Sphinx

Standalone Propeller development system

Project Number

PC097772

Project Description

Sphinx is a Spin compiler that runs on the Propeller chip. It can compile complex and substantial programs (including those containing Propeller assembly language) such as the Parallax tv and graphics objects and even Sphinx itself.
In order to support compilation, Sphinx also performs some of the functions of an operating system. It provides a command-line shell, a text editor, disk utilities, and a memory-resident (well, cog-resident) I/O system.

Hardware requirements

Sphinx can run on basically any Propeller system that has an SD card interface, an NTSC video interface, and a PS/2 keyboard interface. Examples of commercial Sphinx-compatible systems include the HYDRA with SDMAX and the Hybrid board.

Limitations

- Limited memory.
- .spn files can contain only ASCII characters, no UNICODE.
- Minimal floating-point support: floating-point literals are supported, floating-point constant expressions are not.
- The following keywords are not implemented: FLOAT, ROUND, TRUNC, FILE, ORGX, $.
- Sphinx is unable to optimize most long branches to short branches.
- Sphinx does not detect and remove objects that are duplicates at the bytecode level.
- The Sphinx file system supports only short ("8.3") filenames and has no directory support.
- Needs some sort of makefile or build system.
- Needs a better editor.

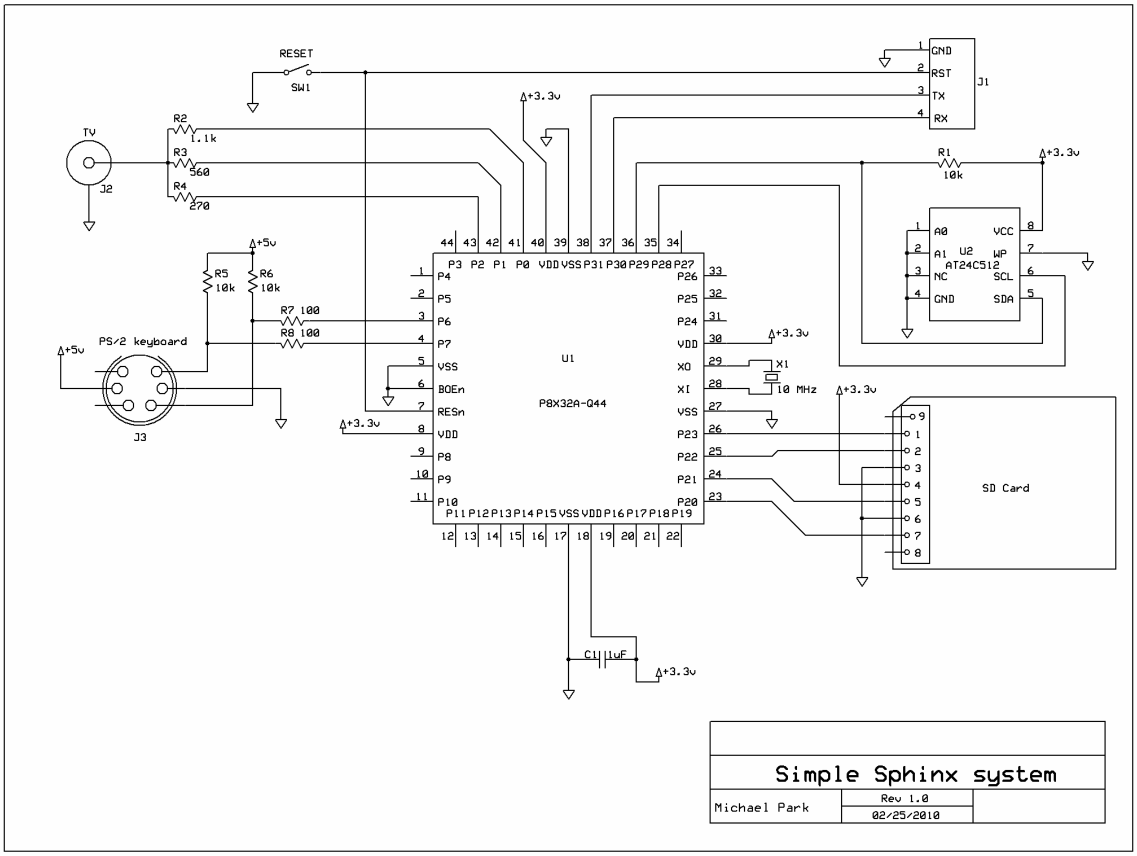
Sphinx is not intended to replace PC-based Spin compilers. It is really more of a proof-of-concept, to show that a Propeller-based Spin compiler is indeed possible.

Usage

On booting up Sphinx, you will see a command-line shell. You can enter commands such as "cl *filename*" to compile and link a file, "dir" to list the files on the SD card, and "ed *filename*" to edit a file. A list of commands is at www.sphinxcompiler.com.

Schematic

Sphinx is a software project and can run on many Propeller platforms. A schematic is a bit beside the point, but for reference, here is an example of a minimal Sphinx-compatible system:

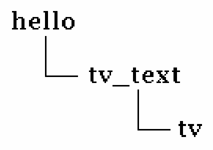


Source Code

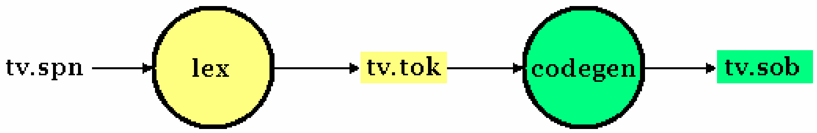
The source code is too large to include here. See the separate .zip archive, or download from www.sphinxcompiler.com. Installation instructions are on the website.
Instead of a source listing, here is a description of how Sphinx works.

Theory of operation (compiler)

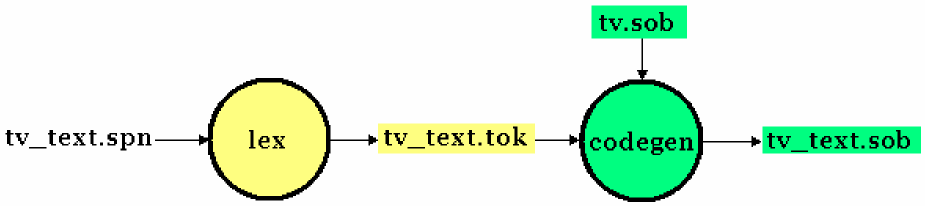
PC-based compilers such as the Propeller Tool compile Spin source directly to executable binary images. Sphinx cannot do this as it would generally require compiling multiple Spin files at the same time, and the Propeller does not have enough memory to accomplish this.
Instead, Sphinx compiles Spin source to Spin Object Binary files. This process can be done on one .spn file at a time. A separate link step combines .sob files into an executable .bin file.
The compilation process is performed by two programs. The first, lex, tokenizes the Spin source and produces an intermediate .tok file. The second, codegen, uses the .tok file and produces the .sob file. Codegen is a two-pass compiler, so generating a .tok file saves codegen the effort of tokenizing twice.
Linking is done by the program named link.
Example: consider a simple "Hello, world" program that uses the tv_text object. The tv_text object in turn uses the tv object.



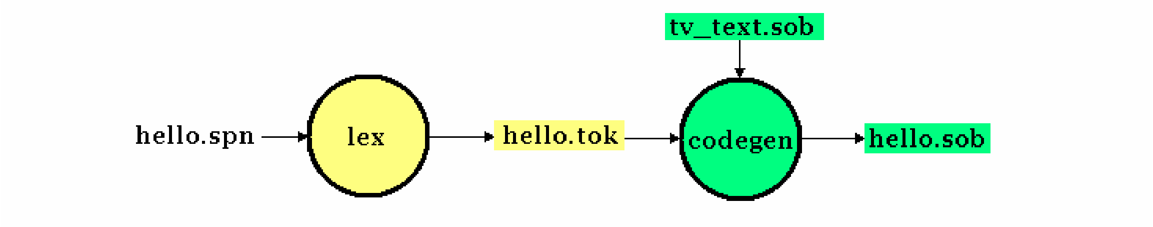
Compiling tv.spn produces tv.sob:



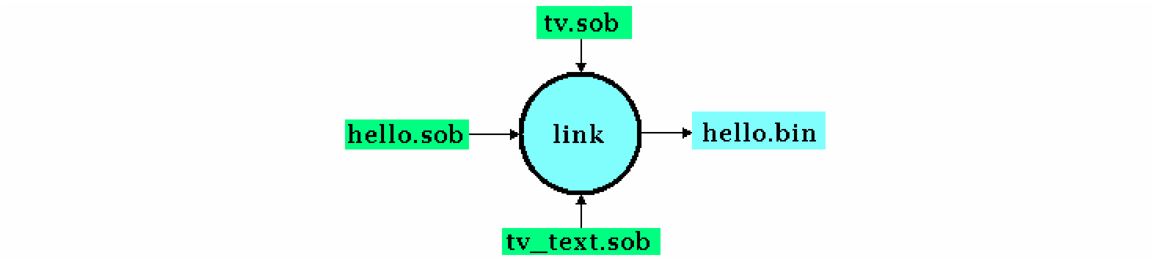
Compiling tv_text.spn produces tv_text.sob. Note that the compiler gets information about the tv object from tv.sob. It does not have to recompile tv.spn.



Similarly, compiling hello.spn produces hello.sob. This time the compiler has to read tv_text.sob.



To generate hello.bin, the linker has to link hello.sob with the two other .sob files:



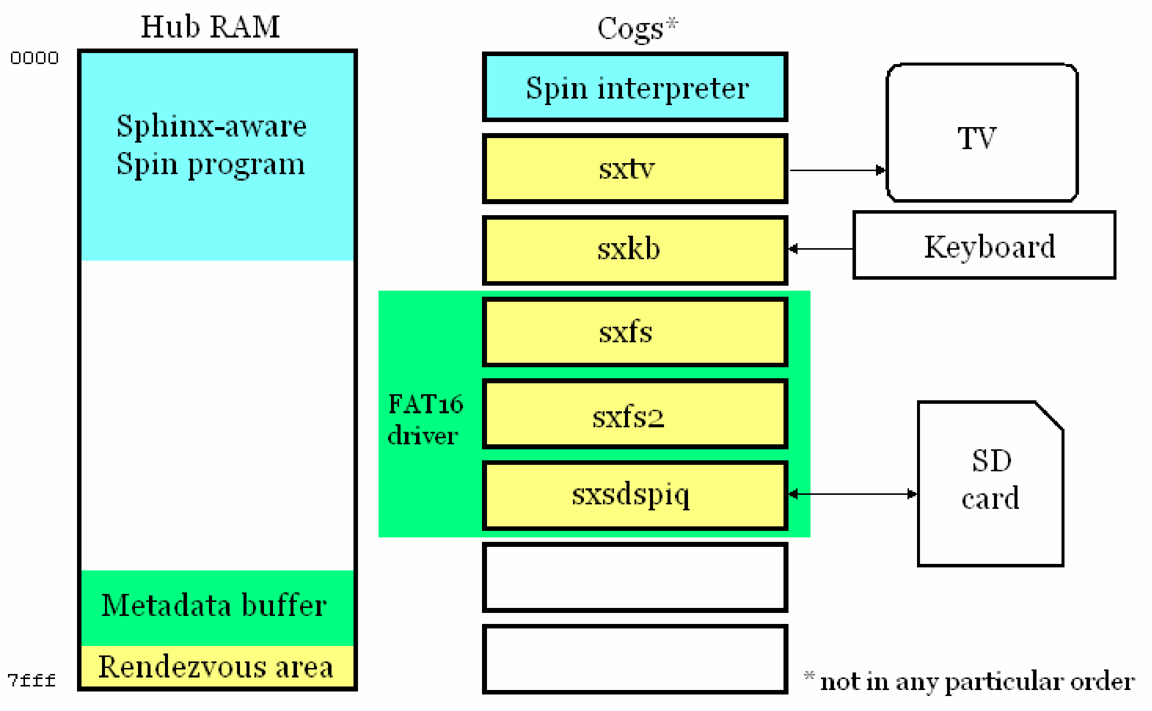
Theory of operation (operating environment)

To free up precious hub RAM, Sphinx I/O is performed by device drivers that reside entirely in the Propeller's cogs, taking up almost no hub memory. For example, the video driver runs in a cog and maintains its screen buffer (13 lines of 40 characters) in its cog memory. It communicates with the rest of Sphinx through a single long memory location (known as a *rendezvous*, to use terminology borrowed from fsrw) at the top of hub RAM space.

Similarly, the keyboard driver occupies no hub memory except for a rendezvous location.

The SD card driver is somewhat more complicated. It takes up three cogs. One is a low-level SD SPI driver (fsrw's sdspiqasm, lightly modified). The other two cogs implement a very barebones FAT16 file system that provides basic read, write, and execute functionality. Programs can open multiple files but must allocate some hub memory for each file. Communication with the file system is through several rendezvous locations. The file system requires a 512-byte buffer for metadata.

There are simple interface objects that hide the details of using the drivers.



Sphinx operating environment

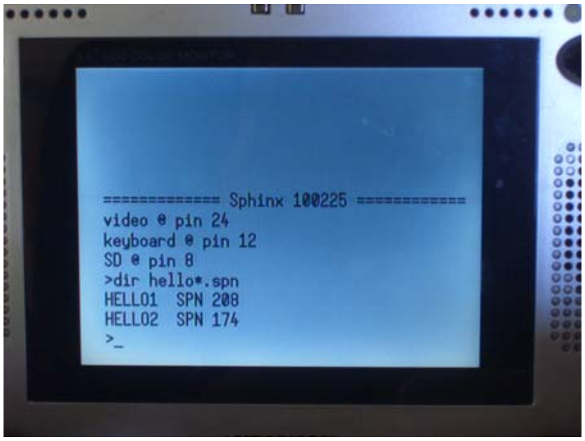
Sphinx installs the drivers when it first boots up. From that point on, the drivers remain resident and running even as different programs execute in hub memory (as long as they are Sphinx-aware programs). For example, if you run link.bin by typing "link" at the command prompt, what happens is this:

1. Sphinx.bin uses the Sphinx file system to load link.bin into hub memory and execute it while leaving all cogs running.
2. Link.bin runs, using the Sphinx drivers to perform TV, keyboard, and file I/O. Note that link.bin does not install any drivers.
3. When link.bin terminates, it uses the file system to load and execute sphinx.bin. Sphinx.bin determines that the drivers are already running and does not re-install them. It prints a prompt and awaits the next command.

Sphinx-aware programs do not have to carry device driver code inside themselves; they can rely on Sphinx drivers. Sphinx-aware programs can execute other programs, typically sphinx.bin, but not necessarily (for example, lex.bin executes codegen.bin so that the two parts of the compiler run seamlessly). Running sphinx.bin puts up a command prompt immediately after a program ends, without the long delay that a reset would cause. This, coupled with the fact that the display accumulates the output from each program (rather than clearing between each program), gives the convincing illusion of a traditional command-line shell.

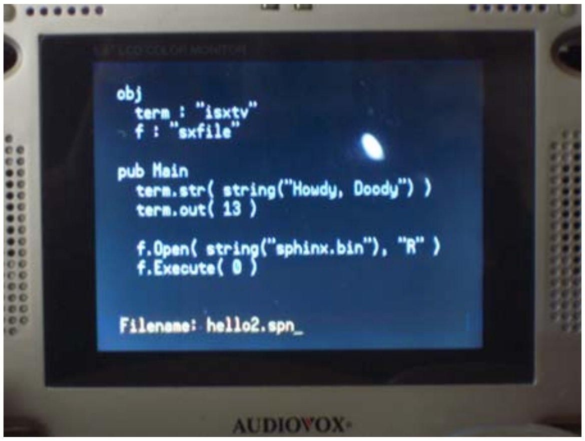
Sphinx can also execute regular (that is, non-Sphinx-aware) programs by performing the equivalent of a reset before running the program. This shuts down the driver cogs. The program must perform its own I/O. When the program finishes, a reboot will presumably occur, either programmatically or manually. At that time, Sphinx will load back in from EEPROM and re-install the Sphinx drivers.

Pictures



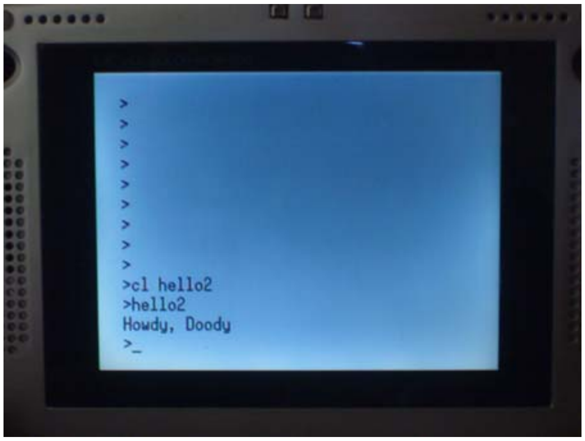
```
============= Sphinx 100225 ============
video @ pin 24
keyboard @ pin 12
SD @ pin 8
>dir hello*.spn
HELLO1  SPN 208
HELLO2  SPN 174
>_
```

Directory listing



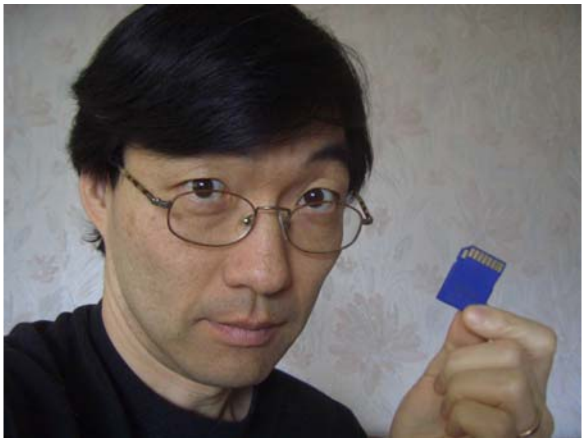
```
obj
  term : "isxtv"
  f : "sxfile"

pub Main
  term.str( string("Howdy, Doody") )
  term.out( 13 )

  f.Open( string("sphinx.bin"), "R" )
  f.Execute( 0 )


Filename: hello2.spn_
```

Editing a program



```
>
>
>
>
>
>
>
>
>
>cl hello2
>hello2
Howdy, Doody
>_
```

Compiling (and linking) a program, then running it



Michael Park with Sphinx on SD card