Coyote-1™ User's Manual

Version 1.1

August 20, 2008

WARRANTY

OpenStomp™ warrants the Coyote-1 against defects in materials and workmanship for a period of 60 days from receipt of product. If you discover a defect, OpenStomp™. will, at its option, repair or replace the merchandise, or refund the purchase price. Before returning the product to OpenStomp™, call for a Return Merchandise Authorization (RMA) number. Write the RMA number on the outside of the box used to return the merchandise to OpenStomp™. Please enclose the following along with the returned merchandise: your name, telephone number, shipping address, and a description of the problem.

COPYRIGHTS AND TRADMEARKS

DISCLAIMER OF LIABILITY

Eric Moyer and OpenStomp™ are not responsible for special, incidental, or consequential damages resulting from any breach of warranty, or under any legal theory, including lost profits, downtime, goodwill, damage to or replacement of equipment or property, or any costs of recovering, reprogramming, or reproducing any data stored in or used with OpenStomp™ products. Eric Moyer and OpenStomp™ are also not responsible for any personal damage, including that to life and health, resulting from use of any of our products. You take full responsibility for your OpenStomp™ application, no matter how life-threatening it may be.

Dedication

This project is dedicated to the memory of my dear friend Larry Altneu, who died immediately after crossing the finish line in the 2007 Orange County Marathon.

Larry was a terrific mentor. He significantly shaped the Engineer I am today, and introduced me to many of the people involved in manufacturing this device. Hardly a week goes by that I don't use some Engineering trick I learned from him.

Engineers tend to live in the future. We make long term plans, dream new things, and force them into existence. When this world occasionally reminds us that we are not in control,
it comes as a bit of a shock.

Thanks

First of all, I have to thank my awesome wife Krisula for sticking by me. This project cut into my free time in a major way for the greater part of a year. She knew it was something I just had to do, and she supported me all the way.

A huge thanks to Steve Wozniak, both for creating the Apple computer and for writing the book "iWoz". I had the audio version of iWoz playing in my car the night I conceived the Coyote-1 and it was instrumental in inspiring me. Whenever this project lost momentum I'd start listening to iWoz again and it would fire me back up. Now that I'm done I've listened to that book at least ten times. It still gets me pumped to make stuff.

Revision History

1.0	Aug 7, 2008	Initial Release
1.1	Aug 20, 2008	Add Expansion Port Section Elaborate on Control Socket Value to Time Conversions

Table of Contents

Chapter 1 Introduction

```
$DO || ! $DO ; try
try: command not found

                 - jma5t3r_y
```

Welcome to my crazy little world. Thanks for joining me.

You have in your hands the product of many late nights, much day dreaming, some speculative hunches, a surprisingly large volume of caffeine, and a willingness to take some risks. If I'd known quite how much work it would be I might not have started, but by the time I realized what I'd gotten into it was already far too late to quit. I hope you enjoy it as much as I do. Personally, I think it's pretty darn cool.

Like many of you, I've dreamed about the possibility of an open source audio effects processor for a long time. A lot of different things kept me from starting until the day I thought of creating one around the Propeller processor, and that idea was so intriguing it dragged me kicking and screaming through just over a year of design. I really had no choice. I just had to do it.

Chapter 2 Licensing Summary

The Coyote-1 project consists of many different assets, released under a mix of different licensing arrangements. The following is a summary of the release licenses:

Asset	Source	License
Coyote-1 O/S Source Code	Open	GPL 3
Coyote-1 Effects Modules Source Code (and associated Dynamic Modules files (*.c1m))	Open	GPL 3
Coyote-1 Schematics (and associated Printed Circuit Board)	Provided	Copyright. Reproduction & Redistribution Prohibited.
Coyote-1 User Manual	Provided	Copyright. Reproduction & Redistribution Prohibited.
Open Stomp™ Workbench	Closed	Copyright. Reproduction & Redistribution Prohibited.
Patches	N/A	Creative Commons 3.0 Attribution Non-commercial (by-nc)
Coyote-1 Hardware	N/A	Copyright. Reproduction & Redistribution Prohibited.

This list is for reference purposes only. The licensing arrangements for each asset are declared on or within the assets.

Chapter 3 Recommended Reading

Propeller Documentation

The Propeller Manual

Describes the architecture and operation of the Propeller chip, the use of the Propeller Tool (i.e. the Propeller IDE), the SPIN language, and the Propeller Assembly language. Avalable from the the "Help" menu within the Propeller tool, or from Parallax's website www.parallax.com .

Propeller Information

The Propeller Forum

Parallax maintains a forum for the Propeller chip here:
http://forums.parallax.com/forums/default.aspx?f=25

deSilva's Machine Language Tutorial

A growing reference of information about programming the Propeller in assembly language. The document is maintained on the forums here:
http://forums.parallax.com/forums/default.aspx?f=25&m=209237

Programming resources

A collection of various Propeller Programming resources is maintained on the forums here:
http://forums.parallax.com/forums/default.aspx?f=25&m=204210

Digital Signal Processing

General Introduction

A pretty good and not overly mathematical introduction to DSP can be found here:
http://www.dsptutor.freeuk.com/

Chapter 4 Recommended Tools

PASD (Propeller Assembly Sourcecode Debugger)

PASD is a fantastic source level debugger tool for Propeller assembly code, written by Andy Schenk. Around August of 2007 I realized that a source level assembly debugger would be immensely useful to Coyote-1 effect authors. I trolled around the Parallax forums to see what people thought, and it turned out Andy had already written one, but the manual was in German and he was waiting to get an English translation before releasing PASD to the public. I ended up writing the English manual translation for him and the rest is history.

A copy is included on the Coyote-1 distribution CD, and the project is maintained here: http://www.insonix.ch/propeller/prop_pasd.html

HAM (Hydra Asset Manager)

HAM is a clever and handy tool written by Richard Benson for the Propeller based Hydra video game system to support loading and archiving data stored in the (typically unused) upper 96K of the Hydra's 128K EEPROM. Because the Coyote-1 also uses a 128K EEPROM, and uses the same pins for video as the Hydra, HAM will run on the Coyote-1 and can be used to archive/restore a "snapshot" of the entire OS, the stored patches, and stored dynamic modules (see Chapter 8).

A copy is included on the Coyote-1 distribution CD, and the project is maintained here: http://forums.parallax.com/forums/default.aspx?f=33&p=1&m=168490

GEAR

GEAR is a cool Propeller chip emulator written by Robert Vandiver. There are a couple of times when I got completely stuck because I couldn't figure out what my code was doing. I was able to throw the offending snippets into GEAR, single step them, and figure out what was really going on.

A copy is included on the Coyote-1 distribution CD, and the project is maintained here: http://forums.parallax.com/forums/default.aspx?f=25&m=164602

Chapter 5 Software Installation

1. Copy the contents of the Coyote-1 CD to a directory on your hard drive (such as C:\Coyote1). The remaining steps are written assuming a "C:\Coyote1" installation.
2. Install the "Propeller Tool" software by running its installer (Located in C:\Coyote1\PropellerTool\"). When prompted check "Automatically install/update driver (recommended)".
   NOTE: This is the software development environment for the Propeller Chip. Even if you are not planning to develop Coyote-1 software at this time, you need to perform this step because the installer loads the USB to Serial chip driver necessary for OpenStomp™ Workbench to communicate with the Coyote-1.
3. Run "OpenStomp™ Workbench Setup.msi" (located in C:\ OpenStomp(TM) Workbench\").
4. If you do not already have the appropriate .NET framework installed, you will be directed to Microsoft's website where you can download and install it. You can install a higher rev .NET framework than required if desired.
   NOTE: A copy of the .NET 2.0 Compact framework is in C:\OpenStomp(TM) Workbench\.NET Framework 2.0, but I have not yet been able to test it on a machine that did not already have the framework installed.
5. Complete the OpenStomp™ Workbench installation.

Chapter 6 Overview

The Coyote-1 is a digital guitar effects pedal based on the Propeller processor from Parallax. The Propeller is a unique embedded microprocessor containing 8 independent "cogs". Each cog is essentially a dedicated microprocessor. All 8 cogs execute simultaneously.

The Coyote-1 was designed to be Open-Source. A big part of that challenge was creating an infrastructure through which developers could create effects modules that could interoperate, be configured by non-technical end users, and be freely exchanged. To accomplish this, the Coyote-1 uses the concept of **effect modules**. An **effect module** is a piece of software which implements one (or possibly more than one) effect and which executes on one of the Propeller processor's 8 cogs.

Effect modules are interconnected by virtual signal pathways called **conduits**, which connect to **sockets** on those **effect modules**. **Sockets** are virtual data exchange ports through which a single 32 bit value is exchanged every audio sample period (i.e. at a rate of 44 KHz, which is approximately once every 22.7 microseconds). There are two different types of **conduits**: **signal conduits** which carry audio data and **control conduits** which carry control data (such as the output of the buttons and knobs).

The pedal contains a number of **system resources** which implement **sockets** on which they output or input data (the audio jacks, the buttons, the knobs, the LEDs, etc.). To turn an **effect module** into something you can actually use, you must specify the conduit routing between the various **system resources** and the **effect module**. This is accomplished using the OpenStomp™ Workbench application.

A specific configuration of **effect moules**, **system resources**, and the **conduit** routing which interconnects them is called a **patch**. OpenStomp™ Workbench provides a graphical interface in which **patches** can be authored, loaded, saved, and transferred to/from the Coyote-1.

OpenStomp™ Workbench also allows users to load/save and transfer **effects modules** to/from the Coyote-1.

The Coyote-1 can hold up to 15 **patches**, and up to 16 **effect modules** in its EEPROM memory, and an additional 4 **effect modules** can be compiled into the O/S at any given time.

Once the Coyote-1 has been loaded with a collection of **patches** and **effects modules**, those **patches** can be accessed using the foot switches without connecting the Coyote-1 to a computer.

The Coyote-1 ships with a factory installed collection of **patches** and **effect modules**. If at any time you wish to restore the factory configuration, instructions for doing so can be found in Chapter 8.

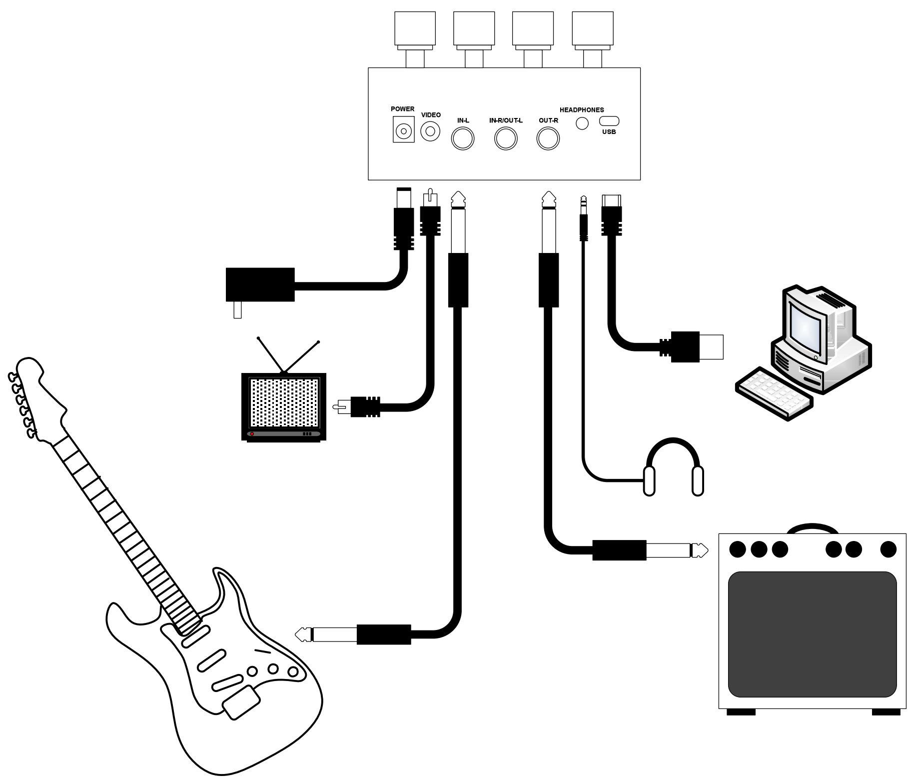
Chapter 7 Using the Pedal

Connecting Equipment

Connect the Coyote-1 to your equipment as shown below. The video output is not necessary unless you are running video applications (the standard O/S build, and standard effects, do not use it). The PC connection via USB is only necessary when configuring the pedal or loading software; the pedal can operate without a PC connection.

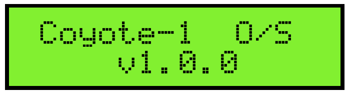
The functional assignment of the three ¼ inch jacks is patch dependant. The standard assignment for monaural single input / single output patches is to use “IN-L” as the input, and “OUT-L” as the output.

Note: The IN-L jack is a very high-impedance input (~1M Ohm), suitable for high impedance devices such as electric guitars. The IN-R input (because of its dual function as OUT-R) is a medium impedance input. Plugging a high impedance device (like an electric guitar) into IN-R will result in a rolling off of the high frequencies. If you need to plug a high impedance device into IN-R, try running it through another guitar stomp box first. The other stomp box will act a pre-amp to convert the signal to low impedance. Guitars which contain built-in preamps will work well with IN-R.

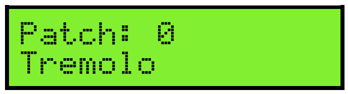


Boot

When you first apply power to the pedal it will display the Coyote-1 O/S revision.



After booting the Coyote-1 O/S will load "Patch 0" and start it running. In the default shipping configuration Patch 0 will be the Tremolo patch.

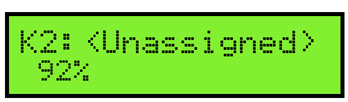


Controlling a Patch: Knobs

To modify patch parameters, spin the knobs. The functional assignment of the knobs is patch dependant. When you rotate a knob the pedal will display the knob being changed ("K0" in the image below, for "Knob 0"), the parameter assigned to that knob ("Delay" below), the current value (835 below), and the units (mSec below, or "milliseconds" (1000 milliseconds = 1 second).

The O/S implements "sticky" knobs, which means that if you rotate a knob the knob will not begin to modify the assigned parameter's value until you have rotated its position to match the current parameter value. If you start spinning a knob and the parameter does not change, just keep spinning the knob across its full range; when you match the current value the knob will become "unstuck", and the display value will begin to follow the knob position.

Some patches may not make use of all 4 knobs. If a knob has not been assigned to a function then the parameter name will be displayed as "<Unassigned>", and rotating the knob will have no effect on the patch being heard.



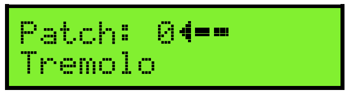
By convention the right most knob (K3) is typically used to control the final gain stage (output volume) of a patch.

Controlling a Patch: Buttons

The function of the two "foot switch" buttons is patch dependant. Typically the buttons are used to turn different effects on and off, and typically the on/off state is represented by the LED associated with each button.

Switching Patches

To switch between patches, step on both button simultaneously. An arrow will appear pointing to the current patch number.



Clicking the right button will advance to the next patch. Clicking the left button will go backwards one patch. Stepping on both buttons simultaneously will load (and start) the currently selected patch.

NOTE: If the selected patch is the same as the patch which was running when patch select mode was entered, the patch will NOT be reloaded, but will continue to run undisturbed.

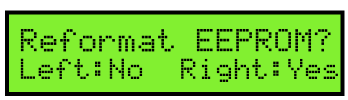
Reformatting the EEPROM

Performing a EEPROM format will erase all patches and modules stored in EEPROM. Once reformatted, patches and modules can be reloaded into EEPROM using the OpenStomp™ Workbench application.

NOTE: It is recommended that the patches and modules you develop be archived on your PC, and that you do not rely on the pedal's EEPROM as your only patch/module storage.

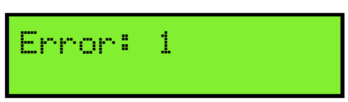
Why reformat? If you create a custom module or patch which crashes the pedal when it attempts to load Patch 0 on boot then reformatting can be used to get the O/S booting again so that it will talk to the Workbench application.

To reformat the EEPROM, hold the left button down while powering up the device, then click the right button when prompted. Clicking the left button will cancel the reformat and boot normally.



Patch / Module Load Errors

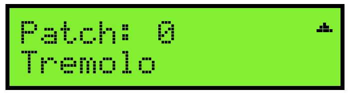
If a patch fails to load successfully due to a patch or module error then a numeric error code will be briefly displayed. A list of error codes can be found in Chapter 13.



The Error codes are defined in the source file `COYOTE1_HW_Definitions.spin`.

Real Time Error: Output Clipping

If the final digital output value (i.e. the value sent to the DAC) reaches its maximum then a small upward pointing triangle will appear in the upper right corner of the display. This is an indication that the output is potentially becoming distorted by going outside of the available output range (i.e. clipping).



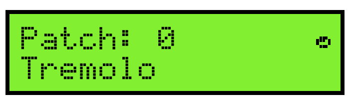
Output clipping may be caused by too high an input signal, or too much gain in the effects chain composing the current patch.

NOTE: Clipping is only detected at the final output stage. It is possible to have audible clipping distortion occur within a patch and not trigger the “Output Clipping” indicator if the clipping occurs in one effect module and some subsequent effect module reduces the gain.

Real Time Error: Time Overflow

The Coyote-1 operates at a standard sampling frequency of 44kHz. That means that each effect module typically has one 44kHz sample window (22.7 microseconds) in which to process a given sample. The OpenStomp™ architecture refers to this interval as a “microframe”.

Effect modules can self-report to the O/S if they take more than their allotted 22.7 microseconds, and the O/S will display a Time Overflow indication in the upper right hand corner of the display.



NOTE: All standard Open Stomp modules implement Time Overflow reporting.

NOTE: It is possible to write a module which operates at a lower sample rate by taking more than one microframe to process a sample. Such a module can still implement Time Overflow reporting by self-reporting when it goes over its self-allotted processing interval.

Chapter 8 Restoring the Factory Configuration

There are two different ways to crash the Coyote-1:

One is to install a bad O/S build, or somehow corrupt the build you have. That situation can be recovered by just recompiling a good O/S build from the CD and using the Propeller Tool to load it onto the Coyote-1.

The second way to crash is to corrupt the EEPROM data with bad patches, or bad dynamic modules, or bad configuration data. That situation can be remedied by erasing the EEPROM per the "Reformatting the EEPROM" section of Chapter 7, but when you're done you'll have an empty EEPROM and you'll need to reload any patches or modules you were using.

If you don't want to go through the motions of restoring your configuration piecewise, you can use a tool called "HAM" (the "Hydra Asset Manager") to reload the entire EEPROM (OS, modules, and patches) to the factory configuration. HAM was written by Richard Benson to manage EEPROM data for Andre' LaMothe's "Hydra" Propeller-based video game system (http://www.xgamestation.com/view_product.php?id=33). Because the Coyote-1 uses the same pins for video as the Hydra, HAM will also work on the Coyote-1.

To restore the factory configuration using HAM:

1. Connect the Coyote-1 to your PC using the mini-USB cable and power it on.
2. Start HAM (It should be locate in your "C:\Coyote-1\Additional Utilities\Hydra Asset Manager (HAM)\" directory).
3. Select the COM port on which the Coyote-1 is attached.
4. It is helpful, but not necessary, to connect a video monitor to the Coyote-1 so you can watch the progress of HAM.
5. Click "Load HAM Driver". You will see the button change states when the load is complete. If you have video attached, you will see the HAM driver screen appear on the attached display.
6. Drag the factory configuration .eeprom file into the black "Memory Map" box in the HAM window. The configuration file shoud be located in your "C:\Coyote-1\Coyote-1 Firmware\" directory, and will be named something like "Coyote-1 Factory Configuration Image 001 (OS 1.0.2).eeprom".
7. Click "Upload to Hydra". When the upload completes a dialog box will appear.
8. Cycle power on the Coyote-1, and it will boot into the restored configuration.

Richard Benson maintains HAM here:
http://forums.parallax.com/forums/default.aspx?f=33&p=1&m=168490

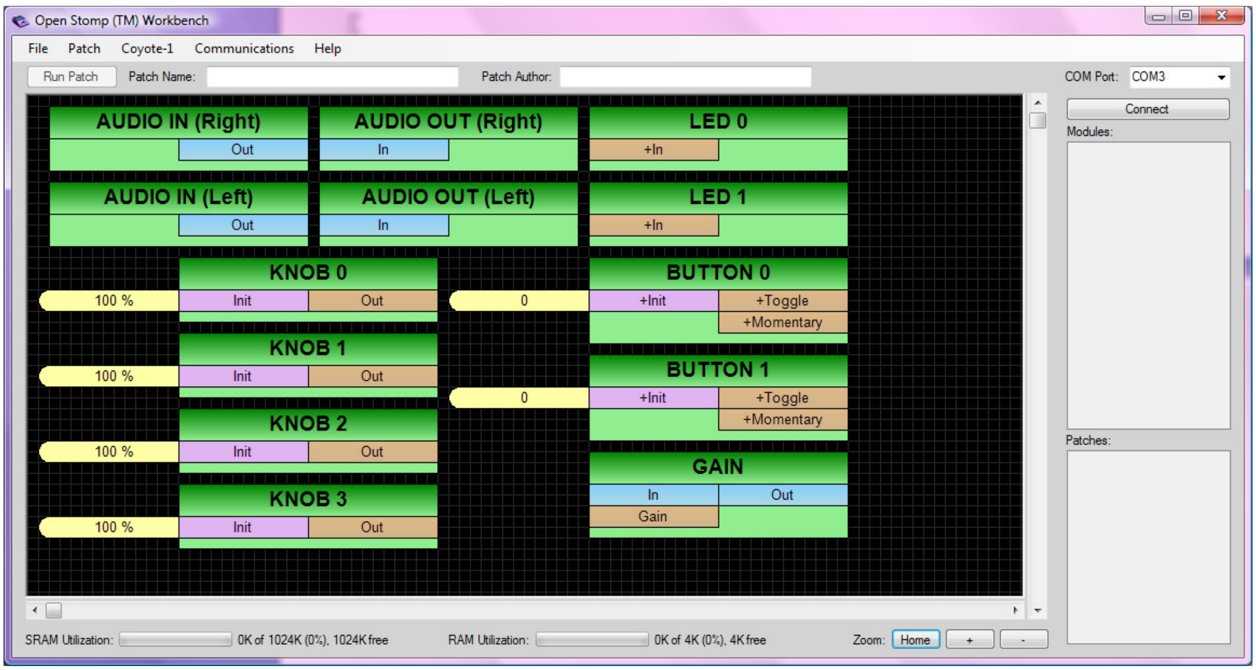
NOTE: It’s also possible to use the “Download from Hydra” button in HAM to take a “snapshot” of your Coyote-1’s EEPROM, which you could then reload at a later time using the method above.

Chapter 9 Using OpenStomp™ Workbench

Connecting to the Coyote-1

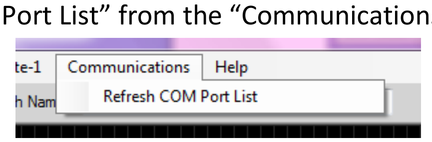
Before starting OpenStomp™ Workbench, attach the Coyote-1 to your PC using the supplied mini-USB cable. Windows will assign the Coyote-1 a dedicated virtual COM port.

When you first start OpenStomp™ Workbench you will see the set of available System Resources (green) in the Patch Editor pane. Both the “Modules” list and the “Patches” list will be blank.



Using the “COM Port:” drop down menu in the upper right corner, select the COM port on which the Coyote-1 is attached and then click “Connect”. The “Modules” list and the “Patches” list will be updated to reflect the Effect Modules and Patches currently present on the attached device.

Note: The Coyote-1’s COM port may not appear in the “COM Port:” drop down menu if you attached it after starting Workbench, or switched USB ports. To update the COM port list select “Refresh COM Port List” from the “Communications” drop down menu.



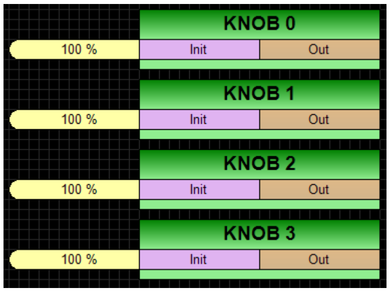
Advanced: OpenStomp™ Workbench closes the COM port when not communicating with the Coyote-1 even though the state appears to be “Connected”. This is done so that new O/S code can be compiled in the Propeller IDE and loaded into the device without quitting Workbench or disconnecting.

Advanced: If you added, changed, or removed Static Modules by compiling and loading a new O/S build, you can update the "Modules" list by clicking "Diconnect", and then clicking "Connect".

System Resources

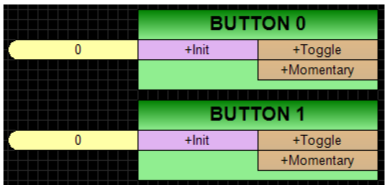
Knobs

There are 4 knob resources, representing the 4 physical knobs on the Coyote-1. Knob 0 is the left-most knob. The Init conduit allows you to specify the initialization value of the knob when creating a patch (0-100%). The output conduit outputs the knob's position.



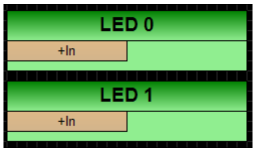
Buttons

There are 2 button resources, representing the 2 physical button on the Coyote-1. Button 0 is the left-most button. The Init conduit allows you to specify the initialization state of the button's "+Toggle" output. The "+Toggle" output toggles its value each time the button is pressed and released. The "+Momentary" output reflects button's current state (down or up).



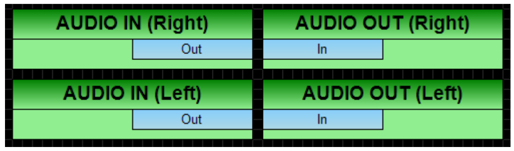
LEDs

There are 2 LED resources, representing the 2 physical LEDs. LED 0 is the left-most LED.



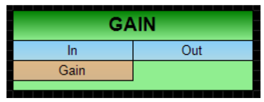
Audio

There are 4 audio resources, representing the 4 audio ports. "Audio In (Right)" and "Audio Out (Right)" share the same physical jack.



Gain

There is a gain resource which is typically used as a final output volume control for a patch, and is typically configured to be controlled by Knob 3. The "In" and "Out" conduits are the input and output audio conduits respectively. The "Gain" conduit (have you guessed already?) sets the gain.



Loading a Patch

You can load a patch into the Editor by either selecting "Open Patch" from the "File" menu, or by right clicking on a Path in the "Patches" list and selecting "Load this Patch into the Editor".

Note: The Editor cannot show an Effect Module in the Editor Pane if that Effect Module does not exist in the connected Coyote-1 device. If you attempt to load a Patch which contains an Effect Module that is not currently present in the Coyote-1 (either as a Static Module or as a Dynamic Module), then an error will be displayed, and the Patch will load without the missing Effect.

Working in the Editor

The large black grid area is the Patch Editor Pane. Patches are created by interconnecting System Resources and Effects Modules inside the Editor Pane.

Zoom and Pan

Rotating the mouse wheel will zoom in and out (the mouse cursor must be over the Editor Pane). Zoom can also be performed by clicking the "+" or "-" buttons at the bottom right of the Editor Pane.

Clicking "Home" will return the Editor Pane to 1:1 zoom and will pan to the home position (the upper left of the work area).

Holding down both the left and the right mouse buttons simultaneously while moving the mouse will pan the editor pane.

Moving Objects

To move an Object (i.e. a System Resource (green) or an Effect (purple)) click the top "Title Bar" region of the object and drag it. Objects cannot be overlapped. If you attempt to overlap objects a red boundary will appear, and if you release the mouse button while in an overlapping position the object will return to its original location.

Adding Effects to a Patch

To add an Effect Module to a Patch, right-click an Effect Module in the "Modules" list and select "Add to Editor Patch". Static Modules (shown in red in the "Modules" list) and Dynamic Modules (shown in purple in the "Modules" list) can both be added to Patches.

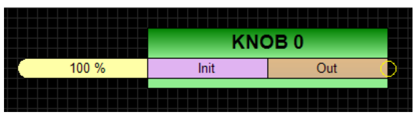
The added Module will appear in the upper left corner of the working area (you may need to zoom/pan to see it), and may overlap existing objects. Play nice and drag it somewhere better before hooking it up.

Connecting Objects

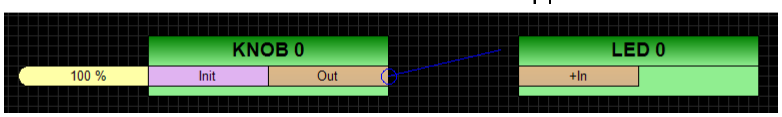
Objects (System Resources and Effects) have connection points called **sockets** which can be connected to one another using wires called **conduits**.

To create a **conduit:**

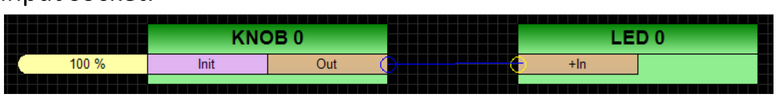
1. Hover your mouse over the right hand edge of an **output socket** (i.e. one on the right hand side of an **object**). A yellow circle will appear at the edge of the **conduit**.



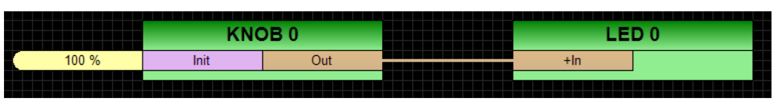
2. Click the left mouse button. A blue line will appear and will follow the mouse.



3. Hover the mouse over the left hand edge of an **input socket** (i.e. one on the left hand side of an **object**). The input socket type (Signal (blue) or Control (brown)) must match the output socket type. A yellow circle will appear when hovering over a valid input socket.

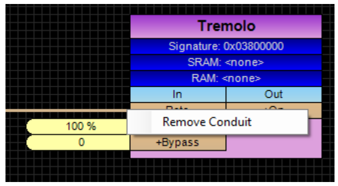


4. Click the left mouse button. A conduit will be created between the two sockets.



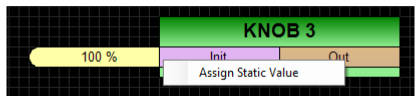
To delete a **conduit**:

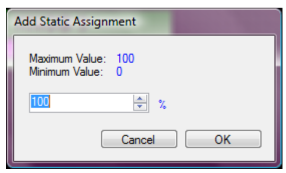
1. Hover the mouse over either the right hand edge of the **output socket** from which the conduit originates, or over the left hand edge of the **input socket** to which the conduit terminates. Click the right mouse button. A popup menu will appear. Select "Remove Conduit".



Modifying Static Assignments

Any unconnected input **socket** (i.e. a **socket** to which a **conduit** has not been attached) has a default **static assignment** value which is shown in a light yellow bubble to the left of the **socket**. To modify the static assignment, hover the mouse over the left edge of the **conduit** and click the right mouse button. A popup menu will appear. Select "Change Static Assignment" from the popup menu. A dialog box will appear allowing you to modify the assigned value.





Main Menu Commands

File

New Patch

Creates a "Blank Slate" in the Patch Editor Pane, containing only the built in System Resources.

Open Patch

Opens a Patch file (.c1p) from disk into the Patch Editor Pane.

Save Patch

Saves the current patch (in the Patch Editor Pane) to a Patch file (.c1p) on disk.

Patch

Run

Loads the current patch onto the Coyote-1 and starts it executing. The patch number on the device will display as "--" to indicate that the patch is a "temporary" load and is not stored in one of the regular numbered patch slots.

Unroute All Conduits

Removes all conduits from the current patch in the Patch Editor Pane.

Coyote-1

Connect

Attempts to establish a connection with the Coyote-1 device on the currently selected COM port. Selecting this option is equivalent to clicking the "Connect" button in the main window.

Reset

Performs a hardware reset on the attached Coyote-1 device.

Format EEPROM

Erases the Effects Modules and Patchs from the Coyote-1 EEPROM. Does not erase the Coyote-1 O/S, which is also stored in EEPROM.

Erase All Dynamic Modules

Erases the Dynamic Modules stored in EEPROM, but does not erase the Patches.

Erase All Patches

Erases the Patches stored in EEPROM, but does not erase the Dynamic Modules.

Communications

Refresh COM Port List

Updates the "COM Port:" drop down list to show all currently existing COM ports. You may need to select this option if the Coyote-1 was attached after starting Workbench, or was moved to a new USB port.

Help

About

Shows software revision, revision history, and copyright info.

Module List Menu Commands

Load From File

Loads a **dynamic module** from a Coyote-1 module file (.c1m), and stores it in the Coyote-1's EEPROM at the selected position. This operation can not be performed on the first four positions, which are reserved for **static modules**.

Save to File

Saves the selected module to a file as a **dynamic module**. This operation can be performed on both **static modules** and **dynamic modules**.

Erase

Erases the selected modules. This operation can only be performed on **dynamic modules**.

Add to Editor Patch

Adds the selected module to the **patch editor pane**.

Copy From →

Copies the **static module** selected from the hierarchical sub-menu to the currently selected **dynamic module** position.

Patch List Menu Commands

Load From File

Loads a patch from a Coyote-1 patch file (.c1p) into the selected patch location in Coyote-1 EEPROM.

Save to File

Saves the selected patch to a Coyote-1 patch file (.c1p).

Erase

Erases the selected patch from the Coyote-1 EEPROM.

Store the Current Editor Patch Here

Copies the current patch from the **patch editor pane** to the selected patch location in Coyote-1 EEPROM.

Load This Patch Into the Editor

Copies the selected patch from Coyote-1 EEPROM to the **patch editor pane**.

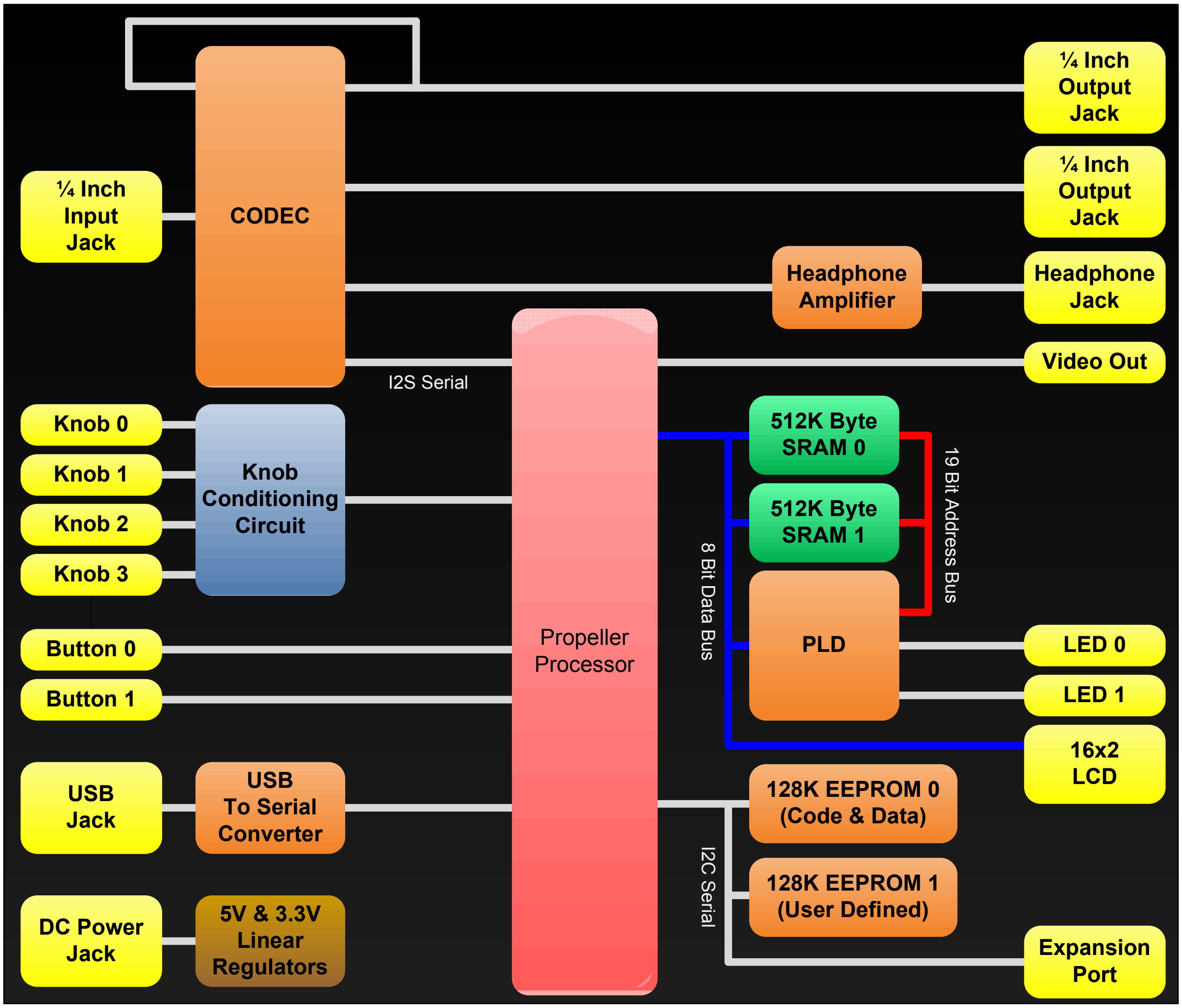
Conduit Routing Restrictions

The following restrictions apply to conduit routing:

1. Output Sockets can only be connected to Input Sockets.
2. Multiple Output Sockets **cannot** be connected to a single Input Socket.
3. A single Output Socket **can** be connected to multiple Input Sockets.
4. Signal Sockets (blue) can only be connected to other Signal Sockets.
5. Data Sockets (brown) can only be connected to other Data Sockets.
6. Initialization Sockets (purple) can only be assigned a static value (they **cannot** be connected to other Sockets).

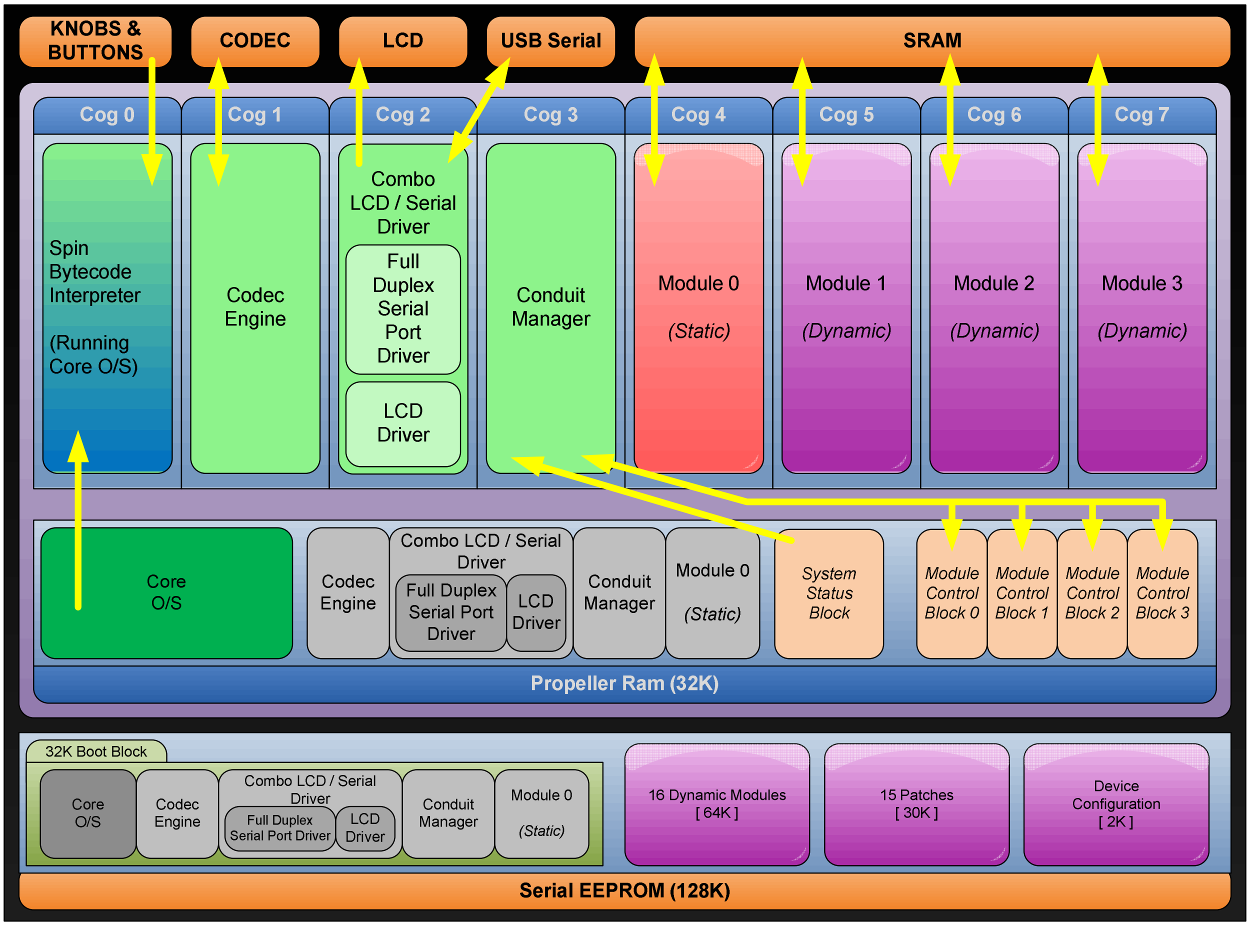
Chapter 10 Under the Hood

Block Diagram



Note: The shipping configuration of the Coyote-1 actually contains 3 512K SRAMs. The block diagram has not yet been updated to reflect this.

Software Architecture



Propeller Pin Assignments

Pin	Primary Function	Alternate Function
P0	MEMBUS_CNTL_0	LCD_REGSEL
P1	MEMBUS_CNTL_1	LCD_READ
P2	MEMBUS_CNTL_2	LCD_ENABLE
P3	KNOB_STROBE	
P4	KNOB_SHUNT	BUTTON_MUX
P5	BUTTON_READ	
P6	MEMBUS_CLK	
P7	KNOB_0	
P8	KNOB_1	
P9	KNOB_2	
P10	KNOB_3	
P11	CODEC_BCK	
P12	CODEC_SYSCLK	
P13	CODEC_DATAO	
P14	CODEC_DATAI	
P15	CODEC_WS	
P16	MEMBUS_D0	
P17	MEMBUS_D1	
P18	MEMBUS_D2	
P19	MEMBUS_D3	
P20	MEMBUS_D4	
P21	MEMBUS_D5	
P22	MEMBUS_D6	
P23	MEMBUS_D7	
P24	VIDEO_0	
P25	VIDEO_1	
P26	VIDEO_2	
P27	LCD_MUX	
P28	EEPROM_SCK	
P29	EEPROM_SDA	
P30	USB_RXD	
P31	USB_TXD	

MEMBUS Interface

MEMBUS_CNTL[2..0]	Function
0x0	Write SRAM Byte
0x1	Read SRAM Byte
0x2	Write SRAM Addr LOW [SRAM_A07..SRAMA00]
0x3	Write SRAM Addr MID [SRAM_A15..SRAMA08]
0x4	Write SRAM Addr HIGH [SRAM_A20..SRAMA16]
0x5	Set CCR (CODEC Control Register)
0x6	Set GPIO0 (General Purpose I/O 0)

PLD Register: CCR (CODEC Control Register)

Bit	Function
7	(Reserved)
6	CODEC_MC2
5	CODEC_MC1
4	CODEC_MP5
3	CODEC_MP4
2	CODEC_MP3
1	CODEC_MP2
0	CODEC_MP1

Note: The CODEC_yyy bits each control the corresponding CODEC_yyy pin directly. For an understanding of the various MCx and MPx pin functions, refer to the NXP UDA1345 CODEC data sheet.

PLD Register: GPIO0 (General Purpose I/O 0)

Bit	Function
7	(Reserved)
6	(Reserved)
5	(Reserved)
4	(Reserved)
3	(Reserved)
2	LED1
1	LED0
0	LCD_BACKLIGHT

Note: The LCD_BACKLIGHT is configured in hardware to be on whenever power is applied. Toggling LCD_BACKLIGHT will have no effect.

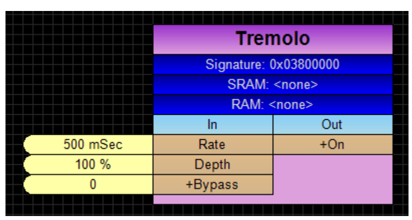
Chapter 11 Creating Custom Effects Modules

> This is the true joy of living. This being used for a purpose recognized by yourself as a mighty one. This being thoroughly used up before being thrown on the scrap heap. This being a force of nature instead of a feverish, selfish little clod, full of ailments and grievances, complaining that the world will not devote itself to making you happy.
>
> - G.B. Shaw

If you have not already familiarized yourself with the Propeller chip's hardware and software architecture, and with the Propeller IDE (the "Propeller Tool" software), you should take a look through the "Propeller Manaul" PDF and do so. The Propeller Manual is installed when you install the Propeller Tool, and can be accessed quickly from the "Help" menu within the Propeller Tool.

Before you learn to write custom effects, take a quick look at the existing effects modules to familiarize yourself with their structure. The effect module source files all have the form "COYOTE1_MODULE_*module_name*.spin".

This chapter will use the Tremolo effect (COYOTE1_MODULE_Tremolo.spin) as an example. Inside OpenStomp™ Workbench, the Tremolo effect looks like this:



The job of creating a custom effect is basically one of defining the effect's inputs/outputs (called **sockets**), and then writing the software to read the input **sockets** and create the proper behavior on the output **sockets**.

The Coyote-1 uses a 44kHz audio sample rate, which means that every 1/44000th of a second (approximately every 22.7 microseconds) the ADC (analog to digital converter) hardware captures a digital sample from each of the two analog input channels (In-L (left) and In-R (right)), and outputs a digital sample on each of the two analog output channels (Out-L (left) and Out-R (right). In general, an audio **effect module** must keep up with this sample rate, which means that it has 22.7 microseconds to read its input **sockets** and write new values to its output **sockets**. The exception to this rule is **effect modules** which are specifically written to operate at a lower sample rate. A 22kHz **effect module** would read its input **sockets** only once every 1/22000th of a second, and write its output **sockets** once every 1/22000th of a second.

In the Coyote-1 system architecture, the native 44kHz sample period is referred to as a **microframe**. The Coyote-1 O/S provides a mechanism by which **effect modules** synchronize to the start of a **microframe**, and by which they can report an overrun error if they ever take more than a single **microframe** to process their data.

Socket Types

There are three types of sockets; **signal** sockets, **control** sockets, and **initialization** sockets. **Signal sockets** carry audio data, and are colored blue in the Workbench editor. Their data is signed, and they have a maximum range of 0xffffffff (-2147483647) to 0x7fffffff (2147483647).

Control sockets carry control information (data values which affect the behavior of **effect modules** or **system resources**), and are colored brown in the Workbench editor. Their data is unsigned, and they have a maximum range of 0x00000000 (zero) to 0x7fffffff (2147483647). Sometimes a **control socket** is designed to implement a two state (on/off) function. In these cases the convention is to name the socket with "+" prefix. Two state inputs treat all input values below 0x40000000 as "false", and all input values above 0x40000000 as "true". Two state outputs are set to 0x00000000 when "false", and 0x7fffffff when true.

Initialization sockets are a special form of input **control socket** which reads its input value only once (during the startup of a patch). They are colored light purple. Their data is unsigned, and they have a maximum range of 0x00000000 (zero) to 0x7fffffff (2147483647).

The Module Descriptor

The **module descriptor** is a data structure which describes the attributes of an **effect module**, including its name, its size, its signature, its revision, how many conduits it has, the conduit definitions, how much SRAM it requires, and how much RAM it requires.

Early on in the **effect module** code you'll see the get_module_descriptor_p function, which allows the Coyote-1 O/S to get a pointer to the **effect modules' module descriptor**:

```
PUB get_module_descriptor_p
  ' Store the main RAM address of the module's code into the module descriptor.
  long[ @_module_descriptor + hw#MDES_OFFSET__CODE_P] := @_module_entry
  ' Return a pointer to the module descriptor
  return (@_module_descriptor)
```

Next, you'll see the module descriptor definition:

```
DAT

'------------------------------------
'Module Descriptor
'------------------------------------
_module_descriptor      long    hw#MDES_FORMAT_1                                'Module descriptor format
                        long    (@_module_descriptor_end - @_module_descriptor)  'Module descriptor size (in bytes)
                        long    (@_module_end - @_module_entry)                 'Module legth
                        long    0                                               'Module code pointer (this is a placeholder which gets overwritten during
                                                                                '   the get_module_descriptor_p() call)
                        long    $03_80_00_00                                    'Module Signature
                        long    $00_01_00_00                                    'Module revision  (xx_AA_BB_CC = a.b.c)
                        long    0                                               'Microframe requirement
                        long    0                                               'SRAM requirement (heap)
                        long    0                                               'RAM  requirement (internal propeller RAM)
                        long    0                                               '(RESERVED0) - set to zero to ensure compatability with future OS versions
                        long    0                                               '(RESERVED1) - set to zero to ensure compatability with future OS versions
                        long    0                                               '(RESERVED2) - set to zero to ensure compatability with future OS versions
```

```
                    long    0                                                       '(RESERVED3) - set to zero to ensure compatability with future OS versions
                    long    6                                                       'Number of sockets

                    'Socket 0
                    byte    "In",0                                                  'Name
                    long    0 | hw#SOCKET_FLAG__SIGNAL | hw#SOCKET_FLAG__INPUT      'Flags and ID
                    byte    0  {null string}                                        'Units
                    long    0                                                       'Range Low
                    long    0                                                       'Range High
                    long    0                                                       'Default Value

                    'Socket 1
                    byte    "Out",0                                                 'Name
                    long    1 | hw#SOCKET_FLAG__SIGNAL                              'Flags and ID
                    byte    0  {null string}                                        'Units
                    long    0                                                       'Range Low
                    long    0                                                       'Range High
                    long    0                                                       'Default Value

                    'Socket 2
                    byte    "Rate",0                                                'Name
                    long    2 | hw#SOCKET_FLAG__INPUT                               'Flags and ID
                    byte    "mSec",0                                                'Units
                    long    LFO_PERIOD_MIN_MSEC                                     'Range Low
                    long    LFO_PERIOD_MAX_MSEC                                     'Range High
                    long    500                                                     'Default Value

                    'Socket 3
                    byte    "Depth",0                                               'Name
                    long    3 | hw#SOCKET_FLAG__INPUT                               'Flags and ID
                    byte    "%",0                                                   'Units
                    long    0                                                       'Range Low
                    long    100                                                     'Range High
                    long    100                                                     'Default Value

                    'Socket 4
                    byte    "+Bypass",0                                             'Name
                    long    5 | hw#SOCKET_FLAG__INPUT                               'Flags and ID
                    byte    0  {null string}                                        'Units
                    long    0                                                       'Range Low
                    long    1                                                       'Range High
                    long    0                                                       'Default Value

                    'Socket 5
                    byte    "+On",0                                                 'Name
                    long    6                                                       'Flags and ID
                    byte    0  {null string}                                        'Units
                    long    0                                                       'Range Low
                    long    1                                                       'Range High
                    long    1                                                       'Default Value

                    byte    "Tremolo",0                                             'Module name
                    long    hw#NO_SEGMENTATION                                      'Segmentation

_module_descriptor_end  byte    0
```

A full definition of the module descriptor format can be found in Chapter 15.

Great care should be taken when creating a module descriptor. The module descriptor must be formatted *exactly* to the module descriptor format definition in Chapter 15 or the O/S will not be able to interpret it properly.

A Word about Socket Ranges

You will see in the **module descriptor** format that a socket range can be specified for each **socket**. The range is used only for the purposes of displaying values to the user when they rotate a knob connected to that **socket** by a **conduit**. For example, a knob always outputs values from 0x00000000 to 0x7fffffff. If you specify a range_low of 0 and a range_high of 100, and connect a knob to that conduit using Workbench, and rotate the knob fully clockwise, the user will see the value "100" displayed, and the value arriving at the socket will be 0x7fffffff.

This "universal" ranging of control socket values (i.e. that control socket values always operate across the full range 0x00000000 to 0x7fffffff) was done to ensure the maximum flexibility when interconnecting objects in Workbench.

The Module Code

The module code will always consist of 6 basic regions:

1) Pointer Loading
2) Initialization
3) The Synchronization Loop
4) The Bypass Block
5) Effect Processing
6) Data Declaration

Pointer Loading

In this section the data pointer to the various system objects (frame counter, module control block, etc) are loaded, and the pointers to all the socket locations are set up.   This section will typically have the same form for all modules, though the socket pointers, their names, and their quantity will be unique.

```
_module_entry
                        mov      p_module_control_block, PAR               'Get pointer to Module Control Block
                        rdlong   p_system_state_block, p_module_control_block  'Get pointer to System State Block

                        'Initialize pointers into System State block
                        mov      p_frame_counter,     p_system_state_block
                        mov      p_ss_overrun_detect,p_system_state_block
                        add      p_ss_overrun_detect,#(hw#SS_OFFSET__OVERRUN_DETECT)

                        mov      p_socket_audio_in,   p_module_control_block
                        add      p_socket_audio_in,   #(hw#MCB_OFFSET__SOCKET_EXCHANGE + (0 << 2))
                        mov      p_socket_audio_out,  p_module_control_block
                        add      p_socket_audio_out,  #(hw#MCB_OFFSET__SOCKET_EXCHANGE + (1 << 2))
                        mov      p_socket_rate,       p_module_control_block
                        add      p_socket_rate,       #(hw#MCB_OFFSET__SOCKET_EXCHANGE + (2 << 2))
                        mov      p_socket_depth,      p_module_control_block
                        add      p_socket_depth,      #(hw#MCB_OFFSET__SOCKET_EXCHANGE + (3 << 2))
                        mov      p_socket_bypass,     p_module_control_block
                        add      p_socket_bypass,     #(hw#MCB_OFFSET__SOCKET_EXCHANGE + (4 << 2))
                        mov      p_socket_on,         p_module_control_block
                        add      p_socket_on,         #(hw#MCB_OFFSET__SOCKET_EXCHANGE + (5 << 2))
```

Initialization

In this section any necessary data initialization is performed. This section will be different for all modules, and will be very application specific. In the case of the tremolo module, there is a single variable to initialize.

```
                        '------------------------------------
                        'Init
                        '------------------------------------
                        mov      angle_16_16_fxp, #0
```

The Synchronization Loop

This code should be identical for all modules. The synchronization loop waits for the start of a **microframe** boundary (i.e. the start of an audio sample interval) before dropping into the execution of the effect code. It also detects overrun conditions (i.e. whether the effect code took longer than its allotted sample interval to execute) and reports them to the O/S if they occur.

```
                        '------------------------------------
                        'Sync
                        '------------------------------------
                        rdlong   previous_microframe, p_frame_counter          'Initialize previous microframe

                        'Wait for the beginning of a new microframe
_frame_sync             rdlong   current_microframe, p_frame_counter
                        cmp      previous_microframe, current_microframe  wz
              if_z      jmp      #_frame_sync                                  'If current_microframe = previoius_microframe

                        'Verify sync, and report an overrun condition if it has occurred.
                        '
                        'NOTE: An overrun condition is reported to the OS by writing a non-zero value to the "overrun detect" field in the
                        '      SYSTEM_STATE block.  The code below writes the value of current_microframe in order to conserve code space,
                        '      achieve portability, and limit execution time. That value will be non-zero 99.9999999767169% of the time,
                        '      which is sufficiently reliable for overrun reporting
                        '
                        add      previous_microframe, #1
                        cmp      previous_microframe, current_microframe  wz
              if_nz     wrlong   current_microframe, p_ss_overrun_detect

                        mov      previous_microframe, current_microframe       'previous_microframe = current_microframe
```

The Bypass Block

Most effects modules will choose to implement "Bypass" functionality, which gives the user the ability to step on one of the foot switches to turn the effect on or off.

```
                  '------------------------------------
                  'Get audio in sample
                  '------------------------------------
                  rdlong  audio_in_sample, p_socket_audio_in

                  '------------------------------------
                  'Bypass
                  '------------------------------------
                  'Read bypass state
                  rdlong  r1, p_socket_bypass
                  cmp     SIGNAL_TRUE, r1   wc, wz

                  'Update on/off indication
if_c_or_z         mov     r2, 0
if_nc_and_nz      mov     r2, SIGNAL_TRUE
                  wrlong  r2, p_socket_on

                  'If bypassed, then just pass audio through
if_c_or_z         wrlong  audio_in_sample, p_socket_audio_out
if_c_or_z         jmp     #_frame_sync
```

Effect Processing

This is where the real work gets done. The code will read the input sockets, work its magic, write the output sockets, and end with a jump back to the synchronization loop to wait for the next sample.

Data Declaration

This is where the variables and data are declared.

The Effects Module Creation Process

To create a custom **effect module** you would:

1) Author the **effect module** as a Propeller code module (the existing COYOTE1_MODULE_Tremolo.spin is an example of an **effect module**).
2) Link the **effect module** into the O/S build by modifying the COYOTE1_static_module_list.spin module to include the new module.
3) Recompile the O/S, creating a version which includes the new module as a **static module** (i.e as a compiled-in module).
4) Load the O/S onto the device.
5) Create a **patch** which uses the new module.
6) Run the **patch** to test the new module.
7) Iteratively test, modify, recompile, load until the **effect module** is complete and functional.
8) Use OpenStomp™ Workbench to save the module as a **dynamic module**. Once saved as a **dynamic module** the new module can be easily distributed to other Coyote-1 users.

Linking an Effect Module as a Static Module

Linking an effect as a "Static Module" means that it will be compiled into the O/S build. To link an effect module as a static module:

1. Load the file *COYOTE1_static_module_list.spin* into the Propeller Tool.
2. List the new module in the OBJ section as shown below. The name in quotes must match the filename of the effect module (excluding the .spin extension).
3. Replace any of the 4 static modules in the case statement with a call to the new module's *get_module_descriptor_p* function.
4. Save the modified *COYOTE1_static_module_list.spin* file.

```
OBJ
  tremolo:    "COYOTE1_MODULE_Tremolo"
  delay:      "COYOTE1_MODULE_Delay"
  chorus:     "COYOTE1_MODULE_Chorus"
  distortion: "COYOTE1_MODULE_Distortion"
  tunstuff:   "COYOTE1_MODULE_Tunstuff"
  testtone:   "COYOTE1_MODULE_TestTone"

PUB get_static_module_desc_p(module_index)
  '------------------------------------
  ' NOTE:  In general only 4 or fewer static MODULES can be declared here because each static MODULE
  '        occupies a cog and there are 4 available cogs in a standard Coyote-1 O/S build.  O/S builds
  '        in which additional cogs are occupied with cutom user modifications will have fewer than
  '        4 cogs available for static MODULES.
  '------------------------------------
  case module_index
    0:
      return tunstuff.get_module_descriptor_p
    1:
      return chorus.get_module_descriptor_p
    2:
      return delay.get_module_descriptor_p
    3:
      return distortion.get_module_descriptor_p

    other:
      'A zero must be returned for any static module which does not exist
      return 0
```

Recompiling and Loading the O/S

To recompile the O/S, open the COYOTE1_OS.spin file in the Propeller tool, select it as the currently displayed file (if it is not already) by clicking its tab, and press:

F11 (to build the O/S and load it into EEPROM)

or

F10 (to build the O/S and load it into RAM)

Useful Coyote-1 Control Socket Value to Time Conversions

It is often desirable to have a **control socket** govern a rage of time, such as the delay interval for an echo effect or the period of a Low Frequency Oscillator (LFO). A quick and efficient way to compute such an interval is to simply perform a right-bit-shift on the **socket** value so that the resulting numeric range translates into some useful time range (when interpreted as a length of time expressed in **microframes**). The following table can be used to determine the time range obtained by right shifting an input socket value by different amounts (and, in the rightmost columns, by multiplying by an additional factor of 3).

		(Unmodified)			(Times 3)	
Bit Shift	**Decimal Max**	**Hex Max**	**Delay in Sec**	**Decimal Max**	**Hex Max**	**Delay in Sec**
0	2147483647	0x7FFFFFFF	48806.446523			
1	1073741823	0x3FFFFFFF	24403.223250			
2	536870911	0x1FFFFFFF	12201.611614	1610612733	0x5FFFFFFD	36604.834841
3	268435455	0xFFFFFFF	6100.805795	805306365	0x2FFFFFFD	18302.417386
4	134217727	0x7FFFFFF	3050.402886	402653181	0x17FFFFFD	9151.208659
5	67108863	0x3FFFFFF	1525.201432	201326589	0xBFFFFFD	4575.604295
6	33554431	0x1FFFFFF	762.600705	100663293	0x5FFFFFD	2287.802114
7	16777215	0xFFFFFF	381.300341	50331645	0x2FFFFFD	1143.901023
8	8388607	0x7FFFFF	190.650159	25165821	0x17FFFFD	571.950477
9	4194303	0x3FFFFF	95.325068	12582909	0xBFFFFD	285.975205
10	2097151	0x1FFFFF	47.662523	6291453	0x5FFFFD	142.987568
11	1048575	0xFFFFF	23.831250	3145725	0x2FFFFD	71.493750
12	524287	0x7FFFF	11.915614	1572861	0x17FFFD	35.746841
13	262143	0x3FFFF	5.957795	786429	0xBFFFD	17.873386
14	131071	0x1FFFF	2.978886	393213	0x5FFFD	8.936659
15	65535	0x0FFFF	1.489432	196605	0x2FFFD	4.468295
16	32767	0x07FFF	0.744705	98301	0x17FFD	2.234114
17	16383	0x03FFF	0.372341	49149	0x0BFFD	1.117023
18	8191	0x01FFF	0.186159	24573	0x05FFD	0.558477
19	4095	0x0FFF	0.093068	12285	0x02FFD	0.279205
20	2047	0x07FF	0.046523	6141	0x017FD	0.139568
21	1023	0x03FF	0.023250	3069	0x0BFD	0.069750
22	511	0x01FF	0.011614	1533	0x05FD	0.034841
23	255	0x0FF	0.005795	765	0x02FD	0.017386
24	127	0x07F	0.002886	381	0x017D	0.008659
25	63	0x03F	0.001432	189	0x0BD	0.004295
26	31	0x01F	0.000705	93	0x05D	0.002114
27	15	0x0F	0.000341	45	0x02D	0.001023
28	7	0x07	0.000159	21	0x015	0.000477
29	3	0x03	0.000068	9	0x09	0.000205
30	1	0x01	0.000023	3	0x03	0.000068

For example, the “Delay” **effect module** has a delay range of 0 to 1489msec (0 to 1.489 seconds). To accomplish this, it declares a range of 0 to 1489 in its Module Descriptor block…

```
'Socket 2
byte    "Delay",0                                   'Socket name
long    2 | hw#SOCKET_FLAG__INPUT                   'Socket flags and ID
byte    "mSec",0                                    'Units
long    0                                           'Range Low
long    1489                                        'Range High    (Delay in samples will be $0000_FFFF max, = 1489 msec)
long    500                                         'Default Value (Half a second)
```

And then computes the delay by shifting the delay socket value right 15 bits…..

```
rdlong  r1, p_socket_delay          'r1 = *p_socket_delay     Read the delay control socket
shr     r1, #15                     'r1 >>= 15                Shift right 15 bits, so max val of $7fff_ffff becomes $0000_ffff
```

When a knob is attached (by a **conduit**) to the delay **socket** and rotated fully clockwise, the socket value will be 0x7fffffff (maximum). After right shifting 15 bits, the value will be 0x0000ffff (or 65535 decimal). When that value is used to control the echo delay time in **microframes**, the resulting delay is…

65535 * (1/44000) = 1.489 seconds

Note that the sample rate is 44kHz, so the sample interval is 1/44000 of a second.

Chapter 12 Working with the Expansion Port

Overview

The Coyote-1 has an expansion port designed to interface with external control devices such as pedal switch boards and analog foot pedals, or more experimental devices like accelerometers or proximity detectors.

The expansion port is an extension of the I^2C interface on the board (on which also sit the 2 local EEPROM devices). External devices can be easily interfaced using any of a number of different I2C chips available commercially. NXP makes I^2C chips that provide anywhere from 4 to 16 digital I/O lines (like the PCA9536 and the PCA9539), as well as ADC and DAC chips (like the PCF8591). Many other manufacturers also make I^2C compatible devices that interface to all kinds of interesting things.

The port can source up to 50mA of current at 3.3V, so low power devices can receive their power directly from the Coyote-1.

Software Paradigm

The paradigm to follow when creating expansion port devices is to write an accompanying **effect module** which effectively functions as a **device driver** for the new hardware. Most **effect modules** have both input and output sockets, but an **effect module** which is the **device driver** for a piece of expansion port hardware like an analog foot pedal might only have an **output socket** (representing the current pedal position).

Tips

I am currently working to create an expansion port device reference design for a simple device. In the meantime, here are some design tips for those interested in tinkering with the expansion port:

On the remote board

- Protect against power reversal with a diode.
- Use an I2C I/O expander to provide digital ins/outs
- Use an I2C ADC or DAC to provide analog ins/outs
- Provide pads for a transient line termination on SCL, SDA just in case you need them. (I have tested without termination on the external device side with no problems and decent looking signals for a 3 ft cable run.)
- Work whatever magic you like on the far side of the I2C devices.
- Design for use with a standard phone cable, and try to keep the cable length around 3 ft or less. Standard phone cables do not "cross over", so pin 1 (which is 3.3V power on the Coyote-1) should connect to pin 1 (also 3.3V power) on the remote device.
- Put two RJ-11 ports on your device, so that multiple devices can be "daisy chained".
- Be mindful that if you disturb the operation of the SCL and SDA lines too much (say, by adding too much load or strong termination) you may prevent the Coyote-1 from booting. This is

because the I^2C is shared with the internal Coyote-1 EEPROMs, one of which contains the Coyote-1's boot code.

On the Coyote-1

- The O/S needs to be updated to lock/unlock the I2C semaphore around the O/S's I2C accesses. I have not put that in yet, but it is very simple to add. The I2C semaphore is declared in *COYOTE1_HW_Definitions.spin* already, just not used yet (*LOCK_ID__I2C*).
- Write an "Effect Module" which is a device driver for the external device.
- I found essential in my own testing to re-initialize the external I2C device(s) on every pass through my main loop in the "device driver" code (typically these devices have one or more control registers which need to be initialized to configure their I/O). Doing so allows the user to arbitrarily unplug and re-plug the external device without having to restart the Coyote-1 just to reinitialize the external device's I2C chips properly.

Chapter 13 Error Codes

The following is a list of error codes which may be reported by the Coyote-1 device.

Value	Constant Definition	Description
0	ERR__SUCCESS	No error (internal return value, never displayed)
1	ERR__MODULE_NOT_FOUND	The patch contains a module which is not currently compiled into the code (static) or located in EEPROM (dynamic).
2	ERR__MAX_ACTIVE_MOD_EXCEEDED	The patch contains more modules than supported.
3	ERR__CONDUIT_ENG_START_FAILED	Could not start the conduit engine.
4	ERR__MODULE_COG_START_FAILED	Could not start a module.
5	ERR__I2C_WRITE_FAIL	Error writing the I2C bus.
6	ERR__I2C_WRITE_TIMEOUT	Timeout writing the I2C bus.
7	ERR__INDEXED_MODULE_DNE	A module was referenced which does not exist.
8	ERR__ILLEGAL_MODULE_INDEX	Module index out of range.
9	ERR__I2C_READ_FAIL	Error reading the I2C bus.
10	ERR__ILLEGAL_PATCH_INDEX	Patch index out of range.
11	ERR__OUT_OF_SRAM	The modules in the patch, taken together, requested more SRAM than the total available SRAM pool.
12	ERR__OUT_OF_RAM_POOL	The modules in the patch, taken together, requested more RAM than the total available RAM pool.

Chapter 14 Glossary of Terms

Term	Definition
DSP	Digital Signal Processing (or Digital Signal Processor). Using a digital computer to represent an analog signal as a sequence of discrete samples, and performing operations on that signal digitally.
Module	In the Propeller chip lexicon, a "Module" is a single .spin program file which may contain a mix of assembly code, Spin code, constants, and data.
Effect Module	An "Effect Module" is a piece of code which occupies a single COG and implements one or more audio effects. Sometimes "Effects Module" is referred to as just a "Module". Effects Modules can be either "Static" or "Dynamic" (see definitions).
Conduit	A data path which connects one output Socket to one or more input Sockets.
Socket	A 32 bit portal thorough which data is exchanged between Effect Modules and System Resources. Any given Socket is either an Input or an Output, and caries either Signal or Control data.
Static Assignment	A value assigned to an input Conduit to which no Conduit is attached.
Static Module	An Effect Module which has been compiled into the Coyote-1 O/S kernel.
Dynamic Module	An Effect Module which is stored in EEPROM.
Patch	A specific collection of Effect Modules and System Resources with a specific Conduit routing.
System Resource	Objects which appear in OpenStomp™ Workbench and which represent available hardware and software resources within the pedal, such as buttons, knobs, I/O ports, etc.
SRAM	Static Radom Access Memory. In this case, "SRAM" refers to the 1.5Mbytes of memory in external chips (i.e. outside the Propeller processor).
RAM	Random Access Memory In this case, "RAM" refers to the 32K of memory inside the Propeller processor.
DAC	Digital to Analog Converter
ADC	Analog to Digital Converter
Microframe	One 44kHz audio sample period (1/44000th of a second, or approximately 22.7 microseconds).
Frame	Eight microframes. (NOTE: The concept of "Frames" is not currently used, but exists for future development).

Chapter 15 Module Descriptor format

The module descriptor consists of 3 regions: The **module descriptor header**, the **socket definitions**, and the **module descriptor trailer**.

Module descriptor header:

Bits	Field	Description
32	module_descriptor_format	Identifies the module descriptor format (for future expansion). At present, the only valid format is 0x3130444d ("MD01" in ASCII).
32	module_descriptor_size	Length of the module descriptor (in byte)
32	module_size	Length of the module code (in bytes)
32	module_code_address	The address of the module code, in Propeller RAM.
32	module_signature	A unique signature used to identify the module. See Chapter 16 for a description of the module signature format.
32	module_revision	The module revision in the form 0xXXAABBCC = Rev AA .BB.CC
32	microframe_requirement	(Reserved for future use. Set to 0.)
32	sram_requirement	The amount of SRAM the module requires.
32	ram_requirement	The amount of Propeller RAM the module requires.
32	reserved0	(Reserved for future use. Set to 0.)
32	reserved1	(Reserved for future use. Set to 0.)
32	reserved2	(Reserved for future use. Set to 0.)
32	reserved3	(Reserved for future use. Set to 0.)
32	num_sockets	The number of sockets the module implements.

Socket definitions (one for each socket):

Bits	Field	Description
8 x n	socket_name_string	The socket name. 32 characters max. Zero terminated.
32	socket_flags	Length of the module descriptor (in byte)
8 x n	socket_units_string	The socket units (e.g. "mSec", "Hz", "%", etc.). 32 characters max. Zero terminated.
32	range_low	The max range, for purposes of display only.
32	range_high	The min range, for purposes of display only.
32	default_value	The module revision in the form 0xXXAABBCC = Rev AA .BB.CC

Module descriptor trailer:

Bits	Field	Description
8 x n	module_name_string	The socket name. 32 characters max. Zero terminated.
32	segmentation_flags	Reserved for future use. Set to 0.

Chapter 16 Module Signature Format

Each **effect module** must have a unique 32-bit **module signature** with the form 0xTTFVVVVV where 'TT' is the **effect type**, 'F' is a set of 4 **signature flag** bits, and 'VVVVV is a unique ID value.

NOTE: It is important that every **effect module's** signature be unique. The Coyote-1 O/S uses the **signature** as the *sole means* of identifying **effects modules**. If two **effects modules** have the same **signature**, then the incorrect module may be loaded when the O/S loads a patch.

The current **effect type** definitions are :

0x0*n*	**MODULATION**
0x01	Chorus
0x02	Flagner
0x03	Tremolo
0x04	Ring Modulator
0x05	Vocoder
0x06	PA Mic Control
0x07	Compressor
0x1*n*	**DELAY**
0x11	Echo / Delay
0x12	Reverb
0x2*n*	**EFFECTS**
0x20	Distortion
0x21	Wah
0x22	Synth
0x3*n*	**EQ**
0x30	Parametric EQ
0x4*n*	**EQ**
0x40	Tuner
0x41	Note Detector
0x42	Signal Generator

The current signature flag bits are:

0x8	OpenStomp™ effect (i.e. released by OpenStomp™) This bit is set for effects written and released by OpenStomp™. This bit should be a zero for user authored effects.
0x4	Experimental effect. This bit should be set when authoring effects for personal experimentation which will not be (or have not yet been) released to the public.
0x2	(Reserved)
0x1	(Reserved)

When you release an effect to the public, it should have a signature of the form 0xTT0VVVVV. You can email support@OpenStomp.com to request a unique ID Value (VVVVV) for your publically released effects.

Chapter 17 Module Control Block format

The **module control block** (MCB) is the structure by which the O/S interfaces with an effects module. A pointer to the MCB is passed to an effects module when it boots, and the effects module reads the various fields of the MCB to determine the location of its allocated memory (if any SRAM or RAM was requested by the module). The MCB is also the mechanism for the exchange of socket data (inputs and outputs) routed to and from the effect.

Bits	**Field**	**Description**
32	ss_block_p	A pointer to the system status block
32	heap_base_p	A pointer to the SRAM block granted to the module (per the module's SRAM request via the sram_requirement field in its **module descriptor**).
32	ram_base_p	A pointer to the RAM block granted to the module (per the module's RAM request via the ram_requirement field in its **module descriptor**).
32	microframe_base	*(Reserved for future use.)*
32	runtime_flags	*(Reserved for future use.)*
32	reserved0	*(Reserved for future use.)*
32	reserved1	*(Reserved for future use.)*
32	reserved2	*(Reserved for future use.)*
32	reserved3	*(Reserved for future use.)*
32*n	socket_exchange[]	The number of sockets the module implements.

Chapter 18 Epilogue

> **`This was a triumph.`**
> **`I'm making a note here:`**
> **`HUGE SUCCESS.`**
>
> *- GLaDOS, Portal*

Man oh man oh man. This has been some ride. It feels great to be finished, but in many ways it's just starting. I began this project because I wanted to play around with weird ideas in audio. There was so much work to do to put the infrastructure in place that it's really only today, this moment of "completion", that the *real* fun can begin.

Here we go…

> "Yes you will," enthused Zaphod, "there's a whole new life stretching out ahead of you."
>
> "Oh, not another one," groaned Marvin.
>
> *- HHGTTH, D. Adams*