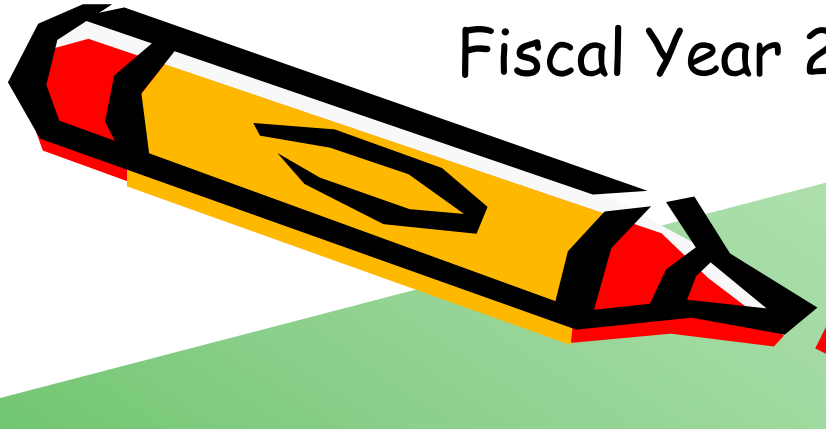
Fiscal Year 2018

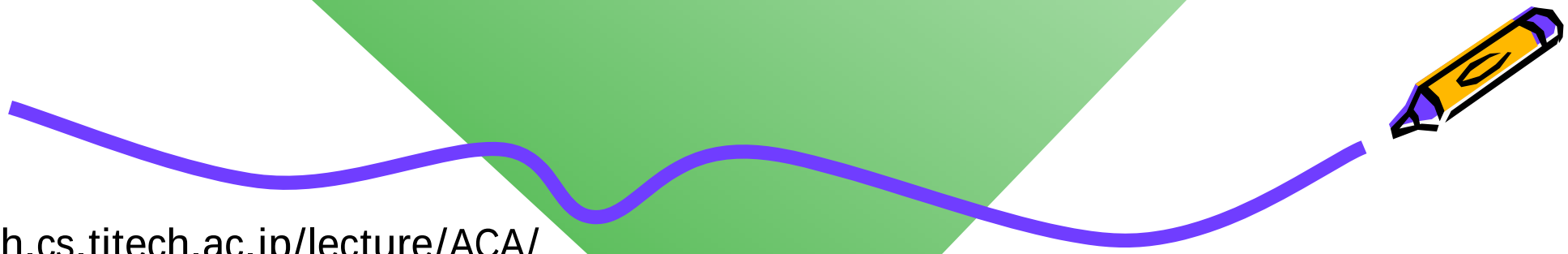
Ver. 2018-11-28a



Course number: CSC.T433
School of Computing,
Graduate major in Computer Science

Advanced Computer Architecture

1. Design and Analysis of Computer Systems



www.arch.cs.titech.ac.jp/lecture/ACA/
Room No.W936
Mon 13:20-14:50, Thr 13:20-14:50

Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

Syllabus (1/3)

Course description and aims

This course aims to provide students with cutting-edge technologies and future trends of computer architecture with focusing on a microprocessor which plays an important role in the downsizing, personalization, and improvement of performance and power consumption of computer systems such as PCs, personal mobile devices, and embedded systems.
In this course, first, along with important concepts of computer architecture, students will learn from instruction set architectures to mechanisms for extracting instruction level parallelism used in out-of-order superscalar processors. After that, students will learn mechanisms for exploiting thread level parallelism adopted in multi-processors and multi-core processors.

Student learning outcomes

By taking this course, students will learn:
(1) Basic principles for building today's high-performance computer systems
(2) Mechanisms for extracting instruction level parallelism used in high-performance microprocessors
(3) Methods for exploiting thread level parallelism adopted in multi-processors and multi-core processors
(4) New inter-relationship between software and hardware

Keywords

Computer Architecture, Processor, Embedded System, multi-processor, multi-core processor

Competencies that will be developed

Intercultural skills	Communication skills	Specialist skills	Critical thinking skills	Practical and/or problem-solving skills
-	-	✓	-	-

Class flow

Before coming to class, students should read the course schedule and check what topics will be covered. Required learning should be completed outside of the classroom for preparation and review purposes.

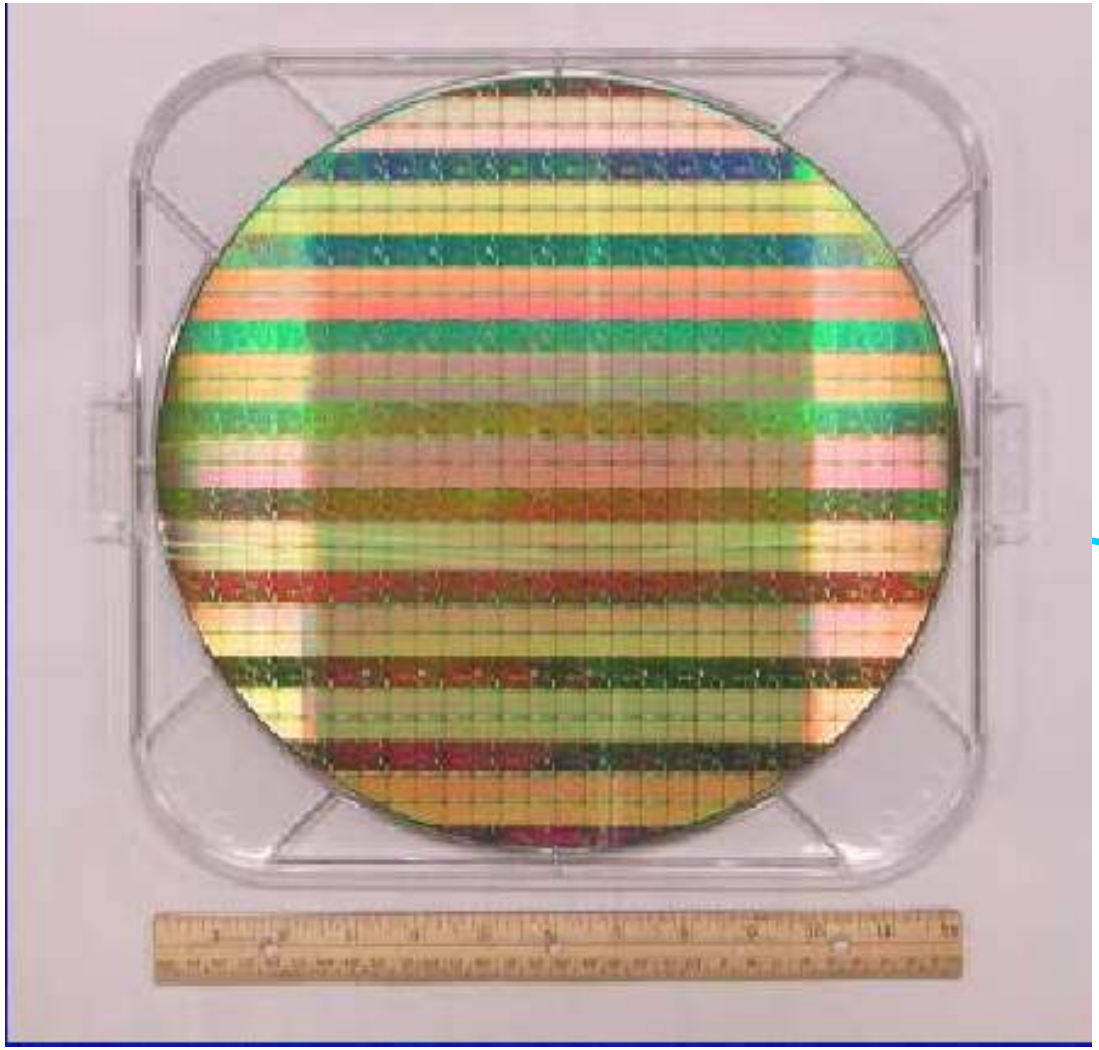
Syllabus (2/3)

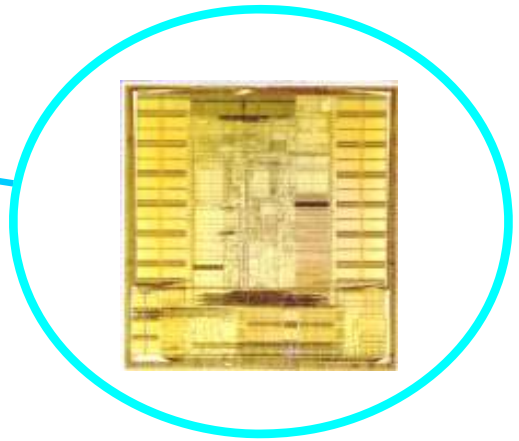
Textbook(s)
John L. Hennessy, David A. Patterson. Computer Architecture A Quantitative Approach, Fifth Edition. Morgan Kaufmann Publishers Inc., 2012
Reference books, course materials, etc.
William James Dally, Brian Patrick Towles. Principles and Practices of Interconnection Networks. Morgan Kaufman Publishers Inc., 2004.
Assessment criteria and methods
Students will be assessed on their understanding of instruction level parallelism, multi-processor, and thread level parallelism. Students' course scores are based on the mid-term report (40%) and final report (60%).
Related courses
CSC.T363 : Computer Architecture CSC.T341 : Computer Logic Design
Prerequisites (i.e., required knowledge, skills, courses, etc.)
No prerequisites are necessary, but enrollment in the related courses is desirable.
Contact information (e-mail and phone)
Kise Kenji: kise@cs.titech.ac.jp, 03-5734-3698 Miyazaki Jun: miyazaki@cs.titech.ac.jp, 03-5734-2687
Office hours
Contact by e-mail in advance to schedule an appointment.

Processor fabrication: ingot and wafer

Silicon, the most abundant element on earth except for oxygen, is used because it is a natural semiconductor.

Processor fabrication: wafer and die

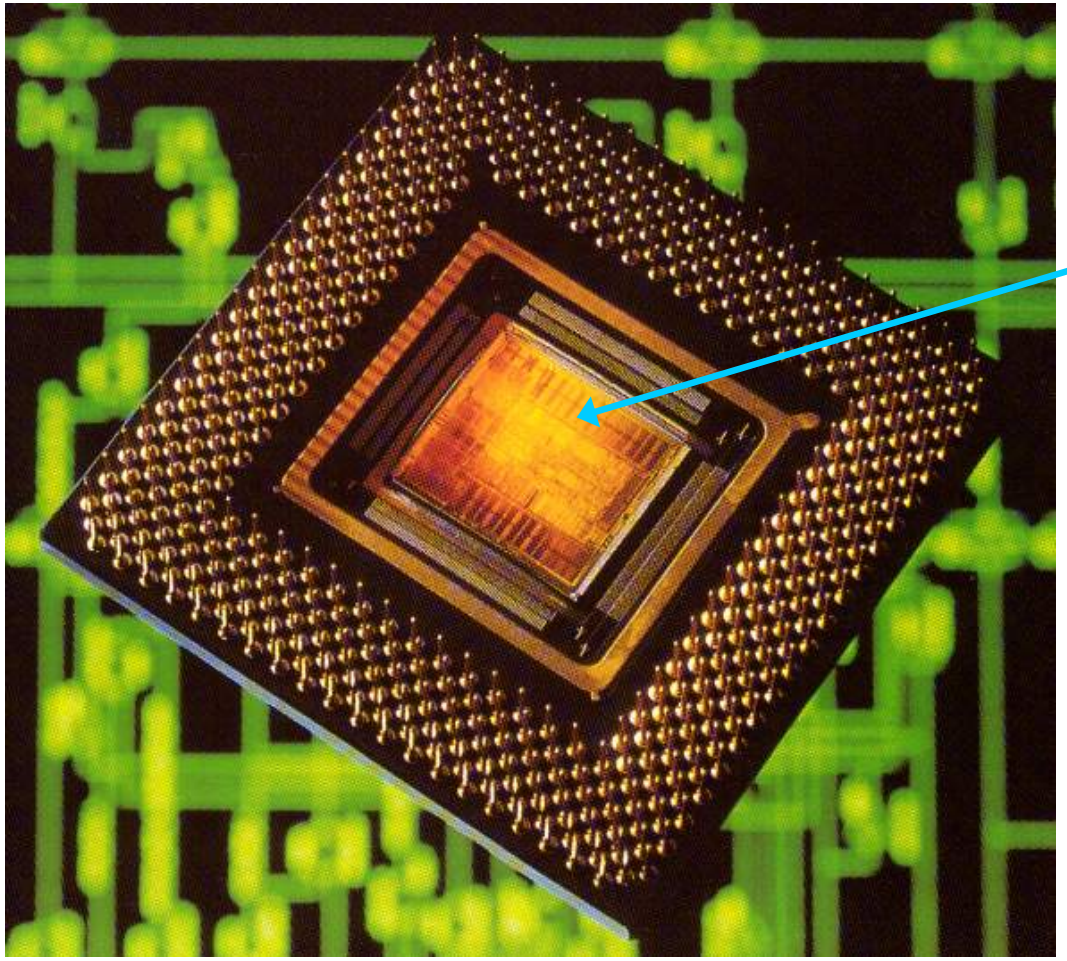




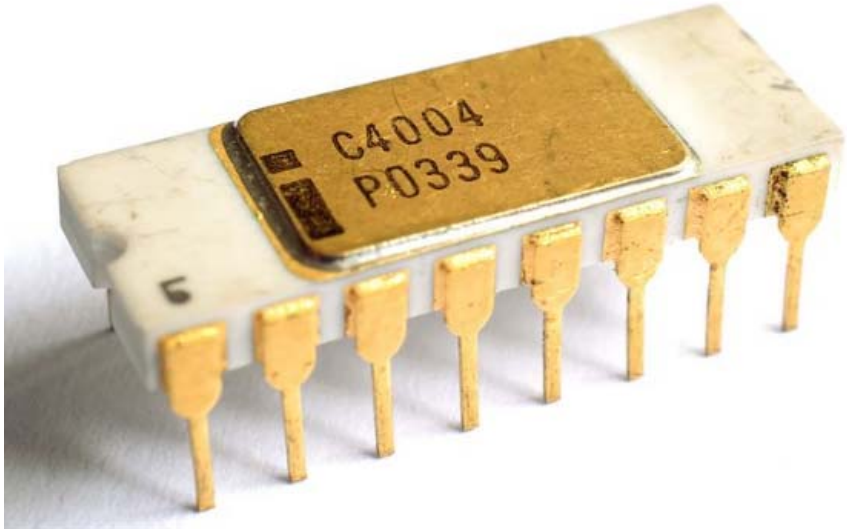
Die

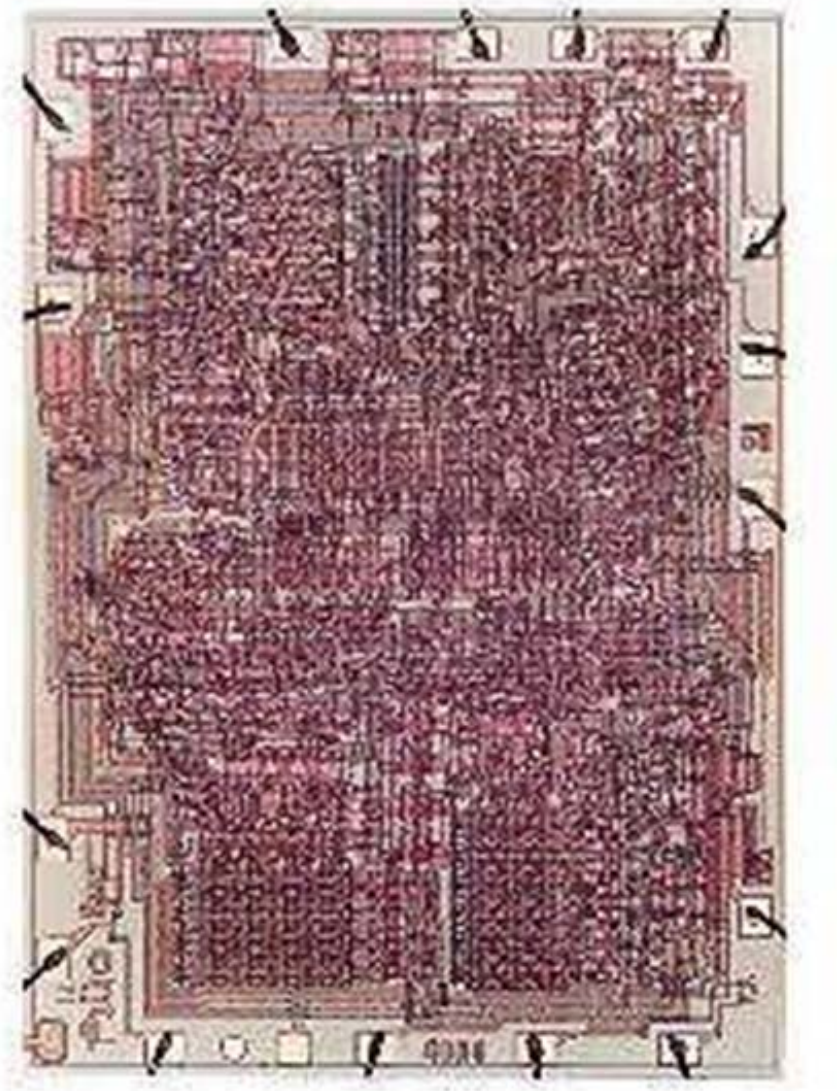
Intel, Industry-Leading Transistor Performance Demonstrated on Intel's 90-nanometer Logic Process

Processor fabrication: die and packaging



The birth of microprocessors





Name	Year	# of transistors
Intel 4004	1971	2,250

Moore's Law

- Moore's law is the observation that the number of transistors in a dense integrated circuit doubles about every two years. The observation is named after Gordon Moore, the co-founder of Fairchild Semiconductor and Intel, whose 1965 paper described a doubling every year in the number of components per integrated circuit, and projected this rate of growth would continue for at least another decade. In 1975, looking forward to the next decade, he revised the forecast to doubling every two years. The period is often quoted as 18 months because of a prediction by Intel executive David House (being a combination of the effect of more transistors and the transistors being faster).

WIKIPEDIA

Moore's Law

VISUALIZING PROGRESS

If transistors were people

If the transistors in a microprocessor were represented by people, the following timeline gives an idea of the pace of Moore's Law.

2,300
Average music hall capacity

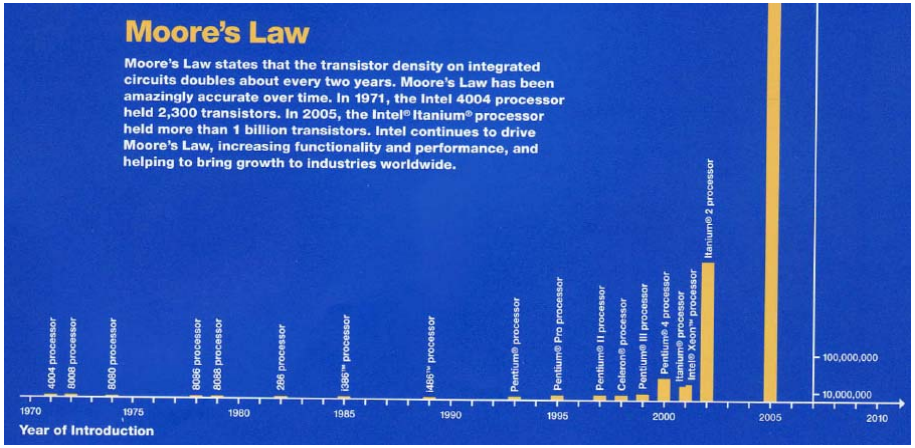
134,000
Large stadium capacity

32 Million
Population of Tokyo

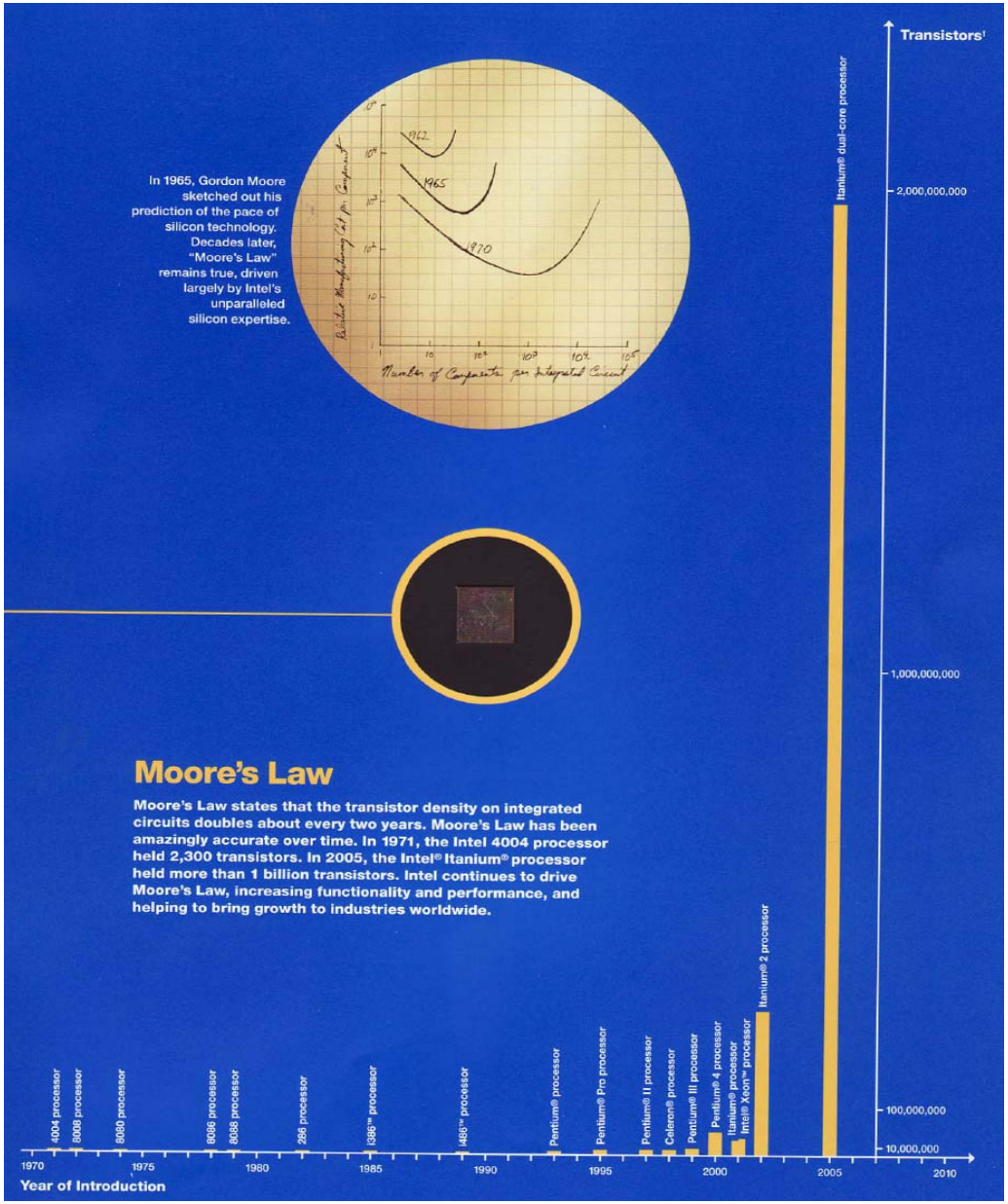
1.3 Billion
Population of China

1970	1980	1990	2000	2011
Intel 4004	Intel 286		Pentium III	Core i7 Extreme Edition

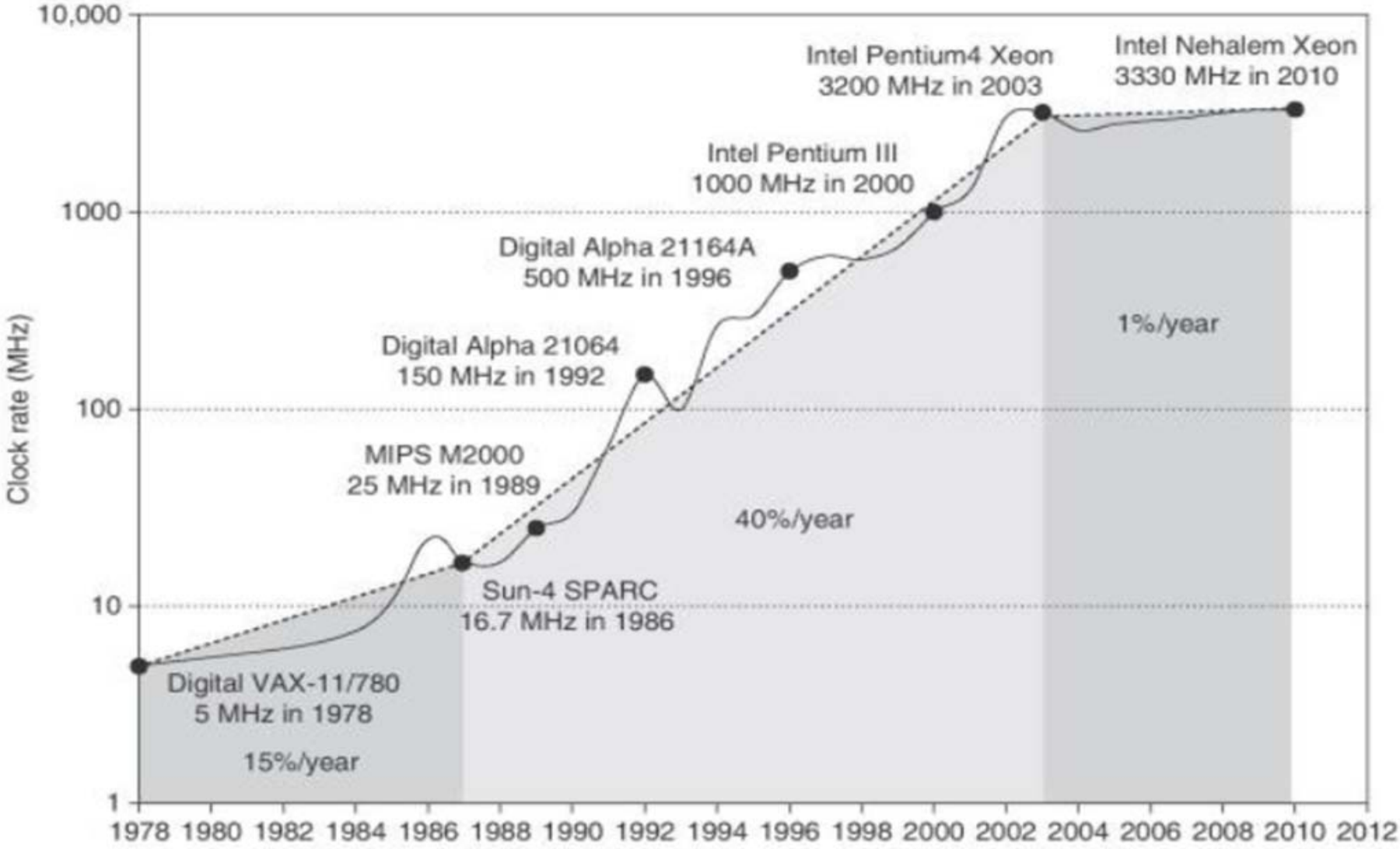
Now imagine that those 1.3 billion people could fit onstage in the original music hall. That's the scale of Moore's Law.



Moore's Law


In 1965, Gordon Moore sketched out his prediction of the pace of silicon technology. Decades later, "Moore's Law" remains true, driven largely by Intel's unparalleled silicon expertise.
Moore's Law
Moore's Law states that the transistor density on integrated circuits doubles about every two years. Moore's Law has been amazingly accurate over time. In 1971, the Intel 4004 processor held 2,300 transistors. In 2005, the Intel® Itanium® processor held more than 1 billion transistors. Intel continues to drive Moore's Law, increasing functionality and performance, and helping to bring growth to industries worldwide.
Transistors
2,000,000,000
1,000,000,000
100,000,000
10,000,000
Itanium® dual-core processor
4004 processor
8008 processor
8080 processor
8086 processor
8088 processor
286 processor
386™ processor
486™ processor
Pentium® processor
Pentium® Pro processor
Pentium® II processor
Celeron® processor
Pentium® III processor
Pentium® 4 processor
Itanium® processor
Intel® Xeon™ processor
Itanium® 2 processor
1970
1975
1980
1985
1990
1995
2000
2005
2010
Year of Introduction

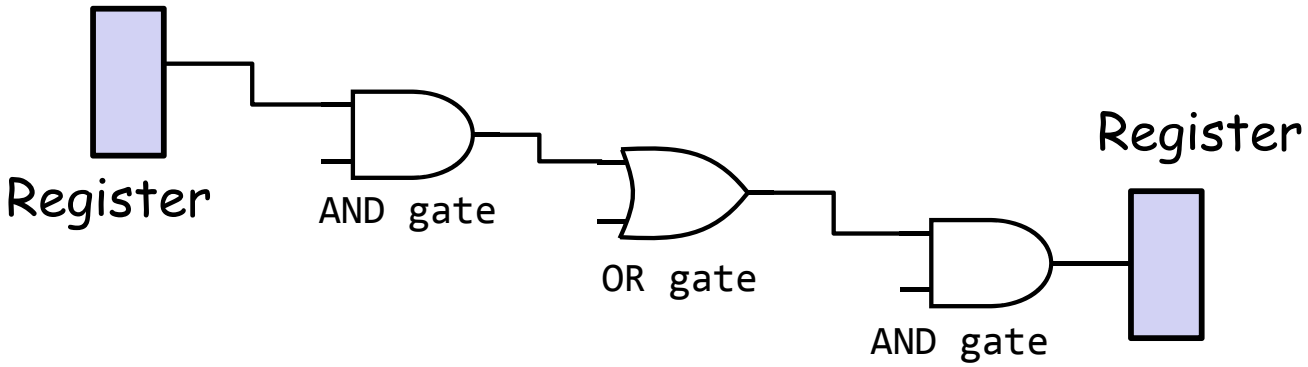
Growth in clock rate of microprocessors



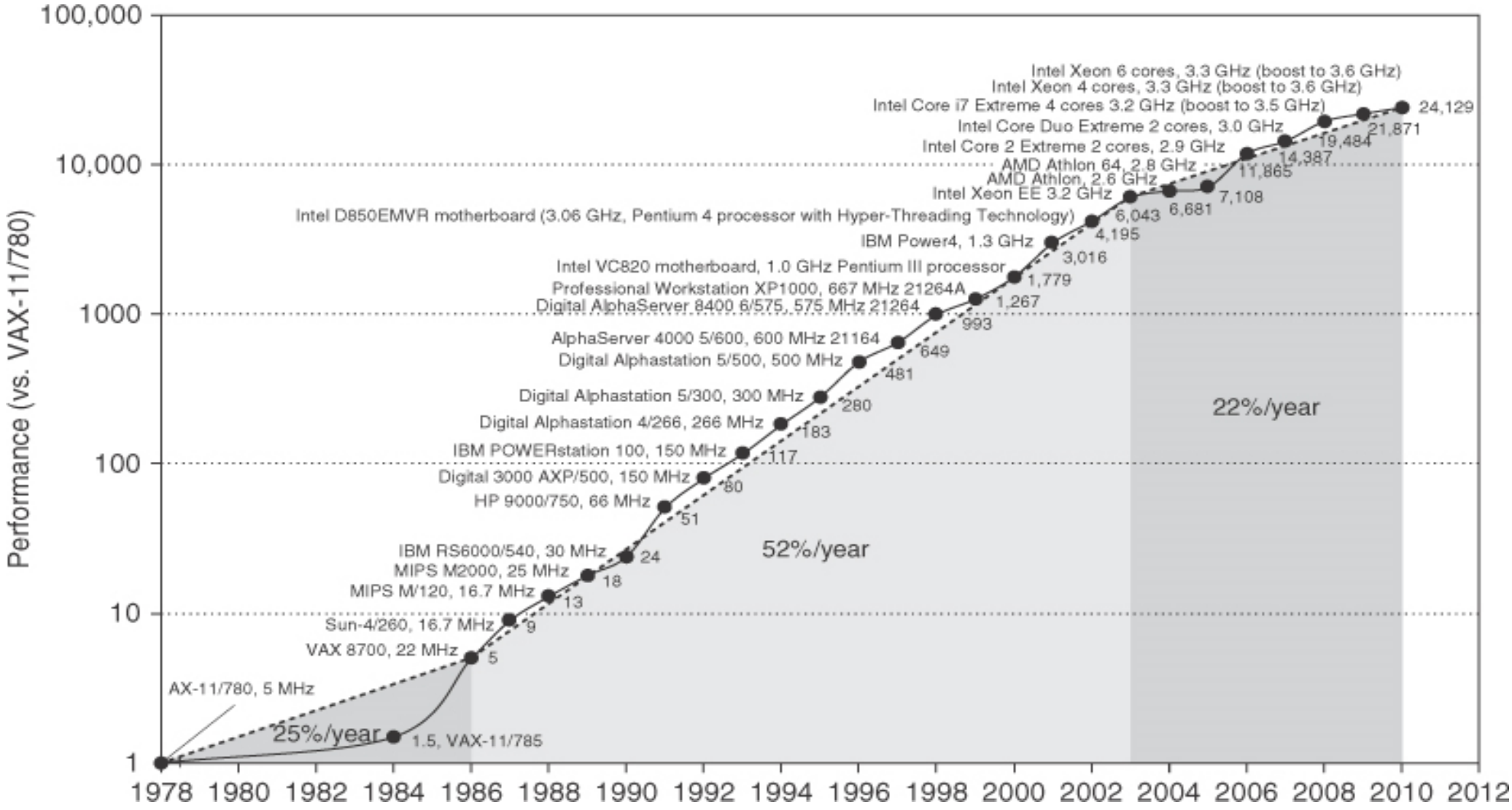
From CAQA 5th edition

Clock rate is mainly determined by

- Switching speed of gates (transistors)
- The number of levels of gates
  - The maximum number of gates cascaded in series in any combinational logics.
  - In this example, the number of levels of gates is 3.
- Wiring delay and fanout



Growth in processor performance



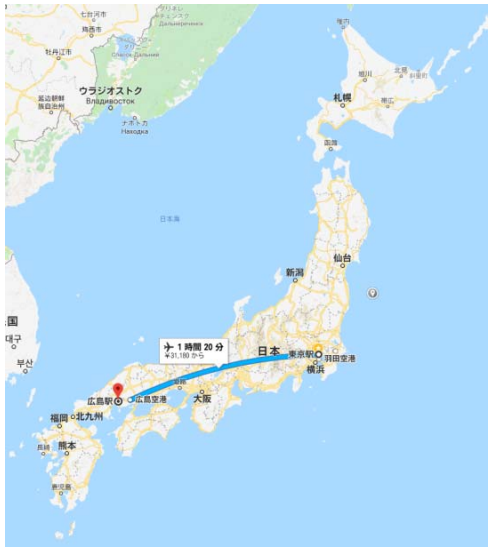
From CAQA 5th edition

Which is faster?

From Tokyo to Hiroshima

	Time & Cost	Max Speed	Passengers	Throughput (Speed x P)
Boeing 737	**1:20** 32,000yen	800km/h (503km/h)	170	85,510 (503 x 170)
Nozomi	**4:00** 18,000yen	270km/h (205km/h)	1,300	266,500 (205 x 1,300)

- Time to run the task (ExTime)
  - Execution time, response time, **latency**
- Tasks per day, hour, week, sec, ns … (Performance)
  - **Throughput**, bandwidth



Defining (Speed) Performance

- Normally interested in reducing
  - Response time (execution time) – the time between the start and the completion of a task or a program
    - Important to individual users
  - Thus, to maximize performance, need to minimize execution time

$$\text{performance}_X = 1 / \text{execution\_time}_X$$

If X is n times faster than Y, then

$$\frac{\text{performance}_X}{\text{performance}_Y} = \frac{\text{execution\_time}_Y}{\text{execution\_time}_X} = n$$

- Throughput – the total amount of work done in a given time
  - Important to data center managers
- Decreasing response time almost always improves throughput

Performance Factors

- Want to distinguish elapsed time and the time spent on our task
- CPU execution time (CPU time) : time the CPU spends working on a task
  - Does not include time waiting for I/O or running other programs

$$\text{CPU execution time for a program} = \text{\# CPU clock cycles for a program} \times \text{clock cycle time}$$

or

$$\text{CPU execution time for a program} = \frac{\text{\# CPU clock cycles for a program}}{\text{clock rate}}$$

- Can improve performance by reducing either the length of the clock cycle or the number of clock cycles required for a program

Performance Factors

$$\text{CPU execution time for a program} = \frac{\text{\# CPU clock cycles for a program}}{\text{clock rate}}$$

Performance = clock rate x 1 / # CPU clock cycles for a program

- Performance = f x IPC
  - f: frequency (clock rate)
  - IPC: retired instructions per cycle

```
int flag = 1;

int foo(){
  while(flag);
}
```

Syllabus (3/3)

Course schedule/Required learning		
	Course schedule	**Required learning**
Class 1	Design and Analysis of Computer Systems	Understand the basic of design and analysis of computer systems.
Class 2	Instruction Set Architecture	Understand the examples of instruction set architectures
Class 3	Memory Hierarchy Design	Understand the organization of memory hierarchy designs
Class 4	Pipelining	Understand the idea and organization of pipelining
Class 5	Instruction Level Parallelism: Concepts and Challenges	Understand the idea and requirements for exploiting instruction level parallelism
Class 6	Instruction Level Parallelism: Instruction Fetch and Branch Prediction	Understand the organization of instruction fetch and branch predictions to exploit instruction level parallelism
Class 7	Instruction Level Parallelism: Advanced Techniques for Branch Prediction	Understand the advanced techniques for branch prediction to exploit instruction level parallelism
Class 8	Instruction Level Parallelism: Dynamic Scheduling	Understand the dynamic scheduling to exploit instruction level parallelism
Class 9	Instruction Level Parallelism: Exploiting ILP Using Multiple Issue and Speculation	Understand the multiple issue mechanism and speculation to exploit instruction level parallelism
Class 10	Instruction Level Parallelism: Out-of-order Execution and Multithreading	Understand the out-of-order execution and multithreading to exploit instruction level parallelism
Class 11	Multi-Processor: Distributed Memory and Shared Memory Architecture	Understand the distributed memory and shared memory architecture for multi-processors
Class 12	Thread Level Parallelism: Coherence and Synchronization	Understand the coherence and synchronization for thread level parallelism
Class 13	Thread Level Parallelism: Memory Consistency Model	Understand the memory consistency model for thread level parallelism
Class 14	Thread Level Parallelism: Interconnection Network	Understand the interconnection network for thread level parallelism
Class 15	Thread Level Parallelism: Many-core Processor and Network-on-chip	Understand the many-core processor and network-on-chip for thread level parallelism

From multi-core era to many-core era

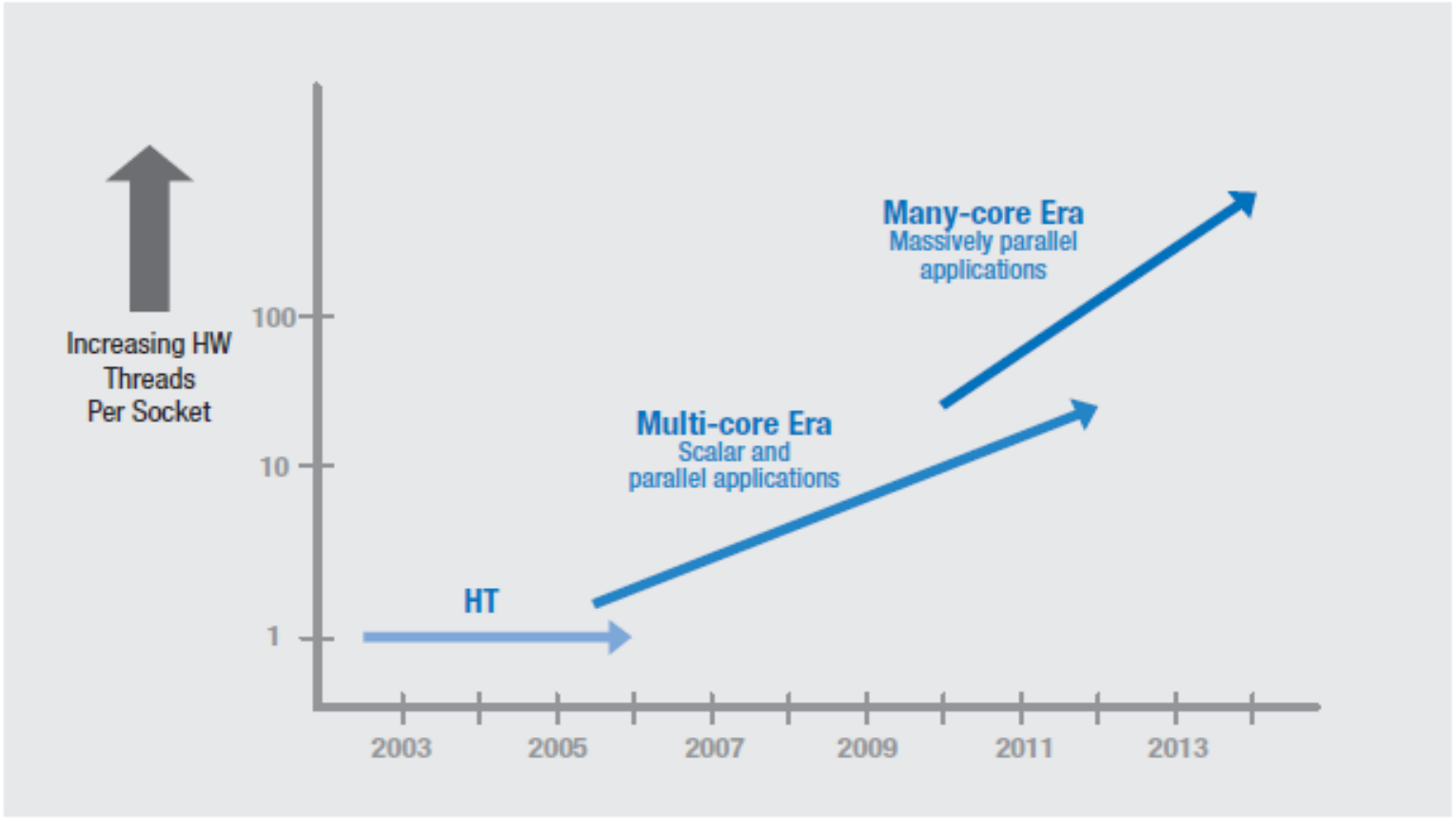


Figure 1: Current and expected eras of Intel® processor architectures

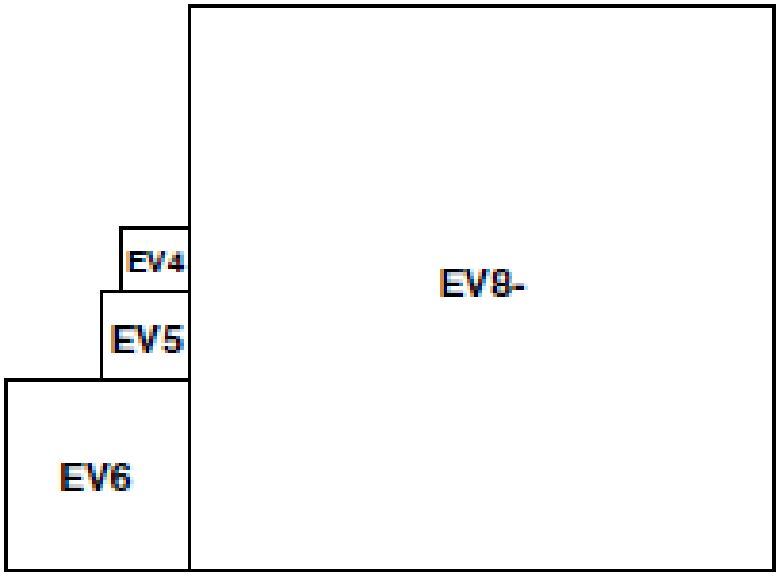
Platform 2015: Intel® Processor and Platform Evolution for the Next Decade, 2005

Pollack's Rule

- Pollack's Rule states that microprocessor "performance increase due to microarchitecture advances is roughly proportional to the square root of the increase in complexity". Complexity in this context means processor logic, i.e. its area.

WIKIPEDIA

From multi-core era to many-core era



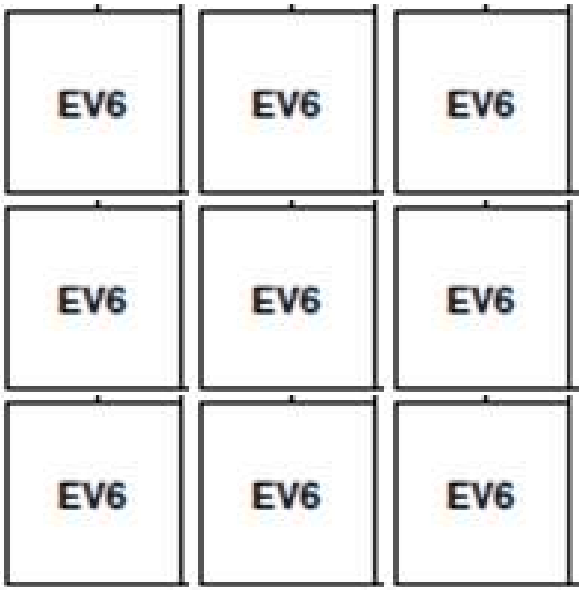
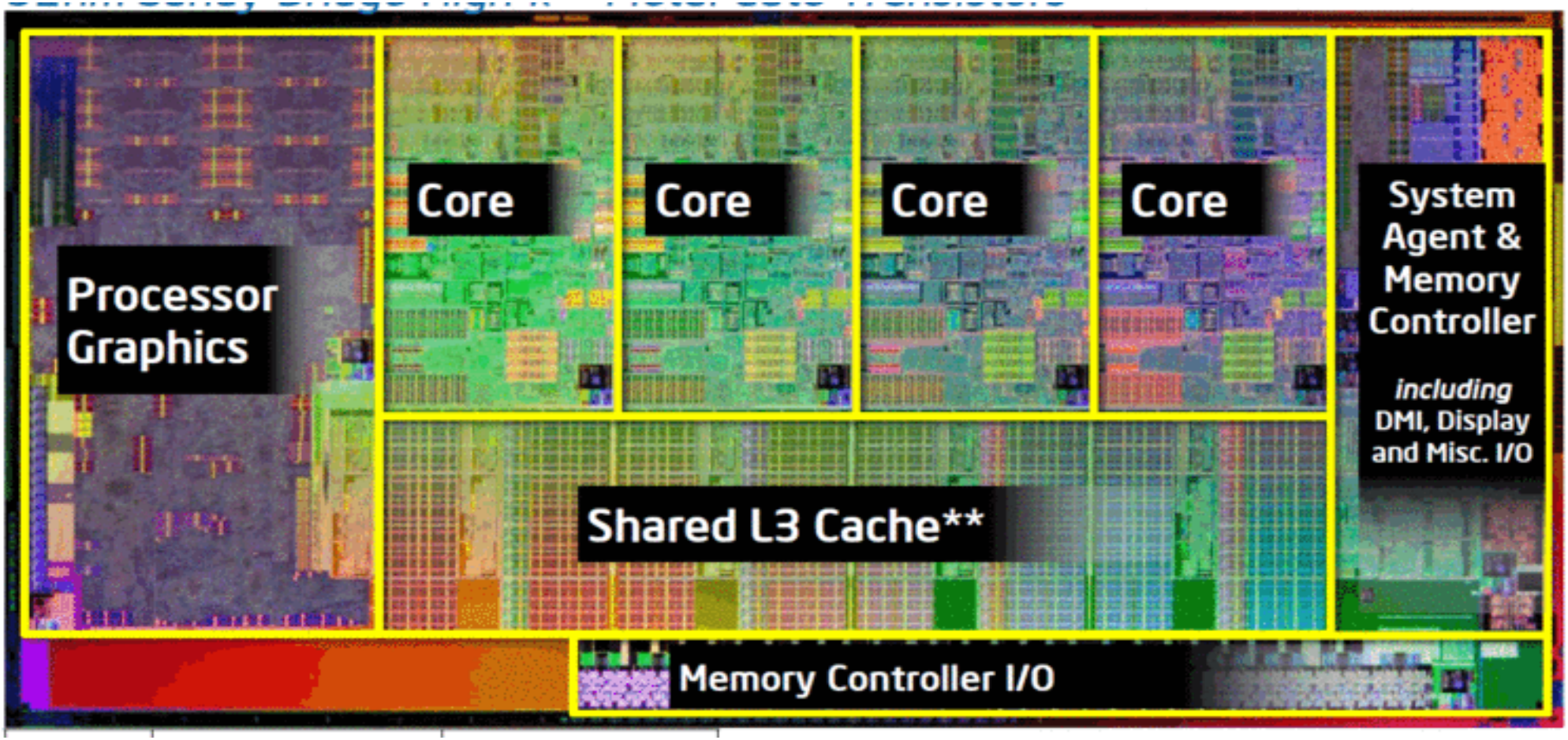


Figure 1. Relative sizes of the cores used in the study

Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction, MICRO-36

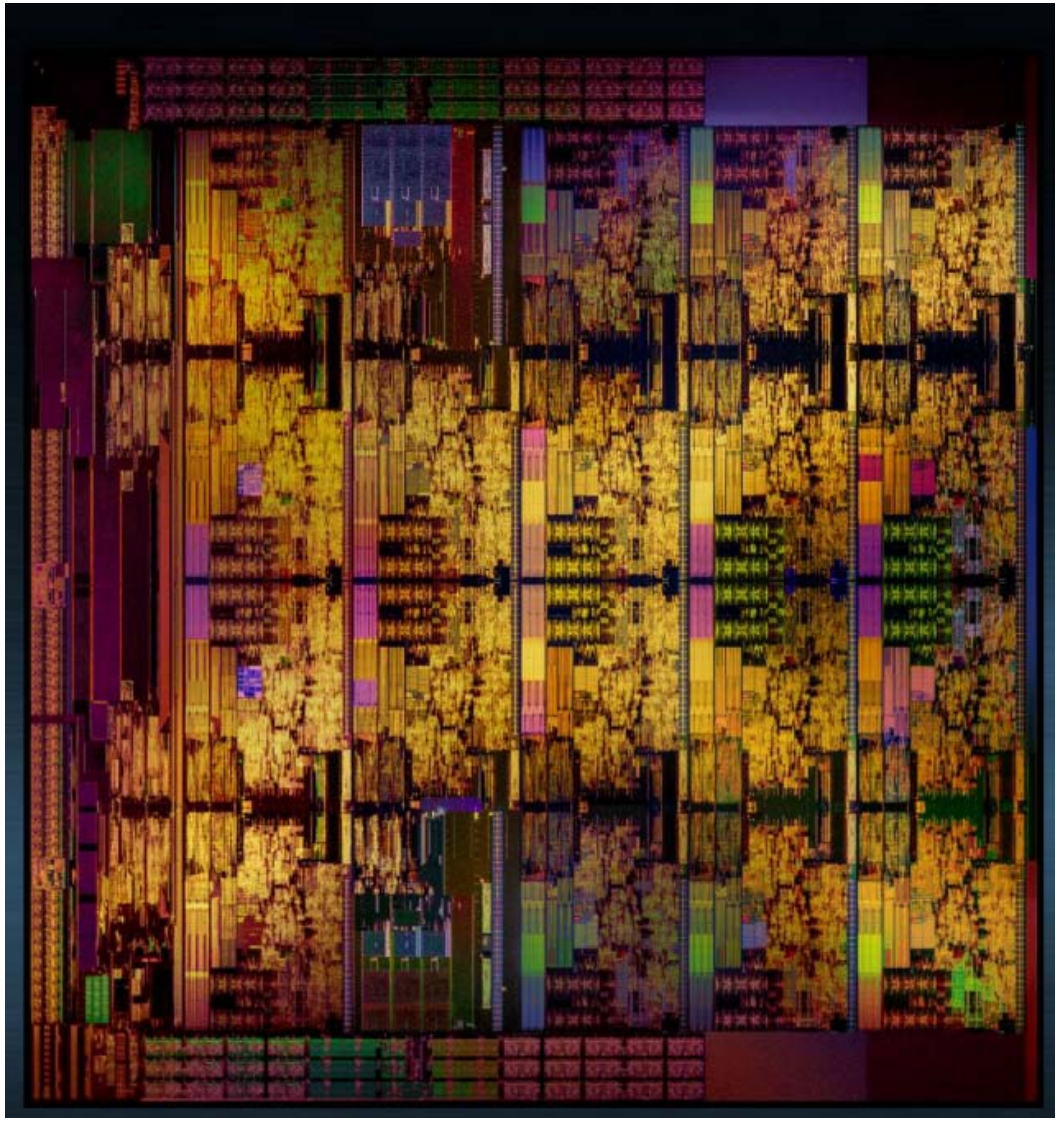
Intel Sandy Bridge, January 2011

- 4 to 8 core



Intel Skylake-X, Core i9-7980XE, 2017

- 18 core



Syllabus (3/3)

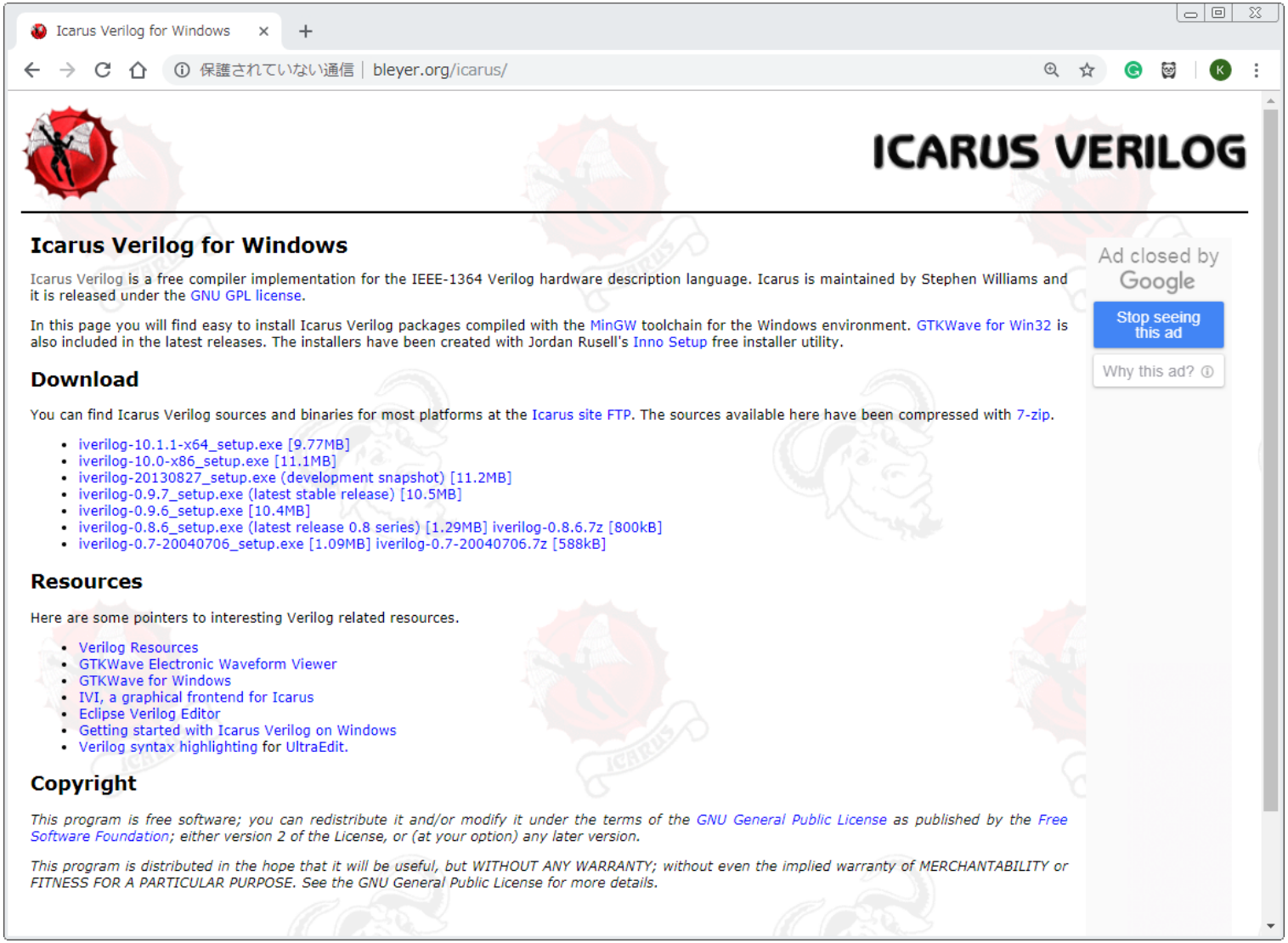
Course schedule/Required learning		
	Course schedule	**Required learning**
Class 1	Design and Analysis of Computer Systems	Understand the basic of design and analysis of computer systems.
Class 2	Instruction Set Architecture	Understand the examples of instruction set architectures
Class 3	Memory Hierarchy Design	Understand the organization of memory hierarchy designs
Class 4	Pipelining	Understand the idea and organization of pipelining
Class 5	Instruction Level Parallelism: Concepts and Challenges	Understand the idea and requirements for exploiting instruction level parallelism
Class 6	Instruction Level Parallelism: Instruction Fetch and Branch Prediction	Understand the organization of instruction fetch and branch predictions to exploit instruction level parallelism
Class 7	Instruction Level Parallelism: Advanced Techniques for Branch Prediction	Understand the advanced techniques for branch prediction to exploit instruction level parallelism
Class 8	Instruction Level Parallelism: Dynamic Scheduling	Understand the dynamic scheduling to exploit instruction level parallelism
Class 9	Instruction Level Parallelism: Exploiting ILP Using Multiple Issue and Speculation	Understand the multiple issue mechanism and speculation to exploit instruction level parallelism
Class 10	Instruction Level Parallelism: Out-of-order Execution and Multithreading	Understand the out-of-order execution and multithreading to exploit instruction level parallelism
Class 11	Multi-Processor: Distributed Memory and Shared Memory Architecture	Understand the distributed memory and shared memory architecture for multi-processors
Class 12	Thread Level Parallelism: Coherence and Synchronization	Understand the coherence and synchronization for thread level parallelism
Class 13	Thread Level Parallelism: Memory Consistency Model	Understand the memory consistency model for thread level parallelism
Class 14	Thread Level Parallelism: Interconnection Network	Understand the interconnection network for thread level parallelism
Class 15	Thread Level Parallelism: Many-core Processor and Network-on-chip	Understand the many-core processor and network-on-chip for thread level parallelism

Homework 1

1. Install Icarus Verilog
2. Install GTKWave
3. Simulate the behavior of a 4-bit counter (sample circuit 1) using Icarus Verilog, and confirm the waveforms using GTKWave.
4. Modify the circuit and HDL code for another typical circuit (like a decoder, LFSR and ALU), then simulate and confirm the waveforms.
5. Submit a report printed on A4 paper at the beginning of the next lecture.
   - The report should include a block diagram, a source code in Verilog HDL, and obtained waveforms of your circuit.

Icarus Verilog for Windows

- Installing Verilog HDL Simulation Environment（Windows）

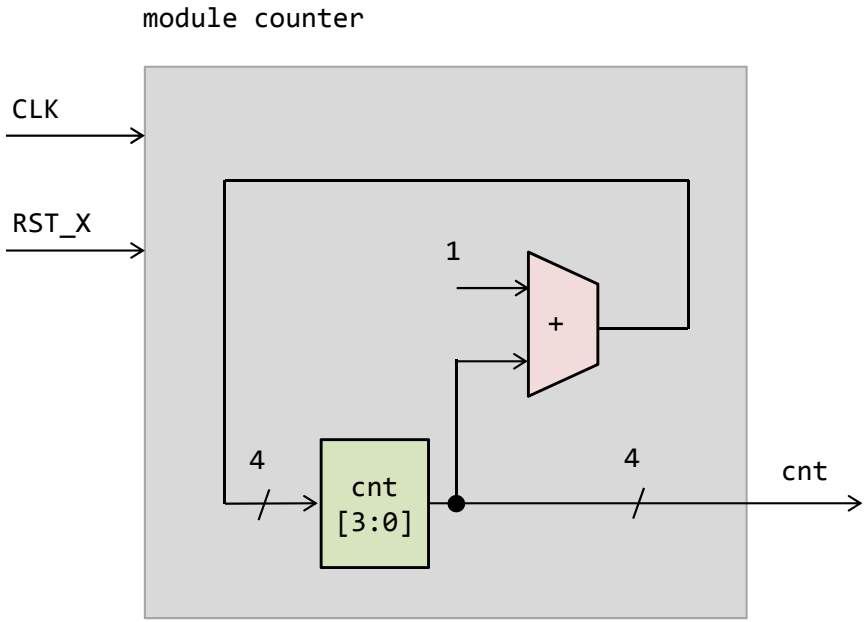


GTKWave, wave viewer

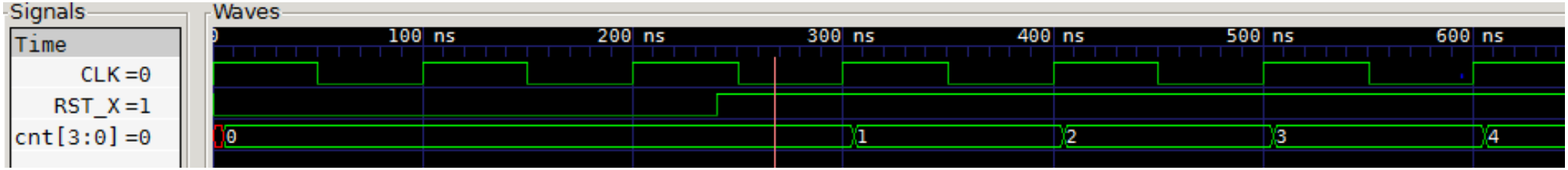
- Installing GTKWave

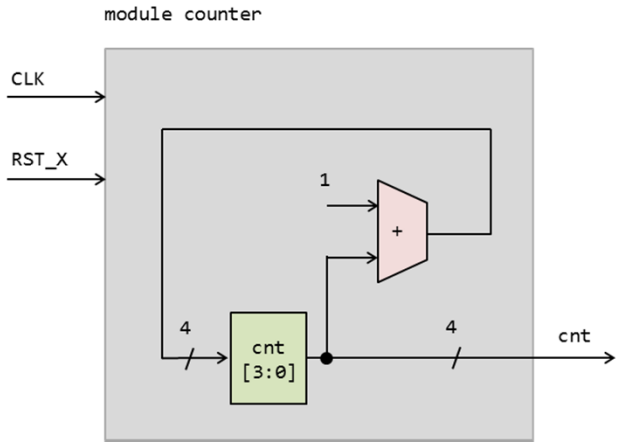
Sample circuit 1

- 4-bit counter
  - synchronous reset
  - negative-logic reset, initialize or reset the value of register cnt to zero if RST_X is low



Sample Verilog HDL Code






```
 8  module top();
 9    reg CLK, RST_X;
10    wire [3:0] w_cnt;
11
12    initial begin CLK = 1; forever #50 CLK = ~CLK; end
13    initial begin RST_X = 0; #240 RST_X = 1; end
14    initial #800 $finish();
15    initial begin
16      $dumpfile("wave.vcd");
17      $dumpvars(0, cnt1);
18    end
19    always @(posedge CLK) $write("cnt1: %d %x¥n", RST_X, w_cnt);
20
21    counter cnt1(CLK, RST_X, w_cnt);
22  endmodule
23
24  /////////////////////////////////////////////////////////////////////////////
25  module counter(CLK, RST_X, cnt);
26    input  wire CLK, RST_X;
27    output reg [3:0] cnt;
28
29    always @(posedge CLK) begin
30      if(!RST_X) cnt <= #5 0;
31      else       cnt <= #5 cnt + 1;
32    end
33  endmodule
```

counter.v