Parallax Propeller

The **Parallax P8X32A Propeller** chip, introduced in 2006, is a multi-core architecture parallel microcontroller with eight 32-bit RISC CPU cores.[1][2]

Parallax Propeller in Dual in-line package

The Parallax Propeller microcontroller, Propeller Assembly language, and Spin interpreter were designed by one person, Parallax's co-founder and president Chip Gracey. The Spin Programming language and "Propeller Tool" integrated development environment were designed by Chip Gracey and Parallax's software engineer Jeff Martin.

Multi-core architecture

Each of the eight 32-bit cores (called a **cog**) has a CPU which has access to 512 32-bit long words (2 kB) of instructions and data. Self-modifying code is possible and is used internally, for example by an instruction that is used to create a subroutine call/return mechanism without the need for a stack. Access to shared memory (32 kB RAM; 32 kB ROM) is controlled in round-robin fashion by an internal bus controller called the **hub**. Each cog also has access to two dedicated hardware counters and two special "video registers" for use in generating PAL, NTSC, VGA, servo-control, or other timing signals.[3]

Speed and power management

The Propeller can be clocked using either an internal, on-chip oscillator (providing a lower total parts count, but sacrificing some accuracy and thermal stability) or an external crystal or resonator (providing higher maximum speed with greater accuracy at an increased total cost). Only the external oscillator may be run through an on-chip PLL clock multiplier, which may be set at 1x, 2x, 4x, 8x, or 16x.

Both the on-board oscillator frequency (if used) and the PLL multiplier value may be changed at run-time. If used correctly, this can improve power efficiency; for example, the PLL multiplier can be decreased before a long "no operation" wait required for timing purposes, then increased afterwards, causing the processor to use less power. However, the utility of this technique is limited to situations where no other cog is executing timing-dependent code (or is carefully designed to cope with the change), since the effective clock rate is common to all cogs.

The effective clock rate ranges from 32 kHz up to 80 MHz (with the exact values available for dynamic control dependent on the configuration used, as described above). When running at 80 MHz, the proprietary interpreted **Spin programming language** executes approximately 80,000 instruction-tokens per second on each core, giving 8 times 80,000 for 640,000 high-level instructions per second. Most machine-language instructions take 4 clock-cycles to execute, resulting in 20 MIPS per cog, or 160 MIPS in total for an 8-cog Propeller.

In addition to lowering the clock rate to that actually required, power consumption can be reduced by turning off cogs (which then use very little power), and by reconfiguring I/O pins which are not needed, or can be safely placed in a high-impedance state ("tristated"), as inputs. Pins can be reconfigured dynamically, but again, the change applies to all cogs, so synchronization is important for certain designs. (Some protection is available for situations where one core attempts to use a pin as an output while another attempts to use it as an input; this is explained in Parallax's technical reference manual.)

On-board peripherals

Each cog has access to some dedicated counter/timer hardware, and a special timing signal generator intended to simplify the design of video output stages, such as composite PAL or NTSC displays (including modulation for broadcast) and VGA monitors. Parallax thus makes sample code available which can generate video signals (text and somewhat low-resolution graphics) using a minimum parts count consisting of the Propeller, a crystal oscillator, and a few resistors to form a crude DAC. The frequency of the oscillator is important, as the correction ability of the video timing hardware is limited to the clock rate. It is possible to use multiple cogs in parallel to generate a single video signal. More generally, the timing hardware can be used to implement various pulse-width modulated (PWM) timing signals.

ROM extensions

In addition to the Spin interpreter and a bootloader, the built-in ROM provides some data which may be useful for certain sound, video, or mathematical applications:

- a bitmap font is provided, suitable for typical character generation applications (but not customizable);
- a logarithm table (base 2, 2048 entries);
- an antilog table (base 2, 2048 entries); and
- a sine table (16-bit, 2049 entries representing first quadrant, angles from 0 to π/2; other three quadrants are created from the same table).

The math extensions are intended to help compensate for the lack of a floating-point unit as well as more primitive missing operations, such as multiplication and division (this is masked in Spin but is a limitation for assembly language routines). The Propeller is a 32-bit processor, however, and these tables may not have sufficient accuracy for higher-precision applications.

Built in SPIN byte code interpreter

Spin is a multitasking high-level computer programming language created by Parallax's Chip Gracey, who also designed the Propeller microcontroller on which it runs, for their line of Propeller microcontrollers.[4]

Spin code is written on the Propeller Tool, a GUI-oriented software development platform written for Windows XP.[5] This compiler converts the Spin code into bytecodes that can be loaded (with the same tool) into the main 32 kB RAM, and optionally into the serial boot FLASH EEPROM, of the Propeller chip. After booting the propeller a bytecode interpreter is copied from the built in ROM into the 2 kB RAM of the primary COG. This COG will then start interpreting the bytecodes in the main 32 kB RAM. More than one copy of the bytecode interpreter can run in other COGs, so several Spin code threads can run simultaneously. Within a Spin code program, assembler code program(s) can be "inline" inserted. These assembler program(s) will then run on their own COGs.

Like Python, Spin uses indentation/whitespace, rather than curly braces or keywords, to delimit blocks.

The Propeller's interpreter for its proprietary multi-threaded SPIN computer language is a byte code interpreter. This interpreter decodes strings of instructions, one instruction per byte, from user code which has been edited, compiled, and loaded onto the Propeller from within a purpose-specific IDE. This IDE, which Parallax simply calls "The Propeller tool", is intended for use under the Windows operating system.

The SPIN language is a high-level language. Because it is interpreted in software, it runs slower than pure Propeller assembly but can be more space-efficient (Propeller assembly opcodes are 32 bits long; SPIN directives are 8 bits long, which may be followed by a number of 8-bit bytes to specify how that directive operates). SPIN also allows users to avoid significant memory segmentation issues that must be considered for assembly code.

At startup, a copy of the byte code interpreter (less than 2 kB in size), will be copied into the dedicated RAM of a cog and will then start interpreting byte code in the main 32 KB RAM. Additional cogs can be started from that

point, loading a separate copy of the interpreter into the new cog's dedicated RAM (a total of eight interpreter threads can, therefore, run simultaneously). Notably, this means that at least a minimal amount of startup code *must* be SPIN code, for all Propeller applications.

Syntax

The syntax of Spin can be broken down into blocks. The blocks are as following.

- *VAR* Holds global variables
- *CON* Holds program constants
- *PUB* Holds code for a public subroutine
- *PRI* Holds code for a private subroutine
- *OBJ* Holds code for objects
- *DAT* Holds predefined data, memory reservations and assembly code

Example keywords

- *reboot:* causes the microcontroller to reboot
- *waitcnt:* wait for the system counter to equal or exceed a specified value
- *waitvid:* Waits for a (video) timing event before outputting (video) data to I/O pins.
- *coginit:* starts a processor on a new task

Example program

An example program, (as it would appear in the "Propeller Tool" editor) which outputs the current system counter every 3,000,000 cycles, then is shut down by another cog after 40,000,000 cycles:

```
VAR
  long Stk[3] ''declare an array of longs

PUB Main
  cognew(DispCnt, @Stk) ''Acvtivate cog 1 and run the DispCnt routine in it
  ''also pass the adress of the array we created, for use as a call stack
  Waitandstop ''Run Wait routine in this cog (cog 0)

PUB DispCnt
  dira~~ ''set all bits in port a direction register to 1 (output)
  repeat
    waitcnt(3_000_000 + cnt)
    outa := cnt ''move value current system counter to port a

PUB Waitandstop
  waitcnt(40_000_000 + cnt) ''wait until counter = current value + 40,000,000 (wait 40mil clocks)
  cogstop(1) ''stop cog 1
  cogstop(0) ''stop cog 0
```

The Parallax Propeller is gradually accumulating software libraries which give it similar functionality to Parallax's older BASIC Stamp product; however there is no uniform list of which PBASIC facilities now have Spin equivalents.

Package and I/O

The initial version of the chip (called the P8X32) provides one 32-bit port in a 40-pin 0.6" DIP, 44-pin LQFP, or QFN package. Of the 40 available pins, 32 are used for I/O, four for power and ground pins, two for an external crystal (if used), one to enable brownout-detection, and one for reset.

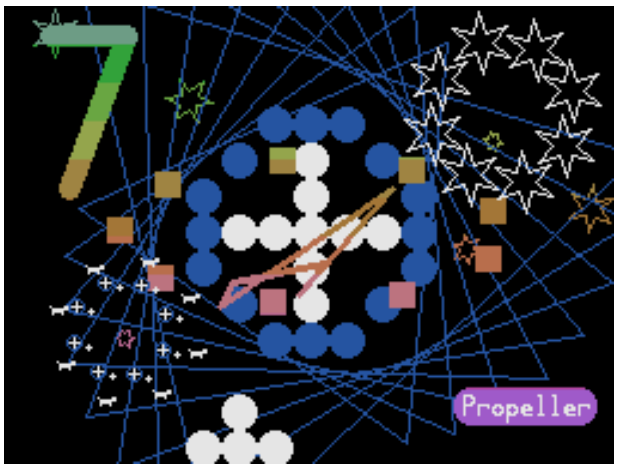
All 8 cores can access the 32-bit port (designated "A"; there is currently no "B") simultaneously. A special control mechanism is used to avoid I/O conflicts if one core attempts to use an I/O pin as an output while another tries to use it as input. Any of these pins can be used for the timing or pulse-width modulation output techniques described above.

Parallax has stated that it expects later versions of the Propeller to offer more I/O pins and/or more memory.[6]

Virtual I/O devices

The Propeller's designers designed it around the concept of "virtual I/O devices". For example, the "HYDRA Game Development Kit", (a computer system designed for hobbyists, to learn to develop "retro-style" video games) uses the built-in character generator and video support logic to generate a virtual Video display generator that outputs VGA colour pictures, PAL/NTSC compatible colour pictures or broadcast RF video+audio in software.[7]

The screen capture displayed here was made using a software "virtual display driver" that sends the pixel data over a serial link to a PC.[8]



Screen capture of the graphics demo that Parallax created to demonstrate the NTSC video library

Software libraries are available to implement several I/O devices ranging from simple UARTs and Serial I/O interfaces such as SPI, I2C and PS/2 compatible serial mouse and keyboard interfaces, motor drivers for robotic systems, MIDI interfaces and LCD controllers.[9]

Dedicated cores instead of interrupts

The design philosophy of the Propeller is that a hard real-time multi-core architecture negates the need for dedicated interrupt hardware and support in assembly. In traditional CPU architecture, external interrupt lines are fed to an on-chip interrupt controller and are serviced by one or more interrupt service routines. When an interrupt occurs, the interrupt controller suspends normal CPU processing and saves internal state (typically on the stack), then vectors to the designated interrupt service routine. After handling the interrupt, the service routine executes a "return from interrupt" instruction which restores the internal state and resumes CPU processing.

To handle an external signal promptly on the Propeller, any one of the 32 I/O lines is configured as an input. A cog is then configured to wait for a transition (either positive or negative edge) on that input using one of the two counter circuits available to each cog. While waiting for the signal, the cog operates in low-power mode, essentially sleeping. Extending this technique, a Propeller can be set up to respond to eight independent "interrupt" lines with essentially zero handling delay. Alternately, a single line can be used to signal the "interrupt" and then additional input lines can be read to determine the nature of the event. The code running in the other cores is not affected by the interrupt handling cog. Unlike a traditional multitasking single-processor interrupt architecture, the signal response timing remains predictable,[10] and indeed using the term "interrupt" in this context can cause confusion, since this functionality can be more properly thought of as polling with a zero loop time.

Boot mechanism

On power up, brownout detection, software reset, or external hardware reset, the Propeller will load a machine-code boot routine from the internal ROM into the RAM of its first (primary) cog and execute it. This code emulates an I^2C interface in software, temporarily using two I/O pins for the needed serial clock and data signals to load user code from an external I^2C EEPROM.

Simultaneously, it emulates a serial port, using two other I/O pins that can be used to upload software directly to RAM (and optionally to the external EEPROM). If the Propeller does not see any commands from the serial port, it will load the user program (the entry code of which must be written in SPIN, as described above) from the serial EEPROM into the main 32kB RAM. After that it will load the SPIN interpreter from its built-in ROM into the dedicated RAM of its first cog, thereby overwriting most of the bootloader.

Regardless of how the user program is loaded, execution starts by interpreting initial user bytecode with the SPIN interpreter running in the primary cog. After this initial SPIN code runs, the application can turn on any unused cog to start a new thread, and/or start assembler code routines.

External persistent memory

The Propeller boots from an external serial EEPROM; once the boot sequence completes, this device may be accessed as an external peripheral.[11]

Other language implementations

Apart from Spin and the Propeller's low-level assembler, a number of other languages have been ported to it.

C compiler

Parallax supports Propeller-GCC which is a port of the GCC C/C++ compiler for Propeller [12] (branch release_1_0). The C compiler and the C Library are ANSI C compliant. The C++ compiler is ANSI-C99 compliant. Full C++ is supported with external memory. The SimpleIDE program[13] provides users a simple way to write programs without requiring makefiles. In 2013 Parallax incorporated Propeller-GCC and Simple Libraries into the Propeller-C Learn series of tutorials.[14] Propeller-GCC is actively maintained. Propeller-GCC and SimpleIDE are officially supported Parallax software products.

The ImageCraft ICCV7 for Propeller C compiler has been marked to End-Of-Life state.[15]

A free ANSI C compiler called Catalina is available.[16] It is based on LCC. Catalina is actively maintained.

FORTH on the Propeller

There are at least 6 different versions of FORTH available for the Propeller, both paid and Open Source.

PropForth

A free version that enjoys extensive development and community support is PropForth[17] PropForth is tailored to the prop architecture, and necessarily deviates from any general standard regarding architectural uniqueness, consistent with the concept of forth.

In addition to the FORTH interpreter, PropForth provides many features that take advantage of the chip's capabilities. "Linked I/O" refers to the method of associating a stream with process, allowing one process to link to the next on the fly, transparent to the application. This can reduce or eliminate the need of a hardware debugging or JTAG interface in many cases. "Mutli-Channel Synchronous Serial (MCS)" refers to the synchronous serial communication between prop chips. 96-bit packets are sent continuously between two cogs, the result is that applications see additional resources (+6 cogs for each prop chip added) with little or no impact on throughput for a

well constructed application.

"LogicAnalyzer" refers to an extension package that implements software logic analyzer. EEPROMfilesystem and SDfilesystem are extensions that implement rudimentary storage using EEPROM and SD flash.

"PagedAssembler" refers to the package of optimizations that allow assembler routines to be swapped in (and out by overwrite) on the fly, allowing virtually unlimited application size. Script execution allows extensions to be loaded on the fly, allowing program source up to the size of storage media.

Propeller and Java

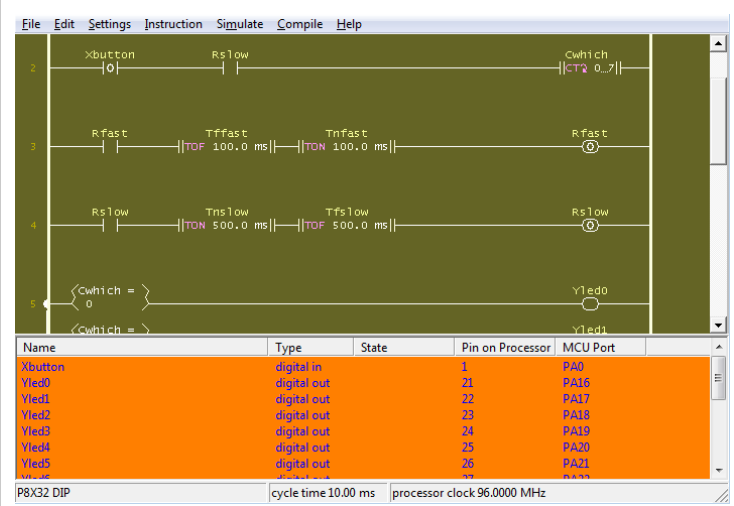
There is also a movement in place to put the JVM on Propeller. A compiler, debugger, and emulator are being developed.[18]

Pascal compiler and runtime

A large subset of Pascal is implemented by a compiler and interpreter based on the P4 system.[19]

Graphical Programming

PICoPLC supports output to Propeller processor. The program is created in a GUI ladder logic editor and resulting code is emitted as SPIN source. PICoPLC also supports P8X32 with create-simulate-run feature. No restrictions on target hardware as the oscillator frequency and IO pins are freely configurable in the ladder editor. PICoPLC is no longer available on the developer website (HTTP 404).



Screen capture of the PICoPLC ladder editor

Future versions

Parallax is currently building a new Propeller[20] with cogs that each will run at about 160 MIPS, whereas the current Propeller's cogs each run at around 20 MIPS. The improved performance would result from a maximum clock speed increase to 160 MHz (from 80 MHz) and an architecture that pipelines instructions, achieving an average execution of nearly one instruction per clock cycle (approximately a four-fold increase).

References

[1] makezine.com (http://blog.makezine.com/archive/2006/02/first_look_at_parallaxs_propel.html)

[2] makezine.com (http://blog.makezine.com/2006/08/21/parallax-propeller-review/)

[3] electronicdesign.com (http://electronicdesign.com/Articles/Index.cfm?AD=1&ArticleID=13329)

[4] David A. Scanlan, Martin A. Hebel. "Programming the eight-core propeller chip" *Journal of Computing Sciences in Colleges*, Volume 23, Issue 1, October 2007. (http://portal.acm.org/citation.cfm?id=1289314)

[5] propeller.wikispaces.com (http://propeller.wikispaces.com/Prop+Tool)

[6] Parallax Forums (http://forums.parallaxinc.com/forums/default.aspx?f=25&m=156993)

[7] selmaware.com (http://www.selmaware.com/propeller_video_app_board.htm); a dedicated video generator board with a propeller

[8] screen capture software (http://forums.parallaxinc.com/forums/pr.aspx?f=33&m=198032)

[9] parallax.com (http://obex.parallax.com/); propeller object exchange software library

[10] propeller wikispaces.com (http://propeller.wikispaces.com/Interrupts)

[11] circuitcellar.com (http://www.circuitcellar.com/library/print/0806/Cantrell193/2.htm)

[12] PropGCC on Google Code (http://code.google.com/p/propgcc/)

[13] SimpleIDE (http://learn.parallax.com/node/640)

[14] Propeller C Learning System (http://learn.parallax.com/propeller-c)

[15] parallax.com (http://www.parallax.com/Store/Microcontrollers/PropellerTools/tabid/143/ProductID/526/List/0/Default.aspx?SortField=ProductName,ProductName)

[16] Catalina - a C compiler for the Propeller (http://forums.parallaxinc.com/forums/default.aspx?f=25&m=388930)

[17] google.com (http://code.google.com/p/propforth/); propforth

[18] Programming Propeller in Java (http://propeller.wikispaces.com/Programming+in+Java)

[19] http://propeller.wikispaces.com/Programming+in+Pascal

[20] parallax.com (http://www.parallax.com/Propeller2FeatureList/tabid/898/Default.aspx)

External links

- Wiki with detailed information about the propeller (http://propeller.wikispaces.com/)
- Propeller forum at Parallax Inc: (http://forums.parallax.com/forumdisplay.php?f=65)
- Propeller Page at Parallax Inc: (http://www.parallax.com/microcontrollers/propeller)
- Propeller GCC Beta Site (https://sites.google.com/site/propellergcc/)
- Article at EiED online (http://www.electronicdesign.com/Articles/Index.cfm?AD=1&ArticleID=13329)
- a second article at EiED online (http://www.electronicdesign.com/Articles/Index.cfm?AD=1&AD=1&ArticleID=12235)
- An article at ferret.com.au (http://www.ferret.com.au/c/R-T-Nollet/Eight-cogs-in-one-chip-n677148)
- List of programming languages running on the Propeller (http://forums.parallax.com/showthread.php?113091)
- Download PICoPLC from APStech (http://www.apstech.hu/download)
- FirstSpin, a weekly educational audio program about the Spin programming language and the Propeller, sponsored by Parallax (http://FirstSpin.tv/)

Article Sources and Contributors

Parallax Propeller *Source*: http://en.wikipedia.org/w/index.php?oldid=588308449 *Contributors*: ASarnat, AbstractBeliefs, Andy Dingley, Apstech.hu, Asher196, Azlandavid, BillyPreset, Bradams, Breakpoint, CesarB, Chuckwolber, Circuitsoft, Crimer, Cybercobra, Davehein, Dedicated96, Ds13, EagleFan, EngineeringCat, Faradayplank, FlyByPC, Fnagaton, Gbruin, Genejockey, Greenrd, Guy Macon, Imroy, Inkling, Intgr, Ipsign, JLaTondre, Jerryobject, Joe07734, Jtcook, Kozuch, Leedude, Loadmaster, Lproven, Mahjongg, MarkMLl, Mblumber, Mgilliland18, Miguel.mateo, Mortense, NZR, Neelix, Norwegian Gadgetman, Nzrihen, Pcap, PerryTachett, Peter L, Psychlohexane, Puchiko, R'n'B, Rabit, Rilak, Rjwilmsi, Smalljim, Sp33dyphil, Teraflop122, TheParallax, Tom d perkins, Tony Sidaway, Val42, Woohookitty, YUL89YYZ, 122 anonymous edits

Image Sources, Licenses and Contributors

File:Propeller Chip.JPG *Source*: http://en.wikipedia.org/w/index.php?title=File:Propeller_Chip.JPG *License*: Creative Commons Attribution-Sharealike 2.5 *Contributors*: MOS6502

File:Example SPIN program.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Example_SPIN_program.png *License*: Public Domain *Contributors*: AbstractBeliefs

File:prop grap demo.gif *Source*: http://en.wikipedia.org/w/index.php?title=File:Prop_grap_demo.gif *License*: GNU Free Documentation License *Contributors*: Jtcook

File:PICoPLC ladder editor v3.0 screenshot.png *Source*: http://en.wikipedia.org/w/index.php?title=File:PICoPLC_ladder_editor_v3.0_screenshot.png *License*: unknown *Contributors*: Attila,Kolinger

License