Project1 Logisim 部件级实验

Logisim 允许用户使用图形用户接口设计并仿真数字电路，它自身包含一些库，库中已有诸如基础门电路，存储器、多路选择器、译码器等简单器件。在后续的实验中，你将使用这些器件搭建自己的 CPU。在本实验中，我们将在 logisim 中完成异或电路以及加法器的构建。通过本实验，达到熟悉 logisim 软件环境的目的，同时完成 CPU 数据通路的若干基础性功能部件的设计。

1. 工具使用视频

请访问 http://mooc.buaa.edu.cn，选择统一认证入口登陆。

注册课程《M_G06B2830 数字系统设计工具集》。

访问课程页面中 logisim 工具介绍相关内容。

2. logisim 安装及初步使用

请参看视频——《logisim 简介》。

2.1. 获取 logisim

在本机安装 logisim：

1. logisim 需要 Java 5 或者更高版本的支持，如果本机未安装 Java 5，请通过 java.sun.com 下载；
2. 从 http://sourceforge.net/projects/circuit/下载：
   - .jar 文件——可运行于任意平台
   - MacOS .tar.gz 文件
   - Windows .exe 文件
3. 运行程序
   - 在 windows 以及 MacOS 系统上，可双击 JAR 文件；在 linux 或者 Solaris 系统上，可以通过命令行，键入 "java -jar logisim-XX.jar"
   - 对于 MacOS 版本，下载的.tar.gz 版本是未经压缩的，只需要双击 logisim 的标识进行启动，可以将标识放入应用程序文件夹
   - 对于 Windows 版本，双击 logisim 标识，可以在启动菜单或者桌面创建快捷启动。

2.2. The beginner's guide

可以从 logisim 提供的指南 http://www.cburch.com/logisim/docs/2.7/en/html/guide/tutorial/index.html 开始学习。

2.3. 搭建异或电路

请参看视频——《初学示例》《库及属性》，认识 logisim，完成异或电路的搭建。

2.4. 构建 8-bit Adder

完成 8 位加法器的构建

2.4.1. 预备知识

定义，参看 http://en.wikipedia.org/wiki/Adder_(electronics)

1-bit 全加器，将三个 1-bit 数进行相加，输入为 A，B 以及 C_{in}，其中 A，B 为操作数，C_{in} 为进位位。全加器以级联的方式构建，可形成 8/16/32 /64 位加法器。

对于 8-bit 加法器，可采用如下描述：

```
Add8:    C = A + B + Cin; V = overflow
Inputs:  A[8], B[8], Cin
Outputs: C[8], V
```

其中：A[8]指示输入名为 input 的 8 位宽的输入管脚，在 logisim 中创建一个单一的输入，并将 data bits 属性更改为 8。属性在 logisim 窗口的左下方。输出 C 由 A，B 以及 C_{in} 来计算。其中，A,B,C 为 2 的补码。如果溢出，则对 V 进行置位。

2.4.2. 子电路 sub-circuits

请参看视频——《子电路》

使用子电路通过构建 1-bit 加法器，再构建 4-bit 加法器，再到 8-bit 加法器，可以使得连线更为简单。在 logisim 中使用子电路类似于编程时编写函数再多次调用。

构建新的电路，选择工具栏中"Project->Add Circuit..."。

如何在电路 B 中使用子电路 A？在 logsim 的资源管理器中双击电路 B，将会有放大镜的图标显示在电路 B 的标识上，之后电路 B 将在 logisim 的右侧画布中显示。之后在资源管理器中点击电路 A，便可在画布的任意位置放置电路 A 的实例，此时即成为电路 B 的子电路。之后任意时刻对电路 A 的更新，将会反映到电路 B 中所有电路 A 的实例中。

2.4.3. 真值表、布尔表达式、卡诺图

真值表用于反映逻辑电路的输出如何相应于输入的组合进行变化。如，对于所有的输入为 0，则输出为 0。完整的真值表如下：

Inputs			Outputs	
A	***B***	C_{in}	C_{out}	***S***
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1

上述真值表可以用如下布尔表达式：

$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = (A \cdot B) + (C_{in} \cdot (A \oplus B))$$

卡诺图可用于进行表达式化简，参见 http://en.wikipedia.org/wiki/Karnaugh_map

2.4.4. 构建 1-bit 全加器

请参看视频——《组合逻辑分析》，在 logisim 中构建 1-bit 全加器：

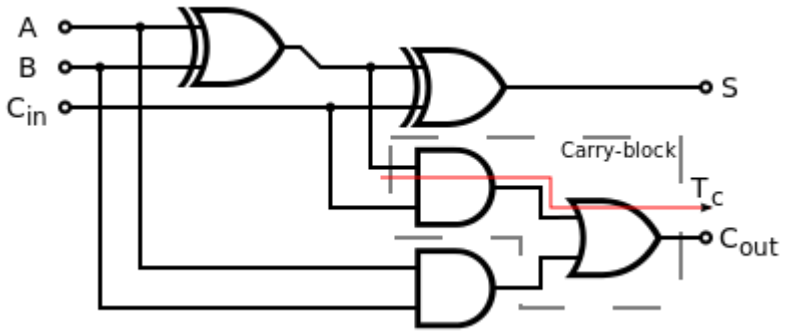


图 1 1-bit 全加器示例

2.4.5. 构建 4-bit 全加器

4-bit 全加器是简单的 4 个 1-bit 加法器的级联，将一个全加器的 carry-out 作为另一个全加器的 carry-in。

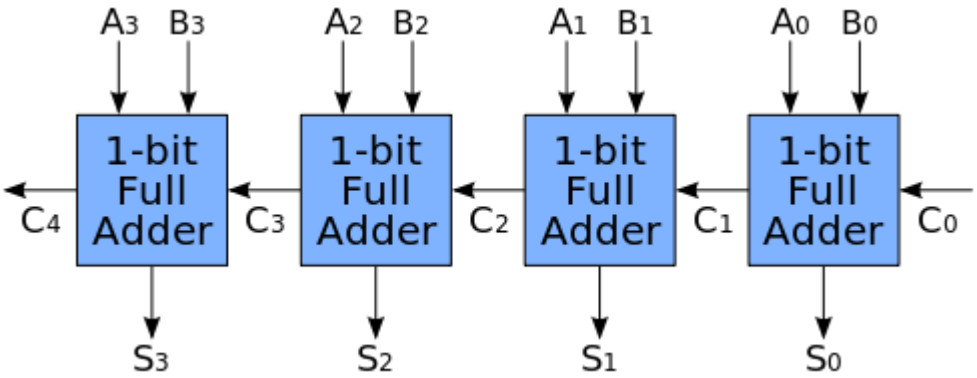


图 2 4-bit 全加器示例

请在 logsim 中构建 4-bit 全加器，要求使用 section 1.4.4 中构建的 1-bit 全加器作为其子电路。

2.5. 提示

2.5.1. 如何构建 1-bit 全加器

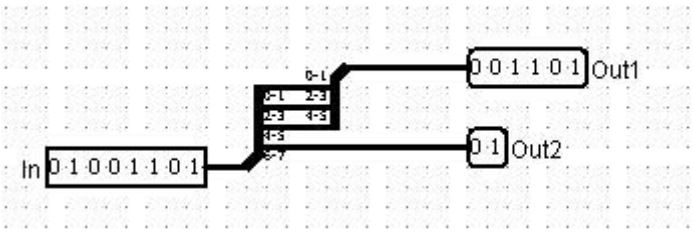
Logisim 可以通过编辑真值表生成相应的逻辑电路。学习如何编辑真值表 http://www.cburch.com/logisim/docs/2.6.0/en/guide/analyze/table.html 以及创建相应的电路 http://www.cburch.com/logisim/docs/2.6.0/en/guide/analyze/gen.html：

- 在画布中放置所需要的输入，输出节点
- 将更改进行保存
- 右键资源管理器中表示当前画布器件的符号，选择'Analyze Circuit' 以弹出 'Combinantional Analysis' 窗口
- 编辑真值表，并创建相应的电路

2.5.2. 使用 splitters

logisim 在基础库中提供 splitter（分裂器）器件，该器件在多位值与多个子序列值之间创建相应关系。同时，该器件可作为 bundler（捆束器）使用——将多个单一的值连接为一个多位的值。

在下图中，使用 Splitter 可以将 8-bit 线分为 2-bit 线以及 6-bit 线：



Splitter 属性包括 fan out 以及 bit width in，前者表明 splitter 扇出数量，后者控制分裂的数据宽度。

在 4-bit 全加器中可以帮助你有效管理输入/输出值，更多关于 splitter 的介绍请参看视频——《wire bundles》。

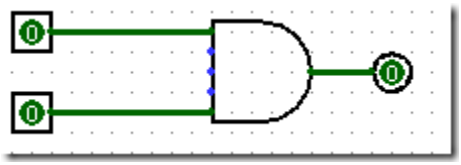
2.5.3. 使用 probes

在进行电路调试时，使用 Probes（探针）可以有效帮助调试。可以将探针与输入/输出总线相连，并设置使用十进制/十六进制/二进制显示值。

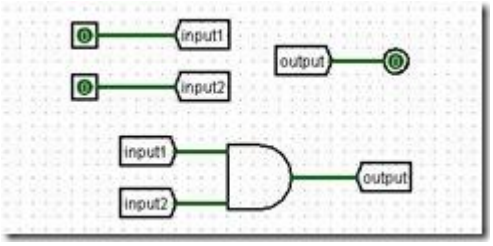
2.5.4. 使用 tunnels

在本实验中暂不需要使用 tunnels，但当电路规模变大时，tunnel 将帮助组织电路使之更为清晰。tunnel 的作用类似于 wire 的作用，将标记相同标签的点进行绑定。

对于与门电路,在 logisim 中可以如下进行构建:



可以使用 tunnel 工具代替复杂的连线，使得在后续复杂设计时，从蜘蛛网一样的连线中解脱，上面的电路可以使用 tunnel 进行构建为：



图中输入、与门模块、输出三部分结构将很清晰。

2.6. 构建 8-bit 全加器

请使用 section 1.4.5 构建的 4-bit 全加器作为子电路搭建 8-bit 全加器。

2.7. 注意事项

- 将完成的作保存为.circ 文件，并提交至 course 平台。
- 请尽量使用标准模块，如果需要建立自己的模块，请学习 logisim 中内建模块的设计风格。
- 请合理划分电路层次结构，设计时清晰定义输入输出端口，尽量使定义

的风格和规范保持一致。

- 对于复杂电路，请使用网络标识代替直接连线，会使你的原理图更加清晰易读。
- 为每个电路模块设计相应测试用例。

3. 使用寄存器及子电路构建电路方法实现循环累加器

练习内容：子电路

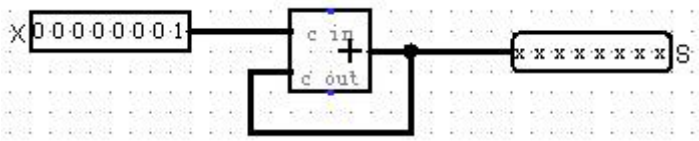
需要实现一个循环累加器：

- 初始状态，S = 0;
- 随着时间推移，for X_1,X_2,X_3

$$S = S + X_i$$

3.1. 如何控制 for 循环中的下一次迭代？

请参看视频——《循环累加电路》

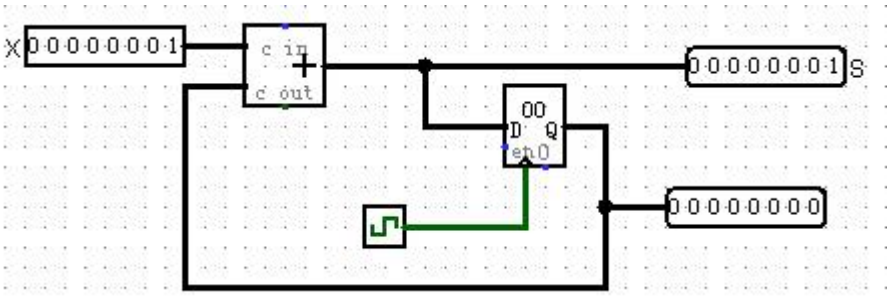


将 S 值反馈至输入，并不能推进 for 循环进入下一个迭代。

3.2. 状态器件

一些器件可以用于在一段时间内存储值：如寄存器文件以及存储器。

这些器件可以帮助在组合逻辑块间控制信息流。

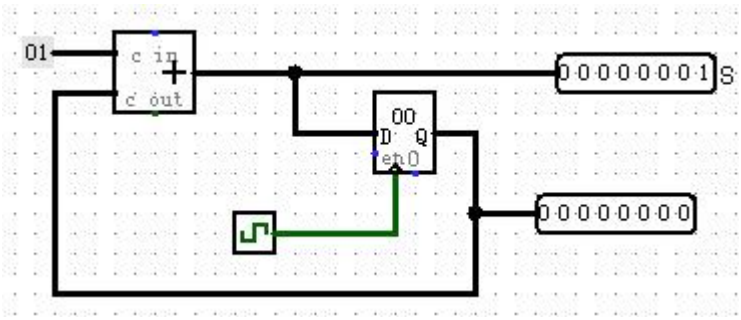


在本例中，寄存器用于持有传输给加法器的数据。

可见，硬件系统由无状态的组合逻辑和含有状态信息的存储逻辑构成。

3.3. 循环累加电路

- 使用 logisim/Arithemetic 中的 Adder，构建自加 1 的无限累加器。
- 创建新文件，点击 Project-->Add Circuit，并命名。
- 加载 Arithmetic library， Project-->Load Library-->Built-in Library... and select "Arithmetic"，从库中选择 adder 子电路
- 加载 Memory library，Project-->Load Library-->Built-in Library... and select "Memory"，从库中选择 register。
- 将 clock（位于 Wiring 文件夹）与 register 连接
- 依据 section2.2 的设计，将 register、adder 相连，此时会出现"Incompatible widths" error，这意味着连线试图将两个不同位宽的管脚相连，点击 register 将在资源管理器的下方看到 register 的属性列表，其中 "Data Bit Width"域控制 register 相加的位宽属性，将其更改为 8。
- 将 8-bit 常量 1（Wiring 库中）连线至 adder 的第二个输入
- 向电路添加两个输出管脚，方便监控电路。
- 最终电路呈现为：



3.4. 电路测试

- 双击 circuit 浏览器中 main 子电路，回到 main 子电路
- 单击 AddMachine 以选择，并放置至 main 电路中
- 可以更改“Facing”属性以调整器件方向
- 将输出管脚与 AddMachine 子电路连接，输出管脚自上而下、自左而右排列，放置鼠标至子电路输出位置可以看到相应的标签值
- 如果需要访问 AddMachine 子电路的内在状态，一种方式在子电路上右键，选择"View AddMachine"，另一种方式，使用 poke 工具，再双击子电路。
- 将寄存器的值初始化为 1，通过使用 poke 工具点击 register
- Simulate-->Go Out To State-->main，返回 main 子电路
- Simulate-->Ticks Enabled，开始进行仿真

- 如果需要电路以更高的频率运行，可以通过 Simulate-->Tick Frequency 选择更高的频率

4. 设计 ALU

你将在 logisim 中构建部分 MIP32 体系结构组成，该实验的目的在于设计小的特定目的的电路以进一步构建较为复杂的、通用 CPU。在 Logisim 相关的实验最后，你将搭建一个 32-bit 的 MIPS CPU，该 CPU 不包含诸如协处理器指令以及 traps/exceptions 特性。

本实验构建简单的 MIPS ALU（算术逻辑单元），该运算单元执行由汇编语言指示的核心计算操作，在 section 2.4 中，你已经构建了 32-bit adder 的部分电路。在这里你需要完善其功能实现减法，请注意此处不能使用 logisim 内置的 Adder 及 Subtractor，加减法器的实现须以门电路为基础。

在之后的实验中，你将会用到 logisim 库中的门器件、存储器件以及多路选择器、译码器和其他简单的器件来搭建最终的 CPU。

4.1. ALU 模块定义

4.1.1. 基本描述

ALU 的主要功能是对输入到 ALU 的两个数进行加法、减法、按位或等操作。ALU 内部包括 32 位加法器、减法器等部件。ALU 除了以上操作外，还需要判断输入的两个值是否相等。

4.1.2. 模块接口

信号名	方向	描述
inputA	I	参与 ALU 计算的第一个值
inputB	I	参与 ALU 计算的第二个值
ALUOp[1:0]	I	ALU 功能的选择信号。 00：ALU 进行加法运算 01：ALU 进行减法运算 10：ALU 进行或运算
result	O	ALU 计算的结果
zero	O	1：result 为 0

		0：result 不为 0

4.1.3. 功能定义

序号	功能名称	功能描述
1	加法运算	将 inputA 与 inputB 进行加法运算。result←inputA+inputB
2	减法运算	将 inputA 与 inputB 进行减法运算。result←inputA-inputB
3	或运算	将 inputA 与 inputB 进行按位或运算。result←inputA|inputB

4.2. 使用的器件库

- wiring（pins, splitters, probes, tunnels 等）
- base(wires, text 等)
- gates
- plexers

4.3. 工作流程

4.3.1. 构建加法功能

参看 section 2.4，完成 32-bit 加法器的构建。

4.3.2. 构建减法功能

A-B=A+[-B]，因而需要先构建一个求[-B]的子电路。

4.3.3. 按位或操作

可使用 splitter。

4.3.4. 基于上述子电路构成 ALU

两个输入值经 ALU 运算的结果通过 MUX 依据 ALUOp[1:0]选择运算功能。

4.3.5. 测试 ALU 子电路

使用多种输入，完整覆盖 ALU 各项功能，并测试正确性。

5. 设计 GPR 模块

5.1. GPR 模块定义

5.1.1. 基本描述

GPR 以 32 个 32 位具有写使能的寄存器为基础，辅以多路选择器。其主要功能是对寄存器堆进行存值取值的操作。为了便于测试，GPR 除了以上操作外，可有清零的操作，即将寄存器的值变为 0x00000000。

5.1.2. 模块接口

信号名	方向	描述
clk	I	时钟信号。
reset	I	复位信号。 1：复位 0：无效
WE	I	写使能信号。 1：可向 GPR 写入数据 0：只可从 GPR 读出数据
A1	I	5 位的地址输入，用于指定 32 个寄存器中的一个。 在单周期 CPU 中为指令的[21:25]即 rs 字段
A2	I	5 位的地址输入，用于指定 32 个寄存器中的一个。 在单周期 CPU 中为指令的[16:20]即 rt 字段
A3	I	5 位的地址输入，用于指定 32 个寄存器中的一个。 在单周期 CPU 中为指令的[16:20]rt 字段或[11:15]rd 字段之一
WD	I	32 位的数据输入，当写使能时，可以向寄存器中写入。
RD1	O	输出 32 位的 A1 指定的寄存器的数据。
RD2	O	输出 32 位的 A2 指定的寄存器的数据。

5.1.3. 功能定义

序号	功能名称	功能描述

1	复位	当复位信号有效时，32 个寄存器中的值被置为'h0。
2	读数据	根据 5 位地址输入确定寄存器，并将寄存器中的数据输出。
3	写数据	当写使能时，根据 5 位地址输入确定寄存器，并将数据写入寄存器。

5.2. 工作流程

5.2.1. 熟悉 memory/register 器件

了解各输入及输出管脚的作用

5.2.2. 将 32 个寄存器进行矩阵排列

你可以选择使用 8 行 4 列的方式，这只是为了方便布局。

5.2.3. 将时钟信号与 32 个寄存器连接

5.2.4. 将写入数据与 32 个寄存器相应端口连接

5.2.5. 将 reset 信号与 32 个寄存器相应端口连接

5.2.6. 使用 mux 进行输出数据选择

思考选用器件，选择 32 个 32bit 寄存器读出数据，作为 RD1，RD2。

//使用 A1 作为 MUX 选择信号以输出 RD1；

//使用 A2 作为 MUX 选择信号以输出 RD2。

5.2.7. 使用 plexers/decoder 进行写入数据寄存器选择

思考选用器件，选中 32 个寄存器中相应于 A3 的寄存器进行写入。

//使用 A3 作为 decoder 的选择信号，decoder 的输出连接至寄存器的使能信号。

5.3. 测试 GPR 子电路

5.3.1. 读数据测试

首先对每个寄存器预置不同的值（例如，将每个寄存器的值预置为其编号），然后分别将几组不同的值赋给 A1 和 A2，同时驱动时钟，观察 R1 和 R2 的输出是否与对应寄存器的值相同。若相同，则说明 GPR 的读数据功能正常。

5.3.2. 写数据测试

首先，将 WE 置为 0，将 WD 预置值为 X，将几组不同的值赋给 A3，同时驱动时钟，观察寄存器的值是否发生变化。若发生变化，则说明写使能功能不正常。

然后，将 WE 置为 1，将 WD 预置值为 X，将几组不同的值赋给 A3，同时驱动时钟，观察对应 A3 值得寄存器的值是否会变成 X。若是，则说明 GPR 的写数据功能正常。

5.3.3. 清零测试

在所有寄存器已预置为非零值的情况下，将 reset 置为 1，观察所有寄存器的值是否变为 0。若是，则说明 GPR 的清零功能正常。