

Chipmunk Introduction and Mini-Manual

Norman Matloff
University of California, Davis
©1995-2003, N. Matloff

October 29, 2003

Contents

1 Overview	4
2 Startup	4
3 Chipmunk Terms	4
4 Chipmunk Modes	4
4.1 ROT, MIR-, MIR—, CNFG	4
4.2 Probe Mode	5
4.3 Glow Mode	5
5 Try It!	5
6 General Features of Chipmunk	6
6.1 Component Catalogs	6
6.1.1 General Symbols and Terminology	6
6.2 Input/Output Mechanisms	7
6.2.1 Digital Switch	7
6.2.2 Digital Pulse-Generative Switch	7
6.2.3 Digital Hexadecimal Keypad	7
6.2.4 LED	7
6.2.5 Hexadecimal Display	7
6.2.6 Tri-State Devices	7

6.3	Connection Wires	8
6.4	Some Less-Clear Components	8
6.4.1	HELP Button	8
6.4.2	SHIFT	9
6.4.3	7482/7483 Adders	9
6.4.4	74157 and 74153 Multiplexors	9
6.4.5	7485 Comparator	9
6.4.6	74175A 4-Bit Register	10
6.4.7	SRAM8K	10
6.4.8	TNEG, DPOS Etc. Flip-Flops	12
6.4.9	A_TRIBUF Tri-State Buffer	12
6.4.10	TO/FROM (Invisible Wires)	12
6.4.11	74150 MUX	12
6.4.12	74138 3-to-8 and 74139 2-to-4 Decoders	13
6.5	Editing Your Circuit	13
6.5.1	Rotation	13
6.5.2	Moving	13
6.5.3	Copying	13
6.5.4	Deleting	13
6.5.5	Labels	14
6.5.6	Resizing, Scrolling and Pages	14
6.5.7	Saving/Loading Circuits	14
6.6	Modular Design	14
6.6.1	Creating a Module	14
6.6.2	Using a Module	15
6.6.3	Multimodule Pages and Files	16
6.6.4	Restrictions	16
6.7	Miscellaneous	17
7	Examples	17
7.1	“If-Then-Else”	17
7.2	ModDiv5	17

7.3	SRAM8K	18
7.4	Fetch-and-Add	19

1 Overview

Chipmunk is an analog/digital-circuit simulator written by David Gillespie of Cal Tech and John Lazzaro of UC Berkeley. They actually refer to it simply as log. The term “Chipmunk” stems from the fact that the package is distributed via from a directory of that name at the Chipmunk homepage, <http://www.cs.berkeley.edu/~lazzaro/chipmunk/>.

The package is freely copyable, i.e. public-domain. It should run on virtually any Unix X11 system, and has a Windows version too. The document here is intended as a fairly thorough introduction to the major features of the digital section of Chipmunk. I have set up some simplified installation instructions.

2 Startup

Make sure the **bin** subdirectory of the Chipmunk directory is in your search path, and then type

```
diglog
```

Two windows will be created, one labeled **mylib** at the top edge, and the other labeled **newcrt**. You will be drawing circuits in the first of these, while the second will be used for things such as text input.

A nice convenience is that in most cases, when input is asked for in the second window, you can type it even if the mouse is still in the first window.¹

3 Chipmunk Terms

The Chipmunk documentation (and ours here) frequently refers to the terms **tap** and **drag**. These are what you would expect them to be, but for completeness they are defined here: tap means to click the left mouse button; drag means to click the left mouse button, without releasing it, and drag the mouse pointer to whatever location is desired.

The documentation also loosely (and wrongly) refers to any object as a “gate.” MSI components, e.g. multiplexers, are considered gates.

4 Chipmunk Modes

4.1 ROT, MIR-, MIR—, CNFG

The documentation also speaks of the current **mode** you have for Chipmunk. Look below the field labeled Misc at the bottom right of the main screen. It will say ROT or MIR- or MIR— or CNFG. You can choose among these by repeatedly tapping on that field. For instance, if it says ROT and you tap ROT, it becomes MIR-. The first three modes are for editing, e.g. rotating a component in ROT mode. CNFG mode is for

¹A less pleasant side effect of that is that the mouse pointer won’t actually change focus to the second window unless you move it there. Once you gain experience in Chipmunk, you can “type blindly” in this way, but it may seem strange at first.

the purpose of getting input from you as to which parameter values you want for certain complex Chipmunk components.

4.2 Probe Mode

You will find the Probe mode especially handy. Just hit the ‘.’ key to enter this mode, and hit the key again when you wish to exit.

The purpose of Probe mode is to quickly and conveniently check the value on a line. While in this mode, whenever you move the mouse pointer near a wire, the 1-or-0 value on that line will be displayed, in the form of a bar or absence of the bar (on color monitors, the bar will be red). The position of the bar will be above the Frills and Editing portion at the lower left of the **mylib** screen (and also lower right).

You should stay in Probe mode only temporarily, because some functions cannot be executed while in that mode. To leave Probe mode, simply select Probe again.

4.3 Glow Mode

This is essentially an extension of Probe mode.

Hit g to enter Glow mode, and hit g again to turn it off. This mode shows wires currently having a value of 1 in red, and makes the ones currently having 0 less visible, in black.² This would help you trace the currently active lines.

5 Try It!

Do the following to get a quick introduction to Chipmunk:

- Find the NOT gate in the menu at the bottom of the **mylib** window. Drag the gate to some place in the main part of the window.
- Find the Digital Switch component in the menu, the pointed rectangle just to the right of Editing and CAT. Drag the Digital Switch so that the point of it just touches the input of the NOT gate.
- Find the LED component in the menu, the square-with-a-circle just to the right of the Digital Switch. Drag the Digital Switch so that it just touches the output of the NOT gate.
- The Digital Switch acts as a source of 0 or 1, and the LED acts as a sensor of 0 or 1. They use a red light for 1, no light for 0.³ Since the Digital Switch is initially 0, the LED should initially be 1 (due to the NOT operation). Tap the Digital Switch a few times, making it change back and forth between 0 and 1, and note the corresponding changes in the LED.

²Lines which are disconnected from the rest of the circuit, e.g. outputs of tri-state devices which are not currently enabled, retain their original green color.

³Chipmunk is designed for color monitors, but is quite usable with monochrome monitors too, in which case the red light here becomes white. To use Chipmunk on a monochrome monitor, before starting add a variable for your X-server, using the X11 **xrdb** command:

```
xrdb -merge mylib.color: black_and_white ctrl-d
```

- Exit Chipmunk. To do this, drag on Misc, producing a subwindow menu, releasing the mouse when the mouse pointer points to Exit. (Or just type Q.) Some questions will then appear in the **newcrt** window, asking whether you want to save the circuit in the **mylib** window; answer no, and leave.

6 General Features of Chipmunk

6.1 Component Catalogs

Chipmunk features a three-tiered component database:

- Tier 1: This consists of the menu at the bottom of the **mylib** window.
To choose a component, simply drag it to the desired position.
- Tier 2: To reach this tier, tap CAT in the **mylib** window. That window will then temporarily display a number of symbols of components. When you move the mouse to a symbol (just move, no clicking), a description of that component, e.g. “Digital pulse-generative switch,” will appear at the bottom of the window.
To choose a component, drag the symbol. The **mylib** window will revert to its original state, i.e. to the space where you are drawing a circuit, and you can release the symbol wherever you want there. (In fact, you can even release it in the Tier 1 menu at the bottom of the screen, replacing one of the symbols there. This is useful if you think you will be using this component frequently.)
To leave Tier 2 if you have not chosen a component there, simply tap a point in the blank area of the screen.
- Tier 3: Tap CAT to get to Tier 2, then LIBR for Tier 3. A list of gate names will then appear in the **newcrt** window. This is a rather long list, occupying several screens; as mentioned at the bottom of the window, use + and - to get from one portion of the list to another. When you move the mouse to a gate name (just move, no clicking), a description of that component, e.g. “Generic 3-input AND gate,” will appear at the bottom of the window.
To choose a component, tap its name. Then exit Tier 3, by hitting the space bar. The component will now appear in Tier 2.

6.1.1 General Symbols and Terminology

- A > on a component’s pin means that this is an input pin for a clock signal.
- A bubble on a component’s pin means that this input is active-low, i.e. 0 means yes, 1 means no.
- A pin is said to be **floated** if it is not connected to the rest of the circuit. For an output pin, this means the value on the pin will have no effect on the rest of the circuit, even if a physical connection exists.

6.2 Input/Output Mechanisms

6.2.1 Digital Switch

The Digital Switch is a source of either 0 or 1, and is initialized to 0. Each time you tap it, it toggles, first to a 1, then a 0, then a 1 again, and so on. A 1 appears as a red-filled circle on color monitors, and as a white-filled circle in monochrome; a 0 appears as an unfilled circle.

6.2.2 Digital Pulse-Generative Switch

Like a Digital Switch, but sends 1's only, and only when you tap it (try to tap it in or near the circle). This is good as a substitute for a clock, when you are testing a circuit. To verify that the pulse is reaching a given point, attach an EDGE component (Tier 3) at that point; it will light up when a pulse comes, and not go out again until you tap it.

6.2.3 Digital Hexadecimal Keypad

This is a convenient way to replace four Digital Switches. Just tap the value you want, and from that point onward, the four output pins will contain the 4-bit representation of the hex digit you tapped (least significant bit next to the F).

The pin on the bottom is a strobe. It will emit a brief pulse each time you tap a digit in the keypad. This is useful, for example, if you are using the keypad to load into a clocked device, say a register. The keypad outputs would be connected to the register data inputs, and the keypad strobe would be connected to the register clock input.

6.2.4 LED

The LED acts as a sensor of 0 or 1. It displays the 0 or 1 in the same manner as for the Digital Switch above.

6.2.5 Hexadecimal Display

This is the output analog of the Digital Hexadecimal Keypad, conveniently replacing four LED's.

6.2.6 Tri-State Devices

Tri-state devices are used for connecting a given entity to, or electrically isolating the entity from, other entities.

The typical usage is to connect registers to buses. Say we have several registers which can output to the same bus. At any given time, we want the value in only one (actually, at most one) of the registers to be copied to the bus, which in turn means that we need a way to electrically isolate the other registers from the bus. We can accomplish the latter by placing tri-state devices between the registers and the bus, and connecting the control pins of those devices to the register-selection digital logic.

For example, the Chipmunk component catalog includes the 74374 device, an 8-bit register which includes its own tri-state devices incorporated into the chip.⁴ The left-side pins are the data inputs to the register, the bottom pin is the clock, and the right-side pins are the outputs, i.e. they contain the value stored in the register.

What makes the 74374 differ from an ordinary register, though, is the pin at the top. If a value of 1 is input to that pin, the output pins will be floated.

Suppose that we are building a CPU which includes a register file of 16 8-bit registers, with output to a bus. We could use 16 74374's, with their tri-state pins connected as follows. Elsewhere in the CPU there would be four lines indicating which register is to be copied onto the bus (typically this would be a 4-bit field within the instruction). We would feed these four lines into a 4-to-16 decoder, whose 16 outputs (negated) would then be attached to the tri-state control pins of the 16 74374's.

For more flexibility, though, we may need to use ordinary registers, without internal tri-state devices, in conjunction with separate, external tri-state devices. Component 74126 provides this capability for individual lines, whereas the 74244 does this for two 4-bit quantities (which could be identical). For the 74126, the left pin is the input and the top pin is control (1 to connect the input to the output, 0 to "float").

The 74245 component provides bidirectional tri-state connections, very useful for buses. If the bottom pin is held high, then all pins on the two sides are floated. If the bottom pin is held low, then the top pin controls the direction of current—left to right if low, right to left if high.

6.3 Connection Wires

You can draw connection wires, which "turn corners," etc.

Say for instance you want to draw an L-shaped wire segment. Tap at the point you want the top of the L, then tap at the place you want the vertex of the L, then tap where you want the right end of the bottom leg to be, then finally right-click to finish.

If the line you are drawing crosses an existing line, they will be considered to not touch each other. If on the other hand they form a T shape, then they will be considered to touch, and a dark circle will appear there to indicate this.

If a line becomes dashed, even temporarily, you have accidentally made one line coincident to another, and should remove one of the lines. Similarly, if you suddenly see some lines flashing (or displaying) red, you have probably created a short circuit; delete the last line or component you drew.

As you will see as you start using Chipmunk, it is fairly good about automatically aligning endpoints of lines.

6.4 Some Less-Clear Components

6.4.1 HELP Button

There is brief documentation obtainable via Chipmunk on-line (HELP button in the **mylib** window). The information is fed through the Unix **more** command, enabling you to move back and forth through: Space

⁴These 74* numbers are standards from TTL catalogs.

bar to go to the next page, / to search forward for a phrase,⁵ ? to search backward, b to go back a page, q to quit, etc.

For some components, more information is available by tapping the component while in CNFG mode.

The details for some major components will be clarified in the next few subsections. **Note, however, that in many cases you will need to experiment with a component in order to determine what each of the pins do.**

6.4.2 SHIFT

The SHIFT component implements a four-bit shift register. Starting with the D pin and then going counter-clockwise, here are the roles of the various pins: D is for serial data input, the main type of input used in shift registers; next are four pins which are used for parallel input; next is the clock input; the next four pins are the contents of the register; finally, the M pin controls whether the loading of the register at the clock pulse is done from D ($M = 0$) or from the parallel inputs ($M = 1$).

6.4.3 7482/7483 Adders

The adder components (2-bit and 4-bit) are numbers 7482 and 7483 in LIBR. The least-significant bit is at the top left, carry-out at the bottom right. Make sure to specify the carry-in at the top (0 if you are not using it); do not leave it floating.

6.4.4 74157 and 74153 Multiplexors

MUXs in Chipmunk are called “data selectors.”

For example, the 74157 will choose one of two 4-bit inputs (pins on the left edge), outputting the one chosen to the four pins on the right edge. Make the right-hand pin at the bottom edge low, and use the left-hand pin for the MUX control (0 selects the upper input, 1 the lower).

The 74153 selects one of 4 1-bit inputs on the left and copies it to the output on the right (as long as the top input is kept low or unconnected). The four inputs are numbered 0-3, starting at the top. The control inputs are the 2 pins on the bottom, the left of which is the more significant bit.

6.4.5 7485 Comparator

This compares the two four-bit numbers input to the bottom pins (MSBs are the leftmost pins). The three pins coming out of the left edge indicate whether the first number is greater than, equal to or less than the second, respectively.

⁵Note that the phrase is case-sensitive.

6.4.6 74175A 4-Bit Register

The input is on the left, output on the right, with the least significant bits uppermost in both cases. The bottom input is the clock, and the top is an asynchronous clear, active-low.

6.4.7 SRAM8K

This is a static RAM chip which stores 8K 8-bit words. Here are how the pins are set up:

- The address pins are on the left, with the least-significant bit on the top.
Be sure not to leave any of the address pins floating, as they will be taken to have the value 1 in that case. If they are 0s, connect them to real 0s.
- The data pins are on the bottom, with the least-significant bit on the right.⁶
- The pin labeled R is Read, active high.
- The pin labeled with a clock symbol is actually CE, the Chip Enable, active low. It does not have to be tied to a clock pulse, though it may be in some applications.
- The OE pin, Output Enable, is active low. It must be 0 for a read (to enable internal tri-state connections to the data pins), and any value for a write.
- To do a read, set R high and CE and OE low. The value to be read will appear on the data pins.⁷
- To do a write, set R and CE low, and put the value to be written on the data pins. The value will be immediately stored into the RAM.

SRAM8K is an example of a Chipmunk component which can be “configured”; Chipmunk allows you to tailor parameters of such components for your application. In the case of SRAM8K, this means for instance that you can conveniently load a set of initial values into a SRAM8K instance from a disk file.

To configure an SRAM8K, enter CNFG mode.⁸ Tap the SRAM8K, and go to the **newert** window, where you will see a menu of SRAM8K parameters.

You can move from one menu item to another via the up- and down-arrow keys, and you can set items to your liking by either using the right- and left-arrow keys (to cycle through choices) or by typing in your choice (e.g. in the case of numeric items). Once you have the choice you want in a given item, you can tell Chipmunk to use it by using an up- or down-arrow key to leave it. (Note: Do not hit the Enter key, as that will simply change the item back to the default value.)

The first menu item is Mode, with choices Read-only, Deposit and Deposit Next. The first of these allows you to examine any location within the SRAM8K; the second allows you to enter a value into any location; and the third is an extension of the second, allowing you to sequentially step through consecutive memory locations, depositing values in each location in turn. Two Address and two Data items in the menu allow you to given addresses and data in either hex or decimal.

⁶For these pins, float seems to be treated as 0.

⁷Be careful not to connect any values, e.g. a Digital Pulse Generative Switch, to the data pins when R is high; a short circuit would result.

⁸See Sec. 4.1.

For example, suppose we wish to put the value 15 (decimal) into location 3 of the chip. We would first get into CNFG mode and then tap the interior of the SRAM8K chip, as mentioned above. A menu would then appear in the **newcrt** window, with the first menu item, Mode, highlighted. If Mode is not already Deposit, we would change to it using the right- and left-arrow keys, and then use the down-arrow key to get to the Address menu item, say the hex version. We would then type 3, and then hit down-arrow again (which will make the decimal version of the item now display 3 too). We would hit the down-arrow key again until we get to Data (decimal), and type 15. To make the latter official, we now hit the down-arrow key again. That brings us to a question about saving the contents of the SRAM8K to a file. Suppose we don't want to do so. If the item says Yes, we use the right- and left-arrow keys to make it say No. Then we leave the menu, by tapping anywhere in the menu window.

To test that this worked, i.e. that location 3 really does contain 15, we could put the value 1 on the top two address pins of the chip, put 0s on the bottom 11 address pins, put 0 on CE, 1 on R, 0 on OE, and LEDs on the rightmost four data pins.

Of particular interest are the menu items dealing with disk files, which allow you to save the current contents of the SRAM8K into a file, or load previously-saved contents. To save to a file, set the "Save in circuit file" item to Yes, and type the file name in the indicated item; a suffix of **.ram** for the file name is recommended (and is used if you supply a suffixless name). Once you have typed in the name, tap it. The name will then disappear from the screen, but the file will indeed be written at that time.

By the way, the file format is straightforward ASCII. Each line lists the contents of 16 bytes in the file. The label at the beginning of a line shows the address of the first byte in that line, so that for example the label 0030 means that this line displays the contents of bytes 0x0030, 0x0031, ..., 0x003f.

For example, I placed the values 3i in locations i, i = 0,1,2,3,4,5,6 and the value 4 in location 0x100, and then saved the memory contents to a file **3.ram**. Here are the first few lines which were then created in that file:

```
0000:000306090C0F12000000000000000000
0010:00000000000000000000000000000000
0020:00000000000000000000000000000000
0030:00000000000000000000000000000000
0040:00000000000000000000000000000000
0050:00000000000000000000000000000000
0060:00000000000000000000000000000000
0070:00000000000000000000000000000000
0080:00000000000000000000000000000000
0090:00000000000000000000000000000000
00A0:00000000000000000000000000000000
00B0:00000000000000000000000000000000
00C0:00000000000000000000000000000000
00D0:00000000000000000000000000000000
00E0:00000000000000000000000000000000
00F0:00000000000000000000000000000000
0100:04000000000000000000000000000000
0110:00000000000000000000000000000000
```

Note: If you write some values into an SRAM8K in a circuit, and then you save the circuit to a file, those values you wrote do not get saved.⁹

There is a more advanced example of using SRAM8K later in this mini-manual.

6.4.8 TNEG, DPOS Etc. Flip-Flops

TNEG is a T (“toggle”) flip-flop. If the T input is 1, then each pulse of the clock changes the stored value Q from 0 to 1 and vice versa. In essence, it is a mod 2 counter. (The word “negative” here means that the input is accepted during the falling edge of the clock pulse.)

You can use TNEG without connecting anything to the top and bottom pins. These have asynchronous Set and Clear functions, meaning that you can use them to set (to 1) or clear (to 0) the value of Q at any time, not just when the clock input is pulsed (i.e. **asynchronously**). If you wish to use these functions, keep in mind that they are active-low. So, for example, if you connect a Digital Pulse Generative Switch to a NOT gate, and connect the latter to the top input of the TNEG, then if you pulse, Q will change 1 (or stay at 1, if it was already 1), no matter what the other inputs are. Similarly, if you connect a Digital Pulse Generative Switch and a NOT to the bottom input, then a pulse will change Q to 0.

DPOS is an ordinary D flip-flop. The input will be copied to Q during the rising part of the clock pulse. The top and bottom inputs work like those of TNEG.

6.4.9 A TRIBUF Tri-State Buffer

This is a tri-state buffer. The output is the box on the right; connect this to whatever you need. The input is on the left side of the triangle. The control is on the upper side of the triangle; if this is 1, then the input is connected to the output.

6.4.10 TO/FROM (Invisible Wires)

Two very handy tools for reducing clutter on your Chipmunk screens are the TO and FROM components, which set up “invisible” wire connections. Here is how to use them:

Say you would like to have an “invisible” wire between two components (for example, wires) A and B, with the invisible connection being named XYZ. Then draw a line connecting A to the “arrowhead” portion (further-right point) of a FROM component. Then tap the arrowhead, resulting in a beep and a red blotch appearing; type XYZ and return, causing an XYZ label to appear there. Now go to B. Draw a line connecting B to the stem of a TO component, and label the latter XYZ in the same manner. Then A and B are connected, just the same as if you had drawn a wire from one to the other.

6.4.11 74150 MUX

This is a 1-out-of-16 multiplexor (MUX). The inputs are numbered 0 to 15, from top to bottom on the left edge, and are selected via the four pins on the bottom edge, whose least significant bit is on the right. The pin at the top should be connected to 0 or left floating. The output is on the right edge, and is active-low.

⁹This is unlike the case of LEDs, registers, etc., which do retain their values.

6.4.12 74138 3-to-8 and 74139 2-to-4 Decoders

For the 74138, connect 1 to the top pin on the left edge, and leave the two “bubble” pins beneath it floating. The input is in the bottom three pins on the left edge, least significant bit on top. Output is on the right edge, least significant bit on top, active-low.

The 74139 is similar. Again, float the top-left pin. Input and output pins have LSB on top; output is active-low.

6.5 Editing Your Circuit

6.5.1 Rotation

In ROT mode, you can rotate any component, typically within the menu at the bottom of the **mylib** screen, simply by repeatedly tapping it.¹⁰ This makes connections easier.

6.5.2 Moving

To move an item, just drag it.

You can move a whole region in one operation. To do this, first drag the mouse in a direction not fully horizontal or fully vertical, which will create a box in the screen. Then release the mouse button, which will change the box borders to dashed-line form. Then move the mouse pointer to the point where you wish to move your region, and click the right mouse button.

6.5.3 Copying

Drag the mouse at Editing in the **mylib** window, releasing on Copy. Then tap the item to be copied. Move the mouse pointer to the desired place, and then hit the left mouse button. Keep moving the mouse pointer and hitting the left mouse button until you’ve made the desired number of copies. Then hit the right mouse button to end copy mode.

If you wish to copy an entire region, first enter Copy mode as described above. Then drag the mouse to produce a box around the region to be copied (see instructions for moving, above). Then copy as described above, by repeatedly moving the mouse and hitting the left button.

6.5.4 Deleting

You can enter delete mode by choosing the Delete item in the Editing portion of the **mylib** window, or by hitting the d key.¹¹ The mouse pointer will change to a picture of a pair of scissors. You can then repeatedly move the mouse pointer to different items, tapping each one you wish to delete. (You may have to move the mouse pointer a little before you find a place at which the cut works.)

¹⁰For some components, e.g. Digital Switches, they can only be rotated while within the menu, i.e. you must rotate one within the menu first, then move it into the circuit.

¹¹Many of the mouse actions have keystroke equivalents.

You can delete a whole region by dragging the mouse to create a box in the screen, while in Delete mode. Everything in the box will be deleted.

When you finish, leave Delete mode by clicking the right mouse button.

Quick trick: A fast way to delete a simple object like a wire is to simply drag it down off the bottom of the screen.

6.5.5 Labels

You should make liberal use of labels in your drawing, as they serve as analogs of program comments.

To make a label, hit the l key, and then start typing your label. It will appear near the lower-left edge of the **mylib** window. When you are finished typing, hit the return key, and you can then use the Move capability (see above) to move the label to wherever you want it in the picture.

6.5.6 Resizing, Scrolling and Pages

The **mylib** window holds one “page” at a time. You can draw different circuits on different pages, and copy from one to another. To switch to page n, just type n.

Each page is essentially infinite in scope; you can scroll using the arrow keys. You can also resize the **mylib** window just like you would any X11 window.

6.5.7 Saving/Loading Circuits

You can save a circuit to a file, by hitting S. You will be prompted in the **newcrt** window for a file name. The standard suffix for circuit file names is **.lgf**.

To load a previously-saved file, hit L.

Note that Chipmunk does not honor tildes (representing your home directory) in file names.

6.6 Modular Design

As with programming, modular design is essential to building complex digital systems. Chipmunk facilitates such design.

IMPORTANT NOTE: Chipmunk is very unforgiving of errors in the module creation and usage processes. Be absolutely sure to follow the directions here to the letter.

6.6.1 Creating a Module

To create a module (recall that Chipmunk refers to this as a **gate**, which is an abuse of the term), do the following:

- Build the circuit which you intend to convert to a module. Test it, but after testing it, remove whatever extra testing items you put in, e.g. LEDs.
- Go to Tier 2 and choose an instance gate (one of the rectangles with arrows), of whatever size you wish, and drag it to the display page where your would-be module is. The box will be called a **template** by Chipmunk, e.g. in error messages.
- Connect the inputs and outputs of the module to the instance gate, at any of the four edges of the latter. Important: Remember which edges you connect which inputs/outputs to; you will need this information later.¹²
- Create a label, say “gy” (the quotes are required) which will serve as the name of your new module type.¹³ Drag the label to the inside of the instance gate.
- Go to CNFG mode. Then “compile” your new module as follows. Tap the inside of the instance gate (careful—don’t tap the label). Then type the same name (with the quote marks), e.g. ”gy”, when asked by the **newcrt** window. then using the down-arrow key to the ”Display name?” part of the form, and hit y to make it say Yes. Then hit the space bar once.

You will get a message that the module is being compiled. This message seems to stay forever, but actually the compilation should only take a second or two unless the circuit is really large. You can make the message disappear by hitting the space bar a couple of times. (Similarly, each time you change the module, you will get a message “recompiling” which will stay for a long time; hit the space bar to get rid of it if it bothers you.)

Note: You are not allowed to connect a wire directly from one point of a module box to another, without having the wire go through any intermediate gates. If you do, Chipmunk will complain that you have shorted together two template pins. This is a problem, because in some cases you may wish to do this; for instance, you may need one of the outputs of a module to be identical to one of the inputs. If so, put in some ”filler” gates, e.g. two NOT gates in series.

- The module definition is now complete. You can use the module now, or save it to an **.lgf** file for later use.

6.6.2 Using a Module

- Keep in mind that you can’t use a module until it is loaded into your current Chipmunk session. So, if the module is not already on a screen page, load it into one.¹⁴
- Go to the page where you wish to use the module (separate from the module page). Drag the instance box from CAT to this page. Warning: Make sure it is the same-numbered (1-5) instance box as the one used in the module, i.e. the same size and shape; the pictures in Tier 2 are misleading, as they are not always the same size as the box which will be used.
- Switch to CNFG mode, and tap inside the box. Again the **newcrt** window will ask you to type the module name with quotes. Don’t hit the return key, but use the down-arrow key to go to the ”Display name?” part of the form, and hit y to make it say Yes. Then hit the space bar once. You should now see the name appear inside the instance gate. (The module name then displayed in the box will not have quotes, even though you typed them.)

¹²Better yet, add this information via labels.

¹³Note: Chipmunk seems to have a problem with nonalphanumeric characters such as an underscore in the label.

¹⁴See Section 6.5.7.

- Then connect wires to the module in the order in which they were declared. Exact position doesn't matter, as long as the correct edges and order are followed.

Each time you connect a wire to the box, a message should appear concerning the number of connections and ports this module has, ports meaning places where you should connect, and connections being the number of places where you have connected so far. (If such a message doesn't appear, then you do not have an official module, and it will not work.) When you have connected all the wires ("ports") needed by a module, its outline will become brighter.

6.6.3 Multimodule Pages and Files

It is possible to have more than one module on the same display page, and to save all the modules on a page into a single **.lgf** file. When building circuits with a large number of modules, this means you have to keep track of fewer files, load fewer files upon startup, and use fewer pages. It thus becomes a great convenience factor, so you are strongly encouraged to make use of this feature.

Suppose I wish to store modules A and B, currently in **a.lgf** and **b.lgf**, into a single **.lgf** file. Here are steps I could take:

- Load **a.lgf** into, say page 1, and **b.lgf** into page 2.
- Draw a dashed box around A in page 1, as follows. Hit the b key to enter Box mode, resulting in the mouse pointer turning into a box symbol. Move the mouse pointer to the upper-left corner of A, and then drag the mouse pointer to the lower-right corner. Click the right mouse button to exit Box mode.
- Using the arrow keys, scroll to move A completely off the screen in page 1, so that you will have room to copy B.
- Go to page 2. Hit the / key to enter Copy mode. Place the mouse pointer at the upper-left corner of B, and then drag the mouse pointer to the lower-right corner. You will now be in Paste mode (an item in the lower-right corner of the **mylib** window will confirm this). Do NOT move the mouse any more.
- Go to page 1. You should see a copy of B there now. Move the mouse pointer so that the copy of B is at the place in page 1 you desire. Press the left mouse button to indicate that "this is the place," and then hit the right mouse button to exit Copy mode.
- Draw a dashed box around B in page 1 using Box mode, as explained above.
- Save page 1 to a file, say **ab.lgf**, which will now contain both A and B.

6.6.4 Restrictions

Most of the Chipmunk components are written in Chipmunk's LOGED language, and any of these can be used in creating a module. However, a small number of components, such as the SRAM8K, are written in Pascal, and these cannot be used in modules.

Note that each instance gate has a limit on the number of pins it can use. The limit is displayed when you position the mouse pointer over the gate in LIBR.

6.7 Miscellaneous

To refresh the screen, several methods are available, such as tapping it, hitting the space bar, and so on.

Chipmunk assumes TTL electronics, which normally should not concern you, except that one implication is that if you do not connect anything to an input pin of a component, this has the same effect as if you had connected a 1 to it.

One drawback of Chipmunk is that it assumes that each component, no matter how complex, takes one clock cycle to operate. This is OK for our purposes, and it avoids the subtle timing problems which plague many digital simulation programs.

I have found that on some machine types, sometimes Chipmunk encounters errors due to insufficient window memory. To avoid this, make sure that your **mylib** and **newcrt** windows do not overlap any other window. You may wish to iconify or temporarily delete some windows. In any case, frequent saving of your Chipmunk pages to their files would be wise.

7 Examples

The **.lgf** files for the examples in this section are available at <http://heather.cs.ucdavis.edu/~matloff/Chipmunk/Examples>.

7.1 “If-Then-Else”

This is an example of how a multiplexor can be used to implement the analog of “if-then-else” from the programming world. This circuit **SM2C.lgf** converts a 4-bit integer stored in signed-magnitude form to 2s complement form. The 4-bit input is via the hex keypad at the middle left.

If the sign bit is a 0, then the output is equal to the input.

On the other hand, if the sign bit is a 1, i.e. the number is negative, the sign bit is changed to a 0, so that the resulting 4-bit string is the positive version of the number. That is the positive version of the number under the 2s complement system too, of course, so now we need to use the algorithm which converts the 2s complement representation of a positive number to the representation for its negative counterpart. That algorithm, as you will recall, exchanges 1s and 0s and then adds 1. So, the circuit takes the modified 4-bit string and inverts it, using the Chipmunk **INV4** component, equivalent to a bank of 4 NOT gates, and then we add 1 (stored in the upper-left hex keypad). The two cases are both fed into the '157 MUX, which is controlled by a line which we have connected to the sign bit.¹⁵

7.2 ModDiv5

Here is an example of the use of various Chipmunk features. It may take you a while to understand every detail of this example, but that time spent will definitely pay off in a good understanding of Chipmunk.

¹⁵In other examples in which a MUX is used for “if-then-else,” then we would have a more complex set of gates whose output feeds into the MUX’s control bit.

Here we wish to design a circuit which will constantly tell us the values of $n \bmod 5$ and $n \text{ div } 5$, where n is the number of clock pulses received so far. We could design this by setting up an ordinary binary counter, say a 16COUNT component, and then feed its output into gates which would compute the mod and div for us. But instead of actually doing that computation, we can take the following simpler route, as we have in the file **ModDiv5.lgf**.

The circuit is in the file **ModDiv5.lgf**. We maintain two 16COUNT components, one for $n \bmod 5$ and the other for $n \text{ div } 5$. The first counter should be incremented on every clock cycle, but should be cleared each time it reaches 5. The second counter should be incremented only at those cycles in which the first counter is cleared. I have placed labels “mod 5” and “div 5” at the two 16COUNTs.

Note that I have attached Hexadecimal Displays at the outputs of the two 16COUNTs. I could have just placed individual LEDs, but the Hexadecimal Displays are more convenient and are clearer.

At the left I have placed a Digital Pulse-Generative Switch, which will serve as “clock” input for testing the circuit. Try tapping it a few times (at least until the mod count cycles back to 0), watching the mod and div counts change.

At the top, I have placed a Digital Switch, and a label “init.” As the label implies, the Digital Switch is used to clear the two binary counters to 0 when we first start the circuit (the C input in a binary counter clears the counter). Note that after tapping this switch to clear the counters, we must tap it again before the circuit can continue to be pulsed.

Remember, we need to put in circuitry to check whether the mod count has reached 5, i.e. 101 binary. I do this using a 74138 Decoder (see Section 6.4.12). Note that the 74138 outputs are active low, i.e. 0 means yes and 1 means no. For example, if the inputs are 101, i.e. 5, then pin 5 of the output will be 0, meaning, “Yes, the input was 5.” This is why I have the NOT gates at the two output pins I am using.

Looking again a pin 5, you will see that it leads ultimately to the C (“clear”) input of the mod counter. Again, this is because I want the count to revert to 0 after reaching 5. At the same time I want the div counter to be incremented. This is why the clock pulse input for the latter is ANDed with the (negated) output of pin 4 of the decoder.¹⁶

7.3 SRAM8K

Here is an example of use of the SRAM8K component, in the file **SRAM8K.lgf**.

In this very simple example we are using an SRAM8K chip to store four 2-bit words. We have 2-bit address and data buses, a clock line, and a read/write line (1 for read, 0 for write), to which we have attached Digital Switches, LEDs and a Digital Pulse Generative Switch.

To write a value to our little memory here, first set the Digital Switches on A0, A1, D0 and D1 according to what value you want to write to which word of our memory. Set the Digital Switch on the Read/Write line to 0. Then tap the “clock” (the Digital Pulse Generative Switch).

Write a few values to memory. Then try reading them: Simply set A0 and A1 to the desired address, and set the Read/Write line to 1. Check via the LEDs on D0 and D1.

Note the tri-state devices on the Digital Switches on D0 and D1, which are crucial since these lines serve

¹⁶You might at first think it should be pin 5. But this would present a timing problem. Experiment with this a little and you will see.

for both reading and writing.

7.4 Fetch-and-Add

Adding something to a variable and then storing the sum back in the variable is one of the most common operations in computing, as in the C code

```
Y += X;
```

Here we need to first read Y, then add X to it, then store the sum back in Y, i.e. write to Y. Some multiprocessor computer systems add extra hardware near the memory to expedite this operation; instead of a CPU doing the addition, it is done by this extra hardware. This operation is called **fetch and add**. The file here is **FetchAndAdd.lgf**.

(The reason for having special fetch-and-add hardware is that in multiprocessor systems, the memory is usually very “far”—in a timing sense—from the CPUs, so that each trip to memory takes a long time. The special hardware then allows us to make only one trip to memory instead of two.)

Now, how does the circuit work? The key point is that we will need two clock cycles for the fetch-and-add, because we cannot both read and write the SRAM8K (labeled “memory”) in one cycle. So, we need something to keep track of “what time it is,” in terms of the count of clock cycles. Since our operation only takes two clock cycles, our “time” can be measured mod 2, i.e. in just a single bit. For that purpose I have placed a TNEG (“toggle”) flip flop near the upper-left section of the circuit.

The “time” is stored in the Q output of that TNEG. If you follow the line to the R input of the SRAM8K, you see that at time 0 we read from memory. If you follow the line to the tri-state buffers, you see that at time 1 we write to memory.

For simplicity, we are using 2 bits as both our word and address size.

In this example we are constructing fetch-and-add-1 hardware. So, we have an ’83 adder, one addend of which comes from reading the SRAM8K and the other addend being the constant 1. The sum is then stored in the ’175 register for safekeeping until the write-cycle.