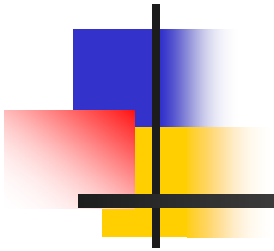
Department of Computer Science
Course number: CSC.T341



コンピュータ論理設計
Computer Logic Design

1. コンピュータシステムの基本構成
Basic Structure of Computer Systems

吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp　www.arch.cs.titech.ac.jp/lecture/CLD/
W621 講義室　　月 10:45-12:15, 木 9:00-12:15

Ver. 2018-03-31a

コンピュータ論理設計の特徴

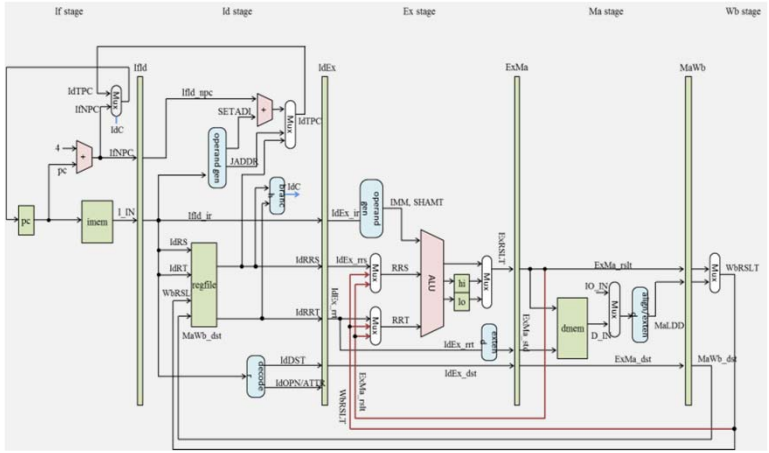
- 講義2単位，演習1単位.
- 1人1台のFPGA (Field-Programmable Gate Array) ボードを用いた演習.
- 4人程度を1グループとした共同作業と問題解決.
- 教科書で説明されるプロセッサを
  ハードウェア記述言語Verilog HDLで記述し，FPGAボードに実装する.
- グループとしてプロセッサの高速化に取り組み，コンテスト形式で成果を競う.
- 3Q開講のコンピュータアーキテクチャ（CSC.T363）のための準備.

```
module main (clk, led);
  input  wire clk;
  output wire led;

  reg [26:0] cnt;
  always @(posedge clk) cnt <= cnt + 1;

  assign led = cnt[26];
endmodule
```

Verilog HDL code



Syllabus (1/3)

講義の概要とねらい

本講義では、「論理回路理論」の講義で習得した知識をベースに、より実用的なディジタル回路について学ぶ。また、簡単なコンピュータを例題として、コンピュータの基本原理とその論理設計の方法を学習する。
演習では、学んだ組合せ回路と順序回路をVerilog HDL等のハードウェア記述言語で記述し、シミュレーションによる回路の動作検証、FPGAが搭載されたハードウェアボード等に実装して動作確認をおこなう。

到達目標

本講義を履修することによって以下を習得する。
・コンピュータシステムの基本構成
・シングルサイクルプロセッサの論理設計に関する知識
・パイプライン処理をおこなうプロセッサの論理設計に関する知識
・ハードウェア記述言語を用いたシンプルなコンピュータシステムの設計能力

キーワード

コンピュータ，命令セットアーキテクチャ，プロセッサ、パイプライン処理，ハードウェア記述言語，Verilog HDL、FPGA

学生が身につける力

国際的教養力	コミュニケーション力	専門力	課題設定力	実践力または解決力
-	-	✓	-	✓

授業の進め方

原則として、90分×2コマの講義の後、90分×1コマのFPGAボードを用いた演習をおこないます。

Syllabus (2/3)

授業計画・課題		
	授業計画	課題
第1回	コンピュータシステムの基本構成	コンピュータシステムの基本構成について理解する。
第2回	論理設計演習(1)	論理設計演習(1)
第3回	ハードウェア記述言語：組合せ回路	組合せ回路の記述を理解する。
第4回	ハードウェア記述言語：順序回路	順序回路の記述を理解する。
第5回	論理設計演習(2)	論理設計演習(2)
第6回	ハードウェア記述言語：よく使われる回路	よく使われる回路の記述を理解する。
第7回	リコンフィギャラブルシステム	Reconfigurable Systems
第8回	論理設計演習(3)	Logic Design Exercise(3)
第9回	命令セットアーキテクチャ：データ表現とアドレス指定形式	ISAにおけるデータ表現とアドレス指定形式について理解する。
第10回	命令セットアーキテクチャ：算術論理演算命令	ISAにおける算術論理演算命令について理解する。
第11回	論理設計演習(4)	論理設計演習(4)
第12回	命令セットアーキテクチャ：ロードストア命令と分岐命令	ISAにおけるロードストア命令と分岐命令について理解する。
第13回	プロセッサの基本構成要素：算術論理演算ユニット	算術論理演算ユニットについて理解する。
第14回	論理設計演習(5)	論理設計演習(5)
第15回	プロセッサの基本構成要素：レジスタファイルとメモリ	レジスタファイルとメモリについて理解する。
第16回	シングルサイクルプロセッサのデータパス	シングルサイクルプロセッサのデータパスについて理解する。
第17回	論理設計演習(6)	論理設計演習(6)
第18回	シングルサイクルプロセッサの制御	シングルサイクルプロセッサの制御について理解する。
第19回	パイプライン処理	パイプライン処理について理解する。
第20回	論理設計演習(7)	論理設計演習(7)
第21回	パイプラインハザードとデータフォワーディング	パイプラインハザードとデータフォワーディングについて理解する。
第22回	論理設計演習(8)	論理設計演習(8)

Syllabus (3/3)

教科書

デイビッド・A. パターソン、ジョン・L. ヘネシー (著)、成田光彰 (翻訳)『コンピュータの構成と設計 第5版 上/下』日経BP社

参考書、講義資料等

無し。

成績評価の基準及び方法

講義で扱うコンピュータ論理設計に関する理解、ハードウェア記述言語を用いたコンピュータシステム実装への応用力を評価する。演習（30%）と期末試験（70%）により評価する。

関連する科目

CSC.T252 ： 論理回路理論
CSC.T262 ： アセンブリ言語
CSC.T372 ： コンパイラ構成
CSC.T363 ： コンピュータアーキテクチャ
CSC.T433 ： 先端コンピュータアーキテクチャ

履修の条件(知識・技能・履修済科目等)

履修条件は特に設けないが、関連する科目の論理回路理論を履修していることが望ましい。

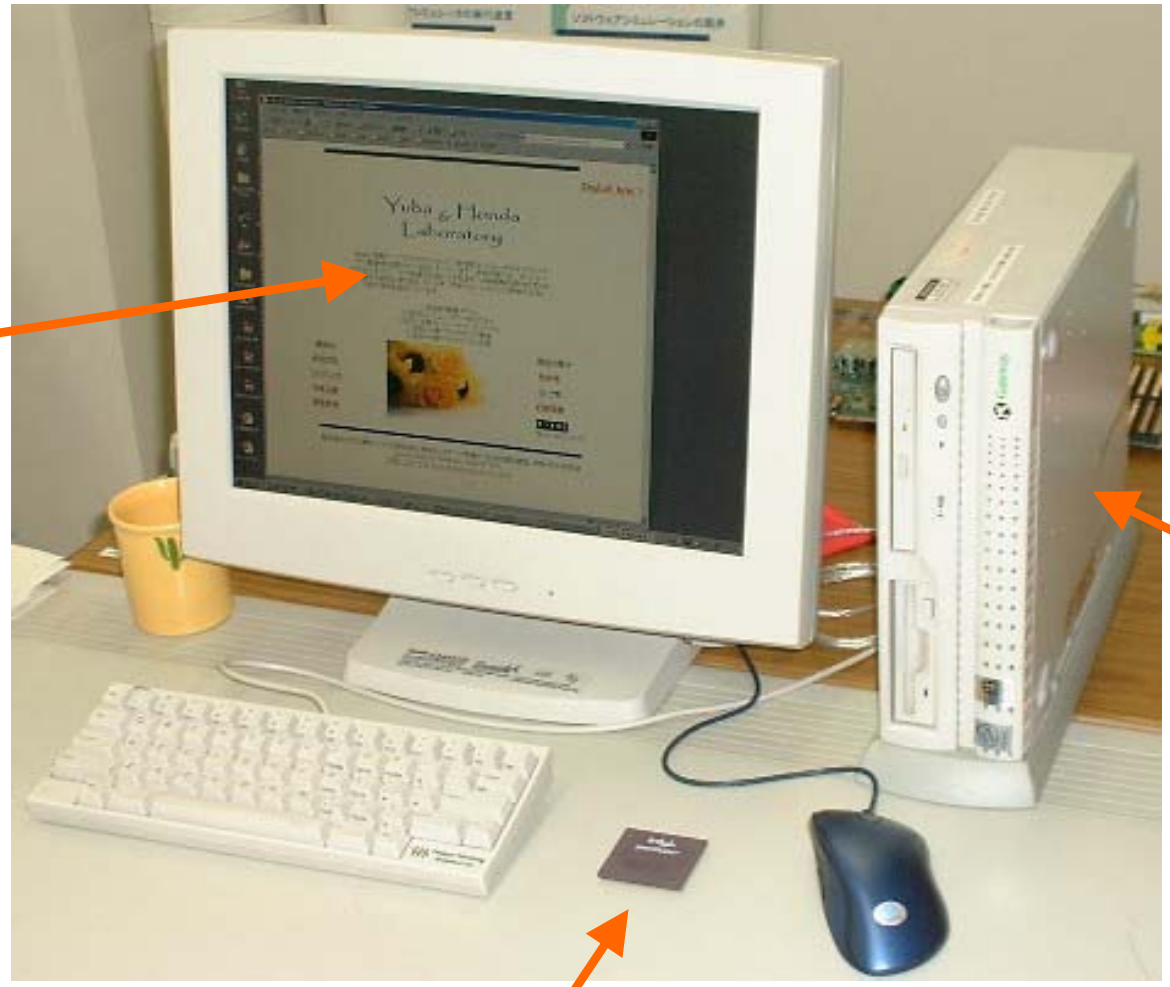
連絡先（メール、電話番号）　※"[at]"を"@"(半角)に変換してください。

吉瀬謙二: kise[at]c.titech.ac.jp

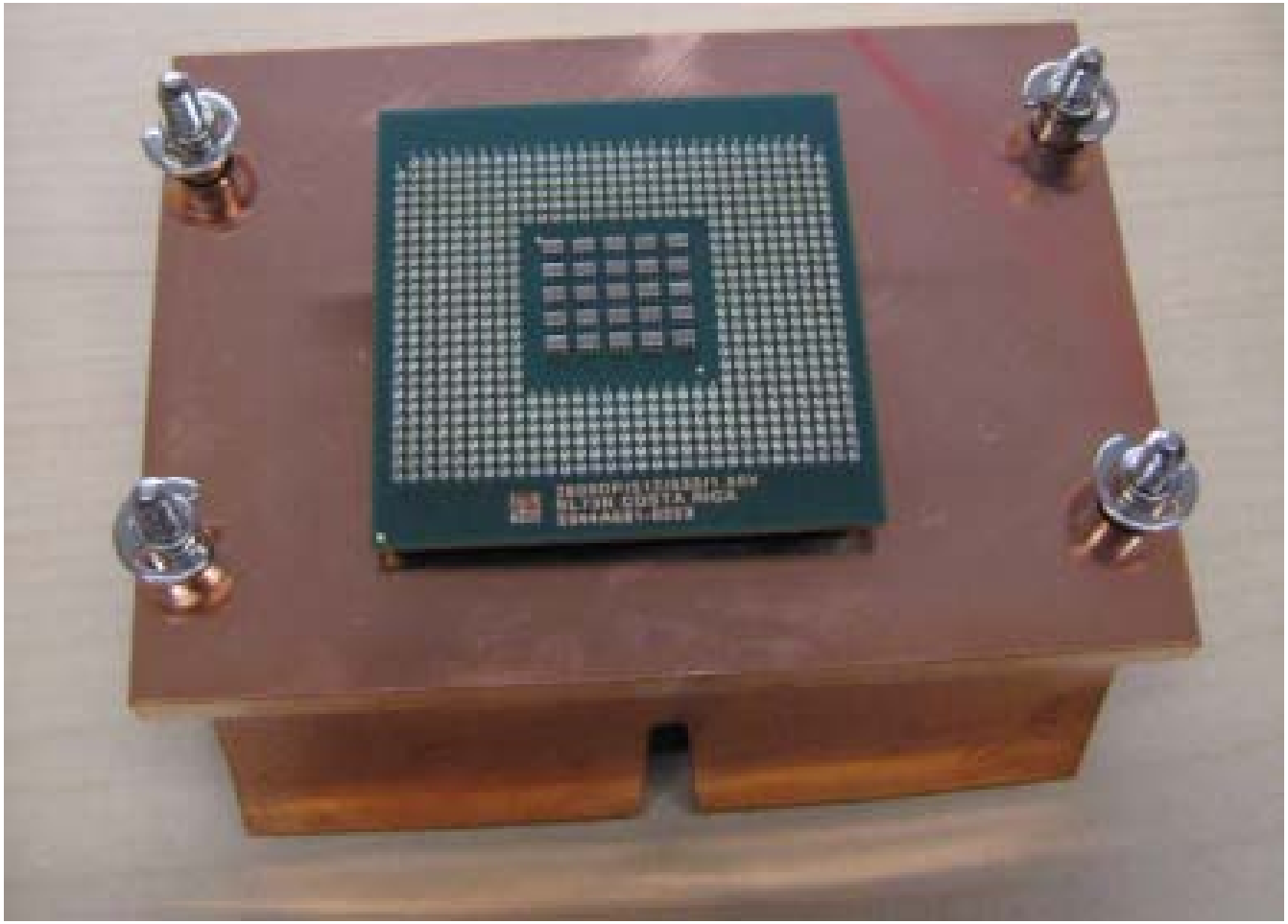
オフィスアワー

メールで事前予約すること。

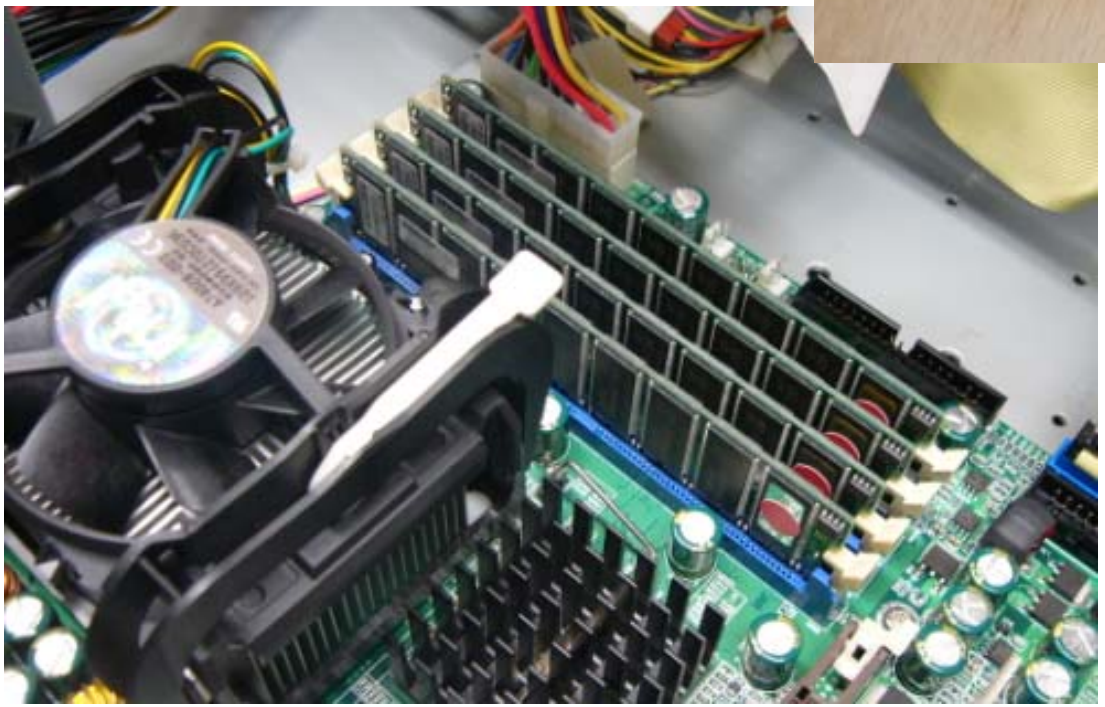
コンピュータ（デスクトップコンピュータ）



マイクロプロセッサ，CPU



メモリ，記憶
DRAM (dynamic random access memory)



ディスク（磁気ディスク，SSD）

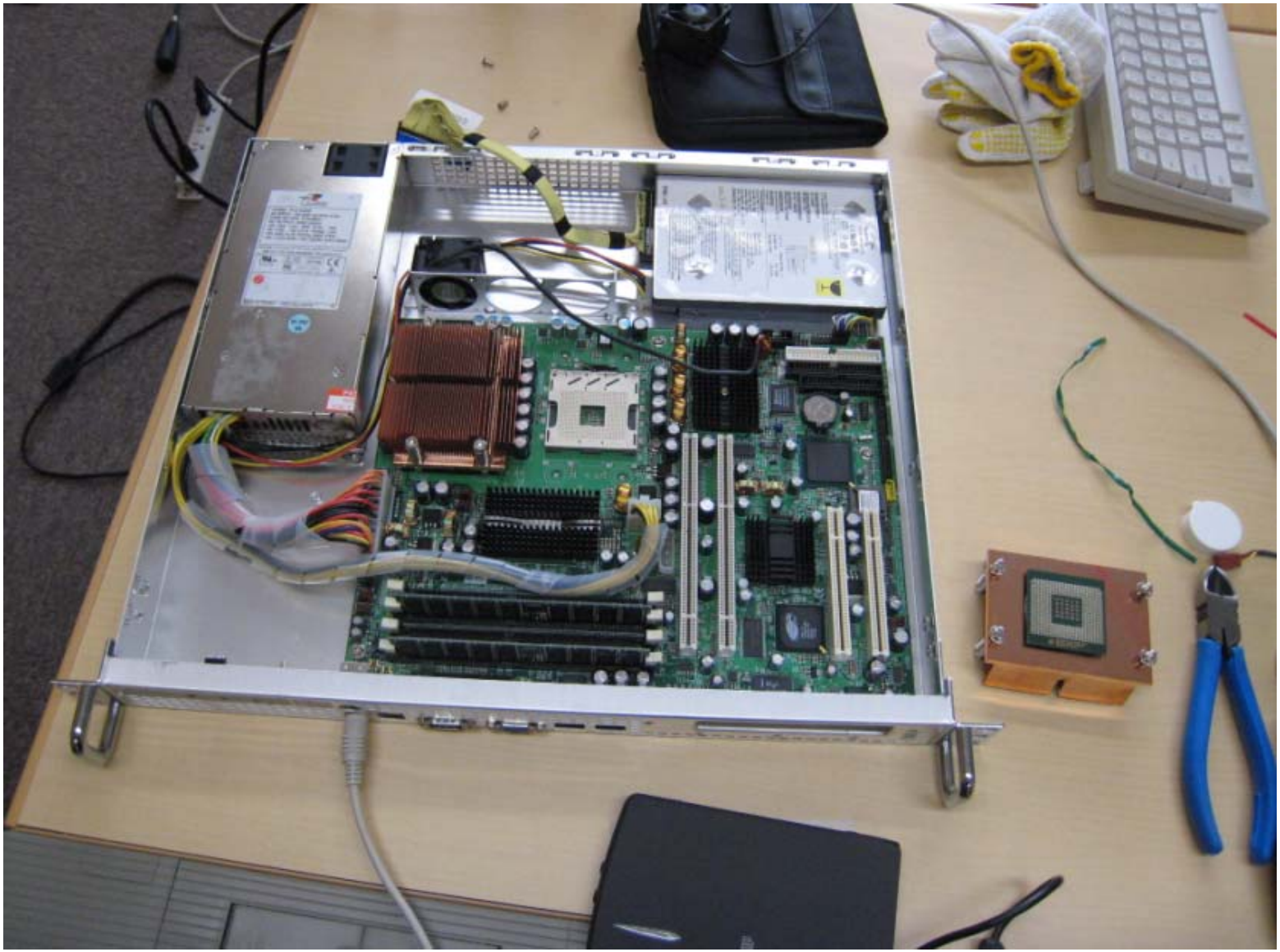
グラフィックカード, GPU

ネットワークカード

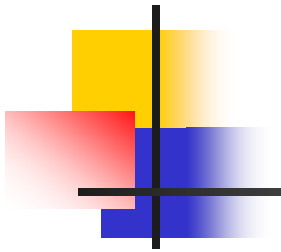
マザーボード

電源

コンピュータ

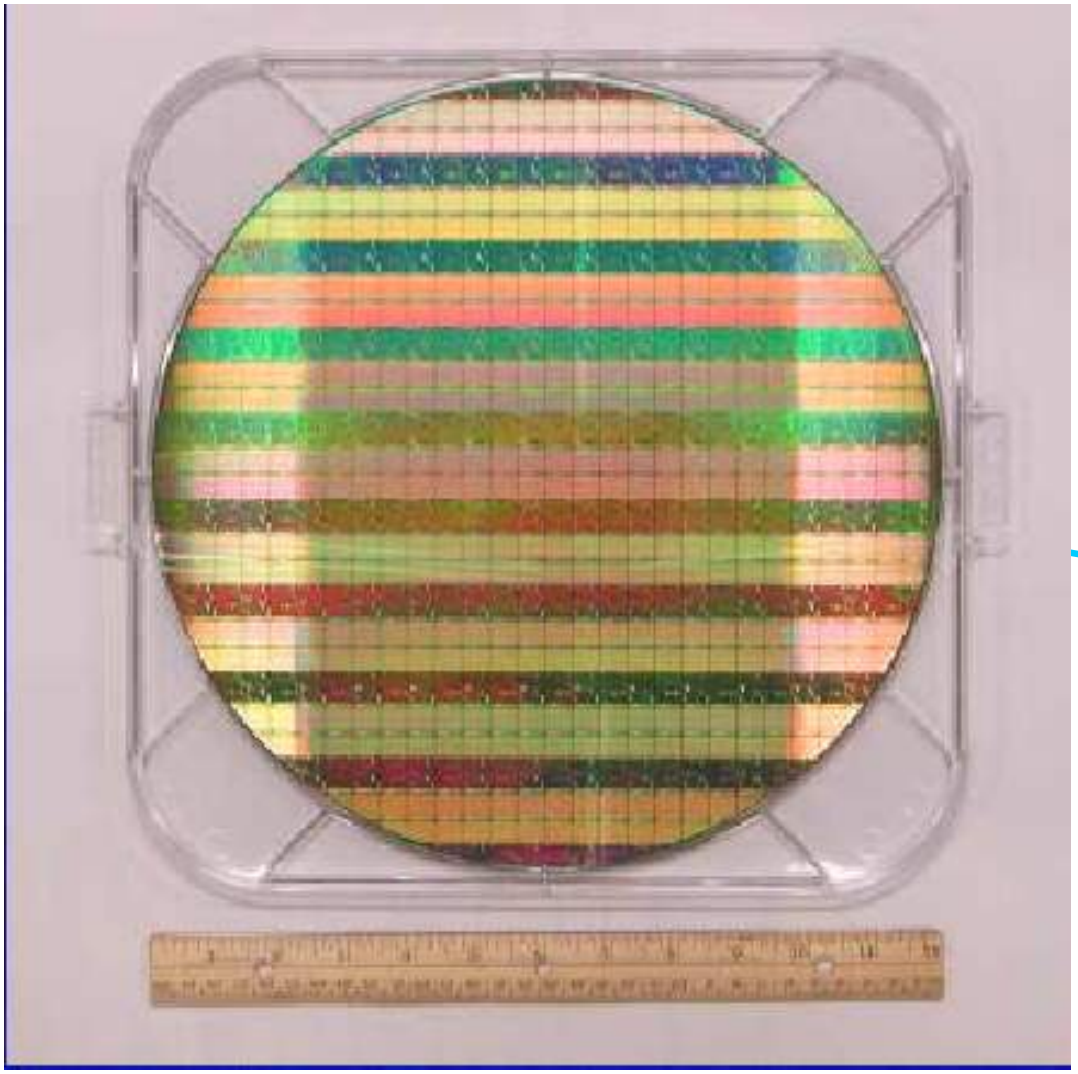


補足：クラスタ型（並列）コンピュータ

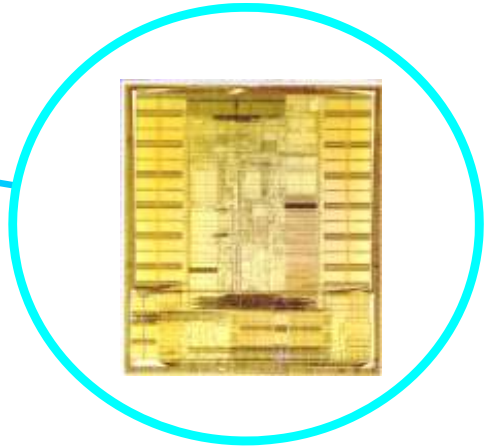


VLSIチップの製造：インゴット，ウエーハ

VLSIチップの製造：ウエーハとダイ



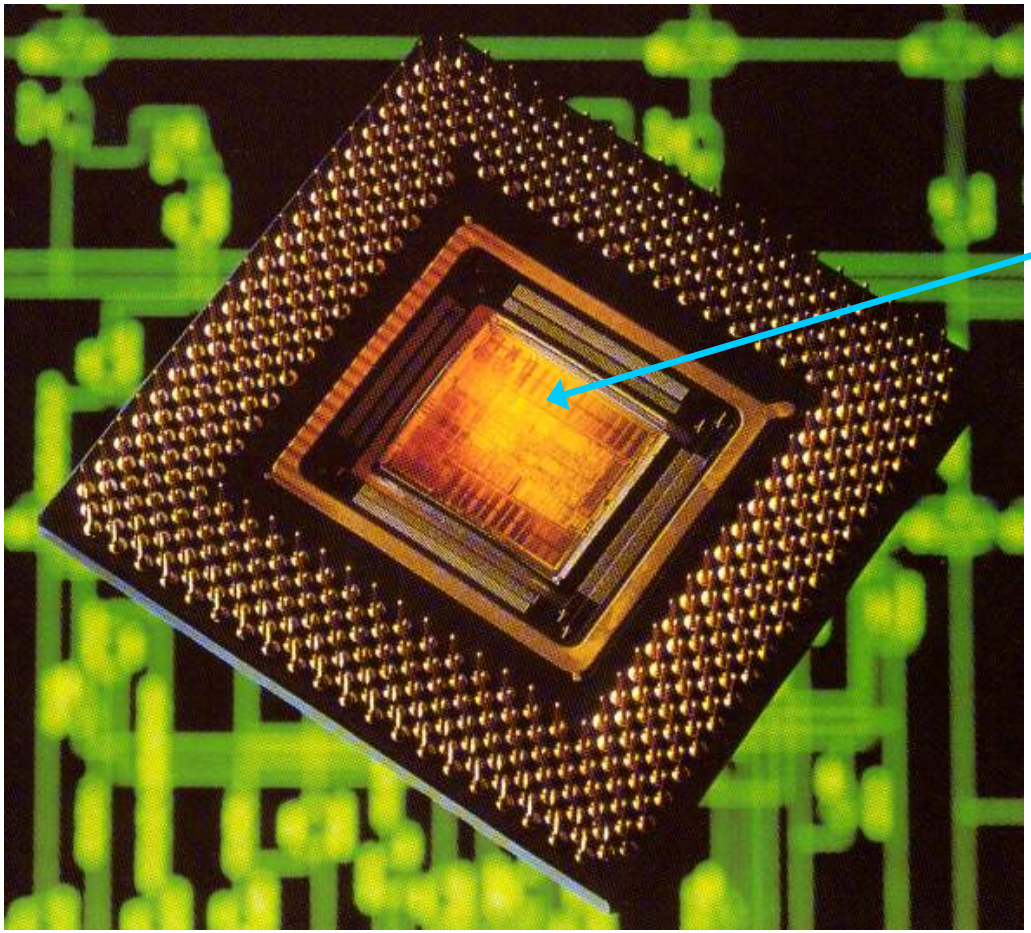
30cmのウエーハ(wafer)
厚さは数ミリで、直径が30cm
大きなCDのような形をしている。



ダイ (die)
（ウエーハから切り出した
個々のチップ）

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

プロセッサの実装：ダイのパッケージ化

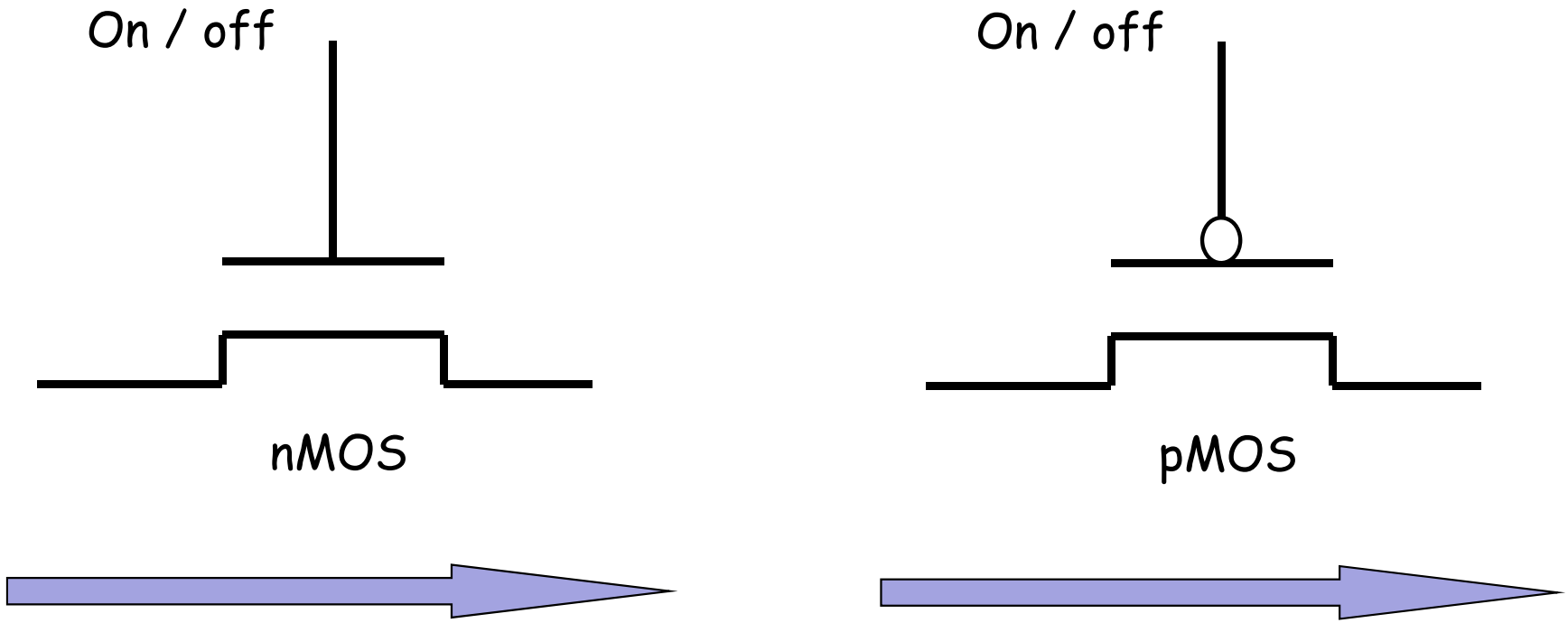


die (dice)：さいころ

Richard L. Sites, Alpha AXP Architecture Reference Manual SECOND EDITION

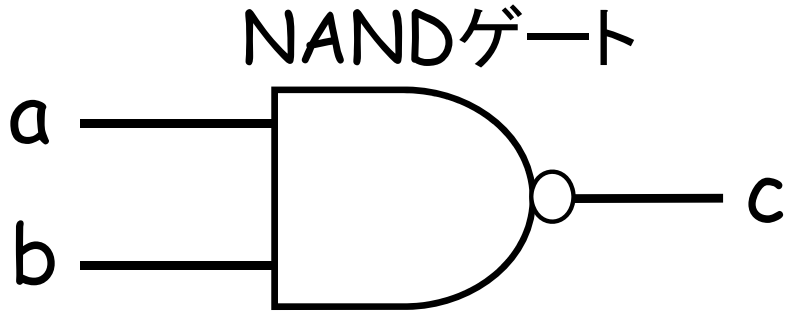
Transistor

- トランジスタは電気的なオン／オフ動作をするスイッチと捉えることができる.

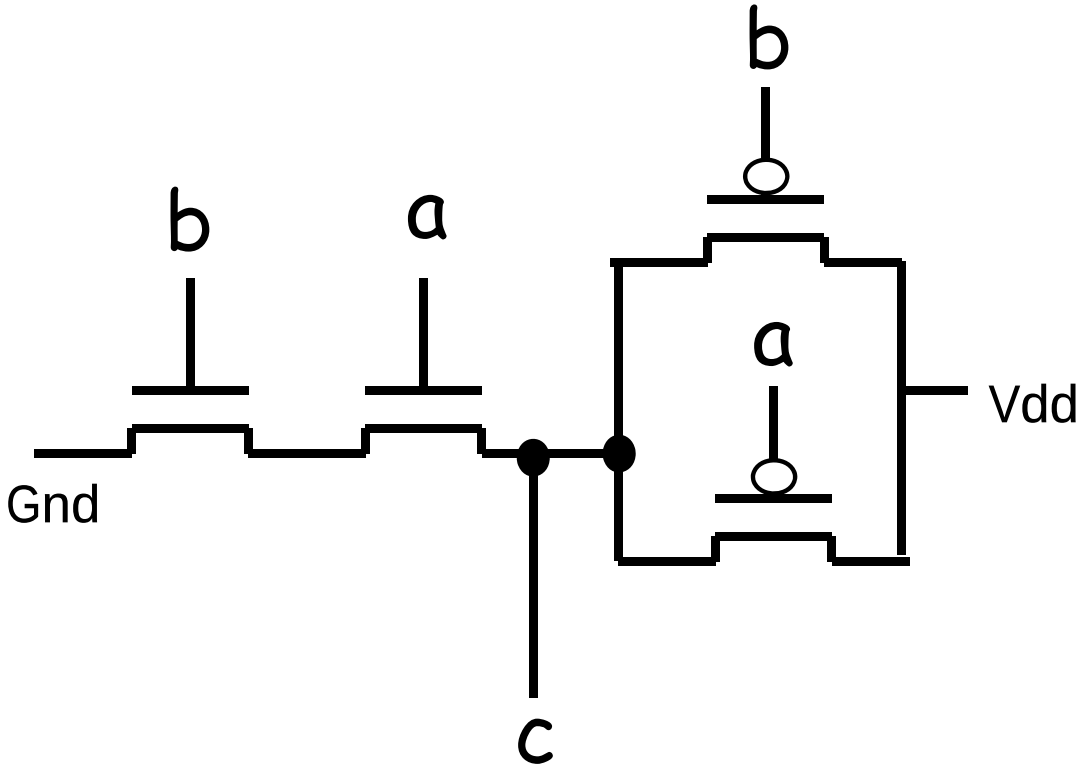


Transistor and Gate

- トランジスタは電気的なオン／オフ動作をするスイッチ
- 幾つかのトランジスタで，少し機能の高いゲートを構成

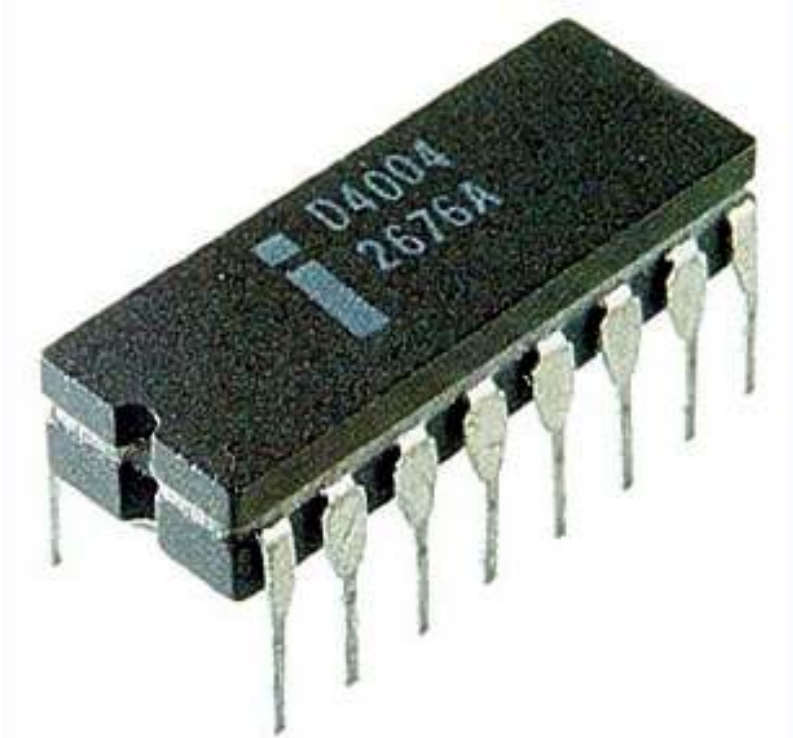


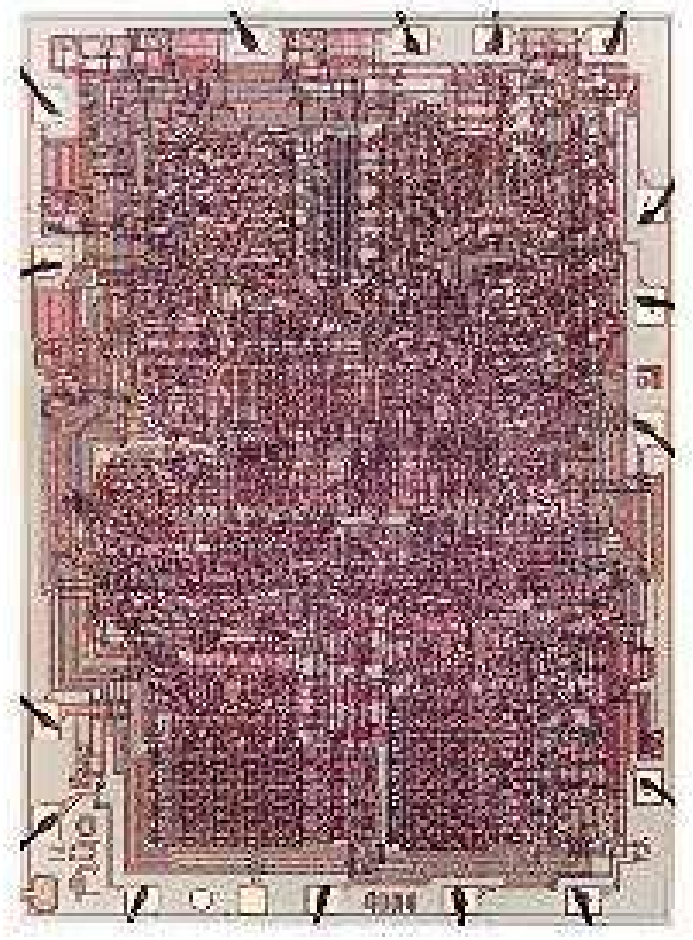
a	b	c
0	0	1
1	0	1
0	1	1
1	1	0



The first commercially available microprocessor

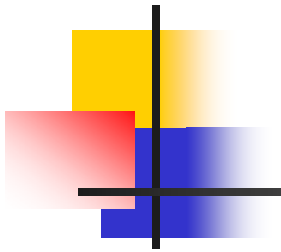
1971年: 4004 マイクロプロセッサ





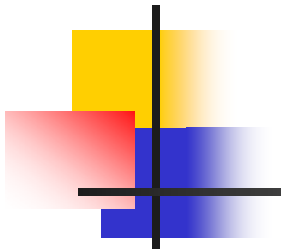
プロセッサ	出荷年	トランジスタ数
4004	1971	2,250

From Wikipedia, Intelミュージアム

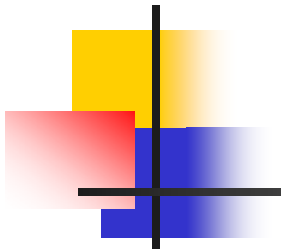


Moore’s law

- Moore's law is the observation that the number of transistors in a dense integrated circuit doubles approximately every two years.



Moore’s Law



Moore’s Law

Moore's Law

The experts look ahead

Cramming more components onto integrated circuits

With unit cost falling as the number of components per circuit rises, by 1975 economics may dictate squeezing as many as 65,000 components on a single silicon chip

By Gordon E. Moore

Director, Research and Development Laboratories, Fairchild Semiconductor division of Fairchild Camera and Instrument Corp.

The future of integrated electronics is the future of electronics itself. The advantages of integration will bring about a proliferation of electronics, pushing this science into many new areas.

Integrated circuits will lead to such wonders as home computers—or at least terminals connected to a central computer—automatic controls for automobiles, and personal portable communications equipment. The electronic wristwatch needs only a display to be feasible today.

But the biggest potential lies in the production of large systems. In telephone communications, integrated circuits in digital filters will separate channels on multiplex equipment. Integrated circuits will also switch telephone circuits and perform data processing.

Computers will be more powerful, and will be organized in completely different ways. For example, memories built of integrated electronics may be distributed throughout the machine instead of being concentrated in a central unit. In addition, the improved reliability made possible by integrated circuits will allow the construction of larger processing units. Machines similar to those in existence today will be built at lower costs and with faster turn-around.

The author

Dr. Gordon E. Moore is one of the new breed of electronic engineers, schooled in the physical sciences rather than in electronics. He earned a B.S. degree in chemistry from the University of California and a Ph.D. degree in physical chemistry from the California Institute of Technology. He was one of the founders of Fairchild Semiconductor and has been director of the research and development laboratories since 1959.

Present and future

By integrated electronics, I mean all the various technologies which are referred to as microelectronics today as well as any additional ones that result in electronics functions supplied to the user as irreducible units. These technologies were first investigated in the late 1950's. The object was to miniaturize electronics equipment to include increasingly complex electronic functions in limited space with minimum weight. Several approaches evolved, including microassembly techniques for individual components, thin-film structures and semiconductor integrated circuits.

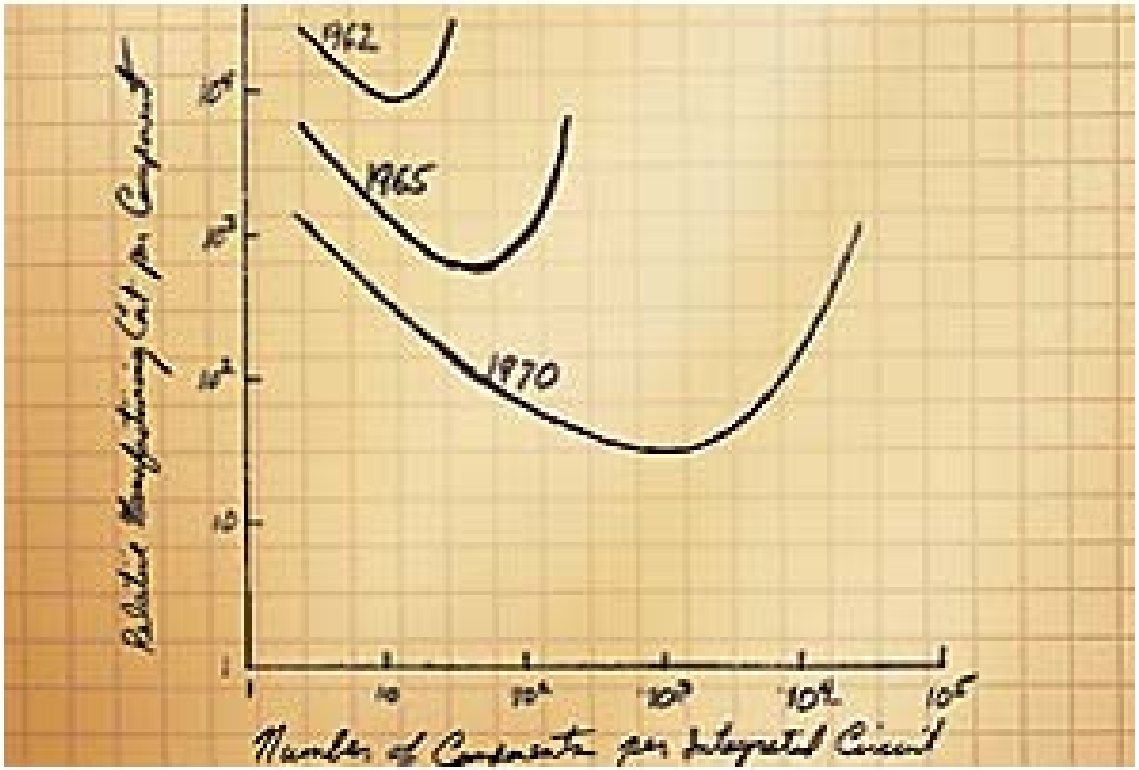
Each approach evolved rapidly and converged so that each borrowed techniques from another. Many researchers believe the way of the future to be a combination of the various approaches.

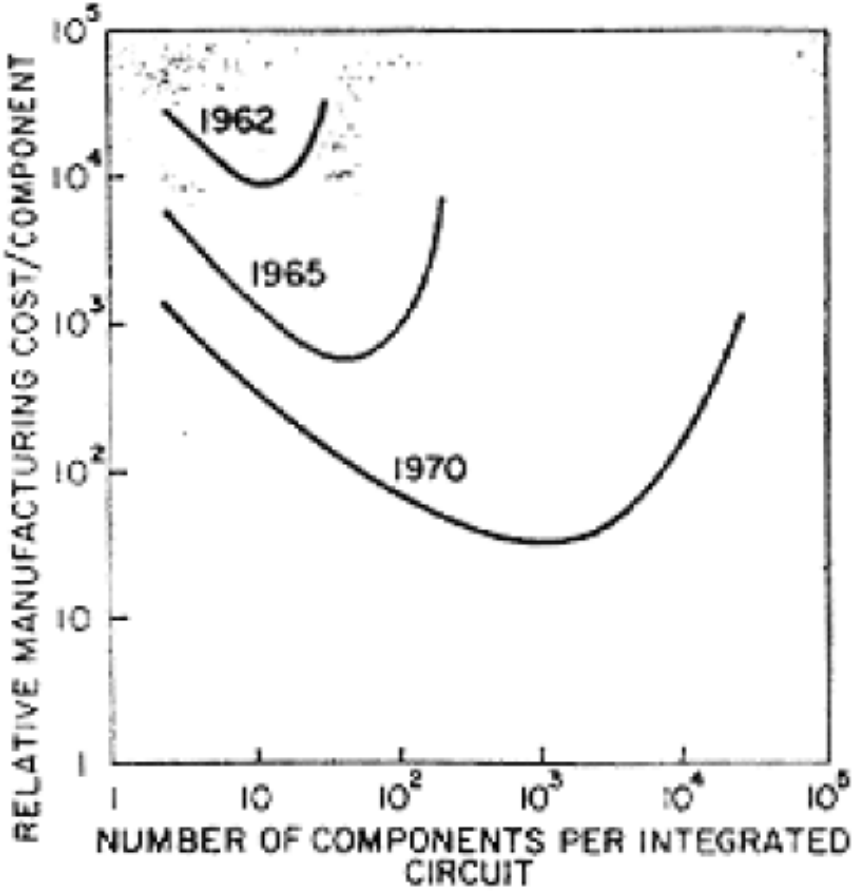
The advocates of semiconductor integrated circuitry are already using the improved characteristics of thin-film resistors by applying such films directly to an active semiconductor substrate. Those advocating a technology based upon films are developing sophisticated techniques for the attachment of active semiconductor devices to the passive film arrays.

Both approaches have worked well and are being used in equipment today.

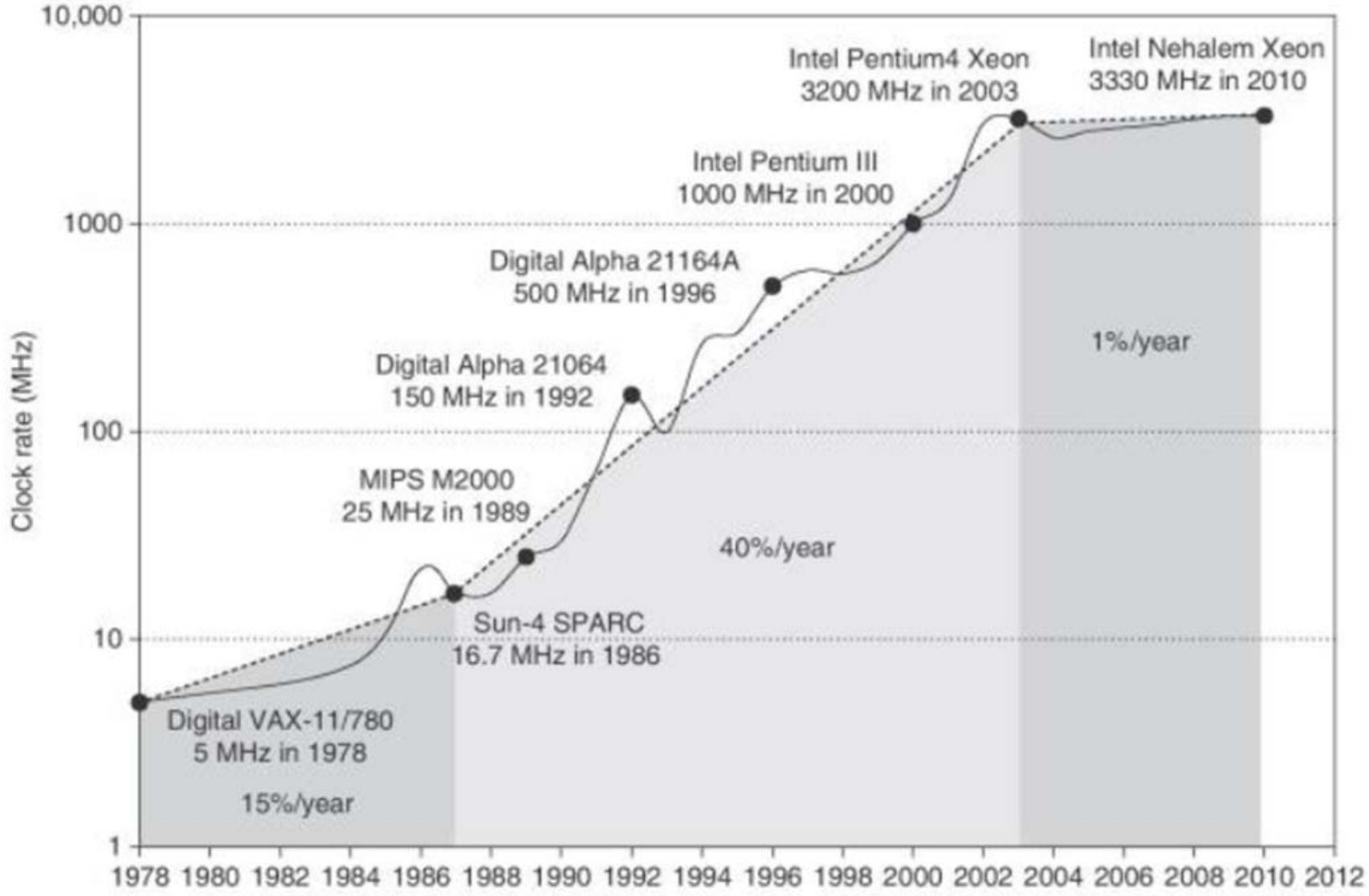
Electronics, Volume 38, Number 8, April 19, 1965

Moore's Law


1962
1965
1970


RELATIVE MANUFACTURING COST/COMPONENT
1962
1965
1970
NUMBER OF COMPONENTS PER INTEGRATED CIRCUIT

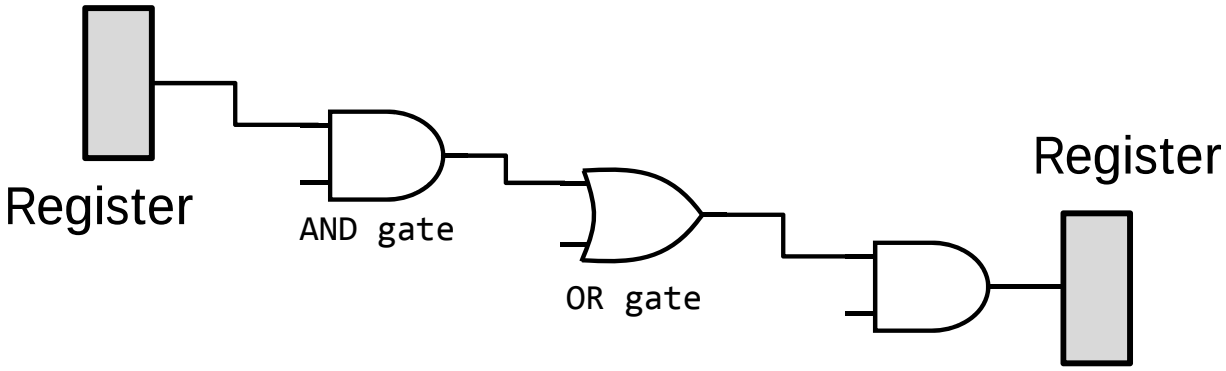
Growth in clock rate of microprocessors



From CAQA 5th edition

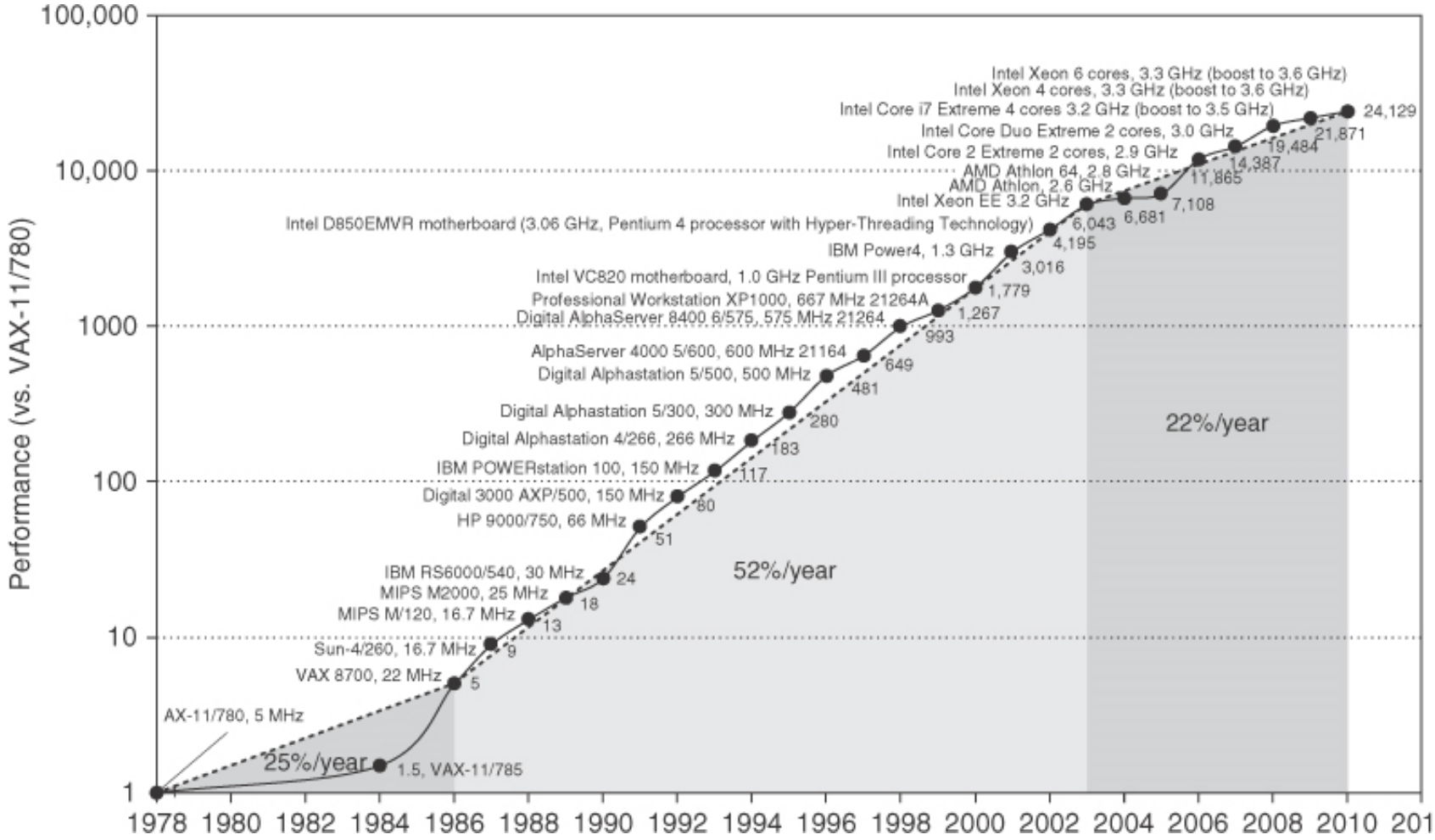
Clock rate is mainly determined by

- 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

- Performance = f x IPC
  - f: frequency (clock rate)
  - IPC: retired machine Instructions Per Cycle



From CAQA 5th edition

マルチコア（2個～数10個）からメニーコアへ

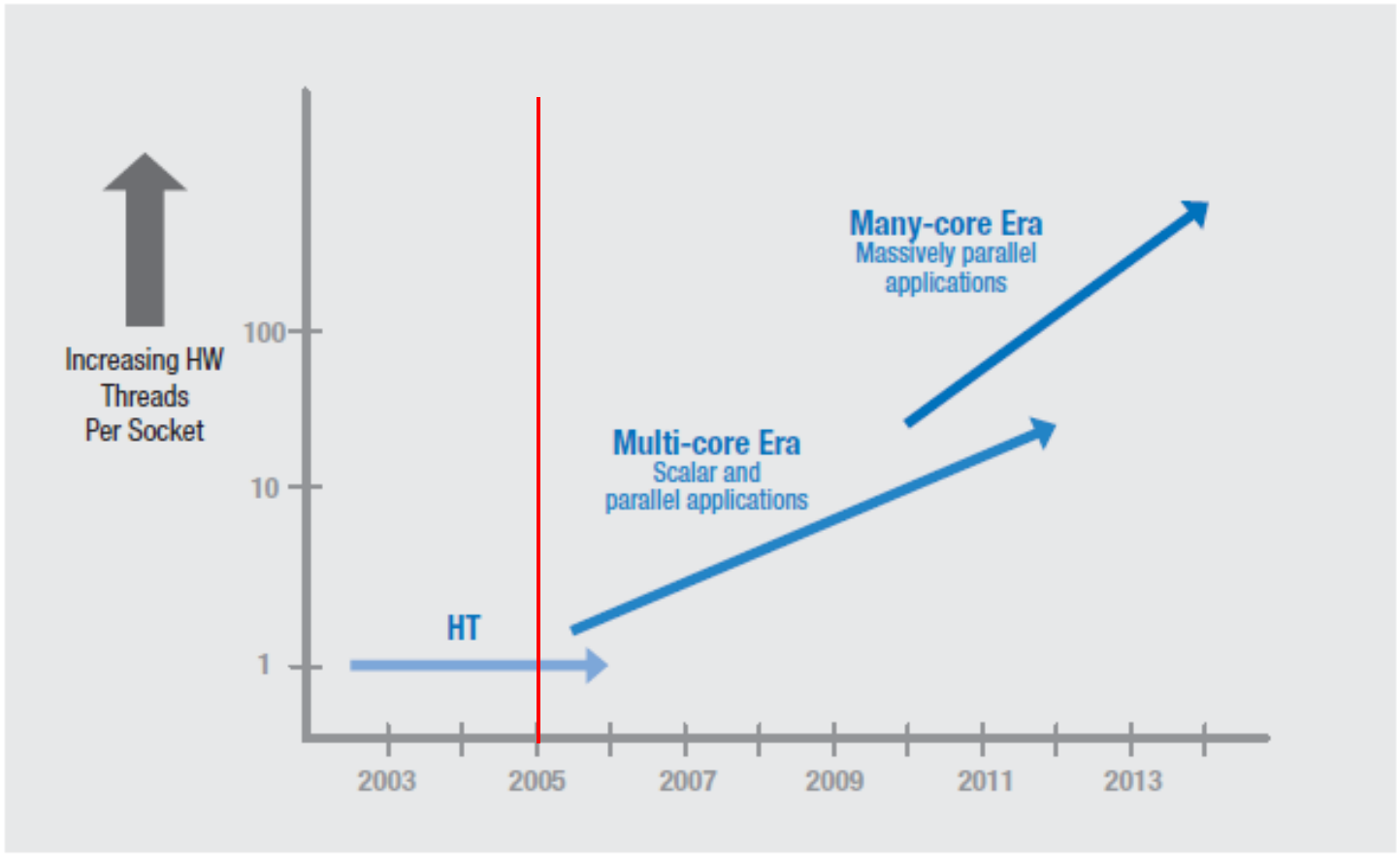
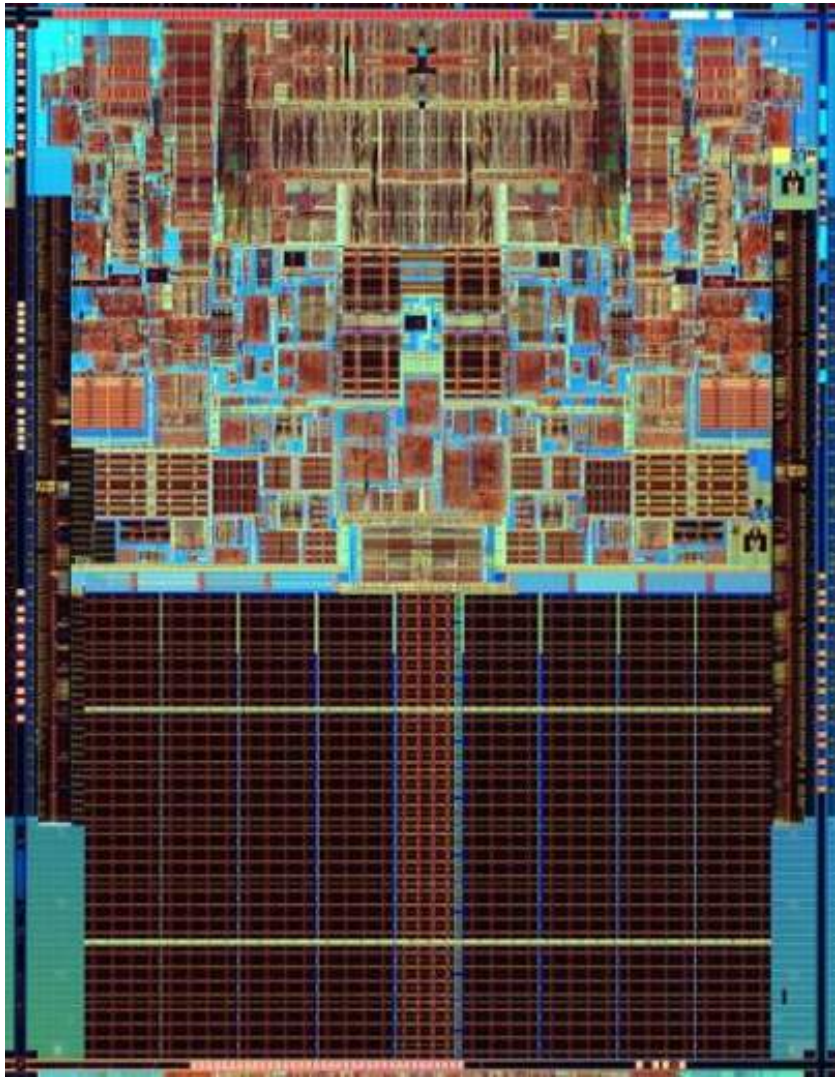


Figure 1: Current and expected eras of Intel® processor architectures

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

マイクロプロセッサ Intel Core 2 Duo

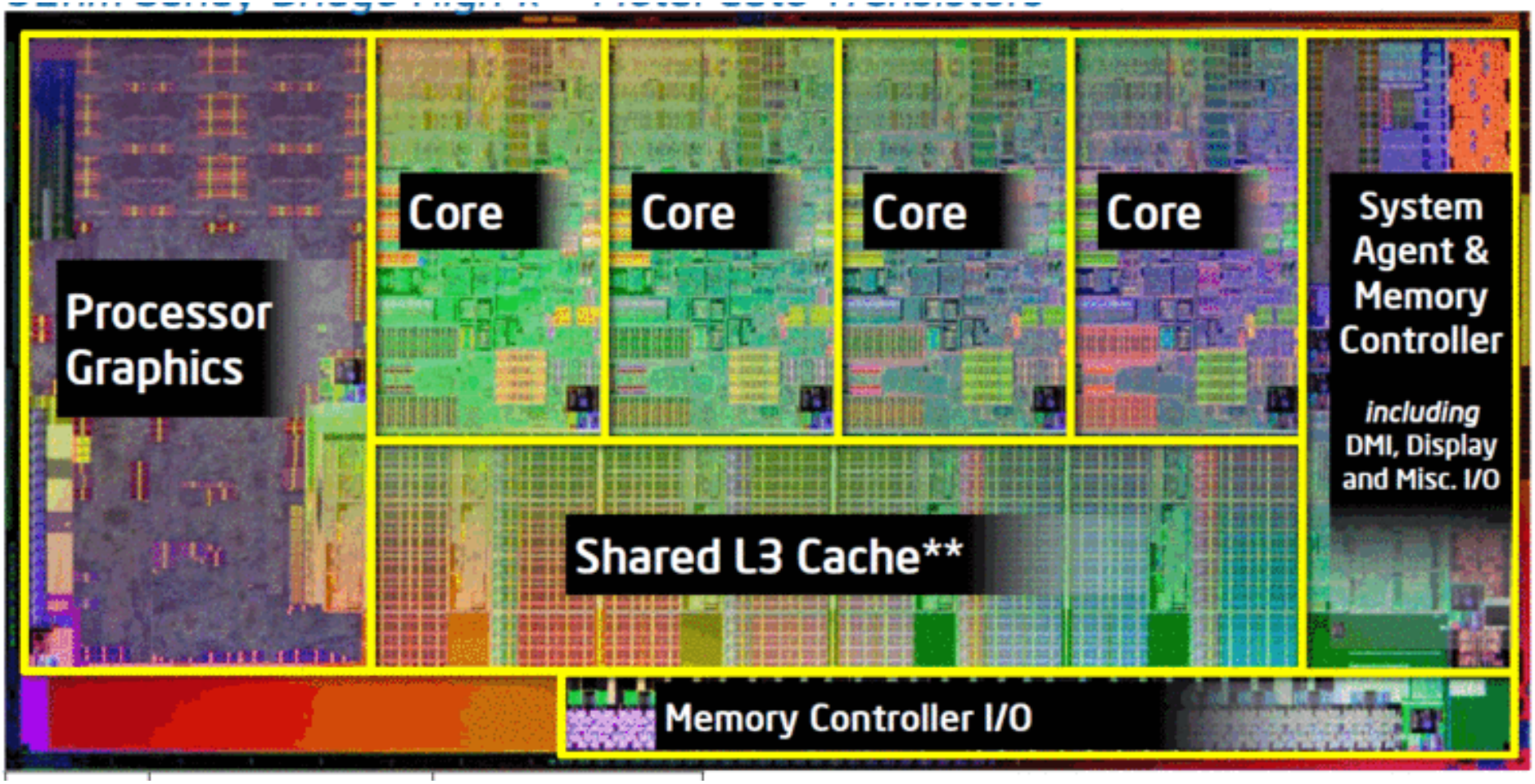
- 2006年7月発表
  - 65nmプロセス
  - 143mm^2
  - 291M トランジスタ
  - 65W
- Core Micro Architecture
  - Intelligent power capability
  - Micro-Fusion
    - RISC vs CISC
  - Advanced Smart Cache



Intel Developer Forum

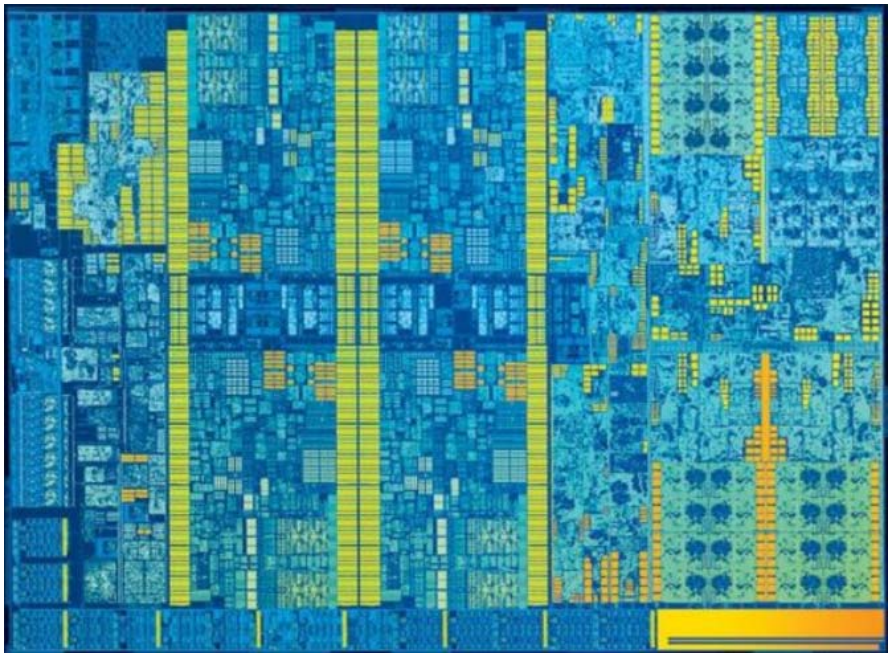
Intel Sandy Bridge, January 2011

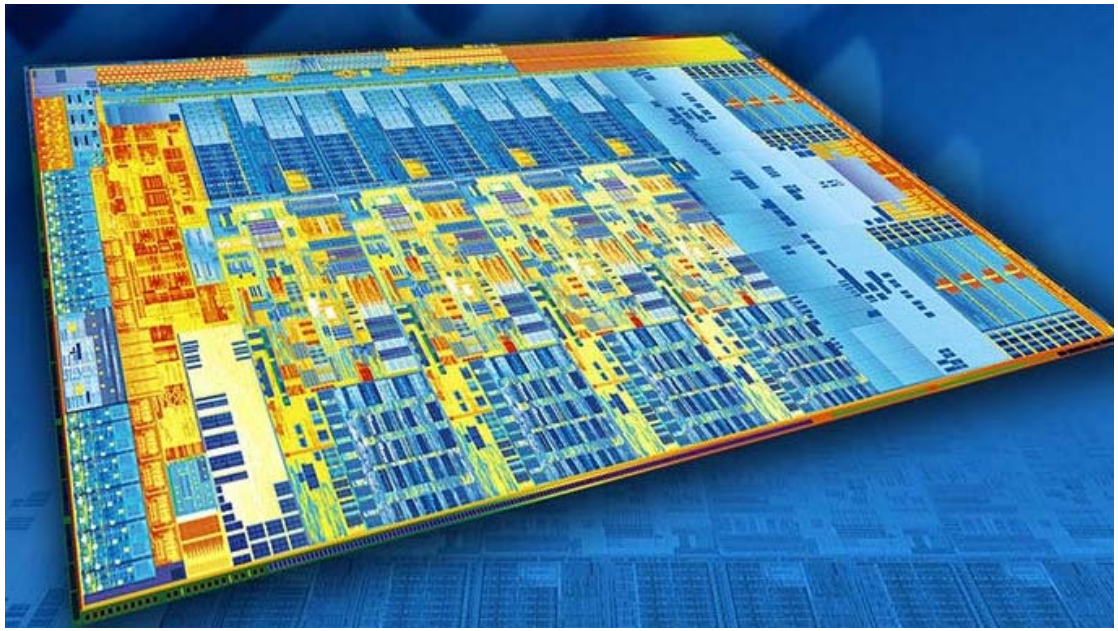
- 4 to 8 core



Intel Skylake, August 2015

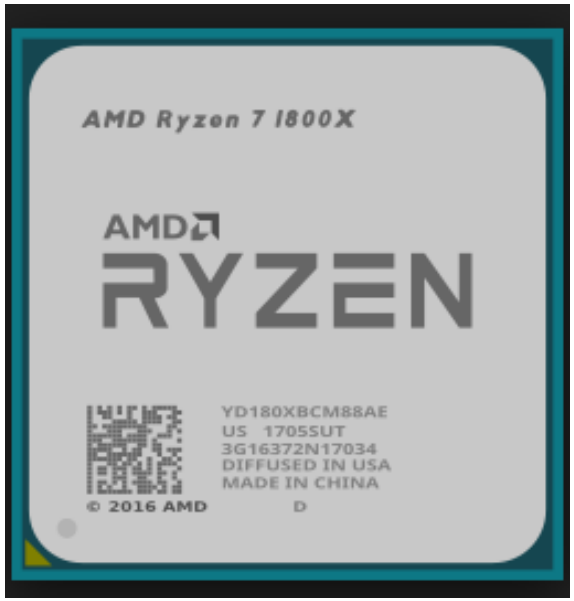
- 4 core

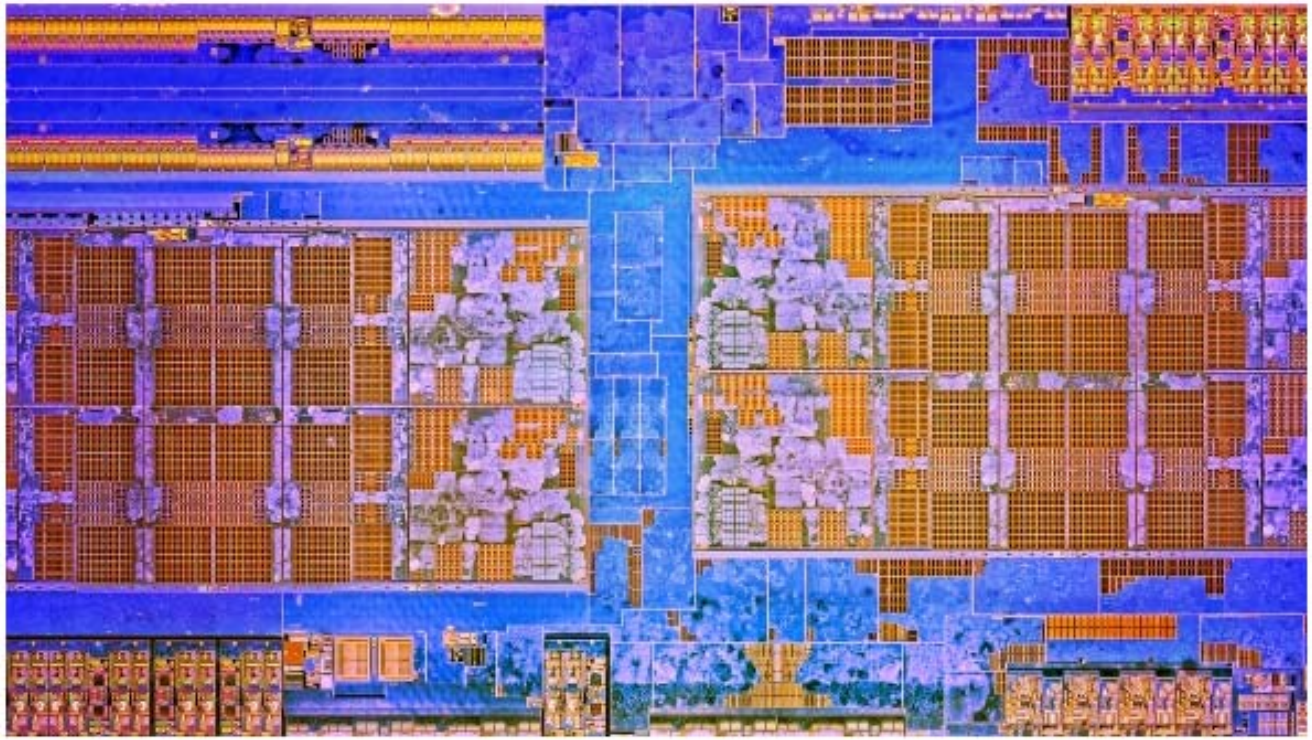




AMD Ryzen 7 1800X, March 2017

- 8 core, 4GHz





教科書

- コンピュータの構成と設計 第5版、パターソン＆ヘネシー（成田光彰　訳）、日経BP社
  - 1. コンピュータの抽象化とテクノロジ
  - 2. 命令:コンピュータの言葉
  - 3. コンピュータにおける算術演算
  - 4. プロセッサ
  - A. アセンブラ，リンカ，SPIMシミュレータ
  - B. 論理設計の基礎

コンピュータ論理設計 演習

- 演習は情報工学系 計算機室（学術国際情報センター 3階）でおこないます.
  計算機室（312号室）の定員は74, 314号室を用いると定員は88です.
- 情報工学系 計算機室のアカウントが必要. まだ無い場合には,「登録申込書」の記入が必要.

- 4人程度のグループを決めます.

Exercise(1)

- Project_1
  - Xilinx Vivado Design Suite の動作を確認する.
  - NEXYS 4 DDR FPGA Board の動作を確認する.
- **Project_2**
  - レポートを作成して提出する.
  - Verilog HDLで記述したコードのシミュレーション方法を学ぶ.
  - Verilog HDLの記述に慣れる.

Inside code001.v

- モジュールの定義はキーワードmoduleからキーワードendmoduleまで.
- moduleの後にモジュール名を書く. この例ではmainがモジュール名.
- モジュール名の後の括弧内に入出力の端子名を列挙する. ここでは端子は何も定義していない.
- セミコロン(;)で, モジュール名と端子の列挙を終える.
- キーワードinitialにより, シミュレーション開始時(時刻0)に一度だけ実行されることを指定する.
- $writeはシステムタスクの1つで, メッセージを出力する. 書式はC言語のprintfと同様.

Verilog HDL code (code001.v)

```
module main ();
  initial $write("hello, world¥n");
endmodule
```

Simulation output

```
hello, world
```

Verilog HDLのコードは青色で, シミュレーションの出力は黄色で示す.

Inside code002.v

- main.vをcode002.vの内容となるように入力して，シミュレーションする．
- 2つのシステムタスク$writeを用いた出力の例．2つのシステムタスクをブロックとしてまとめている．
- ブロックはキーワードbeginで始まり，キーワードendで終わる．C言語の{ }に対応．
- code002_ng1.vは2番目の$writeがinitialブロックに含まれないので文法エラーとなる．

code002.v

```
module main ();
  initial begin
    $write("hello, world¥n");
    $write("in Verilog HDL¥n");
  end
endmodule
```

```
hello, world
in Verilog HDL
```

code002_ng1.v

```
module main ();
  initial $write("hello, world¥n");
  $write("in Verilog HDL¥n");
endmodule
```

Inside code003.v

- main.vをcode003.vの内容となるように入力して，シミュレーションする．
- モジュール内で複数のinitialを用いても良いので，code002.vとcode003.vの出力は同じとなる．

code002.v

```
module main ();
  initial begin
    $write("hello, world¥n");
    $write("in Verilog HDL¥n");
  end
endmodule
```

```
hello, world
in Verilog HDL
```

code003.v

```
module main ();
  initial $write("hello, world¥n");
  initial $write("in Verilog HDL¥n");
endmodule
```

```
hello, world
in Verilog HDL
```

Inside code004.v

- main.vをcode004.vの内容となるように入力して，シミュレーションする．
- 整数integerとforループを用いた温度変換プログラムの例．
- 整数型のfahr, celsiusを定義．
- C言語の様に演算子++は使えない．fahr++ という記述はエラーとなるので注意．

code004.v

```
module main ();
  integer fahr, celsius;
  initial begin
    for (fahr = 0; fahr <= 300; fahr = fahr + 20) begin
      celsius = 5*(fahr-32) / 9;
      $write("%3d %6d¥n", fahr, celsius);
    end
  end
endmodule
```

```
  0    -17
 20     -6
 40      4
 60     15
 80     26
100     37
120     48
140     60
160     71
180     82
200     93
220    104
240    115
260    126
280    137
300    148
```

Inside code005.v

- main.vをcode005.vの内容となるように入力して，シミュレーションする．
- 指定した時間が経過するまで待たせる命令# を用いた例．
- #200 により，ここではシミュレーション開始時（時刻0）から200だけ時間が経過した時刻200に hello, world を表示する．
- #100 により，ここではシミュレーション開始時（時刻0）から100だけ時間が経過した時刻100に in Verilog HDL を表示する．
- 1行目はコメント，Verilog HDLのコメントはC, C++と同様．

code005.v

```
/* sample Verilog code */
module main ();
  initial #200 $write("hello, world¥n");
  initial #100 $write("in Verilog HDL¥n");
endmodule
```

```
in Verilog HDL
hello, world
```

Inside code006.v

- main.vをcode006.vの内容となるように入力して，シミュレーションする．
- $writeによる出力の順番はどうなるか？

code006.v

```
module main ();
  initial #200 $write("hello, world¥n");
  initial begin
    #100 $write("in Verilog HDL¥n");
    #150 $write("When am I displayed?¥n");
  end
endmodule
```

Inside code007.v

- main.vをcode007.vの内容となるように入力して，シミュレーションする．
- 出力はどうなるか？

- Vivadoのデフォルトの設定では1000nsしかシミュレーションしないので，Verilog is easy? は出力されない．

code007.v

```
module main ();
  initial #200 $write("hello, world¥n");
  initial begin
    #100 $write("in Verilog HDL¥n");
    #150 $write("When am I displayed?¥n");
    #1000 $write("Verilog is easy?¥n");
  end
endmodule
```

Inside code008.v

- main.vをcode008.vの内容となるように入力して，シミュレーションする．
- システムタスク$timeは，64ビットのシミュレーション時刻を返す．
- このコードでは，それぞれの $write が表示する時刻を表示する．
- 複雑な回路のシミュレーションでは，どの出力がどの時刻に出力されたのかわかりにくい場合がある．その場合，この例のように時刻を出力すると良い．

code008.v

```
module main ();
  initial #200 $write("%3d hello, world¥n", $time);
  initial begin
    #100 $write("%3d in Verilog HDL¥n", $time);
    #150 $write("%3d When am I displayed?¥n", $time);
  end
endmodule
```

```
100 in Verilog HDL
200 hello, world
250 When am I displayed?
```

Inside code009.v

- main.vをcode009.vの内容となるように入力して，シミュレーションする．
- システムタスク$finishは，シミュレーションを終了させる．
- このコードでは時刻210でシミュレーションが終了する．
- Vivadoのデフォルトの設定では1000nsシミュレーションするが，それより短い時間のシミュレーションや，ある条件でシミュレーションを終了させたい場合等に用いると良い．

code009.v

```
module main ();
  initial #200 $write("%3d hello, world¥n", $time);
  initial begin
    #100 $write("%3d in Verilog HDL¥n", $time);
    #150 $write("%3d When am I displayed?¥n", $time);
  end
  initial #210 $finish;
endmodule
```

```
100 in Verilog HDL
200 hello, world
```