Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Active
Core Processor	-
Core Size	32-Bit 8-Core
Speed	80MHz
Connectivity	-
Peripherals	-
Number of I/O	32
Program Memory Size	32KB (32K x 8)
Program Memory Type	ROM
EEPROM Size	-
RAM Size	32K x 8
Voltage - Supply (Vcc/Vdd)	-
Data Converters	-
Oscillator Type	Internal
Operating Temperature	-
Mounting Type	Through Hole
Package / Case	40-DIP (0.600", 15.24mm)
Supplier Device Package	40-DIP
Purchase URL	https://www.e-xfl.com/product-detail/parallax/p8x32a-d40

Table of Contents

1.3. Key Features and Benefits

The P8X32A design frees developers from common complexities of embedded systems programming.

1.3.1. 32-bit Multicore Architecture

- True parallel processing with eight symmetric 32-bit processors (cogs) in one microcontroller
- Multi-cog run-time management (run/wait/stop) easily solves event-handling problems and eliminates the need for interrupts. This greatly simplifies programming for asynchronous and synchronous events, resulting in a responsive and easily maintained application.
- 20 MIPS per cog, 160 MIPS total with all cogs running
- Solves mixed-bandwidth needs common to embedded applications
- Multi-purpose design lowers part count while increasing system capabilities
- Developer-driven cog assignments bring flexible response and deterministic timing to embedded applications

1.3.2. Clock System and Wait Instructions

- Flexible Clock Modes
  - Two internal, one external, plus optional 1x–16x PLL; up to 80 MHz system clock
  - Switchable in code at run-time; low frequency for low-power periods, high frequency for high-bandwidth moments
- Shared System Clock facilitates synchronization between cogs
- `WAIT` Instructions
  - Deliver powerful synchronous / asynchronous event management
  - Set dedicated event cogs to an "always ready," very low power state

1.3.3. Programming Languages and Resources

- Spin (object-based, high-level) and Assembly (PASM; low-level); used together for thorough development, i.e. fast development in Spin plus fast execution with prewritten high-speed PASM drivers
- Third-party support: C, BASIC, and more
- Enhanced Assembly Language
  - Conditional execution for individual instructions; enables jitter-free signal generation and event handling
  - Optional flag and result writing for individual instructions
- Open-source Objects
  - Objects are shared freely via the Propeller Object Exchange and Propeller Tool libraries
  - Select objects that fit a need, easily integrate them into a Propeller application

1.3.4. Flexible I/O and Peripheral Interface

- 32 I/O Pins
  - All general-purpose I/O after boot-up; accessible by every cog simultaneously
  - Single-instruction access to any individual I/O pin or any contiguous I/O pin group
  - Easily move designed functions between pins for simple system board layout
- Multi-function Counters
  - Configurable state machines generate or sense repetitive signals per clock cycle
  - Measure frequency, detect edges, count cycles, D/A or A/D conversion, and more
  - Operate autonomously with optional run-time monitoring and adjusting
  - Two counters per cog
- Video Generators
  - RGB: VGA; 8 I/O pins
  - Composite: NTSC, PAL; 1-pin (B/W), 3-pin (typical), or 4-pin (optional)
  - One generator per cog
- Software Peripherals
  - Peripheral interfaces built with software and inexpensive passive components; not single-function on-chip hardware
  - Software-based interfaces are flexible; enhance as peripheral needs arise — no need to redesign with a chip variant

1.4. Applications

The P8X32A is particularly useful in projects that can be vastly simplified with simultaneous processing, including:

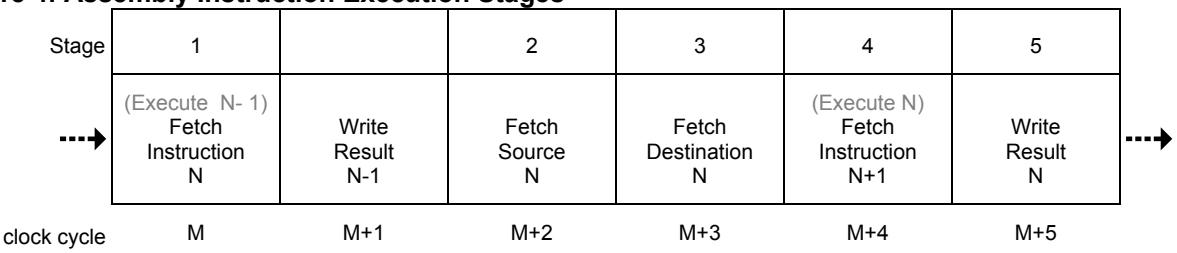
- Industrial control systems
- Sensor integration, signal processing, and data acquisition
- Handheld portable human-interface terminals
- Motor and actuator control
- User interfaces requiring NTSC, PAL, or VGA output, with PS/2 keyboard and mouse input
- Low-cost video game systems
- Industrial, educational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

1.4.1. Corporate and Community Support

- Sales or technical support: (916) 632-4664
- Email sales: sales@parallaxsemiconductor.com
- Email support: support@parallaxsemiconductor.com
- Engineer-moderated Parallax Semiconductor sub-forum is available from http://forums.parallax.com
- Parallax-hosted Propeller Object Exchange library: http://obex.parallax.com

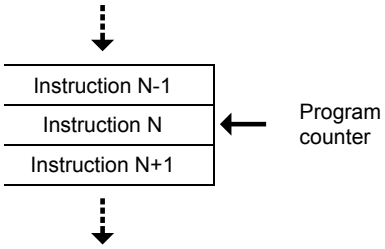
4.8. Assembly Instruction Execution Stages

Figure 4: Assembly Instruction Execution Stages



The Propeller executes assembly instructions in five stages. While an entire instruction takes six cycles to execute, two of those clock cycles are dedicated to the two adjacent instructions. This results in an overall throughput of four clock cycles per instruction.

Figure 5 Cog Memory



In Stage 1, instruction N, pointed to by the Program Counter, is fetched from cog memory during clock cycle M. During cycle M+1 the result from the previous instruction is written to memory. The reason the previous instruction result is written after the current instruction is fetched will be explained shortly.

During Stage 2, if the immediate flag of Instruction N is set, the 9 bit source field is saved as the source value. If the value is not immediate, the location specified by the source field is fetched from cog memory during clock cycle M+2. During clock cycle M+3 the location specified in the destination field is fetched from cog memory (Stage 3).

At this point in time (Stage 4) the Arithmetic Logic Unit (ALU) has all the information needed to execute the instruction. Executing the instruction takes some amount of time before the result is available. The amount of time required for execution is dictated by the slowest operation the ALU performs. To provide enough time for the ALU to execute the instruction, a full clock cycle (M+4) is provided to the ALU for the result to settle into its final state. During this execution, the cog memory is not accessed by instruction N. To speed up the throughput of program execution, the next instruction to be executed is fetched from cog memory while the current instruction is executed in the ALU.

Finally at clock cycle M+5 the result of the current instruction N is written back to cog memory, completing Stage 5.

The partial interleaving of instructions has a couple implications to program flow. First, when code modification occurs through **MOVI**, **MOVS**, **MOVD** or any operations which modifies an assembly instruction, there must be at least one instruction executed before the modified instruction is executed. If the modification is done on the immediately following instruction (N+1), the unmodified version of instruction N+1 will be loaded a clock cycle before the modified version of instruction N+1 is written to cog memory.

Second, conditional jumps do not know for certain if they will jump until the end of clock cycle M+4. Since the next instruction has already been fetched, only one of the two possible branches can be predicted. In the Propeller, conditional branches are always predicted to take the jump. For loops using **DJNZ** where the jump is taken every time except the final loop, a tighter execution time of the loop is achieved.

In the event the jump is not taken, the cog takes no action until the next instruction is fetched. This is equivalent to a **NOP** being inserted before the next instruction is executed. Unconditional jumps always take four clock cycles to execute since the Propeller can always accurately predict what address needs to be loaded into the Program Counter for the next instruction to execute. Examples of unconditional jumps include **JMP**, **JMPRET**, **CALL** and **RET**.

If an instruction needs to access any Hub resource, Stage 4 is extended until the Hub becomes available, increasing execution time to at least 8 and potentially up to 23 clock cycles. See Section 4.4: Hub on page 7.

4.11. 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 6.3 and 6.4) since it automatically updates all the above-mentioned locations with the proper information.

Table 13: 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 14: 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 10 ms 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 (pad only)	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'

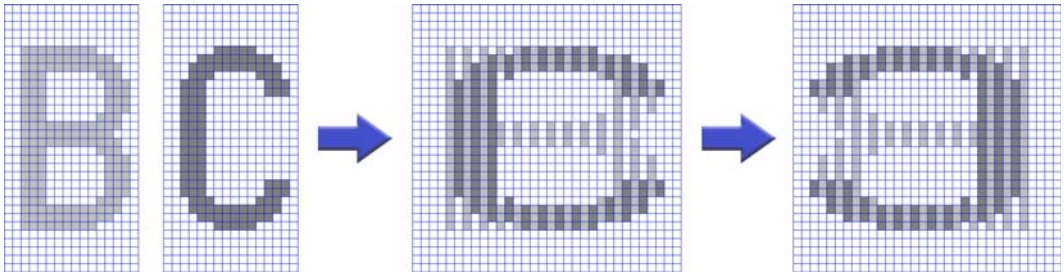


Figure 8

Propeller Character Interleaving

As shown in Figure 8, The character pairs are merged row-by-row such that each character's 16 horizontal pixels are spaced apart and interleaved with their neighbors' so 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's video hardware 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. 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.

5.1.4. Math Function Tables

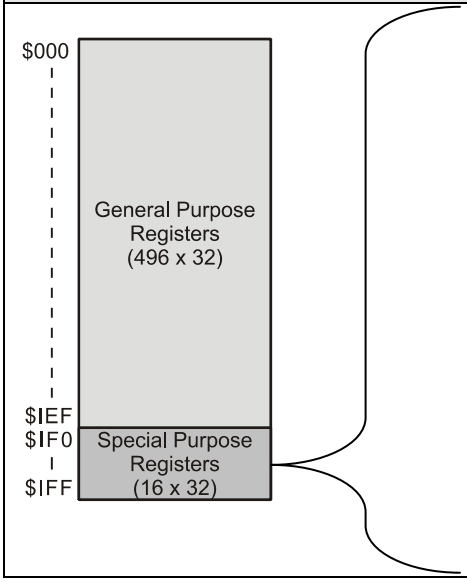
Base-2 Log and Anti-Log tables, each with 2048 unsigned words, facilitate converting values to and from exponent form to facilitate some operations; see the Propeller Manual for access instructions. Also, a sine table provides 2049 unsigned 16-bit sine samples spanning 0° to 90° inclusively (0.0439°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

5.2. Cog RAM

As stated in Section 4.3, the Cog RAM is used for executable code, data, and variables, and the last 16 locations serve as interfaces to the System Counter, I/O pins, and local cog peripherals (see Table 15). Cog RAM is long-addressable only.

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 15: 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[3]
	$1F4	OUTA	Read/Write	Output States for P31 - P0
	$1F5	OUTB	Read/Write	Output States for P63 – P32[3]
	$1F6	DIRA	Read/Write	Direction States for P31 - P0
	$1F7	DIRB	Read/Write	Direction States for P63 - P32[3]
	$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[2]	Counter A Phase:
	$1FD	PHSB	Read/Write[2]	Counter B Phase
	$1FE	VCFG	Read/Write	Video Configuration
	$1FF	VSCL	Read/Write	Video Scale

Note 1: Only accessible as a source register (i.e. `MOV` *Dest, Source*).
Note 2: Only readable as a Source Register (i.e. `MOV` *Dest, Source*); read-modify-write not possible as a Destination Register.
Note 3: Reserved for future use.

6.0 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.

6.1. Reserved Word List

All words listed are always reserved, whether programming in Spin or in Propeller Assembly. **As of Propeller Tool v1.05:**

Table 16: 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

6.1.1. 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.

6.2. Math and Logic Operators

Table 17: 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.

6.3. 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.
〈*Symbol*〉 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* 〈[*Offset*]〉 〈((, ¦ ↳)) *Symbol* 〈[*Offset*]〉 ...		Declare global enumerations (incrementing symbolic constants).
CON *Symbol* 〈[*Offset*]〉 〈((, ¦ ↳)) *Symbol* 〈[*Offset*]〉〉...		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*〉 〈[*Count*]〉 〈,〈*Size*〉 *Data* 〈[*Count*]〉〉...		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. Default is 0 (input) upon cog startup.
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.

Spin Command	Returns Value	Description
((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.
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.
⟨*Symbol*⟩ 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. Default is 0 (ground) upon cog startup.
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.

6.4. 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 (1) or not writing (0) 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 Result	C Result	R	Clocks
000000 000i 1111 ddddddddd sssssssss	**WRBYTE** D,S	Write D[7..0] to main memory byte S[15..0]	-	-	0	8..23 *
000000 001i 1111 ddddddddd sssssssss	**RDBYTE** D,S	Read main memory byte S[15..0] into D (0-extended)	Result = 0	-	1	8..23 *
000001 000i 1111 ddddddddd sssssssss	**WRWORD** D,S	Write D[15..0] to main memory word S[15..1]	-	-	0	8..23 *
000001 001i 1111 ddddddddd sssssssss	**RDWORD** D,S	Read main memory word S[15..1] into D (0-extended)	Result = 0	-	1	8..23 *
000010 000i 1111 ddddddddd sssssssss	**WRLONG** D,S	Write D to main memory long S[15..2]	-	-	0	8..23 *
000010 001i 1111 ddddddddd sssssssss	**RDLONG** D,S	Read main memory long S[15..2] into D	Result = 0	-	1	8..23 *
000011 000i 1111 ddddddddd sssssssss	**HUBOP** D,S	Perform hub operation according to S	Result = 0	-	0	8..23 *
000011 0001 1111 ddddddddd ------000	**CLKSET** D	Set the global CLK register to D[7..0]	-	-	0	8..23 *
000011 0011 1111 ddddddddd ------001	**COGID** D	Get this cog number (0..7) into D	ID = 0	0	1	8..23 *
000011 0001 1111 ddddddddd ------010	**COGINIT** D	Initialize a cog according to D	ID = 0	No cog free	0	8..23 *
000011 0001 1111 ddddddddd ------011	**COGSTOP** D	Stop cog number D[2..0]	Stopped ID = 0	No Cog Free	0	8..23 *
000011 0011 1111 ddddddddd ------100	**LOCKNEW** D	Checkout a new LOCK number (0..7) into D	ID = 0	No lock free	1	8..23 *
000011 0001 1111 ddddddddd ------101	**LOCKRET** D	Return lock number D[2..0]	ID = 0	No lock free	0	8..23 *
000011 0001 1111 ddddddddd ------110	**LOCKSET** D	Set lock number D[2..0]	ID = 0	Prior lock state	0	8..23 *
000011 0001 1111 ddddddddd ------111	**LOCKCLR** D	Clear lock number D[2..0]	ID = 0	Prior lock state	0	8..23 *
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, set new MSB to 0	Result = 0	D[0]	1	4
001011 001i 1111 ddddddddd sssssssss	**SHL** D,S	Shift D left by S[4..0] bits, set new LSB to 0	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)	S = 0	Signed (D < S)	1	4
010001 001i 1111 ddddddddd sssssssss	**MAXS** D,S	Set D to S if signed (D => S)	S = 0	Signed (D < S)	1	4
010010 001i 1111 ddddddddd sssssssss	**MIN** D,S	Set D to S if unsigned (D < S)	S = 0	Unsigned (D < S)	1	4
010011 001i 1111 ddddddddd sssssssss	**MAX** D,S	Set D to S if unsigned (D => S)	S = 0	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
010111 000i 1111 --------- sssssssss	**JMP** S	Set PC to S[8..0]	Result = 0	-	0	4

6.4.1. 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

6.4.2. 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.

6.4.3. Assembly Effects

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

6.4.4. Assembly Operators

Propeller Assembly code can contain constant expressions, which may use any operators that are allowed in constant expressions. The table (a subset of Table 17) lists the operators allowed in Propeller Assembly.

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

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 0. Exposure to absolute maximum ratings for extended periods can adversely affect device reliability.

Table 18: Absolute Maximum Ratings	
Ambient temperature under bias	-55 °C to +125 °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} + 0.3 V)
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

***Note: I/O pin voltages with respect to Vss may be exceeded if internal protection diode forward bias current is not exceeded.**

7.2. DC Characteristics

(Operating temperature range: -55° 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

(Operating temperature range: -55°C < T_a < +125°C unless otherwise noted)

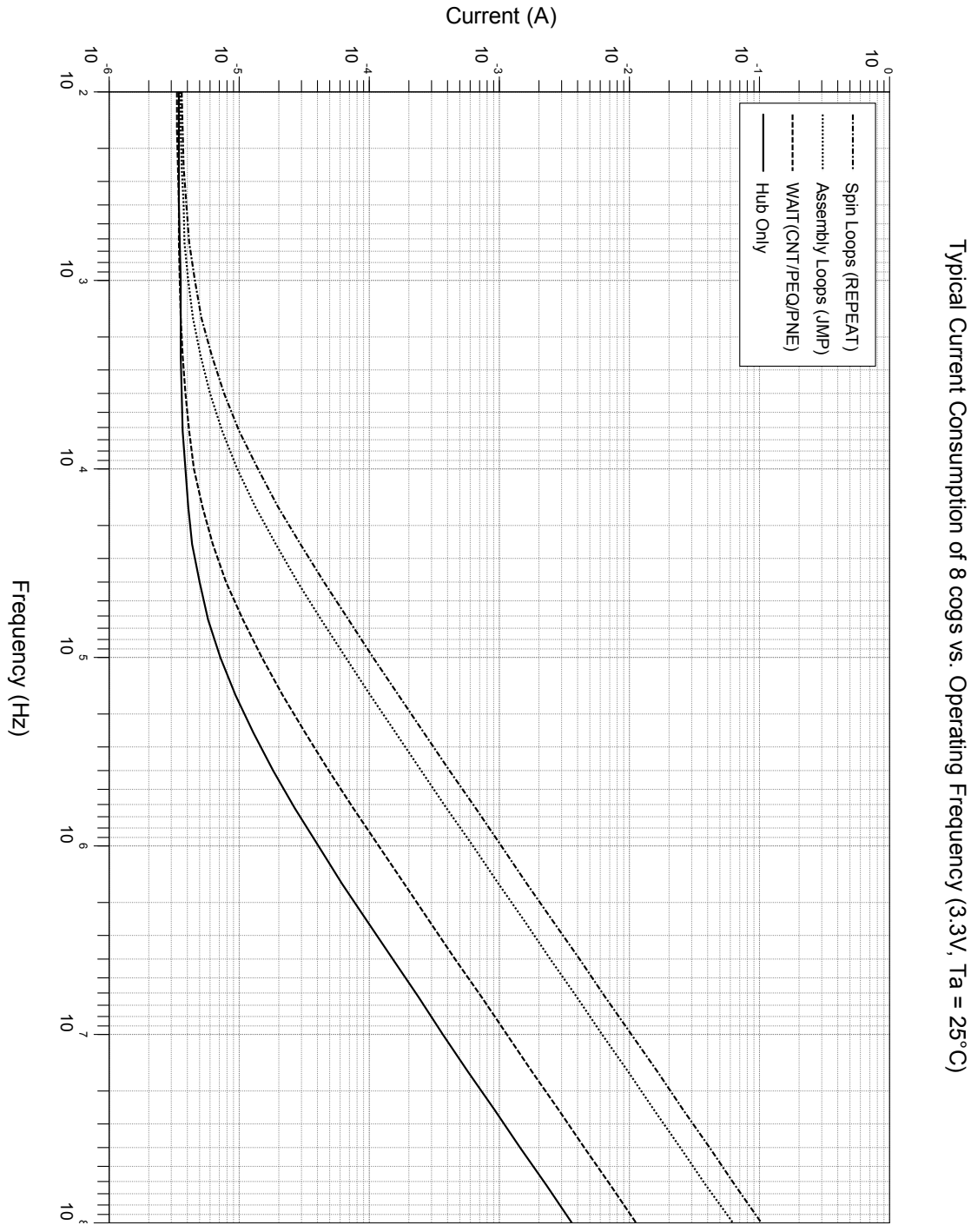
Symbol	Parameter	Min	Typ*	Max	Units	Condition
F_{osc}	External XI Frequency	DC	-	80	MHz	
	Oscillator Frequency	DC 13 8 4	- 20 12 -	80 33 20 8	MHz kHz MHz MHz	Direct drive (no PLL) RCSLOW RCFAST Crystal using PLL
C_{in}	Input Capacitance		6	-	pF	

***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

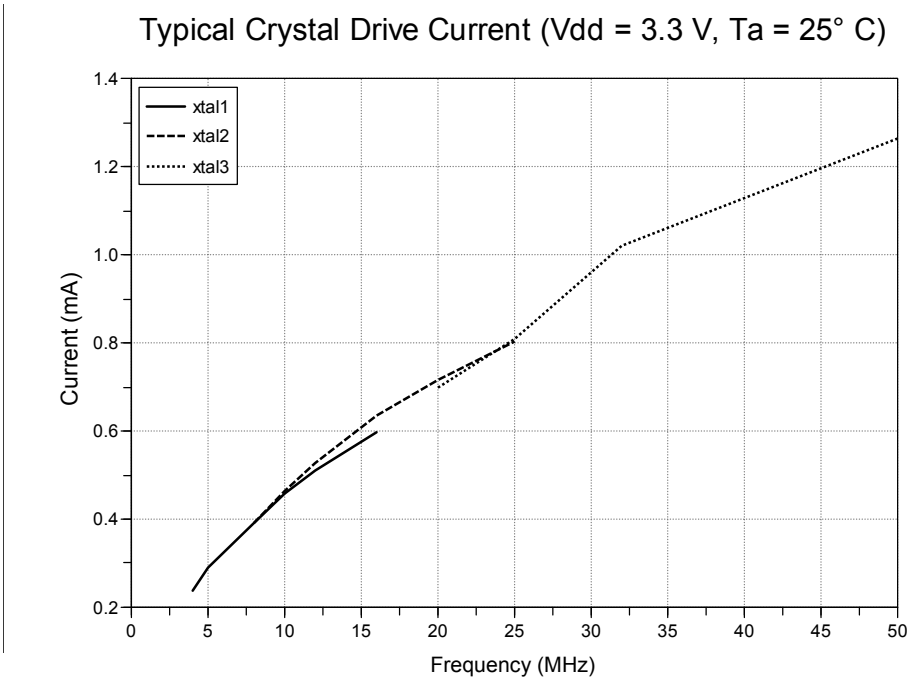
This figure shows the typical current consumption of the Propeller under various operating conditions duplicated across all cogs. Brown out circuitry and the Phase-Locked Loop were disabled for the duration of the test. Current consumption is substantially constant over the operational temperature range.



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

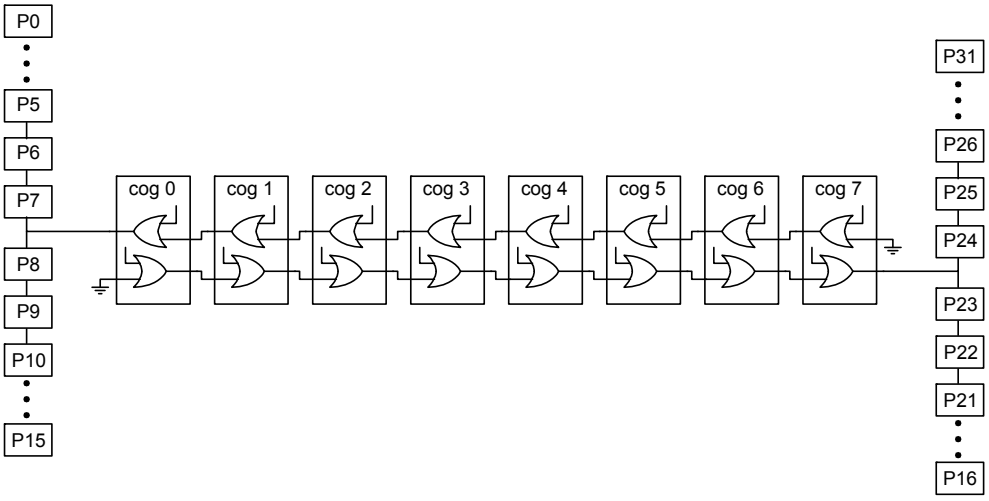
8.4. Typical Crystal Drive Current

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 9

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

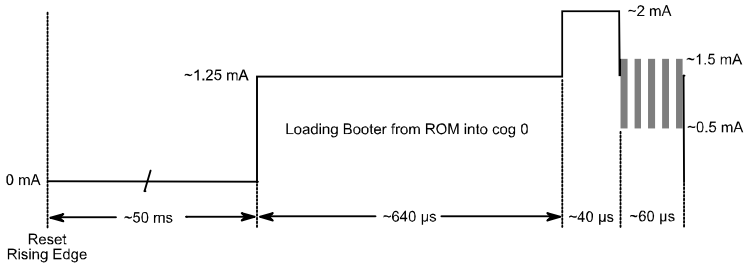


Figure 10

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

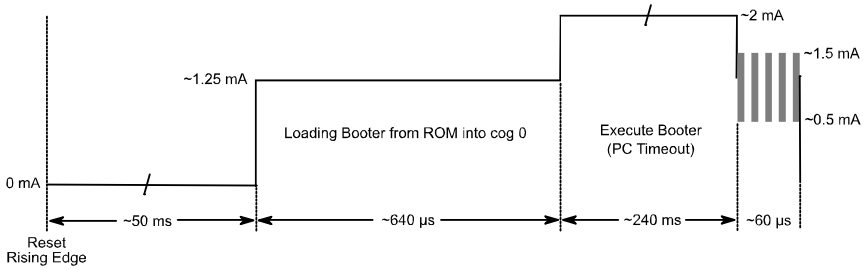


Figure 11

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

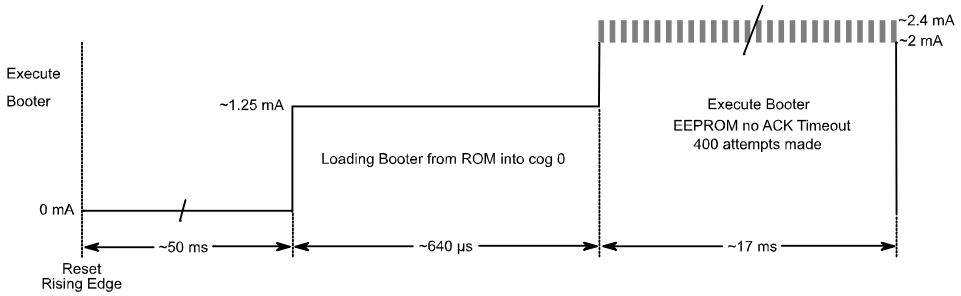


Figure 12

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

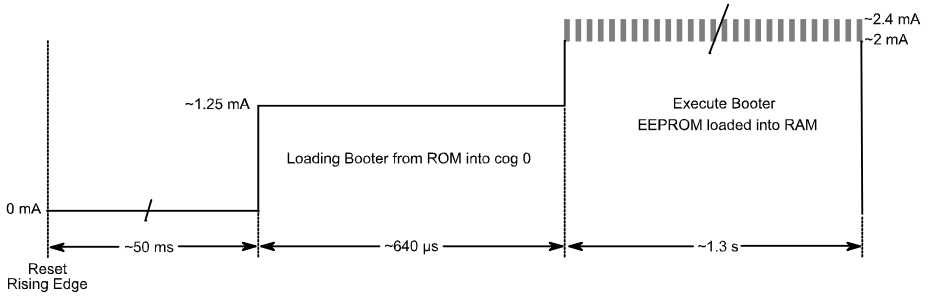
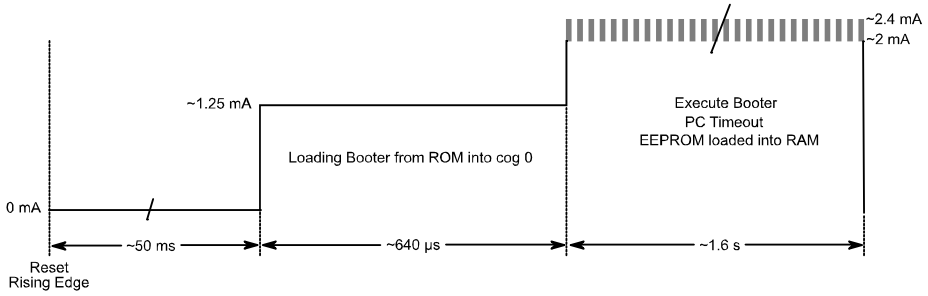


Figure 13

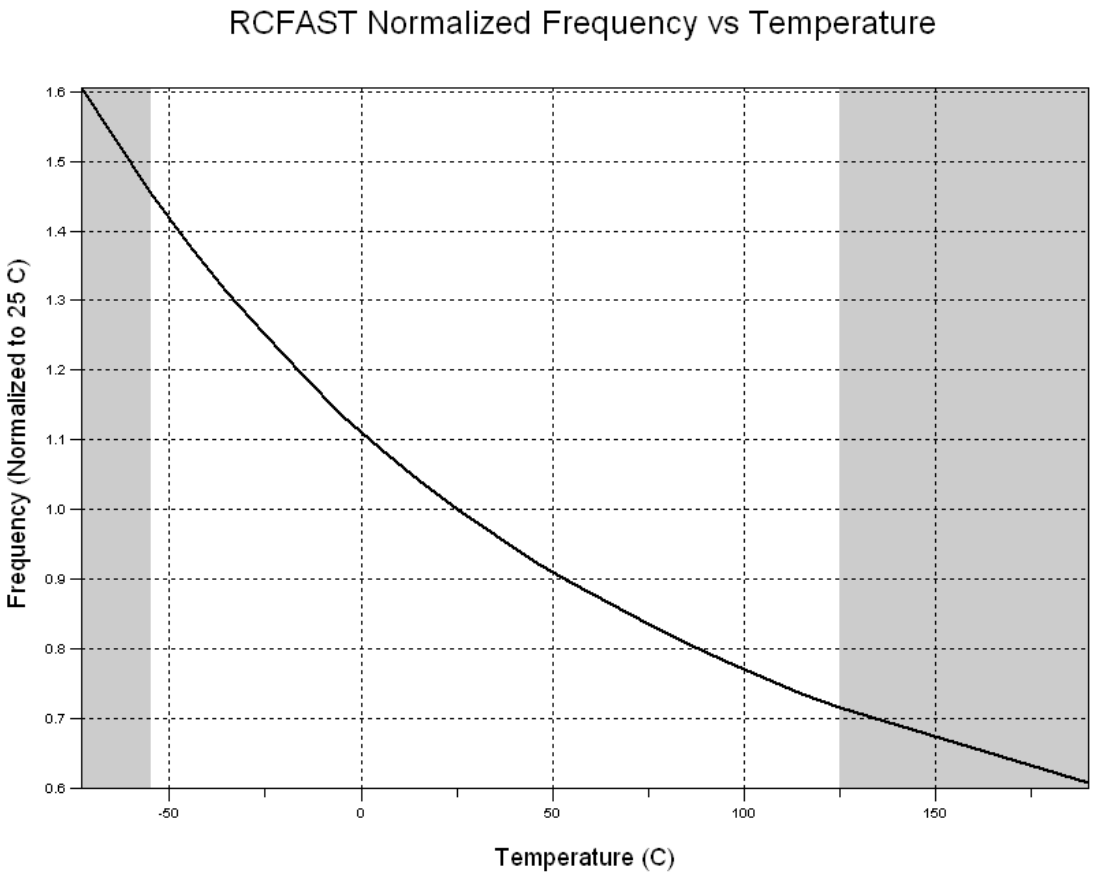
Boot Sequence Current Profile for PC (connected but idle) and EEPROM (P31 held high and P29 connected to EEPROM SDA).



9.0 TEMPERATURE CHARACTERISTICS

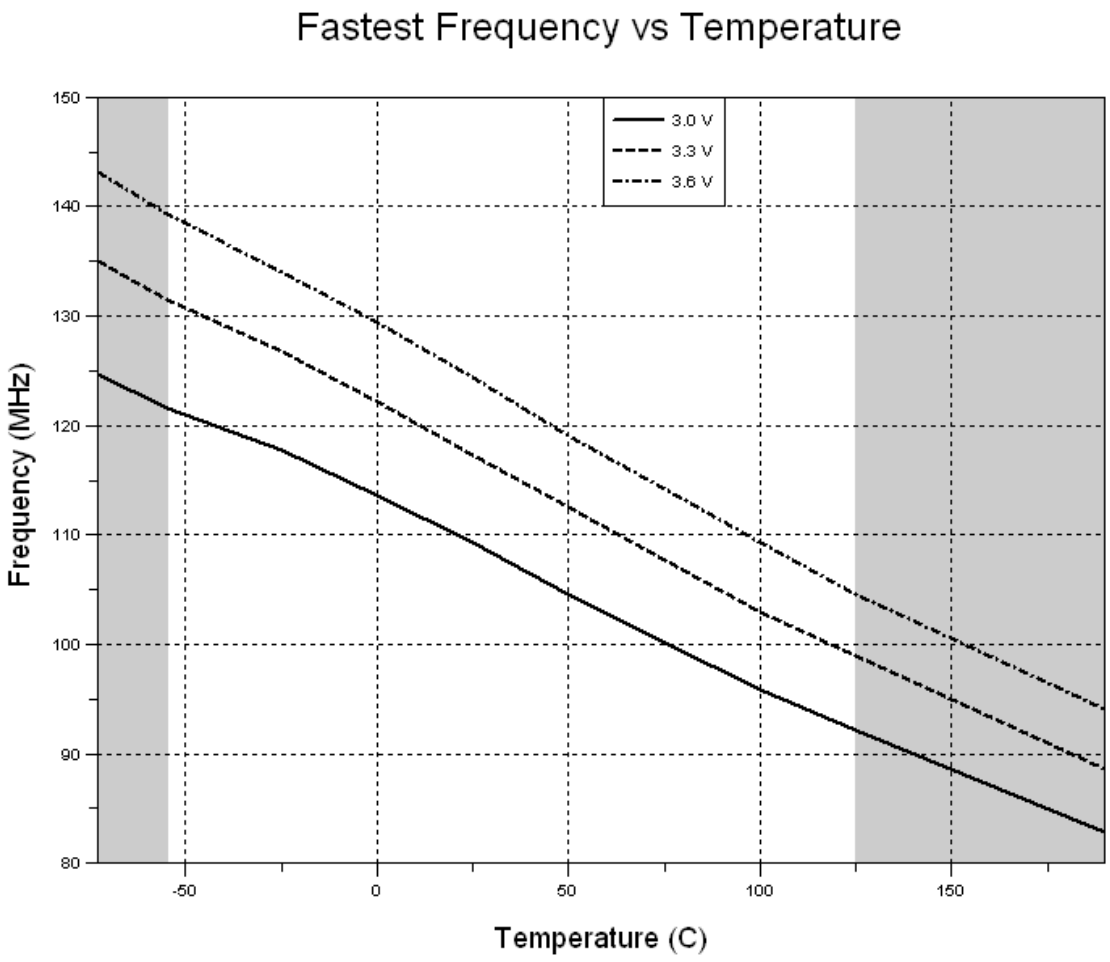
9.1. Internal Oscillator Frequency as a Function of Temperature

While the internal oscillator frequency is variable due to process variation, the rate of change as a function of temperature when normalized provides a chip invariant ratio which can be used to calculate the oscillation frequency when the ambient temperature is other than 25 °C (the temperature to which the graph was normalized). The absolute frequency at 25 °C varied from 13.26 to 13.75 MHz in the sample set. The section of the graph which has a white background is the military range of temperature; the sections in grey represent data which is beyond military temperature specification.



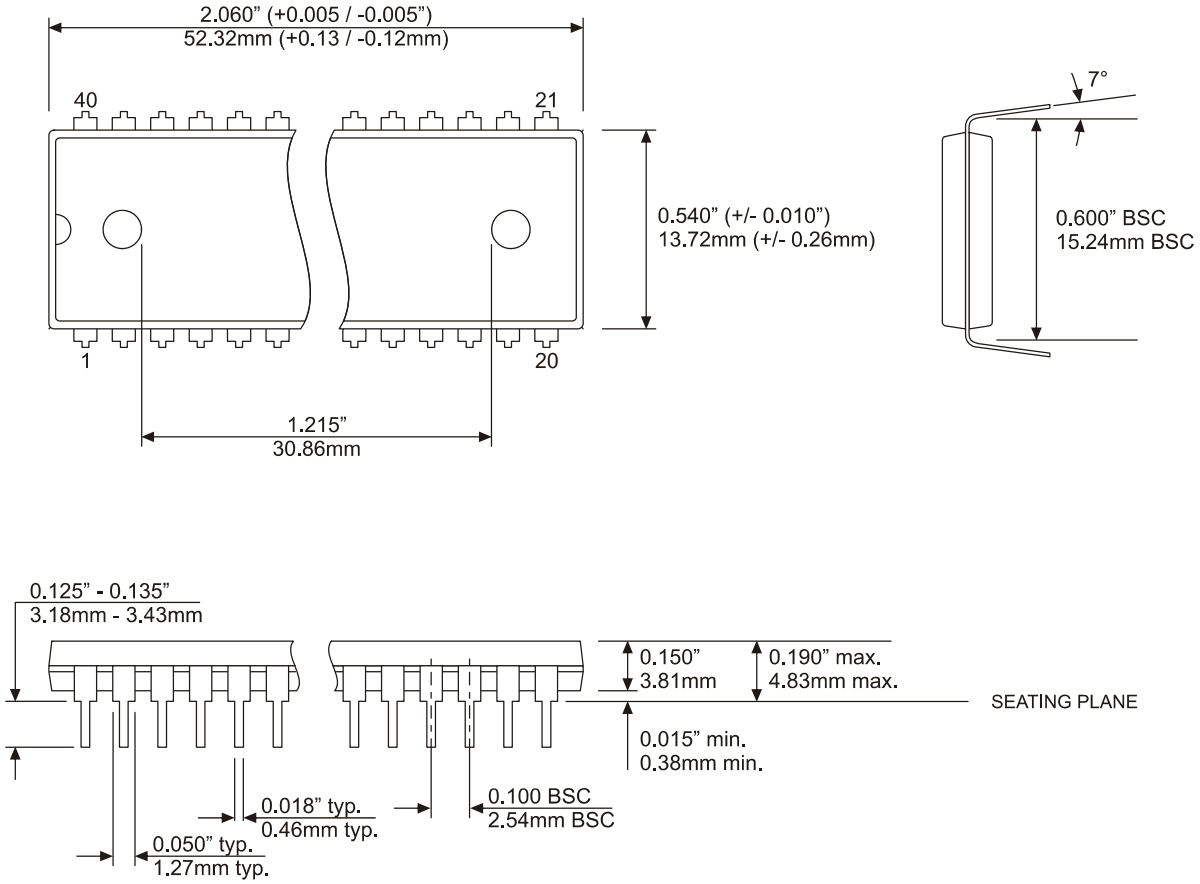
9.2. Fastest Operating Frequency as a Function of Temperature

The following graph represents a small sample average of a Propeller chip's fastest operating range. The test was performed in a forced air chamber using code run on all eight cogs, multiple video generators, and counter modules. A frequency was considered successful if the demo ran without fault for one minute. The curves represent an aggressive testing procedure (averaged, forced air, one minute time limit); therefore the designer must de-rate the curve to arrive at a stable frequency for a particular application. Again the grayed regions represent temperatures beyond the military temperature range.

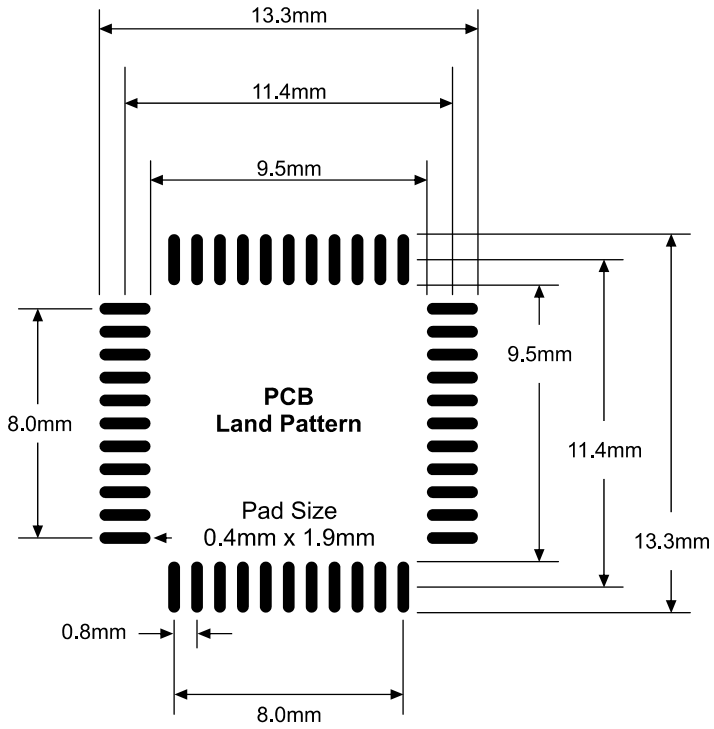


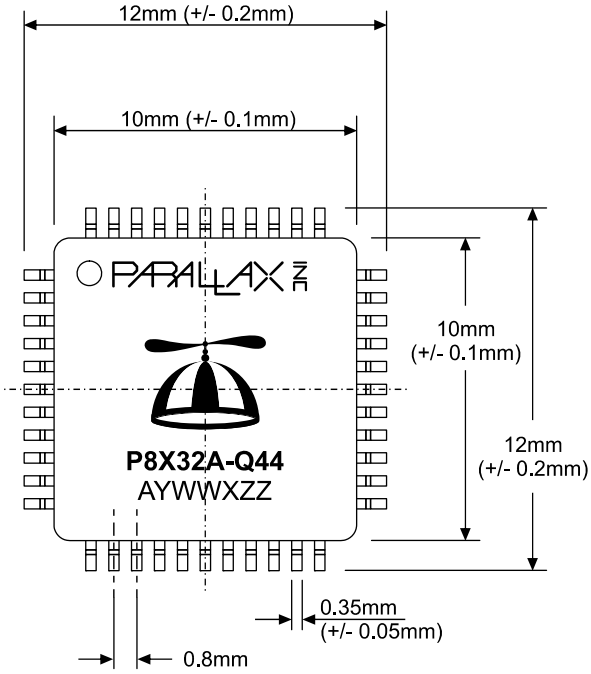
10.0 PACKAGE DIMENSIONS

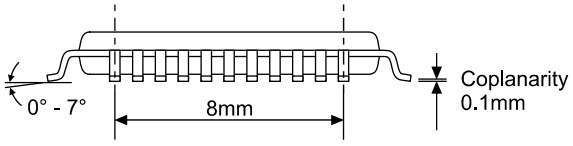
10.1. P8X32A-D40 (40-pin DIP)

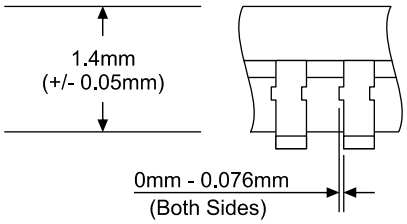


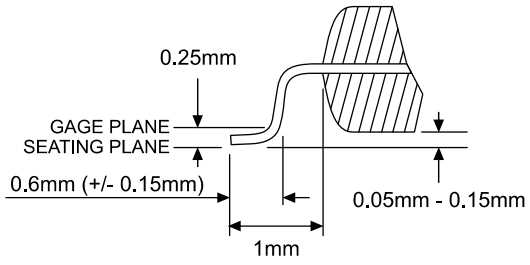
10.2. P8X32A-Q44 (44-pin LQFP)











11.0 MANUFACTURING INFO

11.1. Reflow Peak Temperature

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

11.2. Green/RoHS Compliance

All Parallax Semiconductor Propeller P8X32A chip models are certified Green/RoHS Compliant. RoHS, Green, and ISO certificates are available online at www.parallaxsemiconductor.com.

12.0 REVISION HISTORY

12.1.1. Changes for Version 1.1:

Section 10.3: P8X32A-M44 (44-pin QFN). Image replaced to add stencil pattern diagram. New section inserted: 4.8 Assembly Instruction Execution Stages. Contact Information updated.

12.1.2. Changes for Version 1.2:

Section 6.4: Modified table entries for ADD, ADDABS, ADDS, ADDSX, ADDX, CMP, CMPS, CMPSX, CMPX, COGID, COGINIT, COGSTOP, LOCKCLR, LCOKNEW, LOCKRET, LOCKSET, MAX, MAXS, MIN, MINS, SUB, SUBABS, SUBS, SUBSX, SUBX, SUMC, SUMNC, SUMNZ, SUMZ, TEST, TJNZ, TJZ. Section 4.5 updated. Section 5.1: new sentence added at end of paragraph. Section 5.2: new sentence added at end of first paragraph.

12.1.3. Changes for Version 1.3

Throughout: updated logo and contact information for Parallax Inc., dba Parallax Semiconductor. Section 7.1: footnote added to Table 18: Absolute Maximum Ratings.

12.1.4. Changes for Version 1.4

Section 1.0 changes: 1.3: Key Features and Benefits revised; former sections 1.4 , 1.6 removed. Section 4.4: updated all references to hub timing and replaced both timing diagrams. Section 4.8: reference to hub timing updated. Section 6.4: timing for hub instructions and WAITxxx instructions revised. Former Section 7.0: Propeller Demo Board schematic removed.

Parallax Semiconductor Contact Information

Parallax Semiconductor
599 Menlo Drive
Rocklin, CA 95765
USA

Phone: (916) 632-4664
Fax: (916) 624-8003

sales@parallaxsemiconductor.com
support@parallaxsemiconductor.com
www.parallaxsemiconductor.com
http://obex.parallax.com