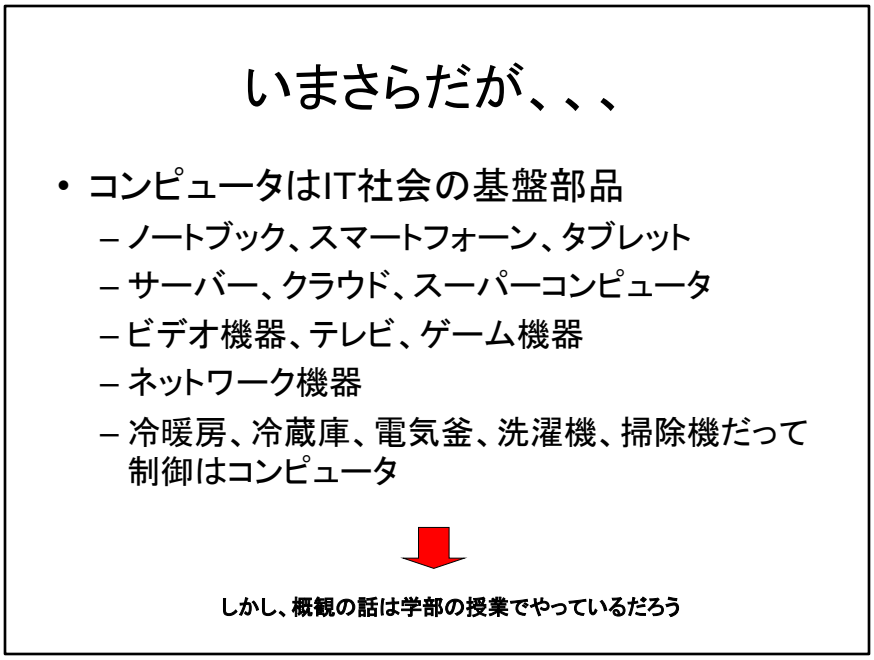
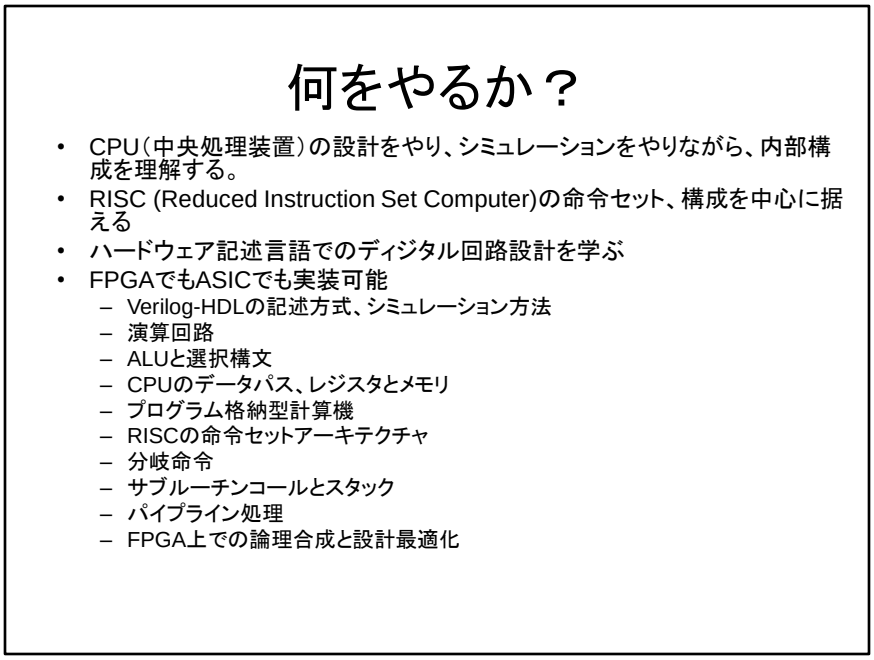
マイクロプロセッサ　第1回
ガイダンス
VerilogHDLのシミュレーション環境

慶應義塾大学理工学部
天野英晴
hunga@am.ics.keio.ac.jp
http://www.am.ics.keio.ac.jp



いまさらですが、コンピュータはIT社会の基盤となるパーツです。ノートブック、スマフォ、タブレットなど身近なコンピュータについては良くご存知だと思います。サーバー、クラウド、スーパコンピュータなど、大規模で普段は目に触れないコンピュータもありますが、皆さんがWebでAmazonやGoogleをクリックするとき、実はそれを使っているのです。ビデオやテレビ、ゲーム機器はコンピュータが主たる働きをします。ネットワーク機器は、もちろんネットワークの接続が主体ですが、制御するのはコンピュータです。これらは比較的コンピュータとして意識しやすいものですが、冷暖房装置、冷蔵庫、電気釜などの家電製品にだってコンピュータは使われています。このコンピュータの概観については既に学部の授業でやっていると思います。

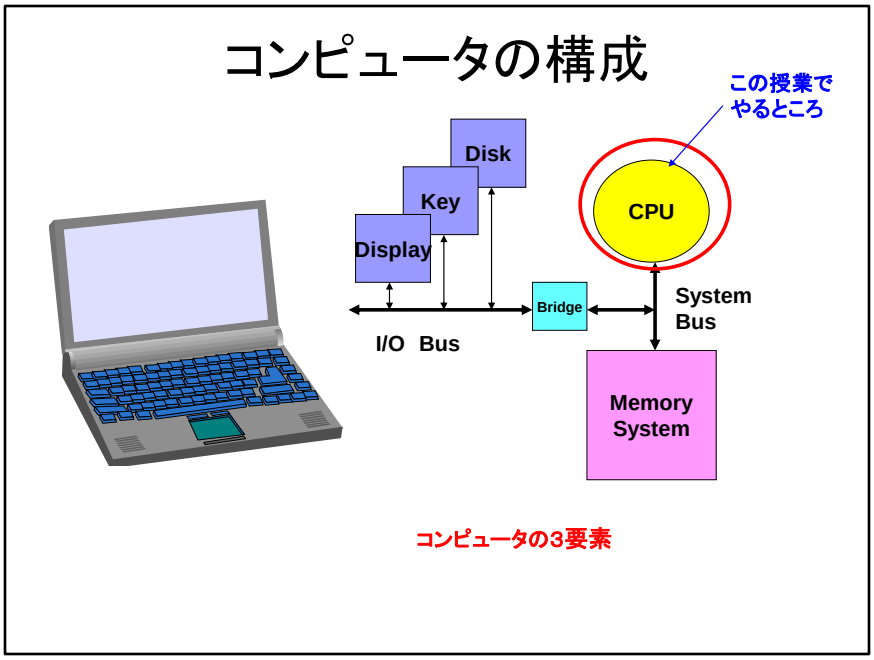


ではこの授業で何をやるかをざっくり説明しましょう。内田樹がどこかで書いていたようにシラバスほどバカバカしいものはありません。そこに書かれていたことを読んで、なんだかちゃんとわかるんだったら授業を受ける必要はないと言えます。とはいえ、雰囲気が分かること、自分の力で履修可能かどうか判断することは重要だと思います。この授業ではCPU（中央処理装置）の設計をやり、シミュレーションをやりながら、内部構成を理解します。CPUはRISC（Reduced Instruction Set Computer)の命令セットと構成を中心に据えます。現在のコンピュータはスマートフォンからスーパーコンピュータまでRISCでできており、これを勉強するのは当然といえます。この授業の特徴はハードウェア記述言語を使ってRISCを設計しながら、コンピュータを勉強していく点にあります。具体的な学習項目はスライドに列挙するとおりで、コンピュータの基本的な事項です。この授業ではコンピュータだけでなく、ハードウェア記述言語を使ったディジタル設計技術を勉強できる点に注意ください。

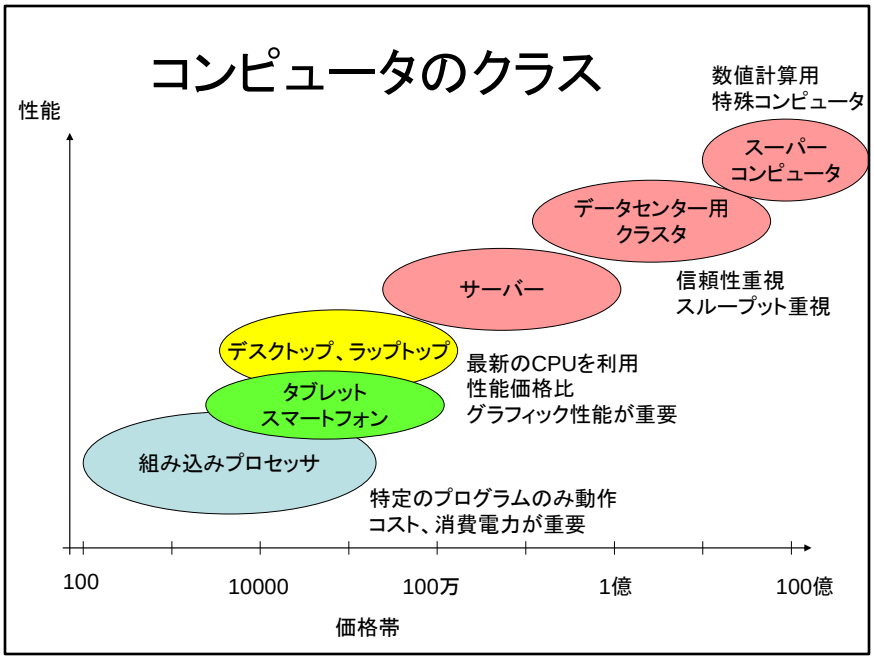
授業のやり方

- テキスト:「作りながら学ぶコンピュータアーキテクチャ」　天野、西村著　培風館→培風館が傾いたため入手困難に
- 授業資料はmabnabaおよびhttp://www.am.ics.keio.ac.jp/chuo_akiに掲示
- 授業を70分、演習を20分
- 演習は次の回までにメールで提出すれば問題ない
  - 休んだら資料をダウンロードして演習をやれば良い。
  - 週一回しか来ないが、メールによる質問は歓迎
- 成績の付け方
  - 最終演習＋毎回の演習によって付ける
  - 最終演習を提出しないと単位は取れない
  - 最後まで付いてこれればほとんどAが付く

この授業ではテキストを指定しますが、Webの教材は充実しているので、買わなくても大丈夫です。しかし、このスライドの説明よりも教科書を読んだ方がわかる場合も多いことは事実です。授業資料はWebに掲示します。今季からC-plusの利用は止めました。Webには用語集やVerilog－HDLの文法、設計事例も載っていますので、ぜひ見てください。この授業は、70分くらい授業をやって残りの20分で演習をやります。演習はごく簡単なものにします。最後に一つ大きな演習（最終演習）を出します。これと各回の演習の成績により最終的な成績を付けます。



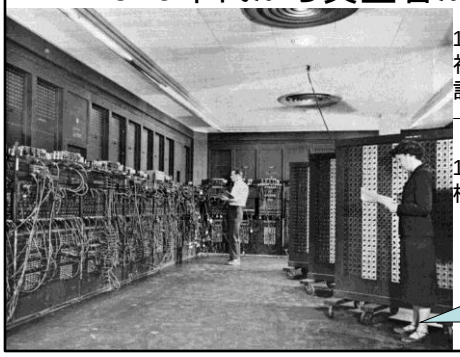
コンピュータは3つの部分に分けて考えます。中央処理装置CPU(Central Processing Unit)は、命令をメモリから取ってきて、それに従って演算処理を行う部分で、コンピュータの中心部です。メモリシステムは、命令、データを蓄えておく部分です。CPUに比べて地味な感じがしますが、実はコンピュータのコスト、性能に与える影響は大きいです。電子回路基礎で紹介したメモリ素子を組み合わせて利用します。最後に、外部との情報をやり取りを行うのが入出力装置（I/O:Input/Output)です。ディスプレイ、キーボード、マウスなどの人間とのやりとりを行う部分、Etherネットワーク、ディスクなどの補助記憶、USBなどを含みます。この３つの要素はどれも重要ですが、この授業では、CPUを中心に学びます。CPUはパイプライン処理まできっちり設計しながら紹介します。



コンピュータほど、値段と性能の幅の大きい製品はちょっと他にないのではないかと思います。もっとも安いものは数10円の組み込みプロセッサで、家電などの製品に組み込まれています。高いものは主に科学技術計算を高速に行うスーパーコンピュータで1000億円掛かるものもあります。（スーパーコンピュータ京は1100億円しました）性能の幅も10の9乗（10億倍）のオーダーで違います。（この辺、非常にアバウトな議論です。念のため）これらは、当然、使われ方、外見、ポイントとなる機能が違っています。大きくわけると図のようなクラスに分けられます。同じクラスのコンピュータは使われ方や重要視される性質が同じだと思って良いです。これらのクラスをざっと紹介しましょう。一つ、覚えていて良いことは、これだけ値段と性能が違うのにも関わらず、その基本的な命令の作り方は共通だと言う事です。もっとも簡単な組み込みプロセッサとスーパーコンピュータは実はほとんど同じ命令の作り方が使われています。これがRISC（Reduced Instruction Set Computer）と呼ぶ方法で、この授業で学びます。

コンピュータ略史
1. コンピュータ誕生

- 機械式の計算機
  - バベジの階差機関、解析機関が有名
- 1940年代から真空管が利用可能に

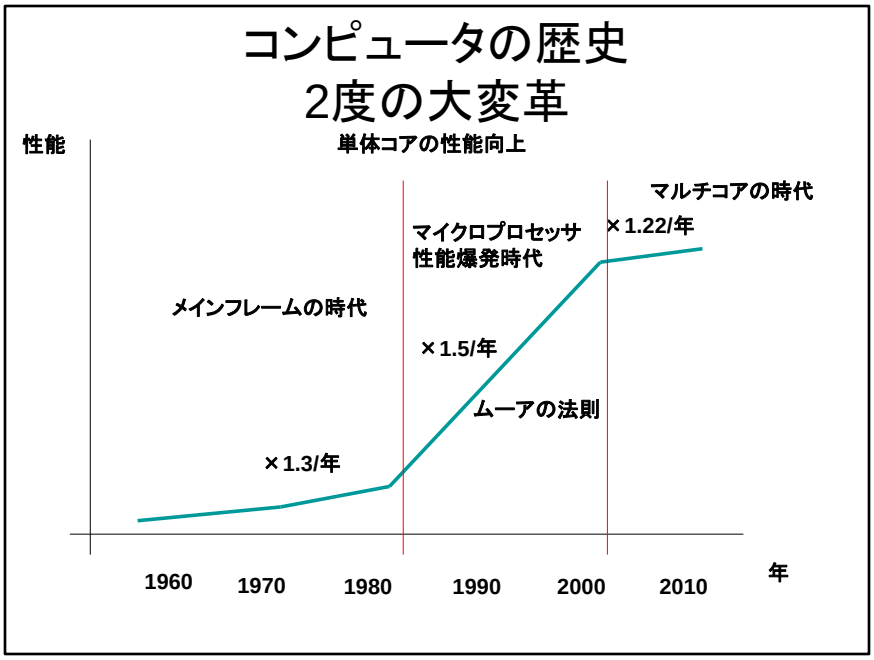


1946年に開発されたENIAC
初めて実用的に利用された電子式計算機
→プログラム格納型の考え方がまとめられる

1949年のEDSAC：世界初の電子式プログラム格納型計算機

ENIACは一部のプログラムを配線変更により行った。

ではここで、コンピュータの歴史を簡単に押さえておきましょう。自動的に計算を行う機械を作ろうとする試みは古くから行われており、中でもバベジの階差機関、解析機関はコンピュータの原理に通じるものがありました。1940年代に真空管が使えるようになると、電子式な計算機を作る試みが各地で行われました。このうちどれを世界最初の電子式コンピュータとするかは、諸説あるのですが、1946年に米国で稼働したENIACは、始めて実際に業務で用いられたコンピュータとして有名です。ENIACは、一部のプログラムを配線変更により行ったため、現在のコンピュータとは動作原理が違っていました。1949年、英国で稼働したEDSACは世界初の電子式プログラム格納型コンピュータであり、この方式が今でも使われています。



コンピュータの性能向上の概略を示します。縦軸は対数表示です。ここで、1980年代のメインフレームからPCへの変革、2003年に起きたマルチコア革命の二つの変化が重要です。

コンピュータ略史

(2)メインフレームの時代

1950年→1980年初頭

- 素子の発達により小型化、複雑化が進む
  - 真空管(第1世代)→トランジスタ(第2世代)→集積回路(第3世代)
- 事務計算、科学技術計算用に普及
  - 企業、大学単位で設置
  - パンチカードによるバッチ処理→ 端末を使った処理TSS(Time Sharing System)に
  - OS、プログラミング環境の発展
  - ミニコンピュータ、スーパーコンピュータ等目的別に分化が進む

コンピュータの基本的な動作原理はプログラム格納型でEDSACより変わっていません。しかし、性能は、基準とするコンピュータによりますが、現在の普通のPCは、EDSACの1億倍以上の計算能力を持っています。（EDSACは水銀遅延線をメモリに使っていたため実行できる命令は1秒間に650回でした。普通のPCでは100億以上の命令を実行することができます）全く基本方式が同じのままで性能の増加とコストの低下がすざまじい点でコンピュータは工業製品としてはかなり特徴的であると言えます。このコンピュータの進歩は、登場直後から始まり、第一世代の真空管から第二世代のトランジスタ、さらに第三世代の集積回路と次々に新しいデバイスを使い、世代の交代を進めていきました。

コンピュータの登場からしばらくは、高価であったため、メインフレームと呼ばれる大型コンピュータを皆が共有で使いました。このメインフレームは、企業や大学単位で設置され、事務計算、科学技術計算に用いられました。当初は、パンチカードを使ってプログラムを打ち込み、しばらく待つと結果がプリンタから出力されるバッチ処理と呼ばれる方法で使われました。この環境は、端末を使ったTSS（Time Sharing System)に置き換わりました。メインフレームは、専用で使うことができなかったため、研究者にとっては使い難い存在でした。このため、占有して使えるミニコンピュータが登場しました。一方、高速な科学計算用にスーパーコンピュータも登場して、コン

ピュータの目的別分化が進みました。

コンピュータ略史

(3)コンピュータ大発展時代

1980年中ごろ→2003年

- 個人が使うパーソナルコンピュータ、ワークステーションへ急激に移行
  - インターネットの発達
  - Ethernetの普及
  - 標準OS（Windows,Linux）の普及
  - 半導体技術の発展
  - RISC（Reduced Instruction Set Computer)とこれに伴う高速化、パイプライン処理、スーパースカラ型
  - マイクロプロセッサの猛烈な性能向上

  → 年間1.5倍に性能が向上（ムーアの法則）

  動作周波数は3GHzに達する
- コンピュータはあらゆる場所で使われ、なくてはならないものとなる

1980年代になると、コンピュータのCPUが1つの半導体素子に収まるようになり、マイクロプロセッサが発展しました。半導体素子自体の発達に加えて、RISCの登場とパイプライン処理、命令レベル並列処理など、この授業で勉強する高速化手法が発達し、全体として年間1.5倍（１８ヶ月で倍）に近いくらいの勢いで性能が上がるようになりました。これをムーアの法則と呼びます。これに、インターネットの発達、標準OSの普及、Ethernetの普及などが加わり、コンピュータの使われ方に大変革が起きました。今までの大型計算機を多数で共同で使う方法から、個人がパーソナルコンピュータを使い、それらがローカルコンピュータネットワークで接続され、インターネットを経由して世界中のコンピュータが接続され、情報を交換し、Webを閲覧するという現在の使われ方に以降したのです。コンピュータは家庭にも進出し、あらゆる電化製品に組み込まれて、世の中のあらゆるところで使われるようになりました。

コンピュータ略史

（4）マルチコア時代

2003年→現在

- 単体CPUの性能が限界に達する
  - 発熱、消費電力の問題
  - 同時に実行できる命令数が限界に
  - CPUのスピードにメモリが付いていけない
  - 半導体の性能向上の限界
- 周波数を上げるのではなくCPU（コア）の数を増やす→マルチコア
  - GPU（Graphic Processing Unit)などのメニーコアによる高速化も普及
- パーソナルコンピュータからタブレット、スマートフォンにコンピュータの主な使われ方が移りつつある

このコンピュータの性能爆発時代は、2003年まで続きました。この間コンピュータの動作クロック周波数（授業中に説明します）は数MHｚから３GHｚまで上昇しました。しかしここで再び転機が訪れます。あまりにクロック周波数が上昇したため、コンピュータで消費する電力が増加し、熱を発散させるのが困難になってきたのです。さらに、CPUの中で行う高速化手法が限界に達し、CPUに比べてメモリの速度が上がらないなどの問題点のため、単体のCPUの性能を向上させるのが困難になってきました。2003年にマイクロプロセッサを主導してきたIntel社はその方針を変更し、単体のCPUの性能を向上させるのはもう止めて、一つの半導体の中のCPU（コア）の数を増やすことによって性能の向上を目指すことを宣言しました。マルチコア時代の到来です。現在、みなさんのお使いのノートPCには4－6個のコアが内蔵されています。また、コンピュータの利用はスマートフォン、タブレットと、大規模なデータセンターによるクラウドコンピューティングが主体となり、80年代以来、コンピュータの中心であったPCが衰退しています。またGPUなど新しいタイプのコンピュータも現れ、コンピュータはますます分化し、拡散して居ます。皆さんは、1980年代のメインフレームからPCへの変革、2003年に起きたマルチコア革命の二つの変化を頭に入れておいてください。

ハードウェア記述言語

- HDL (Hardware Description Language)
- ゲート接続図を使ったハードウェア設計は今は使われない
  - スケマティック設計と呼ばれる
- Verilog-HDLとVHDLの二つが標準
  - 最近は多くのCAD (Computer Aided Design)が両方を受け付ける
  - CADによる論理合成、圧縮によってゲート接続図（ネットリスト）に変換される
- 最近はCレベル設計（HLS）も一般化
  - 用途によって使い分けられている

この授業では、CPUをハードウェア記述言語で設計し、シミュレーションしながら、その動作を理解します。以降、今日はハードウェア記述言語を概観し、シミュレーションの方法、波形の観測の仕方を勉強します。ハードウェア記述言語は二つの勢力が拮抗しています。Verilog HDLとVHDLです。Verilog HDLの後継言語のSystem Verilogも、多少は使われ始めていますが、まだ従来のVerilogの方が良く使われます。この二つの言語は、データの記憶と流れ、その間に行われる処理に着目して、プログラミング言語に似た言語を使ってハードウェアを記述します。このような記述をRTL（Register Transfer Level)と呼びます。RTL設計されたハードウェアは、シミュレーションにより動作を確認した後、CAD(Computer Aided Design)を用いてゲート接続図（ネットリストと呼びます）の形に変換し、最適化を行います。最近はC言語とほとんど同じ（制約つき）言語を使って、ハードウェアで行う処理自体を記述すると、RTL記述に自動的に変換してくれるHLS（High Level Synthesis)技術が発展してきたため、ハードウェアをC言語（Javaで設計する方法もある）で記述する方法が一般的に普及してきました。この方法は複雑なアルゴリズムをハードウェア化するのには便利ですが、CPUやネットワークコントローラなどタイミング制御が重要なハードウェアの記述には向いていません。用途によって使い分けられています。

VerilogとVHDL

	Verilog-HDL	VHDL
出自	論理シミュレーション記述	仕様書
標準化	デファクトスタンダード	国際標準
記述	Pacal風（嘘）	PL/I→ADA
特徴	広い範囲でシミュレーションは可能	記述が厳格

ここではVerilog HDLを採用

Verilog HDLとVHDLは共に良く使われますが、その性格は全く違っています。
Verilog-HDLは論理シミュレーションを記述する言語として誕生し、使われているうちに標準化されたデファクトスタンダードです。構文（シンタックス）はPascal風と称していますが、ウソで、Pascalとは結構違った独自の構文です。シミュレーションが動作すれば、そのハードウェアの様子が分かるので、とにかく動作することが重要です。このため、Verilog HDLは、宣言などをいい加減に書いてもシミュレーション上は動作してしまいます。これはありがたいことなんですが、合成の段階までに問題点に気づかないと、とんでもないバグを含んだハードウェアを生成することになります。Verilog HDLは、文法上不合理な点を色々持っているため、これを改良したSystem Verilogが登場しています。しかし、ツールが対応しないので、ここでは使いません。ここで使う文法の範囲ならばVerilogもSystem Verilogもさほど変わりません。
一方、VHDLはハードウェアの仕様書記述用の言語が発達したもので、厳格な構文を持ちエラーチェックをきっちり行います。米国国防省が制定したPL/I、ADAの流れを汲む構文で、国際標準としてトップダウンに制定されました（PL/IとADAは、米国国防省がこれ以外の言語で書いたソフトウェアの納入を認めなかったため、プログラミング言語として一時期相当使われました。しかし、あまりの融通の利かなさに腹を立てたプログラマたちは、長年

の苦闘の末にこれを絶滅に追い込みました）。記述が厳格なので、シンタックスエラーを修正する段階でバグをかなり減らすことができます。一方、簡単なハードウェアの記述にも多数の行数を要するので不便です。

　ここではVerilog　HDLを使います。僕は最初はVHDLを使っていた（使っていたCADがこれしか受け付けなかった）のですが、この言語があまり好きになれず、Verilog HDLに切り替えました。VHDLでハードウェアを設計していると、新しいシステムを設計するんだ、というクリエイティブなことをやっているのではなくて、わかりきったシステムの仕様書を書いているような気分になってくるんです。もちろん、なんで書こうとクリエイティブな設計はできるのですが、ま、気分の問題です。幸いにして最近のCADは両方を受け付けてくれますので、どちらかで設計できれば問題ありません。

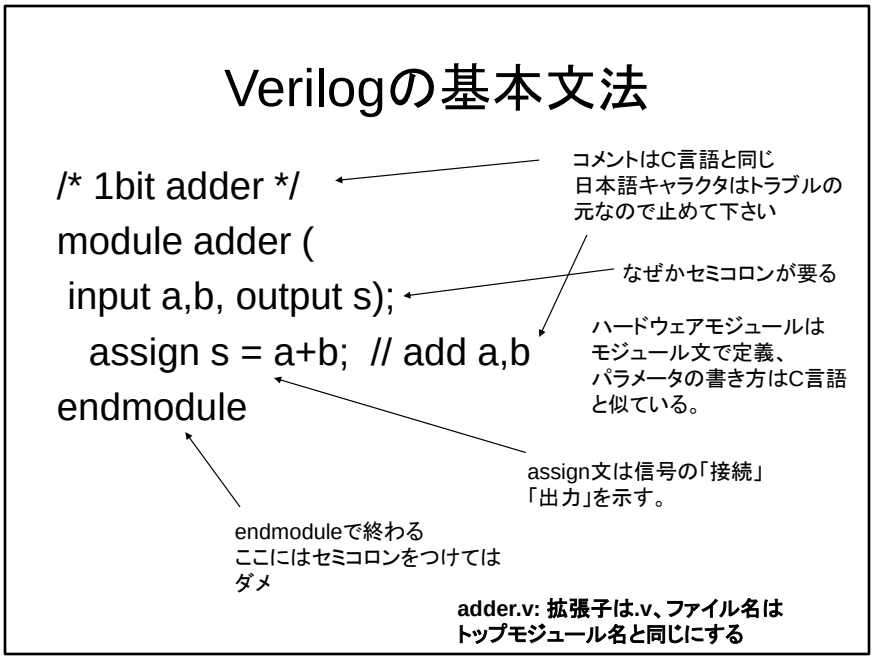
演習の方法

- それぞれITセンターのLinuxマシンにログイン
- 演習用のファイルを持って来る
  - wget http://www.am.ics.keio.ac.jp/chuo_aki/1kai.tar

tar fileの解凍

- cd ディレクトリ名
- tar xvf 1kai.tar
- cd 1kai
- ls

次にこのダウンロードされたファイルを解凍します。このファイルはtarという由緒正しいLinuxのアーカイブ（書庫、ファイルをまとめて一つにするやり方）形式になっています。（なんてったってtarのtはtapeでテープ時代から使われてたものです）ダウンロード先のディレクトリに行き、tar xvf 1kai.tarというコマンドで解凍します。そうすると1kaiというディレクトリが出来てくるので、このディレクトリに入ります。以降、解説する演習用ファイルもこのディレクトリに入っています。



では、演習用ディレクトリ中のadder.vを見てみましょう。これがVerilog HDLで書いた1ビットの加算器です。C言語のプログラムの拡張子を.cにしたのと同様に、Verilog HDLのファイルの拡張子は.vにします。また、ファイル名はトップモジュール名（最上位階層のモジュール名、ここではadder）にします。エディタは皆さんの好きなのを使ってください。僕はviを使いますが、皆さんはemacsがお好きな方が多いかと思います。

さて、まず最初の行はコメントです。コメントの付け方はCと同じで/* */で囲むか、//の後に書くかどちらかです。ここで日本語を使いたくなる人が居ると思いますが、日本語のフォントがトラブルの元となるので止めてください。英語でコメントを付けましょう。さて、Verilog　HDLは（VHDLも）、ハードウェアをモジュールという単位で階層的に記述していきます。この場合1ビットの加算器が1つのモジュールになります。Verilogの記述はモジュールの定義から始まります。

module文の後にモジュール名を書き、これに続く（　）内に入出力端子を定義してやることでモジュールが定義されます。ここではadderがモジュール名で、inputの後のa,bが入力端子名、outputの後のsが出力端子名です。後に紹介する方法で指定しない場合は1ビットの端子として宣言されます。

これらはカンマで区切っていくつでも並べて書くことができます。input文、output文自体が複数出てきても問題ありません。ここでC言語との違いは、な

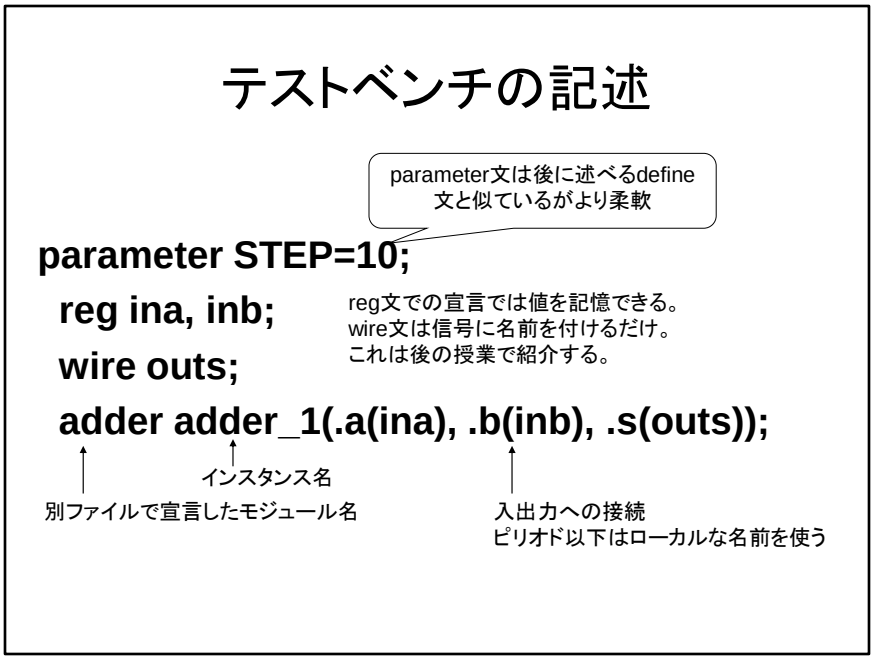
ぜか最後の　）の後にセミコロンが必要な点です。
モジュールを定義したら、次からの文はそのモジュールの構造あるいは動作を書きます。ここではassign s=a+b;でa入力とb入力の加算結果をsに出力する、あるいはaとbを加算器に入れた出力を s に接続する、という意味があります。Verilogでは単純に＝を使うことはできず、必ずassignを先に付けます。（これについては後に詳しく解説します。）文の終わりはC言語と同様にセミコロンを付けます。
このモジュールはこれで終わりなので、endmodule文でモジュールの終わりを宣言します。この文の終わりにはセミコロンを付けてはいけません。

テストベンチ(test.v)

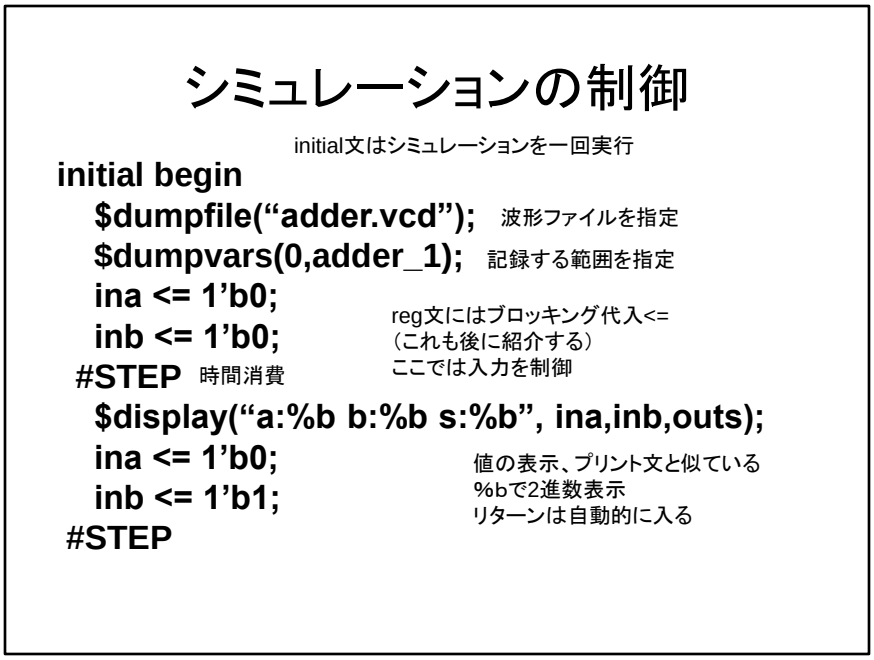
- シミュレーション制御のための記述

```
module test;
  parameter STEP=10;
  reg ina, inb;
  wire outs;
  adder adder_1(.a(ina), .b(inb), .s(outs));
  initial begin
    $dumpfile(“adder.vcd”);
    $dumpvars(0,adder_1);
    ina <= 1’b0;
    inb <= 1’b0;
  #STEP
    $display(“a:%b b:%b s:%b”, ina,inb,outs);
    ina <= 1’b0;
    inb <= 1’b1;
 #STEP
…
```

adder.vを記述したらこれをシミュレーションしてテストする必要があり、このために別のモジュールが必要になります。このモジュールは、実際のハードウェアではなく、特定のモジュールをテストするだけの目的の記述で、テストベンチと呼ばれます。ここではmodule testをtest.vというファイル中に記述してあります。これをエディタで開いて見て下さい。テストベンチはそれ自体の入力はないので、モジュール名の後ろの（　）はありません。
Verilogは、論理シミュレーションの制御用の言語から発達したので、シミュレーション用の記述が充実しています。これは裏を返せばやたらにある機能を使わないと簡単な回路のシミュレーションができないことで、このテストベンチを書くことが入門者の壁になっています。ただし、テストベンチは基本的に標準パターンしか使わないので、あまり深く意味を考えず、シミュレーションをしてみましょう。ここで簡単に説明しますが、あまり深く突っ込まない方がいいです。



まず、最初にparameter文でシミュレーションの実行の1ステップの時間を定義しています。ここでは10nsecがシミュレーションの単位時間である旨を宣言しています。このSTEPという記号を使ってシミュレーションの時間を進めて行きます。次にina, inb, outsというテストベンチ中の信号名を定義しています。ここでregはregisterの略で、データを記憶します。これに対してwireは単に信号に名前をつけているだけです。この辺は後の授業でよく説明しますので、ここではあまり触れないでおきます。さて、次にadderから始まる行で、adder.vで宣言したadderの実体を生成します。ここで、モジュール名は先ほど定義したadderを使う必要があります。これに対して、実体名は何でもいいのですが、ここではadder_1という名前にします。実体名の後の（　）の中で、モジュールの入力にテストベンチの信号を割り当てます。ここではaにinaをbにinbを、sにoutsを繋いでいます。モジュールの入出力名にはピリオドを付けてその後の（　）内に上の階層の信号名を書いてやります。信号名が直接指定されているので、書く順番はどうでも良くなっています。



initial文以下はシミュレーションの制御を行う部分です。この文はシミュレーションを順に一回実行します。beginからはじまってendで終わります。最初に＄dumpfileと＄dumpvarsで、シミュレーション結果を記録する波形ファイルと、そこに記録する信号の範囲を指定します。Verilogでは＄から始まるのは標準関数です。この授業では波形ファイルはvcd形式を使います。この形式はファイルサイズが大きくなる欠点があるのですが、簡単で昔から使われているのでほとんど全てのCADで扱ってくれます。拡張子にはvcdを付けてください。ファイル名の部分は何でもいいのですが、adderをテストするのでadderという名前にしています。次のdumpvarsは記録する信号の範囲を示し、ここでは実体名がなければなりません。今回adder_1を指定してこのモジュールの信号を全て記録します。最初の0は全て記録することを示します。
次に入力信号に値を設定します。ina, inbはregで宣言したので、値を覚えておいてくれます。これに＜＝で0、0を設定します。この＜＝はブロッキング代入文といい、レジスタに値を設定するときに使います。これも後ほど詳しく紹介します。それから＃STEPで10nsec時間を進めます。それから$display文で結果を表示します。このdisplay文はC言語のprintfとほとんど同じですが、%bというフォーマットで、2進数の表示ができます。また、改行は自動的に入ります。これで入力が０，０の時の入出力信号が表示されます。また、＃STEPでシミュレーション時間を進め、全ての組み合わせの入力を入れて結

果を表示し、最後に $ finihshでシミュレーションを終了します。その後に
initial文のbeginに対応するendを書きます。
テストベンチもモジュールの一つなので最後はendmodule文が必要です。

Verilog-HDLのシミュレーション

- Ikarus Verilogを利用
  - ○コンパイル型のフリーソフトウェア
    - Linux, Windowsマシンにインストール可能
    - iverilog XX.vでコンパイル、必要なファイルを全部書く
    - vvp a.out（あるいは単に./a.out)で実行、かなり高速
  - ○Verilog2000に対応
  - ×遅延付シミュレーションができない
- 波形Viewerはgtkwave
  - ○フリーソフトウェア
    - Linux,Windowsマシンにインストール可能
    - gtkwave XX.vcdで起動
    - 基本的なViewerの機能は全て持つ
  - ×他のViewerに比べて少し使いにくいかも、、

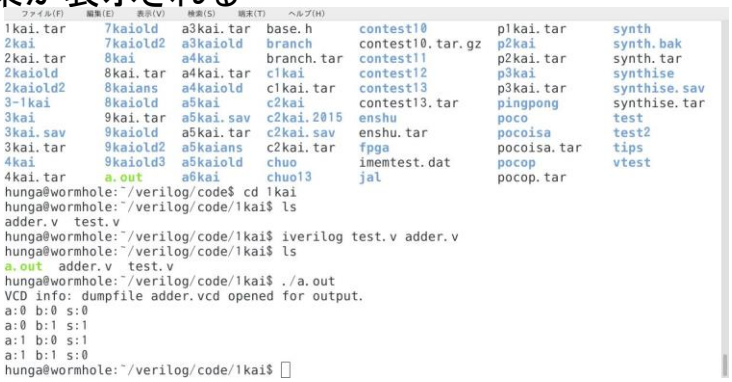
さて、では今まで紹介したテストベンチtest.vを用いてadder.vをシミュレーションしてみましょう。ここで使うのはIkarus Verilogというフリーソフトウェアを利用します。Ikarus　Verilogはコンパイル型のシミュレータで、まずiverilog test.v adder.vと打ち込んでコンパイルします。C言語同様、必要とするファイルを全て並べます。シンタックスエラーがなければa.outという実行形ができます。ここで、vvp a.outというコマンドを打ち込むことによりシミュレーションができます。ありおはC言語同様、./a.outでも実行できます。フリーソフトウェアにしては高速で、元々のVerilogだけでなく改訂版のVerilog2000にも対応してくれます。Linux, Windows両方に対応するので、皆さんのPCにインストールすることも可能です。「作りながら学ぶコンピュータアーキテクチャのページ」にサイトが紹介されています。

ではやってみよう

iverilog test.v adder.v

./a.out

結果が表示される

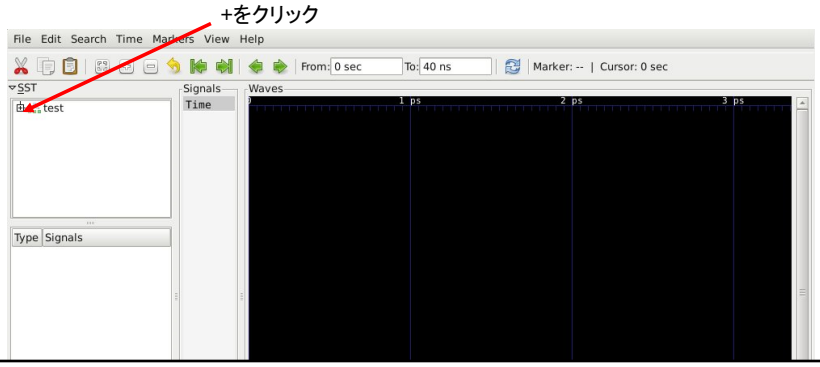


以下はリモートデスクトップが
使える場合に限る

gtkwaveを使って見よう

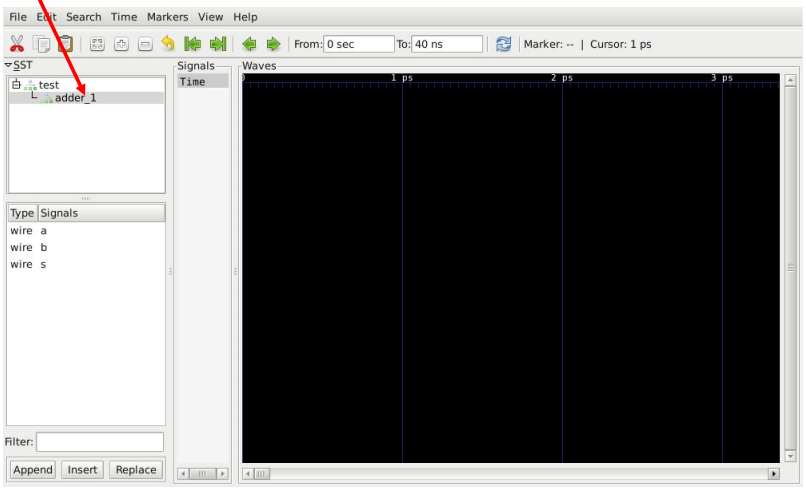
- gtkwaveは強力なデバッグ用ツール
- これなしではとても演習は乗り切れない！

gtkwave adder.vcd

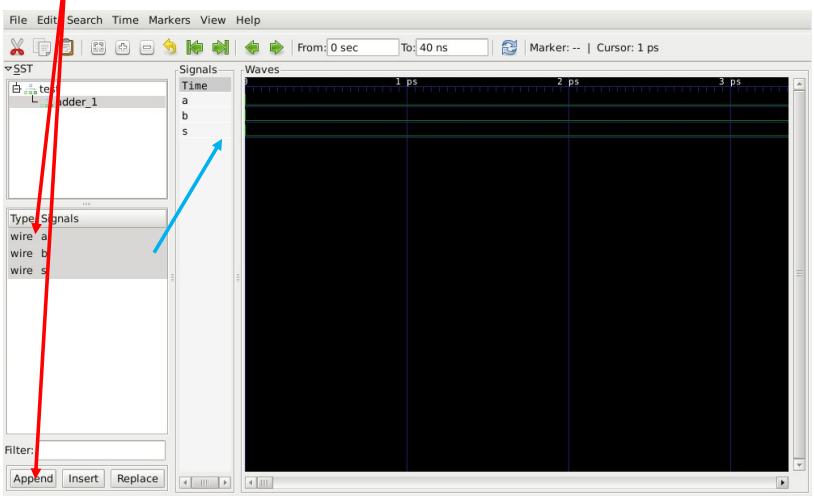


gtkwaveは波形ビュアー、すなわちシミュレーション結果を波形として観測するツールです。大変強力なツールなので、ぜひ使いこなせるようにしてください。

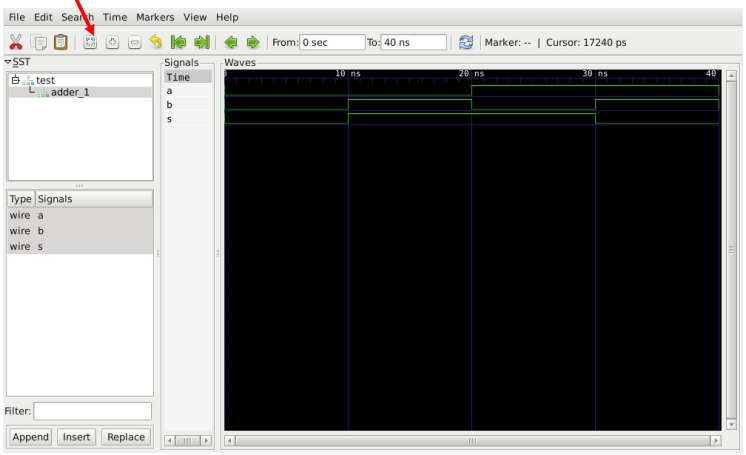
モジュール名をクリックすると信号名が下の窓に表れる



信号を選択してAppendを押すと右の窓に波形が表れる
信号名をDrag-and-Dropしてもいい



Zoom Fitを押すといい感じに表示される
ー印（Zoom Out)を連打してもいい



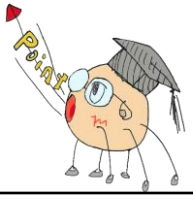
コンピュータの基礎　まとめ

- コンピュータはCPU,メモリ、I/Oの3要素からできている
- コンピュータは色々なクラスがあって、性能、価格が10億倍（いい加減）も違うけれど、どれもほとんど同じ命令体系で動く
- コンピュータはできてから75年くらいの歴史を持つが基本方式はプログラム格納型で変わっていない。しかし性能は1億倍（いい加減）くらい良くなっている
- コンピュータの歴史で重要なのは1980年代の使い方の革命と、2003年のマルチコア革命。今はマルチコア時代が続いている。

ではインフォ丸によるまとめです。コンピュータの基礎はこの４点を覚えておいていいでしょう。コンピュータは非常に特殊な工業製品といえます。

HDLのまとめ

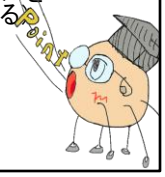
- HDLとは、レジスタに対するデータの記憶と、レジスタ間のデータの流れをプログラム言語風に記述する（RTL設計）ハードウェア設計用言語
- Verilog HDLはmodule単位でハードウェアを記述する
  - module/endmodule
  - input/output
  - assign
- テストベンチはシミュレーションのやり方を記述する
  - initial
  - $display
  - $dumpfile
  - $dumpvars
  - $finish



次はHDLのまとめです。

Verilogシミュレーションの実行

- シミュレーションのコンパイル
  - iverilog *.v
  - ディレクトリ内に同一モジュールがある時は、ファイルを全て指定する
  - iverilog test_poco.v poco1.v rfile.v alu.v など
  - エラーが出た場合、メッセージを良く読んで！
- シミュレーションの実行
  - vvp a.out　あるいは./a.out
  - iverilog 実行時に-oで実行ファイル名を指定することができる。
- 波形の表示
  - gtkwave XX.vcd
  - モジュールを選択すると信号名が表示される
  - これをクリックして選択→Appendをクリックすると波形が表示される
  - スケールがpsecなのでマイナス（－）をクリックしまくってスケールを調整（一番左のZoomFitをクリックすると自動的に調整してくれる）



最後は演習実行の方法のまとめです。これは、この授業で何度も使います。

演習

加算を論理AND（＆）に置き換えたan.vを作ろう。
andは予約語なので、注意

提出は
hunga4125@gmail.comあるいはmanabaのレポート
　で提出しやすい方にする。

Subject: Chuo 学籍番号　名前
名前はローマ字で書いてください

では実際にシミュレーションをやってみましょう。加算を論理積に書き換えるきわめて簡単な課題をやってみます。