THUSC2023 Day2 学习手册

THUSC / THUWC 工程题补全计划

目录

引言 3
符号和约定 4
提交答案格式与限制 4
门电路 5
组合门电路 6
非法门电路示例 6
正边沿 D-触发器 6
测评方式 7
模拟器使用规范 7
模拟器使用示例 9
各部分分数 12
推荐做题顺序 13

引言

现代社会，计算机已经成为人类不可或缺的一部分。随技术进步，计算机的结构愈加复杂。不过就一种通俗的说法而言，计算机本质上可以视为是处理一系列电压（信号）的数字逻辑电路——将特定的高电压（高电平）视为 1，将特定的低电压（低电平）视为 0，那么所有数据都可以以二进制的形式被表示为一系列电压（信号）。那么计算机的功能可以抽象为：将在一段时间内输入的以电压的形式表示的数据，进行特定的运算，随后以电压的形式输出。

数字逻辑电路可以简单的分为组合逻辑电路，以及时序逻辑电路。

对组合逻辑电路，其功能可以抽象为一个函数 $f:\{0,1\}^n\to\{0,1\}^m$（其含义为，该函数接受 n 个输入 $x_1,\cdots,x_n$，满足 $\forall i\in[1,n],x_i\in\{0,1\}$，并产生 m 个输出 $y_1,\cdots,y_m\in\{0,1\}$，下同）。可以证明，仅通过复合“逻辑与非”函数，就能得到任意的这样的函数 f，其中，“逻辑与非” $\mathrm{nand}(x,y)=0\Leftrightarrow x=1\wedge y=1$。对应到现实世界就是“与非门”，这是一种特定的电路，其接受两个为高电平或低电平的电压（信号），并输出一个高电平或者低电平的电压（信号）。除了“与非门”，还有诸如“与门”、“或门”、“非门”等电路，它们分别对应“逻辑与”、“逻辑或”和“逻辑非”运算，也对应于 `C++` 中的 `&`、`|`、`!` 三种操作符。这些电路统称为“门电路”。通过这些“门电路”，我们可以实现任意一种组合逻辑电路的功能，在这个题目的前半部分（即组合逻辑部分），需要你尝试通过使用指定的门电路，完成实现诸如“加法器”、“比较器”、“（简单的）算术逻辑单位”等基础的组合逻辑电路。

对时序逻辑电路，其功能则复杂一些。在其整个运行过程中，可以划分为若干个周期；在每个周期中，仍然如同组合逻辑电路一样，接受 n 个输入，并产生 m 个输出；但与之不同的是，第 k 个周期产生的输出不仅与第 k 个周期的输入相关，还与之前的周期相关。因此，电路必须具有某种“记忆”功能，才能存储之前周期的输入（更准确地说，之前周期的输入进行组合逻辑运算得到的中间结果），再和当前周期的输入一同经过某个组合逻辑电路，产生本周期的输出。

在时序逻辑电路中，与这种“记忆”相关重要电路被称为“寄存器”；“寄存器”的实现方式有很多，其中一种实现方式通过“触发器”实现。触发器接受若干输入，产生若干输出；但只在输入满足特定条件的情况下，输出才会发生改变；反之，如果不满足该条件，那么无论输入信号怎么变化，输出都不会改变；这样就能够实现“记忆”。触发器的种类有很多，不同触发器对应的条件不尽相同。例如，某种被称为“正边沿 D-触发器”的触发器接受两个输入 D,CP，产生一个输出 Q；当 CP 从 0 变为 1 时（**注意，不是 CP 为 1 时**），输出 Q 会变成 D；其余时间，无论 D 如何变化，Q 的值总是不发生改变。通常，CP 会被设置为“时钟信号”，由特定的电路生成，其信号值依次为保持一段时间的 0，随后保持一段时间的 1，随后再保持一段时间的 0，随后再保持一段时间的 1……一段完整的“保持一段时间的 0，再保持一段时间的 1”通常被称为一个周期。

在这个题目的后半部分（也就是时序逻辑部分），假定提供了这样一个“时钟信号”，你需要尝试使用门电路和正边沿 D-触发器，完成实现诸如“校验器”、“计数器”、“乘法器”等，直到最后实现一个简单的计算机的 CPU！

符号和约定

在本文中，使用正体字母代表输入信号，比如 A，D 和 CP；使用斜体字母代表输出信号，比如 Y，R 和 CNT；使用打印体数字代表逻辑值，比如 0 和 1。

使用 $\wedge$ 符号表示 **逻辑与**，其等价于 `C++` 语言中的 `&` 运算符。$x \wedge y = 1$ 当且仅当 x 和 y 都是 1。

使用 $\vee$ 符号表示 **逻辑或**，其等价于 `C++` 语言中的 `|` 运算符。$x \vee y = 1$ 当且仅当 x 和 y 中有一个是 1。

使用 $\oplus$ 符号表示 **逻辑异或**，其等价于 `C++` 语言中的 `^` 运算符。$x \oplus y = 1$ 当且仅当 x 和 y 中恰好有一个 1。

使用 $\neg$ 符号表示 **逻辑非**，其等价于 `C++` 语言中的 `!` 运算符。总是有 $x \oplus \neg x = 1$。

使用较大的逻辑符号代表对多个数字进行的操作，例如 $\bigwedge_{i=1}^{n} x_i$，$\bigvee_{i=1}^{n} x_i$ 和 $\bigoplus_{i=1}^{n} x_i$，分别代表将 $x_{1\ldots n}$ 序列中所有数字进行 **逻辑或**、**逻辑与** 和 **逻辑异或** 后得到的结果。

此外，定义信号状态 `X`，代表在实际情况下，由于对未利用数据的不确定导致的 **未知态**。一次运行中，除了输入信号外的其他信息将会全部置为 `X`，而你的答案应当无视这个影响，输出一个固定的结果。

对于 `X`，有运算 $\texttt{X} \wedge 0 = 0$ 和 $\texttt{X} \vee 1 = 1$ 成立，而其余运算均会返回 `X`。需要注意的是，在本题组中，哪怕你确定运算左右的 `X` 存在实际上的内在联系，对其进行的运算也严格遵守上述规定。比如说，你需要将两个 `X` 进行异或操作，并且已经知道两个来源于一个相同的输入信号，此时的返回结果也会是 `X` 而不是 0。

提交答案格式与限制

本题中，使用非负整数表示信号，信号与整数两两对应。

其中编号为 0 的信号的逻辑值恒定为 0，且不接受逻辑门的信息覆盖，也就是 **不能成为任何一个逻辑门的输出信号**。除此之外，可以使用在 1 和 MaxID 之间的编号。在所有任务中，$\mathrm{MaxID} = 10^5$。接下来的指令默认保证对应的编号在 $[0, \mathrm{MaxID}]$ 范围内。

注意，在此处我们无需考虑负载问题，每个信号都可以作为多个门电路或者 触发器的输入信号，但是 **只能作为一个门电路或触发器或分配指令的输出信号**。

你的答案长度不应超过 3×10^5 个字符。你可以在适当的地方加入空白字符增加可读性，但是不应当截断数字或指令名称。为了防止出现全部填入空白字符卡评测的情况，我们在读入时将空白字符计入长度中，而在解析过程中自动忽略。

下面通过分配指令简单介绍一条指令需要满足的格式。分配指令格式为：$y = x;$，其中 x 代表 **输入对应的信号编号**，y 代表 **输出对应的信号编号**。这条指令将 x 对应信号的逻辑值分配到 y 对应信号中，类似于拷贝。每条指令后都应加上分号。

你的答案需要受到最大门延迟的限制。所有逻辑门电路均视为 **一级门延迟**。一个组合门电路的门延迟，简单来说，等价于组合逻辑电路部分在拓扑序下运行所有逻辑门指令需要的最小门延迟。在所有任务中，这个值都需要小于一个固定的值，根据任务而定。

门电路

我们引入了 7 个逻辑门，作为逻辑门电路的基本单元。它们分别为 **与门**、**或门**、**异或门**、**与非门**、**或非门**、**同或门** 和 **非门**。

本题中除非门外的逻辑门允许多个输入。对于满足运算结合律的逻辑门（**与门**、**或门**、**异或门**），将会直接计算所有输入信息在运算下的值并输出。否则，需要寻找在运算取反后对应的、符合结合律的运算（与非门对应与门，或非门对应或门，同或门对应异或门），在新运算下计算输入信息的结果后取反得到真正的输出信息。

在你的答案中，可以使用类似于函数参数的形式描述一个逻辑门。下表给出具体的定义：

逻辑门	对应的指令表示形式	算式
与门	$y = \mathrm{AND}(x_1, \cdots, x_n);$	$y = \bigwedge_{i=1}^{n} x_i$
或门	$y = \mathrm{OR}(x_1, \cdots, x_n);$	$y = \bigvee_{i=1}^{n} x_i$
异或门	$y = \mathrm{XOR}(x_1, \cdots, x_n);$	$y = \bigoplus_{i=1}^{n} x_i$
与非门	$y = \mathrm{NAND}(x_1, \cdots, x_n);$	$y = \neg\left(\bigwedge_{i=1}^{n} x_i\right)$
或非门	$y = \mathrm{NOR}(x_1, \cdots, x_n);$	$y = \neg\left(\bigvee_{i=1}^{n} x_i\right)$
同或门	$y = \mathrm{NXOR}(x_1, \cdots, x_n);$	$y = \neg\left(\bigoplus_{i=1}^{n} x_i\right)$
非门	$y = \mathrm{NOT}(x);$	$y = \neg x$

指令中需要满足 $1 \leq n \leq \mathrm{MaxDegree}$。

上述指令均需要 **一级门延迟**。值得注意的是，你可以利用函数嵌套，实现逻辑门的集成。集成后该指令对应的操作在运行时只需要 **一级门延迟**。该指令应该满足如下形式：

$y = \mathrm{OP1}(\mathrm{OP2}(...), \mathrm{OP2}(...), \mathrm{OP2}(...), ...);$

该式子需要满足如下限制：

- OP2 应当为 AND、OR 和 XOR 中的一种，且处处相同。
- OP1 不能为 NOT，且不能等于 OP2。

- OP1 的参数应当全部为 OP2 运算后的结果，而不能是其余运算或输入信号。

组合门电路

组合门电路全部由门电路构成。组合门电路任一时刻的输出状态，只决定于该时刻输入信号的状态，而与输入信号作用前电路原来的状态无关。

组合门电路不存在反馈线，用更加数学化的方式描述，**组合逻辑电路部分不应当存在环**。由于每个信号只能被一个逻辑门或分配写入信息，而能被多个逻辑门或分配利用，故存在多个逻辑门或分配对一个逻辑门或分配的依赖关系，此处相当于保证依赖关系不会形成闭环。需要注意的是，包含寄存器的门电路环是被允许的。这句话涉及到时序逻辑电路的模拟方式，后面再展开叙述。在此基础上，逻辑门在有向图的拓扑序下运行。

在组合逻辑部分，你的答案应当形成一个符合上述要求的组合门电路。你可以使用多个逻辑门指令和分配指令的组合表示一个组合门电路。更加详细的要求请参考上文中对指令的描述。

非法门电路示例

`4=AND(1,OR(2,3));` 错误，在集成逻辑门指令中，参数类型并不相同。可以考虑改为 `4=AND(OR(0,1),OR(2,3))`。

`2=NOT(AND(1));` 错误，`AND(1)` 语法本身正确，但是不允许使用 `NOT(AND())` 嵌套，而应当使用 `NAND()`。

`1=AND(0,1);` 错误，存在逻辑门依赖关系的自环。

`2=AND(0,1);1=AND(0,2);` 错误，存在逻辑门依赖关系的环。

`4=5;5=6;6=4;` 错误，存在分配依赖关系的环。

`3=AND(0,1);3=AND(0,2);` 错误，同时向一个信号写入值。

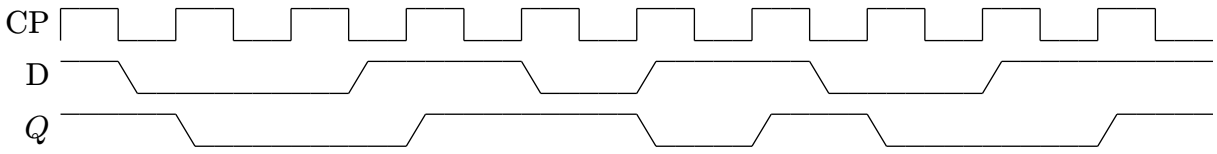
正边沿 D-触发器

为简化问题，本题仅允许使用正边沿 D-触发器，并且输入的时钟信号固定为 **全局同步**（即所有触发器的时钟信号保持完全一致）的时钟信号 CP，其 **功能表** 如下:

CP	D	Q
↑	D	D

该功能表含义是，当时钟信号 CP 从 0 跳变为 1 时（即时钟信号处于正边沿），输出信号 Q 的逻辑值会变为此时输入信号 D 的逻辑值；除此之外的任何情形，Q 总是保持不变（无论 D 发生何种变化）。

例如，假定有如下所示时钟信号 CP 和 D 信号，且 Q 初值为 1，则每次时钟信号正跳变（对应图中自下而上的竖线）时，Q 信号值变化如下图所示。



使用 $q = \mathrm{D\_FF}(d);$ 描述一个输入信号为 d，输出信号为 q 的正边沿 D-触发器。注意，本题中所有正边沿 D-触发器的输入时钟均统一为全局同步的时钟变量，由评测程序生成，无需选手生成或指定。

在实际运行中，我们会先将正边沿 D-触发器移除，然后通过拓扑排序检测是否存在环。这样的实现原则基于正边沿 D-触发器响应时间的特殊性。对于组合逻辑部分而言，正边沿 D-触发器可以认为是输入信号的一部分，只是受到组合逻辑部分电路的控制。下面给出一个带环的简单例子：

$$2 = \mathrm{D\_FF}(1);$$
$$1 = \mathrm{NOT}(2);$$

假定周期运行前正边沿 D-触发器储存的值为 1，那么在接下来的周期，编号为 2 的输出信号将会不断产生 010101010101... 的输出。

测评方式

我们为每道题目开放了一个提交窗口。你可以通过 **答案生成代码** 或 **答案文件** 提交答案。需要注意的是，子任务都是利用单测试数据实现的，而根据目前发现的已知问题，在洛谷上提交答案生成器可能会使排行榜罚时出现问题，故推荐大家 **在罚时出现异常时使用文件提交**。

评测程序使用了和模拟器一致的核心代码，并且进行了足够多次的运行模拟，保证在随机数据下的公平性。

模拟器使用规范

在下发的文件中包含 `simulator.cpp`，用于对你的答案进行模拟。经过测试，该模拟器可以在最低 C++98 的规范下通过编译，并且理论上支持多系统（在一些系统上行为可能有差异，下文将会提到）。

假定模拟器编译出的可执行文件为 `simulator.exe` 或者 `./simulator`（下面以 Windows 系统为标准，因此使用 `simulator.exe` 代表）。命令行参数如下：

```
simulator.exe <INPUT_FILE> <OUTPUT_FILE> <DEBUG_FILE>
```

三个参数均为可选参数，默认分别为 `data.in`，`data.out` 和空。运行后，模拟器将会从标准输入中读入你的答案，随后进行模拟。下面介绍各个参数。

INPUT_FILE 代表一个输入文件，模拟器将会从中读取运行限制和输入数据。

第一行包括 9 个非负整数，分别代表：（其中带星号的参数不影响组合逻辑电路）

- 测试点总数 CaseNum，代表运行总次数
- 测试点类型（组合逻辑电路填 0，时序逻辑电路填 1）
- 最大的信号编号 MaxID，需要满足 MaxID $\leq 10^5$
- 运行的周期个数 ClockLoopTime*
- 最大 D_FF 个数 DFFMaximum*
- 最大允许门延迟 Tcomb
- 每个逻辑门可以接受的最多输入个数 MaxDegree
- 输入信号个数 InSignalNum
- 输出信号个数 OutSignalNum

第二行包括若干个正整数，代表输入对应的信号编号。信号编号个数应当恰好等于 InSignalNum，且两两不同。

第三行包括若干个非负整数，代表输出对应的信号编号。信号编号个数应当恰好等于 OutSignalNum，且两两不同。

随后需要给出每次运行的输入，其中组合逻辑类型中应当按照第二行的顺序给出每次运行中所有输入的信号逻辑值；而时序逻辑部分应当按顺序给出每次运行中每个周期的输入信号逻辑值。

OUTPUT_FILE 代表一个输出文件，用于给出每次运行的输出。

对于组合逻辑电路，将会输出 CaseNum 行，每行包含若干个用空格分隔的 `01` 字符串，表示当前运行输出的逻辑值。

对于时序逻辑电路，将会输出 CaseNum 组，组间使用空行分隔。每一组包含 ClockLoopTime 行，每行包括若干个用空格分隔的 `01X` 字符串，表示当前周期 **结束时** 输出的逻辑值。

DEBUG_FILE 代表一个可选的调试文件，其输出格式和输出文件类似，但是每一行会包含 MaxID 个空格分隔的字符，表示在每次获取输出的同时，对所有信号进行采样，转化为 `01X` 三个字符并输出。

你可以使用 `CTRL+Z` 快捷键结束输入。同时，你可以选择以空行结束输入，只需要将第 13 行进行更改，就可以选择开启或者关闭这个特性：

```cpp
// Enable feature
const bool SET_NEWLINE_AS_END = true;
// Disable feature
const bool SET_NEWLINE_AS_END = false;
```

同时，为了保证 Windows 系统下的正确行为，我们将输入中的 `\r` 字符进行了过滤，方便获取一个字符所在的行和列。如果你是 MacOS 的用户，并且确认命令行或者重定向读入的文件使用 `\r` 作为回车符，则可以选择注释掉第 92 行，这一行对输入的所有 `\r` 进行了过滤。

模拟器配有一系列的事件检查，方便选手检查答案出现的问题。在答案文本解析部分会提供具体到字符位置的提示，比如：

```
1=AND(OR(1,2),OR(3,4),5);

Error: The types of parameters should be the same
- pos: 1:3 ~ 1:24
- content: 'AND(OR(1,2),OR(3,4),5)'.
```

以下介绍模拟器的实现方式，方便选手理解模拟器工作原理。

- 从输入文件获取该测试下的信息
- 从标准输入获取选手答案，并检测长度限制
- 对选手答案进行解析，获取所有涉及的逻辑门、分配和正边沿 D-触发器。
- 去除正边沿 D-触发器后，进行拓扑排序，取 **最长逻辑门链长度** 为门延迟，并确定运行顺序。
- 对每次运行（每个周期），读取输入逻辑值，然后按照顺序处理逻辑门。如果是时序逻辑电路，则将正边沿 D-触发器 **同时赋值** 后，保持输入不变并再次运行，得到真实输出。
- 内存池在运行中途不会清空。在每次运行结束后，清空内存池。

模拟器使用示例

接下来将会通过一个简单的案例说明模拟器的使用方式。

- 给定两个输入信号，编号分别为 1 和 2。请在 3 处输出两个信号逻辑值在同或后的值。规定门延迟不超过 2。

根据上述说明，可以写出如下输入文件：

```
4 0 10 0 0 2 10 2 1
1 2
3

0 0
1 0
0 1
1 1
```

这个文件指定了一个包含四次运行的模拟，其中向两个输入填入了四种情况下的逻辑值。随后就可以运行模拟器并输入答案。下面给出三种实现方式以及模拟器输出：

`3=NXOR(1,2);`	`Tcomb = 1, D_FF count = 0` `4 case(s) runned.`
`4=XOR(1,2);` `3=NOT(4);`	`Tcomb = 2, D_FF count = 0` `4 case(s) runned.`

`4=NOT(1);`
`5=NOT(2);`
`6=AND(1,2);`
`7=AND(4,5);`
`3=OR(6,7);`
	`Tcomb = 3, D_FF count = 0`
	`Error: Tcomb exceeds MAX_TCOMB.`

对第二种情况，输出文件内容为：

```
1
0
0
1
```

调试文件内容为：

```
Case 1:
  0 0 1 0 X X X X X X

Case 2:
  1 0 0 1 X X X X X X

Case 3:
  0 1 0 1 X X X X X X

Case 4:
  1 1 1 0 X X X X X X
```

还有一个时序逻辑电路的案例：

- （折半时钟信号）给出输入信号 F，代表是否处于第一个周期，在编号为 2 的输出信号中给出 10101010... 的周期性输出。此时该信号的频率实际上为全局时钟信号的一半，故得名折半时钟信号。运行 6 个周期，最大门延迟限制为 1000，最大 D_FF 个数限制为 1，只能用编号不超过 5 的信号。

可以写出输入文件：

```
1 1 5 6 1 1000 10 1 1
1
```

```
2

1 0 0 0 0 0
```

此处提供一种解答，表格中分别展示了指令、模拟器输出、输出文件和调试文件的内容。

`4=NOT(2);`
`3=OR(1,4);`
`2=D_FF(3);`
	`Tcomb = 2, D_FF count = 1`
	`1 case(s) runned.`

`1`
`0`
`1`
`0`
`1`
`0`
	`Case 1:`
	`  Loop 1: 1 1 1 0 X`
	`  Loop 2: 0 0 1 1 X`
	`  Loop 3: 0 1 0 0 X`
	`  Loop 4: 0 0 1 1 X`
	`  Loop 5: 0 1 0 0 X`
	`  Loop 6: 0 0 1 1 X`

各部分分数

任务名称	分数	依赖任务
投票器	40	
译码器	20	
选择器	20	译码器
比较器	20	
加法器	20	投票器
超前进位加法器	40	加法器
ALU	40	选择器、加法器
串行奇偶校验器	30	
移位寄存器	25	
可变模计数器	25	
乘法器	40	超前进位加法器
寄存器堆	30	选择器

奋斗四小时，手搓 CPU	50	ALU、寄存器堆

推荐做题顺序

这里推荐大家按照表格从上到下的顺序完成任务，不过实际上，**超前进位加法器** 和 **乘法器** 与整个实现流程并不存在很强的关联性，属于“支线任务”。因此，如果你实在不会这两个任务的话，不妨先跳过，去做关联性更强的任务。