

nand2tetris

目录

1 Module0 Introduction	2
2 Module1 Boolean Functions And Gate Logic	2
2.1 Boolean Logic	3
2.2 Boolean Functions	3
2.3 Logic Gates	3
2.4 Hardware Description Language	3
2.5 Hardware Simulation	4
2.6 Multi-Bit Buses	4
2.7 Project 1 Overview	4
2.8 Perspective	5
3 Module 2 Boolean Arithmetic And The ALU	5
3.1 Binary Numbers	5
3.2 Binary Addition	5
3.3 Negative Numbers	6
3.4 Arithmetic Logic Unit	7
3.5 Project 2 Overview	9
3.6 Perspectives	9
4 Module3 Memory	9
4.1 Sequential Logic	9
4.2 Flips Flops	10
4.3 Memory Units	11
4.4 Counters	12
4.5 Project 3 Overview	13
4.6 Perspectives	13
5 Module4 Machine Language	14
5.1 Machine Language Overview	14
5.2 Machine Language Elements	14
6 Appendix 2 Hardware Description Language	14
6.1 HDL basics	14
6.2 Multi-bit busses	?
6.3 Built-in chips	?
6.4 Sequential chips	?
6.4.1 Clock	?
6.4.2 Sequential, built-in chips	?
6.4.3 Sequential, regular chips	?
6.4.4 Feedback loops	?
6.5 Visualizing chips	?
6.6 HDL Survival Guide	?
6.6.1 Chip implementation order	?
6.6.2 HDL files and test scripts	?
6.6.3 Testing chips in isolation	?

6.6.4	HDL syntax errors	?
6.6.5	Unconnected pins	?
6.6.6	Customized testing	?
6.6.7	Bit numbering and bus syntax	?
6.6.8	Sub-bussing (indexing) internal pins	?
6.6.9	Multiple outputs	?
6.6.10	Chip-parts "auto complete"	?
7	Appendix 4 The Hack Chip Set	?

课程网站: [Home | nand2tetris](#)

coursera: 依据基本原理构建现代计算机: 从与非门到俄罗斯方块 (基于项目的课程) | Coursera
Build a Modern Computer from First Principles: Nand to Tetris Part II (project-centered course) | Coursera

书籍笔记: [The Elements of Computing Systems](#)

1 Module0 Introduction

[Module 0: Introduction Roadmap | Coursera](#)

Nand2tetris的相关软件设置分为两个部分。

文件夹project分为了14个等待修改和完成的项目。tools文件夹包含着项目中必须的工具。具体的细节官网建议用到的时候再细看[Software | nand2tetris](#)。

阅读任务: Introduction chapter of *The element of computer systems*

课程将分为两个部分, 第一个部分将用来构造硬件, 第二个部分将用来构造软件。

- unit 0.0 introduction: 老师的自我介绍
- unit 0.1 the road ahead: 介绍interface,abstraction,implementation,hierarchy的概念
- unit 0.2 from nand to hack: 介绍part1的架构以及相应的要实现的内容。
- unit 0.3 from hack to tetris: 介绍part2的内容。
- unit 0.4 project 0 overview: 介绍了part1中要用到的tools(Hardware simulator,CPU simulator and Assembler)和projects(00—06)。
- project 0: 提交一个txt文件压缩后的zip版本。用来熟悉coursera的提交系统。

2 Module1 Boolean Functions And Gate Logic

[Module 1: Boolean Functions and Gate Logic Roadmap | Coursera](#)

Slides : [chapter1](#)

布尔代数, 基本逻辑门, HDL以及硬件模拟是主要的学习内容。

阅读任务: chapter1, [appendix2](#) and [appendix4](#) of *The element of computer systems* and the possible question about HDL in [FAQ | Coursera](#). Also [HDL survival guide](#) and [Hardware Simulator Tutorial](#) (图很多, 我就不做笔记了)。

项目内容: [Project 01 | nand2tetris](#). pdf: [project01](#)

在动手之前请先阅读[appendix2\(HDL guide\)](#) 以及 [appendix4\(chip set\)](#)

2.1 Project 1 Overview

unit 1.7: 讲明了project1的一些注意事项。简单介绍了mux和demux以及add16的设计思路, 相关的设计文档以及最佳实践。

Best practice advice

- Implement the chips in the order in which they appear in the project guidelines
- If you don't implement some chips, you can still use them as chip-parts in other chips (use their built-in implementations)
- You can invent additional, "helper chips"; However, this is not necessary. Implement and use only the chips that the architects (we) specified
- In each chip implementation, strive to use as few chip-parts as possible
- When defining 16-bit chips, the same chip-parts may appear many times. That's fine, use copy-paste-edit.

That's It!
Go Do Project 1!

图 1. best practice

Not. 直接Nand(a=in,b=in,out=out)即可。

And. Not(Nand)的逻辑即可。

Or. $\neg(\neg a \wedge \neg b) = (a \vee b)$

Xor. $(a \wedge \neg b) \vee (\neg a \wedge b) = a \oplus b$

Mux. $(a \wedge \neg \text{sel}) \vee (b \wedge \text{sel})$

Dmux. $a = (\text{in} \wedge \neg \text{sel}) \quad b = (\text{in} \wedge \text{sel})$

Not16. 复制16个Not

And16. 复制16个And

Or16. 复制16个Or

Mux16. 复制16个Mux

Or8Way. 7个Or门二分。

Mux4Way16. 三个Mux16二分

Mux8Way16. 七个Mux16二分

DMux4Way. 三个Dmux二分

DMux4Way. 七个Dmux二分

2.2 Perspective

unit 1.8: 一个类似Q&A的环节。剩下的project提交都需要付费后的许可，就暂时不限定时间了。这一部分中最有意思的是nand与非门的原理介绍。

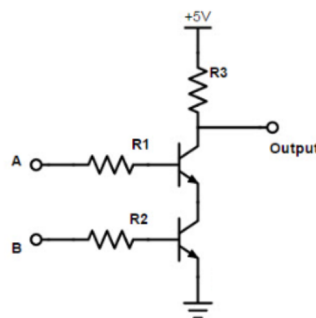


图 2. nand

A或B有一个不通时，output联通+5v，为1。两者都联通时接地(+5处有电阻)，为0。

3 Module 2 Boolean Arithmetic And The ALU

Module 2: Boolean Arithmetic and the ALU Roadmap | Coursera

Slides : [chapter2](#)

二进制及其加法，补码，半加器全加器，ALU，组合逻辑是主要的学习内容。

阅读任务：应当是 *chapter2 of The element of computer systems*。

项目内容：Project 02 | nand2tetris。pdf: [project02](#)

3.1 Project 2 Overview

简要介绍了Project2中几个chip的设计思路(half-adder use xor,and. full-adder use two half-adder)。

- You will have to use chips implemented in Project 1;
For efficiency and consistency's sake, use their built-in versions, rather than your own HDL implementations
Simple rule: Don't add any HDL files to the project 2 folder.

That's It!
Go Do Project 2!

图 3. Best practice project 2

设计思路如下：

Half-adder. Xor=sum And=carry进位

Full-adder. 共有三个输入a,b,c。第一个半加器a+b，输出sum1和carry1。第二个半加器计算sum1+c，输出sum以及carry2。carry1和carry2使用加法逻辑Xor即可。

Add16. 16个Full-adder逐级相连即可。

Inc16. 本来打算用Add16，后来发现用16个Half-Adder应该性价比更高。

ALU. ALU有两份测试文件，第一份是不带那两个状态位输出的。第二份是完整的输出文件。对于control bits的控制作用只需用Mux16的sel对应即可。ng直接对应与out的符号位，而zr则可以用两个Or8Way和一个Or一个Not来解决。

3.2 Perspectives

ALU is extremely simplified. multiplication and division is the work of the software in nand2tetris II. Using built-in chips so you can focus on your project working now.

4 Module3 Memory

Module 3: Memory Roadmap | Coursera

Slides : [chapter3](#)

组合逻辑与时序逻辑，时钟，寄存器，RAM是主要的学习内容。

阅读任务：应当是 *chapter3 of The element of computer systems*。

项目内容：Project 03 | nand2tetris pdf: project03

4.1 Project 3 Overview

简要介绍了project3中要实现的chip。

Given :

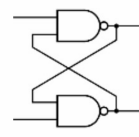
- All the chips built in projects 1 and 2
- Data Flip-Flop (built-in DFF gate)

DFF implementation

- The DFF gate logic implementation requires, among other things, connecting Nand gates with feedback loops
- The resulting implementation is elegant, yet impossible to realize on the supplied hardware simulator

The hardware simulator does not allow loops of combinational chips (like Nand)

- Instead, we use a built-in DFF implementation:



Part of a possible DFF implementation

```
/** Data Flip-flop: out(t) = in(t-1)
 * where t is the current time unit. */
CHIP DFF {
    IN in;
    OUT out;
    BUILTIN DFF;
    CLOCKED in;
}
```

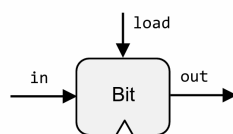
Implementation notes:

We need the DFF as a chip-part in one chip only: Bit

But... All the other memory chips are built on top of it.

图 4. DFF

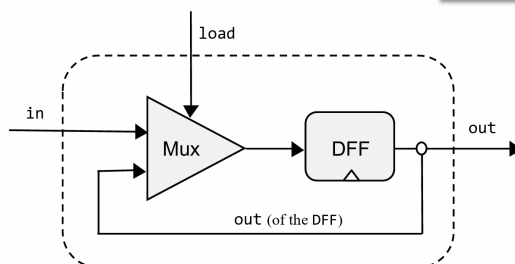
1-Bit register



Bit.hdl

```
/** 1-Bit register:
 * if load(t) then out(t+1) = in(t)
 * else out(t+1) = out(t) */
CHIP Bit {
    IN in, load;
    OUT out;

    PARTS:
    // Put your code here:
}
```

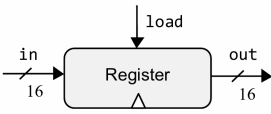


Implementation tip:

Follow the chip diagram

图 5. 1-bit register

16-bit Register

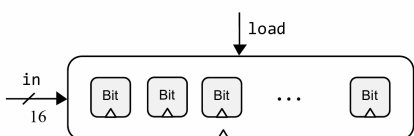


Register.hdl

```
/** 16-bit register:
  if load(t) then out(t + 1) = in(t)
  else          out(t + 1) = out(t) */

CHIP Bit {
  IN in[16], load;
  OUT out[16];

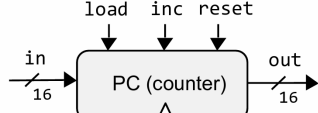
  PARTS:
    // Put your code here:
}
```



Implementation tip:
Follow the chip diagram

图 6. 16-bit register

16-bit Counter



```
/**
  A 16-bit counter.
  if      reset(t) out(t + 1) = 0      // resetting
  else if load(t)  out(t + 1) = in(t)   // setting
  else if inc(t)   out(t + 1) = out(t) + 1 // incrementing
  else            out(t + 1) = out(t)   // maintaining
  */

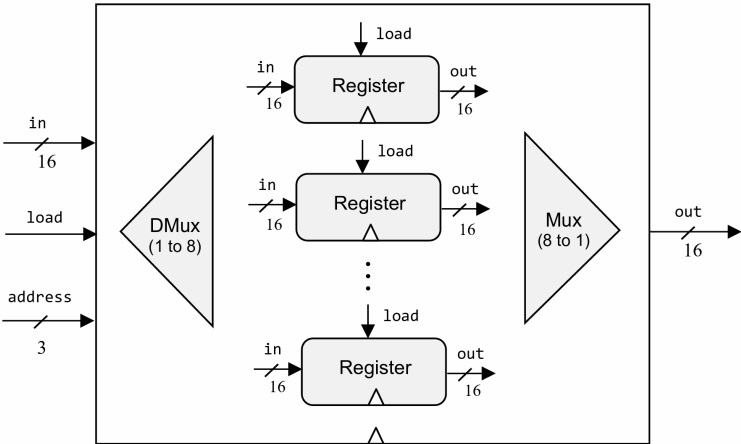
CHIP PC {
  IN in[16], load, inc, reset;
  OUT out[16];

  PARTS:
    // Put your code here:
}
```

Implementation tip:
Can be built from a Register, an Incrementor, and Mux's

图 7. 16-bit counter

8-Register RAM

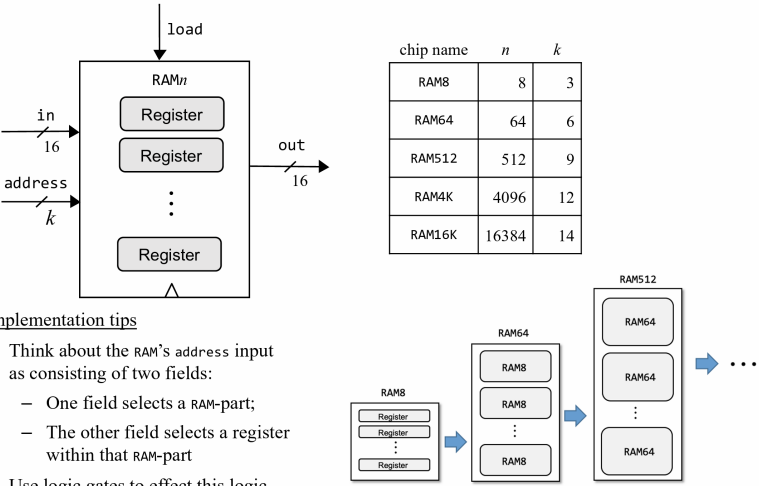


Partial diagram, showing some of the chip-parts, without connections

Implementation tip:
Follow the chip diagram, and figure out the missing connections.

图 8. RAM8

n -Register RAM



- Implementation tips
- Think about the RAM's address input as consisting of two fields:
 - One field selects a RAM-part;
 - The other field selects a register within that RAM-part
 - Use logic gates to effect this logic.

图 9. RAM-n

- 1-bit register.** 如图所示，一个Mux和DFF即可。
- 16-bit register.** 如图，16个1-bit register即可。
- 16-bit counter.** 如图，可以从图中的if的顺序看出控制位的优先级。3个Mux16，一个Inc16以及一个16bit-register即可。
- RAM8.** 如图即可，1个Dmux和1个Mux，以及8个16-bit register。
- RAM64.** 如法炮制，只是register换成RAM8。

RAM512—RAM16K. 逐级推进即可。

4.2 Perspectives

最有意思的部分是介绍DFF的具体原理的部分。这一部分的具体解释我推荐去看《数字逻辑基础与verilog设计》中对基本锁存器的解释。

5 Module4 Machine Language

Module 4: Machine Language Roadmap | Coursera

Slides : [chapter3](#)

阅读任务：应当是 *chapter4 of The element of computer systems*。

项目内容：Project 04 | [nand2tetris](#) pdf: [project04](#)

6 Appendix 2 Hardware Description Language

6.1 HDL basics

link: [appendix2](#)

chip内部的引脚可以自由命名，但输入输出的引脚一般来说是不可以的。它们的名字一般已经被芯片的架构以及文档中写好的api规定好了。举个例子，在课程中老师们提供了一批stub files，规定好了相应的chip interface。

HDL不是编程语言，HDL其实是规格描述语言。HDL并不关心过程(processes)，其只关心连接(connection)。故而芯片中的实现部分的流程并不是定死的。

HDL对大小写敏感(case-sensitive)。注释可以使用//，/* */ 以及 /** */。

在文件中声明的chip名称必须与文件名称相同。Chip Xxx —> Xxx.hdl。

一个HDL程序包含一个接口(interface)以及一个相应的实现(implementation)。接口包含chip的api，chip的名字，输入输出的引脚名。实现部分则包含了在PARTS下的语句。其形如：

```
/** API documentation: what the chip does. */
CHIP ChipName {
    IN inputPin1, inputPin2, ... ;
    OUT outputPin1, outputPin2, ... ;
PARTS:
    // Here comes the implementation.
}
```

Parts: The chip implementation is a series of chip-part statements, as follows:

```
PARTS:
    chipPart(connection, ... , connection);
    chipPart(connection, ... , connection);
    ...
```

图 10. HDL program

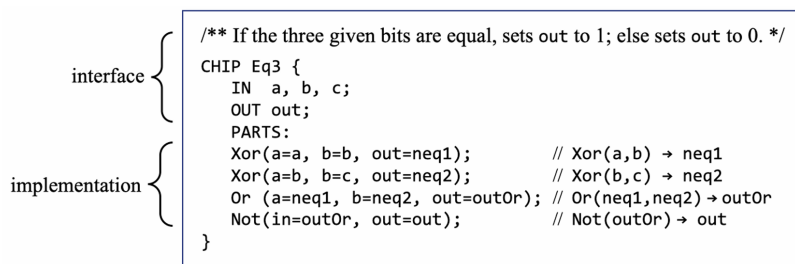


Figure A2.1: HDL Program example.

图 11. interface and implementation

6.2 Multi-bit busses

a multi-bit value = bus

内部针脚的Bus似乎不允许下标,形如u[i]是不允许的。ture和false一样可以用来定义bus, 例如x[0..2]=true。Bus内部的bit默认为0。

6.3 Built-in chips

在本课程中, nand和dff可以直接使用, 无需构造。

built-in chips 简单来说就是使用高级语言实现的chips。

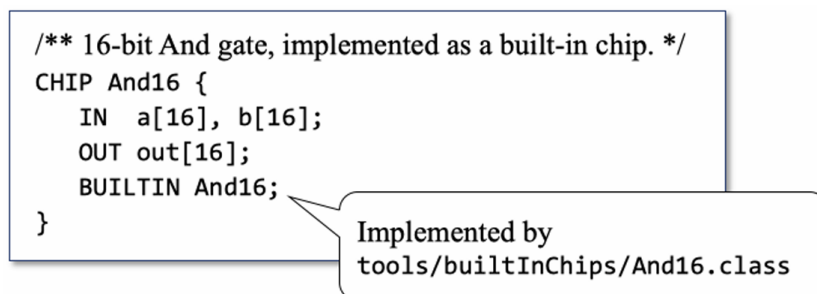


Figure A2.3: Built-in chip definition example.

图 12. built-in chip

built-in chip 的优点:

- Foundation: 可直接提供芯片的implementation, 方便使用。
- Efficiency: 有一些chip使用高级语言实现可以更便捷, 更快速。
- Unit testing: 只需抽象地关心接口, 方便测试。
- Visualization: 更方便地展现原理。
- Extension: 易于扩展。

6.4 Sequential chips

6.4.1 Clock

chip可分为组合逻辑与时序逻辑(combinational, or sequential)。组合逻辑是与时间(时钟)无关的, 当输入的值改变, 输出的值便改变。时序逻辑则与时序相关, 或者说与时钟clock有关: 当输入的值改变, 输出的值仅会在下个时间单元(time unit)的开始改变。

时钟通过两个模拟命令tick和tock控制。tick moves the clock value from t to $t+$, and a tock from $t+$ to $t+1$, bringing upon the next time unit.

时钟可以通过gui的按钮控制，也可以通过测试脚本控制。默认情况下chip是组合逻辑的。

6.4.2 Sequential, built-in chips

一个built-in chip可以显式地声明其对于时钟的依赖。使用语句CLOCKED pin, pin, ..., pin;即可。输入的pin x 规定了 x 的变化应当只在下一个时间单元的开始改变芯片的输出。输出的pin x 规定了此芯片任意输入的变化应当只在下一个时间单元的开始改变 x 。

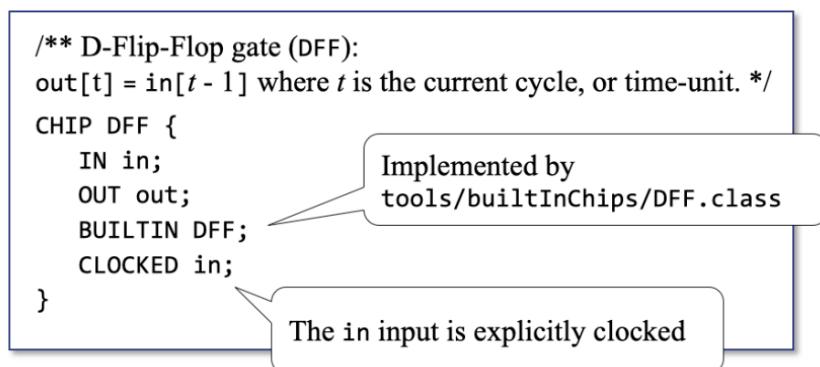


Figure A2.4: DFF definition.

图 13. dff definition

也可以在CLOCKED关键字后不做任何针脚的声明。这表示芯片内部的针脚可能是时序逻辑的，但其的输入输出仍然是组合逻辑的。

6.4.3 Sequential, regular chips

非built-in chip该怎样知道其的逻辑属性呢？

当芯片的组成部分是时序逻辑时，其会被认定为时序逻辑。时序逻辑的属性是递归的。If such a chip is found, it renders every chip that depends on it (up the hierarchy) "clocked"。如果一块芯片不是built-in chip，那么我们是无法从HDL代码中查明其是否是时序逻辑的。

最佳实践是在chip api中提供相应的芯片属性信息。

6.4.4 Feedback loops

当一块芯片的输出来源直接或间接（通过很长的一段依赖回路）与自己的输出中的一个相关联，我们便称之为feedback loops（反馈环）。例子如下：

```

Not(in=loop1, out=loop1) // Invalid feedback loop
DFF(in=loop2, out=loop2) // Valid feedback loop

```

两者都形成了反馈环，不同之处在于Not是组合逻辑，而Dff是时序逻辑。在Not中，loop1形成了一个瞬时且不受控的在in与out直接的依赖，我们有时称之为数据竞争(data race)。而Dff则受制于时钟。于是out(t)并非in(t)的函数，而是in($t-1$)的函数。

当模拟器加载芯片时，其会检测芯片的链接是否牵涉反馈环。如果有，模拟器会检测此循环是否是通过时序逻辑实现的。如果是便没问题，如果不是便会报错以防止不受控制的数据竞争。

6.5 Visualizing chips

hardware simulator 具有将built-in chips 使用GUI表示的能力。有一些芯片会具有改变GUI显示的能力(GUI—side effect)，我们可以通过hardware simulator将其显示出来。

```
// Demo of GUI-empowered chips.
// The logic of this chip is meaningless, and is used merely to force
// the simulator to display the GUI effects of its built-in chip-parts.
CHIP GUIDemo {
    IN in[16], load, address[15];
    OUT out[16];
    PARTS:
        RAM16K(in=in, load=load, address=address[0..13], out=a);
        Screen(in=in, load=load, address=address[0..12], out=b);
        Keyboard(out=c);
}
```

Figure A2.5: A chip that activates some GUI-empowered chip-parts.

图 14. Gui-empowered chip

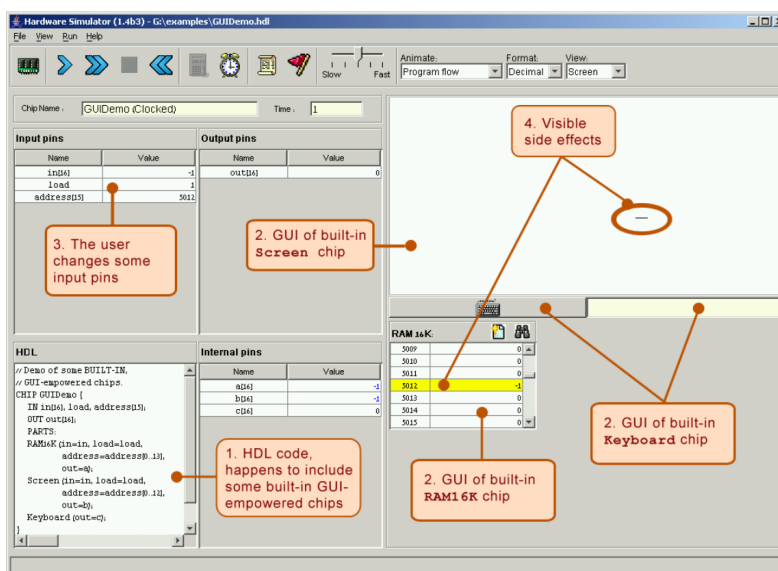


Figure A2.6: GUI-empowered chips demo. Since the loaded HDL program uses GUI-empowered chip-parts (step 1), the simulator renders their respective GUI images (step 2). When the user changes the values of the chip input pins (step 3), the simulator reflects these changes in the respective GUIs (step 4).

图 15. Hardware simulator

图中的操作是：将-1(1111111111111111)存入地址为5012的内存单元，计算机在screen上描绘了一条长为16像素的细线。一种简易地操作显存的方法。

6.6 HDL Survival Guide

6.6.1 Chip implementation order

nand2tetris/projects中与硬件相关的文件夹为01, 02, 03, 05。每一个文件夹中都包含着提供好的你需要实现的stub files。你需要按顺序设计好。例如你在实现Xor之前应当实现And和Or。

当模拟器运行chip-part时，它会搜寻当前文件夹。会有三种情况发生：

- 未发现相应hdl文件。此时built-in的实现会生效，覆盖未发现的相应芯片的实现。
- 发现了stub file。模拟器会试着运行，但因为没有相应的实现，运行失败。
- 发现了具有实现的文件。顺利运行或报错。

最佳实践：按项目以及书籍中的顺序实现chip，你将不会遇到实现顺序相关的问题。

一个推荐的方法是创建一个stub文件夹并将所有提供的stub文件移入到里面去。之后将你想要实现的芯片的stub文件一个一个移入工作目录。完成后将文件移入completed的文件夹，从而迫使simulator一直使用built-in chip。

6.6.2 HDL files and test scripts

你正在实现的hdl文件应当和测试脚本同处一个文件夹。当有疑惑时，查看显示的hdl文件的名字并查看显示的hdl文件的代码。

最佳实践：使用模拟器的file菜单加载hdl文件或是tst文件，但是不要同时加载。

6.6.3 Testing chips in isolation

如果你的测试出现了问题，但你又感觉自己的实现是完美的，那么可能是你实现中用到的chip-part出现了问题。硬件设计的困境之一就是测试脚本无法保证被测试的芯片在任何情况下都运行完美，特别是复杂的芯片。

你可以将自己实现的芯片的hdl,tst以及out文件移动到一个独立的次级目录中并测试，如果通过便说明是chip-part的问题。之后再将chip-part移动到文件夹中测试直到找出问题芯片。

6.6.4 HDL syntax errors

Hdl报错会在模拟器下方显示，有时会因为屏幕过小导致不显示。此时移动一下窗口就好。或是使用Alt+Space, M,再加上方向键。

6.6.5 Unconnected pins

模拟器不会将未连接的针脚视作错误。默认情况下其会将未连接的针脚置为0。这很可能导致看起来很神秘的错误。如果有什么输入输出的值总是0，检查是否有针脚对齐时的拼写错误，并检查针脚之间是否连接好了。如果希望自己有某个针脚不输出，将其链接到一个不存在的针脚即可。例子如下：FullAdder(a=a[15],b=b[15],c=carry15,sum=out[15],carry=ignored);

6.6.6 Customized testing

测试失败时请通过输出文件查明失败的原因，可以通过模拟器或是文本编辑器查看。你也可以进行个性化的测试。拷贝测试文件后改名，然后更改测试命令。改动output-file行上的output文件名，删除compart-to行。这将会使测试文件直接运行到结束(默认情况下测试会在output行与compare行相异时停止)。也可以修改output-list行查看内部针脚的值。appendix 3有着Test Scripting Language的详细命令。

6.6.7 Bit numbering and bus syntax

sel = 110 —> sel[2]=1,sel[1]=1,sel[0]=0

多位的bus输入想要置为0/1只需直接将针脚连接至false/true即可。

例子如下：And16(a=x,b=false,out=fx1) (x=x[16]);

6.6.8 Sub-bussing (indexing) internal pins

只有输入输出可以使用bus的下标形式，内部针脚是不可以使用的。但我们可以有如下的解决方法：

```
CHIP Foo {
  IN in[16];
  OUT out;
  PARTS:
    Not16 (in=in, out=notIn);
    Or8Way (in=notIn[4..11], out=out); // Error: internal bus cannot be indexed.
}
```

Possible fix, using the workaround:

```
Not16 (in=in, out[4..11]=notIn);
Or8Way (in=notIn, out=out); // Works!
```

图 16. internal pins subscripted workaround

6.6.9 Multiple outputs

将bus分为独立的两个bus可以使用如下方法：

```
CHIP Foo {
    IN  in[16];
    OUT out[8];
    PARTS:
        Not16  (in=in, out[0..7]=low8, out[8..15]=high8); // Splitting the out value
        Bar8Bit (a=low8, b=high8, out=out)
}
```

图 17. bus split

本质上是chip输出多个out并连接到内部引脚上。

将输出分为独立的两份使用：

```
CHIP Foo {
    IN  a, b, c;
    OUT out1, out2;
    PARTS:
        Bar (a=a, b=b, out=x, out=out1); // Bar's output feeds the out1 output of Foo
        Baz (a=x, b=c, out=out2);        // A copy of Bar's output also feeds Baz's a input
}
```

图 18. output split

6.6.10 Chip-parts "auto complete"

如上所有提及的chip接口都在appendix 4中列出来了。若想使用直接拷贝即可。

7 Appendix 4 The Hack Chip Set

link: [appendix4](#)。需要实现的chip的api以及signature。HDL survival guide的内容已被appendix 2&4覆盖，故不在此赘述。