



Propeller™ P8X32A Hoja de Datos

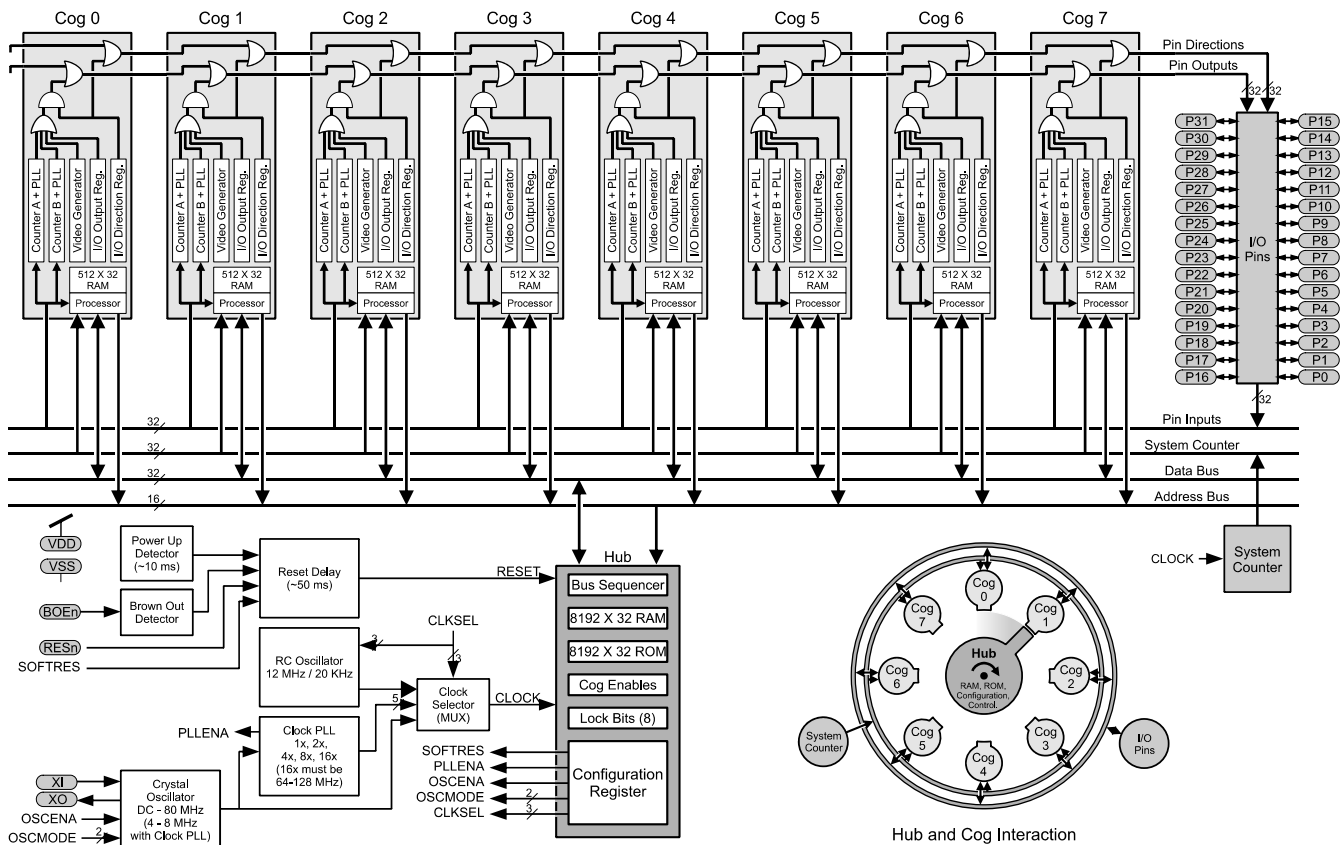
Micro Controlador Multiprocesador de 8-Cogs

1.0 GENERALIDADES DEL PRODUCTO

1.1. Introducción

El chip Propeller se diseñó para proporcionar alta velocidad de procesamiento para sistemas integrados mientras se mantiene un bajo consumo de corriente y una impresión de tarjeta pequeño. Además de ser rápido, el chip Propeller proporciona flexibilidad y potencia a través de sus 8 procesadores llamados cogs, que pueden desarrollar tareas simultaneas independientemente o cooperativamente, todo mientras se mantiene una arquitectura simple y sencilla de aprender y utilizar. Están disponibles dos programas: Spin (lenguaje del alto nivel basado en objetos) y Ensamblador Propeller. Ambos incluyen instrucciones que administran fácilmente el chip Propeller con características únicas.

Figura 1: Propeller P8X32A Diagrama de Bloque



1.2. Códigos de Almacenaje

Tabla 1: Códigos de Almacenaje del Chip Propeller

Mecanismo # de Parte	Tipo de Empaque	Pins E/S	Requerimiento de Potencia	Velocidad Externa de Reloj	Oscilador Interno RC	Velocidad de Ejecución Interna	ROM/RAM Global	RAM del Cog
P8X32A-D40	40-pin DIP	32 CMOS	3.3 volts DC	DC a 80 MHz	12 MHz o 20 kHz*	0 a 160 MIPS (20 MIPS/cog)	64 K bytes; 32768 bytes ROM / 32768 bytes RAM	512 x 32 bits por cog
P8X32A-Q44	44-pin LQFP							
P8X32A-M44	44-pin QFN							

*Aproximado; Puede tener un rango de 8 MHz – 20 MHz, o 13 kHz – 33 kHz, respectivamente.

Tabla de Contenido

1.0	Generalidades del producto.....	1	6.0	LEGUAJES DE programación	22
1.1.	Introducción	1	6.1.	Lista de Palabras reservadas	22
1.2.	Códigos de Almacenaje	1	6.1.1.	Palabras reservadas para Uso Futuro	22
1.3.	Características Clave	3	6.2.	Operadores Matemáticos y Lógicos	23
1.4.	Ventajas de Programación	3	6.3.	Tabla de Lenguaje Spin	24
1.5.	Aplicaciones	3	6.3.1.	Constantes	26
1.6.	Soporte al Programar la Plataforma	3	6.4.	Tabla de Instrucciones Ensamblador Propeller	28
1.7.	Corporativo y Comunidades	4	6.4.1.	Condiciones Ensamblador	31
2.0	DIAGRAMAS DE CONEXION	5	6.4.2.	Directivas Ensamblador	31
2.1.	Asignación De Pins	5	6.4.3.	Efectos Ensamblador	31
2.2.	Descripción de Pins	5	6.4.4.	Operadores Ensamblador	32
2.3.	Diagramas Típicos de Conexión	6	7.0	esquemático DE LA TARJETA DEMO PROPELLER33	
2.3.1.	Conexión Propeller Clip o Conexión Propeller Plug - Recomendada	6	8.0	CARACTERISTICAS ELECTRICAS.....	34
2.3.2.	Conexión alternativa Puerto Serial	6	8.1.	Rangos Máximos Absolutos	34
3.0	Procedimiento de operacion	7	8.2.	Características DC	34
3.1.	Procedimiento de Inicio	7	8.3.	Características AC	34
3.2.	Procedimiento en Tiempo de Uso	7	9.0	CARACTERISTICAS DE CONSUMO DE CORRIENTE	
3.3.	Procedimiento de Apagado	7	35		
4.0	ORGANIZACION DEL SISTEMA	7	9.1.	Consumo Típico de Corriente de 8 Cogs	35
4.1.	Recursos Compartidos	7	9.2.	Corriente Típica de un Cog vs. Frecuencia de Operación	36
4.2.	Reloj del Sistema	7	9.3.	Corriente Típica PLL vs. Frecuencia VCO	36
4.3.	Cogs (procesadores)	8	9.4.	Corriente Típica del Controlador del Cristal	37
4.4.	Hub	8	9.5.	Relación del Cog y Pin E/S	37
4.5.	Pins E/S	9	9.6.	Perfil de Corriente en diversas condiciones de inicio	38
4.6.	Contador del Sistema	10	10.0	CARACTERISTICAS DE TEMPERATURA.....	39
4.7.	Seguros	10	10.1.	Frecuencia de Oscilador Interno como función de la	
4.8.	Etapas de Ejecución en Instrucciones Ensamblador	10	Temperatura	39	
4.9.	Contadores del Cog	12	10.2.	La Frecuencia de Operación mas Rápida como función de la	
4.9.1.	Registros de Control - CTRA / CTRB	12	Temperatura	40	
4.9.2.	Registros de Frecuencia - FRQA / FRQB	13	10.3.	Consumo de Corriente como Función de Temperatura	41
4.9.3.	Registros de fase - PHSA / PHSB	13	11.0	DIMENSIONES DE EMPAQUE	42
4.10.	Generador de Video	14	11.1.	P8X32A-D40 (DIP 40-pin)	42
4.10.1.	Registro de Configuración de Video - VCFG	14	11.2.	P8X32A-Q44 (LQFP 44-pin)	43
4.10.2.	Registro de Escala de Video - VSCL	15	11.3.	P8X32A-M44 (QFN 44-pin)	44
4.10.3.	Instrucción WAITVID	15	12.0	INFORMACION DE FABRICACION.....	45
4.11.	Registro CLK	17	12.1.	Pico de Temperatura de Reflujo	45
5.0	organización de memoria	19	12.2.	Conformidad en Green/RoHS	45
5.1.	Memoria Principal	19	13.0	HISTORIA DE REVISIONES	46
5.1.1.	RAM principal	19	13.1.1.	Cambios a la Versión 1.1:	46
5.1.2.	ROM principal	19	13.1.2.	Cambios a la Versión 1.2:	46
5.1.3.	Definición de Caracteres	19			
5.1.4.	Tablas Matemáticas de Funciones	20			
5.2.	RAM del Cog	21			

1.3. Características Clave

El diseño del chip Propeller libera a los desarrolladores de complejidades en las aplicaciones de sistemas integrados de programación. Por ejemplo:

- Ocho procesadores (cogs) desarrollan simultáneamente procesos independientes o cooperativos compartiendo recursos comunes a través de un hub central. El diseñador de la aplicación tiene control sobre cómo y cuando se emplea cada cog; no hay manejo del compilador o manejo de sistemas operativos que dividan las tareas. Este método le da poder al desarrollador para obtener tiempo determinado absoluto, consumo de potencia y respuesta de la aplicación integrada.
- Los eventos asíncronos son más sencillos de manejar que con equipos que usan interrupciones. El Propeller no necesita interrupciones; solo asigna algunos cogs individuales a tareas para ancho de banda alto y mantiene otros cogs libres y sin una carga fuerte. El resultado es una aplicación más rápida y fácil de mantener.
- Sistema de reloj compartido permite mantener la misma referencia de tiempo, ejecuciones sincronizadas verdaderas.

1.4. Ventajas de Programación

- El lenguaje Spin de alto nivel basado en objetos es fácil de aprender, con instrucciones especiales que permiten a los desarrolladores explotar rápidamente las características únicas y poderosas del chip Propeller.
- Las instrucciones del Ensamblador Propeller proporcionan ejecuciones condicionales y banderas opcionales así como resultados para cada instrucción individual. Esto da a los bloques de código temporizadas mayor consistencia; manejadores de tiempo menos propensos a variaciones de frecuencia y se pierde menos tiempo limpiando ciclos.

1.5. Aplicaciones

El chip Propeller es útil particularmente en proyectos que pueden ser simplificados con procesos simultáneos incluyendo:

- Sistemas de Control Industrial
- Integración de sensores, procesamiento de señales y adquisición de datos.
- Terminales portables de interface humana
- Control de motores
- Interfaces de usuario de salidas NTSC, PAL, o VGA con entradas PS/2 de teclado y ratón.
- Sistemas de video juegos de bajo costo
- Robótica Industrial educacional o de uso personal
- Transmisión de video inalámbrica (NTSC o PAL)

1.6. Soporte al Programar la Plataforma

Parallax Inc. soporta el chip Propeller con una variedad de herramientas y tarjetas:

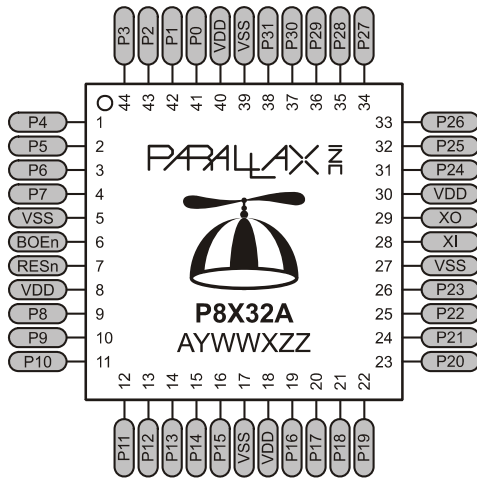
- Propeller Clip (#32200) y Propeller Plug (#32201). Estas tarjetas proporcionan conexiones de Puerto convenientes para programar, ver Diagramas Típicos de en Pág. 6.
- La tarjeta Demo Propeller (#32100) proporciona una forma de probar el Propeller y sus capacidades a través de diseños e interfaces en una tarjeta compacta. Los diagramas se proporcionan en la Pág. 33. Principales características:
 - P8X32A-Q44 Propeller Chip
 - 24LC256-1/ST EEPROM para almacenaje
 - Cristal reemplazable de 5.000 MHz
 - Reguladores 3.3 V y 5 V con interruptor
 - Interface USB-serial para programar y comunicaciones
 - Salida VGA y TV
 - Salida Estéreo con amplificador de 16 Ω
 - Entrada de micrófono
 - Dos conectores PS/2 ratón y teclado
 - 8 LEDs (pins compartidos VGA)
 - Botón para reinicio
 - Postes de tierra para conexión a medidores
 - Pins E/S P0-P7 libres y con salida de conector
 - Tarjeta de prueba para circuitos propios
- La tarjeta Prototipo Propeller (#32212) tiene un chip Propeller de montaje en superficie con los componentes necesarios para programar, con pads listos para una variedad de conectores E/S y chips DIP/SIP, y suficientes perforaciones para área de prototipo
- El PropStick USB (#32210) tiene un chip Propeller, EEPROM, reguladores 3.3 VDC y 5 VDC, botón de reinicio, cristal y conexión USB en un paquete de 0.6" de ancho par uso fácil en prototipos.

1.7. Corporativo y Comunidades

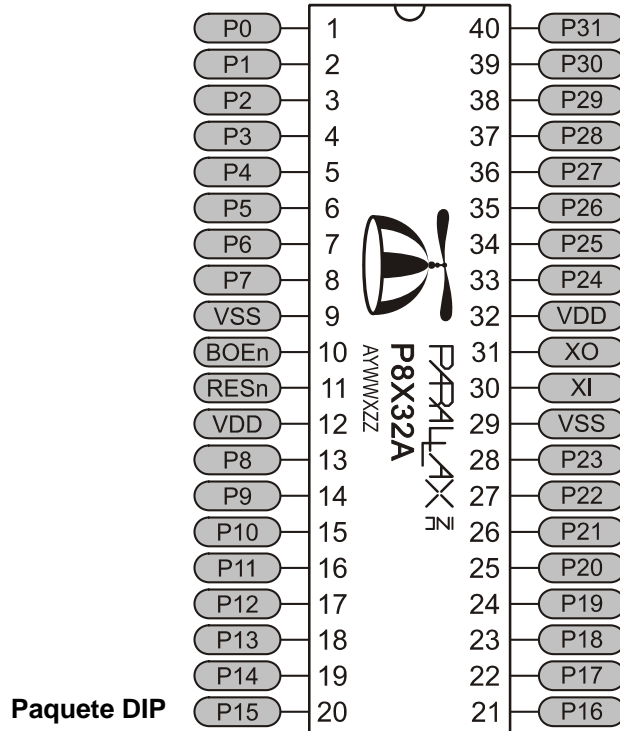
- Parallax proporciona soporte técnico gratuito. En USA llame gratis a (888) 512-1024; fuera de USA llame a (916) 624-8333. O envíe un correo electrónico a: support@parallax.com.
- Parallax tiene foros moderados únicamente para el Propeller: <http://forums.parallax.com/forums>.
- Parallax tiene de igual forma foros en español: <http://espanol.groups.yahoo.com/group/ParallaxenEspanol>.
- Busque a través de las comunidades objetos Propeller que puede compartir con otros usuarios a través de la librería de intercambio de Objetos: <http://obex.parallax.com>.

2.0 DIAGRAMAS DE CONEXION

2.1. Asignación De Pins



Paquete LQFP y QFN



Paquete DIP

2.2. Descripción de Pins

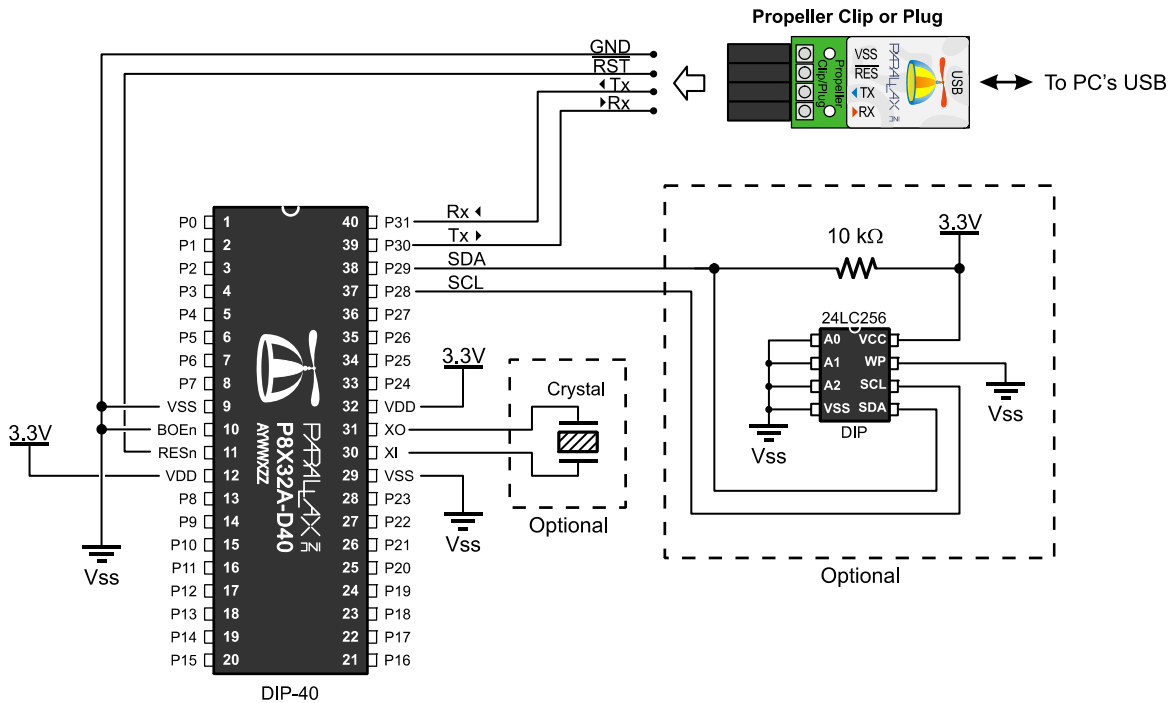
Tabla 2: Descripciones de Pins		
Nombre de Pin	Dirección	Descripción
P0 – P31	E/S	<p>Puerto A de E/S de propósito general. Puede soportar 40mA cada uno a 3.3 VDC. Nivel lógico CMOS con nivel de disparo de $\approx \frac{1}{2} VDD$ o 1.6 VDC @ 3.3 VDC.</p> <p>Los pins mostrados abajo tienen un propósito especial durante el arranque y reinicio pero posterior a esto son de uso general.</p> <ul style="list-style-type: none"> P28 - I2C SCL conexión opcional a EEPROM externa. P29 - I2C SDA conexión opcional a EEPROM externa. P30 - Serial Tx. P31 - Serial Rx.
VDD	---	3.3 volt (2.7 – 3.6 VDC)
VSS	---	Tierra
BOEn	E	Reducción de consumo (activo bajo). Debe estar conectado a VDD o VSS. Si esta en bajo, RESn se convierte en una salida frágil (entregando VDD a través de 5 kΩ) para propósitos de monitoreo pero se puede manejar bajo para ocasionar un reinicio. Si esta en alto, RESn es entrada CMOS con disparo Schmitt.
RESn	E/S	Reinicio (activo bajo). Cuando está en bajo reinicia el chip Propeller: Todos los cogs se deshabilitan y las E/S flotan, el Propeller se reinicia después de 50ms posteriores a la transición RESn de bajo a alto.
XI	E	Cristal de entrada. Puede conectarse a la salida del cristal oscilador (con XO desconectada), o a una punta del cristal (con XO conectada a la otra punta del cristal o resonador) dependiendo de los parámetros del registro CLK. No se necesitan resistencias o capacitares externos.
XO	S	Cristal de Salida. Proporciona retroalimentación para un cristal externo, o puede dejarse desconectado dependiendo de los parámetros de CLK. No se necesitan resistencias o capacitares externos.

2.3. Diagramas Típicos de Conexión

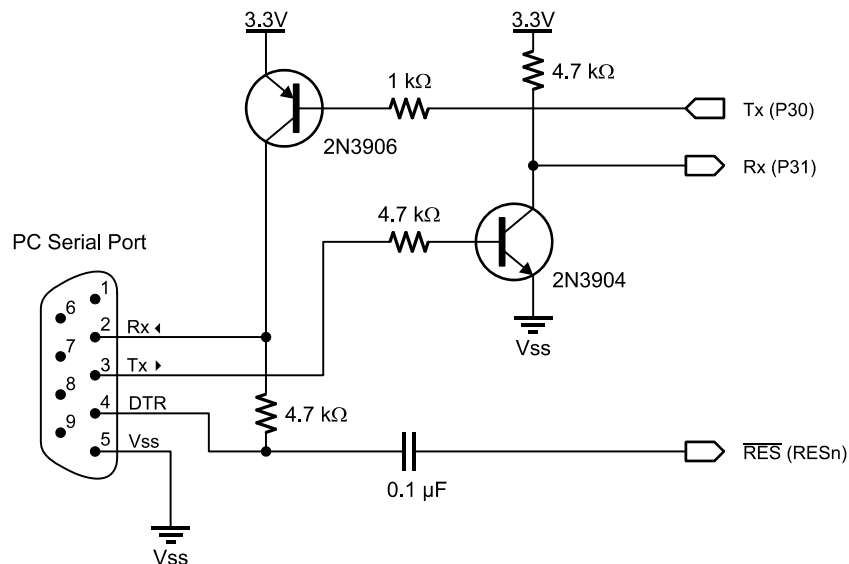
2.3.1. Conexión Propeller Clip o Conexión Propeller Plug - Recomendada

Observe que las conexiones al oscilador externo y a la EEPROM, las cuales están encerradas en líneas punteadas son opcionales.

El chip Propeller #32200; Propeller Plug #32201. El diagrama del Propeller Clip/Plug está disponible para su descarga en www.parallax.com.



2.3.2. Conexión alternativa Puerto Serial



3.0 PROCEDIMIENTO DE OPERACION

3.1. Procedimiento de Inicio

Durante el inicio o reinicio:

1. El oscilador interno RC del chip Propeller comienza a correr a 20 KHz, después de 50 ms reinicia un retraso, cambia a 12 MHz. Después el primer procesador (Cog 0) carga y corre el programa de inicio.
2. El programa de inicio desarrolla una o más de las siguientes tareas, en orden:
 - a. Detecta la comunicación de algún equipo, tales como PC, en pins P30 y P31. Si se detecta comunicación de un equipo, el programa comienza una conversación para identificar el chip Propeller y posiblemente bajar un programa en la RAM global y opcionalmente en la EEPROM externa de 32KB.
 - b. Si no se detecta comunicación con un equipo el programa busca la memoria externa EEPROM de 32 KB en los pins P28 y P29. Si se detecta la EEPROM la imagen entera de los 32 KB se carga en la RAM global del chip Propeller.
 - c. Si no se detecta EEPROM el programa se detiene, el Cog 0 se termina y el chip Propeller se va a modo de apagado con todos los pins de E/S puestos a entradas.
3. Si ninguno de los pasos 2a o 2b tuvo éxito en cargar un programa en la RAM global, y un comando de suspensión no se dio por el equipo, entonces el Cog 0 se recarga con el interprete Spin y el código de usuario se corre desde la RAM global.

3.2. Procedimiento en Tiempo de Uso

Una aplicación Propeller es un programa de usuario compilado en su forma binaria y descargada a la RAM del chip Propeller o a la EEPROM externa. La aplicación consiste en código escrito en lenguaje Spin en el chip propeller (código de alto nivel) con lenguaje ensamblador Propeller opcionalmente (lenguaje de bajo nivel). El código escrito en el lenguaje Spin se interpreta durante el tiempo de ejecución por un Cog usando el interprete Spin mientras el código escrito en Ensamblador en su forma pura corre directamente del cog. Cada aplicación Propeller consiste en al menos un poco de código spin y quizá completamente estará escrito completamente en Spin o varias partes con Ensamblador. El intérprete del chip propeller se inicia en la parte 3 del procedimiento de arranque para poder correr la aplicación.

Una vez que el procedimiento de inicio se completa y la aplicación está corriendo en el Cog 0, el resto de la actividad se define por sí misma. La aplicación tiene completo control sobre detalles como la velocidad del reloj interno, uso de pins E/S, configuración de registros, y cuando qué y cómo se corren los cogs en un determinado tiempo. Todo esto es variable en tiempo de ejecución ya que se controla por la aplicación.

3.3. Procedimiento de Apagado

Cuando el Propeller se va a modo de apagado, el reloj interno se detiene haciendo que todos los cogs se detengan y todos los pins E/S se activen como entradas (alta impedancia). El modo de apagado se dispara por uno de los siguientes eventos:

1. VDD cae debajo del nivel de voltaje de disparo (~2.7 VDC), cuando la característica está habilitada ,
2. el pin RESn se va a modo bajo, o
3. la aplicación requiere un reinicio (ver instrucción **REBOOT** en el manual Propeller).

El modo de apagado se termina cuando el voltaje alcanza nuevamente el nivel de disparo y el pin RESn se pone en alto.

4.0 ORGANIZACION DEL SISTEMA

4.1. Recursos Compartidos

Hay dos tipos de recursos compartidos en el propeller: 1) común y 2) mutuamente exclusivo. Los recursos comunes se pueden acceder en cualquier momento. Los mutuamente exclusivos también se pueden acceder por cualquier cog, pero solo un cog a la vez. Los recursos comunes son los pins E/S y el contador del sistema. Todos los otros recursos compartidos son mutuamente exclusivos por naturaleza y su acceso está controlado por el hub. Ver la sección Hub 4.4 en la Pág. 8.

4.2. Reloj del Sistema

El reloj del sistema (mostrado como "CLOCK" en la Figura 1, Pág. 1) es la fuente de reloj central para cada componente del Propeller. La señal del reloj del sistema viene de tres posibles fuentes:

- El oscilador interno RC (~12 MHz o ~20 kHz)
- El pin X1 de entrada (funcionando como entrada de alta impedancia o como cristal oscilador en conjunto con el pin X0)

- El reloj PLL (phase-locked loop) alimentado por la entrada X1

La fuente se determina por los parámetros del registro CLK, los cuales se seleccionan en tiempo de compilación y son re seleccionables en tiempo de ejecución. El hub y el bus interno operan a la mitad de la velocidad del reloj del sistema.

4.3. Cogs (procesadores)

El Propeller contiene ocho (8) procesadores idénticos e independientes llamados cogs, numerados del 0 al 7. Cada Cog contiene un bloque procesador, 2KB de RAM configurada como 512 longs (512 x 32 bits), dos módulos contadores avanzados con PLL, un generador de video, Registro de E/S, Registro de dirección E/S y otros registros no mostrados en el diagrama de bloque.

Los ocho Cogs se manejan desde el reloj del sistema; cada uno mantiene la misma referencia de tiempo y todos los cogs activos ejecutan instrucciones simultáneamente. Tienen acceso a los mismos recursos compartidos.

Los Cogs pueden iniciar y detenerse en tiempo de ejecución o pueden programarse para desarrollar tareas simultáneas, independientemente o en coordinación con otros cogs a través de la RAM principal. Cada Cog tiene su propia RAM que contiene 512 registros de 32 bits cada uno. La RAM del cog es para propósito general excepto los últimos 16 registros, que son registros de propósito especial, como se describe en la **Error! Reference source not found.** en Pág. **Error! Bookmark not defined.**

4.4. Hub

Para mantener la integridad del sistema, los recursos mutuamente exclusivos no deben accederse por más de un cog a la vez. El hub controla el acceso a los recursos dando a cada cog entrada por medio del “round robin” del cog 0 al cog 7. El Hub y su bus corren a la mitad de velocidad del reloj del sistema, dando acceso al cog a los recursos mutuamente exclusivos una vez cada 16 ciclos de reloj. Las instrucciones del Hub y las instrucciones ensamblador que accesan los recursos mutuamente exclusivos requieren de 7 ciclos para ejecutarse, pero al principio necesitan sincronizarse para iniciar en la ventana de acceso del Hub .

Toma hasta 15 ciclos sincronizar el hub mas los 7 ciclos de ejecución, por lo tanto las instrucciones de hub toman del 7 a 22 ciclos para completarse.

La Figura 2 y Figura 3 muestran ejemplos de donde el cog 0 tiene una instrucción para ejecutar. La Figura 2 muestra el mejor escenario; la instrucción esta lista justo al inicio de la ventana de acceso. La instrucción se realiza inmediatamente (7 ciclos) y deja 9 ciclos adicionales para otras instrucciones antes de que llegue la próxima ventana de acceso.

La Figura 3 muestra el peor caso; donde la instrucción esta lista justo después de que paso la ventana de acceso. El cog espera hasta la siguiente ventana de acceso (15 ciclos) y luego ejecuta la instrucción (7 ciclos) para un total del 22 ciclos para esa instrucción. Nuevamente hay 9 ciclos adicionales después de ejecutar la instrucción para poder ejecutar otras instrucciones antes de que llegue la siguiente ventana de acceso. Para tener mayor eficiencia de las rutinas ensamblador Propeller que tienen que acceder frecuentemente a los recursos mutuamente exclusivos, puede ser benéfico intercambiar instrucciones de hub con instrucciones de no-hub para reducir el número de ciclos esperando por el siguiente acceso. Como la mayoría de las instrucciones de ensamblador Propeller toma 4 ciclos, dos de estas instrucciones pueden ejecutarse entre instrucciones seguidas de Hub.

Tome en cuenta que una instrucción particular de hub no interfiera, de ninguna forma, con otras instrucciones de cog debido al mecanismo del hub. El Cog 1 por ejemplo puede iniciar una instrucción de hub durante el ciclo 2 del sistema de reloj, en ambos ejemplos, posiblemente se encimen las ejecuciones con el Cog 0 sin tener ningún efecto dañino. Mientras tanto todos los demás cogs continúan ejecutando instrucciones no-hub, o esperando para su acceso individual sin importar lo que los otros estén haciendo.

Figura 2: Interacción Cog-Hub – Mejor escenario

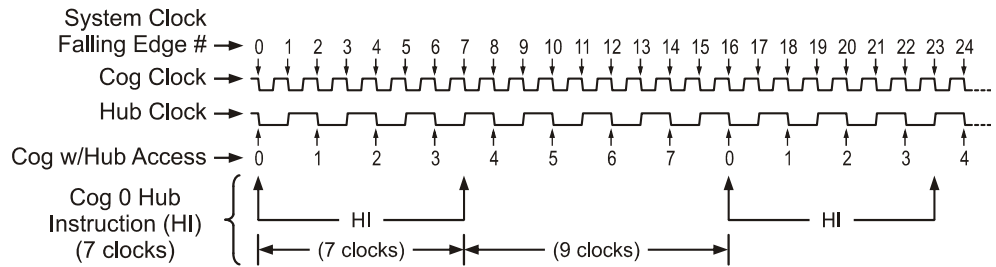
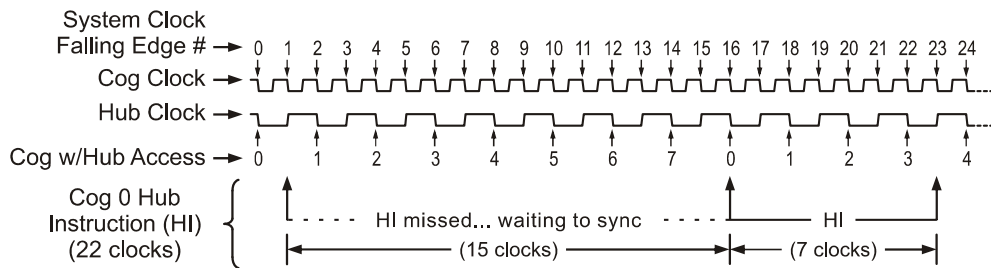


Figura 3: Interacción Cog-Hub – Peor Escenario



4.5. Pines E/S

El Propeller tiene 32 pines E/S, 28 de los cuales son de propósito general. Los pines 28-31 son de propósito especial al inicio, después están disponibles como propósito general; ver sección 2.2, Pág. 5. Después del arranque cualquier pin E/S puede usarse por cualquier cog en cualquier momento. El desarrollador debe asegurarse que no hay dos cogs tratando de usar el mismo pin E/S para diferentes propósitos al mismo tiempo.

Vea la Figura 1, Pág. 1. Cada cog tiene sus propios Registro de Dirección de 32-bit y su Registro de Dirección de Salida 32-bit para influir en el estado del pin correspondiente. Una dirección E/S deseada del cog y su estado de salida se comunica a través del cog para convertirse en “Dirección de Pin” y “Salida de Pin”.

Las direcciones de pin son el resultado de la OR de los registros de dirección de los cogs juntos. Una salida de cog consiste en los bits de sus módulos E/S (los contadores, el generador de video y el registro de salida E/S) se hacen OR juntos y luego AND con los bits de la dirección del registro.

Todos los cogs pueden acceder e influenciar los pines E/S simultáneamente sin ninguna contención eléctrica, como se describe por estas reglas:

- A. Un pin es una entrada solo si un cog activo no lo activo como salida.
- B. Una salida de pin es baja solo si todos los cogs activos que la activaron como salida la activaron también como baja.
- C. Una salida de pin es alta solo si todos los cogs activos que la activaron como salida la activaron también como alta.

La Tabla 3 demuestra algunas combinaciones de la influencia colectiva de los cogs en un pin particular de E/S, P12 en este ejemplo. Para simplificar, estos ejemplos asumen que el bit 12 de cada hardware E/S del cog es puesto a cero.

Cualquier cog que está apagado tiene su Registro de Dirección y estados de salida puestas a cero, efectivamente removiéndolos de la influencia del estado final de los pines E/S que los cogs activos están controlando.

Cada cog tiene también su propio Registro de Entrada de 32-bit. Este registro de entrada es realmente un pseudo registro; cada vez que se lee, el estado actual de los pines E/S se leen, sin importar si son entradas o salidas.

Tabla 3: Ejemplos de E/S Compartidas

	Bit 12 de Registro de Dirección E/S del Cog	Bit 12 de Registro de Salida E/S del Cog	Estado del Pin E/S Pin P12	Regla
Cog ID	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7		
Ejemplo 1	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	Entrada	A
Ejemplo 2	1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	Salida baja	B
Ejemplo 3	1 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0	Salida Alta	C
Ejemplo 4	1 0 0 0 0 0 0 0	0 1 0 0 0 0 0 0	Salida baja	B
Ejemplo 5	1 1 0 0 0 0 0 0	0 1 0 0 0 0 0 0	Salida Alta	C
Ejemplo 6	1 1 1 1 1 1 1 1	0 1 0 1 0 0 0 0	Salida Alta	C
Ejemplo 7	1 1 1 1 1 1 1 1	0 0 0 1 0 0 0 0	Salida Alta	C
Ejemplo 8	1 1 1 0 1 1 1 1	0 0 0 1 0 0 0 0	Salida baja	B

Nota: Para el Registro de Dirección E/S, un 1 en una localidad activa el pin correspondiente E/S como salida; un 0 lo activa como entrada.

4.6. Contador del Sistema

El contador del sistema es global, solo lectura, contador de 32-bit que incrementa una vez cada ciclo del sistema de reloj. Los cogs pueden leer el contador del sistema (a través de sus registros CNT, ver **Error! Reference source not found.** en Pág. **Error! Bookmark not defined.**) para desarrollar cálculos de tiempo y pueden usar la instrucción `WAITCNT` (ver sección 6.3 en Pág. 24 y sección 6.4 en Pág. 28) para crear retrasos efectivos entre procesos. El contador del sistema es un recurso común el cual cada cog puede leer simultáneamente. El contador del sistema no se limpia al inicio ya que su uso práctico es para diferenciales de tiempo. Si un cog necesita rastrear un tiempo específico solo necesita leer y guardar el valor inicial del contador y compararlo en subsecuentes momentos con ese valor.

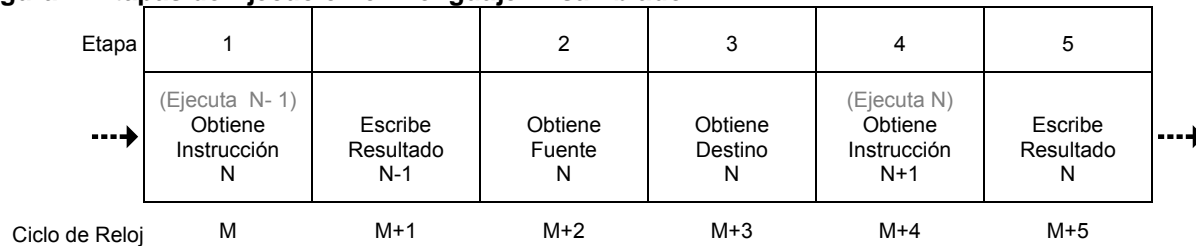
4.7. Seguros

Hay ocho bits de seguro (semáforos) disponibles para facilitar el acceso exclusivo a recursos definidos por usuario entre los cogs. Si un bloque de memoria se usara por dos o más cogs al mismo tiempo y ese bloque consiste en más de un long (cuatro bytes), el cog realizaría múltiples lecturas y escrituras para recolectar o actualizar ese bloque de memoria. Esto lleva a la posibilidad de tener contener lecto/escritura en ese bloque de memoria donde un cog puede estar escribiendo mientras el otro está leyendo, resultando en lecturas o escrituras falsas o perdidas.

Los seguros son bits globales que se accesan a través del hub con `LOCKNEW`, `LOCKRET`, `LOCKSET`, y `LOCKCLR`. Debido a que los seguros se accesan solo por medio del hub, solo un cog puede afectarlo, esto hace un efectivo mecanismo de control. El hub mantiene un inventario de que seguros están en uso y cuál es su estado, los cogs pueden usar, regresar, activar y limpiar según se necesite en tiempo de ejecución.

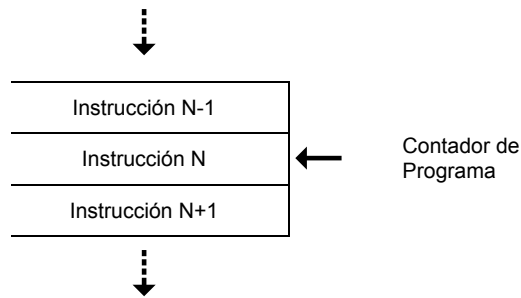
4.8. Etapas de Ejecución en Instrucciones Ensamblador

Figura 4: Etapas de Ejecución en Lenguaje Ensamblador



El Propeller ejecuta instrucciones ensamblador es cinco etapas. Mientras una instrucción entera toma seis ciclos para ejecutarse, dos de esos ciclos de reloj son dedicados a las dos instrucciones adyacentes. Esto da como resultado una optimización de las salidas en los cuatro ciclos de reloj por instrucción.

Figura 5 Memoria Cog



En la etapa 1 la instrucción N apunta al contador de programa, se obtiene de la memoria del cog durante el ciclo de reloj M. En el ciclo M+1 el resultado anterior se escribe en memoria. La razón por la que la instrucción previa se escribe después de la instrucción actual se explicara brevemente.

Durante la etapa 2, si una bandera inmediata de la instrucción N se activa, el bit 9 del campo fuente se guarda como valor fuente. Si el valor no es inmediato, la localidad especificada por el campo fuente se obtiene de la memoria del cog durante el ciclo M+2. En el ciclo M+3 la localidad especificada en el campo destino se obtiene de la memoria del cog (Etapa 3).

A este momento (Etapa 4) la unidad lógica aritmética (ALU) tiene toda la información necesaria para ejecutar la instrucción. Ejecutar la instrucción toma algo de tiempo antes de tener el resultado disponible. El monto de tiempo requerido para la ejecución se dicta por la operación más lenta que desarrolla la ALU. Para proporcionar suficiente tiempo a la ALU para ejecutar la instrucción, un ciclo completo de reloj (M+4) se le da a la ALU para dejar el resultado en su estado final. Durante esta ejecución, la memoria del cog no es accesa por la instrucción N. Para acelerar la ejecución del programa, la siguiente instrucción se obtiene de la memoria del cog mientras la instrucción actual se ejecuta en la ALU.

Finalmente en el ciclo M+5 el resultado de la instrucción N se escribe de regreso en la memoria, completando la etapa 5.

Los intervalos parciales de instrucciones tienen un par de implicaciones en el flujo del programa. Primero, cuando se modifica el código a través de **MOVI**, **MOVS**, **MOVD** o cualquier operación que modifique una instrucción ensamblador, debe existir al menos una instrucción ejecutada antes de que la instrucción modificada se ejecute. Si la modificación se hace en la instrucción inmediata (N+1) la versión no modificada N+1 se cargara un ciclo de reloj antes de que la versión modificada de la instrucción N+1 se escriba en la memoria del cog.

Segundo, para los saltos condicionales, solo se sabe con certeza si saltaran hasta el final del ciclo M+4. Como la siguiente instrucción ya ha sido obtenida solo una de dos posibles subdivisiones pueden predecirlo. En el Propeller, las subdivisiones condicionales se predicen siempre para tomar el salto. Por ciclos que usan **DJNZ** donde el salto se cada vez excepto al final del ciclo, se logra un tiempo de ejecución más cerrado.

En caso de que el salto no se haga, el cog no toma acciones hasta que se obtiene la siguiente instrucción. Esto es equivalente a un **NOP** insertado antes de la siguiente instrucción. Saltos incondicionales siempre toman 4 ciclos de reloj para ejecutarse ya que el Propeller puede predecir que dirección se necesita cargar en el contador del programa para que se ejecute la siguiente instrucción. Los ejemplos de saltos incondicionales incluyen **JMP**, **JMPRET**, **CALL** y **RET**.

Si una instrucción necesita acceder recursos del Hub, la etapa 4 se extiende hasta que el hub está disponible, incrementando el tiempo de ejecución por hasta al menos 7 y potencialmente 22 ciclos de reloj. Ver Sección 4.4: Hub en Pág. 8.

4.9. Contadores del Cog

Cada cog tiene dos módulos contadores: **CTRA** y **CTRB**. Cada modulo contador puede controlar o monitorear hasta dos pins E/S y desarrollar acumulaciones condicionales de 32-bit de sus registros FRQ en su registro PHS en cada ciclo de reloj.

Cada modulo contador también tiene su propia ciclo de fase cerrada (PLL) que puede usarse para sintetizar frecuencias hasta de 128 MHz.

Con un poco de programación o supervisión del cog el contador puede usarse para:

- Síntesis de frecuencias
- Medición de frecuencias
- Contadores de pulsos
- Mediciones de pulso
- Medición de estados de multi pins
- Modulación de ancho de pulso
- Medición de ciclos de tareas
- Conversión análoga digital
- Conversión digital análoga

Para algunas de estas operaciones, el cog puede activarse y dejarse en modo libre. Para otras quizá se use **WAITCNT** para alinear en tiempo las lecturas del contador y escribirlas en un ciclo, creando el efecto de una maquina más compleja.

Observe que para la frecuencia de cog de 80 MHz, el periodo de actualización del contador es 12.5 ns. Esta alta velocidad, combinada con la precisión de 32-bit permite generación y medición de cada señal dinámica.

EL objetivo de diseño para el contador fue crear un subsistema simple y flexible el cual podría realizar algunas tareas repetitivas en cada ciclo de reloj, por lo tanto liberaba al cog permitiéndole realizar algunas súper tareas computacionales más ricas. Como los contadores tienen solo 32 modos básicos de operación, no hay límites de cómo pueden usarse dinámicamente con el software. Integrando a este concepto el uso de instrucciones como **WAITPEQ**, **WAITPNE**, y **WAITCNT** se puede alinear eventos o alinear en tiempo un cog y sus contadores.

Cada contador tiene tres registros:

4.9.1. Registros de Control - CTRA / CTRB

El registro (CTRA y CTRB) selecciona el modo de operación. Tan pronto como se escribe el registro, el nuevo modo de operación tiene efecto. Escribir un cero a CTR deshabilitara inmediatamente el contador, deteniendo todos los pins de salida y la acumulación PHS.

Tabla 4: Registros CTRA y CTRB						
31	30..26	25..23	22..15	14..9	8..6	5..0
-	CTRMODE	PLLDIV	-	BPIN	-	APIN

El campo CTRMODE selecciona uno de 32 modos de operación para el contador, escrito convenientemente (junto con PLLDIV) usando la instrucción **MOVI**. Estos modos de operación se enlistan en la Tabla 6 en Pág. 13.

Tabla 5: Campo PLLDIV								
PLLDIV	%000	%001	%010	%011	%100	%101	%110	%111
Salida	$\frac{VCO}{128}$	$\frac{VCO}{64}$	$\frac{VCO}{32}$	$\frac{VCO}{16}$	$\frac{VCO}{8}$	$\frac{VCO}{4}$	$\frac{VCO}{2}$	$\frac{VCO}{1}$

PLLDIV selecciona una salida PLL y se puede ignorar si no se usa.

Los modos PLL (%00001 a %00011) hacen que ocurra una acumulación FRQ-a-PHS en cada ciclo de reloj. Esto hace un oscilador numéricamente controlado (NCO) en PHS [31], el cual alimenta la entrada de referencia PLL del contador. El PLL multiplicara esta frecuencia por 16 usando su oscilador de voltaje controlado (VCO). Para operaciones estables se recomienda que la frecuencia VCO se mantenga en 64MHz a 128 MHz. Esto es una frecuencia NCO de 4 MHz a 8 MHz.

El campo PLLDIV del registro CTR selecciona cual división de potencia de dos de la frecuencia VCO se usara como salida final PLL. Esto permite un rango de PLL de 500 kHz a 128 MHz.

BPIN selecciona un pin como salida secundaria E/S. Puede ignorarse si no se usa y puede escribirse con la instrucción **MOVD**.

APIN selecciona el pin primario E/S. Puede ignorarse si no se usa y puede escribirse usando la instrucción **MOVS**.

4.9.2. Registros de Frecuencia - FRQA / FRQB

FRQ (FRQA y FRQB) tiene el valor que será acumulado en el registro PHS. Para algunas aplicaciones, FRQ puede escribirse una vez, y luego ignorarse. Para otras puede modularse rápidamente.

4.9.3. Registros de fase - PHSA / PHSB

El registro PHS (PHSA y PHSB) puede escribirse y leerse con instrucciones cog, pero también trabaja como acumulador libre, agregando el registro FRQ a si mismo potencialmente en cada ciclo de reloj. Cualquier instrucción escrita a PHS sobrescribe cualquier acumulación del ciclo de reloj. PHS solo puede leerse a través de operandos fuente (igual a PAR, CNT, INA, e INB). Cuidate que al hacer una lectura-modificación-escritura en PHS como "ADD PHSA, #1", hace que el ultimo valor escrito se use como operando de entrada en vez de una actual acumulación.

Tabla 6: Modos Contadores (Valores de Campo CTRMODE)

CTRMODE	Descripción	Acumulado FRQx a PHSx	APIN Salida*	BPIN Salida*
%00000	Contador deshabilitado (apagado)	0 (nunca)	0 (ninguno)	0 (ninguno)
%00001	PLL interno (modo video)	1 (siempre)	0	0
%00010	PLL terminación sencilla	1	PLLx	0
%00011	PLL diferencial	1	PLLx	!PLLx
%00100	NCO terminación sencilla	1	PHSx[31]	0
%00101	NCO diferencial	1	PHSx[31]	!PHSx[31]
%00110	DUTY terminación sencilla	1	PHSx-Carry	0
%00111	DUTY diferencial	1	PHSx-Carry	!PHSx-Carry
%01000	POS detector	A ¹	0	0
%01001	POS detector con retroalimentación	A ¹	0	!A1
%01010	POSEDGE detector	A ¹ & !A ²	0	0
%01011	POSEDGE detector con retroalimentación	A ¹ & !A ²	0	!A1
%01100	NEG detector	!A ¹	0	0
%01101	NEG detector con retroalimentación	!A ¹	0	!A1
%01110	NEGEDGE detector	!A ¹ & A ²	0	0
%01111	NEGEDGE detector con retroalimentación	!A ¹ & A ²	0	!A1
%10000	LOGICO nunca	0	0	0
%10001	LOGICO !A & !B	!A ¹ & !B ¹	0	0
%10010	LOGICO A & !B	A ¹ & !B ¹	0	0
%10011	LOGICO !B	!B ¹	0	0
%10100	LOGICO !A & B	!A ¹ & B ¹	0	0
%10101	LOGICO !A	!A ¹	0	0
%10110	LOGICO A <> B	A ¹ <> B ¹	0	0
%10111	LOGICO !A !B	!A ¹ !B ¹	0	0
%11000	LOGICO A & B	A ¹ & B ¹	0	0
%11001	LOGICO A == B	A ¹ == B ¹	0	0
%11010	LOGICO A	A ¹	0	0
%11011	LOGICO A !B	A ¹ !B ¹	0	0
%11100	LOGICO B	B ¹	0	0
%11101	LOGICO !A B	!A ¹ B ¹	0	0
%11110	LOGICO A B	A ¹ B ¹	0	0
%11111	LOGICO siempre	1	0	0

*Activar DIR para afectar el pin correspondiente. A1=APIN entrada retrasada 1 ciclo. A2=APIN entrada retrasada 2 ciclos. B1=BPIN entrada retrasada 1 ciclo.

4.10. Generador de Video

Cada cog tiene un generador de video que facilita la transmisión de datos de imágenes de video a un rango constante. Hay dos registros y una instrucción que proporciona control y acceso al generador de video. El contador A del cog debe estar corriendo en modo PLL y se usa para generar la señal de tiempo para el generador de video. El registro de la escala de video especifica el número de ciclos de reloj de PLL Contador A (PLLA) para cada pixel y el número de ciclos de reloj antes de obtener otro cuadro de datos proporcionado por la instrucción **WAITVID** la cual se ejecuta en el cog. El registro de configuración de video establece el modo en el que el generador de video debe operar, y puede generar Video Compuesto o VGA (NTSC o PAL).

El generador de video debe inicializarse primero iniciando el contador A, Activando el Registro de Escala de Video, Activando el Registro de Configuración de Video y finalmente proporcionando datos con la instrucción **WAITVID**. Si falla la inicialización del generador de video el cog quedara colgado indefinidamente cuando se ejecute la instrucción **WAITVID**.

4.10.1. Registro de Configuración de Video - VCFG

El registro de configuración de video contiene la configuración del generador de video y se muestra en la Tabla 7.

En ensamblador Propeller los campos VMode a AuralSub pueden escribirse convenientemente usando la instrucción **MOVI**, el campo VGroup puede escribirse con la instrucción **MOVD**, y el campo VPins puede escribirse con la instrucción **MOVS**.

Tabla 7: Registro VCFG									
31	30..29	28	27	26	25..23	22..12	11..9	8	7..0
-	VMode	CMode	Chroma1	Chroma0	AuralSub	-	VGroup	-	VPins

El campo de 2-bit VMode (Modo de video) selecciona el tipo y orientación de la salida de video de acuerdo a la Tabla 8.

Tabla 8: El Campo Modo de Video	
VMode	Modo de Video
00	Deshabilitado, no se genera video
01	Modo VGA; 8-bit salida paralela en VPins 7:0
10	Modo Compuesto 1; transmite en VPins 7:4, banda base en VPins 3:0
11	Modo Compuesto 2; banda base en VPins 7:4, transmite en VPins 3:0

El campo CMode (Modo de Color) selecciona dos de cuatro modos de color. 0 = dos colores; datos de pixel 32-bits por 1 bit y solo se usa color 0 o 1. 1 = cuatro colores; datos de pixel 16 bits por 2 bits, se usan colores 0 al 3.

El bit Chroma1 (transmisión color) habilita o deshabilita chroma en señal de transmisión. 0 = deshabilitado, 1 = habilitado.

El bit Chroma0 (transmisión color) habilita o deshabilita chroma en señal banda base. 0 = deshabilitado, 1 = habilitado.

El campo AuralSub (aural sub-carrier) selecciona la fuente de la frecuencia de subcarreo del aura FM (audio) a ser modulada. La fuente es el PLLA de uno de los cogs, identificado por el valor de AuralSub. Este audio debe estar modulado en el subcarreo de 4.5MHz por la fuente PLLA.

Tabla 9: El Campo AuralSub	
AuralSub	Frecuencia Fuente de Sub Acarreo
000	Cog 0's PLLA
001	Cog 1's PLLA
010	Cog 2's PLLA
011	Cog 3's PLLA
100	Cog 4's PLLA
101	Cog 5's PLLA
110	Cog 6's PLLA
111	Cog 7's PLLA

El campo VGroup (Grupo de Pin Salida de Video) selecciona el grupo de 8 pins E/S por el cual saldrá el video.

Tabla 10: El Campo VGroup	
VGroup	Grupo de Pins
000	Grupo 0: P7..P0
001	Grupo 1: P15..P8
010	Grupo 2: P23..P16
011	Grupo 3: P31..P24
100-111	<reservado para uso futuro >

El campo VPins (Pins de Salida de Video) es una máscara aplicada a los pins de VGroup que indica por cual pin saldrá el video.

Tabla 11: El Campo VPins	
VPins	Efecto
00001111	Controla Video en los 4 pins bajos, compuesto
11110000	Controla Video en los 4 pins altos, compuesto
11111111	Controla Video en los 8 pins VGA
XXXXXXXX	Cualquier valor es válido en este campo; los mostrados arriba son los más comunes.

4.10.2. Registro de Escala de Video - VSCL

El registro de Escala de Video activa el rango al cual se generan los datos y se muestra en la Tabla 12.

Tabla 12: Registro VSCL		
Bits VSCL		
31..20	19..12	11..0
-	PixelClocks	FrameClocks

Los 8-bit del campo PixelClocks indican el numero de ciclos por pixel; el numero de ciclos que deben pasar antes de que cada pixel se cambia por el modulo generador de video. Estos ciclos son los ciclos PLLA, no los ciclos del sistema. Un valor de 0 para este campo se interpreta como 256.

El campo de 12-bit FrameClocks indica el numero de ciclos por cuadro; el numero de ciclos que pasara antes de que el modulo generador de video cambia el cuadro. Estos ciclos son los ciclos PLLA, no los ciclos del sistema. Un cuadro es un long de datos de pixel (entregado por la instrucción **WAITVID**). Como el dato de pixel es ya sea 16 bits por 2 o 32 bits por 1 bit (significa 16 pixeles ancho con 4 colores o 32 pixeles de ancho con 2 colores respectivamente), el FrameClocks es típicamente 16 o 32 veces el valor de PixelClocks. Un valor de 0 para este campo se interpreta como 4096.

4.10.3. Instrucción WAITVID

La instrucción **WAITVID** es el mecanismo de entrega para datos al hardware generador de video del cog. Como el generador de video trabaja independientemente del mismo cog, los dos deben sincronizar datos cada vez que se necesita desplegar información en un aparato. La frecuencia a la cual ocurre se dicta por la frecuencia del PLLA y el Registro de Escala de Video. El cog debe tener datos nuevos disponibles antes del momento en que el generador de video los necesite. El Cog usa **WAITVID** para esperar el momento preciso y luego entregar estos datos al generador de video.

Dos longs de datos se pasan al Generador de Video de acuerdo a la sintaxis **WAITVID Colors, Pixels**.

El parámetro *Colors* es un valor de 32-bit conteniendo cuatro valores de 8-bit de color (para modo 4 colores) o dos valores de 8-bit de color en los 16 bits bajos (para modo de 2 colores). Para modo VGA, cada valor de 8-bit se escribe a los pins especificados por los campos VGroup y VPins. Para VGA típicamente los 8 bits se agrupan en 2 bits por color primario y

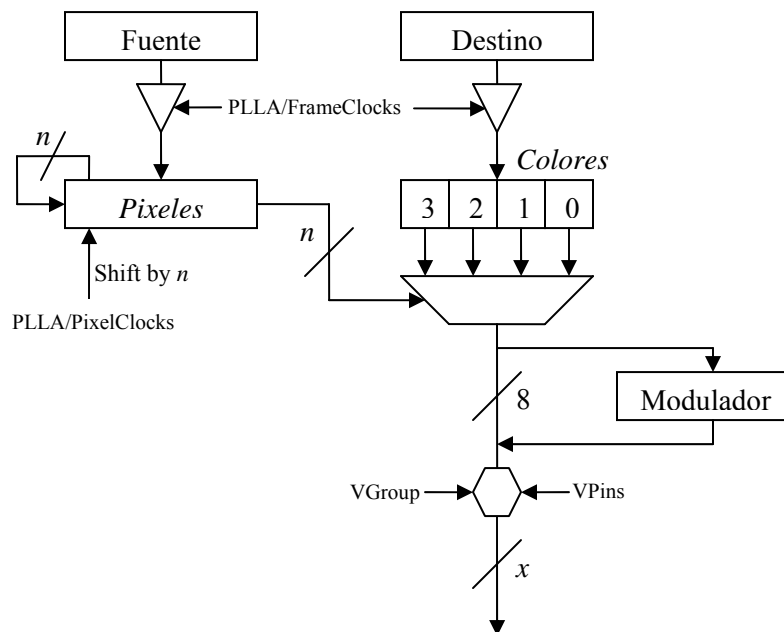
control de líneas verticales y horizontales, pero esto depende de la aplicación y software. Para video compuesto cada valor de 8-bit de color se compone de 3 campos. Bits 0-2 son el valor de iluminación de la señal generada. Bit 3 es la modulación la cual dicta si la información de color se generara en los bits 4-7 indicando el ángulo de fase del valor de color. Cuando el bit de modulación se pone a 0, la información de color se ignora. Cuando el bit de modulación esta en 1 el valor de luminosidad se modula ± 1 con un ángulo de fase activado por los bits 4-7. Para alcanzar la máxima resolución del valor de color, el PLLA debe estar activado 16 veces la frecuencia de modulación (en video compuesto esto se llama ruptura de frecuencia de color). El PLLB del cog se usa para generar la frecuencia de transmisión, si esta se genera depende de si PLLB está corriendo y de los valores de VMode y VPins.

El parámetro *Pixels* describe el patrón a desplegar, ya sea 16 pixeles o 32 pixeles dependiendo de la configuración de color del generador de video. Cuando el modo cuatro-colores se especifica, *Pixels* tiene un patrón de 16x2 donde cada pixel 2-bit es un índice en *Colors* en el cual el patrón de datos deberá presentarse a los pins. Cuando se especifica modo 2-colores, *Pixels* es un patrón de 32x1 donde cada bit especifica cuál es el valor de los dos patrones de colores en los 16 bits bajos de *Colors* que deberán salir a los pins. En los datos de Pixel se cambian primero los bits menos significativos (LSB). Cuando el valor de FrameClocks es mayor que 16 veces el valor de PixelClocks y el modo 4-colores se especifica, los dos bits más significativos se repiten hasta que los ciclos FrameClocks PLLA hayan ocurrido. Cuando el valor de FrameClocks es mayor de 32 veces el valor de PixelClocks y se especifico modo de 2-colores, el bit más significativo se repite hasta que los ciclos FrameClocks PLLA han pasado. Cuando los ciclos FrameClocks pasan y el cog no está en la instrucción **WAITVID**, los datos en la fuente y destino se obtendrán y usaran así que es importante estar en una instrucción **WAITVID** antes de que esto pase.

Aunque el Generador de Video fue creado para desplegar señales de video, sus aplicaciones son potencialmente más diversas. El modo de video compuesto puede usarse para generar comunicaciones phase-shift keying de una granularidad de 16 o menor y el modo VGA pueden usarse para generar cualquier patrón de bits con un rango completamente predecible.

La Figura 6 es un diagrama de bloques de cómo se organiza el modo VGA. Los dos triángulos invertidos son el mecanismo de carga de *Pixels* y *Colors*; n es 1 o 2 bits dependiendo del valor de CMode. El trapezoide invertido es un multiplexor 4-way 8-bit que escoge el byte de *Colors* para salida. Cuando está en modo de video compuesto el modulador transforma el byte en señal de luminosidad y color y transmite la señal. VGroup maneja los 8 bits a un bloque de pins de salida y pasa a aquellos pins que están activos en 1 de acuerdo a VPins; esta funcionalidad combinada se representa por el hexágono.

Figura 6: Generador de Video



4.11. Registro CLK

El registro CLK es la configuración de control del registro del reloj del sistema; determina la fuente y características del reloj del sistema. Configura el Oscilador RC, Reloj PLL, Cristal Oscilador y Circuito Selector de Reloj (Ver diagrama de Bloque, Pág. 1). Se configura al tiempo de compilación por la declaración `_CLKMODE` y se puede escribir en tiempo de ejecución a través de la instrucción `CLKSET`. Donde sea que el registro CLK se escriba, un retraso de ~75 μ s ocurre en la transición de la fuente de reloj.

Cada que el registro se cambia, una copia del valor escrito debe colocarse en el valor de la localidad del Modo Clock (que es el BYTE [4] en RAM principal) y la frecuencia resultante del reloj maestro deberá escribirse a la localidad del valor de la Frecuencia Clock (el cual es LONG [0] en RAM principal) así los objetos a los que hace referencia este dato tendrán la información actual para los cálculos de tiempo.

Use la instrucción Spin `CLKSET` cada que sea posible (ver secciones 6.3 y 6.4) ya que automáticamente actualiza todas las localidades mencionadas con la información adecuada.

Tabla 13: Modos Clock Validos

Expresión Valida	Valor CLK Reg.	Expresión Valida	Valor CLK Reg.
RCFAST	0_0_0_00_000	XTAL1 + PLL1X	0_1_1_01_011
RCSLOW	0_0_0_00_001	XTAL1 + PLL2X	0_1_1_01_100
		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
XTAL1	0_0_1_01_010	XTAL2 + PLL1X	0_1_1_10_011
		XTAL2 + PLL2X	0_1_1_10_100
		XTAL2 + PLL4X	0_1_1_10_101
		XTAL2 + PLL8X	0_1_1_10_110
XTAL2	0_0_1_10_010	XTAL2 + PLL16X	0_1_1_10_111
XTAL3	0_0_1_11_010	XTAL3 + PLL1X	0_1_1_11_011
XINPUT + PLL1X	0_1_1_00_011	XTAL3 + PLL2X	0_1_1_11_100
XINPUT + PLL2X	0_1_1_00_100	XTAL3 + PLL4X	0_1_1_11_101
XINPUT + PLL4X	0_1_1_00_101	XTAL3 + PLL8X	0_1_1_11_110
XINPUT + PLL8X	0_1_1_00_110	XTAL3 + PLL16X	0_1_1_11_111
XINPUT + PLL16X	0_1_1_00_111		

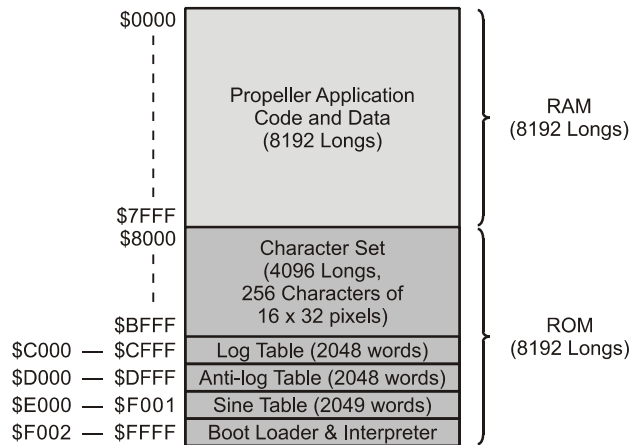
Tabla 14: Campo de Registros CLK

Bit	7	6	5	4	3	2	1	0
Nombre	RESET	PLLENA	OSCENA	OSCM1	OSCM2	CLKSEL2	CLKSEL1	CLKSEL0
RESET	Efecto							
0	Siempre escriba "0" aquí a menos que pretenda reiniciar el chip.							
1	Igual que reinicio de hardware- reinicia el chip.							
PLLENA	Efecto							
0	Deshabilita el circuito PLL							
1	Habilita el circuito PLL. El PLL interno multiplica la frecuencia del pin por 16. OSCENA debe ser '1' para propagar la señal de XIN al PLL. La frecuencia interna PLL debe mantenerse entre 64 MHz a 128 MHz – esto hace un rango de frecuencia en XIN de 4 MHz a 8 MHz. Permite 100 μ s para el PLL a estabilizar antes de cambiar a una de sus salidas a través de los bits CLKSEL. Una vez que los circuitos OSC y PLL están habilitados y estabilizados, usted puede cambiar libremente las fuentes Clock con solo cambiar los bits CLKSEL.							
OSCENA	Efecto							
0	Deshabilita el circuito OSC							
1	Habilita el circuito OSC para que la señal Clock pueda entrar por XIN, o para que XIN y XOUT puedan funcionar juntos como oscilador de retroalimentación. Los bits OSCM seleccionan el modo de operación del circuito OSC. Observe que no hay resistencias externas o capacitores para los cristales o resonadores. Permite estabilizar un cristal o resonador 10ms antes de cambiar a una salida OSC o PLL a través de los bits CLKSEL. Cuando habilita el circuito OSC, el PLL puede habilitar al mismo tiempo así estos pueden compartir el periodo de estabilización.							
OSCM1	OSCM2	Resistencia XOUT		Capacitancia XIN y XOUT		Rango de Frecuencia		
0	0	Infinito		6 pF (pad solo)		DC a 80 MHz Input		
0	1	2000 Ω		36 pF		4 MHz a 16 MHz Cristal/Resonador		
1	0	1000 Ω		26 pF		8 MHz a 32 MHz Cristal/Resonador		
1	1	500 Ω		16 pF		20 MHz a 60 MHz Cristal/Resonador		
CLKSEL2	CLKSEL1	CLKSEL0	Clock Maestro		Fuente	Notas		
0	0	0	~12 MHz		Interna	No partes externas (8 a 20 MHz)		
0	0	1	~20 kHz		Interna	No partes externas, baja potencia (13-33 kHz)		
0	1	0	XIN		OSC	OSCENA debe ser '1'		
0	1	1	XIN \times 1		OSC+PLL	OSCENA y PLLENA debe ser '1'		
1	0	0	XIN \times 2		OSC+PLL	OSCENA y PLLENA debe ser '1'		
1	0	1	XIN \times 4		OSC+PLL	OSCENA y PLLENA debe ser '1'		
1	1	0	XIN \times 8		OSC+PLL	OSCENA y PLLENA debe ser '1'		
1	1	1	XIN \times 16		OSC+PLL	OSCENA y PLLENA debe ser '1'		

5.0 ORGANIZACIÓN DE MEMORIA

5.1. Memoria Principal

La memoria principal es un bloque de 64Kbytes (16k Longs) que se accesa por todos los cogs como un recurso mutuamente exclusivo a través del hub. Consiste en 32KB de RAM y 32KB de ROM. La memoria principal es direccionable en byte, word y long. Words y Longs se almacenan en un formato endian; el byte menos significativo primero.



5.1.1. RAM principal

Los 32KB de RAM principal son de propósito general y es el destino de la aplicación Propeller ya sea para descargar de un host o de la memoria externa de 32 KB EEPROM.

5.1.2. ROM principal

Los 32 KB de ROM principal contiene todo el código y datos de recursos vitales para el funcionamiento del chip Propeller: definiciones de caracteres, tablas de log, anti-log y seno así como el cargador de inicio y el interprete Spin.

5.1.3. Definición de Caracteres

La primera mitad de ROM está dedicada para activar la definición de un grupo de 256 caracteres. Cada definición de carácter es 16 pixeles de ancho por 32 de alto. Estas definiciones de caracteres pueden usarse para generación de video, gráficos LCD, impresión, etc.

El grupo de caracteres está basado en una base Norte Americana / Este Europeo, con muchos caracteres especiales agregados e insertados. Hay formas de conexión de ondas y caracteres ara construir bloques, caracteres griegos usados comúnmente en electrónica y varias flechas y marcadores. (Una fuente Parallax está instalada y se usa por el Software de la herramienta Propeller y está disponible para otras aplicaciones Windows.)

La definición de caracteres están numerados del 0 al 255 de izquierda a derecha y de arriba a abajo de acuerdo a la Figura 7 de abajo. Están acomodados como sigue: Cada par de adyacentes par-impar están mezclados juntos para formar 32 longs. El primer par de caracteres está localizado en \$8000-\$807F. El Segundo par ocupa \$8080-\$80FF, y así sucesivamente, hasta el último par que llena \$BF80-\$BFFF.

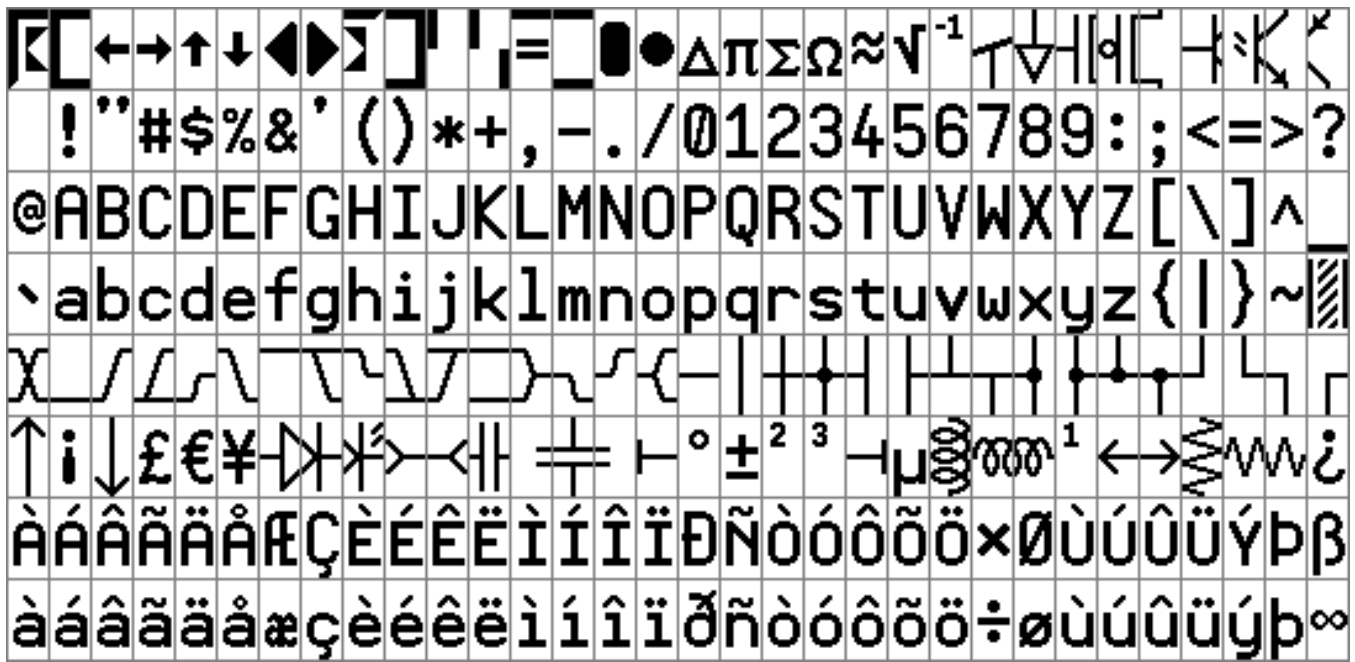


Figura 7: Grupo de caracteres de la Fuente Propeller

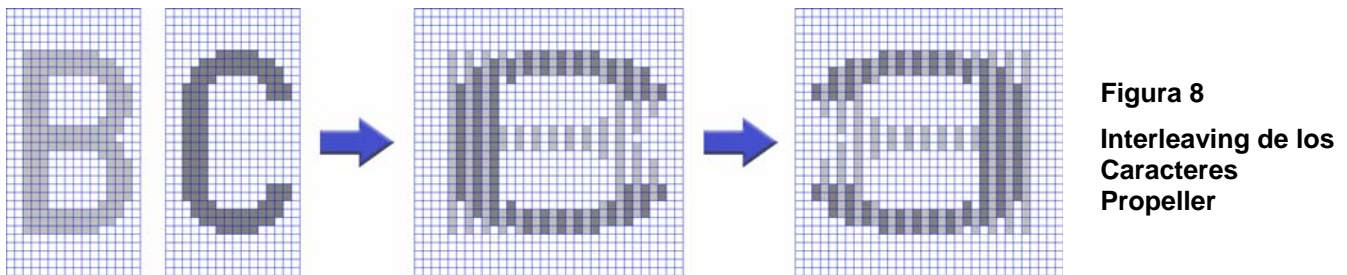


Figura 8
Interleaving de los
Caracteres
Propeller

Como se muestra en la Figura 8, los pares de caracteres están mezclados renglón por renglón de modo que cada 16 pixeles horizontales de carácter están espaciados aparte y segregados con sus vecinos para que los caracteres pares tomen los bits 0, 2, 4, ... 30 y los caracteres impares tomen los bits 1, 3, 5, ...31. Los pixels de la extrema izquierda están en los bits bajos mientras que los de la derecha están en los altos. Esto forma un long por cada renglón de pixeles en el par de caracteres. 32 de estos longs, de arriba abajo hace la definición completa de la definición de pares mezclados. Las definiciones están codificadas de esta forma para que el hardware de video del cog pueda manejar los longs mezclados directamente, usando selección de color para desplegar los caracteres pares o impares.

Algunos códigos de caracteres tienen significados inevitables, tal como el 9 para Tab, 10 para línea de alimentación y 13 para el retorno. Estos caracteres invocan acciones y no es igual a las definiciones estáticas. Por esta razón las definiciones de caracteres se han usado para caracteres especiales de cuatro colores. Estos caracteres de cuatro colores son usados para dibujar contornos de cajas 3d en tiempo de ejecución y se implementan como celdas de 16 x 16 pixeles, opuesto a las celdas normales de 16 x 32 pixeles. Ellos ocupan caracteres par-impares 0-1, 8-9, 10-11, y 12-13.

5.1.4. Tablas Matemáticas de Funciones

Tablas de Logaritmo base 2 Anti Logaritmo, cada una con 2048 words no signadas, facilitan la conversión de valores a y de formas exponentes para facilitar operaciones; ver el manual Propeller para acceso a las instrucciones. También una tabla de seno proporciona 2049 muestras no signadas de 16-bit para 0° a 90° (resolución de 0.0439°).

6.0 LEGUAJES DE PROGRAMACIÓN

El chip Propeller se programa usando dos lenguajes designados específicamente para el: 1) Spin, un lenguaje de alto nivel basado en objetos, y 2) Ensamblador Propeller, un nivel de bajo nivel de lenguaje ensamblador altamente optimizado. Hay muchas instrucciones basadas en hardware en ensamblador Propeller que tienen sus equivalentes directos en lenguaje Spin.

El lenguaje spin se compila en la herramienta del Software Propeller en símbolos que se interpretan en tiempo de ejecución por el intérprete Spin integrado en el chip propeller. El Ensamblador Propeller se ensambla en código maquina puro por la herramienta Propeller y se ejecuta en forma pura al momento de ejecución.

Los objetos propeller pueden escribirse enteramente en Spin o pueden usar combinaciones de Spin y Ensamblador Propeller. Frecuentemente tiene ventajas escribir objetos mayormente en Ensamblador Propeller completamente, pero al menos dos líneas de código en Spin se necesitan para iniciar la aplicación final.

6.1. Lista de Palabras reservadas

Las siguientes palabras son reservadas, no importa si programa en Spin o Ensamblador. **De la herramienta Propeller v1.05:**

Tabla 16: Lista de Palabras Reservadas

_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	MOV ^d	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#}	POX ^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	RCLOW ^s	TEST ^a	
CMPSX ^a	IF_ALWAYS ^a	INB ^{d#}	NEGNZ ^a	RDBYTE ^a	TESTN ^a	
CMPX ^a	IF_B ^a	JMP ^a	NEGZ ^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 = Elemento Ensamblador; s = Elemento Spin; d = Dual (disponible en ambos lenguajes); # = Reservada para uso Futuro

6.1.1. Palabras reservadas para Uso Futuro

- **DIRB, INB, y OUTB:** Reservadas para uso futuro con un posible modelo de 64 pins E/S. Cuando se usa el P8X32A, estas etiquetas pueden usarse para acceder la RAM del Cog en esas localidades para propósito general solamente.
- **ENC, MUL, MULS, ONES:** Usar con la arquitectura actual P8X32A resulta en alcances indeterminados.

6.2. Operadores Matemáticos y Lógicos

Tabla 17: Operadores Matemáticos y Lógicos						
Nivel ¹	Operador		Expresiones Constantes ³		Es Unario	Descripción
	Normal	Asignado ²	Entero	Flotante		
Mas alto (0)	--	Siempre			✓	Pre-decremento (--X) o post-decremento (X--).
	++	Siempre			✓	Pre-incremento (++X) o post-incremento (X++).
	~	Siempre			✓	Signo-extendido bit 7 (~X) o post-limpiar a 0 (X~).
	~~	Siempre			✓	Signo-extendido bit 15 (~~X) o post-activar a -1 (X~~).
	?	Siempre			✓	Número aleatorio al frente (?X) o reversa (X?).
	e	Nunca	✓		✓	Símbolo de dirección.
	ee	Nunca			✓	Dirección de objeto mas símbolo
1	+	Nunca	✓	✓	✓	Positivo (+X); forma unaria de suma.
	-	si solo	✓	✓	✓	Negado (-X); forma unaria de resta.
	^^	si solo	✓	✓	✓	Raíz cuadrada.
		si solo	✓	✓	✓	Valor Absoluto.
	<	si solo	✓		✓	Bitwise: Decodifica 0 – 31 a long con un bit sencillo alto.
	>	si solo	✓		✓	Bitwise: Codifica Long a 0 – 32; bit de alta prioridad.
	!	si solo	✓		✓	Bitwise: NOT.
2	<-	<-=	✓			Bitwise: Rotación a la izquierda.
	->	->=	✓			Bitwise: Rotación a la derecha.
	<<	<<=	✓			Bitwise: Mueve a la izquierda.
	>>	>>=	✓			Bitwise: Mueve a la derecha.
	~>	~>=	✓			Corrimiento aritmético a la derecha.
	><	><=	✓			Bitwise: Reversa.
3	&	&=	✓			Bitwise: AND.
4		=	✓			Bitwise: OR.
	^	^=	✓			Bitwise: XOR.
5	*	*=	✓	✓		Multiplica y regresa los32 bits bajos (signados).
	**	**=	✓			Multiplica y regresa los 32 bits altos (signados).
	/	/=	✓	✓		Divide (signado).
	//	//=	✓			Modulus (signado).
6	+	+=	✓	✓		Suma.
	-	-=	✓	✓		Resta.
7	#>	#>=	✓	✓		Limite mínimo (signado).
	<#	<#=	✓	✓		Limite máximo (signado).
8	<	<=	✓	✓		Booleano: Menor que (signado).
	>	>=	✓	✓		Booleano: Mayor que (signado).
	<>	<>=	✓	✓		Booleano: No es igual.
	==	===	✓	✓		Booleano: Igual.
	=<	=<=	✓	✓		Booleano: Es igual o menor (signado).
	=>	=>=	✓	✓		Booleano: Es igual o mayor (signado).
9	NOT	si solo	✓	✓	✓	Booleano: NOT (promueve un no 0 a -1).
10	AND	AND=	✓	✓		Booleano: AND (promueve un no 0 a -1).
11	OR	OR=	✓	✓		Booleano: OR (promueve un no 0 a -1).
Mas Bajo (12)	=	Siempre	n/a ³	n/a ³		Asignación de constantes (Bloque CON).
	:=	Siempre	n/a ³	n/a ³		Asignación de Variable (Bloque PUB/PRI).

¹ Nivel de precedencia: operadores de mayor nivel se evalúan antes que los de menor nivel. Operadores en el mismo nivel son conmutables; el orden de la evaluación no importa.

² Asignación de operadores de forma binaria (no unaria) están en menor precedencia (nivel 12).

³ Asignación de formas de operadores no están permitidos en expresiones constantes.

6.3. Tabla de Lenguaje Spin

Instrucción Spin	Regresa Valor	Descripción
ABORT <i><Value></i>	✓	Sale de métodos PUB/PRI usando abort con opción a valor de regreso.
BYTE <i>Symbol <[Count]></i>		Declara símbolos de tamaño byte en bloques VAR.
<i><Symbol></i> BYTE <i>Data <[Count]></i>		Declara datos byte alineado o tamaño byte en cloques DAT.
BYTE [<i>BaseAddress</i>] <i><[Offset]></i>	✓	Lee/Escribe byte en memoria principal.
<i>Symbol</i> .BYTE <i><[Offset]></i>	✓	Lee/Escribe componente tamaño byte o variable tamaño long/word.
BYTEFILL (<i>StartAddress</i> , <i>Value</i> , <i>Count</i>)		Llena bytes de memoria principal con un valor.
BYTEMOVE (<i>DestAddress</i> , <i>SrcAddress</i> , <i>Count</i>)		Copia bytes de una región a otra en memoria principal.
CASE <i>CaseExpression</i> → <i>MatchExpression</i> : → <i>Statement(s)</i> <i><MatchExpression</i> : → <i>Statement(s)</i> <i><MatchExpression</i> : → <i>OTHER</i> : → <i>Statement(s)</i>		Compara expresiones contra expresiones compatibles, ejecuta bloque de código si encuentra igualdad. <i>MatchExpression</i> puede contener expresiones simples o múltiples de coma delimitada. Las expresiones pueden ser de un valor sencillo (ej: 10) o un rango de valores (ej: 10..15).
CHIPVER	✓	Versión del chip propeller (Byte en \$FFFF)
CLKFREQ	✓	Frecuencia del Reloj del Sistema, en Hz (Long en \$0000)
CLKMODE	✓	Parámetros actuales del modo Clock (Byte en \$0004)
CLKSET (<i>Mode</i> , <i>Frequency</i>)		Activa Modo del reloj y Frecuencia del Reloj del Sistema en tiempo de ejecución.
CNT	✓	Valor actual 32-bit del contador del sistema.
COGID	✓	Numero de ID del Cog actual; 0-7.
COGINIT (<i>CogID</i> , <i>SpinMethod <(ParameterList)></i> , <i>StackPointer</i>)		Inicia o reinicia un cog por ID para correr código Spin.
COGINIT (<i>CogID</i> , <i>AsmAddress</i> , <i>Parameter</i>)		Inicia o reinicia un cog por ID para correr Ensamblador Propeller.
COGNEW (<i>SpinMethod <(ParameterList)></i> , <i>StackPointer</i>)	✓	Inicia un nuevo cog para código spin y obtiene ID del cog; 0-7 = si tiene éxito, -1 = si falla.
COGNEW (<i>AsmAddress</i> , <i>Parameter</i>)	✓	Inicia nuevo cog para Ensamblador Propeller y obtiene ID del cog; 0-7 = si tiene éxito, -1 = si falla.
COGSTOP (<i>CogID</i>)		Tiene un cog por su ID.
CON <i>Symbol = Expr <((, [L]) Symbol = Expr)...</i>		Declara constante global simbólica
CON <i>*Expr ((, [L]) Symbol <[Offset]> <((, [L]) Symbol <[Offset]> ...</i>		Declara numeraciones globales (incrementando constantes simbólicas).
CON <i>Symbol <[Offset]> <((, [L]) Symbol <[Offset]>)...</i>		Declara numeraciones globales (incrementando constantes simbólicas).
CONSTANT (<i>ConstantExpression</i>)	✓	Declara expresiones constantes en línea para resolverse completamente en tiempo de compilación.
CTRA	✓	Control del Registro Contador A.
CTRB	✓	Control del Registro Contador B.
DAT <i><Symbol> Alignment <Size> <Data> <[Count]> <(<Size> Data <[Count]>)...</i>		Declara tabla de datos, alineación y tamaño especificado.
DAT <i><Symbol> <Condition> Instruction <Effect(s)></i>		Denota instrucción de Ensamblador propeller.
DIRA <i><[Pin(s)]></i>	✓	Registro de Dirección D para 32-bit puerto A. Por defecto es 0 (entrada) cuando inicia el cog.
FILE " <i>FileName</i> "		Importa archivo extremo como datos en un bloque DAT.
FLOAT (<i>IntegerConstant</i>)	✓	Convierte expresión constante entera a un valor de punto flotante en tiempo de compilación en cualquier bloque.
FRQA	✓	Registro de Contador de Frecuencia A.
FRQB	✓	Registro de Contador de Frecuencia B.

Instrucción Spin	Regresa Valor	Descripción
((IF <input type="checkbox"/> IFNOT)) <i>Condition(s)</i> → <i>IfStatement(s)</i> <ELSEIF <i>Condition(s)</i> → <i>ElseIfStatement(s)</i> ... <ELSEIFNOT <i>Condition(s)</i> → <i>ElseIfStatement(s)</i> ... <ELSE → <i>ElseStatement(s)</i>		Prueba condición y ejecuta bloque de código si es valido. IF y ELSEIF cada uno prueba TRUE . IFNOT y ELSEIFNOT cada uno prueba por FALSE .
INA <i><[Pin(s)]></i>	✓	Registro de entrada puerto A 32-bit.
LOCKCLR <i>(ID)</i>	✓	Limpia semáforo a falso y obtiene estado previo; TRUE o FALSE .
LOCKNEW	✓	Verifica nuevo semáforo y obtiene su ID; 0-7, o -1 si no hay disponibles.
LOCKRET <i>(ID)</i>		Regresa el semáforo a la pila, liberándolo para futuras LOCKNEW .
LOCKSET <i>(ID)</i>	✓	Activa semáforo a verdad y obtiene su estado previo; TRUE o FALSE .
LONG <i>Symbol <[Count]</i>		Declara símbolo tamaño long en bloque VAR .
<i><Symbol></i> LONG <i>Data <[Count]</i>		Declara dato long alineado o tamaño long en bloque DAT .
LONG <i>[BaseAddress] <[Offset]</i>	✓	Lee/escrbe un long en memoria principal.
LONGFILL <i>(StartAddress, Value, Count)</i>		Llena con longs memoria principal con un valor.
LONGMOVE <i>(DestAddress, SrcAddress, Count)</i>		Copia longs de una región a otra en memoria principal.
LOOKDOWN <i>(Value: ExpressionList)</i>	✓	Obtiene el índice base-uno de un valor en la lista.
LOOKDOWNZ <i>(Value: ExpressionList)</i>	✓	Obtiene el índice base-cero de un valor en la lista.
LOOKUP <i>(Index: ExpressionList)</i>	✓	Obtiene el índice base-uno de un valor en la lista.
LOOKUPZ <i>(Index: ExpressionList)</i>	✓	Obtiene el índice base-cero de un valor en la lista.
NEXT		Salta las instrucciones restantes del ciclo REPEAT y continua con el siguiente ciclo.
OBJ <i>Symbol <[Count]: "Object" <↳ Symbol <[Count]: "Object">...</i>		Declara referencias del símbolo de objeto .
OUTA <i><[Pin(s)]></i>	✓	Registro de salida Puerto a 32-bit. 0 por defecto 9tierra0 cuando inicia cog.
PAR	✓	Registro de parámetros de inicio de cog .
PHSA	✓	Registro Contador A de Ciclo de Fase cerrada (PLL).
PHSB	✓	Registro Contador B de Ciclo de Fase cerrada (PLL).
PRI <i>Name <{Par <,Par>...}> (:RVal) < LVar <[Cn]</i>		Declara método privado con parámetros opcionales, valor de regreso y variables locales .
PUB <i>Name <{Par <,Par>...}> (:RVal) < LVar <[Cn]</i>		Declara método publico con parámetros opcionales, regresa valor y variables locales .
QUIT		Ale del ciclo REPEAT Inmediatamente.
REBOOT		Reinicia el chip Propeller.
REPEAT <i><Count></i> → <i>Statement(s)</i>		Ejecuta bloque de código repetitivamente, o infinitamente o por un numero finito de iteraciones.
REPEAT <i>Variable FROM Start TO Finish <STEP Delta></i> → <i>Statement(s)</i>		Ejecuta bloques de código repetidamente, por un número de iteraciones finitas.
REPEAT ((UNTIL <input type="checkbox"/> WHILE)) <i>Condition(s)</i> → <i>Statement(s)</i>		Ejecuta bloque de código repetidamente de cero a muchas iteraciones condicionales.
REPEAT → <i>Statement(s)</i> ((UNTIL <input type="checkbox"/> WHILE)) <i>Condition(s)</i>		Ejecuta bloque de código repetidamente de una a muchas iteraciones condicionales.
RESULT	✓	Regresa el valor variable para métodos PUB/PRI .
RETURN <i><Value></i>	✓	Sale de un método PUB/PRI con opción a un valor de regreso.
ROUND <i>(FloatConstan)</i>	✓	Redondea constantes de puntos flotantes al siguiente entero.
SPR <i>[Index]</i>	✓	Arreglo de Registros de Propósito Especial.
STRCOMP <i>(StringAddress1, StringAddress2)</i>	✓	Compara dos cadenas para igualdad.
STRING <i>(StringExpression)</i>	✓	Declara cadenas constantes en línea y obtiene su dirección.

Instrucción Spin	Regresa Valor	Descripción
STRSIZE (<i>StringAddress</i>)	✓	Obtiene tamaño en bytes de una cadena cero terminada.
TRUNC (<i>FloatConstant</i>)	✓	Remueve la parte fraccional de una constante de punto flotante en tiempo de compilación en cualquier bloque.
VAR <i>Size Symbol <[Count]> <<((, ↪ Size)) Symbol <[Count]>>...</i>		Declara variables globales simbólicas .
VCFG	✓	Registro de configuración de video.
VSCL	✓	Registro de Escala de video.
WAITCNT (<i>Value</i>)		Detiene la ejecución del cog temporalmente.
WAITPEQ (<i>State, Mask, Port</i>)		Detiene la ejecución del cog hasta que un pin E/S coincide con el estado designado.
WAITPNE (<i>State, Mask, Port</i>)		Detiene la ejecución del cog hasta que un pin E/S no coincide con el estado designado.
WAITVID (<i>Colors, Pixels</i>)		Detiene la ejecución del cog hasta que el generador de video esta disponible para datos de pixeles.
WORD <i>Symbol <[Count]></i>		Declara símbolo tamaño word en un bloque VAR.
<i><Symbol> WORD Data <[Count]></i>		Declara datos word alineado o tamaño word en un bloque DAT.
WORD [<i>BaseAddress</i>] <i><[Offset]></i>	✓	Lee/Escribe Word en memoria principal.
<i>Symbol.WORD <[Offset]></i>	✓	Lee/Escribe componente tamaño word de una variable tamaño long.
WORDFILL (<i>StartAddress, Value, Count</i>)		Llena words de memoria principal con un valor.
WORDMOVE (<i>DestAddress, SrcAddress, Count</i>)		Copia words de una región a otra en memoria principal.

6.3.1. Constantes

Constantes (pre-definidas)		
Constante ¹	Descripción	
_CLKFREQ	Activable en objeto superior para especificar frecuencia del sistema de reloj.	
_CLKMODE	Activable en objeto superior para especificar modo de reloj .	
_XINFREQ	Activable en objeto superior para especificar frecuencia de cristal externo.	
_FREE	Activable en objeto superior para especificar espacio libre .	
_STACK	Activable en objeto superior para especificar espacio de pila.	
TRUE	Lógico verdadero:	-1 (\$FFFFFFF)
FALSE	Lógico falso:	0 (\$0000000)
POSX	Max. entero positivo:	2,147,483,647 (\$7FFFFFFF)
NEGX	Max. entero negativo:	-2,147,483,648 (\$8000000)
PI	Punto Flotante PI:	≈ 3.141593 (\$40490FDB)
RCFAST	Oscilador Interno Rápido:	\$00000001 (%000000000001)
RCLOW	Oscilador Interno Lento:	\$00000002 (%000000000010)
XINPUT	Reloj/Oscilador Externo:	\$00000004 (%00000000100)
XTAL1	Cristal externo de baja velocidad:	\$00000008 (%00000001000)
XTAL2	Cristal externo media velocidad:	\$00000010 (%00000010000)
XTAL3	Cristal externo alta velocidad:	\$00000020 (%00000100000)
PLL1X	Frecuencia externa 1 vez:	\$00000040 (%00001000000)
PLL2X	Frecuencia externa 2 veces:	\$00000080 (%000010000000)
PLL4X	Frecuencia externa 4 veces:	\$00000100 (%00100000000)
PLL8X	Frecuencia externa 8 veces:	\$00000200 (%01000000000)
PLL16X	Frecuencia externa 16 veces:	\$00000400 (%10000000000)

¹"Las constantes "activables" se definen en objetos superiores de bloque **CON**. Ver modos validos para **_CLKMODE**. Otras constantes activables usa números completos.

6.4. Tabla de Instrucciones Ensamblador Propeller

La tabla de instrucciones ensamblador Propeller enlista las instrucciones de código operacional 32-bit, salidas y numero de ciclos de reloj. El código operacional consiste en bits de instrucciones (**iiiiii**), el estado de "efecto" para la bandera Z, bandera C, resultado y estado indirecto inmediato (**zcri**), los bits de ejecución condicional (**cccc**), y los bits fuente y destino (**ddddddddd** y **sssssssss**). El significado de la bandera Z y C, si lo hay, se muestra en los campos Z Result y C Result; indicando el significado de un 1 en esas banderas. El campo Result (R) muestra el comportamiento por defecto de la instrucción para escribir (1) o no escribir (0) el valor del resultado de la instrucción. El campo Clocks muestra el número de ciclos que requiere la instrucción para su ejecución.

- 0 1 Ceros (0) y unos (1) significa binario 0 y 1.
- i minúscula "i" indica un bit afectado por el estado inmediato.
- d s minúscula "d" y "s" indica bits destino y fuente.
- ? signos de interrogación indica que es un grupo activado dinámicamente en compilación.
- guiones indican partes que no aplican o no son importantes.
- .. doble punto representa un rango de valores contiguos.

iiiiii	zcri	cccc	ddddddddd	sssssssss	Instrucción	Descripción	Resultado Z	Resultado C	R	Ciclos
000000	000i	1111	ddddddddd	sssssssss	WRBYTE D, S	Escribe D[7..0] a mem principal byte S[15..0]	-	-	0	7.22 *
000000	001i	1111	ddddddddd	sssssssss	RDBYTE D, S	Lee memoria principal byte S[15..0] en D (0-extendida)	Resultado = 0	-	1	7.22 *
000001	000i	1111	ddddddddd	sssssssss	WRWORD D, S	Escribe D[15..0] a mem principal word S[15..1]	-	-	0	7.22 *
000001	001i	1111	ddddddddd	sssssssss	RDWORD D, S	Lee memoria principal word S[15..1] en D (0-extendida)	Resultado = 0	-	1	7.22 *
000010	000i	1111	ddddddddd	sssssssss	WRLONG D, S	Escribe D a memoria principal long S[15..2]	-	-	0	7.22 *
000010	001i	1111	ddddddddd	sssssssss	RDLONG D, S	Lee memoria principal long S[15..2] en D	Resultado = 0	-	1	7.22 *
000011	000i	1111	ddddddddd	sssssssss	HUBOP D, S	Desarrolla operación hub de acuerdo a S	Resultado = 0	-	0	7.22 *
000011	0001	1111	ddddddddd	-----000	CLKSET D	Activa registro global CLK a D[7..0]	-	-	0	7.22 *
000011	0011	1111	ddddddddd	-----001	COGID D	Obtiene numero de cog (0..7) en D	ID = 0	0	1	7.22 *
000011	0001	1111	ddddddddd	-----010	COGINIT D	Inicializa un cog de acuerdo a D	ID = 0	Cog no libre	0	7.22 *
000011	0001	1111	ddddddddd	-----011	COGSTOP D	Detiene un número de cog D[2..0]	Detenido ID = 0	Cog no libre	0	7.22 *
000011	0011	1111	ddddddddd	-----100	LOCKNEW D	Obtiene un número de LOCK (0..7) en D	ID = 0	Seguro no libre	1	7.22 *
000011	0001	1111	ddddddddd	-----101	LOCKRET D	Regresa un número de seguro D[2..0]	ID = 0	Seguro no libre	0	7.22 *
000011	0001	1111	ddddddddd	-----110	LOCKSET D	Activa un número de seguro D[2..0]	ID = 0	Edo seg. anterior	0	7.22 *
000011	0001	1111	ddddddddd	-----111	LOCKCLR D	Limpia un número de seguro D[2..0]	ID = 0	Edo seg anterior	0	7.22 *
000100	001i	1111	ddddddddd	sssssssss	MUL D, S	Multiplica un no signado D[15..0] por S[15..0]	Resultado = 0	-	1	Futuro
000101	001i	1111	ddddddddd	sssssssss	MULS D, S	Multiplica un no signado D[15..0] por S[15..0]	Resultado = 0	-	1	Futuro
000110	001i	1111	ddddddddd	sssssssss	ENC D, S	Codifica magnitud de S en D, resulta = 0..31	Resultado = 0	-	1	Futuro
000111	001i	1111	ddddddddd	sssssssss	ONES D, S	Obtiene numero de 1 en S en D, resulta = 0..31	Resultado = 0	-	1	Futuro
001000	001i	1111	ddddddddd	sssssssss	ROR D, S	Rota D a la derecha por S[4..0] bits	Resultado = 0	D[0]	1	4
001001	001i	1111	ddddddddd	sssssssss	ROL D, S	Rota D a la izquierda por S[4..0] bits	Resultado = 0	D[31]	1	4
001010	001i	1111	ddddddddd	sssssssss	SHR D, S	Mueve D a la derecha por S[4..0] bits, activa nuevo MSB a 0	Resultado = 0	D[0]	1	4
001011	001i	1111	ddddddddd	sssssssss	SHL D, S	Mueve D a la izquierda por S[4..0] bits, activa nuevo LSB a 0	Resultado = 0	D[31]	1	4
001100	001i	1111	ddddddddd	sssssssss	RCR D, S	Rota acarreo a la derecha en D por S[4..0] bits	Resultado = 0	D[0]	1	4
001101	001i	1111	ddddddddd	sssssssss	RCL D, S	Rota acarreo a la izquierda en D por S[4..0] bits	Resultado = 0	D[31]	1	4
001110	001i	1111	ddddddddd	sssssssss	SAR D, S	Mueve D aritméticamente a la derecha por S[4..0] bits	Resultado = 0	D[0]	1	4
001111	001i	1111	ddddddddd	sssssssss	REV D, S	Reversa 32-S[4..0] bits de arriba en D y 0-extendidos	Resultado = 0	D[0]	1	4
010000	001i	1111	ddddddddd	sssssssss	MINS D, S	Activa D a S si es signado (D < S)	S = 0	Signado (D < S)	1	4
010001	001i	1111	ddddddddd	sssssssss	MAXS D, S	Activa D a S si es signado (D => S)	S = 0	Signado (D < S)	1	4
010010	001i	1111	ddddddddd	sssssssss	MIN D, S	Activa D a S si es no signado (D < S)	S = 0	No signado (D < S)	1	4
010011	001i	1111	ddddddddd	sssssssss	MAX D, S	Activa D a S si es no signado (D => S)	S = 0	No signado (D < S)	1	4

iiii zcri cccc dddddddd ssssssss	Instrucción	Descripción	Resultado Z	Resultado C	R	Ciclos
010100 001i 1111 dddddddd ssssssss	MOVS D, S	Inserta S[8..0] en D[8..0]	Resultado = 0	-	1	4
010101 001i 1111 dddddddd ssssssss	MOVD D, S	Inserta S[8..0] en D[17..9]	Resultado = 0	-	1	4
010110 001i 1111 dddddddd ssssssss	MOVI D, S	Inserta S[8..0] en D[31..23]	Resultado = 0	-	1	4
010111 001i 1111 dddddddd ssssssss	JMPRET D, S	Inserta PC+1 en D[8..0] y activa PC a S[8..0]	Resultado = 0	-	1	4
010111 000i 1111 ----- ssssssss	JMP S	Activa PC a S[8..0]	Resultado = 0	-	0	4
010111 0011 1111 ????????? ssssssss	CALL #S	Como JMPRET, pero ensamblador maneja detalles	Resultado = 0	-	1	4
010111 0001 1111 -----	RET	Como JMP, pero ensamblador maneja detalles	Resultado = 0	-	0	4
011000 000i 1111 dddddddd ssssssss	TEST D, S	AND S con D solo para afectar banderas	D = 0	Paridad Resultado	0	4
011001 000i 1111 dddddddd ssssssss	TESTN D, S	AND !S en D solo para afectar banderas	Resultado = 0	Paridad Resultado	0	4
011000 001i 1111 dddddddd ssssssss	AND D, S	AND S en D	Resultado = 0	Paridad Resultado	1	4
011001 001i 1111 dddddddd ssssssss	ANDN D, S	AND !S en D	Resultado = 0	Paridad Resultado	1	4
011010 001i 1111 dddddddd ssssssss	OR D, S	OR S en D	Resultado = 0	Paridad Resultado	1	4
011011 001i 1111 dddddddd ssssssss	XOR D, S	XOR S en D	Resultado = 0	Paridad Resultado	1	4
011100 001i 1111 dddddddd ssssssss	MUXC D, S	Copia C a bits en D usando S como mascara	Resultado = 0	Paridad Resultado	1	4
011101 001i 1111 dddddddd ssssssss	MUXNC D, S	Copia !C a bits en D usando S como mascara	Resultado = 0	Paridad Resultado	1	4
011110 001i 1111 dddddddd ssssssss	MUXZ D, S	Copia Z a bits en D usando S como mascara	Resultado = 0	Paridad Resultado	1	4
011111 001i 1111 dddddddd ssssssss	MUXNZ D, S	Copia !Z a bits en D usando S como mascara	Resultado = 0	Paridad Resultado	1	4
100000 001i 1111 dddddddd ssssssss	ADD D, S	Suma S en D	D + S = 0	Acarreo no signo	1	4
100001 001i 1111 dddddddd ssssssss	SUB D, S	Resta S de D	D - S = 0	Prestado no signo	1	4
100001 000i 1111 dddddddd ssssssss	CMP D, S	Compara D con S	D = S	No signado (D < S)	0	4
100010 001i 1111 dddddddd ssssssss	ADDABS D, S	Suma absoluta de S y D	D + S = 0	Acarreo no signo ¹	1	4
100011 001i 1111 dddddddd ssssssss	SUBABS D, S	Resta absoluta S de D	D - S = 0	Prestado no sig ²	1	4
100100 001i 1111 dddddddd ssssssss	SUMC D, S	Suma -S si C o S si !C en D	D ± S = 0	Sobreflujo signo	1	4
100101 001i 1111 dddddddd ssssssss	SUMNC D, S	Suma S si C o -S si !C en D	D ± S = 0	Sobreflujo signo	1	4
100110 001i 1111 dddddddd ssssssss	SUMZ D, S	Suma -S si Z o S si !Z en D	D ± S = 0	Sobreflujo signo	1	4
100111 001i 1111 dddddddd ssssssss	SUMNZ D, S	Suma S si Z o -S si !Z en D	D ± S = 0	Sobreflujo signo	1	4
101000 001i 1111 dddddddd ssssssss	MOV D, S	Activa D a S	Resultado = 0	S[31]	1	4
101001 001i 1111 dddddddd ssssssss	NEG D, S	Activa D a -S	Resultado = 0	S[31]	1	4
101010 001i 1111 dddddddd ssssssss	ABS D, S	Activa D a S absoluta	Resultado = 0	S[31]	1	4
101011 001i 1111 dddddddd ssssssss	ABSNEG D, S	Activa D a -absoluta S	Resultado = 0	S[31]	1	4
101100 001i 1111 dddddddd ssssssss	NEGC D, S	Activa D a -S si C o S si !C	Resultado = 0	S[31]	1	4
101101 001i 1111 dddddddd ssssssss	NEGNC D, S	Activa D a S si C o -S si !C	Resultado = 0	S[31]	1	4
101110 001i 1111 dddddddd ssssssss	NEGZ D, S	Activa D a -S si Z o S si !Z	Resultado = 0	S[31]	1	4
101111 001i 1111 dddddddd ssssssss	NEGNZ D, S	Activa D a S si Z o -S si !Z	Resultado = 0	S[31]	1	4
110000 000i 1111 dddddddd ssssssss	CMPS D, S	Compara signado D a S	D = S	Signo (D < S)	0	4
110001 000i 1111 dddddddd ssssssss	CMPSX D, S	Compara signado extendido D a S+C	Z & (D = S+C)	Signo (D < S+C)	0	4
110010 001i 1111 dddddddd ssssssss	ADDX D, S	Suma extendida S+C en D	Z & (D+S+C = 0)	Acarreo no signo	1	4
110011 001i 1111 dddddddd ssssssss	SUBX D, S	Resta extendida S+C de D	Z & (D-(S+C)=0)	Prestado no signo	1	4
110011 000i 1111 dddddddd ssssssss	CMPX D, S	Compara-extendida D a S+C	Z & (D = S+C)	Signo (D < S+C)	0	4
110100 001i 1111 dddddddd ssssssss	ADDS D, S	Suma signada S en D	D + S = 0	Sobreflujo signo	1	4
110101 001i 1111 dddddddd ssssssss	SUBS D, S	Resta signada S de D	D - S = 0	Sobreflujo signo	1	4
110110 001i 1111 dddddddd ssssssss	ADDSX D, S	Suma signada extendida S+C en D	Z & (D+S+C = 0)	Sobreflujo signo	1	4
110111 001i 1111 dddddddd ssssssss	SUBSX D, S	Resta signada extendida S+C de D	Z & (D-(S+C)=0)	Sobreflujo signo	1	4
111000 001i 1111 dddddddd ssssssss	CMPSUB D, S	Resta S de D si D => S	D = S	No signo (D => S)	1	4
111001 001i 1111 dddddddd ssssssss	DJNZ D, S	Dec D, salta si no es cero a S (no salto = 8 clocks)	Resultado = 0	Prestado no signo	1	4 o 8
111010 000i 1111 dddddddd ssssssss	TJNZ D, S	Prueba D, salta si no es cero a S (no salto = 8 clocks)	D = 0	0	0	4 o 8
111011 000i 1111 dddddddd ssssssss	TJZ D, S	Prueba D, salta si es cero a S (no salto = 8 clocks)	D = 0	0	0	4 o 8
111100 000i 1111 dddddddd ssssssss	WAITPEQ D, S	Espera por pins igual - (INA & S) = D	-	-	0	5+
111101 000i 1111 dddddddd ssssssss	WAITPNE D, S	Espera por pins no igual- (INA & S) != D	-	-	0	5+

iiiiii zcri cccc dddddddd ssssssss	Instrucción	Descripción	Resultado Z	Resultado C	R	Ciclos
111110 001i 1111 dddddddd ssssssss	WAITCNT D, S	Espera por CNT = D, luego suma S en D	-	Acarreo no signo	1	5+
111111 000i 1111 dddddddd ssssssss	WAITVID D, S	Espera por periférico de video para obtener D y S	-	-	0	5+
----- 0000 -----	NOP	No operaciones, solo deja pasar 4 clocks	-	-	-	4

* Ver Hub, sección 4.4 en Pág. 8.

1. ADDABS C out: si S es negativo, C = el inverso de un prestado sin signo (para D-S).
2. SUBABS C out: si S es negativo, C = el inverso de un prestado sin signo (para D+S).

6.4.1. Condiciones Ensamblador

Condición	Ejecuta instrucción
IF_ALWAYS	Siempre
IF_NEVER	Nunca
IF_E	Si es igual (Z)
IF_NE	Si no es igual (!Z)
IF_A	Si arriba (!C & !Z)
IF_B	Si debajo (C)
IF_AE	Si arriba igual (!C)
IF_BE	Si debajo igual (C Z)
IF_C	Si C Activo
IF_NC	Si C limpio
IF_Z	Si Z activo
IF_NZ	Si Z limpio
IF_C_EQ_Z	Si C igual a Z
IF_C_NE_Z	Si C no igual a Z
IF_C_AND_Z	Si C activo y Z activo
IF_C_AND_NZ	Si C activo y Z limpio
IF_NC_AND_Z	Si C limpio y Z activo
IF_NC_AND_NZ	Si C limpio y Z limpio
IF_C_OR_Z	Si C activo o Z activo
IF_C_OR_NZ	Si C activo o Z limpio
IF_NC_OR_Z	Si C limpio o Z activo
IF_NC_OR_NZ	Si C limpio o Z limpio
IF_Z_EQ_C	Si Z igual a C
IF_Z_NE_C	Si Z no igual a C
IF_Z_AND_C	Si Z activo y C activo
IF_Z_AND_NC	Si Z activo y C limpio
IF_NZ_AND_C	Si Z limpio y C activo
IF_NZ_AND_NC	Si Z limpio y C limpio
IF_Z_OR_C	Si Z activo o C activo
IF_Z_OR_NC	Si Z activo o C limpio
IF_NZ_OR_C	Si Z limpio o C activo
IF_NZ_OR_NC	Si Z limpio o C limpio

6.4.2. Directivas Ensamblador

Directiva	Descripción
FIT <i><Address></i>	Valida la instrucción/dato previo que ajusta debajo de dirección.
ORG <i><Address></i>	Ajusta en tiempo de compilación un apuntador de cog.
<i><Symbol></i> RES <i><Count></i>	Reserva el siguiente long para símbolo.

6.4.3. Efectos Ensamblador

Efecto	Resulta En
WC	Bandera C modificada
WZ	Bandera Z modificada
WR	Registro destino modificado
NR	Registro destino no modificado

6.4.4. Operadores Ensamblador

El código propeller Ensamblador puede contener expresiones constantes, las cuales pueden usar cualquier operador que este permitido en una expresión constante. la tabla (una sub tabla de la Tabla 17) enlista los operadores permitidos en Ensamblador Propeller.

Operador	Descripción
+	Suma
+	Positivo (+X); forma unaria de suma
-	Resta
-	Negación (-X); forma unaria de resta
*	Multiplica y regresa los bits mas bajos (signados)
**	Multiplica y regresa los 32 bits mas altos (signados)
/	Divide (signado)
//	Modulus (signado)
#>	Limite mínimo (signado)
<#	Limite máximo (signado)
^^	Raíz cuadrada (unario)
	Valor Absoluto (unario)
~>	Movimiento aritmético a la derecha
<	Bitwise: Decodifica valor (0-31) en un bit simple long; unario
>	Bitwise: Codifica un long en un valor (0 - 32) como bit de alta prioridad; unario
<<	Bitwise: Movimiento a la izquierda
>>	Bitwise: movimiento a la derecha
<-	Bitwise: Rota a la izquierda
->	Bitwise: Rota a la derecha
><	Bitwise: Reversa
&	Bitwise: AND
	Bitwise: OR
^	Bitwise: XOR
!	Bitwise: NOT; unario
AND	Booleano: AND (promueve un no-0 a -1)
OR	Booleano: OR (promueve no-0 a -1)
NOT	Booleano: NOT (promueve no-0 a -1); unario
==	Booleano: Es igual
<>	Booleano: No es igual
<	Booleano: Menor que (signado)
>	Booleano: Mayor que (signado)
=<	Booleano: Igual o menor (signado)
=>	Booleano: Igual o mayor (signado)
e	Símbolo dirección, unario

8.0 CARACTERISTICAS ELECTRICAS

8.1. Rangos Máximos Absolutos

Exceder los rangos absolutos máximos ocasiona daño permanente al equipo. Estos son los valores absolutos del equipo. La operación funcional del diseño no esta implicada en estas u otras condiciones de exceso dados en la sección remanente 7.0. La exposición a los rangos máximos absolutos por periodos extendidos puede afectar la confiabilidad del equipo.

Tabla 18: Rangos Máximos Absolutos	
Temperatura ambiente en condiciones normales	-55 °C a +125 °C
Temperatura de almacenaje	-65 °C a +150 °C
Voltaje en V_{dd} respecto a V_{ss}	-0.3 V a +4.0 V
Voltaje en todos los otros pins respecto a V_{ss}	-0.3 V a ($V_{dd} + 0.3$ V)
Disipación de potencia total	1 W
Max. corriente de salida de pins V_{ss}	300 mA
Max. corriente en pins V_{dd}	300 mA
Max. corriente DC en un pin de entrada con protección interna	±500 μ A
Max. corriente permitida por pin E/S	40 mA
ESD (Modelo de Cuerpo Humano) pin de entrega	3 kV
ESD (Modelo de Cuerpo Humano) todos los pins de no entrega	8 kV

8.2. Características DC

(Rango de temperatura de operación: $-55^{\circ}\text{C} < T_a < +125^{\circ}\text{C}$ a menos que se especifique otra cosa)

Símbolo	Parámetro	Condiciones	Min	Típico	Max	Unidades
V_{dd}	Voltaje		2.7	-	3.6	V
V_{ih}, V_{il}	Alto Lógico Bajo Lógico		$0.6 V_{dd}$ V_{ss}		V_{dd} $0.3 V_{dd}$	V V
I_{il}	Corriente de entrada	$V_{in} = V_{dd}$ o V_{ss}	-1.0		+1.0	μ A
V_{oh}	Voltaje de salida alto	$I_{oh} = 10$ mA, $V_{dd} = 3.3$ V	2.85			V
V_{ol}	Voltaje de salida bajo	$I_{ol} = 10$ mA, $V_{dd} = 3.3$ V			0.4	V
I_{Bo}	Detector de interrupción de corriente			3.8		μ A
I	Corriente inactiva	RESn = 0V, BOEn = V_{dd} , P ₀ -P ₃₁ =0V		600		nA

Note: Datos en la columna ("Típico") es $T_a = 25^{\circ}\text{C}$ a menos que se especifique otra cosa.

8.3. Características AC

(Rango de temperatura de operación: $-55^{\circ}\text{C} < T_a < +125^{\circ}\text{C}$ a menos que se especifique otra cosa)

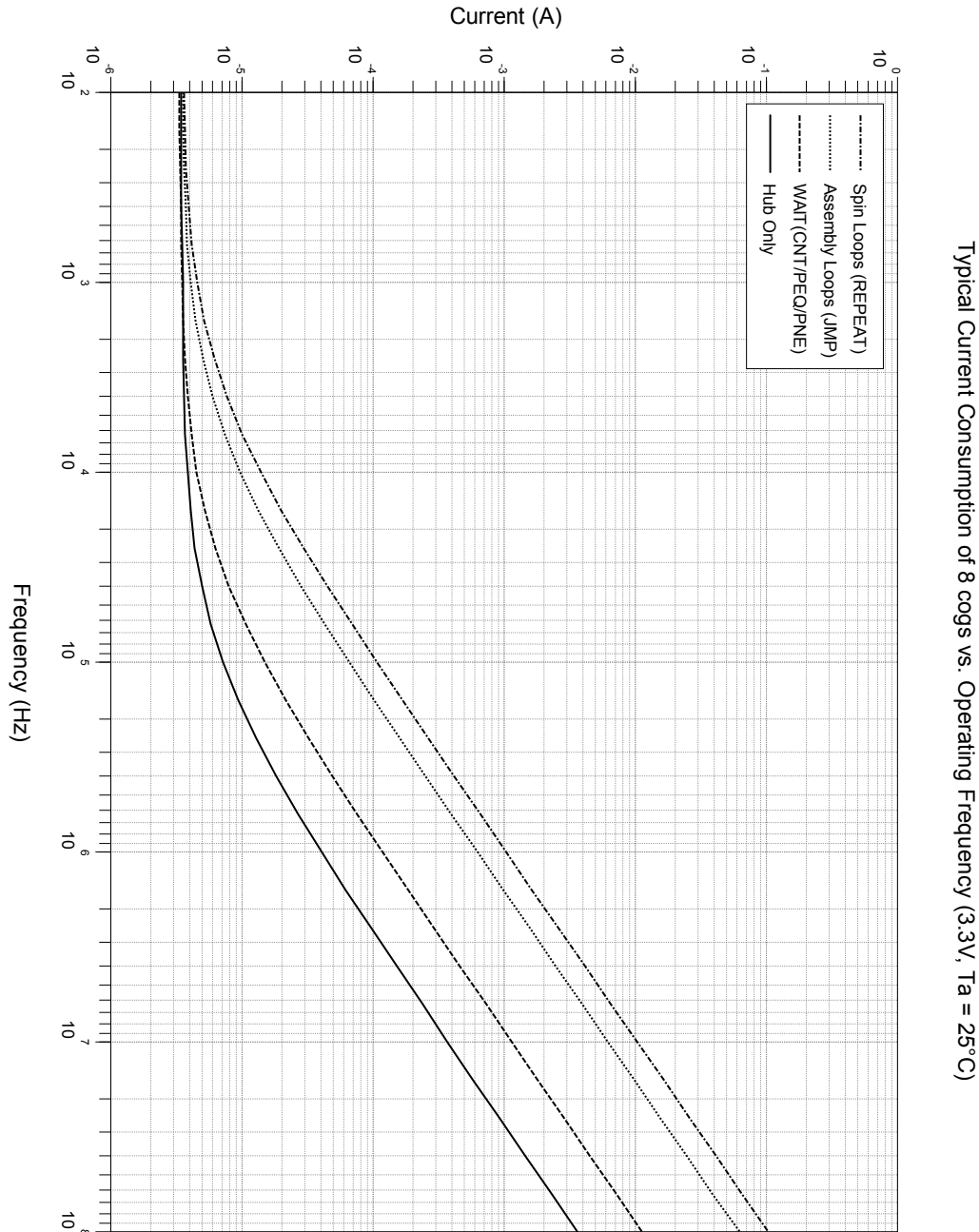
Símbolo	Parámetro	Min	Típico	Max	Unidades	Condición
F_{osc}	Frecuencia Externa X1	DC	-	80	MHz	
	Frecuencia del Oscilador	DC 13 8 4	- 20 12 -	80 33 20 8	MHz kHz MHz MHz	Manejo directo (no PLL) RCSLOW RCFAST Cristal usando PLL
C_{in}	Capacitancia de Entrada		6	-	pF	

Note: Datos en la columna ("Típico") es $T_a = 25^{\circ}\text{C}$ a menos que se especifique otra cosa.

9.0 CARACTERISTICAS DE CONSUMO DE CORRIENTE

9.1 Consumo Típico de Corriente de 8 Cogs

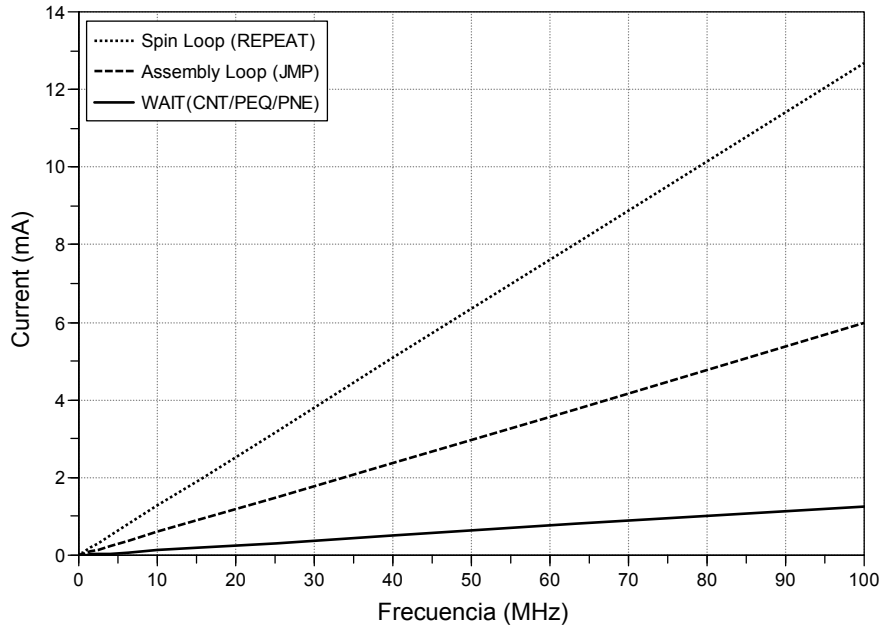
Esta figura muestra el consumo típico de corriente del Propeller bajo diversas condiciones duplicadas con todos los cogs. Circuito detector de interrupción y el Ciclo de Fase Cerrada donde se deshabilita para la duración de la prueba. EL consumo de corriente es substancialmente constante en el rango operacional de temperatura.



9.2. Corriente Típica de un Cog vs. Frecuencia de Operación

Esta grafica muestra el consumo típico de corriente de un cog bajo diversas condiciones, separado de otras fuentes de corriente en el chip Propeller.

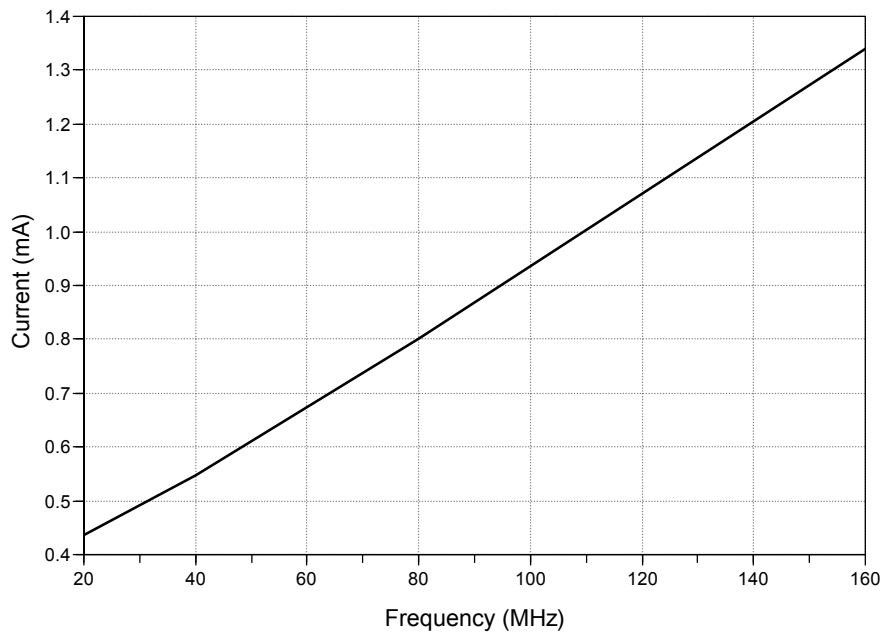
Typical Current of a Cog vs. Operating Frequency (Vdd = 3.3 V, Ta = 25° C)



9.3. Corriente Típica PLL vs. Frecuencia VCO

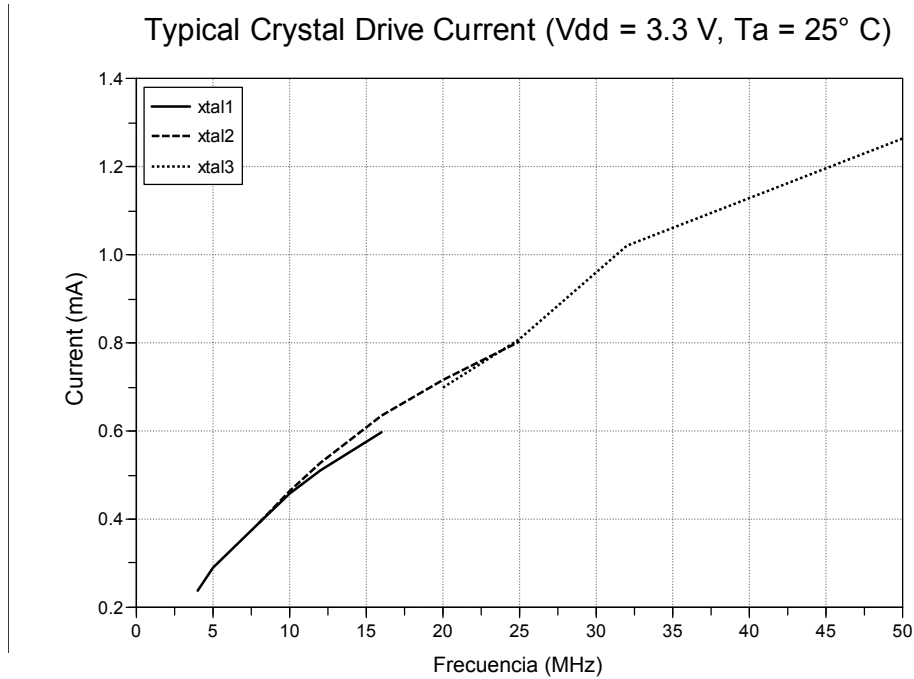
Esta grafica muestra el monto de corriente consumido por un Ciclo de Fase Cerrada como una función de la frecuencia del Oscilador de Voltaje Controlado que es 16 veces la frecuencia de la entrada de reloj.

Typical PLL Current vs. VCO Frequency (Vdd = 3.3 V, Ta = 25° C)



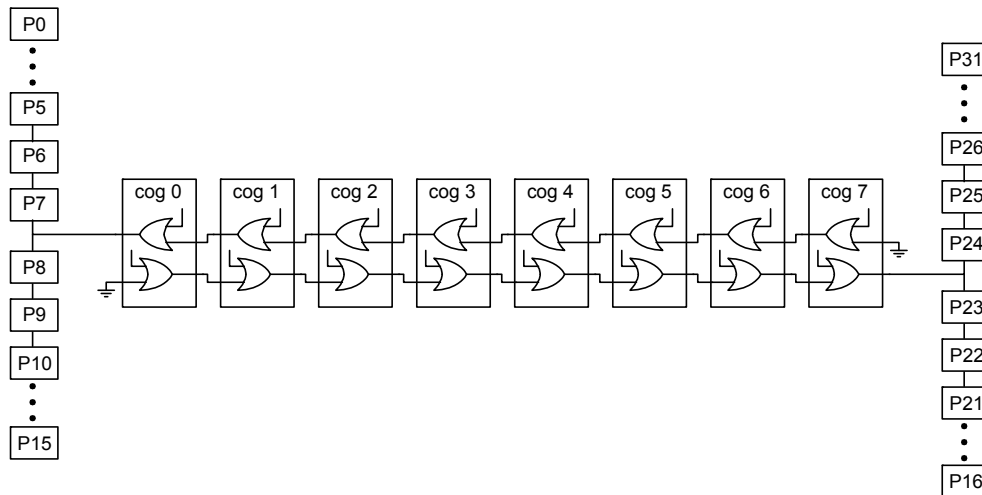
9.4. Corriente Típica del Controlador del Cristal

Esta grafica muestra el consumo de corriente del controlador del cristal sobre un rango de frecuencias y parámetros del cristal, todos los puntos de datos sobre 25 MHz se obtuvieron usando un resonador ya que el controlador no opera en la 3ª armónica requerido para controlar cristales por encima de 25 MHz.



9.5. Relación del Cog y Pin E/S

La figura muestra la relación física entre los cogs y los pins E/S. Puede existir un retraso de propagación de 1 a 1.5 ns en las transiciones de salida entre en camino mas corto y el mas largo, el propósito de la figura es ilustrar la longitud de los pins y de su capacitancia parasita asociada. Esta capacitancia incrementa el monto de energía requerido para la transición del estado de un pin y por lo tanto incrementa la corriente necesaria para cambiar un pin. Así la corriente consumida por el Cog 7 al cambiar P0 a 20 MHz será mayor que el Cog 0 cambiando P7 a 20 MHz. El monto de corriente consumida por la transición del estado del pin depende de muchos factores incluyendo temperatura, frecuencia de transiciones, carga externa y carga interna. Como se menciona, la carga interna depende de cual cog y pin se usan. La corriente de carga interna en temperatura ambiente cambiando un pin a 20 MHz para un Propeller en empaque DIP varia en el orden de los 300 µA.



9.6. Perfil de Corriente en diversas condiciones de inicio

Los diagramas muestran el perfil de diversas condiciones de arranque del chip Propeller dependiendo de la presencia de una EEPROM y PC.

Figura 9

Secuencia de arranque sin PC y sin EEPROM (P31 esta bajo y P29 no conectado (igual que estar bajo))

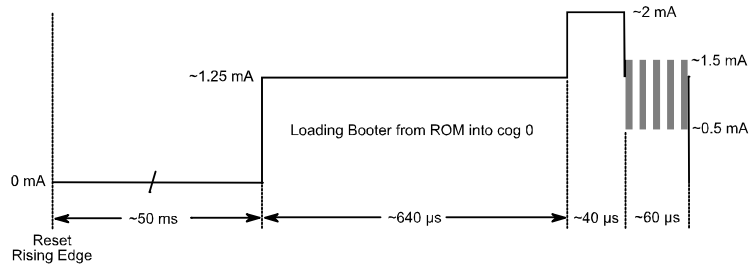


Figura 10

Secuencia de arranque con PC (Conectada pero inactiva) y no EEPROM. (P31 esta en alto y P29 no conectado).

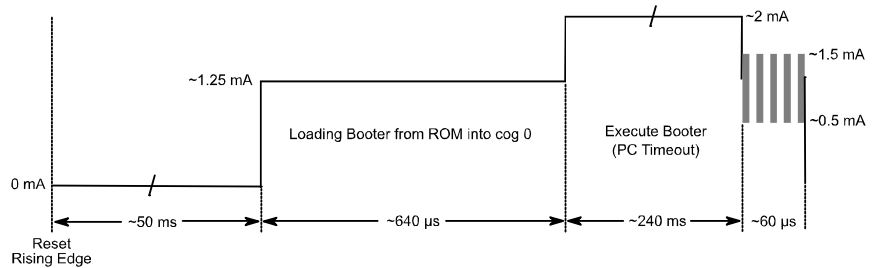


Figura 11

Secuencia de arranque sin PC y sin EEPROM (P31 en bajo y P29 en alto).

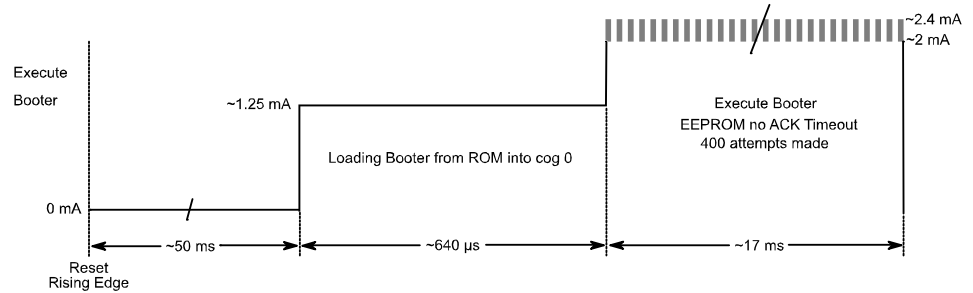


Figura 12

Secuencia de arranque sin PC con EEPROM (P31 en bajo y P29 conectado a EEPROM SDA).

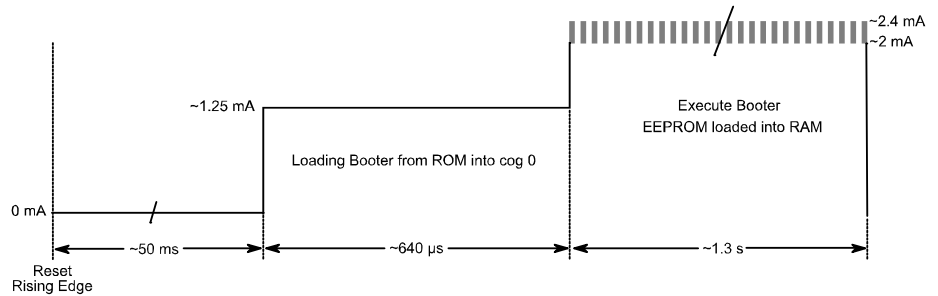
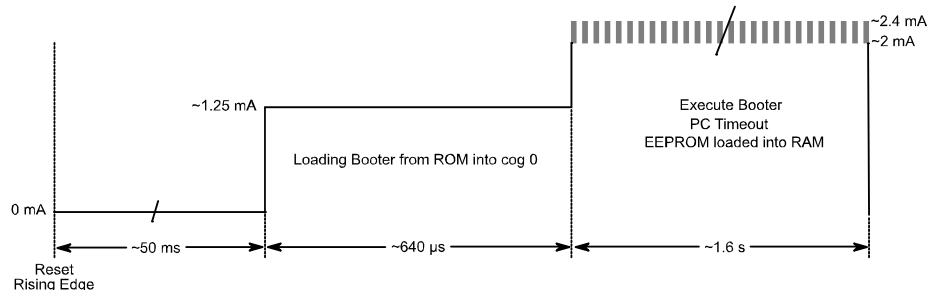


Figura 13

Secuencia de arranque con PC (conectada pero inactiva) con EEPROM (P31 en alto y P29 conectado a EEPROM SDA).

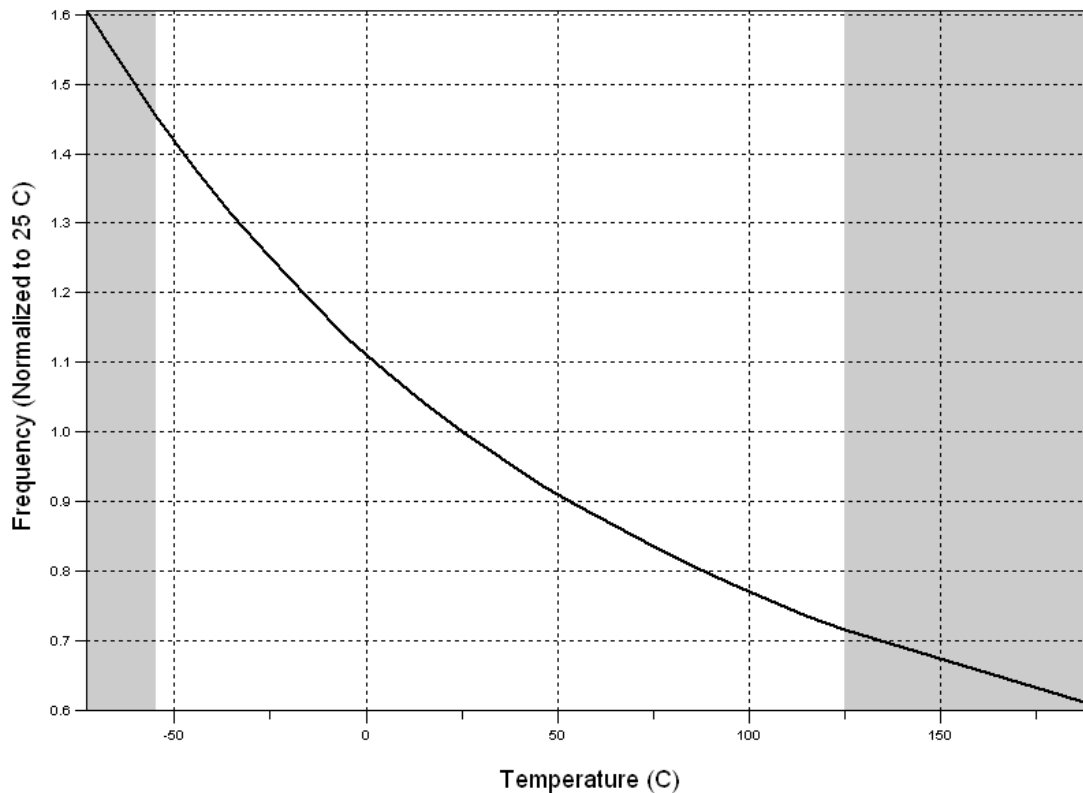


10.0 CARACTERISTICAS DE TEMPERATURA

10.1. Frecuencia de Oscilador Interno como función de la Temperatura

La frecuencia del oscilador interno es variable debido a la variación de proceso, el rango de cambio como función de la temperatura cuando se normaliza proporciona un chip sin rango de variación el cual se puede usar para calcular la frecuencia de oscilación cuando la temperatura ambiente es diferente a 25°C (la temperatura a la cual se normalizo la grafica). La frecuencia absoluta a 25°C varía de 13.26 a 13.75 MHz en el grupo de muestra. La sección de la grafica que esta en blanco es el rango militar de temperatura, la sección en gris representa datos que van más allá de la especificación militar de temperatura.

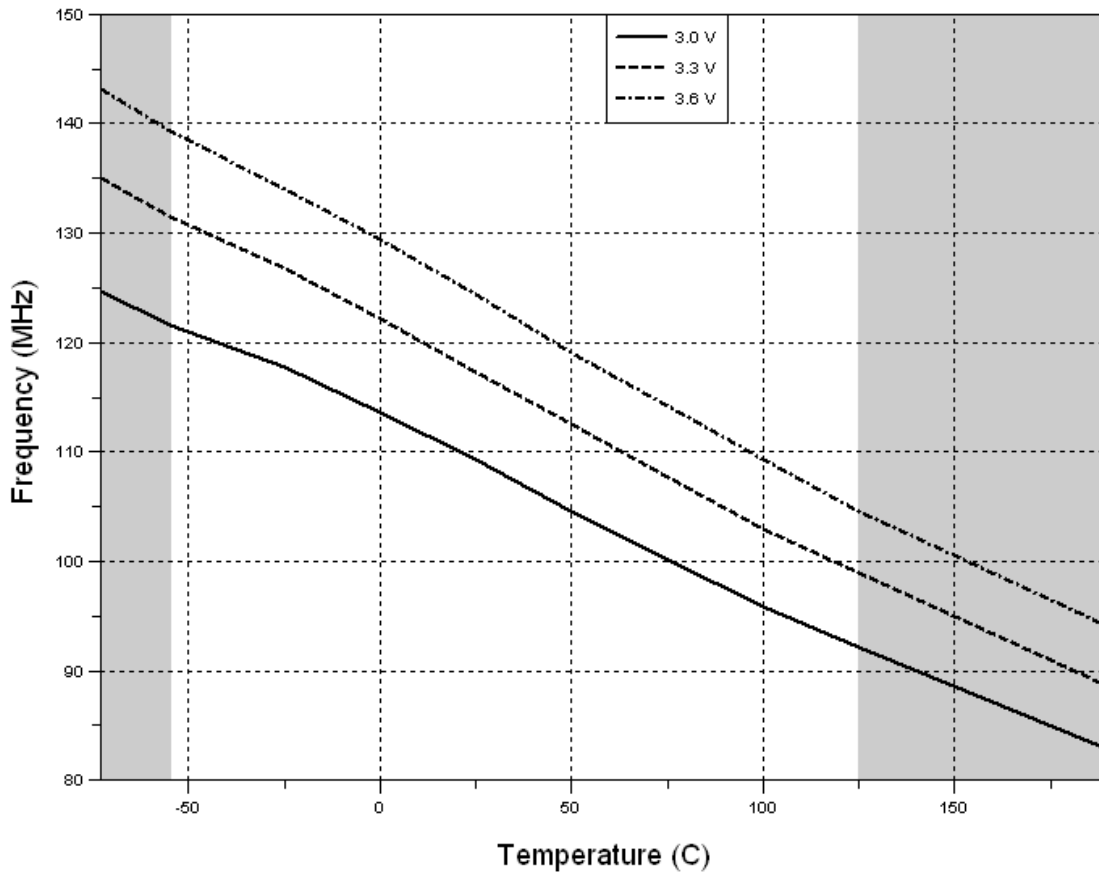
RCAFAST Normalized Frequency vs Temperature



10.2. La Frecuencia de Operación mas Rápida como función de la Temperatura

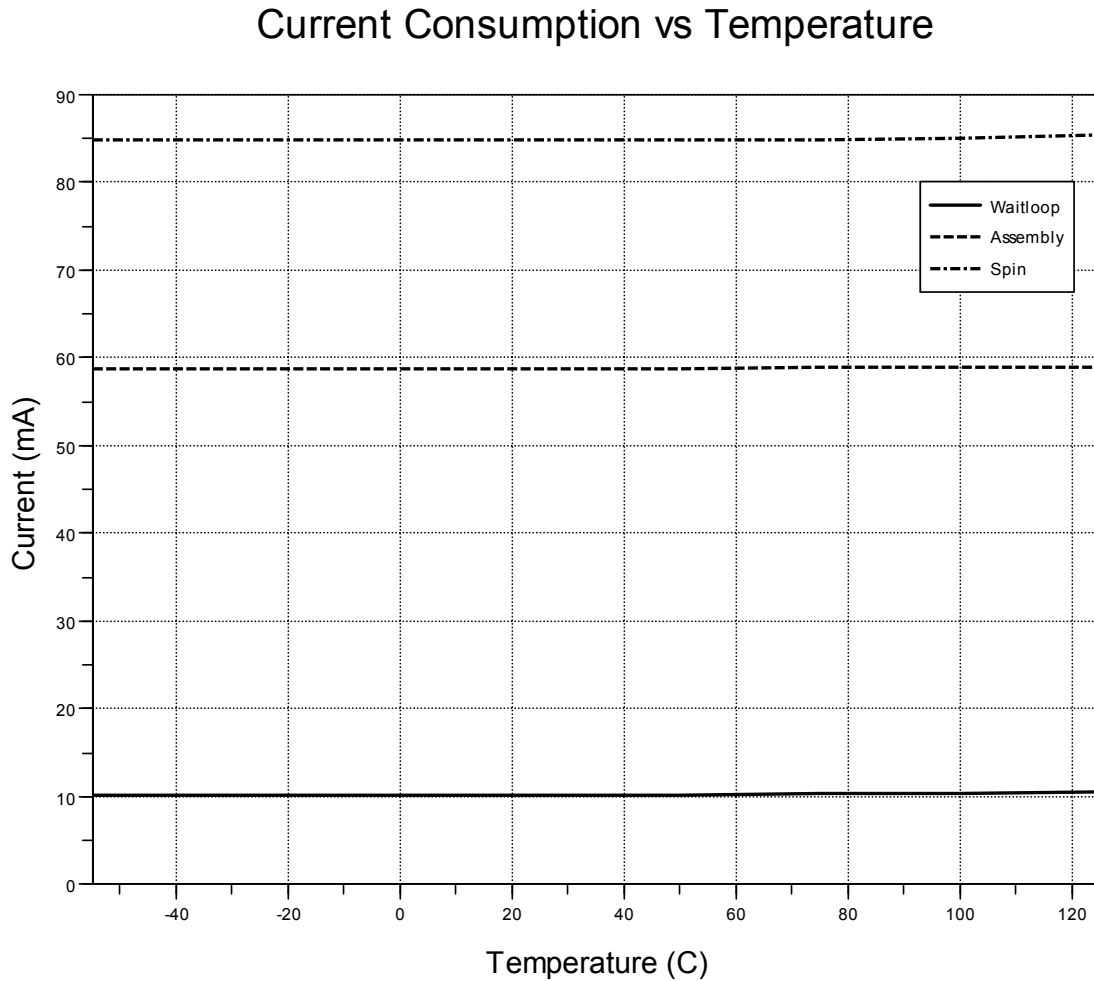
La siguiente grafica representa una muestra pequeña del promedio del rango de operación más rápido del chip propeller. La prueba se desarrollo en una cámara de presión de aire usando código de ejecución en todos los cogs, generadores múltiples de video y módulos contadores. Una frecuencia se consideraba exitosa si el demo corría sin problemas por un minuto. Las curvas representan una prueba de procedimiento agresivo (promediado, fuerza de aire tiempo límite de un minuto); por lo tanto el diseñador debe reducir la curva para llegar a una frecuencia estable para una aplicación particular. Nuevamente la parte gris representa temperaturas más allá de los rangos militares de temperatura.

Fastest Frequency vs Temperature



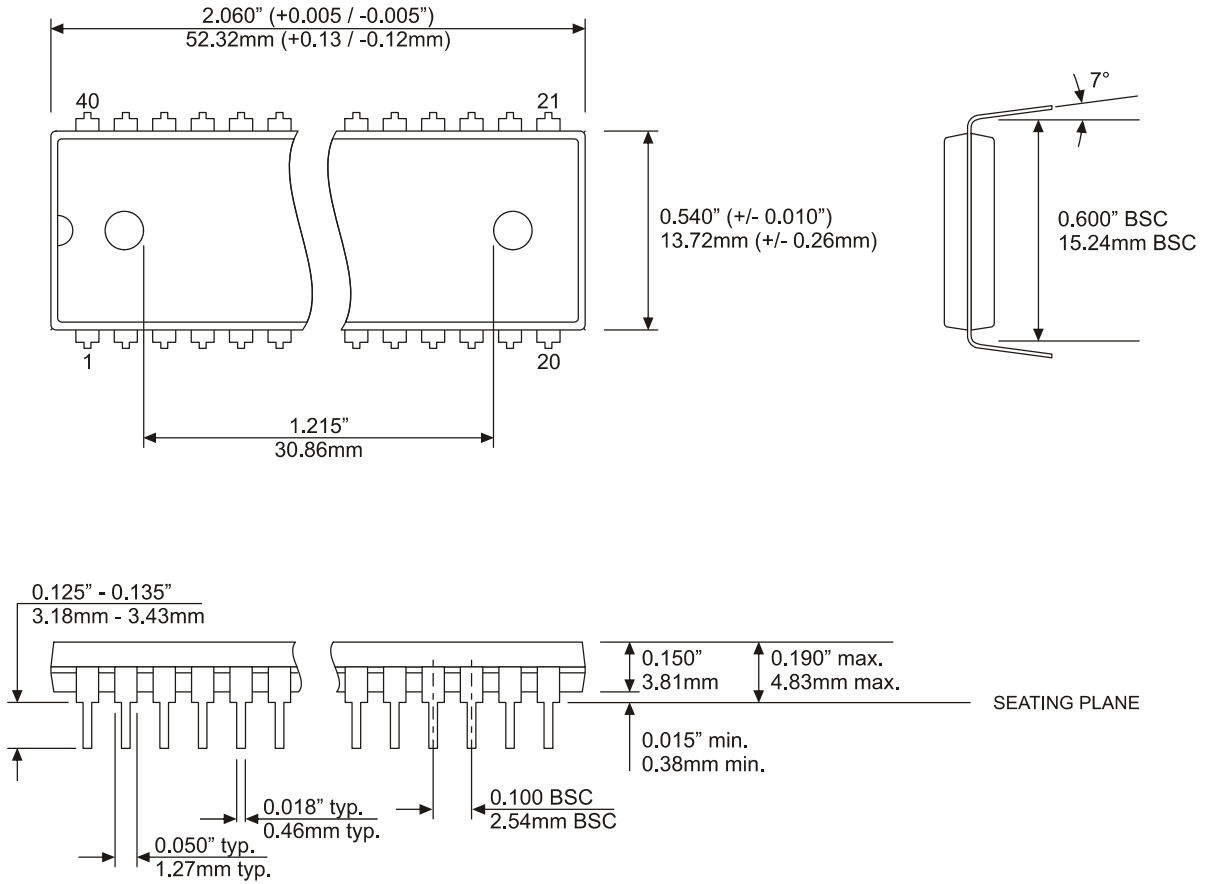
10.3. Consumo de Corriente como Función de Temperatura

La siguiente grafica demuestra el consumo de corriente del Propeller en función de la temperatura. Es claro de la grafica que la corriente es casi independiente de la temperatura en todo el rango militar de temperatura.

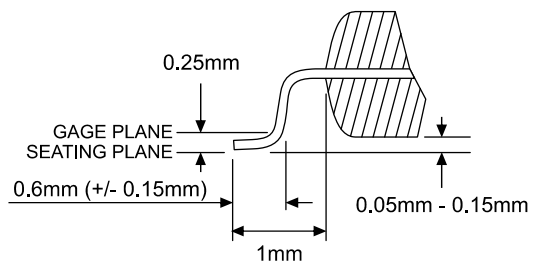
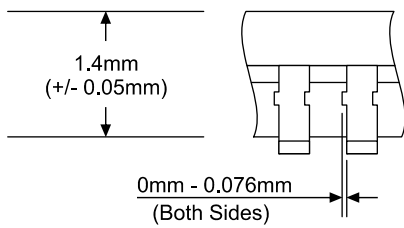
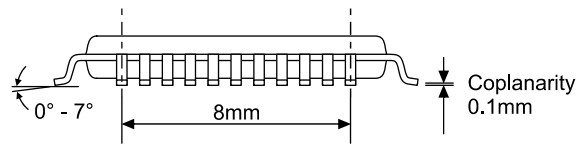
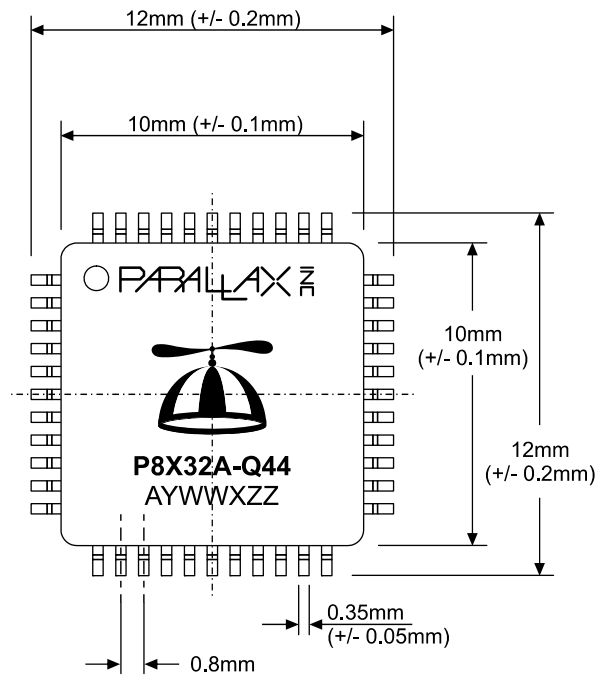
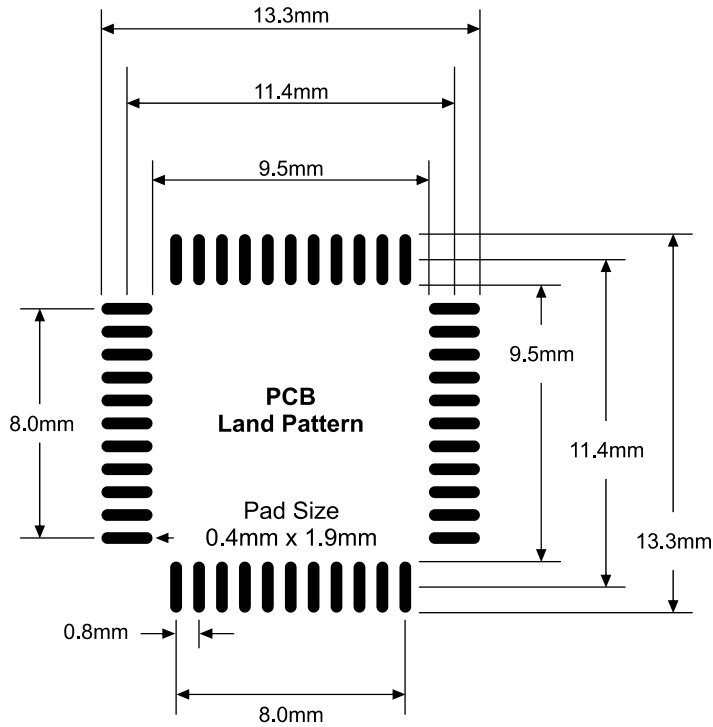


11.0 DIMENSIONES DE EMPAQUE

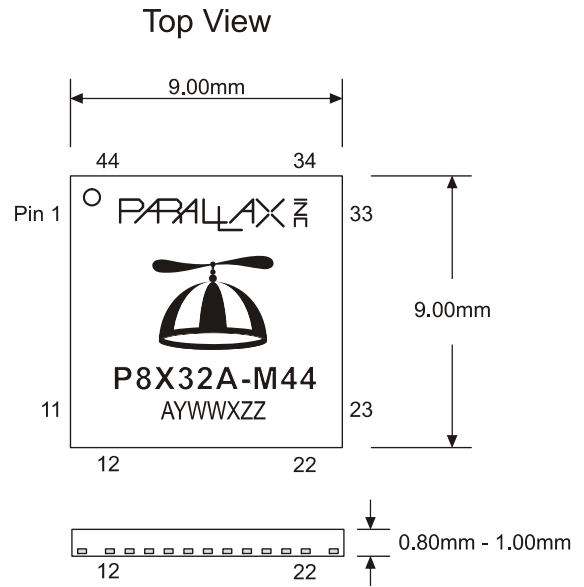
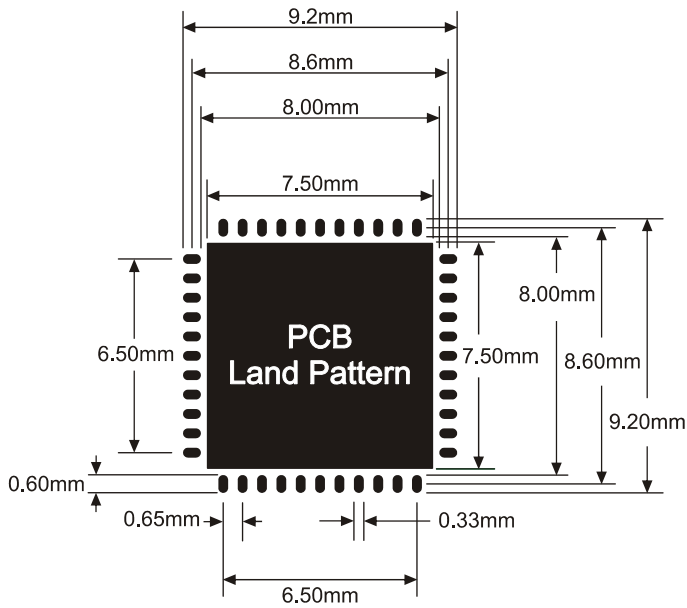
11.1. P8X32A-D40 (DIP 40-pin)



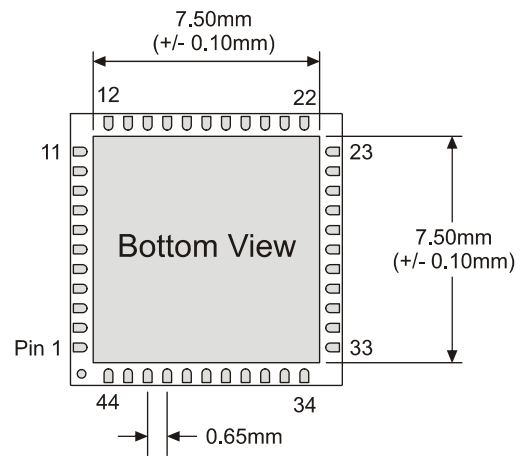
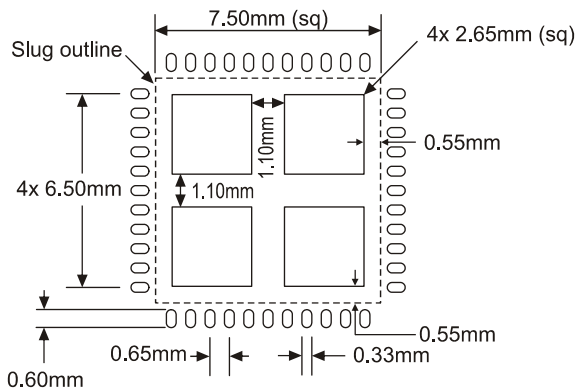
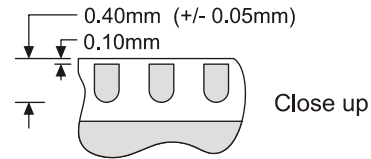
11.2. P8X32A-Q44 (LQFP 44-pin)



11.3. P8X32A-M44 (QFN 44-pin)



Stencil Pattern
 Drawing is larger than actual size
 Typical shrink is 0.05mm



12.0 INFORMACION DE FABRICACION

12.1. Pico de Temperatura de Reflujo

Tipo de Empaque	Temp. Pico de Reflujo
DIP	255+5/-0 °C
LQFP	255+5/-0 °C
QFN	255+5/-0 °C

12.2. Conformidad en Green/RoHS

Todos los modelos de chip Propeller Parallax están certificados y cumplen con las normas de Green/RoHS. El certificado de conformidad esta disponible bajo petición y puede obtenerse contactando al equipo de ventas de Parallax.

13.0 HISTORIA DE REVISIONES

13.1.1. Cambios a la Versión 1.1:

Sección 11.3: P8X32A-M44 (QFN 44-pin). Imagen reemplazada para agregar el patrón del estencil. Nueva sección integrada: 4.8 Etapas de Ejecución en Instrucciones Ensamblador. Información de Contacto actualizada.

13.1.2. Cambios a la Versión 1.2:

Sección 6.4: Modificada la tabla de entradas para ADD, ADDABS, ADDS, ADDSX, ADDX, CMP, CMPS, CMPSX, CMPX, COGID, COGINIT, COGSTOP, LOCKCLR, LCKNEW, LOCKRET, LOCKSET, MAX, MAXS, MIN, MINS, SUB, SUBABS, SUBS, SUBSX, SUBX, SUMC, SUMNC, SUMNZ, SUMZ, TEST, TJNZ, TJZ. Sección 4.5 actualizada. Sección 5.1: nueva instrucción agregada al final del párrafo. Sección 5.2: nueva instrucción agregada al final del primer párrafo.

Parallax. Información de Ventas y Soporte Técnico

Para la más reciente información de los chips Propeller y herramientas de programación, tarjetas de desarrollo, materiales educativos y ejemplos de aplicaciones por favor visite www.parallax.com/propeller.

Parallax, Inc.

599 Menlo Drive

Rocklin, CA 95765

USA

Teléfono: (916) 624-8333

Fax: (916) 624-8003

Ventas: 1-888-512-1024

Soporte Técnico: 1-888-997-8267

Ventas: sales@parallax.com

Soporte Técnico: support@parallax.com

Web: <http://forums.parallax.com>

Foros: <http://www.parallax.com/Support/DiscussionForums/tabid/392/Default.aspx>

Foros en Español: <http://espanol.groups.yahoo.com/group/ParallaxenEspanol/>

Intercambio de Objetos: <http://obex.parallax.com>