Propeller™ P8X32A Preliminary Datasheet

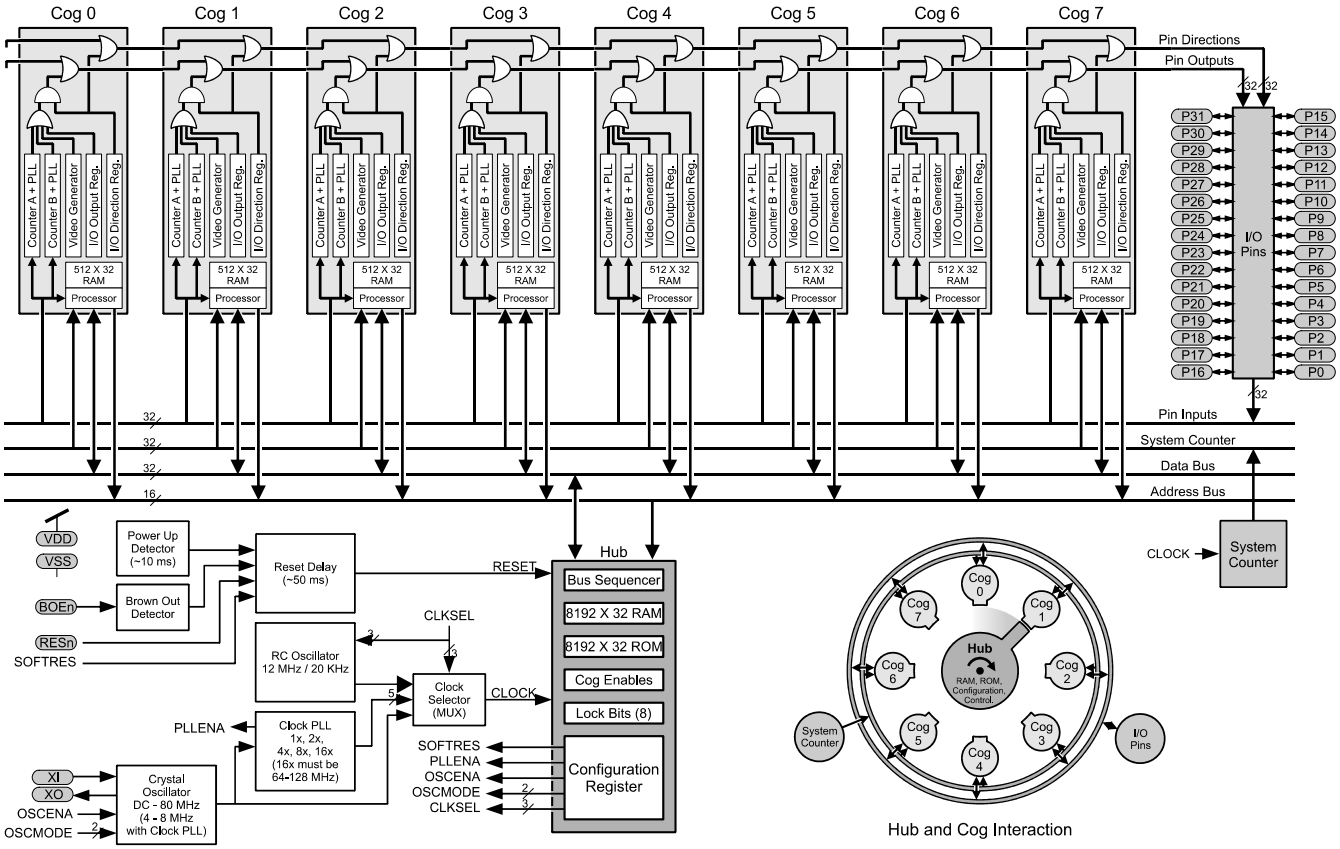
8-Cog Multiprocessor Microcontrollers

1.0 PRODUCT OVERVIEW

1.1. Introduction

The Propeller chip is designed to provide high-speed processing for embedded systems while maintaining low current consumption and a small physical footprint. In addition to being fast, the Propeller chip provides flexibility and power through its eight processors, called cogs, that can perform simultaneous tasks independently or cooperatively, all while maintaining a relatively simple architecture that is easy to learn and utilize. Two programming languages are available: Spin (a high-level object-based language) and Propeller Assembly. Both include custom commands to easily manage the Propeller chip's unique features.

Figure 1: Propeller P8X32A Block Diagram



1.2. Stock Codes

Table 1: Propeller Chip Stock Codes

Device Stock #	Package Type	I/O Pins	Power Requirements	External Clock Speed	Internal RC Oscillator	Internal Execution Speed	Global ROM/RAM	Cog RAM
P8X32A-D40	40-pin DIP	32 CMOS	3.3 volts DC	DC to 80 MHz	12 MHz or 20 KHz*	0 to 160 MIPS (20 MIPS/cog)	64 K bytes; 32768 bytes ROM / 32768 bytes RAM	512 x 32 bits per cog
P8X32A-Q44	44-pin LQFP							
P8X32A-M44	44-pin QFN							

***Approximate; may range from 8 MHz – 20 MHz, or 13 kHz – 33 kHz, respectively.**

1.3. Key Features

The design of the Propeller chip frees application developers from common complexities of embedded systems programming. For example:

- Eight processors (cogs) perform simultaneous processes independently or cooperatively, sharing common resources through a central hub. The Propeller application designer has full control over how and when each cog is employed; there is no compiler-driven or operating system-driven splitting of tasks between multiple cogs. This method empowers the developer to deliver absolutely deterministic timing, power consumption, and response to the embedded application.
- Asynchronous events are easier to handle than with devices that use interrupts. The Propeller has no need for interrupts; just assign some cogs to individual, high-bandwidth tasks and keep other cogs free and unencumbered. The result is a more responsive application that is easier to maintain.
- A shared System Clock allows each cog to maintain the same time reference, allowing true synchronous execution.

1.4. Programming Advantages

- The object-based high-level Spin language is easy to learn, with special commands that allow developers to quickly exploit the Propeller chip's unique and powerful features.
- Propeller Assembly instructions provide conditional execution and optional flag and result writing for each individual instruction. This makes critical, multi-decision blocks of code more consistently timed; event handlers are less prone to jitter and developers spend less time padding, or squeezing, cycles.

1.5. Applications

The Propeller chip is particularly useful in projects that can be vastly simplified with simultaneous processing, including:

- Industrial control systems
- Sensor integration, signal processing, and data acquisition
- Handheld portable human-interface terminals
- Motor and actuator control
- User interfaces requiring NTSC, PAL, or VGA output, with PS/2 keyboard and mouse input
- Low-cost video game systems
- Industrial, educational or personal-use robotics
- Wireless video transmission (NTSC or PAL)

1.6. Programming Platform Support

Parallax Inc. supports the Propeller chip with a variety of hardware tools and boards:

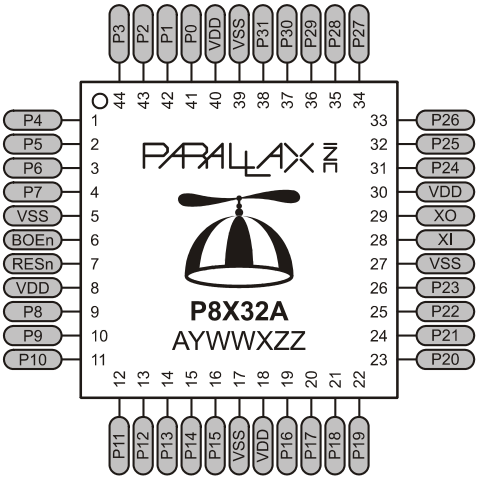
- Prop Clip (#32200) and Prop Plug (#3220). These boards provide convenient programming port connections, see the Typical Connection Diagrams on Page 4.
- The Propeller Demo Board (Stock #32100) provides a convenient means to test-drive the Propeller chip's varied capabilities through a host of device interfaces on one compact board. The schematic is provided on page 19. Main features:
  - P8X32A-Q44 Propeller Chip
  - 24LC256-I/ST EEPROM for program storage
  - Replaceable 5.000 MHz crystal
  - 3.3 V and 5 V regulators with on/off switch
  - USB-to-serial interface for programming and communication
  - VGA and TV output
  - Stereo output with 16-ohm headphone amplifier
  - Electret microphone input
  - Two PS/2 mouse and keyboard I/O connectors
  - 8 LEDs (share VGA pins)
  - Pushbutton for reset
  - Big ground post for scope hookup
  - I/O pins P0-P7 are free and brought out to header
  - Breadboard for custom circuits
- The Propeller Proto Board (#32212) features a surface-mount Propeller chip with the necessary components to achieve a programming interface, with pads ready for a variety of I/O connectors and DIP/SIP chips, and a generous through-hole prototyping area.
- The PropSTICK USB (#32210) features a Propeller chip, EEPROM, 3.3VDC and 5VDC regulators, reset button, crystal and USB connection on a 0.6" wide DIP package for easy prototyping on perfboard and breadboard.

1.7. Corporate and Community Support

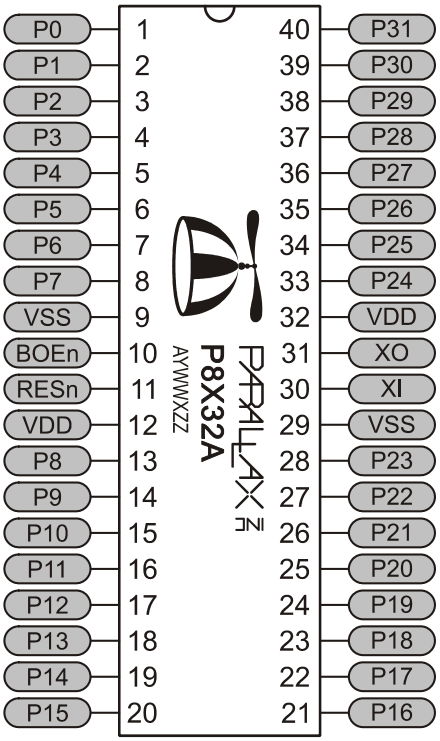
- Parallax provides technical support free of charge. In the Continental US, call toll free (888) 512-1024; from outside please call (916) 624-8333. Or, email: support@parallax.com.
- Parallax hosts a moderated public users forum just for the Propeller: http://forums.parallax.com/forums.
- Browse through community-created Propeller objects and share yours with others via Parallax-hosted Propeller Object Exchange Library: look for the link on the http://www.parallax.com/downloads page.

CONNECTION DIAGRAMS

1.8. Pin Assignments



LQFP and QFN Packages



DIP Package

1.9. Pin Descriptions

Table 2: Pin Descriptions

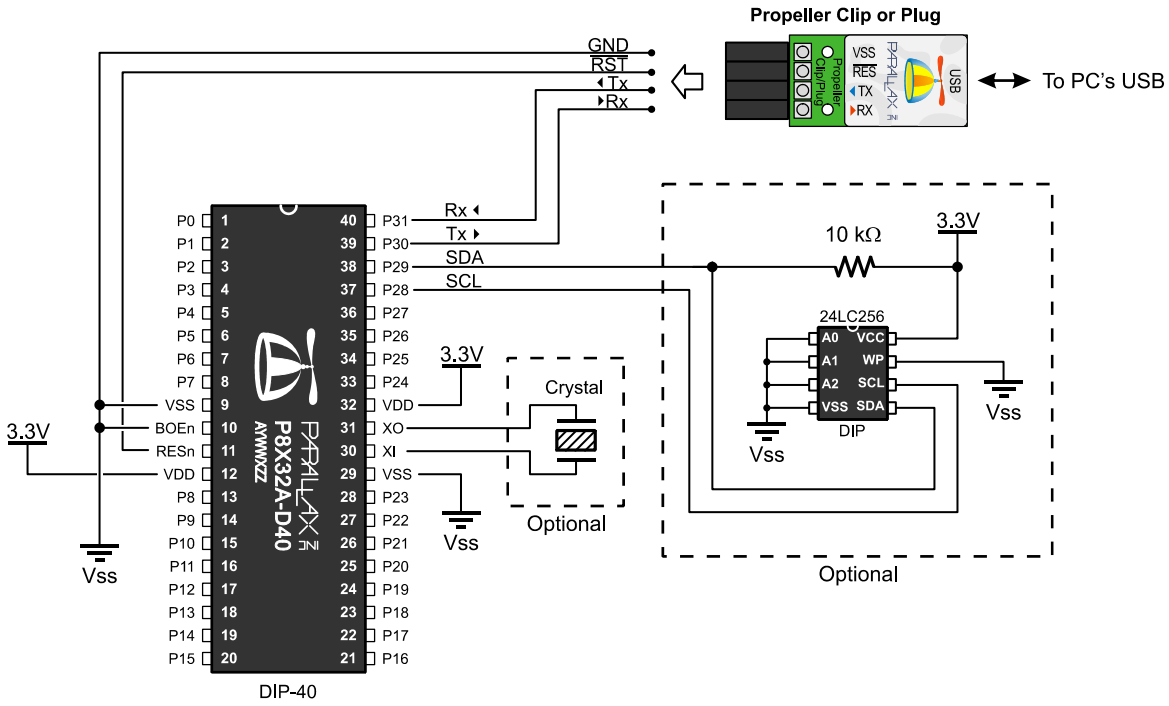
Pin Name	Direction	Description
P0 – P31	I/O	General purpose I/O Port A. Can source/sink 40 mA each at 3.3 VDC. CMOS level logic with threshold of ≈ ½ VDD or 1.6 VDC @ 3.3 VDC. The pins shown below have a special purpose upon power-up/reset but are general purpose I/O afterwards. P28 - I2C SCL connection to optional, external EEPROM. P29 - I2C SDA connection to optional, external EEPROM. P30 - Serial Tx to host. P31 - Serial Rx from host.
VDD	---	3.3 volt power (2.7 – 3.6 VDC)
VSS	---	Ground
BOEn	I	Brown Out Enable (active low). Must be connected to either VDD or VSS. If low, RESn becomes a weak output (delivering VDD through 5 KΩ) for monitoring purposes but can still be driven low to cause reset. If high, RESn is CMOS input with Schmitt Trigger.
RESn	I/O	Reset (active low). When low, resets the Propeller chip: all cogs disabled and I/O pins floating. Propeller restarts 50 ms after RESn transitions from low to high.
XI	I	Crystal Input. Can be connected to output of crystal/oscillator pack (with XO left disconnected), or to one leg of crystal (with XO connected to other leg of crystal or resonator) depending on CLK Register settings. No external resistors or capacitors are required.
XO	O	Crystal Output. Provides feedback for an external crystal, or may be left disconnected depending on CLK Register settings. No external resistors or capacitors are required.

1.10. Typical Connection Diagrams

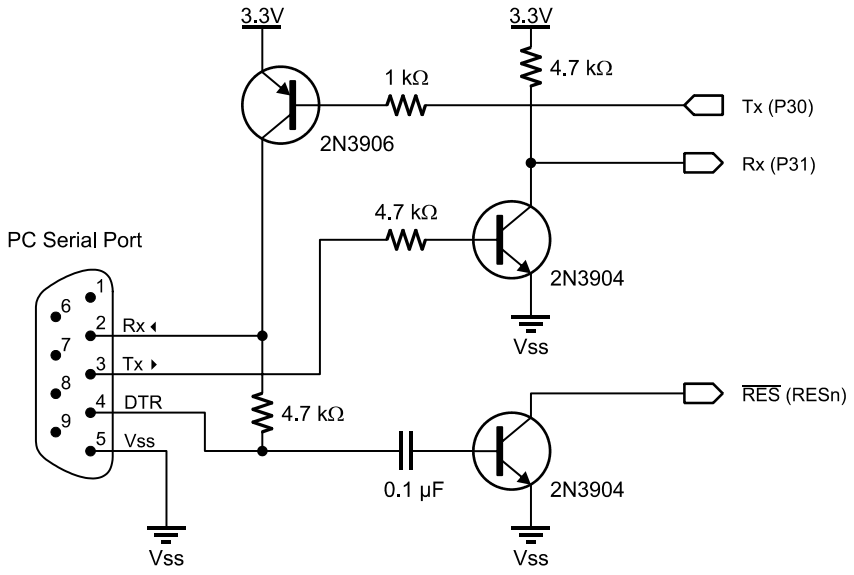
1.10.1. Propeller Clip or Propeller Plug Connection - Recommended

Note that the connections to the external oscillator and EEPROM, which are enclosed in dashed lines, are optional.

Propeller Clip: Stock #32200; Propeller Plug: Stock #32201. The Propeller Clip/Plug schematic is available for download from www.parallax.com.



1.10.2. Alternative Serial Port Connection



2.0 OPERATING PROCEDURES

2.1. Boot-Up Procedure

Upon power-up, or reset:

1. The Propeller chip's internal RC oscillator begins running at 20 kHz, then after a 50 ms reset delay, switches to 12 MHz. Then the first processor (Cog 0) loads and runs the built-in Boot Loader program.
2. The Boot Loader performs one or more of the following tasks, in order:
   a. Detects communication from a host, such as a PC, on pins P30 and P31. If communication from a host is detected, the Boot Loader converses with the host to identify the Propeller chip and possibly download a program into global RAM and optionally into an external 32 KB EEPROM.
   b. If no host communication was detected, the Boot Loader looks for an external 32 KB EEPROM on pins P28 and P29. If an EEPROM is detected, the entire 32 KB data image is loaded into the Propeller chip's global RAM.
   c. If no EEPROM was detected, the boot loader stops, Cog 0 is terminated, the Propeller chip goes into shutdown mode, and all I/O pins are set to inputs.
3. If either step 2a or 2b was successful in loading a program into the global RAM, and a suspend command was not given by the host, then Cog 0 is reloaded with the built-in Spin Interpreter and the user code is run from global RAM.

2.2. Run-Time Procedure

A Propeller Application is a user program compiled into its binary form and downloaded to the Propeller chip's RAM/EEPROM. The application consists of code written in the Propeller chip's Spin language (high-level code) with optional Propeller Assembly language components (low-level code). Code written in the Spin language is interpreted during run time by a cog running the Spin Interpreter while code written in Propeller Assembly is run in its pure form directly by a cog. Every Propeller Application consists of at least a little Spin code and may actually be written entirely in Spin or with various amounts of Spin and assembly. The Propeller chip's Spin Interpreter is started in Step 3 of the Boot Up Procedure, above, to get the application running.

Once the boot-up procedure is complete and an application is running in Cog 0, all further activity is defined by the application itself. The application has complete control over things like the internal clock speed, I/O pin usage, configuration registers, and when, what and how many cogs are running at any given time. All of this is variable at run time, as controlled by the application.

2.3. Shutdown Procedure

When the Propeller goes into shutdown mode, the internal clock is stopped causing all cogs to halt and all I/O pins are set to input direction (high impedance). Shutdown mode is triggered by one of the three following events:

1. VDD falling below the brown-out threshold (~2.7 VDC), when the brown out circuit is enabled,
2. the RESn pin going low, or
3. the application requests a reboot (see the REBOOT command in the Propeller Manual).

Shutdown mode is discontinued when the voltage level rises above the brown-out threshold and the RESn pin is high.

3.0 SYSTEM ORGANIZATION

3.1. Shared Resources

There are two types of shared resources in the Propeller: 1) common, and 2) mutually-exclusive. Common resources can be accessed at any time by any number of cogs. Mutually-exclusive resources can also be accessed by any number of cogs, but only by one cog at a time. The common resources are the I/O pins and the System Counter. All other shared resources are mutually-exclusive by nature and access to them is controlled by the Hub. See Section 0 on page 6.

3.2. System Clock

The System Clock (shown as "CLOCK" in Figure 1, page 1) is the central clock source for nearly every component of the Propeller chip. The System Clock's signal comes from one of three possible sources:

- The internal RC oscillator (~12 MHz or ~20 kHz)
- The XI input pin (either functioning as a high-impedance input or a crystal oscillator in conjunction with the XO pin)
- The Clock PLL (phase-locked loop) fed by the XI input

The source is determined by the CLK register's settings, which is selectable at compile time and reselectable at run time. The Hub and internal Bus operate at half the System Clock speed.

3.3. Cogs (processors)

The Propeller contains eight (8) identical, independent processors, called cogs, numbered 0 to 7. Each cog contains a Processor block, local 2 KB RAM configured as 512 longs (512 x 32 bits), two advanced counter modules with PLLs, a Video Generator, I/O Output Register, I/O Direction Register, and other registers not shown in the Block Diagram.

All eight cogs are driven from the System Clock; they each maintain the same time reference and all active cogs execute instructions simultaneously. They also all have access to the same shared resources.

Cogs can be started and stopped at run time and can be programmed to perform tasks simultaneously, either independently or with coordination from other cogs through Main RAM. Each cog has its own RAM, called Cog RAM, which contains 512 registers of 32 bits each. The Cog RAM is all general purpose RAM except for the last 16 registers, which are special purpose registers, as described in Table 6 on page 10.

3.4. Hub

To maintain system integrity, mutually-exclusive resources must not be accessed by more than one cog at a time. The Hub controls access to mutually-exclusive resources by giving each cog a turn in a "round robin" fashion from Cog 0 through Cog 7 and back to Cog 0 again. The Hub and its bus run at half the System Clock rate, giving a cog access to mutually-exclusive resources once every 16 System Clock cycles. Hub instructions, the Propeller Assembly instructions that access mutually-exclusive resources, require 7 cycles to execute but they first need to be synchronized to the start of the Hub Access Window.

It takes up to 15 cycles (16 minus 1, if we just missed it) to synchronize to the Hub Access Window plus 7 cycles to execute the hub instruction, so hub instructions take from 7 to 22 cycles to complete.

Figure 2 and Figure 3 show examples where Cog 0 has a hub instruction to execute. Figure 2 shows the best-case scenario; the hub instruction was ready right at the start of that cog's access window. The hub instruction executes immediately (7 cycles) leaving an additional 9 cycles for other instructions before the next Hub Access Window arrives.

Figure 3 shows the worst-case scenario; the hub instruction was ready on the cycle right after the start of Cog 0's access window; it just barely missed it. The cog waits until the next Hub Access Window (15 cycles later) then the hub instruction executes (7 cycles) for a total of 22 cycles for that hub instruction. Again, there are 9 additional cycles after the hub instruction for other instructions to execute before the next Hub Access Window arrives. To get the most efficiency out of Propeller Assembly routines that have to frequently access mutually-exclusive resources, it can be beneficial to interleave non-hub instructions with hub instructions to lessen the number of cycles waiting for the next Hub Access Window. Since most Propeller Assembly instructions take 4 clock cycles, two such instructions can be executed in between otherwise contiguous hub instructions.

Keep in mind that a particular cog's hub instructions do not, in any way, interfere with other cogs' instructions because of the Hub mechanism. Cog 1, for example, may start a hub instruction during System Clock cycle 2, in both of these examples, possibly overlapping its execution with that of Cog 0 without any ill effects. Meanwhile, all other cogs can continue executing non-hub instructions, or awaiting their individual hub access windows regardless of what the others are doing.

Figure 2: Cog-Hub Interaction – Best Case Scenario

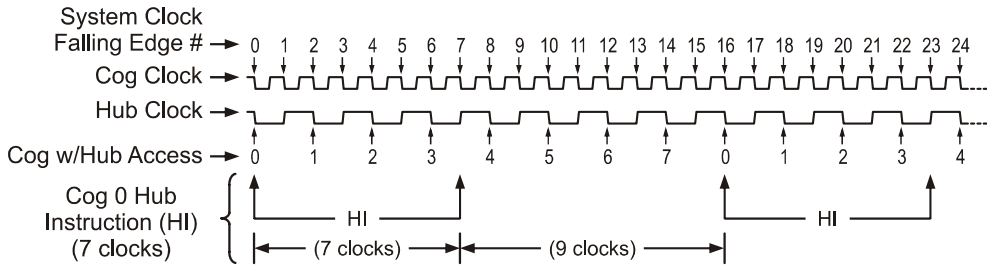
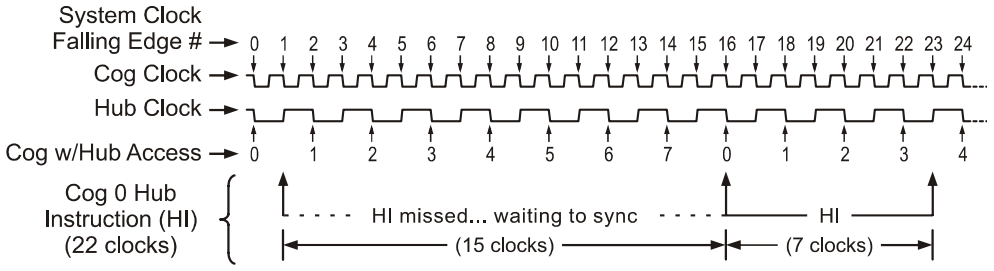


Figure 3: Cog-Hub Interaction – Worst Case Scenario



3.5. I/O Pins

The Propeller has 32 I/O pins, 28 of which are general purpose. I/O Pins 28 - 31 have a special purpose at Boot Up and are available for general purpose use afterwards; see section 1.9, page 3. After boot up, any I/O pins can be used by any cogs at any time. It is up to the application developer to ensure that no two cogs try to use the same I/O pin for different purposes during run-time.

Each cog has its own 32-bit I/O Direction Register and 32-bit I/O Output Register. The state of each cog's Direction Register is OR'd with that of the previous cogs' Direction Registers, and each cog's output states is OR'd with that of the previous cogs' output states. Note that each cog's output states are made up of the OR'd states of its internal I/O hardware and that is all AND'd with its Direction Register's states. The result is that each I/O pin's direction and output state is the "wired-OR" of the entire cog collective. No electrical contention between cogs is possible, yet they can all still access the I/O pins simultaneously. The result of this I/O pin wiring configuration can be described in the following rules:

A. A pin is an input only if no active cog sets it to an output.
B. A pin outputs low only if all active cogs that set it to output also set it to low.
C. A pin outputs high if any active cog sets it to an output and also sets it high.

Table 3 demonstrates a few possible combinations of the collective cogs' influence on a particular I/O pin, P12 in this example. For simplification, these examples assume that bit 12 of each cog's I/O hardware, other than its I/O Output Register, is cleared to zero (0).

Any cog that is shut down has its Direction Register and output states cleared to zero, effectively removing it from influencing the final state of the I/O pins that the remaining active cogs are controlling.

Each cog also has its own 32-bit Input Register. This input register is really a pseudo-register; every time it is read, the actual states of the I/O pins are read, regardless of their input or output direction.

Table 3: I/O Sharing Examples

	Bit 12 of Cogs' I/O Direction Register	Bit 12 of Cogs' I/O Output Register	State of I/O Pin P12	Rule Followed
Cog ID	**0 1 2 3 4 5 6 7**	**0 1 2 3 4 5 6 7**		
Example 1	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	Input	A
Example 2	1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	Output Low	B
Example 3	1 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0	Output High	C
Example 4	1 0 0 0 0 0 0 0	0 1 0 0 0 0 0 0	Output Low	B
Example 5	1 1 0 0 0 0 0 0	0 1 0 0 0 0 0 0	Output High	C
Example 6	1 1 1 1 1 1 1 1	0 1 0 1 0 0 0 0	Output High	C
Example 7	1 1 1 1 1 1 1 1	0 0 0 1 0 0 0 0	Output High	C
Example 8	1 1 1 0 1 1 1 1	0 0 0 1 0 0 0 0	Output Low	B

Note: For the I/O Direction Register, a 1 in a bit location sets the corresponding I/O pin to the output direction; a 0 sets it to an input direction.

3.6. System Counter

The System Counter is a global, read-only, 32-bit counter that increments once every System Clock cycle. Cogs can read the System Counter (via their `CNT` register, see Table 6 on page 10) on page to perform timing calculations and can use the `WAITCNT` command (see section 5.3 on page 13 and section 5.4 on page 16) to create effective delays within their processes. The System Counter is a common resource which every cog can read simultaneously. The System Counter is not cleared upon startup since its practical use is for differential timing. If a cog needs to keep track of time from a specific, fixed moment in time, it simply needs to read and save the initial counter value at that moment in time, and compare subsequent counter values against that initial value.

3.7. Locks

There are eight lock bits (semaphores) available to facilitate exclusive access to user-defined resources among multiple cogs. If a block of memory is to be used by two or more cogs at once and that block consists of more than one long (four bytes), the cogs will each have to perform multiple reads and writes to retrieve or update that memory block. This leads to the likely possibility of read/write contention on that memory block where one cog may be writing while another is reading, resulting in misreads and/or miswrites.

The locks are global bits accessed through the Hub via `LOCKNEW`, `LOCKRET`, `LOCKSET`, and `LOCKCLR`. Because locks are accessed only through the Hub, only one cog at a time can affect them, making this an effective control mechanism. The Hub maintains an inventory of which locks are in use and their current states; cogs can check out, return, set, and clear locks as needed during run time.

3.8. CLK Register

The CLK register is the System Clock configuration control; it determines the source and characteristics of the System Clock. It configures the RC Oscillator, Clock PLL, Crystal Oscillator, and Clock Selector circuits (See the Block Diagram, page 1). It is configured at compile time by the **_CLKMODE** declaration and is writable at run time through the **CLKSET** command. Whenever the CLK register is written, a global delay of ~75 µs occurs as the clock source transitions.

Whenever this register is changed, a copy of the value written should be placed in the Clock Mode value location (which is BYTE[4] in Main RAM) and the resulting master clock frequency should be written to the Clock Frequency value location (which is LONG[0] in Main RAM) so that objects which reference this data will have current information for their timing calculations.

Use Spin's **CLKSET** command when possible (see sections 5.3 and 5.4) since it automatically updates all the above-mentioned locations with the proper information.

Table 4: Valid Clock Modes

Valid Expression	CLK Reg. Value	Valid Expression	CLK Reg. Value
RCFAST	0_0_0_00_000	XTAL1 + PLL1X	0_1_1_01_011
		XTAL1 + PLL2X	0_1_1_01_100
RCSLOW	0_0_0_00_001	XTAL1 + PLL4X	0_1_1_01_101
		XTAL1 + PLL8X	0_1_1_01_110
XINPUT	0_0_1_00_010	XTAL1 + PLL16X	0_1_1_01_111
		XTAL2 + PLL1X	0_1_1_10_011
XTAL1	0_0_1_01_010	XTAL2 + PLL2X	0_1_1_10_100
XTAL2	0_0_1_10_010	XTAL2 + PLL4X	0_1_1_10_101
XTAL3	0_0_1_11_010	XTAL2 + PLL8X	0_1_1_10_110
		XTAL2 + PLL16X	0_1_1_10_111
XINPUT + PLL1X	0_1_1_00_011	XTAL3 + PLL1X	0_1_1_11_011
XINPUT + PLL2X	0_1_1_00_100	XTAL3 + PLL2X	0_1_1_11_100
XINPUT + PLL4X	0_1_1_00_101	XTAL3 + PLL4X	0_1_1_11_101
XINPUT + PLL8X	0_1_1_00_110	XTAL3 + PLL8X	0_1_1_11_110
XINPUT + PLL16X	0_1_1_00_111	XTAL3 + PLL16X	0_1_1_11_111

Table 5: CLK Register Fields

Bit	7	6	5	4	3	2	1	0
Name	RESET	PLLENA	OSCENA	OSCM1	OSCM2	CLKSEL2	CLKSEL1	CLKSEL0

RESET	Effect
0	Always write '0' here unless you intend to reset the chip.
1	Same as a hardware reset – reboots the chip.

PLLENA	Effect
0	Disables the PLL circuit.
1	Enables the PLL circuit. The PLL internally multiplies the XIN pin frequency by 16. OSCENA must be '1' to propagate the XIN signal to the PLL. The PLL's internal frequency must be kept within 64 MHz to 128 MHz – this translates to an XIN frequency range of 4 MHz to 8 MHz. Allow 100 µs for the PLL to stabilize before switching to one of its outputs via the CLKSEL bits. Once the OSC and PLL circuits are enabled and stabilized, you can switch freely among all clock sources by changing the CLKSEL bits.

OSCENA	Effect
0	Disables the OSC circuit
1	Enables the OSC circuit so that a clock signal can be input to XIN, or so that XIN and XOUT can function together as a feedback oscillator. The OSCM bits select the operating mode of the OSC circuit. Note that no external resistors or capacitors are required for crystals and resonators. Allow a crystal or resonator 10ms to stabilize before switching to an OSC or PLL output via the CLKSEL bits. When enabling the OSC circuit, the PLL may be enabled at the same time so that they can share the stabilization period.

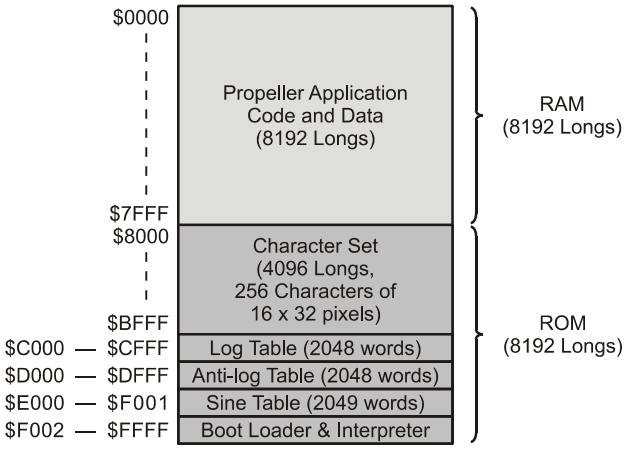
OSCM1	OSCM2	XOUT Resistance	XIN and XOUT Capacitance	Frequency Range
0	0	Infinite	6 pF	DC to 80 MHz Input
0	1	2000 Ω	36 pF	4 MHz to 16 MHz Crystal/Resonator
1	0	1000 Ω	26 pF	8 MHz to 32 MHz Crystal/Resonator
1	1	500 Ω	16 pF	20 MHz to 60 MHz Crystal/Resonator

CLKSEL2	CLKSEL1	CLKSEL0	Master Clock	Source	Notes
0	0	0	~12 MHz	Internal	No external parts (8 to 20 MHz)
0	0	1	~20 kHz	Internal	No external parts, very low power (13-33 kHz)
0	1	0	XIN	OSC	OSCENA must be '1'
0	1	1	XIN × 1	OSC+PLL	OSCENA and PLLENA must be '1'
1	0	0	XIN × 2	OSC+PLL	OSCENA and PLLENA must be '1'
1	0	1	XIN × 4	OSC+PLL	OSCENA and PLLENA must be '1'
1	1	0	XIN × 8	OSC+PLL	OSCENA and PLLENA must be '1'
1	1	1	XIN × 16	OSC+PLL	OSCENA and PLLENA must be '1'

4.0 MEMORY ORGANIZATION

4.1. Main Memory

The Main Memory is a block of 64 K bytes (16 K longs) that is accessible by all cogs as a mutually-exclusive resource through the Hub. It consists of 32 KB of RAM and 32 KB of ROM. Main memory is byte, word and long addressable.



4.1.1. Main RAM

The 32 KB of Main RAM is general purpose and is the destination of a Propeller Application either downloaded from a host or from the external 32 KB EEPROM.

4.1.2. Main ROM

The 32 KB of Main ROM contains all the code and data resources vital to the Propeller chip's function: character definitions, log, anti-log and sine tables, and the Boot Loader and Spin Interpreter.

4.1.3. Character Definitions

The first half of ROM is dedicated to a set of 256 character definitions. Each character definition is 16 pixels wide by 32 pixels tall. These character definitions can be used for video generation, graphical LCD's, printing, etc.

The character set is based on a North American / Western European layout, with many specialized characters added and inserted. There are connecting waveform and schematic building-block characters, Greek characters commonly used in electronics, and several arrows and bullets. (A corresponding Parallax True-Type Font is installed with and used by the Propeller Tool software, and is available to other Windows applications.)

The character definitions are numbered 0 to 255 from left-to-right, then top-to-bottom, per Figure 4 below. They are arranged as follows: Each pair of adjacent even-odd characters is merged together to form 32 longs. The first character pair is located in $8000-$807F. The second pair occupies $8080-$80FF, and so on, until the last pair fills $BF80-$BFFF.

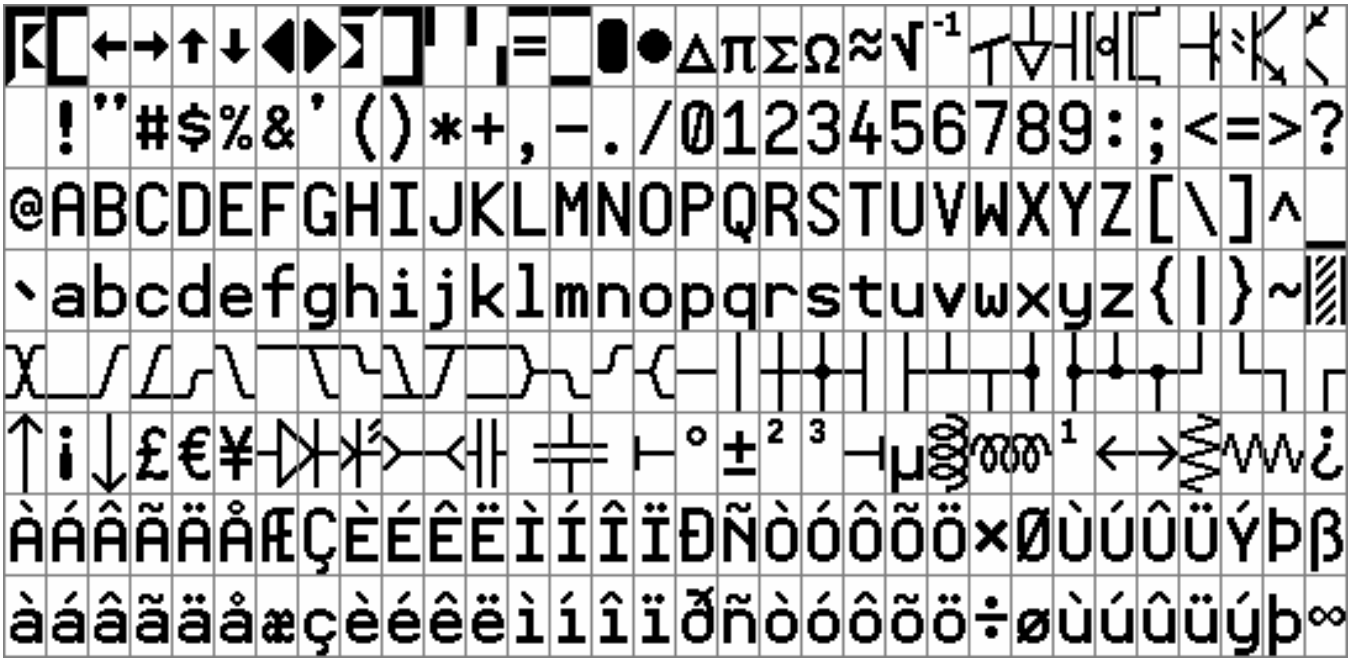


Figure 4: Propeller Font Character Set

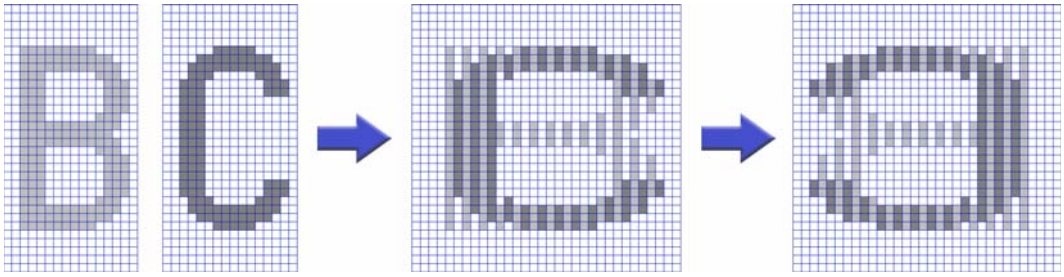


Figure 5

Propeller Character Interleaving

As shown in Figure 5, The character pairs are merged such that each character's 16 horizontal pixels (per row) are spaced apart and interleaved with their neighbors' such that the even character takes bits 0, 2, 4, ...30, and the odd character takes bits 1, 3, 5, ...31. The leftmost pixels are in the lowest bits, while the rightmost pixels are in the highest bits. This forms a long for each row of pixels in the character pair. 32 such longs, building from top row down to bottom, make up the complete merged-pair definition. The definitions are encoded in this manner so that a COG video peripheral can handle the merged longs directly, using color selection to display either the even or the odd character.

Some character codes have inescapable meanings, such as 9 for Tab, 10 for Line Feed, and 13 for Carriage Return (0 can also be touchy). These character codes invoke actions and do not equate to static character definitions. For this reason, their character definitions have been used for special four-color characters. These four-color characters are used for drawing 3-D box edges at run-time and are implemented as 16 x 16 pixel cells, as opposed to the normal 16 x 32 pixel cells. They occupy even-odd character pairs 0-1, 8-9, 10-11, and 12-13.

4.1.4. Math Function Tables

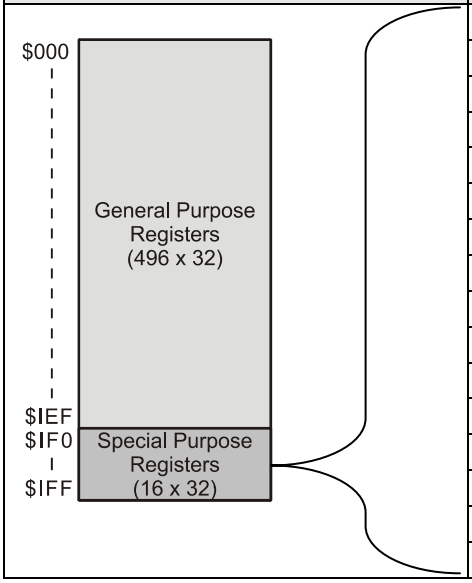
Base-2 Log and Anti-Log tables, each with 2048 unsigned words, facilitate converting values to and from exponent form to facilitate some operations; see the Propeller Manual for access instructions. Also, a sine table provides 2049 unsigned 16-bit sine samples spanning 0° to 90° inclusively (0.0439° resolution).

4.2. Cog RAM

As stated in Section 3.3, the Cog RAM is used for executable code, data, variables, and the last 16 locations serve as interfaces to the System Counter, I/O pins, and local cog peripherals (see Table 6).

When a cog is booted up, locations 0 ($000) through 495 ($1EF) are loaded sequentially from Main RAM / ROM and its special purpose locations, 496 ($1F0) through 511 ($1FF), are cleared to zero. Each Special Purpose register may be accessed via its physical address, its predefined name, or indirectly in Spin via a register array variable `SPR` with an index of 0 to 15, the last four bits of the register's address.

Table 6: Cog RAM Special Purpose Registers

Cog RAM Map	Address	Name	Type	Description
	$1F0	PAR	Read-Only[1]	Boot Parameter
	$1F1	CNT	Read-Only[1]	System Counter
	$1F2	INA	Read-Only[1]	Input States for P31 - P0
	$1F3	INB	Read-Only[1]	Input States for P63- P32[2]
	$1F4	OUTA	Read/Write	Output States for P31 - P0
	$1F5	OUTB	Read/Write	Output States for P63 – P32[2]
	$1F6	DIRA	Read/Write	Direction States for P31 - P0
	$1F7	DIRB	Read/Write	Direction States for P63 - P32[2]
	$1F8	CTRA	Read/Write	Counter A Control
	$1F9	CTRB	Read/Write	Counter B Control
	$1FA	FRQA	Read/Write	Counter A Frequency
	$1FB	FRQB	Read/Write	Counter B Frequency
	$1FC	PHSA	Read/Write	Counter A Phase:
	$1FD	PHSB	Read/Write	Counter B Phase
	$1FE	VCFG	Read/Write	Video Configuration
	$1FF	VSCL	Read/Write	Video Scale

Note 1: Only readable as a Source Register (i.e. `MOV DEST, SOURCE`); read-modify-write not possible as a Destination Register.
Note 2: Reserved for future use.

5.0 PROGRAMMING LANGUAGES

The Propeller chip is programmed using two languages designed specifically for it: 1) Spin, a high-level object-based language, and 2) Propeller Assembly, a low-level, highly-optimized assembly language. There are many hardware-based commands in Propeller Assembly that have direct equivalents in the Spin language.

The Spin language is compiled by the Propeller Tool software into tokens that are interpreted at run time by the Propeller chip's built-in Spin Interpreter. The Propeller Assembly language is assembled into pure machine code by the Propeller Tool and is executed in its pure form at run time.

Propeller Objects can be written entirely in Spin or can use various combinations of Spin and Propeller Assembly. It is often advantageous to write objects almost entirely in Propeller Assembly, but at least two lines of Spin code are required to launch the final application.

5.1. Reserved Word List

All words listed are always reserved, whether programming in Spin or in Propeller Assembly.

Table 7: Reserved Word List

_CLKFREQ[s]	COGINIT[d]	IF_C_AND_NZ[a]	LOCKNEW[d]	NOP[a]	REPEAT[s]	TRUE[d]
_CLKMODE[s]	COGNEW[s]	IF_C_AND_Z[a]	LOCKRET[d]	NOT[s]	RES[a]	TRUNC[s]
_FREE[s]	COGSTOP[d]	IF_C_EQ_Z[a]	LOCKSET[d]	NR[a]	RESULT[s]	UNTIL[s]
_STACK[s]	CON[s]	IF_C_NE_Z[a]	LONG[s]	OBJ[s]	RET[a]	VAR[s]
_XINFREQ[s]	CONSTANT[s]	IF_C_OR_NZ[a]	LONGFILL[s]	ONES[a#]	RETURN[s]	VCFG[d]
ABORT[s]	CTRA[d]	IF_C_OR_Z[a]	LONGMOVE[s]	OR[d]	REV[a]	VSCL[d]
ABS[a]	CTRB[d]	IF_E[a]	LOOKDOWN[s]	ORG[a]	ROL[a]	WAITCNT[d]
ABSNEG[a]	DAT[s]	IF_NC[a]	LOOKDOWNZ[s]	OTHER[s]	ROR[a]	WAITPEQ[d]
ADD[a]	DIRA[d]	IF_NC_AND_NZ[a]	LOOKUP[s]	OUTA[d]	ROUND[s]	WAITPNE[d]
ADDABS[a]	DIRB[d#]	IF_NC_AND_Z[a]	LOOKUPZ[s]	OUTB[d#]	SAR[a]	WAITVID[d]
ADDS[a]	DJNZ[a]	IF_NC_OR_NZ[a]	MAX[a]	PAR[d]	SHL[a]	WC[a]
ADDSX[a]	ELSE[s]	IF_NC_OR_Z[a]	MAXS[a]	PHSA[d]	SHR[a]	WHILE[s]
ADDX[a]	ELSEIF[s]	IF_NE[a]	MIN[a]	PHSB[d]	SPR[s]	WORD[s]
AND[d]	ELSEIFNOT[s]	IF_NEVER[a]	MINS[a]	PI[d]	STEP[s]	WORDFILL[s]
ANDN[a]	ENC[a#]	IF_NZ[a]	MOV[a]	PLL1X[s]	STRCOMP[s]	WORDMOVE[s]
BYTE[s]	FALSE[d]	IF_NZ_AND_C[a]	MOVD[a]	PLL2X[s]	STRING[s]	WR[a]
BYTEFILL[s]	FILE[s]	IF_NZ_AND_NC[a]	MOVI[a]	PLL4X[s]	STRSIZE[s]	WRBYTE[a]
BYTEMOVE[s]	FIT[a]	IF_NZ_OR_C[a]	MOVS[a]	PLL8X[s]	SUB[a]	WRLONG[a]
CALL[a]	FLOAT[s]	IF_NZ_OR_NC[a]	MUL[a#]	PLL16X[s]	SUBABS[a]	WRWORD[a]
CASE[s]	FROM[s]	IF_Z[a]	MULS[a#]	POSX[d]	SUBS[a]	WZ[a]
CHIPVER[s]	FRQA[d]	IF_Z_AND_C[a]	MUXC[a]	PRI[s]	SUBSX[a]	XINPUT[s]
CLKFREQ[s]	FRQB[d]	IF_Z_AND_NC[a]	MUXNC[a]	PUB[s]	SUBX[a]	XOR[a]
CLKMODE[s]	HUBOP[a]	IF_Z_EQ_C[a]	MUXNZ[a]	QUIT[s]	SUMC[a]	XTAL1[s]
CLKSET[d]	IF[s]	IF_Z_NE_C[a]	MUXZ[a]	RCFAST[s]	SUMNC[a]	XTAL2[s]
CMP[a]	IFNOT[s]	IF_Z_OR_C[a]	NEG[a]	RCL[a]	SUMNZ[a]	XTAL3[s]
CMPS[a]	IF_A[a]	IF_Z_OR_NC[a]	NEGC[a]	RCR[a]	SUMZ[a]	
CMPSUB[a]	IF_AE[a]	INA[d]	NEGNC[a]	RCSLOW[s]	TEST[a]	
CMPSX[a]	IF_ALWAYS[a]	INB[d#]	NEGNZ[a]	RDBYTE[a]	TESTN[a]	
CMPX[a]	IF_B[a]	JMP[a]	NEGX[d]	RDLONG[a]	TJNZ[a]	
CNT[d]	IF_BE[a]	JMPRET[a]	NEGZ[a]	RDWORD[a]	TJZ[a]	
COGID[d]	IF_C[a]	LOCKCLR[d]	NEXT[s]	REBOOT[s]	TO[s]	

a = Assembly element; s = Spin element; d = dual (available in both languages); # = reserved for future use

5.1.1. Words Reserved for Future Use

- **DIRB**, **INB**, and **OUTB**: Reserved for future use with a possible 64 I/O pin model. When used with the P8X32A, these labels can be used to access Cog RAM at those locations for general-purpose use.
- **ENC**, **MUL**, **MULS**, **ONES**: Use with the current P8X32A architecture yields indeterminate results.

5.2. Math and Logic Operators

Table 8: Math and Logic Operators

Level[1]	Operator		Constant Expressions[3]		Is Unary	Description
	Normal	Assign[2]	Integer	Float		
Highest (0)	--	always			✓	Pre-decrement (--X) or post-decrement (X--).
	++	always			✓	Pre-increment (++X) or post-increment (X++).
	~	always			✓	Sign-extend bit 7 (~X) or post-clear to 0 (X~).
	~~	always			✓	Sign-extend bit 15 (~~X) or post-set to -1 (X~~).
	?	always			✓	Random number forward (?X) or reverse (X?).
	@	never	✓		✓	Symbol address.
	@@	never			✓	Object address plus symbol.
1	+	never	✓	✓	✓	Positive (+X); unary form of Add.
	-	if solo	✓	✓	✓	Negate (-X); unary form of Subtract.
	^^	if solo	✓	✓	✓	Square root.
	||	if solo	✓	✓	✓	Absolute value.
	|<	if solo	✓		✓	Bitwise: Decode 0 – 31 to long w/single-high-bit.
	>|	if solo	✓		✓	Bitwise: Encode long to 0 – 32; high-bit priority.
	!	if solo	✓		✓	Bitwise: NOT.
2	<-	<-=	✓			Bitwise: Rotate left.
	->	->=	✓			Bitwise: Rotate right.
	<<	<<=	✓			Bitwise: Shift left.
	>>	>>=	✓			Bitwise: Shift right.
	~>	~>=	✓			Shift arithmetic right.
	><	><=	✓			Bitwise: Reverse.
3	&	&=	✓			Bitwise: AND.
4	|	|=	✓			Bitwise: OR.
	^	^=	✓			Bitwise: XOR.
5	*	*=	✓	✓		Multiply and return lower 32 bits (signed).
	**	**=	✓			Multiply and return upper 32 bits (signed).
	/	/=	✓	✓		Divide (signed).
	//	//=	✓			Modulus (signed).
6	+	+=	✓	✓		Add.
	-	-=	✓	✓		Subtract.
7	#>	#>=	✓	✓		Limit minimum (signed).
	<#	<#=	✓	✓		Limit maximum (signed).
8	<	<=	✓	✓		Boolean: Is less than (signed).
	>	>=	✓	✓		Boolean: Is greater than (signed).
	<>	<>=	✓	✓		Boolean: Is not equal.
	==	===	✓	✓		Boolean: Is equal.
	=<	=<=	✓	✓		Boolean: Is equal or less (signed).
	=>	=>=	✓	✓		Boolean: Is equal or greater (signed).
9	NOT	if solo	✓	✓	✓	Boolean: NOT (promotes non-0 to -1).
10	AND	AND=	✓	✓		Boolean: AND (promotes non-0 to -1).
11	OR	OR=	✓	✓		Boolean: OR (promotes non-0 to -1).
Lowest (12)	=	always	n/a[3]	n/a[3]		Constant assignment (CON blocks).
	:=	always	n/a[3]	n/a[3]		Variable assignment (PUB/PRI blocks).

[1] Precedence level: higher-level operators evaluate before lower-level operators. Operators in same level are commutable; evaluation order does not matter.
[2] Assignment forms of binary (non-unary) operators are in the lowest precedence (level 12).
[3] Assignment forms of operators are not allowed in constant expressions.

5.3. Spin Language Summary Table

Spin Command	Returns Value	Description
ABORT ⟨*Value*⟩	✓	Exit from PUB/PRI method using abort status with optional return value.
BYTE *Symbol* ⟨[*Count*]⟩		Declare byte-sized symbol in VAR block.
BYTE *Data* ⟨[*Count*]⟩		Declare byte-aligned and/or byte-sized data in DAT block.
BYTE [*BaseAddress*] ⟨[*Offset*]⟩	✓	Read/write byte of main memory.
Symbol.BYTE ⟨[*Offset*]⟩	✓	Read/write byte-sized component of word/long-sized variable.
BYTEFILL (*StartAddress*, *Value*, *Count*)		Fill bytes of main memory with a value.
BYTEMOVE (*DestAddress*, *SrcAddress*, *Count*)		Copy bytes from one region to another in main memory.
CASE *CaseExpression* →| *MatchExpression* : →| *Statement(s)* ⟨→| *MatchExpression* : →| *Statement(s)*⟩ ⟨→| OTHER : →| *Statement(s)*⟩		Compare expression against matching expression(s), execute code block if match found. *MatchExpression* can contain a single expression or multiple comma-delimited expressions. Expressions can be a single value (ex: 10) or a range of values (ex: 10..15).
CHIPVER	✓	Version number of the Propeller chip (Byte at $FFFF)
CLKFREQ	✓	Current System Clock frequency, in Hz (Long at $0000)
CLKMODE	✓	Current clock mode setting (Byte at $0004)
CLKSET (*Mode*, *Frequency*)		Set both clock mode and System Clock frequency at run time.
CNT	✓	Current 32-bit System Counter value.
COGID	✓	Current cog's ID number; 0-7.
COGINIT (*CogID*, *SpinMethod* ⟨(*ParameterList*)⟩, *StackPointer*)		Start or restart cog by ID to run Spin code.
COGINIT (*CogID*, *AsmAddress*, *Parameter*)		Start or restart cog by ID to run Propeller Assembly code.
COGNEW (*SpinMethod* ⟨(*ParameterList*)⟩, *StackPointer*)	✓	Start new cog for Spin code and get cog ID; 0-7 = succeeded, -1 = failed.
COGNEW (*AsmAddress*, *Parameter*)	✓	Start new cog for Propeller Assembly code and get cog ID; 0-7 = succeeded, -1 = failed.
COGSTOP (*CogID*)		Stop cog by its ID.
CON *Symbol* = *Expr* ⟨((, ¦ ↳)) *Symbol* = *Expr*⟩…		Declare symbolic, global constants.
CON ⟨#*Expr* ((, ¦ ↳))⟩ *Symbol* ⟨⟨((, ¦ ↳)) #*Expr*⟩ ((, ¦ ↳)) *Symbol*⟩…		Declare global enumerations (incrementing symbolic constants).
CONSTANT (*ConstantExpression*)	✓	Declare in-line constant expression to be completely resolved at compile time.
CTRA	✓	Counter A Control register.
CTRB	✓	Counter B Control register.
DAT ⟨*Symbol*⟩ *Alignment* ⟨*Size*⟩ ⟨*Data*⟩ ⟨,⟨*Size*⟩ *Data*⟩…		Declare table of data, aligned and sized as specified.
DAT ⟨*Symbol*⟩ ⟨*Condition*⟩ *Instruction* ⟨*Effect(s)*⟩		Denote Propeller Assembly instruction.
DIRA ⟨[*Pin(s)*]⟩	✓	Direction register for 32-bit port A.
FILE "*FileName*"		Import external file as data in DAT block.
FLOAT (*IntegerConstant*)	✓	Convert integer constant expression to compile-time floating-point value in any block.
FRQA	✓	Counter A Frequency register.
FRQB	✓	Counter B Frequency register.
((IF ¦ IFNOT)) *Condition(s)* →| *IfStatement(s)* ⟨ELSEIF *Condition(s)* →| *ElseIfStatement(s)*⟩… ⟨ELSEIFNOT *Condition(s)* →| *ElseIfStatement(s)*⟩… ⟨ELSE →| *ElseStatement(s)*⟩		Test condition(s) and execute block of code if valid. IF and ELSEIF each test for TRUE. IFNOT and ELSEIFNOT each test for FALSE.

Spin Command	Returns Value	Description
INA 〈[*Pin(s)*]〉	✓	Input register for 32-bit ports A.
LOCKCLR (*ID*)	✓	Clear semaphore to false and get its previous state; TRUE or FALSE.
LOCKNEW	✓	Check out new semaphore and get its ID; 0-7, or -1 if none were available.
LOCKRET (*ID*)		Return semaphore back to semaphore pool, releasing it for future LOCKNEW requests.
LOCKSET (*ID*)	✓	Set semaphore to true and get its previous state; TRUE or FALSE.
LONG *Symbol* 〈[*Count*]〉		Declare long-sized symbol in VAR block.
LONG *Data* 〈[*Count*]〉		Declare long-aligned and/or long-sized data in DAT block.
LONG [*BaseAddress*] 〈[*Offset*]〉	✓	Read/write long of main memory.
LONGFILL (*StartAddress*, *Value*, *Count*)		Fill longs of main memory with a value.
LONGMOVE (*DestAddress*, *SrcAddress*, *Count*)		Copy longs from one region to another in main memory.
LOOKDOWN (*Value*:*ExpressionList*)	✓	Get the one-based index of a value in a list.
LOOKDOWNZ (*Value*:*ExpressionList*)	✓	Get the zero-based index of a value in a list.
LOOKUP (*Index*:*ExpressionList*)	✓	Get value from a one-based index position of a list.
LOOKUPZ (*Index*:*ExpressionList*)	✓	Get value from a zero-based index position of a list.
NEXT		Skip remaining statements of REPEAT loop and continue with the next loop iteration.
OBJ *Symbol* 〈[*Count*]〉:"*Object*" 〈 ↳ *Symbol* 〈[*Count*]〉: "*Object*"〉...		Declare symbol object references.
OUTA 〈[*Pin(s)*]〉	✓	Output register for 32-bit port A.
PAR	✓	Cog Boot Parameter register.
PHSA	✓	Counter A Phase Lock Loop (PLL) register.
PHSB	✓	Counter B Phase Lock Loop (PLL) register.
PRI *Name* 〈(*Par* 〈,*Par*〉...)〉 〈:*RVal*〉 〈| *LVar* 〈[*Cnt*]〉〉 〈,*LVar* 〈[*Cnt*]〉〉... *SourceCodeStatements*		Declare private method with optional parameters, return value and local variables.
PUB *Name* 〈(*Par* 〈,*Par*〉...)〉 〈:*RVal*〉 〈| *LVar* 〈[*Cnt*]〉〉 〈,*LVar* 〈[*Cnt*]〉〉... *SourceCodeStatements*		Declare public method with optional parameters, return value and local variables.
QUIT		Exit from REPEAT loop immediately.
REBOOT		Reset the Propeller chip.
REPEAT 〈*Count*〉 →! *Statement(s)*		Execute code block repetitively, either infinitely, or for a finite number of iterations.
REPEAT *Variable* FROM *Start* TO *Finish* 〈STEP *Delta*〉 →! *Statement(s)*		Execute code block repetitively, for finite, counted iterations.
REPEAT ((UNTIL ¦ WHILE)) *Condition(s)* →! *Statement(s)*		Execute code block repetitively, zero-to-many conditional iterations.
REPEAT →! *Statement(s)* ((UNTIL ¦ WHILE)) *Condition(s)*		Execute code block repetitively, one-to-many conditional iterations.
RESULT	✓	Return value variable for PUB/PRI methods.
RETURN 〈*Value*〉	✓	Exit from PUB/PRI method with optional return *Value*.
ROUND (*FloatConstant*)	✓	Round floating-point constant to the nearest integer at compile-time, in any block.
SPR [*Index*]	✓	Special Purpose Register array.
STRCOMP (*StringAddress1*, *StringAddress2*)	✓	Compare two strings for equality.
STRING (*StringExpression*)	✓	Declare in-line string constant and get its address.
STRSIZE (*StringAddress*)	✓	Get size, in bytes, of zero-terminate string.
TRUNC (*FloatConstant*)	✓	Remove fractional portion from floating-point constant at compile-time, in any block.
VAR *Size Symbol* 〈[*Count*]〉 〈((, ¦ ↳ *Size*)) *Symbol* 〈[*Count*]〉〉...		Declare symbolic global variables.
VCFG	✓	Video Configuration register.

Spin Command	Returns Value	Description
VSCL	✓	Video Scale register.
WAITCNT (*Value*)		Pause cog's execution temporarily.
WAITPEQ (*State*, *Mask*, *Port*)		Pause cog's execution until I/O pin(s) match designated state(s).
WAITPNE (*State*, *Mask*, *Port*)		Pause cog's execution until I/O pin(s) do not match designated state(s).
WAITVID (*Colors*, *Pixels*)		Pause cog's execution until its Video Generator is available for pixel data.
WORD *Symbol* 〈[*Count*]〉		Declare word-sized symbol in VAR block.
WORD *Data* 〈[*Count*]〉		Declare word-aligned and/or word-sized data in DAT block.
WORD [*BaseAddress*] 〈[*Offset*]〉	✓	Read/write word of main memory.
Symbol.WORD 〈[*Offset*]〉	✓	Read/write word-sized component of long-sized variable.
WORDFILL (*StartAddress*, *Value*, *Count*)		Fill words of main memory with a value.
WORDMOVE (*DestAddress*, *SrcAddress*, *Count*)		Copy words from one region to another in main memory.

5.3.1. Constants

Constants (pre-defined)			
Constant[1]	Description		
_CLKFREQ	Settable in Top Object File to specify System Clock frequency.		
_CLKMODE	Settable in Top Object File to specify application's clock mode.		
_XINFREQ	Settable in Top Object File to specify external crystal frequency.		
_FREE	Settable in Top Object File to specify application's free space.		
_STACK	Settable in Top Object File to specify application's stack space.		
TRUE	Logical true:	-1	($FFFFFFFF)
FALSE	Logical false:	0	($00000000)
POSX	Max. positive integer:	2,147,483,647	($7FFFFFFF)
NEGX	Max. negative integer:	-2,147,483,648	($80000000)
PI	Floating-point PI:	≈ 3.141593	($40490FDB)
RCFAST	Internal fast oscillator:	$00000001	(%00000000001)
RCSLOW	Internal slow oscillator:	$00000002	(%00000000010)
XINPUT	External clock/oscillator:	$00000004	(%00000000100)
XTAL1	External low-speed crystal:	$00000008	(%00000001000)
XTAL2	External medium-speed crystal:	$00000010	(%00000010000)
XTAL3	External high-speed crystal:	$00000020	(%00000100000)
PLL1X	External frequency times 1:	$00000040	(%00001000000)
PLL2X	External frequency times 2:	$00000080	(%00010000000)
PLL4X	External frequency times 4:	$00000100	(%00100000000)
PLL8X	External frequency times 8:	$00000200	(%01000000000)
PLL16X	External frequency times 16:	$00000400	(%10000000000)

[1] "Settable" constants are defined in Top Object File's CON block. See Valid Clock Modes for _CLKMODE. Other settable constants use whole numbers.

5.4. Propeller Assembly Instruction Table

The Propeller Assembly Instruction Table lists the instruction's 32-bit opcode, outputs and number of clock cycles. The opcode consists of the instruction bits (**iiiiii**), the "effect" status for the Z flag, C flag, result and indirect/immediate status (zcri), the conditional execution bits (**cccc**), and the destination and source bits (**ddddddddd** and **sssssssss**). The meaning of the Z and C flags, if any, is shown in the Z Result and C Result fields; indicating the meaning of a 1 in those flags. The Result field (R) shows the instruction's default behavior for writing or not writing the instruction's result value. The Clocks field shows the number of clocks the instruction requires for execution.

- `0 1` Zeros (0) and ones (1) mean binary 0 and 1.
- `i` Lower case "i" denotes a bit that is affected by immediate status.
- `d s` Lower case "d" and "s" indicate destination and source bits.
- `?` Question marks denote bits that are dynamically set by the compiler.
- `---` Hyphens indicate items that are not applicable or not important.
- `..` Double-periods represent a range of contiguous values.

iiiiii zcri cccc ddddddddd sssssssss	Instruction	Description	Z out	C out	R	Clocks
000000 000i 1111 ddddddddd sssssssss	**WRBYTE** D,S	Write D[7..0] to main memory byte S[15..0]	-	-	0	7..22 *
000000 001i 1111 ddddddddd sssssssss	**RDBYTE** D,S	Read main memory byte S[15..0] into D (0-extended)	Result = 0	-	1	7..22 *
000001 000i 1111 ddddddddd sssssssss	**WRWORD** D,S	Write D[15..0] to main memory word S[15..1]	-	-	0	7..22 *
000001 001i 1111 ddddddddd sssssssss	**RDWORD** D,S	Read main memory word S[15..1] into D (0-extended)	Result = 0	-	1	7..22 *
000010 000i 1111 ddddddddd sssssssss	**WRLONG** D,S	Write D to main memory long S[15..2]	-	-	0	7..22 *
000010 001i 1111 ddddddddd sssssssss	**RDLONG** D,S	Read main memory long S[15..2] into D	Result = 0	-	1	7..22 *
000011 000i 1111 ddddddddd sssssssss	**HUBOP** D,S	Perform hub operation according to S	Result = 0	-	0	7..22 *
000011 0001 1111 ddddddddd ------000	**CLKSET** D	Set the global CLK register to D[7..0]	-	-	0	7..22 *
000011 0011 1111 ddddddddd ------001	**COGID** D	Get this cog number (0..7) into D	Result = 0	-	1	7..22 *
000011 0001 1111 ddddddddd ------010	**COGINIT** D	Initialize a cog according to D	Result = 0	No COG free	0	7..22 *
000011 0001 1111 ddddddddd ------011	**COGSTOP** D	Stop cog number D[2..0]	-	-	0	7..22 *
000011 0011 1111 ddddddddd ------100	**LOCKNEW** D	Checkout a new LOCK number (0..7) into D	Result = 0	No LOCK free	1	7..22 *
000011 0001 1111 ddddddddd ------101	**LOCKRET** D	Return lock number D[2..0]	-	-	0	7..22 *
000011 0001 1111 ddddddddd ------110	**LOCKSET** D	Set lock number D[2..0]	-	Prior LOCK state	0	7..22 *
000011 0001 1111 ddddddddd ------111	**LOCKCLR** D	Clear lock number D[2..0]	-	Prior LOCK state	0	7..22 *
000100 001i 1111 ddddddddd sssssssss	**MUL** D,S	Multiply unsigned D[15..0] by S[15..0]	Result = 0	-	1	future
000101 001i 1111 ddddddddd sssssssss	**MULS** D,S	Multiply signed D[15..0] by S[15..0]	Result = 0	-	1	future
000110 001i 1111 ddddddddd sssssssss	**ENC** D,S	Encode magnitude of S into D, result = 0..31	Result = 0	-	1	future
000111 001i 1111 ddddddddd sssssssss	**ONES** D,S	Get number of 1's in S into D, result = 0..31	Result = 0	-	1	future
001000 001i 1111 ddddddddd sssssssss	**ROR** D,S	Rotate D right by S[4..0] bits	Result = 0	D[0]	1	4
001001 001i 1111 ddddddddd sssssssss	**ROL** D,S	Rotate D left by S[4..0] bits	Result = 0	D[31]	1	4
001010 001i 1111 ddddddddd sssssssss	**SHR** D,S	Shift D right by S[4..0] bits	Result = 0	D[0]	1	4
001011 001i 1111 ddddddddd sssssssss	**SHL** D,S	Shift D left by S[4..0] bits	Result = 0	D[31]	1	4
001100 001i 1111 ddddddddd sssssssss	**RCR** D,S	Rotate carry right into D by S[4..0] bits	Result = 0	D[0]	1	4
001101 001i 1111 ddddddddd sssssssss	**RCL** D,S	Rotate carry left into D by S[4..0] bits	Result = 0	D[31]	1	4
001110 001i 1111 ddddddddd sssssssss	**SAR** D,S	Shift D arithmetically right by S[4..0] bits	Result = 0	D[0]	1	4
001111 001i 1111 ddddddddd sssssssss	**REV** D,S	Reverse 32–S[4..0] bottom bits in D and 0-extend	Result = 0	D[0]	1	4
010000 001i 1111 ddddddddd sssssssss	**MINS** D,S	Set D to S if signed (D < S)	D = S	Signed (D < S)	1	4
010001 001i 1111 ddddddddd sssssssss	**MAXS** D,S	Set D to S if signed (D => S)	D = S	Signed (D < S)	1	4
010010 001i 1111 ddddddddd sssssssss	**MIN** D,S	Set D to S if unsigned (D < S)	D = S	Unsigned (D < S)	1	4
010011 001i 1111 ddddddddd sssssssss	**MAX** D,S	Set D to S if unsigned (D => S)	D = S	Unsigned (D < S)	1	4
010100 001i 1111 ddddddddd sssssssss	**MOVS** D,S	Insert S[8..0] into D[8..0]	Result = 0	-	1	4
010101 001i 1111 ddddddddd sssssssss	**MOVD** D,S	Insert S[8..0] into D[17..9]	Result = 0	-	1	4
010110 001i 1111 ddddddddd sssssssss	**MOVI** D,S	Insert S[8..0] into D[31..23]	Result = 0	-	1	4
010111 001i 1111 ddddddddd sssssssss	**JMPRET** D,S	Insert PC+1 into D[8..0] and set PC to S[8..0]	Result = 0	-	1	4

iiiiii zcri cccc dddddddd ssssssss	Instruction		Description	Z out	C out	R	Clocks
010111 000i 1111 --------- ssssssss	JMP	S	Set PC to S[8..0]	Result = 0	-	0	4
010111 0011 1111 ????????? ssssssss	CALL	#S	Like JMPRET, but assembler handles details	Result = 0	-	1	4
010111 0001 1111 --------- ---------	RET		Like JMP, but assembler handles details	Result = 0	-	0	4
011000 000i 1111 ddddddddd sssssssss	TEST	D,S	AND S with D to affect flags only	Result = 0	Parity of Result	0	4
011000 001i 1111 ddddddddd sssssssss	AND	D,S	AND S into D	Result = 0	Parity of Result	1	4
011001 001i 1111 ddddddddd sssssssss	ANDN	D,S	AND !S into D	Result = 0	Parity of Result	1	4
011010 001i 1111 ddddddddd sssssssss	OR	D,S	OR S into D	Result = 0	Parity of Result	1	4
011011 001i 1111 ddddddddd sssssssss	XOR	D,S	XOR S into D	Result = 0	Parity of Result	1	4
011100 001i 1111 ddddddddd sssssssss	MUXC	D,S	Copy C to bits in D using S as mask	Result = 0	Parity of Result	1	4
011101 001i 1111 ddddddddd sssssssss	MUXNC	D,S	Copy !C to bits in D using S as mask	Result = 0	Parity of Result	1	4
011110 001i 1111 ddddddddd sssssssss	MUXZ	D,S	Copy Z to bits in D using S as mask	Result = 0	Parity of Result	1	4
011111 001i 1111 ddddddddd sssssssss	MUXNZ	D,S	Copy !Z to bits in D using S as mask	Result = 0	Parity of Result	1	4
100000 001i 1111 ddddddddd sssssssss	ADD	D,S	Add S into D	Result = 0	Unsigned Carry	1	4
100001 001i 1111 ddddddddd sssssssss	SUB	D,S	Subtract S from D	Result = 0	Unsigned Borrow	1	4
100001 000i 1111 ddddddddd sssssssss	CMP	D,S	Compare D to S	Result = 0	Unsigned Borrow	0	4
100010 001i 1111 ddddddddd sssssssss	ADDABS	D,S	Add absolute S into D	Result = 0	Unsigned Carry	1	4
100011 001i 1111 ddddddddd sssssssss	SUBABS	D,S	Subtract absolute S from D	Result = 0	Unsigned Borrow	1	4
100100 001i 1111 ddddddddd sssssssss	SUMC	D,S	Sum either –S if C or S if !C into D	Result = 0	Signed Overflow	1	4
100101 001i 1111 ddddddddd sssssssss	SUMNC	D,S	Sum either S if C or –S if !C into D	Result = 0	Signed Overflow	1	4
100110 001i 1111 ddddddddd sssssssss	SUMZ	D,S	Sum either –S if Z or S if !Z into D	Result = 0	Signed Overflow	1	4
100111 001i 1111 ddddddddd sssssssss	SUMNZ	D,S	Sum either S if Z or –S if !Z into D	Result = 0	Signed Overflow	1	4
101000 001i 1111 ddddddddd sssssssss	MOV	D,S	Set D to S	Result = 0	S[31]	1	4
101001 001i 1111 ddddddddd sssssssss	NEG	D,S	Set D to –S	Result = 0	S[31]	1	4
101010 001i 1111 ddddddddd sssssssss	ABS	D,S	Set D to absolute S	Result = 0	S[31]	1	4
101011 001i 1111 ddddddddd sssssssss	ABSNEG	D,S	Set D to –absolute S	Result = 0	S[31]	1	4
101100 001i 1111 ddddddddd sssssssss	NEGC	D,S	Set D to either –S if C or S if !C	Result = 0	S[31]	1	4
101101 001i 1111 ddddddddd sssssssss	NEGNC	D,S	Set D to either S if C or –S if !C	Result = 0	S[31]	1	4
101110 001i 1111 ddddddddd sssssssss	NEGZ	D,S	Set D to either –S if Z or S if !Z	Result = 0	S[31]	1	4
101111 001i 1111 ddddddddd sssssssss	NEGNZ	D,S	Set D to either S if Z or –S if !Z	Result = 0	S[31]	1	4
110000 000i 1111 ddddddddd sssssssss	CMPS	D,S	Compare-signed D to S	Result = 0	Signed Borrow	0	4
110001 000i 1111 ddddddddd sssssssss	CMPSX	D,S	Compare-signed-extended D to S+C	Z & (Result = 0)	Signed Borrow	0	4
110010 001i 1111 ddddddddd sssssssss	ADDX	D,S	Add-extended S+C into D	Z & (Result = 0)	Unsigned Carry	1	4
110011 001i 1111 ddddddddd sssssssss	SUBX	D,S	Subtract-extended S+C from D	Z & (Result = 0)	Unsigned Borrow	1	4
110011 000i 1111 ddddddddd sssssssss	CMPX	D,S	Compare-extended D to S+C	Z & (Result = 0)	Unsigned Borrow	0	4
110100 001i 1111 ddddddddd sssssssss	ADDS	D,S	Add-signed S into D	Result = 0	Signed Overflow	1	4
110101 001i 1111 ddddddddd sssssssss	SUBS	D,S	Subtract-signed S from D	Result = 0	Signed Overflow	1	4
110110 001i 1111 ddddddddd sssssssss	ADDSX	D,S	Add-signed-extended S+C into D	Z & (Result = 0)	Signed Overflow	1	4
110111 001i 1111 ddddddddd sssssssss	SUBSX	D,S	Subtract-signed-extended S+C from D	Z & (Result = 0)	Signed Overflow	1	4
111000 001i 1111 ddddddddd sssssssss	CMPSUB	D,S	Subtract S from D if D => S	D = S	Unsigned (D => S)	1	4
111001 001i 1111 ddddddddd sssssssss	DJNZ	D,S	Dec D, jump if not zero to S (no jump = 8 clocks)	Result = 0	Unsigned Borrow	1	4 or 8
111010 000i 1111 ddddddddd sssssssss	TJNZ	D,S	Test D, jump if not zero to S (no jump = 8 clocks)	Result = 0	0	0	4 or 8
111011 000i 1111 ddddddddd sssssssss	TJZ	D,S	Test D, jump if zero to S (no jump = 8 clocks)	Result = 0	0	0	4 or 8
111100 000i 1111 ddddddddd sssssssss	WAITPEQ	D,S	Wait for pins equal - (INA & S) = D	-	-	0	5+
111101 000i 1111 ddddddddd sssssssss	WAITPNE	D,S	Wait for pins not equal - (INA & S) != D	-	-	0	5+
111110 001i 1111 ddddddddd sssssssss	WAITCNT	D,S	Wait for CNT = D, then add S into D	-	Unsigned Carry	1	5+
111111 000i 1111 ddddddddd sssssssss	WAITVID	D,S	Wait for video peripheral to grab D and S	-	-	0	5+
------ ---- 0000 --------- ---------	NOP		No operation, just elapses 4 clocks	-	-	–	4

* The Hub allows each cog an opportunity to execute a Hub instruction every 16 clocks. Because each cog runs independently of the Hub, each cog must sync to the Hub when executing a Hub instruction. This will cause a Hub instruction to take between 7 and 22 clocks. Afterwards, there will be 9 free clocks before a subsequent Hub instruction can execute and take the minimal 7 clocks. This is enough time to execute two 4-clock instructions without missing the next sync. So, to minimize clock waste, you can insert two normal instructions between any two otherwise-contiguous Hub instructions, without any increase in execution time. Beware that Hub instructions can cause execution timing to appear indeterminate - particularly, the first Hub instruction in a sequence.

5.4.1. Assembly Conditions

Condition	Instruction Executes
IF_ALWAYS	always
IF_NEVER	never
IF_E	if equal (Z)
IF_NE	if not equal (!Z)
IF_A	if above (!C & !Z)
IF_B	if below (C)
IF_AE	if above/equal (!C)
IF_BE	if below/equal (C | Z)
IF_C	if C set
IF_NC	if C clear
IF_Z	if Z set
IF_NZ	if Z clear
IF_C_EQ_Z	if C equal to Z
IF_C_NE_Z	if C not equal to Z
IF_C_AND_Z	if C set and Z set
IF_C_AND_NZ	if C set and Z clear
IF_NC_AND_Z	if C clear and Z set
IF_NC_AND_NZ	if C clear and Z clear
IF_C_OR_Z	if C set or Z set
IF_C_OR_NZ	if C set or Z clear
IF_NC_OR_Z	if C clear or Z set
IF_NC_OR_NZ	if C clear or Z clear
IF_Z_EQ_C	if Z equal to C
IF_Z_NE_C	if Z not equal to C
IF_Z_AND_C	if Z set and C set
IF_Z_AND_NC	if Z set and C clear
IF_NZ_AND_C	if Z clear and C set
IF_NZ_AND_NC	if Z clear and C clear
IF_Z_OR_C	if Z set or C set
IF_Z_OR_NC	if Z set or C clear
IF_NZ_OR_C	if Z clear or C set
IF_NZ_OR_NC	if Z clear or C clear

5.4.2. Assembly Directives

Directive	Description
FIT ⟨*Address*⟩	Validate previous instr/data fit below an address.
ORG ⟨*Address*⟩	Adjust compile-time cog address pointer.
⟨*Symbol*⟩ RES ⟨*Count*⟩	Reserve next long(s) for symbol.

5.4.3. Assembly Effects

Effect	Results In
WC	C Flag modified
WZ	Z Flag modified
WR	Destination Register modified
NR	Destination Register not modified

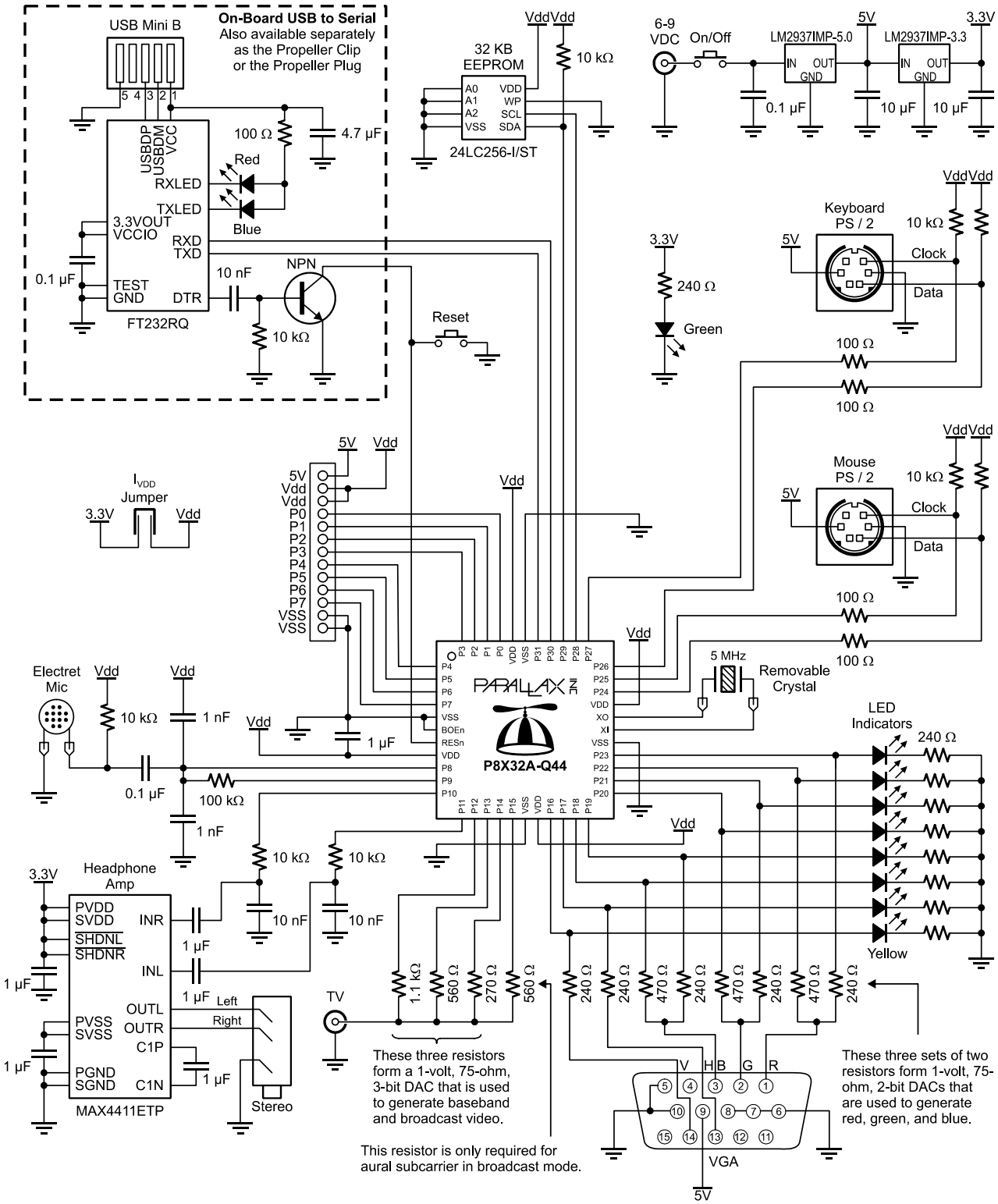
5.4.4. Assembly Operators

Propeller Assembly code can contain constant expressions, and those expressions may use any operators that are allowed in constant expressions. The table below summarizes the operators allowed in Propeller Assembly code; it is a subset of those shown in Section 5.2.

Operator	Description
+	Add
+	Positive (+X); unary form of Add
-	Subtract
-	Negate (-X); unary form of Subtract
*	Multiply and return lower 32 bits (signed)
**	Multiply and return upper 32 bits (signed)
/	Divide (signed)
//	Modulus (signed)
#>	Limit minimum (signed)
<#	Limit maximum (signed)
^^	Square root; unary
||	Absolute value; unary
~>	Shift arithmetic right
|<	Bitwise: Decode value (0-31) into single-high-bit long; unary
>|	Bitwise: Encode long into value (0 - 32) as high-bit priority; unary
<<	Bitwise: Shift left
>>	Bitwise: Shift right
<-	Bitwise: Rotate left
->	Bitwise: Rotate right
><	Bitwise: Reverse
&	Bitwise: AND
|	Bitwise: OR
^	Bitwise: XOR
!	Bitwise: NOT; unary
AND	Boolean: AND (promotes non-0 to -1)
OR	Boolean: OR (promotes non-0 to -1)
NOT	Boolean: NOT (promotes non-0 to -1); unary
==	Boolean: Is equal
<>	Boolean: Is not equal
<	Boolean: Is less than (signed)
>	Boolean: Is greater than (signed)
=<	Boolean: Is equal or less (signed)
=>	Boolean: Is equal or greater (signed)
@	Symbol address; unary

6.0 PROPELLER DEMO BOARD SCHEMATIC

The Propeller Demo Board (Stock #32100) provides convenient connections to 32K EEPROM, replaceable 5 MHz crystal, 3.3V and 5V regulators, USB-to-serial programming/communication interface, VGA and NTSC video output, stereo output with 16Ω headphone amplifier, microphone input, two PS2 mouse and keyboard jacks, eight LEDs, eight free I/O pins brought to a header for breadboard for prototyping, and a ground post for an oscilloscope probe. Overall PCB size: 3" x 3".



7.0 ELECTRICAL CHARACTERISTICS

7.1. Absolute Maximum Ratings

Stresses in excess of the absolute maximum ratings can cause permanent damage to the device. These are absolute stress ratings only. Functional operation of the device is not implied at these or any other conditions in excess of those given in the remainder of Section 6.0. Exposure to absolute maximum ratings for extended periods can adversely affect device reliability.

Table 9: Absolute Maximum Ratings	
Ambient temperature under bias [1]	0 °C to +70 °C
Storage temperature	-65 °C to +150 °C
Voltage on V_{dd} with respect to V_{ss}	-0.3 V to +4.0 V
Voltage on all other pins with respect to V_{ss}	-0.3 V to (V_{dd} + 0.3V)
Total power dissipation	1 W
Max. current out of V_{ss} pins	300 mA
Max. current into V_{dd} pins	300 mA
Max. DC current into an input pin with internal protection diode forward biased	±500 µA
Max. allowable current per I/O pin	40 mA
ESD (Human Body Model) Supply pins	3 kV
ESD (Human Body Model) all non-supply pins	8 kV

[1] Ambient temperature under bias has not been tested; the listed range is a very conservative estimate and will be greatly expanded in the non-Preliminary datasheet.

7.2. DC Characteristics

(Simulated temperature range: -40° C < T_a < +125° C unless otherwise noted)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{dd}	Supply Voltage		2.7	-	3.6	V
V_{ih}, V_{il}	Logic High Logic Low		0.6 V_{dd} V_{ss}		V_{dd} 0.3 V_{dd}	V V
I_{il}	Input Leakage Current	V_{in} = V_{dd} or V_{ss}	-1.0		+1.0	µA
V_{oh}	Output High Voltage	I_{oh} = 10 mA, V_{dd} = 3.3 V	2.85			V
V_{ol}	Output Low Voltage	I_{ol} = 10 mA, V_{dd} = 3.3 V			0.4	V
I_{BO}	Brownout Detector Current			3.8		µA
I	Quiescent Current	RESn = 0V, BOEn = V_{dd}, P_0-P_{31}=0V		600		nA

Note: Data in the Typical ("Typ") column is T_a = 25 °C unless otherwise stated.

7.3. AC Characteristics

(Simulated temperature range: -40°C < T_a < +125°C unless otherwise noted)

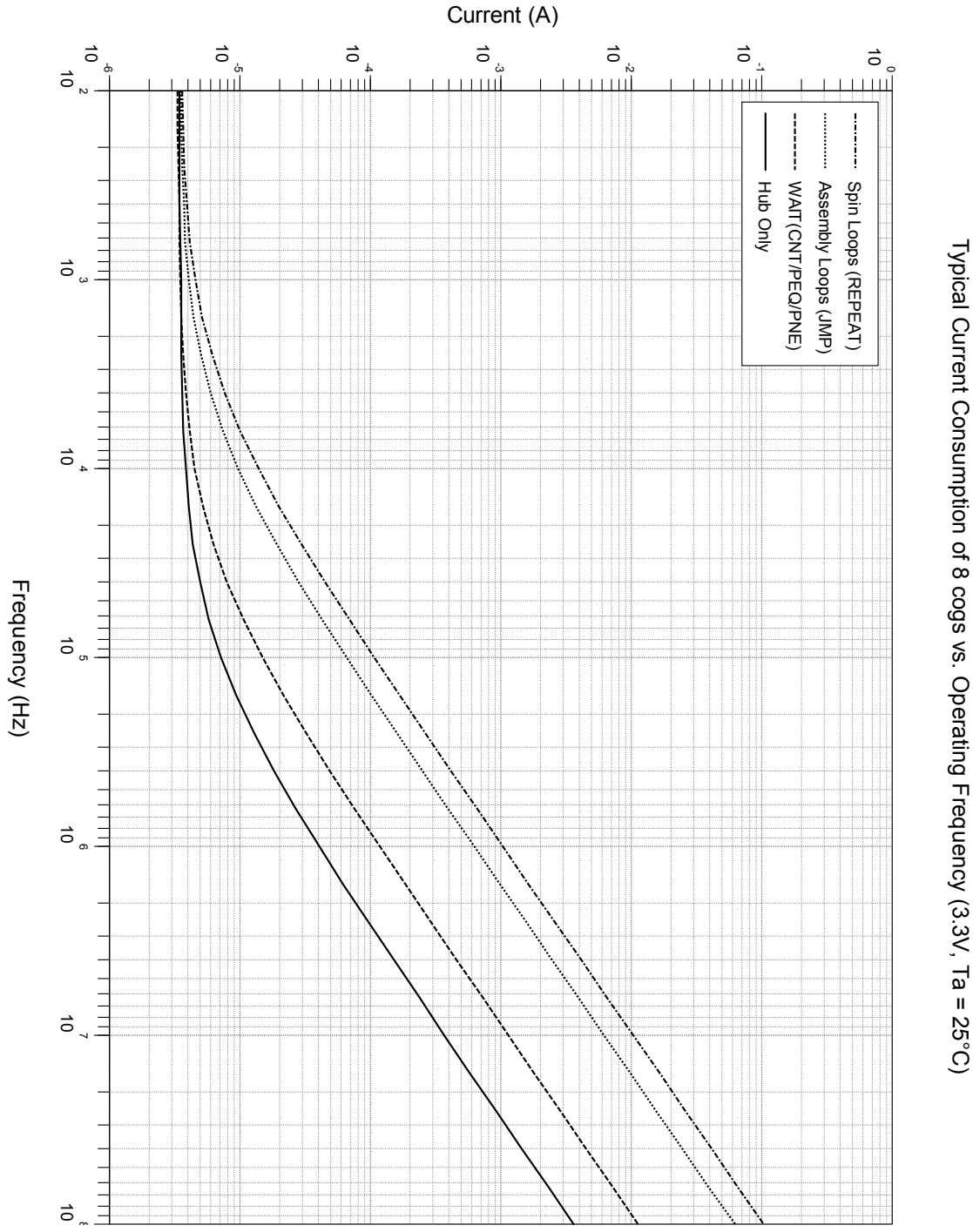
Symbol	Parameter	Min	Typ	Max	Units	Condition
F_{osc}	External XI Frequency	DC	-	80	MHz	
	Oscillator Frequency	DC 13 8 4	- 20 12 -	80 33 20 8	MHz kHz MHz MHz	Direct drive (no PLL) RCSLOW RCFAST Crystal using PLL
C_{in}	Input Capacitance		6	-	pF	

Note: Data in the Typical ("Typ") column is T_a = 25 °C unless otherwise stated.

8.0 CURRENT CONSUMPTION CHARACTERISTICS

8.1. Typical Current Consumption of 8 Cogs vs. Operating Frequency

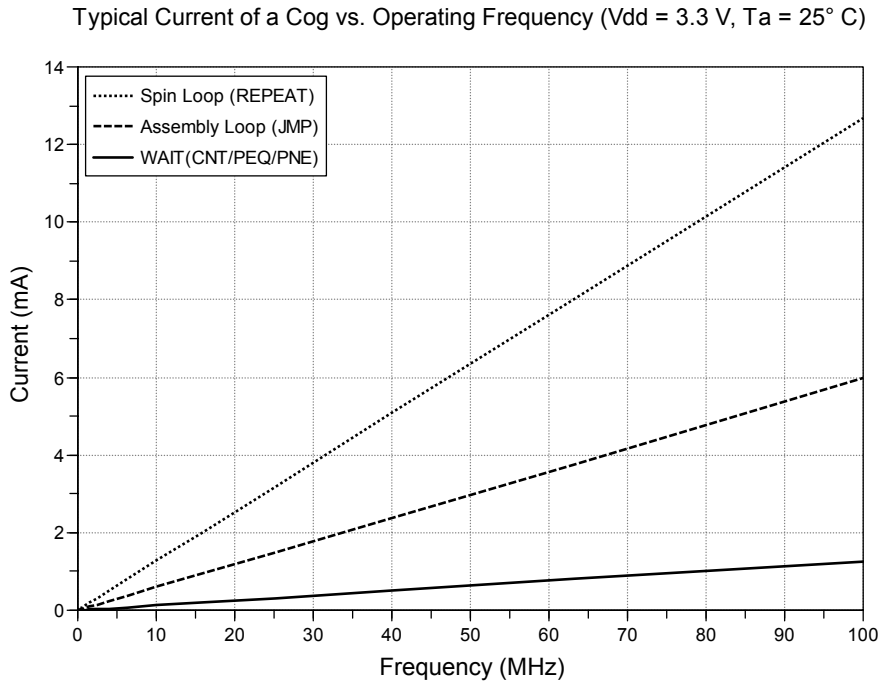
This figure shows the typical current consumption of the Propeller under various operating conditions duplicated across all cogs. Brown out circuitry and the Phase-Locked Loop were disabled for the duration of the test.



Typical Current Consumption of 8 cogs vs. Operating Frequency (3.3V, Ta = 25°C)

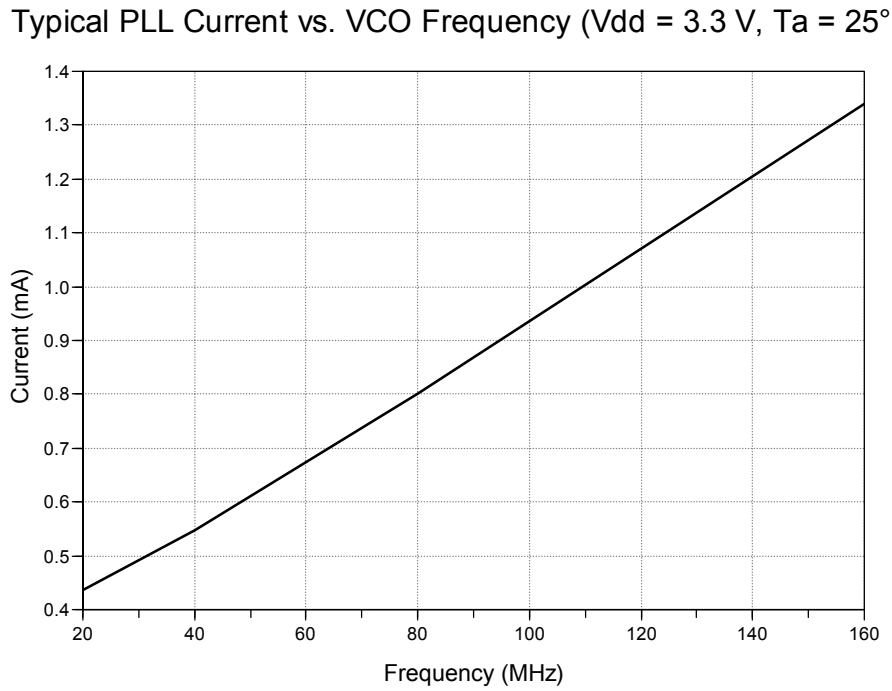
8.2. Typical Current of a Cog vs. Operating Frequency

This graph shows a cog's typical current consumption under various conditions, in isolation of other sources of current within the Propeller chip.



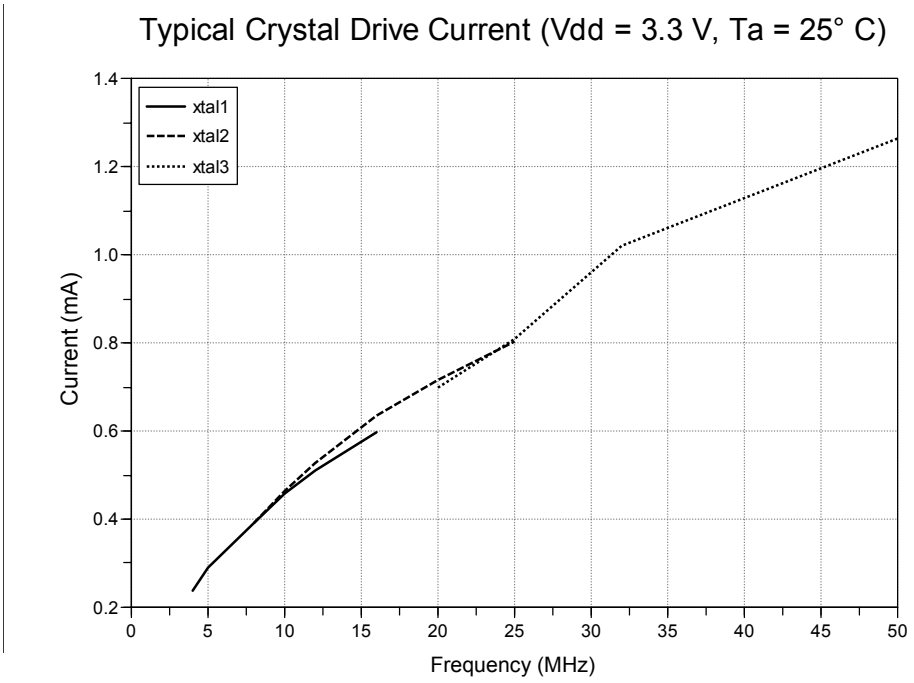
8.3. Typical PLL Current vs. VCO Frequency

This graph shows the typical amount of current consumed by a Phase-Locked Loop as a function of the frequency of the Voltage Controlled Oscillator which is 16 times the frequency of the input clock.



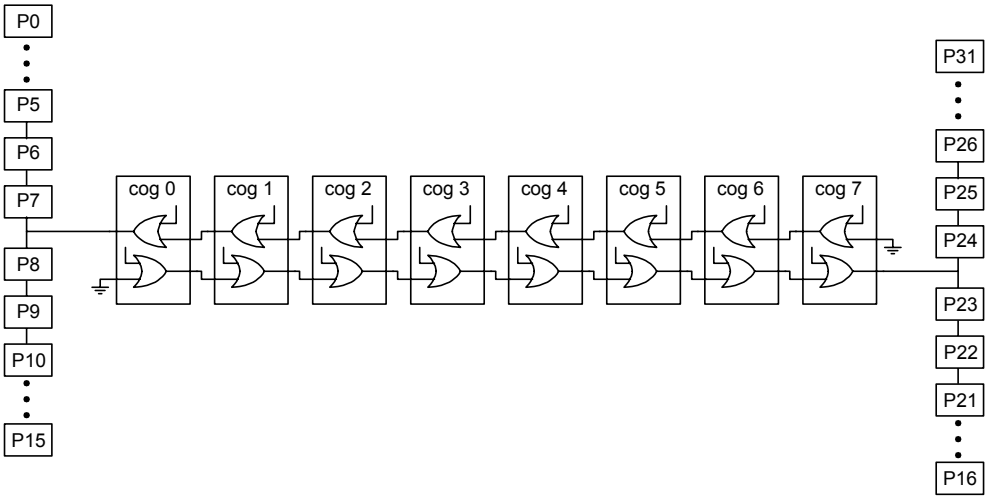
8.4. Typical Crystal Drive Current

This graph shows the current consumption of the crystal driver over a range of crystal frequencies and crystal settings, all data points above 25 MHz were obtained by using a resonator since the driver does not perform 3rd harmonic overtone driving required for crystals over 25 MHz.



8.5. Cog and I/O Pin Relationship

The figure below illustrates the physical relationship between the cogs and I/O pins. While there can be a 1 to 1.5 ns propagation delay in output transitions between the shortest and longest paths, the purpose of the figure is to illustrate the length of leads and their associated parasitic capacitance. This capacitance increases the amount of energy required to transition a pin's state and therefore increases the current draw for toggling a pin. So the current consumed by Cog 7 toggling P0 at 20 MHz will be greater than Cog 0 toggling P7 at 20 MHz. The amount of current consumed by transitioning a pin's state is dependent on many factors including: temperature, frequency of transitions, external load and internal load. As mentioned the internal load is dependent upon which cog and pin are used. Internal load current for room temperature toggling of a pin at 20 MHz for a Propeller in a DIP package varies on the order of 300 μA.



8.6. Current Profile at Various Startup Conditions

The diagrams below show the current profile for various startup conditions of the Propeller chip dependent upon the presence of an EEPROM and PC.

Figure 6

Boot Sequence Current Profile for no PC and no EEPROM (P31 held low and P29 not connected (same as held low)).

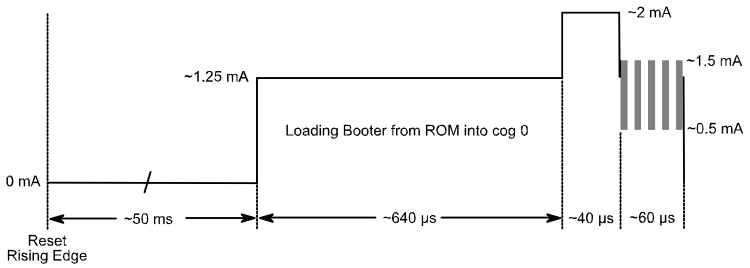


Figure 7

Boot Sequence Current Profile for PC (connected but idle) and no EEPROM. (P31 held high and P29 not connected).

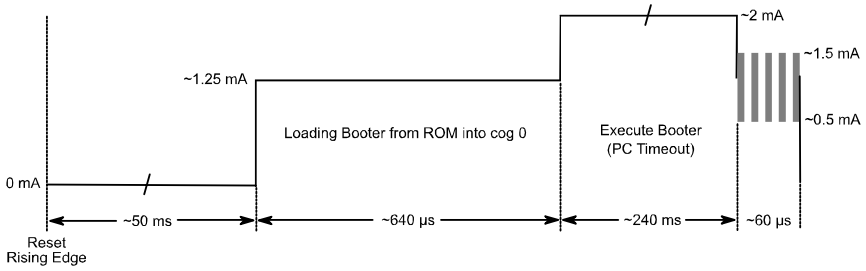


Figure 8

Boot Sequence Current Profile for no PC and no EEPROM (P31 held low and P29 held high).

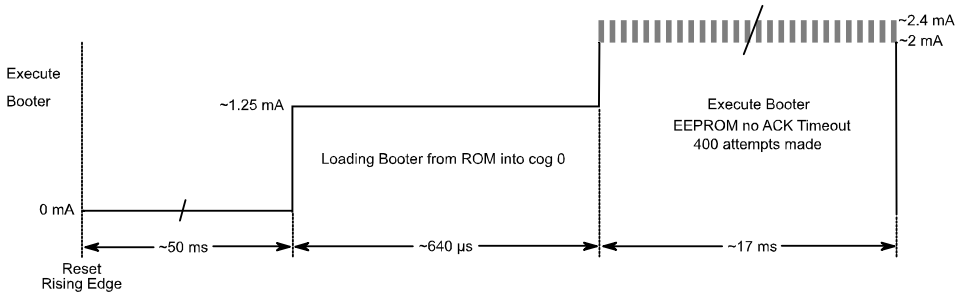


Figure 9

Boot Sequence Current Profile for no PC and EEPROM (P31 held low and P29 connected to EEPROM SDA).

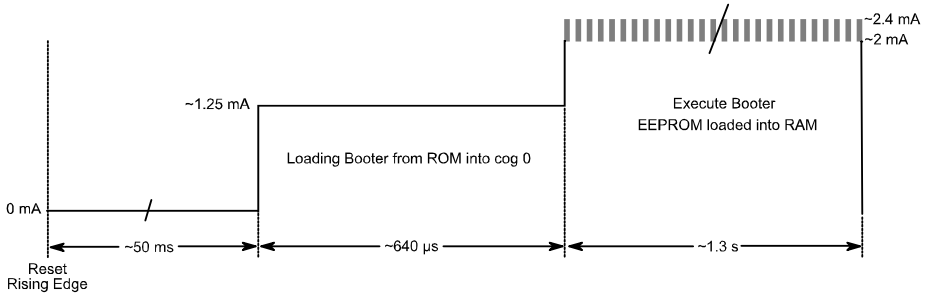
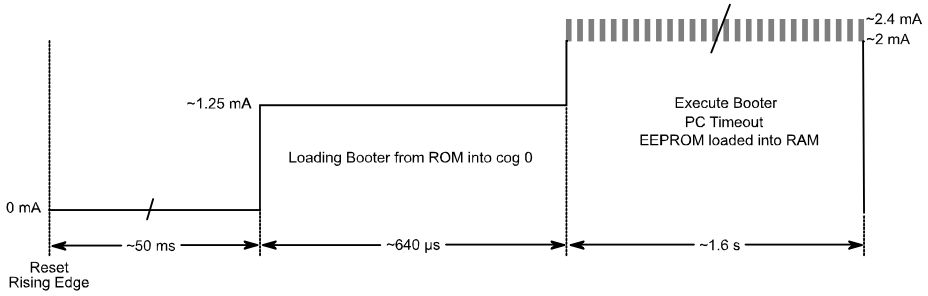


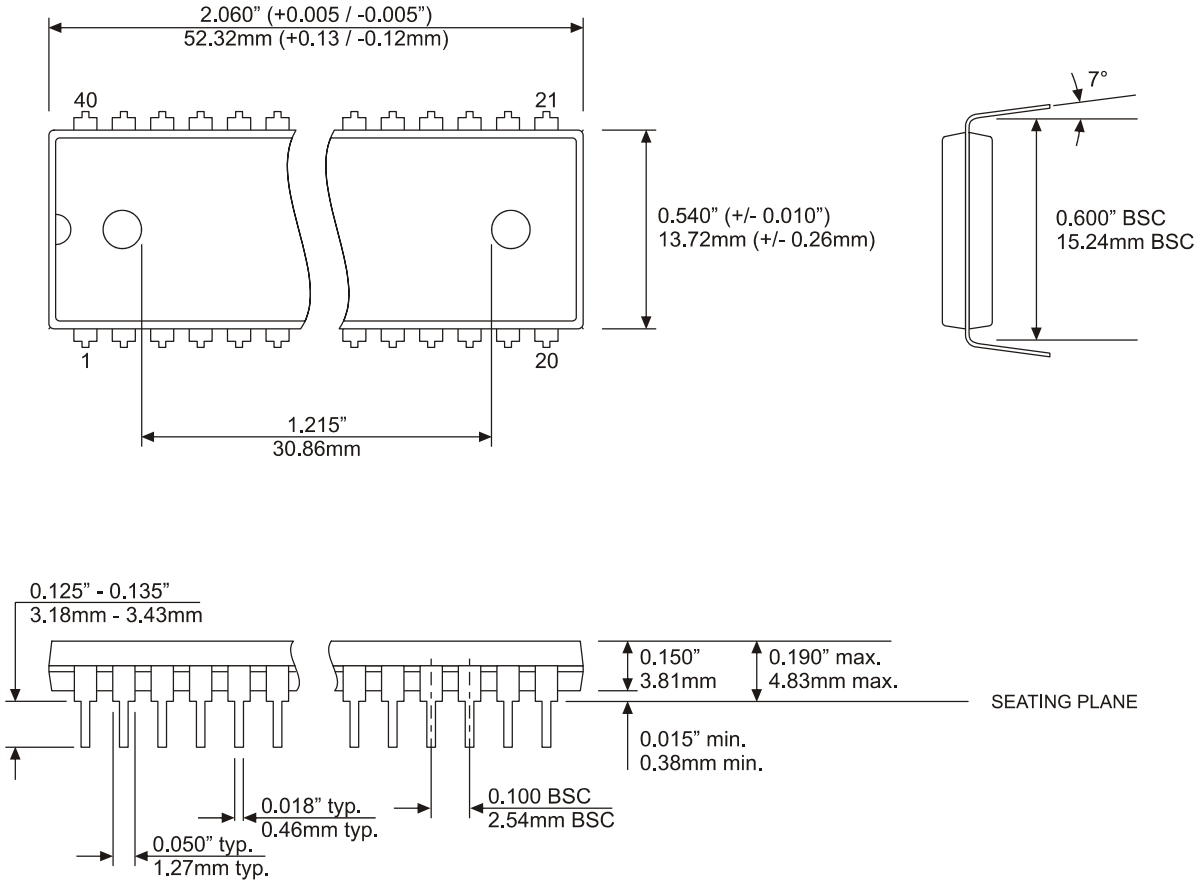
Figure 10

Boot Sequence Current Profile for PC (connected but idle) and EEPROM (P31 held high and P29 connected to EEPROM SDA).

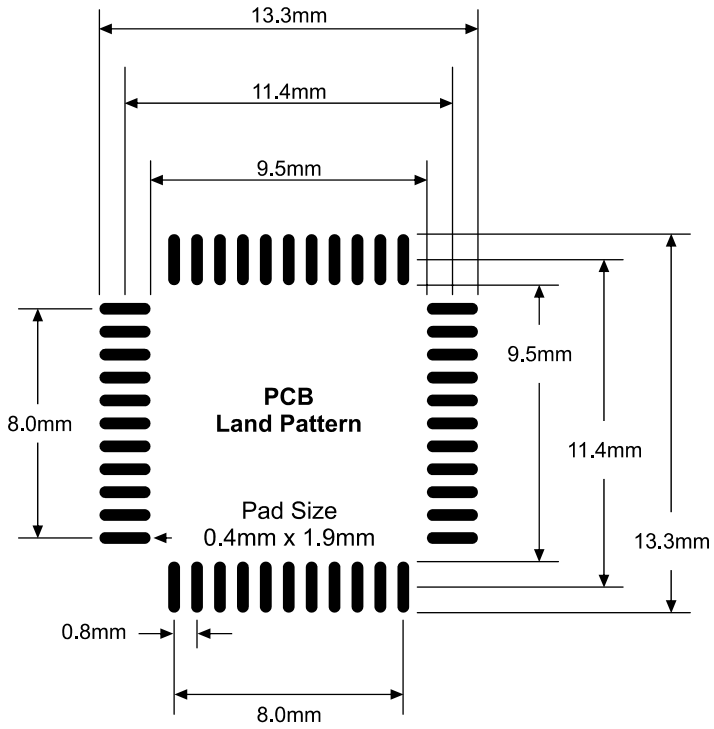


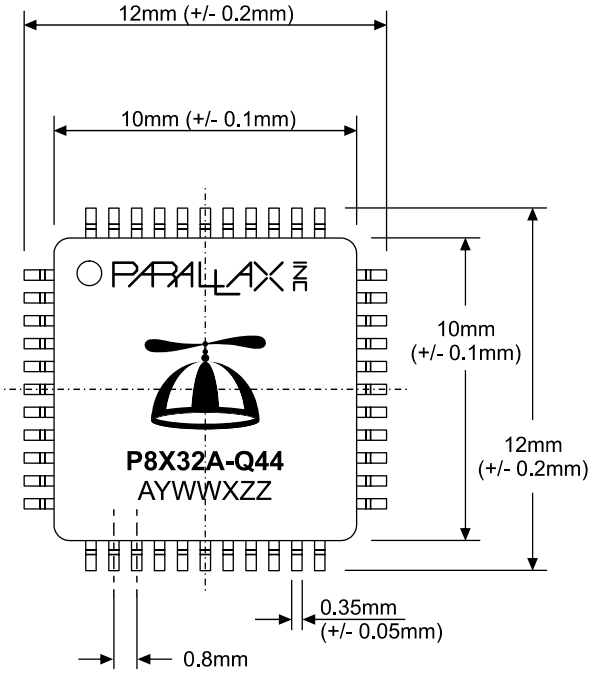
9.0 PACKAGE DIMENSIONS

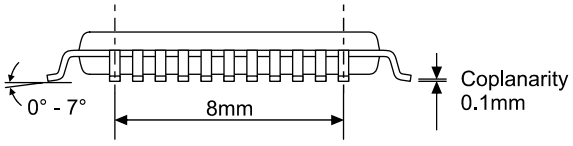
9.1. P8X32A-D40 (40-pin DIP)

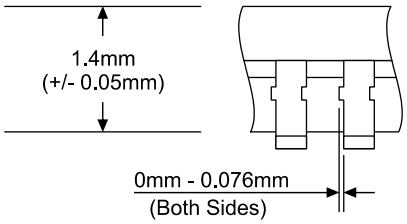


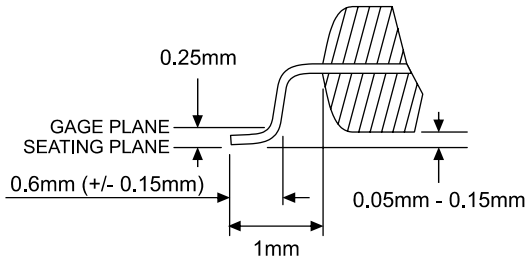
9.2. P8X32A-Q44 (44-pin LQFP)



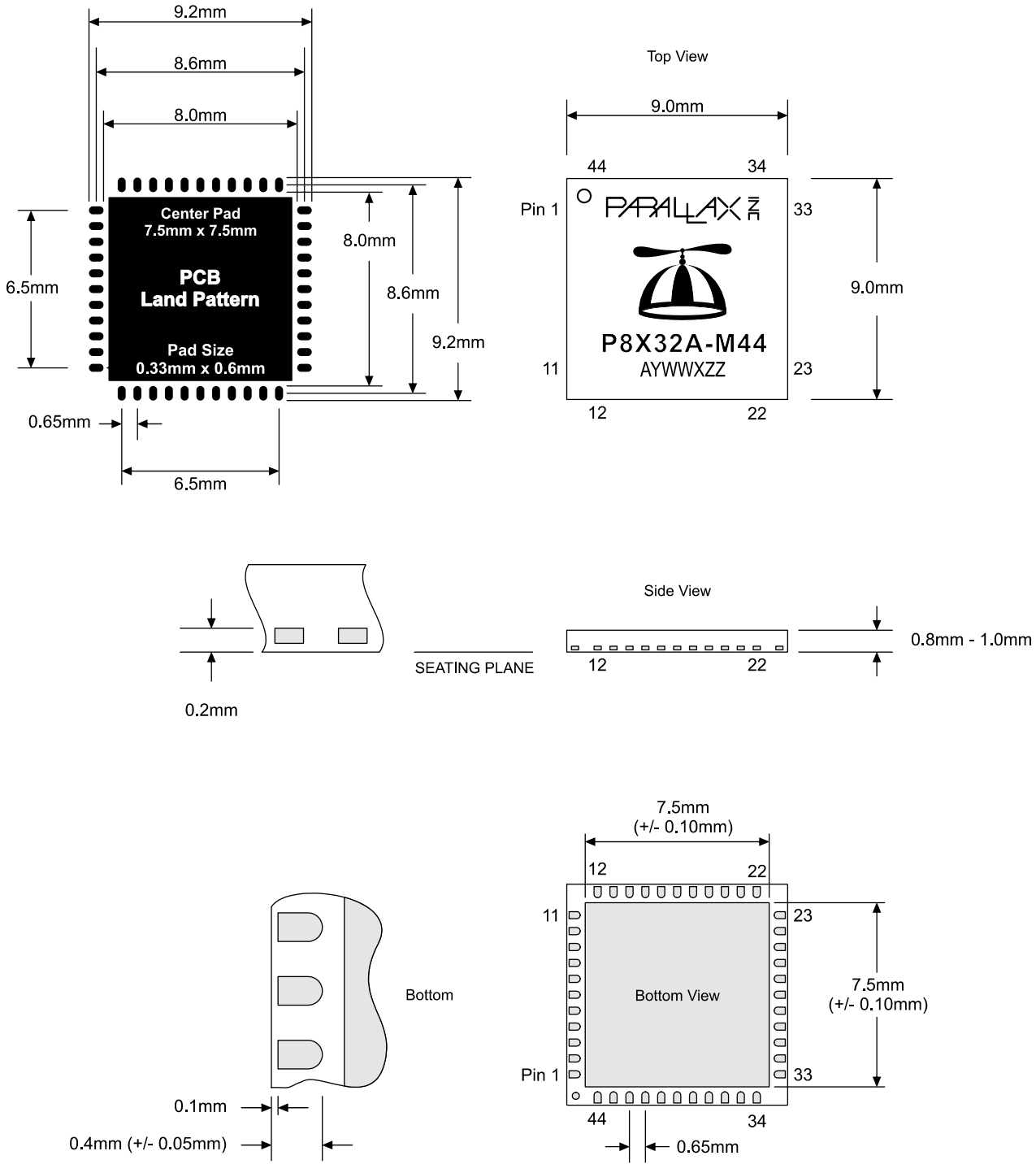








9.3. P8X32A-M44 (44-pin QFN)



10.0 MANUFACTURING INFORMATION

10.1. Reflow Peak Temperature

Package Type	Reflow Peak Temp.
DIP	255+5/-0 °C
LQFP	255+5/-0 °C
QFN	255+5/-0 °C

10.2. Green/RoHS Compliance

All Parallax Propeller chip models are certified Green/RoHS Compliant. The Certificate of Compliance is available upon request and be obtained by contacting the Parallax Sales Team.

Parallax Sales and Tech Support Contact Information

For the latest information on Propeller chips and programming tools, development boards, instructional materials, and application examples, please visit www.parallax.com/propeller.

Parallax, Inc.
599 Menlo Drive, Suite 100
Rocklin, CA 95765

Sales/Tech Support: (916) 624-8333
Toll Free in the US: 1-888-512-1024
Sales: sales@parallax.com
Tech Support: support@parallax.com