part of Secure Key Technology extension (RdRand, RDSEED)
2014	PREFETCH	Broadwell	PREFETCH instructions (previously part of 3DNow!)
2015	AVX-512	Airmont	512 bit register operations
2015	MPX	Skylake	Memory Protection Extensions
2015	SGX	Skylake	Software Guard Extensions
2016	SHA	Goldmont	SHA Extensions
2017	SME	Zen	Secure Memory Extensions
2019	TME	Ice Lake	Total Memory Encryption

Relevance for Data Science

```
>>> @tff.federated_computation
... def hello_world():
...   return "Hello, World!"
...
>>> hello_world()
2020-09-02 11:49:40.005036: I tensorflow/compiler/jit/xla_cpu_device.cc:41] Not creating XLA devices, tf_xla_enable_xla_
devices not set
2020-09-02 11:49:40.005232: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized wit
h oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:
 AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2020-09-02 11:49:40.022599: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR optimizatio
n passes are enabled (registered 1)
b'Hello, World!'
```

CISC Instruction Sets

Complex Instruction Set Computer

- e.g., IA32

Stack-oriented instruction set

- Use stack to pass arguments, save program counter
- Explicit push and pop instructions

Arithmetic instructions can access memory

```
addq %rax, 12(%rbx,%rcx,8)
```

- requires memory read and write
- Complex address calculation

Condition codes

- Set as side effect of arithmetic and logical instructions

Philosophy

- Add instructions to perform “typical” programming tasks

RISC Instruction Sets

Reduced Instruction Set Computer

- Internal project at IBM, later popularized by Hennessy and Patterson
- ARM, RISC-V

Fewer, simpler instructions

- Might take more to get given task done
- Can execute them with small and fast hardware

Register-oriented instruction set

- Many more registers
- Use for arguments, return pointer, temporaries

Only load and store instructions can access memory

- Similar to X86-64 `mov`

No Condition codes

- Test instructions return 0/1 in register

RISC V

RISC-V®

RISC-V: The Free and Open RISC Instruction Set Architecture

https://www.youtube.com/watch?v=RI2NFy9Xe-0

The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA

Andrew Waterman
Yunsup Lee
David A. Patterson
Krste Asanovic

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2011-62

EPI

The European Processor Initiative (EPI) is a project currently implemented under the first stage of the Framework Partnership Agreement signed by the Consortium with the European Commission (FPA: 800928), whose aim is to design and implement a roadmap for a new family of low-power European processors for extreme scale computing, high-performance Big-Data and a range of emerging applications.

First RISC-V computer chip lands at the European Processor Initiative

EPAC accelerator runs its first 'Hello, World!' in-silico

Gareth Halfacree

Wed 22 Sep 2021 // 14:28 UTC

IT UNIVERSITY OF COPENHAGEN

CISC vs. RISC

- Original Debate
  - Strong opinions!
  - CISC proponents---easy for compiler, fewer code bytes
  - RISC proponents---better for optimizing compilers, can make run fast with simple chip design
- Current Status
  - For desktop processors, choice of ISA not a technical issue
    - With enough hardware, can make anything run fast
    - Code compatibility more important
  - x86-64 adopted many RISC features
    - More registers; use them for argument passing
  - For embedded processors, RISC makes sense
    - Smaller, cheaper, less power
    - Most cell phones use ARM processor

Relevance for Data Science

Amazon EC2 A1 instances deliver significant cost savings for scale-out and Arm-based applications such as web servers, containerized microservices, caching fleets, and distributed data stores that are supported by the extensive Arm ecosystem. A1 instances are the first EC2 instances powered by AWS Graviton Processors that feature 64-bit Arm Neoverse cores and custom silicon designed by AWS. These instances will also appeal to developers, enthusiasts, and educators across the Arm developer community. Most architecture-agnostic applications that can run on Arm cores could also benefit from A1 instances.

Reflections on ISA

G S Madhusudan • 1st
CEO/Co-Founder at InCore Semiconductors
2d • Edited •

Nvidia buys ARM - impact on various entities

A point to note - Technical merits of any ISA do not really matter.
Flexibility and openness of ISA, diversity of the Eco-system and broad industry ownership are key to the success of any ISA.

1. The x86 folks just got a nasty jolt. This will make the x86 story passe long term. Apple was the first blow, Intel should wake up and become ISA agnostic. Switching ISAs is not a big deal for Intel , the SoC/Uncore components are where Intel really provides value not the the x86 core per see. Intel has become a broad based supplier. AMD has a more difficult challenge. Moving to RISC-V is more of an imperative for them in the long run.

2. if NVIDIA acts dumb then RISC-V wins big time. I do not expect that to happen. I expect ARM ISA to be opened and embedded licenses to be made free. Only sensible thing to do. Now we can all stop drinking the RISC-V kool aid And battle ARM purely on technical merits. This is a battle worth fighting !

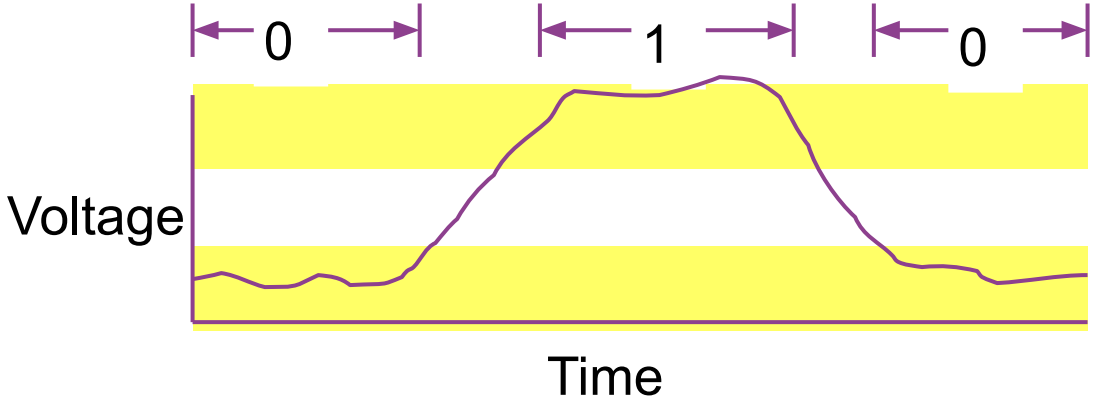
3. Qualcomm and Mediatek should be mildly worried. Building mobile SoCs is not for the faint of heart and NVIDIA even if it wants to (I am sure it does not) will take a while to get one out.

4 Anybody in the HPC, server and big chip AI/ML space. Your space just got crowded.

Logic Design => Digital Design

- Everything expressed in terms of values 0 and 1
- Communication
  - Low or high voltage on wire
- Computation
  - Compute Boolean functions
- Storage
  - Store bits of information

Digital Signals



Use voltage thresholds to extract discrete values from continuous signal

Simplest version: 1-bit signal

- Either high range (1) or low range (0)
- With guard range between them

Not strongly affected by noise or low quality circuit elements

- Can make circuits simple, small, and fast

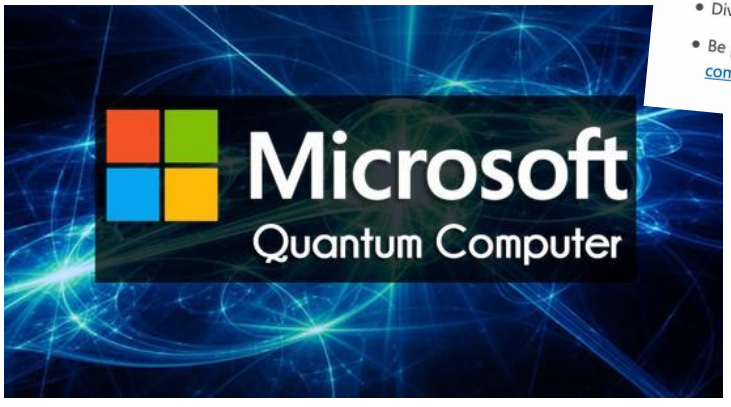
Looking Forward

Get ready for your Azure Quantum experience with the Quantum development kit

The Quantum development kit is an open-source development kit to develop quantum applications and solve optimization problems. It includes the high-level quantum programming language Q#, a set of libraries, simulators, support for Q# in environments like Visual Studio Code and Jupyter Notebooks, and interoperability with Python or .NET languages.

- Get started in Azure Quantum with Microsoft Learn.
- Dive into Q# programming with the Quantum katas.
- Be part of the Quantum development kit open-source community.

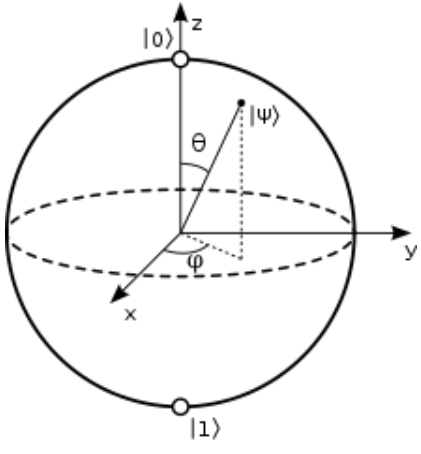
https://qdev.nbi.ku.dk/
https://qmath.ku.dk/



Solid-state qubits

Principal: Charles Marcus, Scientific Director

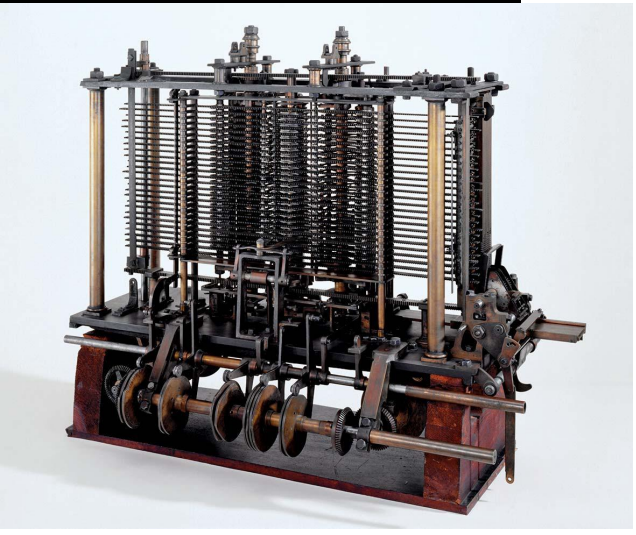
Microsoft Quantum Lab Copenhagen engages in the control and study of the properties of Majorana fermions, including characterization, fabrication, and test and measurement.



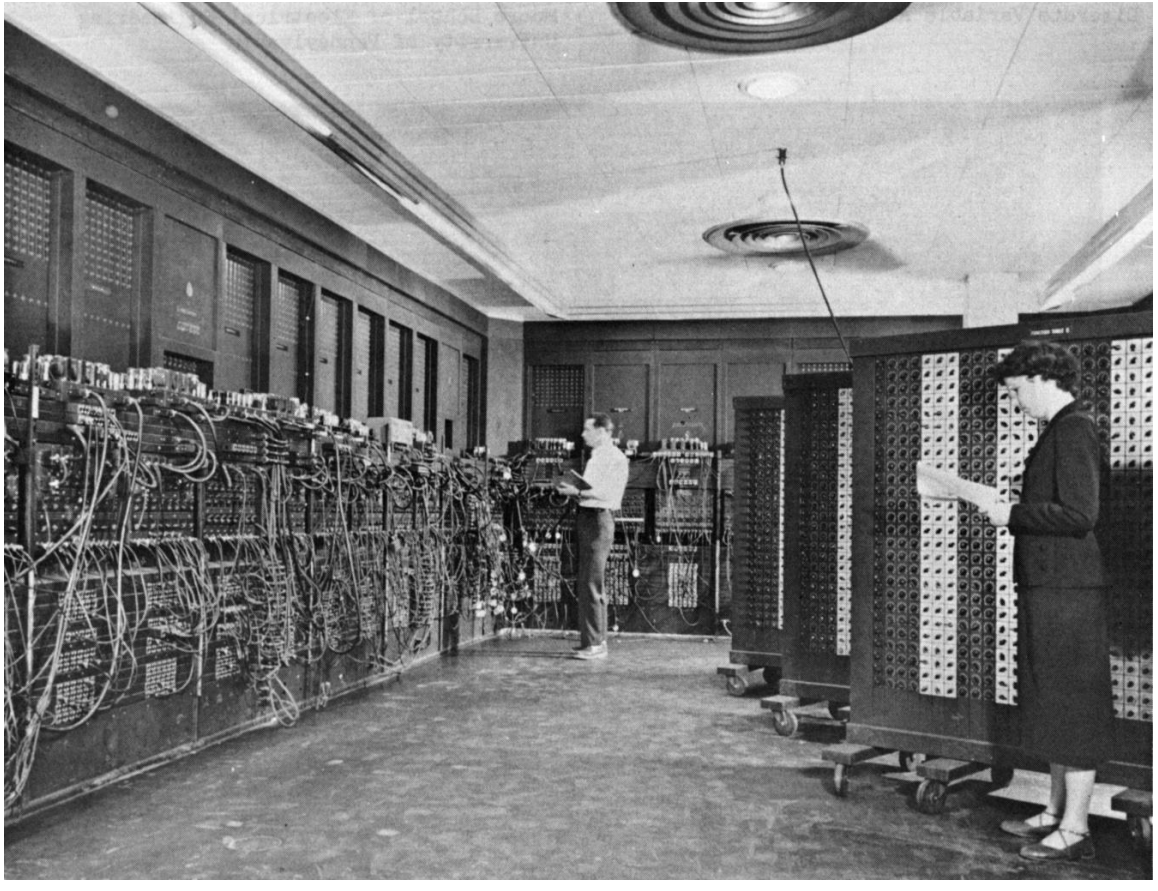
A qubit is a two-state (or two-level) quantum-mechanical system. The state space of a two-level quantum mechanical system (qubit) can be represented as a Riemann sphere.

IT UNIVERSITY OF COPENHAGEN

Looking Back

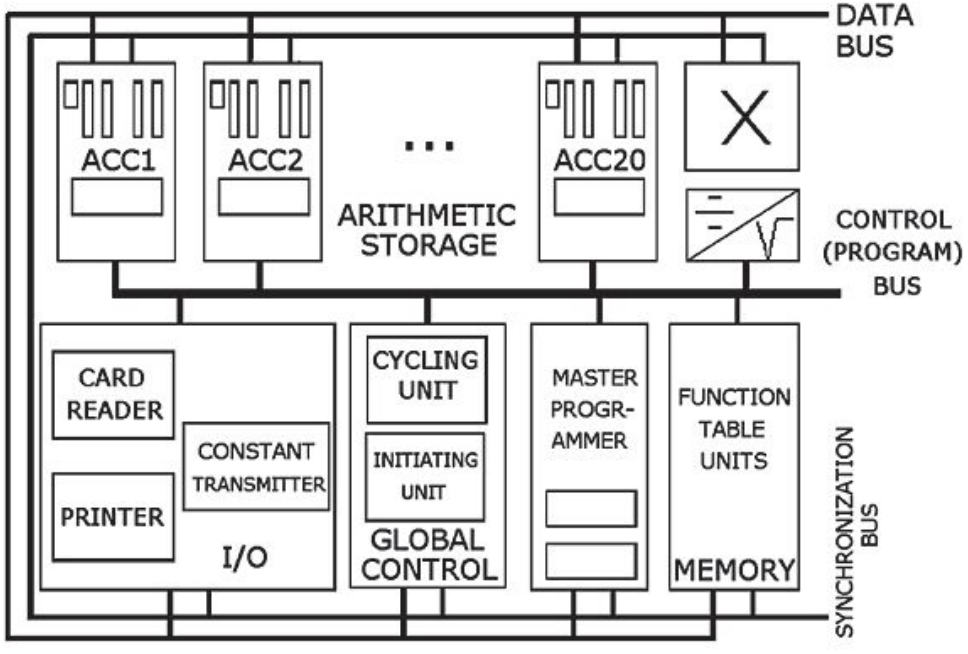


Analytical Engine, C.Babbage, 1837,



ENIAC, U.Penn, 1945

ENIAC Architecture



- Representation of digits by electric pulses
- Base 10 (data bus)
  - 10 wires for data
  - 1 wire for sign
- Hardwired functional units (add/multiply)
  - Vacuum tubes as basic component
  - Programming with flip-flops, switches and cables

Von Neumann Architecture

https://ieeexplore.ieee.org/document/238389

First Draft of a Report on the EDVAC

by

John von Neumann

Contract No. W–670–ORD–4926

Between the

United States Army Ordnance Department

and the

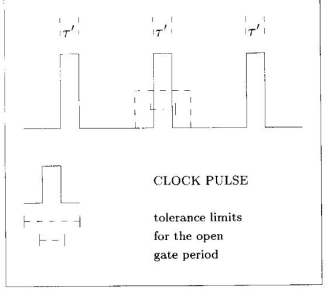
University of Pennsylvania

Moore School of Electrical Engineering
University of Pennsylvania

June 30, 1945

In order to achieve this, it is necessary to conceive the device as synchronous in the sense of 4.1. The central clock is best thought of as an electrical oscillator, which emits in every period τ a short, standard pulse of a length τ' of about $(1/5)\tau - (1/2)\tau$. The stimuli emitted nominally by an E-element are actually pulses of the clock, for which the pulse acts as a gate. There is clearly a wide tolerance for the period during which the gate must be kept open, to pass the clock-pulse without distortion. Cf. Figure 1. Thus the opening of the gate can be controlled by any electric delay device with a mean delay time τ, but considerable permissible dispersion. Nevertheless the effective synaptic delay will be τ with the full precision of the clock, and the stimulus is completely renewed and synchronized after each step. For a more detailed description in terms of vacuum tubes, cf. {}.

FIGURE 1



6.4 Let us now return to the description of the E-elements.

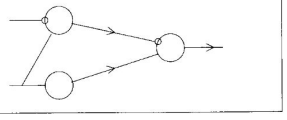
An E-element receives the stimuli of its antecedents across excitatory synapses: —○→, or inhibitory synapses: —●→. As pointed out in 4.2, we will consider E-elements with thresholds 1, 2, and 3, that is, which get excited by these minimum numbers of simultaneous excitatory stimuli. All inhibitory stimuli, on the other hand, will be assumed to be absolute. E-elements with the above thresholds will be denoted by ○, ②, ③, respectively.

Since we have a strict synchronism of stimuli arriving only at times which are integer multiples of τ, we may disregard phenomena of tiring, facilitation, etc. We also disregard relative inhibition, temporal summation of stimuli, changes of threshold, changes of synapses, etc. In all this we are following the procedure of W.J. MacCulloch and W. Pitts (cf. loc. cit. 4.2). We will also use E-elements with double synaptic delay 2τ: —○→→, and mixed types: —○→←.

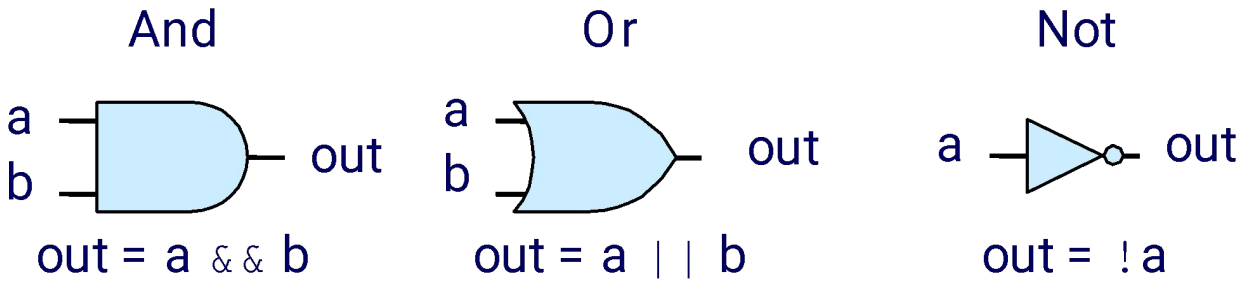
The reason for our using these variants is that they give a greater flexibility in putting together simple structures, and they can all be realized by vacuum tube circuits of the same complexity.

It should be observed that the authors quoted above have shown that most of these elements can be built up from each other. Thus —○→→ is clearly equivalent to —○→○→, and in the case of ②→ at least ⊐②→ is equivalent to the network of Figure 2. However, it would seem to be misleading in our application to represent these functions as if they required 2 or 3 E-elements, since their complexity in a vacuum tube realization is not essentially greater than that of the simplest E-element —○→, cf. {}.

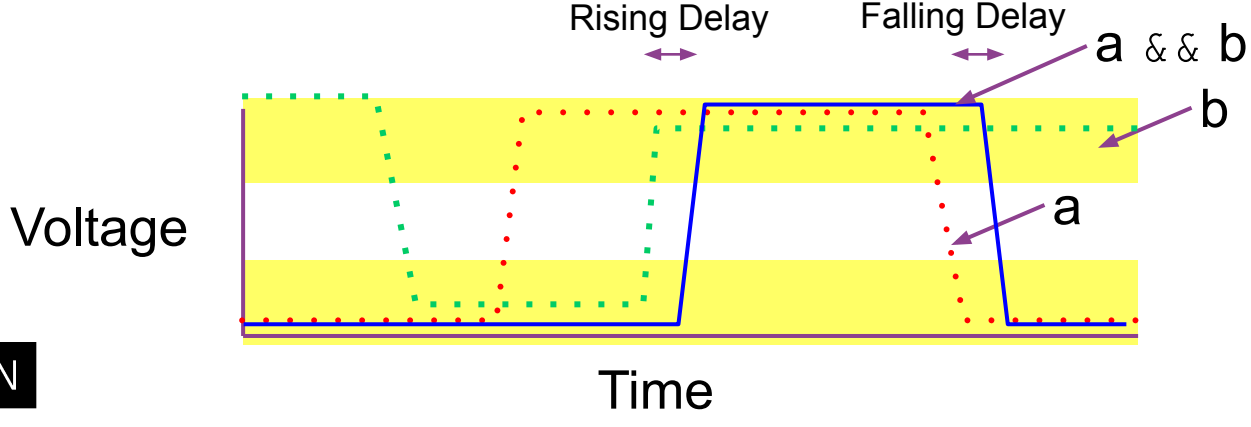
FIGURE 2



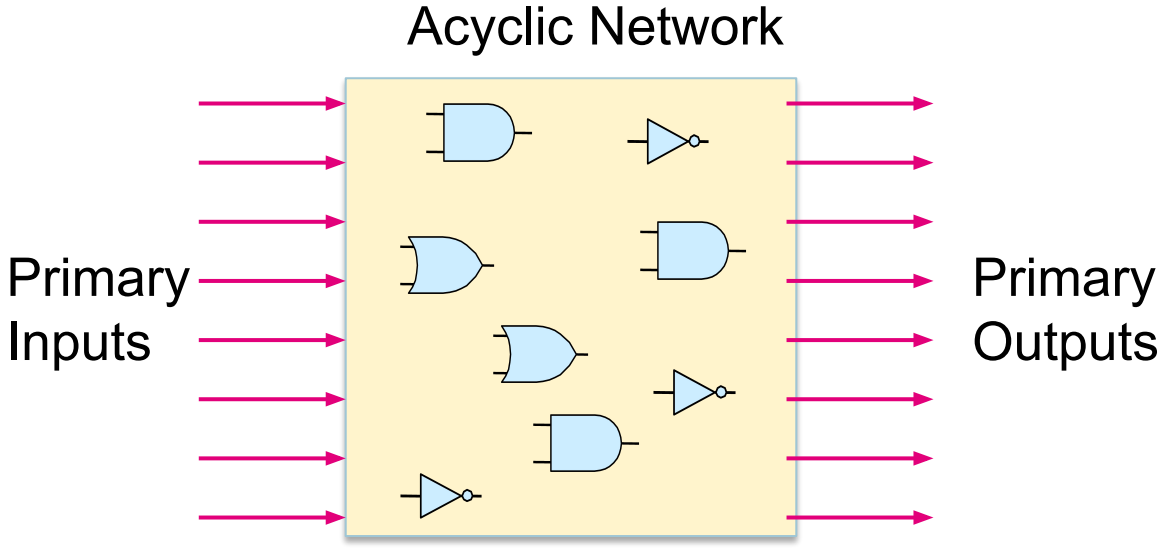
Computing with Logic Gates



- Outputs are Boolean functions of inputs
- Respond continuously to changes in inputs
  - With some, small delay



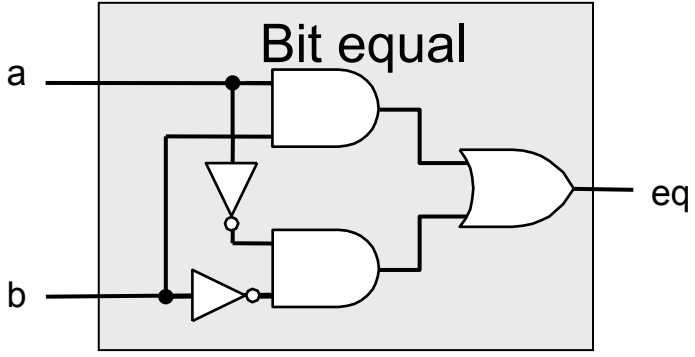
Combinational Circuits



Acyclic Network of Logic Gates

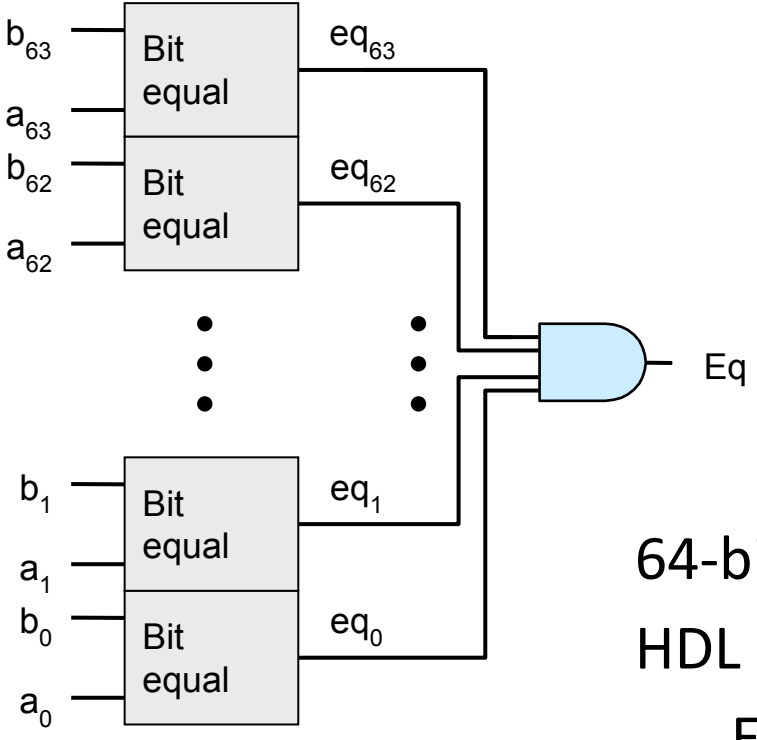
- Vertices are logic gates; edges are signal transport
- Continously responds to changes on primary inputs
- Primary outputs become (after some delay) Boolean functions of primary inputs

Bit Equality

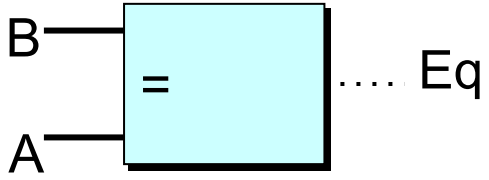



```
bool eq = (a&&b)||(!a&&!b)
```

Word Equality



Word-Level Representation



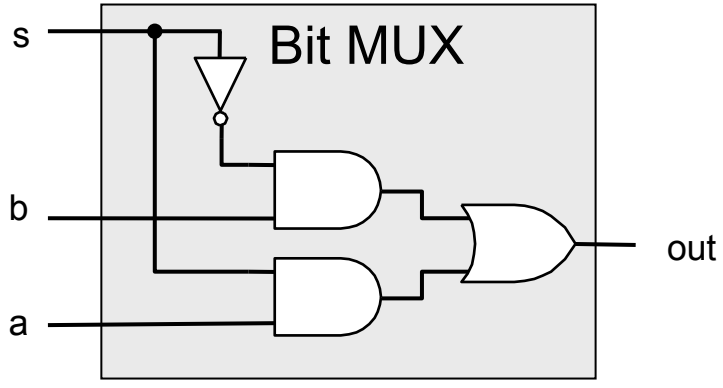

```
bool Eq = (A == B)
```

64-bit word size

HDL representation

- Equality operation
- Generates Boolean value

Bit-Level Multiplexor



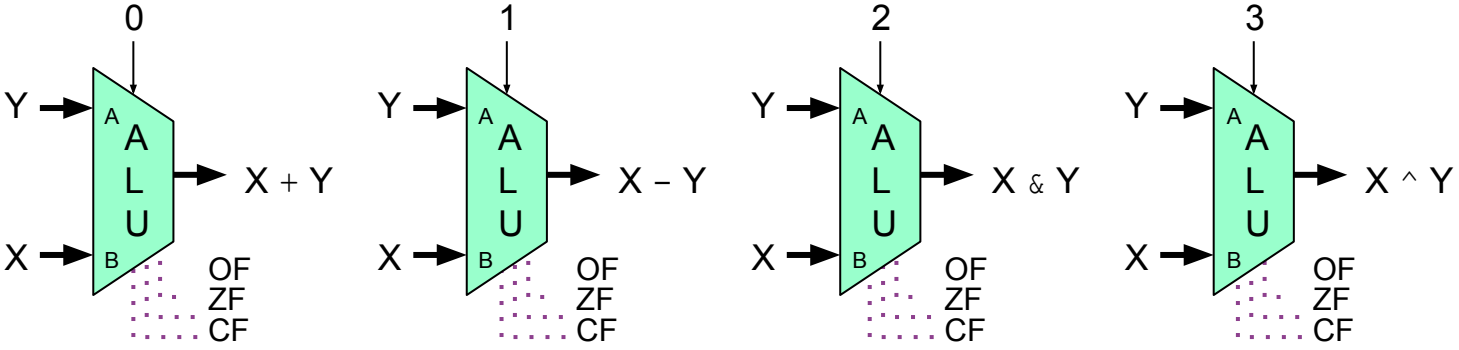

```
bool out = (s&&a)||(!s&&b)
```

Control signal s

Data signals a and b

Output a when s=1, b when s=0

Arithmetic Logic Unit



Combinational logic

- Continuously responding to inputs

Control signal selects function computed

- Corresponding to 4 arithmetic/logical operations
- Overflow, Carry, Zero flags: OF, CF, ZF

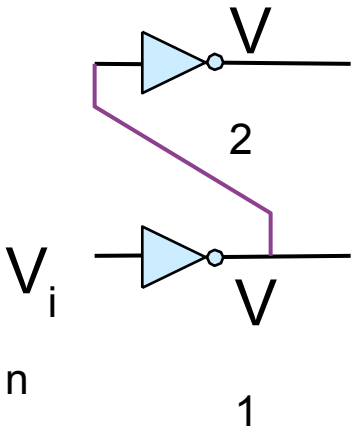
Storing bits

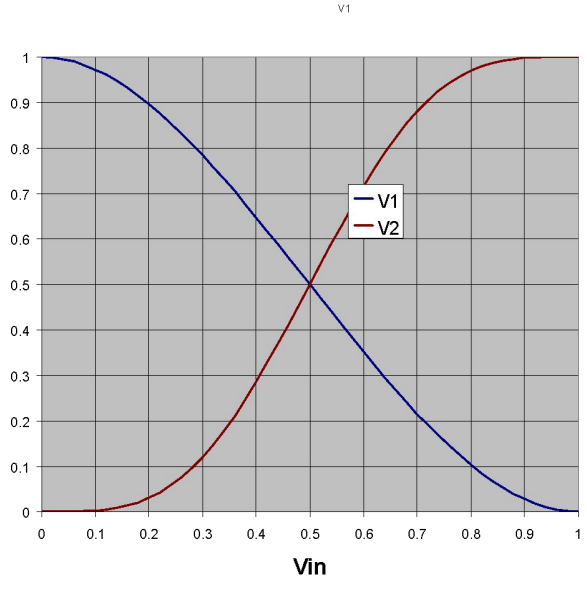
Boolean logic and acyclic circuit of logic gates used to represent arithmetic operations

Can logic gates be used to store bits?

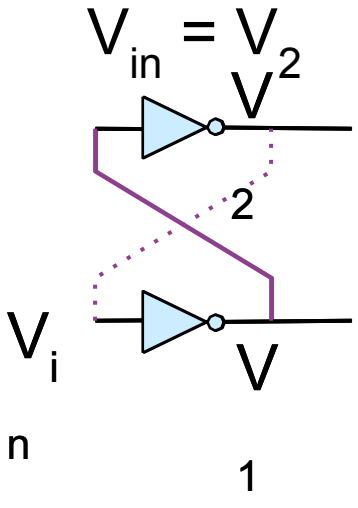
Yes! With cyclic circuits of logic gates.

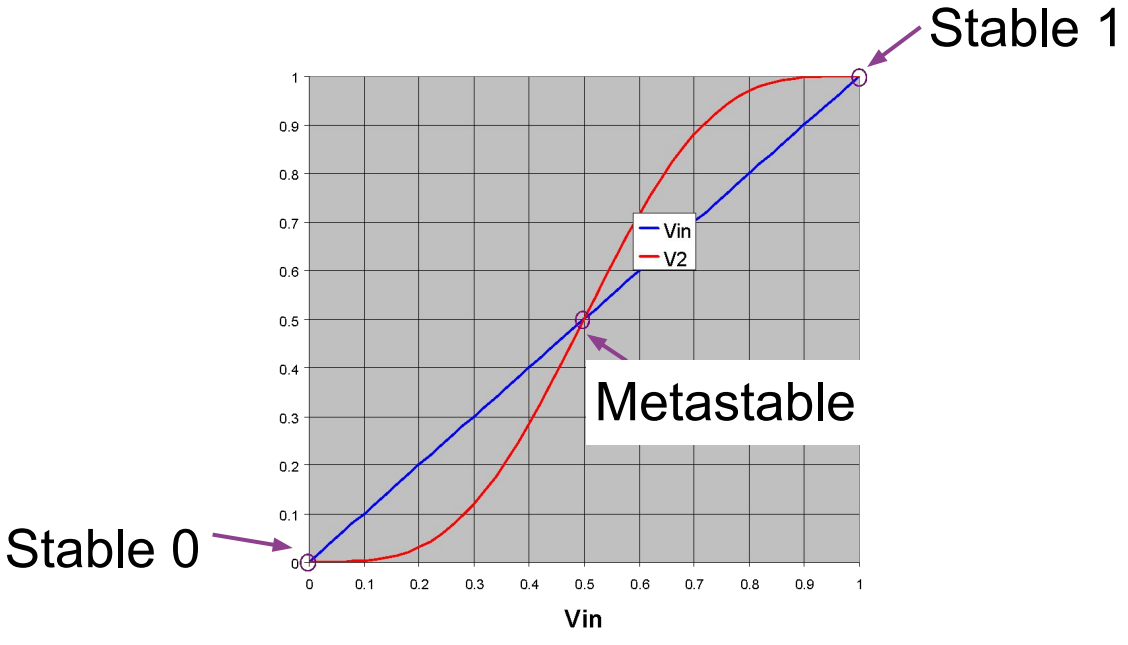
Storing 1 Bit



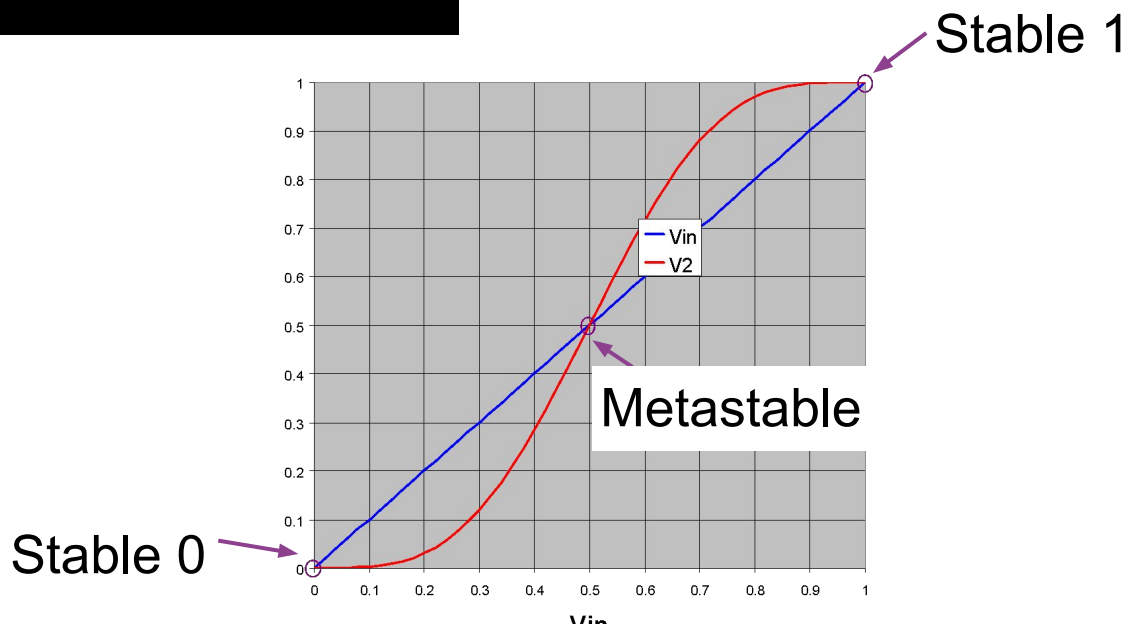


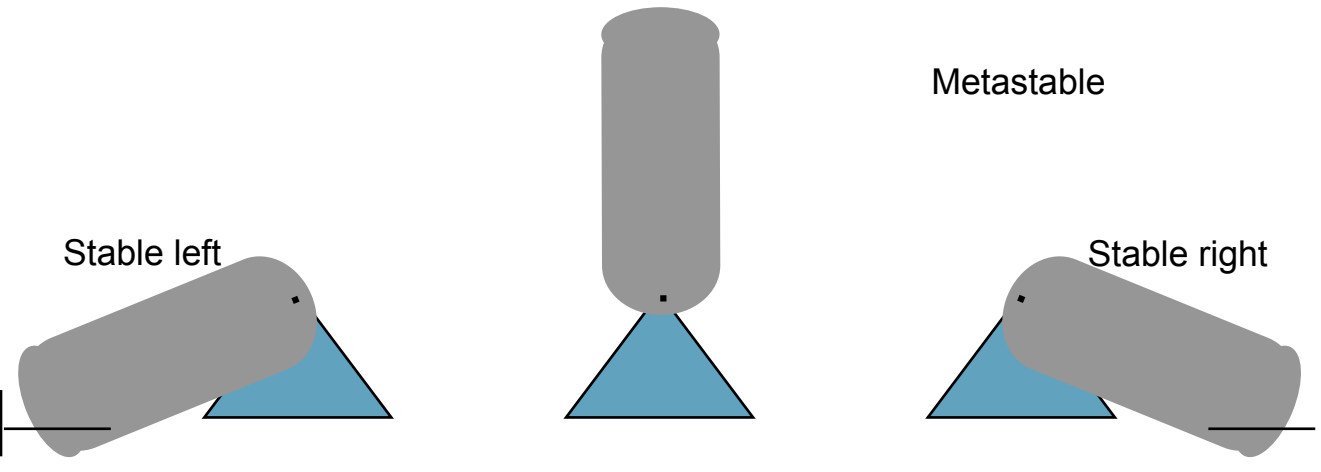
Storing 1 Bit (cont.)





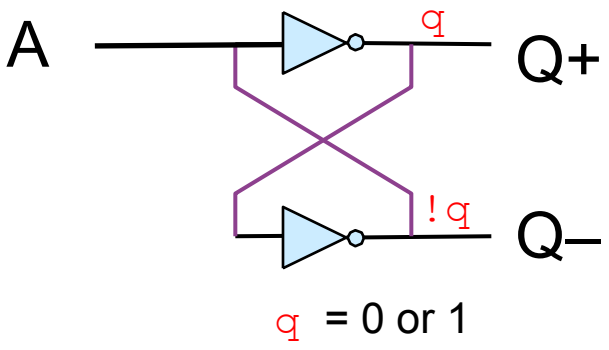
Physical Analogy





Storing 1 Bit (cont.)

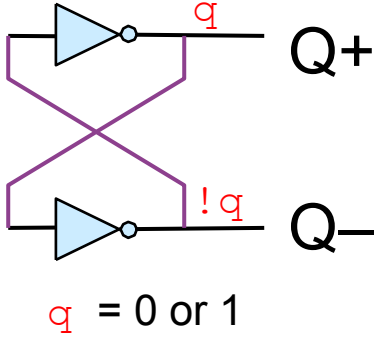
Bistable Element



When A is 0

Q+ = q = 1

Q- = !q = 0



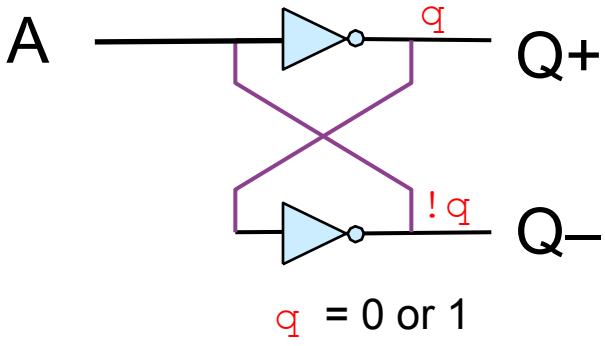
When no input

Q+ = q = 1

Q- = !q = 0

Storing 1 Bit (cont.)

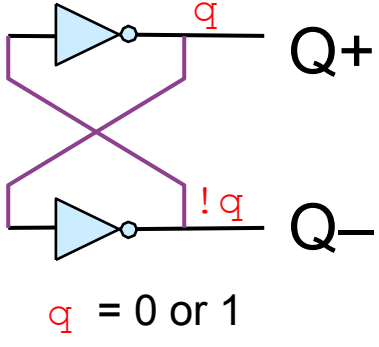
Bistable Element



When A is 1

Q+ = q = 0

Q- = !q = 1



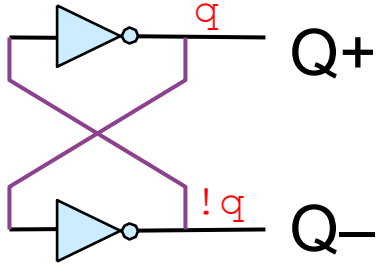
When no input

Q+ = q = 0

Q- = !q = 1

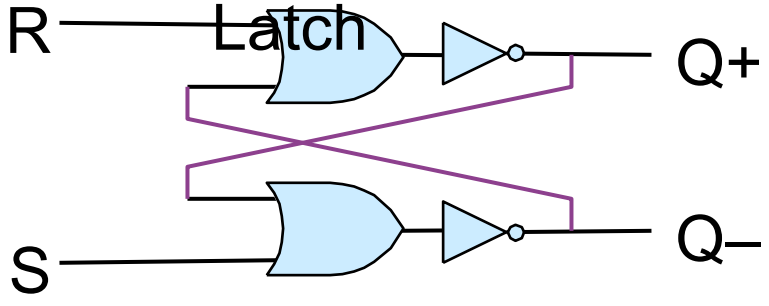
Storage Element: Latch

Bistable element

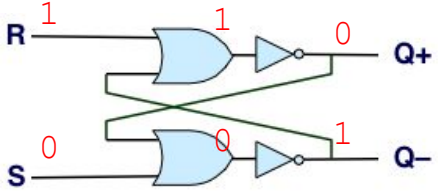


q = 0 or 1

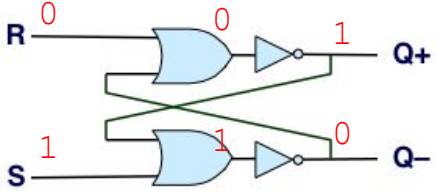
R-S Latch



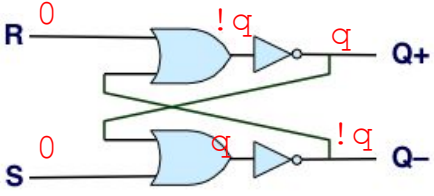
Resetting



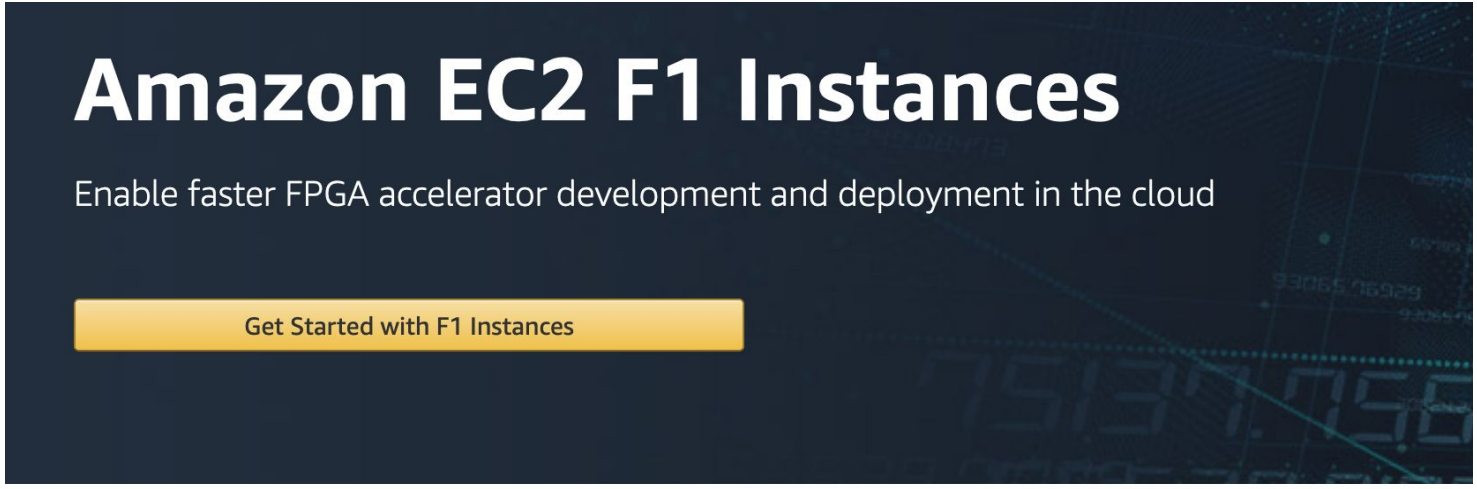
Setting



Storing

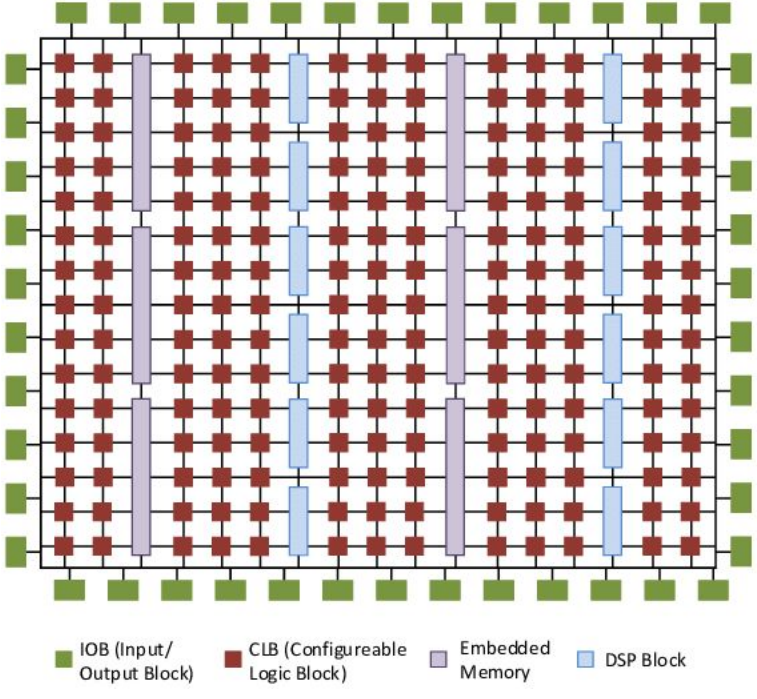


Relevance for Data Science



Amazon EC2 F1 instances use FPGAs to enable delivery of custom hardware accelerations. F1 instances are easy to program and come with everything you need to develop, simulate, debug, and compile your hardware acceleration code, including an FPGA Developer AMI and supporting hardware level development on the cloud. Using F1 instances to deploy hardware accelerations can be useful in many applications to solve complex science, engineering, and business problems that require high bandwidth, enhanced networking, and very high compute capabilities. Examples of target applications that can benefit from F1 instance acceleration are genomics, search/analytics, image and video processing, network security, electronic design automation (EDA), image and file compression and big data analytics.

FPGAs



Configurable Logic Block:

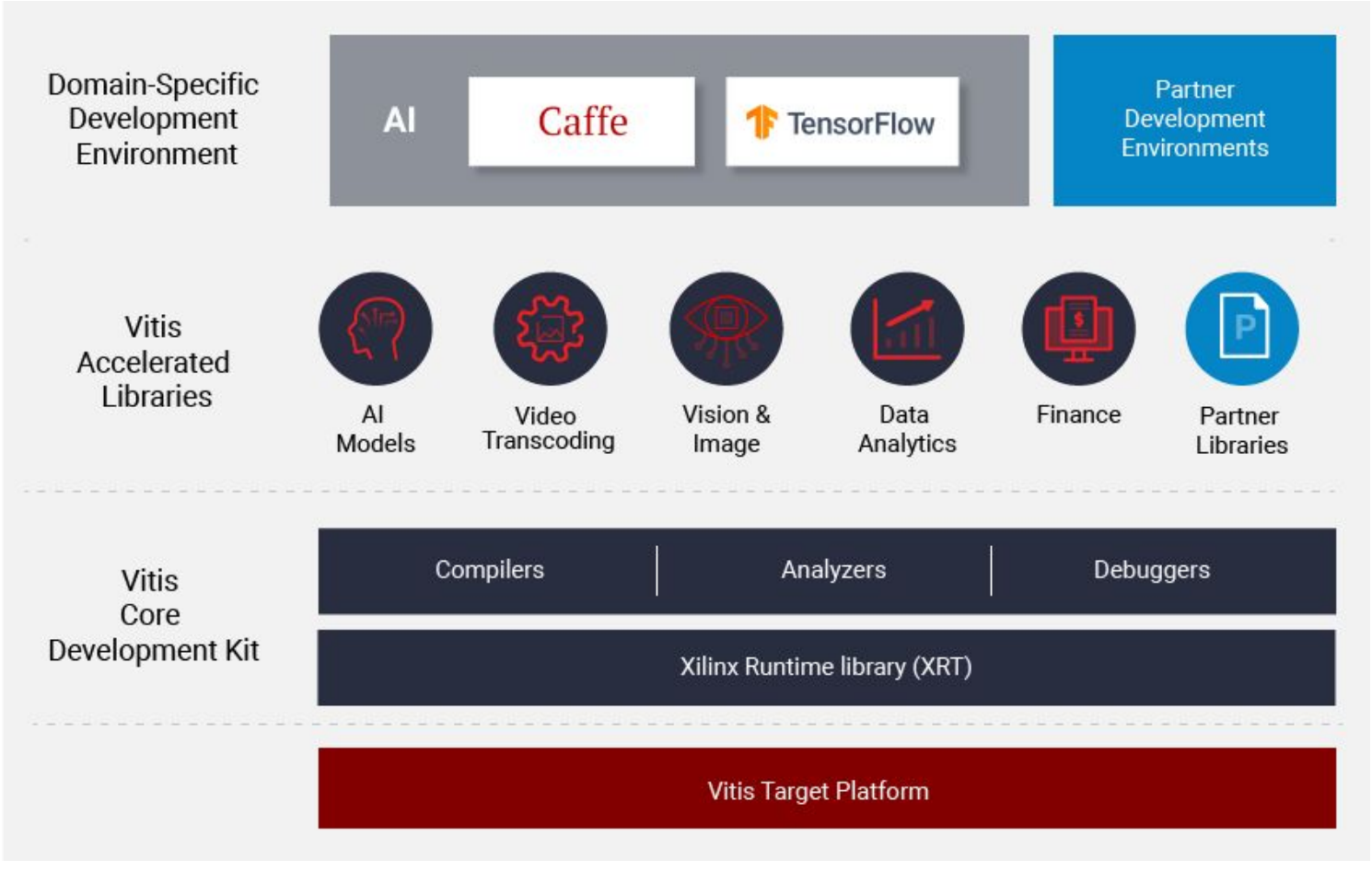
- Lookup Tables (LUT)
  - Truth table
  - Up to 6 inputs
- Storage Elements
  - Flip-flop or latch
  - Control signals

+ Multiplexers, Carry Logic, Distributed RAM, Shift register

https://www.nandland.com/articles/boolean-algebra-using-look-up-tables-lut.html
https://www.nandland.com/articles/flip-flop-register-component-in-fpga.html

https://www.xilinx.com/support/documentation/user_guides/ug574-ultrascale-clb.pdf

Programming FPGAs

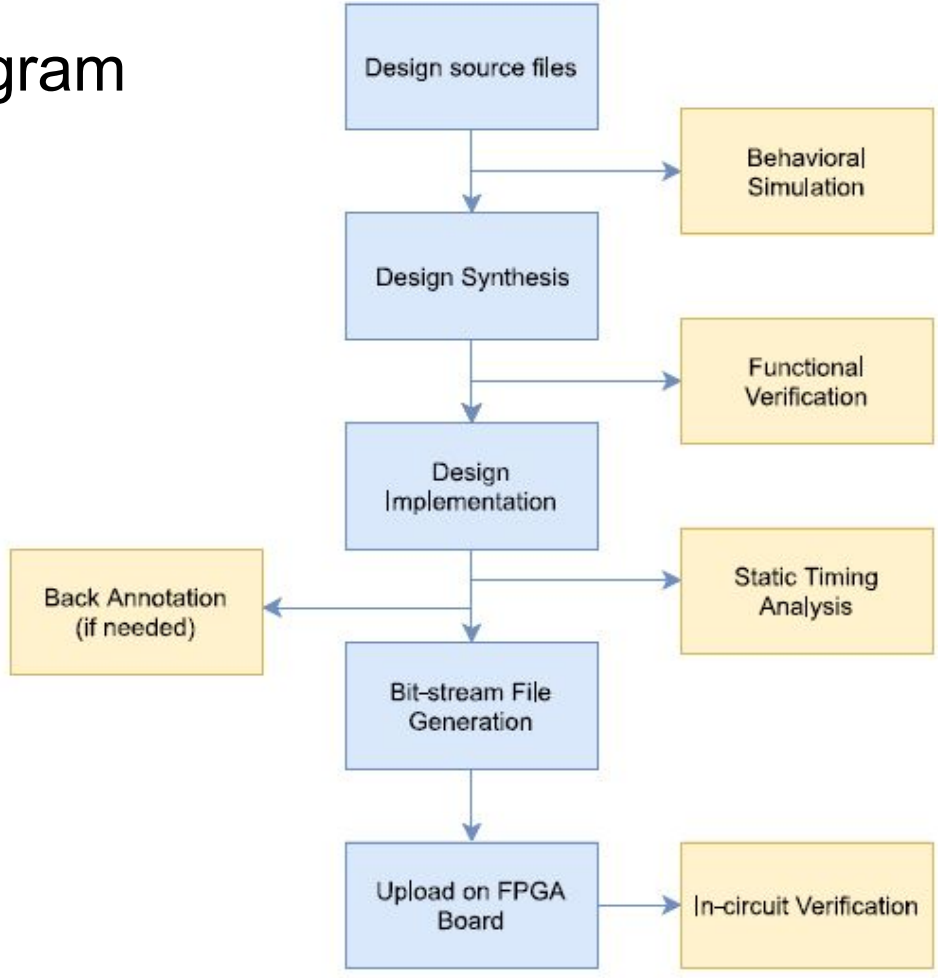


https://www.xilinx.com/products/design-tools/vitis.html

FPGA Synthesis Process

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity toplevel_tb_wfile is
--  Port ( );
end toplevel_tb_wfile;

architecture Behavioral of toplevel_tb_wfile is

signal clk : std_logic := '1';
signal rst: std_logic := '1';

signal a : std_logic_vector(7 downto 0);

begin
    process begin
        clk <= not clk;
        wait for 10ns;
    end process;

    process begin
        wait for 40ns;
        rst <= '0';
    end process;

    process (clk) begin
        if rising_edge(clk)
        then
            if rst='1' then
                a <= (others => '0');
            else
                a <= a+1;
            end if;
        end if;
    end process;

    variable line_v : line;
    file read_file : text;
    variable slv_v : std_logic_vector(7 downto 0);

    file_open(read_file, "source.txt", read_mode);
    while not endfile(read_file) loop
      readline(read_file, line_v);
      hread(line_v, slv_v);
    end loop;
    file_close(read_file);
end Behavioral;
```

VHDL program



Hardware Description Language

Very simple hardware description language

Can only express limited aspects of hardware operation

Data Types

`bool`: Boolean

- `a`, `b`, `c`, …

`int`: words

- `A`, `B`, `C`, …
- Does not specify word size---bytes, 64-bit words, …

Statements

```
bool a = bool-expr ;

int A = int-expr ;
```

HDL Operations

Classify by type of value returned

Boolean Expressions

Logic Operations

- `a && b`, `a || b`, `!a`

Word Comparisons

- `A == B`, `A != B`, `A < B`, `A <= B`, `A >= B`, `A > B`

Set Membership

- `A in { B, C, D }`
  - Same as `A == B || A == C || A == D`

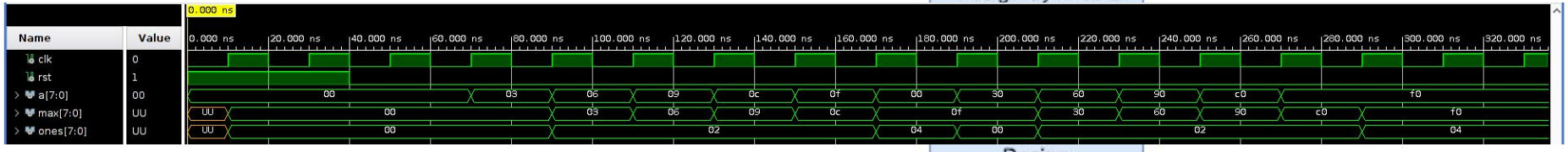
Word Expressions

Case expressions

- `[ a : A; b : B; c : C ]`
- Evaluate test expressions `a`, `b`, `c`, … in sequence
- Return word expression `A`, `B`, `C`, … for first successful test

FPGA Synthesis Process

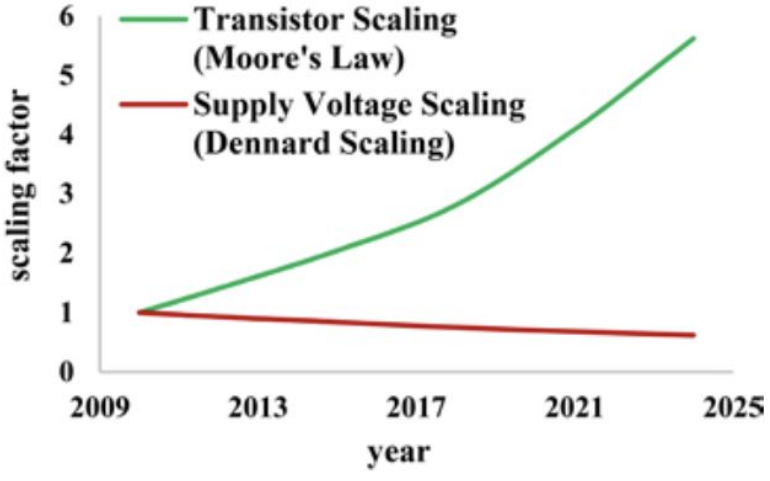
Functional Verification



Logic Design Summary

- Computation
  - Performed by combinational logic
  - Computes Boolean functions
  - Continuously reacts to input changes (clock)
- Storage elements combined into
  - Random-access memories
    - Hold multiple words
    - Possible multiple read or write ports
    - Read word when address input changes
    - Write word as clock rises

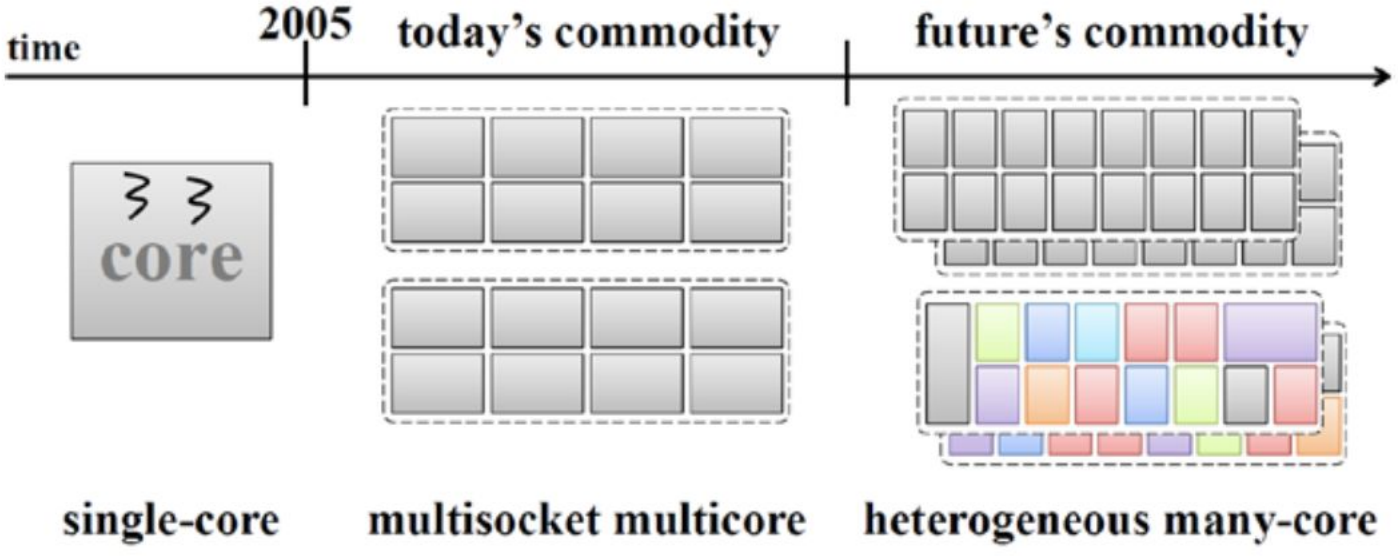
Moore’s Law and Dennard Scaling



Gordon Moore (1965): The number of transistors in a dense integrated circuit doubles approximately every two years.

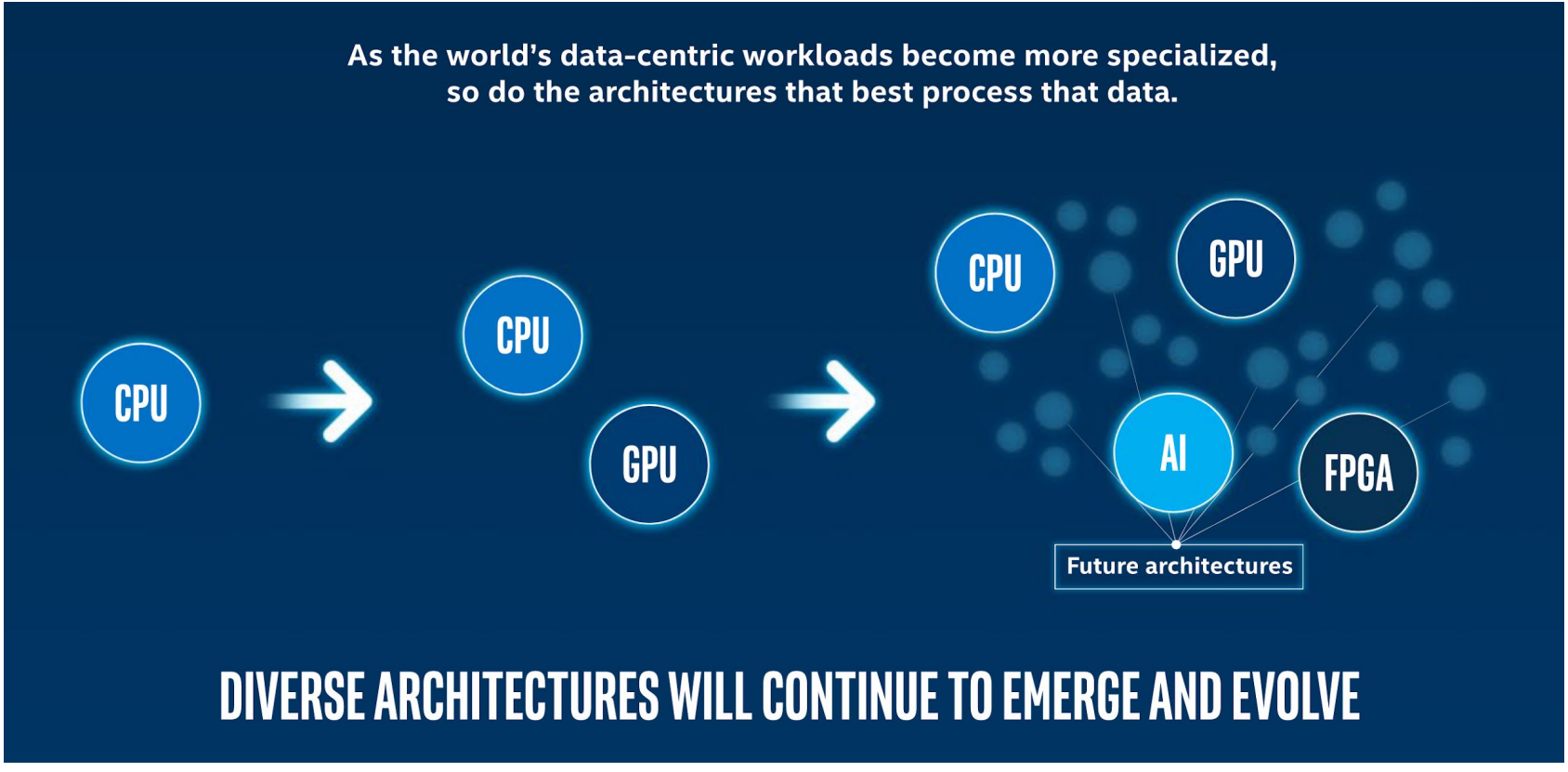
Robert Dennard (1974): As transistors get smaller their power density stays constant

Consequence

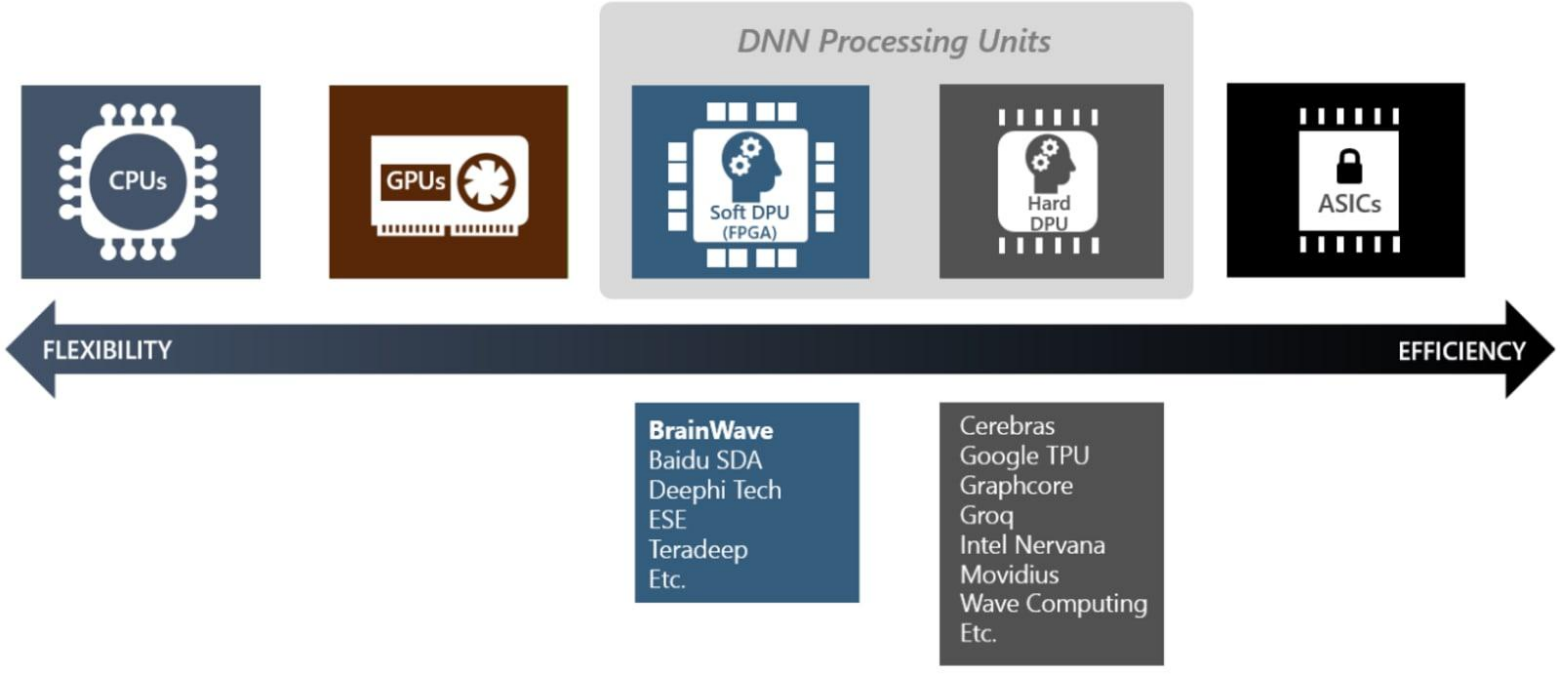


The situation is getting more complex

https://newsroom.intel.com/wp-content/uploads/sites/11/2019/11/intel-oneapi-info.pdf



Diverse Cores



Take-Aways

Choice of CPU based on ISA + extensions (security, performance).

FPGA-based accelerators programmed as acyclic networks of logic gates.

Trends lead to diverse cores (Dennard scaling).