FlexProp Reference

5.9.14

Total Spectrum Software

07/22/2022

Contents

Flexspin

Flexspin is a compiler from the Spin language to assembly (PASM). It also now supports BASIC and C input.

Normally Spin is translated to a special kind of bytecode which is interpreted by a program in the Propeller's ROM. This bytecode is very compact, but it is slow to run. Flexspin produces much larger code, but it is also much faster. Flexspin also supports the Propeller 2 as well as the original Propeller chip.

The command line options for flexspin are very similar to those of the openspin compiler.

The ordinary usage is very simple:

```
flexspin program.spin
```

will compile `program.spin` into `program.binary`, which may then be loaded into the Propeller. There are many methods to do this; most IDEs (including the Propeller Tool) have a way to run a binary file. There are also command line tools (such as `propeller-load`) to run binaries. My workflow with Spin programs typically looks like:

```
emacs program.spin & # or use your favorite text editor
flexspin program.spin
propeller-load program.binary -r -t
```

Propeller 2 Support

flexspin supports the Propeller 2 instruction set (both rev A and rev B). To compile programs for Propeller 2, you can use the -2 option. Binaries may be downloaded to the board using Dave Hein's loadp2 program:

```
flexspin -2 program.spin2
loadp2 -b230400 program.binary -t
```

If the flexspin program is named something that ends in "spin2" (for example "flexspin2.exe") then it will use the -2 flag automatically. This may be more convenient for integration with IDEs.

Speed and Size

If you compile with flexspin, your binary program will be much larger than when compiled with openspin or bstc. That's because flexspin outputs native Propeller instructions (PASM) instead of Spin bytecode. It also means the flexspin compiled binary is much faster.

For example, the fftbench demo program compiled with

```
bstc -b -Oa fftbench.spin
```

is 3048 bytes long and runs in 1460 milliseconds. With

```
flexspin -O fftbench.spin
```

it is 4968 bytes long and runs in 170 milliseconds; so it is a bit less than twice as big and runs more than 8 times as fast.

The SPI test benchmark gives:

```
openspin -u: 11732816 cycles  796 bytes
bstc -Oa:    11699984 cycles  796 bytes
flexspin -O:    98848 cycles 2056 bytes
```

Spin wrappers

The simplest way to use flexspin is just to compile a whole program (convert everything to PASM). However, sometimes a program compiled this way may be too big to fit in memory; or sometimes you may want to convert some Spin module to PASM and make it easy to use in other Spin projects (which may be compiled with openspin or bstc).

flexspin may be used to convert a Spin object into PASM code that has Spin wrappers. This is achieved with the `-w` ("wrap") command line flag. The output is a generic Spin module with a `.cog.spin` extension. The wrapped Spin must fit in a single COG (so no LMM mode is used) and is designed basically for converting device drivers from Spin to PASM easily. All of the PUB functions of the original `.spin` will be available in in the `.cog.spin`, but instead of running Spin bytecode they will send a message to the PASM code (which must be running in another COG) for execution there. There will also be a `__cognew` method to start a COG up. `__cognew` must be called before any other methods.

Example

Suppose you have a file `Fibo.spin` that has:

```
PUB fibo(n)
  if (n < 2)
    return n
  return fibo(n-1) + fibo(n-2)
```

To use the Spin version of this you would do something like:

```
OBJ f : "Fibo"

PUB test
  answer1 := f.fibo(1)
  answer9 := f.fibo(9)
```

To convert this to COG PASM you would do:

```
flexspin -w Fibo.spin
```

which would produce `Fibo.cog.spin`; and you would modify your program to

```
OBJ f : "Fibo.cog"

PUB test
  f.__cognew '' start the COG PASM running
  answer1 := f.fibo(1)
  answer9 := f.fibo(9)
```

The usage is exactly the same, except that you have to insert the call to `__cognew` to start up the remote COG; but now the time critical `fibo` function will actually run in the other COG, as PASM code.

Command Line Options

There are various command line options which may modify the compilation:

```
  [ -h ]              display this help
  [ -L or -I <path> ] add a directory to the include path
  [ -o ]             output filename
  [ -b ]             output binary file format
  [ -e ]             output eeprom file format
  [ -c ]             output only DAT sections
  [ -l ]             output a .lst listing file
  [ -f ]             output list of file names
  [ -g ]             enable debug statements (default printf method)
  [ -gbrk ]          enable BRK based debugging
  [ -q ]             quiet mode (suppress banner and non-error text)
  [ -p ]             disable the preprocessor
  [ -O[#] ]          set optimization level
                       -O0 disable all optimization
                       -O1 apply default optimization (same as no -O flag)
```

```
             -O2 apply all optimization (same as -O)
  [ -D <define> ]    add a define
  [ -2 ]             compile for Prop2
  [ -w ]             produce Spin wrappers for PASM code
  [ -H nnnn ]        change the base HUB address (see below)
  [ -E ]             omit any coginit header
  [ --code=cog  ]    compile to run in COG memory instead of HUB
  [ --fcache=N  ]    set size of FCACHE space in longs (0 to disable)
  [ --fixed ]        use 16.16 fixed point instead of IEEE floating point
```

The `-2` option is new: it is for compiling for the Propeller 2.

`flexspin.exe` checks the name it was invoked by. If the name starts with the string "bstc" (case matters) then its output messages mimic that of the bstc compiler; otherwise it tries to match openspin's messages. This is for compatibility with Propeller IDE. For example, you can use flexspin with the PropellerIDE by renaming `bstc.exe` to `bstc.orig.exe` and then copying `flexspin.exe` to `bstc.exe`.

Changing Hub address

In P2 mode, you may want to change the base hub address for the binary. Normally P2 binaries start at the standard offset of `0x400`. But if you want, for example, to load a flexspin compiled program from TAQOZ or some similar program, you may want to start at a different address (TAQOZ uses the first 64K of RAM). To do this, you may use some combination of the `-H` and `-E` flags.

`-H nnnn` changes the base HUB address from `0x400` to `nnnn`, where `nnnn` is either a decimal number like `65536` or a hex number prefixed with `0x`. By default the binary still expects to be loaded at address 0, so it starts with a `coginit #0, ##nnnn` instruction and then zero padding until the hub start. To skip the `coginit` and padding, add the `-E` flag.

Example

To compile a program to start at address 65536 (at the 64K boundary), do:

```
flexspin -2 -H 0x10000 -E fibo.bas
```

Spin Language Extensions

flexspin supports a number of extensions to the Spin language, including a preprocessor (also found in some other Spin compilers), multiple values in return and assignments, conditional assignment, inline assembly, and abstract function pointers.

See the `spin.md` file (or `spin.pdf`) in the `docs` directory for more details.

BASIC and C support

If an input file ends in `.bas` it is compiled as a BASIC program. See the `basic.md` file (or `basic.pdf`) for details of the BASIC language supported by flexspin.

If an input file ends in `.c`, `.cc`, or `.cpp` it is compiled as a C program. See the `c.md` file for details of the C language support in flexspin.

Limitations

Beware when compiling P1 objects that contain PASM for P2: some instructions have changed in subtle ways.

Programs compiled with flexspin will always be larger than those compiled with openspin or bstc. If the spin code is mostly PASM (as is the case with, for example, PropBASIC compiler output) then the flexspin overhead will be relatively small and probably fixed, but for large programs with lots of Spin methods the difference could be significant.

License