

Propeller™ P8X32A 数据表

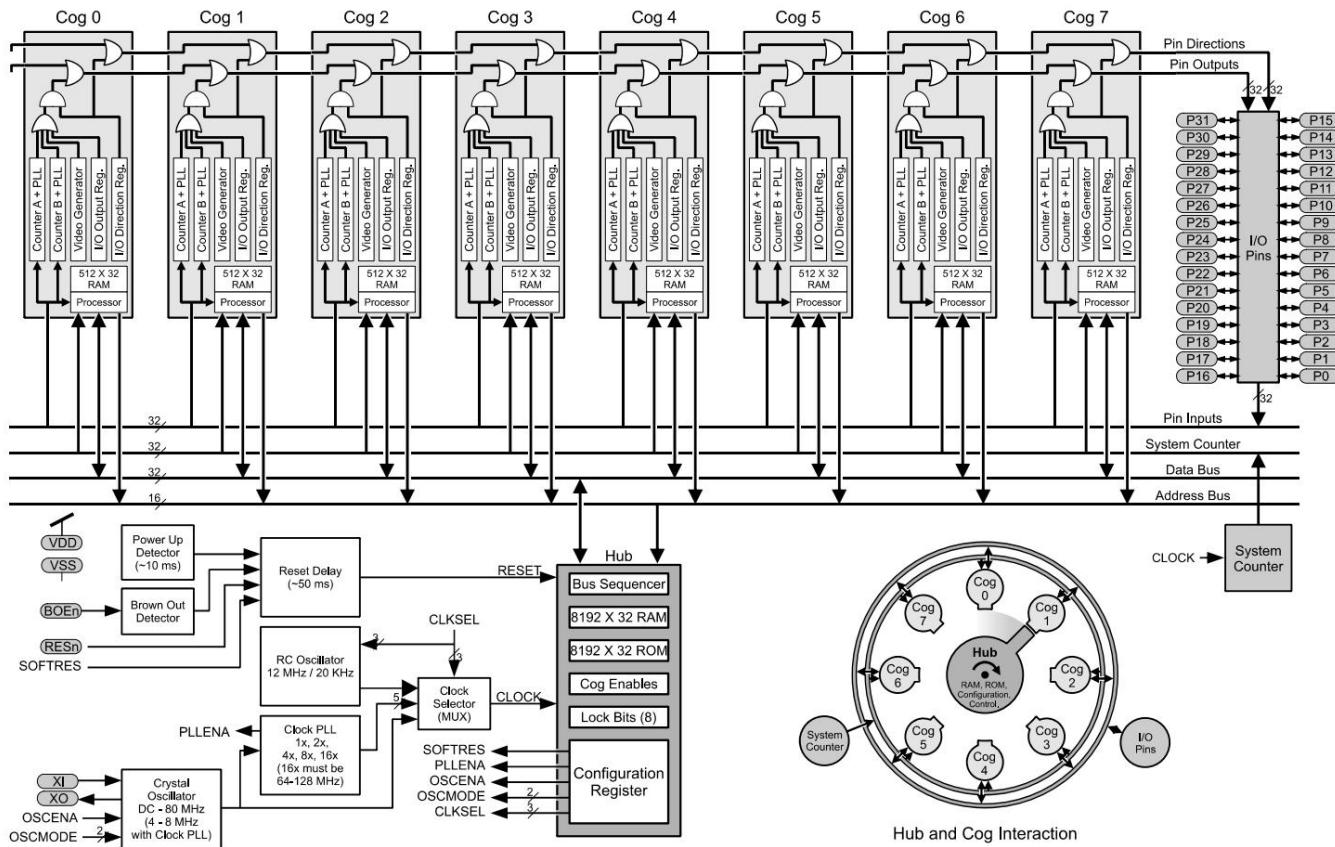
8-Cog 多处理器微控制器

1.0 产品概述

1.1. 简介

Propeller 芯片旨在为嵌入式系统提供高速处理能力,同时保持低电流消耗和较小的物理占用空间。除了速度快之外,Propeller 芯片还通过其八个处理器 (称为 cogs) 提供灵活性和强大功能,这些处理器可以独立或协作地执行同时执行的任务,同时保持相对简单的架构,易于学习和使用。有两种编程语言可供选择:Spin (一种高级基于对象的语言) 和 Propeller Assembly。两者都包含自定义命令,可轻松管理 Propeller 芯片的独特功能。

图 1:螺旋桨 P8X32A 框图



1.2. 股票代码

表 1:螺旋桨芯片库存代码

设备 库存 #	封装类型	输入/输出 别针	力量 要求	外部的 钟 速度	内部 RC 振荡器	内部的 执行 速度	全球的 只读存储器/随机存取存储器	齿轮内存
P8X32A-D40 40 针 DIP		三十二		3.3伏直流电	直流至 80 MHz	0 至 160 MIPS (20 MIPS/齿轮)	64 K 字节; 32768 字节 ROM / 32768 字节 RAM	
P8X32A-Q44 44 引脚 QFP								
P8X32A-M44 44 引脚 QFN								512 x 32 位 每个齿轮

*近似值;范围分别可能为 8 MHz – 20 MHz 或 13 kHz – 33 kHz。

目录

1.0	产品概述.....	1
	简介.....	1
	股票代号.....	1
1.1.	主要特点和优点.....	3
1.1.1.	32 位多核架构.....	3
1.2.	时钟系统和等待指令.....	3
1.3.	编程语言和资源.....	3
1.3.1.	灵活的 I/O 和外设接口.....	3
	应用.....	3
1.3.2. 1.3.3. 1.3.4. 1.4.	14 和性能支持.....	3
2.0	连接图.....	4
	引脚分配.....	4
	引脚说明.....	4
2.1.	典型连接图.....	5
2.2. 2.3. 2.3.1. 2.3.2.	螺旋桨夹或螺旋桨插头连接 - 推荐.....	5
2.2. 2.3. 2.3.1. 2.3.2.	2.3.2. 行端口连接.....	5
3.0	操作程序.....	6
3.1.	启动程序.....	6
3.2.	运行时程序.....	6
3.3.	关机程序.....	6
4.0	系统组织.....	6
	共享资源.....	6
	系统时钟.....	6
	齿轮 (处理器)	7
	集线器.....	7
	输入/输出引脚.....	8
	系统计数器.....	8
	锁.....	8
	汇编指令执行阶段.....	9
	齿轮计数器.....	10
4.1.	CTRA / CTRB - 控制寄存器.....	10
4.2.	FRQA / FROB - 频率寄存器.....	10
4.3.	PHSA / PHSB - 相位寄存器.....	10
4.4.	视频生成器	11
4.5.	VCFG - 视频配置寄存器.....	11
4.6.	VSCL - 视频比例寄存器.....	12
4.7.	WAITVID 命令/指令.....	12
4.8. 4.9. 4.9.1. 4.9.2. 4.9.3. 4.10. 4.10.1. 4.10.2. 4.1.3. 0.4.11.	4.11. CLN 寄存器.....	14
5.0	内存组织.....	15
5.1.	主存储器.....	15
5.1.1.	主 RAM.....	15
5.1.2.	字符定义.....	15
5.1.3. 5.1.4.	数学函数表.....	16
5.2.	齿轮 RAM.....	16
6.0	编程语言.....	17
	保留字列表.....	17
6.1.	为将来使用而保留的字.....	17
	数学和逻辑运算符.....	18
6.1.1.	Spin 语言汇总表	19
6.1.1.1.	常量.....	21
6.2.	螺旋桨装配说明表.....	22
6.3.	装配条件.....	24
6.3.1.	汇编指令.....	24
6.4.	组装效果.....	24
6.4.1. 6.4.2. 6.4.3. 6.4.4.	6.4.4. 螺旋桨脚.....	24
7.0	电气特性.....	25
7.1.	绝对最大额定值.....	25
7.2.	直流特性.....	25
7.3.	交流特性.....	25
8.0	电流消耗特性.....	26
	8 个 Cogs 的典型电流消耗.....	26
	Cog 的典型电流与工作频率.....	27
	典型 PLL 电流与 VCO 频率的关系.....	27
	典型晶体驱动电流.....	28
	Cog 和 I/O 引脚关系.....	28
8.1. 8.2. 8.3.	8.3. 各种启动条件下的电流分布.....	29
9.0	温度特性.....	30
9.1.	内部振荡器频率与温度的关系.....	30
9.2.	最快工作频率与温度的关系.....	31
9.3.	电流消耗与温度的关系.....	32
10.0	封装尺寸.....	33
10.1.	P8X32A-D40 (40 针 DIP).....	33
10.2.	P8X32A-Q44 (44 引脚 LQFP).....	34
10.3.	P8X32A-M44 (44 针 QFN)	35
11.0	制造信息.....	36
11.1.	回流峰值温度.....	36
11.2.	绿色/RoHS 合规性	36
12.0	修订历史.....	36
12.1.1.	1.1 版的变更内容:	36
12.1.2.	1.2 版变更内容:	36
12.1.3.	1.3 版变更.....	36
12.1.4.	1.4 版变更.....	36

1.3. 主要特性和优点 P8X32A 的设计使开发人员摆脱了嵌入式系统编程的常见复杂性。

1.3.1. 32 位多核架构

- 一个微控制器中配备八个对称 32 位处理器 (cog), 实现真正的并行处理
- 多 cog 运行时管理 (运行/等待/停止) 可轻松解决事件处理问题, 无需中断。这大大简化了异步和同步事件的编程, 从而实现响应迅速且易于维护的应用程序。
- 每个齿轮 20 MIPS, 所有齿轮总计 160 MIPS 跑步
- 解决混合带宽需求
- 嵌入式应用 · 多用途设计减少了零件数量, 同时提高系统能力
- 开发人员驱动的齿轮分配为嵌入式应用程序带来灵活的响应和确定性的时间

1.3.2. 时钟系统和等待指令

- 灵活的时钟模式
 - 两个内置、一个外置, 以及可选的 1x-16x PLL; 高达 80 MHz 的系统时钟
 - 可在运行时在代码中切换; 低频适用于低功耗时段, 高频适用于高带宽时刻
- 共享系统时钟促进齿轮之间的同步 · WAIT 指令
 - 提供强大的同步 / 异步事件管理
 - 将专用事件齿轮设置为“始终就绪”、极低功耗状态

1.3.3. 编程语言和资源

- Spin (基于对象, 高级) 和 Assembly (PASM; 低级); 一起使用以进行全面开发, 即在 Spin 中快速开发, 并使用预先编写的高速 PASM 驱动程序快速执行
- 第三方支持: C、BASIC 等 · 增强型汇编语言
 - 单独指令的条件执行; 实现无抖动信号生成和事件处理
 - 单独指令的可选标志和结果写入
 - 说明 · 开源对象
 - 对象可通过 Propeller 对象自由共享
 - Exchange 和 Propeller 工具库
 - 选择适合需要的对象, 轻松将它们集成到 Propeller 应用程序中

1.3.4. 灵活的 I/O 和外设接口

- 32 个 I/O 引脚
 - 启动后所有 I/O 均为通用 I/O; 每个 cog 可同时访问
 - o 单指令访问任何单个 I/O 引脚或任何连续的 I/O 引脚组
 - o 可轻松在引脚之间移动设计功能, 以实现

简单的系统板布局 · 多功能计数器

- o 可配置状态机在每个时钟周期生成或检测重复信号
- o 测量频率、检测边缘、计数周期,

D/A 或 A/D 转换等 o 自主操作, 可选运行时间

监控和调整

- o 每个齿轮两个计数器
- 视频发生器
- o RGB:VGA; 8 个 I/O 引脚
- o 复合:NTSC、PAL; 1 引脚 (黑白) 、3 引脚
 - (典型) , 或 4 针 (可选) o
 - 每个齿轮一个发电机
- 软件外设
- o 用软件和廉价无源元件构建的外设接口; 而不是单一功能的片上硬件

- o 基于软件的接口非常灵活; 随着外围设备需求的增加而增强
- 无需使用芯片变体重新设计

1.4. 应用 P8X32A 在可通过同时处理而大大简化的项目中特别有用, 包括:

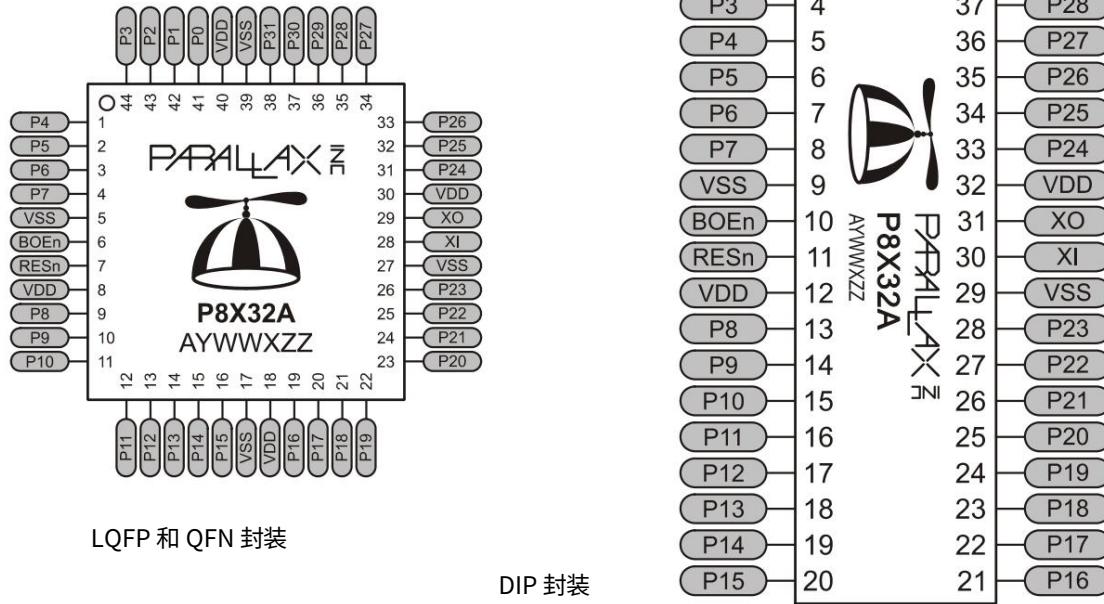
- 工业控制系统 · 传感器集成、信号处理
- 和数据采集 · 手持式便携式人机界面终端 · 电机和执行器控制 · 需要 NTSC、PAL 或 VGA 输入
- 出的用户界面, 带有 PS/2 键盘和鼠标输入 · 低成本视频游戏系统 · 工业、教育或个人用途的机器人 · 无线视频传输 (NTSC 或 PAL)

1.4.1. 企业和社区支持

- 销售或技术支持: (916) 632-4664 · 电子邮件销售: sales@parallaxsemiconductor.com · 电子邮件支持: support@parallaxsemiconductor.com · 工程师主持的 Parallax Semiconductor 子论坛位于 <http://forums.parallax.com> · Parallax 托管的 Propeller Object Exchange 库: <http://obex.parallax.com>

2.0 连接图

2.1. 引脚分配



2.2. 引脚说明

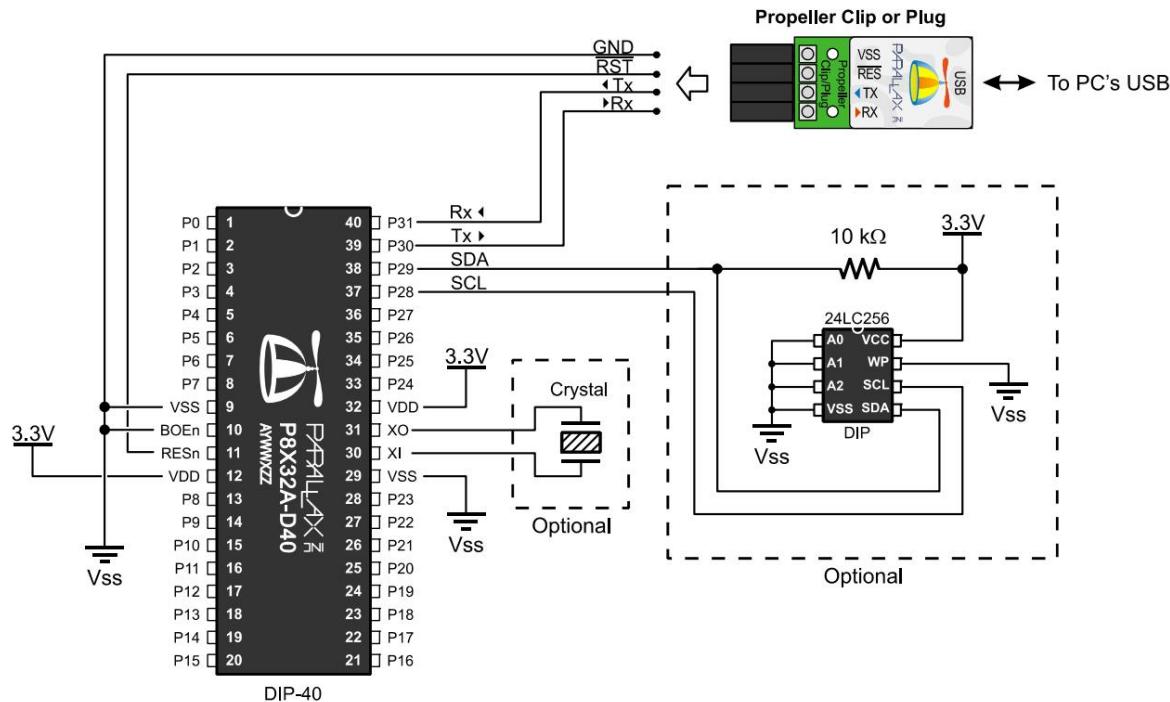
表 2:引脚说明

引脚名称	方向	描述
P0 – P31	输入/输出	<p>通用 I/O 端口 A。可在 3.3 VDC 下分别提供/吸收 40 mA 电流。CMOS 电平逻辑, 阈值为 $\approx \frac{1}{2} VDD$ 或 1.6 VDC @ 3.3 VDC。</p> <p>下面显示的引脚在上电/复位时有特殊用途, 但之后成为通用 I/O。</p> <p>P28 I2C SCL 连接到可选的外部 EEPROM。 P29 I2C SDA 连接至可选的外部 EEPROM。 P30 串行 Tx 至主机。 P31 来自主机的串行 Rx。</p>
电压源	---	3.3 伏电源 (2.7 – 3.6 VDC)
虚拟安全系统	---	地面
伯恩	.	掉电使能 (低电平有效)。必须连接到 VDD 或 VSS。如果为低电平, RESn 会变为弱输出 (通过 $5 k\Omega$ 提供 VDD) 以用于监控, 但仍可驱动至低电平以导致复位。如果为高电平, RESn 是带有施密特触发器的 CMOS 输入。
稀土元素	输入/输出	重置 (低电平有效)。当为低电平时, 重置 Propeller 芯片, 所有齿轮禁用, I/O 引脚浮动。 RESn 由低变高 50 毫秒后螺旋桨重新启动。
+	.	晶体输入。可根据 CLK 寄存器设置连接到晶体/振荡器组的输出 (XO 保持断开状态), 或连接到晶体的一条支路 (XO 连接到晶体或谐振器的另一条支路)。无需外部电阻器或电容器。
酒类	哦	晶体输出。为外部晶体提供反馈, 或者根据 CLK 寄存器设置保持断开。无需外部电阻器或电容器。

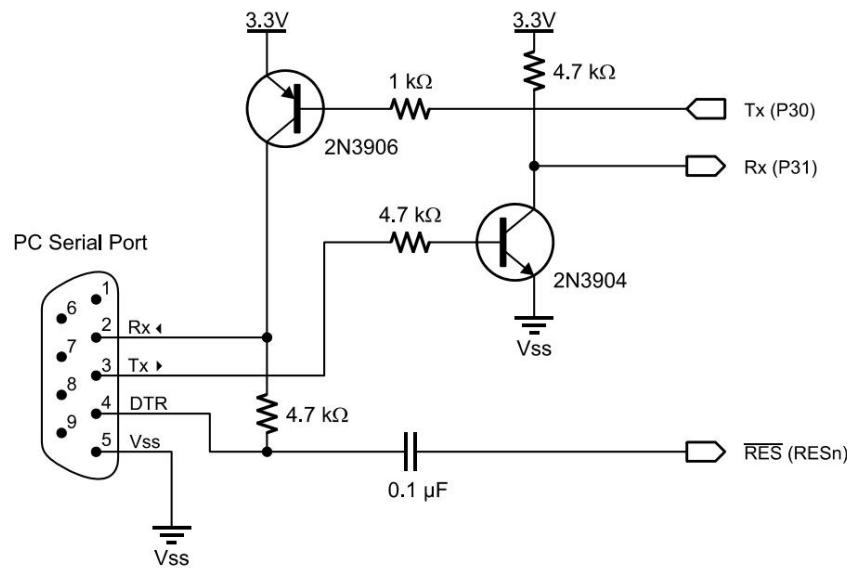
2.3. 典型连接图

2.3.1. 螺旋桨夹或螺旋桨插头连接 - 推荐请注意,虚线部分中的外部振荡器和 EEPROM 的连接是可选的。

螺旋桨夹,库存编号 32200;螺旋桨插头,库存编号 32201。螺旋桨夹/插头示意图可从 www.parallax.com 下载。



2.3.2. 备选串行端口连接



3.0 操作程序

3.1. 启动程序上电或复位时： 1.

Propeller 芯片的内部 RC 振

荡器开始以 20 kHz 运行,然后在 50 ms 复位延迟后切换到 12 MHz。然
后第一个处理器 (Cog 0) 加载并运行内置的 Boot Loader 程序。

2. 引导加载程序按顺序执行以下一项或多项任务:a. 通过引脚 P30 和 P31 检测来自主机
(例如 PC) 的通信。如果检测到来自主
机的通信,则引导加载程序将与主机对话以识别 Propeller 芯片,并可能将程序下载
到全局 RAM 中,也可以选择下载到外部 32 KB EEPROM 中。

b. 如果未检测到主机通信,则引导加载程序会在引脚 P28 和 P29 上查找外部 32
KB EEPROM。如果检测到 EEPROM,则整个 32 KB 数据映像将加载到
Propeller 芯片的全局 RAM 中。c. 如果未检测到 EEPROM,则引导加载程序
停止,Cog 0 终止,Propeller 芯片进入关机模式,所有 I/O 引脚均设置为输
入。

3. 如果步骤 2a 或 2b 成功将程序加载到全局 RAM 中,并且主机未给出
暂停命令,则 Cog 0 将使用内置的 Spin 解释器重新加载,并且用户
代码将从全局 RAM 运行。

3.2. 运行时程序

Propeller 应用程序是编译为二进制形式的用户程序,并下载到 Propeller
芯片的 RAM 或外部 EEPROM。该应用程序由用 Propeller 芯片的 Spin
语言 (高级代码) 编写的代码和可选的 Propeller 汇编语言组件 (低级
代码) 组成。用 Spin 语言编写的代码在运行时由运行 Spin 解释器的
cog 解释,而用 Propeller 汇编编写的代码则由 cog 以纯形式直接运行。
每个 Propeller 应用程序至少包含一点 Spin 代码,实际上可能完全用
Spin 编写,也可能使用不同数量的 Spin 和汇编编写。Propeller 芯片的
Spin 解释器在上述启动过程的第 3 步中启动,以运行应用程序。

一旦启动过程完成并且应用程序在 Cog 0 中运行,所有进一步的活动都
由应用程序本身定义。应用程序可以完全控制内部时钟速度。

I/O 引脚使用情况、配置寄存器以及在特定时间内运行的齿轮的时间、内容
和数量。所有这些都在运行时可变,由应用程序控制。

3.3. 关机程序

当 Propeller 进入关机模式时,内部时钟停止,导致所有齿轮停止,所有
I/O 引脚设置为输入方向 (高阻抗)。关机模式由以下三个事件之一触
发:1. VDD 低于电压过低阈值 (~2.7 VDC),当电压过低电路启用时,2.
RESn 引脚变为低电平,或 3. 应用程序请求重新启动 (请参阅
REBOOT

命令)。

当电压水平升至欠压阈值以上且 RESn 引脚为高电平时,关断模式停止。

4.0 系统组织

4.1. 共享资源

Propeller 中有两种类型的共享资源:1) 公共资源和 2) 互斥资源。公共资源
可由任意数量的 cog 随时访问。互斥资源也可由任意数量的 cog 访
问,但每次只能由一个 cog 访问。

公共资源是 I/O 引脚和系统计数器。所有其他共享资源本质上都是互斥
的,对它们的访问由集线器控制。请参阅第 7 页上的第 4.4 节。

4.2. 系统时钟系统时钟 (图 1,第

1 页) 是 Propeller 芯片几乎所有组件的中央时钟源。系统时钟信号来自
以下三个可能的来源之一:

- 内部 RC 振荡器 (~12 MHz 或 ~20 kHz)
- XI 输入引脚 (与
XO 引脚一起用作高阻抗输入或晶体振荡器)

- 时钟 PLL (锁相环)由 XI 供电
输入

源由 CLK 寄存器的设置决定,可在编译时选择,也可在运行时重新选择。
集线器和内部总线以系统时钟速度的一半运行。

4.3. 齿轮 (处理器)

Propeller 包含八 (8) 个相同的独立处理器,称为 cog,编号为 0 到 7。每个 cog 包含一个处理器块、配置为 512 个长度 (512 x 32 位) 的本地 2 KB RAM、两个带 PLL 的高级计数器模块、一个视频生成器、I/O 输出寄存器、I/O 方向寄存器,以及框图中未显示的其他寄存器。

所有八个齿轮均由系统时钟驱动;它们各自保持相同的时间参考,并且所有活动齿轮同时执行指令。它们还可以访问相同的共享资源。

Cog 可以在运行时启动和停止,并且可以编程为同时执行任务,可以独立执行,也可以通过主 RAM 与其他 cog 协调执行。每个 cog 都有自己的 RAM,称为 Cog RAM,其中包含 512 个 32 位寄存器。

除了最后 16 个寄存器之外,Cog RAM 全部都是通用 RAM,这些寄存器是特殊用途寄存器,如第 16 页表 15 所述。

4.4. 枢纽

为了保持系统完整性,互斥资源不能一次由多个 cog 访问。Hub 通过让每个 cog 以“循环”方式从 Cog 0 到 Cog 7 再回到 Cog 0 来控制对互斥资源的访问。Hub 及其总线以系统时钟速率的一半运行,让 cog 每 16 个系统时钟周期访问一次互斥资源。Hub 指令 (访问互斥资源的 Propeller Assembly 指令)需要 8 个周期才能执行,但它们首先需要与 Hub 访问窗口的开始同步。

它需要最多 15 个周期 (如果我们刚好错过了,则为 16 减 1) 来同步到集线器访问窗口再加上 8 个周期来执行集线器指令,因此集线器指令需要 8 到 23 个周期才能完成。

图 2 和图 3 显示了 Cog 0 需要执行集线器指令的示例。图 2 显示了最佳情况;集线器指令在该 cog 的访问窗口开始时就已准备就绪。集线器指令立即执行 (8 个周期),在下一个集线器访问窗口到来之前,为其他指令留出额外的 8 个周期。

图 3 显示了最坏的情况;在 Cog 0 的访问窗口开始后,集线器指令在周期内就已准备就绪;它差点错过了。Cog 等待直到下一个集线器访问窗口 (15 个周期后),然后集线器指令执行 (8 个周期),该集线器指令总共需要 23 个周期。同样,在下一个集线器访问窗口到来之前,集线器指令之后还有 8 个额外的周期供其他指令执行。为了从必须频繁访问互斥资源的 Propeller Assembly 例程中获得最高效率,将非集线器指令与集线器指令交错以减少等待下一个集线器访问窗口的周期数会很有帮助。由于大多数 Propeller Assembly 指令需要 4 个时钟周期,因此可以在原本连续的集线器指令之间执行两个这样的指令。

图 2:Cog-Hub
互动 最佳案例
设想

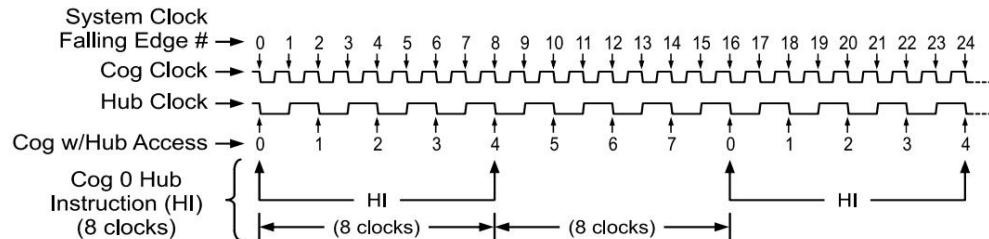
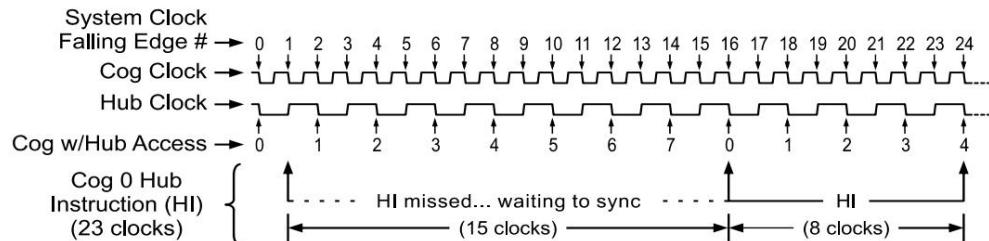


图 3:Cog-Hub
交互 最坏情况
设想



4.5. 输入/输出引脚

Propeller 有 32 个 I/O 引脚,其中 28 个是通用的。I/O 引脚 28 - 31 在启动时有特殊用途,之后可用于一般用途;请参阅第 4 页第 2.2 节。启动后,任何 I/O 引脚都可以随时由任何 cog 使用。应用程序开发人员需要确保在运行时不会有两个 cog 尝试将同一 I/O 引脚用于不同用途。

请参阅第 1 页的图 1,每个 cog 都有自己的 32 位 I/O 方向寄存器和 32 位 I/O 输出寄存器,以影响 Propeller 芯片对应的 32 个 I/O 引脚的状态。cog 所需的 I/O 方向和输出状态通过整个 cog 集合进行通信,成为“引脚方向”和“引脚输出”。

引脚方向是将 cog 的方向寄存器进行“或”运算的结果。引脚输出是将 cog 的输出状态进行“或”运算的结果。cog 的输出状态由其 I/O 模块 (计数器、视频生成器和 I/O 输出寄存器) 的位组成,这些位先进行“或”运算,然后与其方向寄存器的位进行“与”运算。所有 cog 仍然可以同时访问和影响 I/O 引脚,而不会发生电气争用,

正如这些规则所述:

答:仅当没有活动齿轮将引脚设置为输出时,该引脚才是输入。

B. 仅当所有将引脚设置为输出的活动齿轮也将引脚设置为低电平时,引脚才会输出低电平。

C. 如果任何活动 cog 将其设置为,则引脚输出高电平输出并将其设置为高电平。

表 3 展示了 cogs 集体对特定 I/O 引脚 (本例中为 P12) 影响的几种可能组合。为简单起见,这些示例假设每个 cog 的 I/O 硬件 (除其 I/O 输出寄存器外) 的第 12 位被清除为零 (0)。

任何关闭的齿轮都会将其方向寄存器和输出状态清除为零,从而有效地使其不再影响其余活动齿轮控制的 I/O 引脚的最终状态。

每个 cog 还具有自己的 32 位输入寄存器。此输入寄存器实际上是一个伪寄存器;每次读取它时,都会读取 I/O 引脚的实际状态,无论其输入或输出方向如何。

表 3:I/O 共享示例

	Cogs I/O 方向寄存器的第 12 位	Cogs I/O 输出寄存器的第 12 位	状态	规则
齿轮编号	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	输入/输出引脚 P12	已关注
示例 1	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	输入	—
示例 2	1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	输出低	乙
示例 3	1 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0	输出高	碳
示例 4	1 0 0 0 0 0 0 0	0 1 0 0 0 0 0 0	输出低	乙
示例 5	1 1 0 0 0 0 0 0	0 1 0 0 0 0 0 0	输出高	碳
示例 6	1 1 1 1 1 1 1 1	0 1 0 1 0 0 0 0	输出高	碳
示例 7	1 1 1 1 1 1 1 1	0 0 0 1 0 0 0 0	输出高	碳
例 8 注意:对	1 1 1 0 1 1 1 1	0 0 0 1 0 0 0 0	输出低	乙

于 I/O 方向寄存器,位位置为 1 表示相应的 I/O 引脚为输出方向;位位置为 0 表示将其设置为输入方向。

4.6. 系统计数器系统计数器是一个

全局、只读、32 位计数器,每个系统时钟周期递增一次。Cog 可以读取系统计数器 (通过其 CNT 寄存器,参见第 16 页上的表 15) 来执行计时计算,并可以使用 WAITCNT 命令 (参见第 19 页上的第 6.3 节和第 22 页上的第 6.4 节) 在其进程中创建有效延迟。系统计数器是一种公共资源,每个 cog 都可以同时读取。系统计数器在启动时不会被清除,因为它的实际用途是用于差分计时。如果 cog 需要从特定的固定时刻开始跟踪时间,它只需读取并保存该时刻的初始计数器值,然后将后续计数器值与该初始值进行比较。

4.7. 锁

有八个锁定位 (信号量) 可用于促进多个 cog 之间对用户定义资源的独占访问。如果两个或多个 cog 同时使用一个内存块,并且该内存块由多个 long (四个字节) 组成,则 cog 必须分别执行多次读取和写入才能检索或更新该内存块。这可能导致该内存块上出现读/写争用,其中一个 cog 可能正在写入,而另一个 cog 正在读取,从而导致误读和/或误写。

锁是通过 Hub 的 LOCKNEW、LOCKRET、LOCKSET 和 LOCKCLR 访问的全局位。由于锁只能通过 Hub 访问,因此每次只有一个 cog 可以影响它们,因此这是一种有效的控制机制。Hub 维护正在使用的锁及其当前状态的清单;cog 可以在运行时根据需要签出、返回、设置和清除锁。

4.8. 汇编指令执行阶段

图 4:汇编指令执行阶段

阶段	1	2	3	4	5
	(执行 N-1) 拿来 操作说明 否	写 结果 N-1	拿来 来源 否	拿来 目的地 否	(执行 N) 拿来 操作说明 N+1
时钟周期	M	M+1	M+2	M+3	M+4

M+5

Propeller 分五个阶段执行汇编指令。虽然执行一条完整指令需要六个时钟周期,但其中两个时钟周期专门用于执行两条相邻指令。这样每条指令的总吞吐量为四个时钟周期。

由指令 N 访问。为了加快程序执行的吞吐量,在 ALU 中执行当前指令的同时,从 cog 存储器中获取下一条要执行的指令。

最后在时钟周期 M+5 时,当前指令 N 的结果被写回到 cog 内存,完成第 5 阶段。



指令的部分交错对程序流有几个影响。首先,当通过 MOVI、MOVS、MOVD 或任何修改汇编指令的操作进行代码修改时,在执行修改后的指令之前必须至少执行一条指令。如果修改是在紧接着的指令 (N+1) 上进行的,则指令 N+1 的未修改版本将在将修改后的指令 N+1 版本写入 cog 内存之前一个时钟周期加载。

在第 1 阶段,程序计数器指向的指令 N 在时钟周期 M 期间从 cog 内存中取出。在周期 M+1 期间,上一条指令的结果被写入内存。在取出当前指令后写入上一条指令结果的原因将很快解释。

其次,直到时钟周期 M+4 结束,条件跳转才能确定是否会跳转。由于已经获取了下一条指令,因此只能预测两个可能分支中的一个。在 Propeller 中,总是预测条件分支会进行跳转。对于使用 DJNZ 的循环,除了最后一个循环外,每次都会进行跳转,因此可以实现更紧凑的循环执行时间。

在第 2 阶段,如果设置了指令 N 的立即数标志,则将 9 位源字段保存为源值。如果该值不是立即数,则在时钟周期 M+2 期间从 cog 内存中提取源字段指定的位置。在时钟周期 M+3 期间,从 cog 内存中提取目标字段中指定的位置(第 3 阶段)。

如果未执行跳转,cog 将不执行任何操作,直到获取下一条指令。这相当于在执行下一条指令之前插入一个 NOP。

此时(第 4 阶段),算术逻辑单元(ALU)已拥有执行指令所需的所有信息。执行指令需要一段时间才能获得结果。执行所需的时间由 ALU 执行的最慢操作决定。为了给 ALU 提供足够的时间来执行指令,为 ALU 提供了一个完整的时钟周期(M+4),以使结果稳定在最终状态。在此执行过程中,cog 内存不会

无条件跳转总是需要四个时钟周期才能执行,因为 Propeller 总是能够准确预测需要将哪个地址加载到程序计数器中才能执行下一条指令。无条件跳转的示例包括 JMP、JMPRET、CALL 和 RET。

如果指令需要访问任何集线器资源,则第 4 阶段将延长,直到集线器可用,从而将执行时间增加到至少 8 个时钟周期,最多可能增加到 23 个时钟周期。请参阅第 7 页上的第 4.4 节:集线器。

4.9. 齿轮计数器

每个 cog 有两个计数器模块 :CTRA 和 CTRB。每个计数器模块可以控制或监控最多两个 I/O 引脚,并在每个时钟周期将其 FRQ 寄存器有条件地 32 位累加到其 PHS 寄存器中。

每个计数器模块还具有自己的锁相环 (PLL),可用于合成高达 128 MHz 的频率。

只需进行一些设置或由齿轮进行监督,计数器便可用于:

- 频率合成 · 频率测量 · 脉冲
- 计数 · 脉冲测量 · 多引脚状态测量 ·
- 脉冲宽度调制 · 占空比测
- 量 · 数模转换 · 模数转换

对于其中的一些操作,cog 可以设置并处于自由运行模式。对于其他操作,它可以使用 WAITCNT 对循环内的计数器读取和写入进行时间对齐,从而产生更复杂的状态机效果。

请注意,对于 80 MHz 的齿轮时钟频率,计数器更新周期仅为 12.5 ns。如此高的速度与 32 位精度相结合,可实现非常动态的信号生成和测量。

计数器的设计目标是创建一个简单而灵活的子系统,该子系统可以在每个时钟周期执行一些重复性任务,从而释放 cog 来执行一些计算量更大的超级任务。虽然计数器只有 32 种基本操作模式,但它们通过软件动态使用的方式却没有限制。这一概念不可或缺的是使用 WAITPEQ、WAITPNE 和 WAITCNT 指令,这些指令可以使 cog 与其计数器进行事件对齐或时间对齐。

每个计数器有三个寄存器:

4.9.1. CTRA / CTRB – 控制寄存器CTR (CTRA 和 CTRB)寄存器选

择计数器的工作模式。写入此寄存器后,新的工作模式立即生效。向 CTR 写入零将立即禁用计数器,停止所有引脚输出和 PHS 累积。

CTRMODE 字段用于选择计数器的 32 种操作模式之一,可以使用 MOVI 指令方便地写入 (与 PLLDIV 一起)。这些操作模式列在第 11 页的表 6 中。

表 5:PLLDIV 字段								
PLLDIV %0001 %001 %010 %011 %100 %101 %110 %111								
输出	压控振荡器 128	压控振荡器 64	压控振荡器 三十二	压控振荡器 16	压控振荡器 8	压控振荡器 4	压控振荡器 2	压控振荡器 1

PLLDIV 选择 PLL 输出抽头,如果不使用,则可以忽略。

PLL 模式 (%000001 至 %000011) 导致每个时钟周期发生 FRQ 到 PHS 的累积。这会在 PHS[31] 中创建一个数控振荡器 (NCO),它为计数器 PLL 的参考输入提供信号。PLL 将使用其压控振荡器 (VCO) 将此频率乘以 16。为了稳定运行,建议将 VCO 频率保持在 64 MHz 至 128 MHz 之间。这相当于 NCO 频率为 4 MHz 至 8 MHz。

CTR 寄存器的 PLLDIV 字段选择将 VCO 频率的哪个二分频幕用作最终 PLL 输出。这提供了 500 kHz 至 128 MHz 的 PLL 范围。

BPIN 选择一个引脚作为辅助 I/O。如果不使用,可以忽略它,也可以使用 MOVD 写入操作说明。

APIN 选择一个引脚作为主 I/O。如果不使用,可以忽略它,也可以使用 MOVS 写入操作说明。

4.9.2. FRQA / FRQB – 频率寄存器FRQ (FRQA 和 FRQB)保存将累积到 PHS 寄存器中的值。对于某些应用,FRQ 可能被写入一次,然后被忽略。对于其他应用,它可能被快速调制。

4.9.3. PHSA / PHSB – 相位寄存器PHS (PHSA 和 PHSB)寄存器可

以通过 cog 指令进行写入和读取,但它也可用作自由运行累加器,可能在每个时钟周期将 FRQ 寄存器累加到自身中。任何写入 PHS 的指令都将覆盖该时钟周期的任何累加。只能通过源操作数读取 PHS (与 PAR.CNT.INA 和 INB 相同)。请注意,在 PHS 上执行读取-修改-写入指令 (如 “ADD PHSA, #1”) 会导致最后写入的值用作目标操作数输入,而不是当前累加值。

表 4:CTRA 和 CTRB 寄存器						
31	30..26	25..23 22..15 14..9	8..6		5..0	
- CTR 模式 PLLDIV -			国别密码	- 亚太密码		

表 6:计数器模式 (CTRMODE 字段值)				
控制模式	描述	积累 FRQx 至 PHSx	输出* 0 (无)	输出* 0 (无)
%00000	计数器已禁用 (关闭)	0 (从不) 1		
%00001	PLL 内部 (视频模式)	(总是) 11	0 锁相环 锁相环	0 0 !PLLx
%00010	PLL 单端			
%00011	PLL 差分			
%00100	NCO 单端	1	小灵通[31]	
%00101	NCO 差分	1	小灵通[31]	0 !PHSx[31]
%00110	DUTY 单端	1	PHSx-携带	
%00111	差速器	1	PHSx-携带	0 !PHSx-进位
%01000	POS 检测器	A1	0	0!
%01001	带反馈的 POS 检测器	A1	0	A1
%01010	POSEdge 检测器	A1 和 IA2	0	0!
%01011	带反馈的 POSEdge 检测器	A1 和 IA2	0	A1
%01100	NEG 检测器	!A1	0	0!
%01101	带反馈的 NEG 检测器	!A1	0	A1
%01110	NEGEdge 检测器	!A1 & A2	0	0!
%01111	带反馈的 NEGEDGE 检测器	!A1 & A2	0	A1
%10000	逻辑从不 逻辑 IA & !		0	0
%10001	B 逻辑 A & IB 逻辑 IB	0 !A1 & IB1	0	0
%10010	逻辑 IA & B 逻辑 IA 逻辑	A1 和 IB1	0	0
%10011	逻辑 A <> B 逻辑!	!B1	0	0
%10100	A IB	IA1 & B1	0	0
%10101		!A1	0	0
%10110		A1 <> B1	0	0
%10111		!A1 IB1	0	0
%11000	逻辑 A & B 逻辑 A ==	A1 和 B1	0	0
%11001	B 逻辑 A 逻辑 A IB 逻辑	A1 == B1	0	0
%11010	逻辑 B 逻辑 IA	A1	0	0
%11011	B 逻辑 A B 逻辑 总是	A1 !B1	0	0
%11100		B1	0	0
%11101		IA1 B1	0	0
%11110		A1 B1	0	0
%11111		1	0	0

*必须设置相应的 DIR 位才能影响引脚。A1 = APIN 输入延迟 1 个时钟。A2 = APIN 输入延迟 2 个时钟。B1 = BPIN 输入延迟 1 个时钟。

4.10. 视频生成器

每个 cog 都有一个视频生成器模块,便于以恒定速率传输视频图像数据。有两个寄存器和一个指令,用于控制和访问视频生成器。cog 的计数器 A 必须在 PLL 模式下运行,用于生成视频生成器的定时信号。视频比例寄存器指定每个像素的计数器 A PLL (PLLA) 时钟周期数,以及在获取由 cog 内执行的 WAITVID 指令提供的另一帧数据之前的时钟周期数。

视频配置寄存器,然后最终通过 WAITVID 指令提供数据。如果未能通过首先启动 PLLA 正确初始化视频生成器,则会导致 cog 在执行 WAITVID 指令时无限期挂起。

4.10.1. VCFG – 视频配置寄存器视频配置寄存器包含视频生成器的配置设置,如表 7 所示。

视频配置寄存器建立视频发生器应运行的模式,并可以生成 VGA 或复合视频 (NTSC 或 PAL)。

视频发生器应首先启动计数器 A,设置视频比例寄存器,设置

在 Propeller Assembly 中,可以使用 MOVI 方便地写入 VMode 到 AuralSub 字段

指令,可以使用 MOVD 写入 VGroup 字段
指令,而 VPins 字段可以用 MOVS 指令写入。

表 7:VCFG 寄存器									
31	30..29	28..	27..	26..	25..23	22..12	11..9	8	7..0
-	虚拟模式	控制模式	色度1	Chroma0 AuralSub	-	维基百科	-	-	VPins

2 位 VMode (视频模式)字段根据表 8 选择视频输出的类型和方向 (如果有)。

表 8:视频模式字段	
VMode 视频模式	
00	已禁用,未生成视频。
01	VGA 模式;VPins 7:0 上的 8 位并行输出
10	复合模式 1;在 VPins 7:4 上广播,在 VPins 3:0 上基带
11	复合模式 2;VPins 7:4 上的基带,广播 在 VPins 3:0

CMode (颜色模式)字段选择双色或四色模式。0 = 双色模式;像素数据为 32 位乘以 1 位,并且仅使用颜色 0 或 1。1 = 四色模式;像素数据为 16 位乘以 2 位,并且使用颜色 0 至 3。

Chroma1 (广播色度)位启用或禁用广播信号上的色度 (颜色)。0 = 禁用,1 = 启用。

Chroma0 (基带色度)位启用或禁用基带信号上的色度 (颜色)。0 = 禁用,1 = 启用。

AuralSub (听觉副载波)字段选择要调制的 FM 听觉 (音频)副载波频率的源。源是其中一个 cog 的 PLLA,由 AuralSub 的值标识。此音频必须已由源 PLLA 调制到 4.5 MHz 副载波上。

表 9:AuralSub 字段	
听觉辅助	副载波频率源
000	Cog 0 的 PLLA
001	Cog 1 的 PLLA
010	Cog 2 的 PLLA
011	Cog 3 的 PLLA
100	Cog 4 的 PLLA
101	Cog 5 的 PLLA
110	Cog 6 的 PLLA
111	Cog 7 的 PLLA

VGroup (视频输出引脚组)字段选择在哪一组 8 个 I/O 引脚上输出视频。

表 10:VGroup 字段	
VGroup 引脚组	
000	第 0 组:P7..P0
001	第 1 组:P15..P8
010	第 2 组:P23..P16
011	第 3 组:P31..P24
100-111	<保留以供将来使用>

VPins (视频输出引脚)字段是应用于 VGroup 引脚的掩码,指示在哪些引脚上输出视频信号。

表 11:VPins 字段	
VPins	影响
00001111	仅在下部 4 个针脚上驱动视频;复合
11110000	仅上方 4 个引脚驱动视频;复合
11111111	在所有 8 个引脚上驱动视频;VGA 此字段任何值均有效;
XXXXXX	以上值是最常见的。

4.10.2. VSCL – 视频比例寄存器视频比例寄存器设置视频数据的生成速率,如表 12 所示。

表 12:VSCL 寄存器		
VSCL 位		
31..20	19..12	11..0
—	像素时钟	帧时钟

8 位 PixelClocks 字段指示每个像素的时钟数;视频生成器模块移出每个像素之前应经过的时钟数。

这些时钟是 PLLA 时钟,而不是系统时钟。

该字段的值 0 将被解释为 256。

12 位 FrameClocks 字段表示每帧的时钟数;即视频生成器模块移出每帧之前经过的时钟数。这些时钟是 PLLA 时钟,而不是系统时钟。一帧是一像素数据 (通过 WAITVID 命令传送)。由于像素数据要么是 16 位乘以 2 位,要么是 32 位乘以 1 位 (分别表示 16 像素宽,4 种颜色,或 32 像素宽,2 种颜色),因此 FrameClocks 通常是 PixelClocks 值的 16 倍或 32 倍。此字段的值为 0 表示为 4096。

4.10.3. WAITVID 命令/指令

WAITVID 指令是将数据传送到 cog 的视频生成器硬件的机制。由于视频生成器独立于 cog 本身工作,因此每次显示设备需要数据时,两者必须同步。发生这种情况的频率由 PLLA 和视频比例寄存器的频率决定。cog 必须在视频生成器需要新数据之前获得新数据。cog 使用 WAITVID 等待正确的时间,然后将这些数据“传递”给视频生成器。

两个长整型数据通过语法“WAITVID颜色、像素”传递给视频生成器。

Colors参数是一个 32 位值,包含四个 8 位颜色值 (用于 4 色模式)或两个 8 位颜色值 (低 16 位) (用于 2 色模式)。对于

VGA 模式,每个 8 位颜色值都写入 VGroup 和 VPins 字段指定的引脚。对于 VGA,通常将 8 位分组为每原色 2 位以及水平和垂直同步控制线,但这取决于软件和应用程序如何使用这些位。对于复合视频,每个 8 位颜色值由 3 个字段组成。位 0-2 是生成信号的亮度值。位 3 是调制位,它决定是否生成色度信息,位 4-7 表示色度值的相位角。

当调制位设置为 0 时,色度信息将被忽略,只有亮度值输出到引脚。当调制位设置为 1 时,亮度值将以位 4-7 设置的相位角调制 ± 1 。为了实现色度值的全分辨率,PLLA 应设置为调制频率的 16 倍 (在复合视频中,这称为色突发频率)。cog 的 PLLA 用于生成广播频率;是否生成取决于 PLLB 是否正在运行以及 VMode 和 VPins 的值。

Pixels 参数描述要显示的像素模式,根据视频生成器的颜色深度配置,可以是 16 像素或 32 像素。当指定四色模式时, Pixels 是一个 16×2 位模式,其中每个 2 位像素是 Colors 的索引,根据该索引,数据模式应呈现到引脚。当指定双色模式时, Pixels 是一个 32×1 位模式,其中每个位指定 Colors 的低 16 位中的两种颜色模式中的哪一种应输出到引脚。Pixel 数据首先移出最低有效位 (LSB)。

当 FrameClocks 值大于 PixelClocks 值的 16 倍并且指定了 4 色模式时,将重复两个最高有效位,直到发生 FrameClocks PLLA 周期。当 FrameClocks 值大于 PixelClocks 值的 32 倍并且指定了 2 色模式时,将重复最高有效位,直到发生 FrameClocks PLLA 周期。当发生 FrameClocks 周期并且 cog 不在 WAITVID 中时

指令,此时源总线和目标总线上的任何数据都将被提取和使用。因此,在此之前使用 WAITVID 指令非常重要

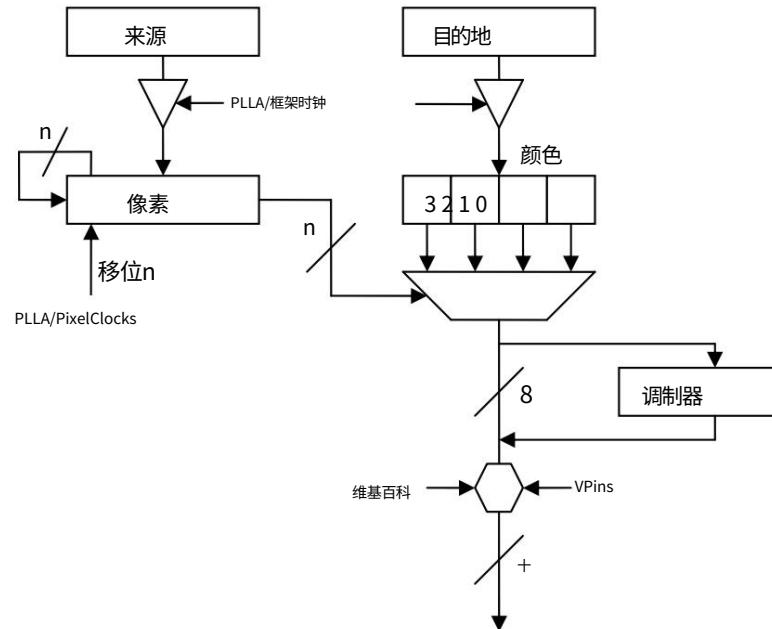
发生。

虽然视频发生器是为了显示视频信号而创建的,但其潜在应用却更加多样化。

复合视频模式可用于生成粒度为 16 或更小的相移键控通信,VGA 模式可用于生成具有完全可设置和可预测速率的任意位模式。

图 6 是 VGA 模式的组织结构框图。两个倒三角形是像素和颜色的加载机制; n 为 1 位或 2 位,具体取决于 CMode 的值。倒梯形是一个 4 路 8 位多路复用器,用于选择输出哪个颜色字节。在复合视频模式下,调制器将字节转换为亮度和色度信号并输出广播信号。VGroup 将 8 位引导到一组输出引脚,并输出到 VPins 中设置为 1 的引脚;这种组合功能由六边形表示。

图 6:视频生成器



4.11. CLK 寄存器 CLK 寄存器是系统时

钟配置控制;它决定系统时钟的源和特性。它配置 RC 振荡器、时钟 PLL、晶体振荡器和时钟选择器电路 (参见第 1 页的框图)。它在编译时通过 _CLKMODE 声明进行配置,并在运行时通过 CLKSET 命令可写入。每当写入 CLK 寄存器时,时钟源转换时都会发生约 75 μ s 的全局延迟。

每当此寄存器发生变化时,应将写入值的副本放置在时钟模式值位置 (主 RAM 中的 BYTE[4]) ,并将得到的主时钟频率写入时钟频率值位置 (主 RAM 中的 LONG[0]) ,以便引用此数据的对象具有用于其时序计算的最新信息。

尽可能使用 Spin 的 CLKSET 命令 (参见第 6.3 和 6.4 节) ,因为它会自动使用正确的信息更新所有上述位置。

表 13:有效时钟模式

有效表达式 CLK 寄存器值	有效表达式 CLK 寄存器值
射频快速接头 0_0_0_00_000	XTAL1 + PLL1X 0_1_1_01_011 XTAL1 + PLL2X 0_1_1_01_100 XTAL1 + PLL4X 0_1_1_01_101 XTAL1 + PLL8X 0_1_1_01_110 XTAL1 + PLL16X 0_1_1_01_111
慢速 0_0_0_00_001	
新输入 0_0_1_00_010	
晶振1 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 + PLL16X 0_1_1_10_111
晶振 0_0_1_10_010	
晶振3 0_0_1_11_010	
XINPUT + PLL1X 0_1_1_00_011	XTAL3 + PLL1X 0_1_1_11_011 XTAL3 + PLL2X 0_1_1_11_100 XTAL3 + PLL4X 0_1_1_11_101 XTAL3 + PLL8X 0_1_1_11_110 XTAL3 + PLL16X 0_1_1_11_111
XINPUT + PLL2X 0_1_1_00_100	
XINPUT + PLL4X 0_1_1_00_101	
XINPUT + PLL8X 0_1_1_00_110	
XINPUT + PLL16X 0_1_1_00_111	

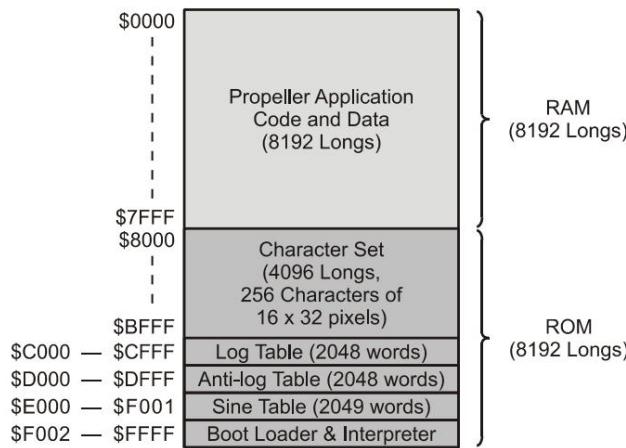
表 14:CLK 寄存器字段

少量	7	6	5	4	3	2	1	0
姓名	重置	普莲娜	欧塞纳	OSCM1	操作系统	时钟选择2	时钟选择1	时钟选择0
重置	影响							
0	除非您打算重置芯片,否则请始终在此处写入“0”。							
1	与硬件重置相同 - 重新启动芯片。							
普莲娜	影响							
0	禁用 PLL 电路。							
1	启用 PLL 电路。PLL 内部将 XIN 引脚频率乘以 16。OSCENA 必须为“1”才能将 XIN 信号传播到 PLL。PLL 的内部频率必须保持在 64 MHz 至 128 MHz 之间 - 这意味着 XIN 频率范围为 4 MHz 至 8 MHz。在通过 CLKSEL 位切换到其输出之一之前,请等待 100 μ s 以使 PLL 稳定下来。一旦 OSC 和 PLL 电路启用并稳定下来,您就可以通过更改 CLKSEL 位在所有时钟源之间自由切换。							
欧塞纳	影响							
0	禁用 OSC 电路							
1	启用 OSC 电路,以便时钟信号可以输入到 XIN,或者使 XIN 和 XOUT 可以一起用作反馈振荡器。OSC 位选择 OSC 电路的工作模式。请注意,晶体和谐振器不需要外部电阻器或电容器。允许晶体或谐振器稳定 10 毫秒,然后通过 CLKSEL 位切换到 OSC 或 PLL 输出。启用 OSC 电路时,可以同时启用 PLL,以便它们可以共享稳定期。							
OSCM1	操作系统	XOUT 电阻		XIN 和 XOUT 电容		频率范围		
0	0	无限		6 pF (仅限焊盘)		DC 至 80 MHz 输入		
0	1	2000 Ω		36 皮法		4 MHz 至 16 MHz 晶体/谐振器		
1	0	1000 Ω		26 皮法		8 MHz 至 32 MHz 晶体/谐振器		
1	1	500 Ω		16 皮法		20 MHz 至 60 MHz 晶体/谐振器		
时钟选择2	时钟选择1	时钟选择0	主时钟		来源	笔记		
0	0	0	~12 MHz		内部无需外部部件 (8 至 20 MHz)			
0	0	1	~20 kHz		内部无需外部部件,功率极低 (13-33 kHz)			
0	1	0	欣		OSC OSCENA 必须为“1”			
0	1	1	心 \times 1		OSC+PLL OSCENA 和 PLLENA 必须为‘1’			
1	0	0	心 \times 2		OSC+PLL OSCENA 和 PLLENA 必须为‘1’			
1	0	1	心 \times 4		OSC+PLL OSCENA 和 PLLENA 必须为‘1’			
1	1	0	心 \times 8		OSC+PLL OSCENA 和 PLLENA 必须为‘1’			
1	1	1	鑫 \times 16		OSC+PLL OSCENA 和 PLLENA 必须为‘1’			

5.0 内存组织

5.1. 主存储器

主存储器是一个 64 K 字节 (16 K 长) 的块,所有 cog 都可以通过 Hub 作为互斥资源进行访问。它由 32 KB 的 RAM 和 32 KB 的 ROM 组成。主存储器是字节、字和长寻址的。字和长以小端格式存储,最低有效字节优先。



5.1.1. 主内存

32 KB 的主 RAM 是通用的,是从主机或外部 32 KB EEPROM 下载的 Propeller 应用程序的目标。

5.1.2. 主 ROM

32 KB 的主 ROM 包含 Propeller 芯片功能所必需的所有代码和数据资源:字符定义、对数、反对数和正弦表,以及引导加载程序和旋转解释器。

5.1.3. 角色定义

ROM 的前半部分专用于一组 256 个字符定义。每个字符定义宽 16 像素,高 32 像素。这些字符定义可用于视频生成、图形 LCD、打印等。

字符集基于北美/西欧布局,并添加和插入了许多专业字符。有连接波形和原理图构建块字符、电子学中常用的希腊字符以及一些箭头和项目符号。(相应的 Parallax True-Type 字体随 Propeller Tool 软件一起安装并使用,并可供其他 Windows 应用程序使用。)

字符定义从左到右、从上到下按 0 到 255 进行编号,如下图 7 所示。它们的排列方式如下:每对相邻的奇偶字符合并在一起形成 32 个长字符。第一个字符对位于 \$8000-\$807F。第二对字符占用 \$8080-\$80FF,依此类推,直到最后一对字符填充 \$BF80-\$BFFF。

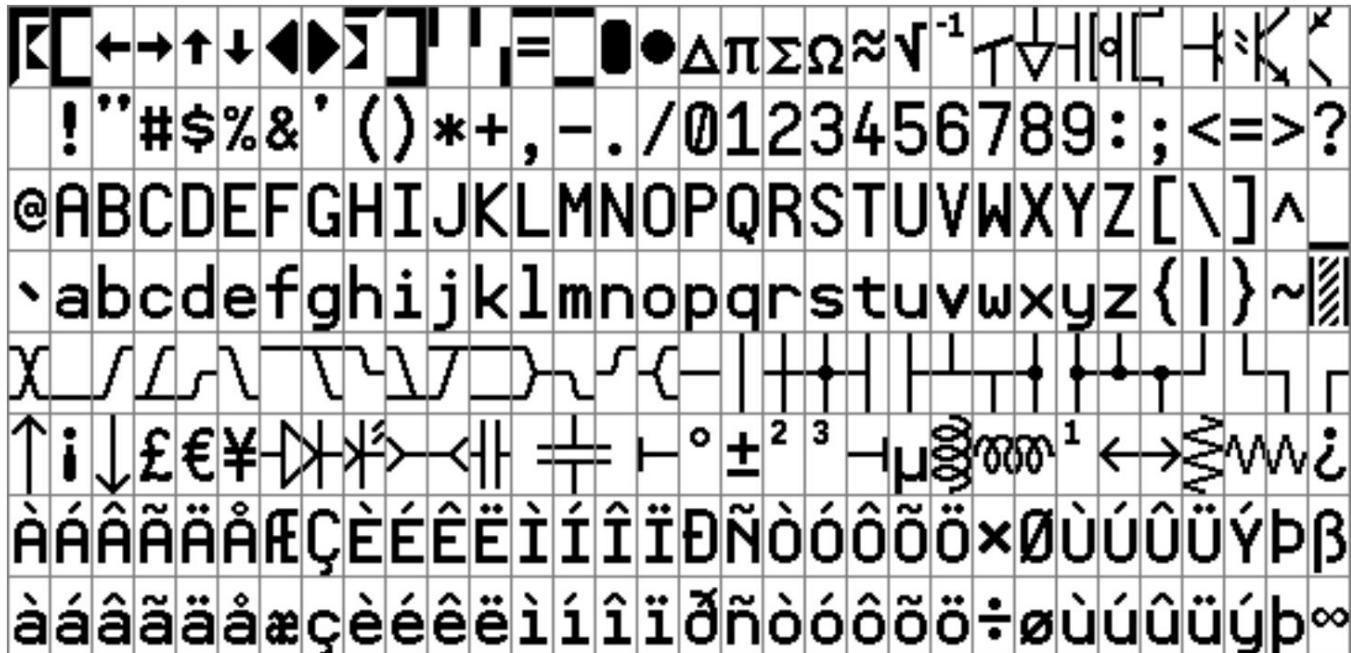


图 7:Propeller 字体字符集

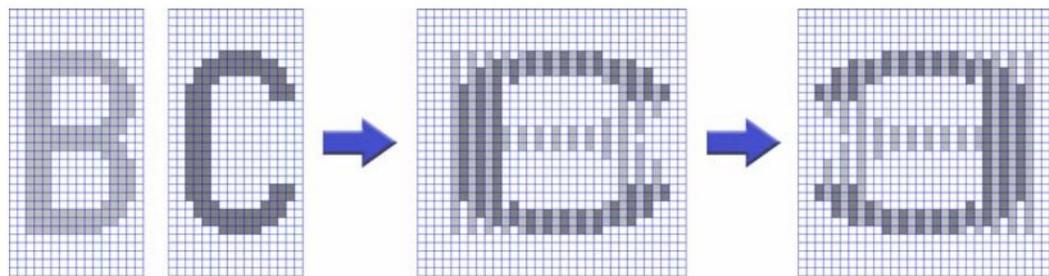


图 8

螺旋桨角色
交错

如图 8 所示,字符对逐行合并,这样每个字符的 16 个水平像素就相互隔开,并与相邻字符交错,这样偶数字符占用位 0.2.4. ...30,奇数字符占用位 1.3.5. ...31。最左边的像素位于最低位,而最右边的像素位于最高位。这为字符对中的每一行像素形成了一个长整型。从上行到下行构建的 32 个这样的长整型构成了完整的合并对定义。定义以这种方式进行编码,以便 cog 的视频硬件可以直接处理合并的长整型,使用颜色选择来显示偶数或奇数字符。

5.1.4. 数学函数表

基数为 2 的对数和反对数表各有 2048 个无符号字,便于将值转换为指数形式或从指数形式转换为指数形式,以方便进行某些操作;有关访问说明,请参阅 Propeller 手册。此外,正弦表提供 2049 个无符号 16 位正弦样本,范围从 0° 到 90° (含 0°) (分辨率为 0.0439°)。

5.2. Cog RAM 如第 4.3 节所

述,Cog RAM 用于存储可执行代码、数据和变量,最后 16 个位置用作系统计数器、I/O 引脚和本地 cog 外设的接口 (见表 15)。Cog RAM 仅可长寻址。

有些字符代码具有不可避免的含义,例如 9 表示制表符 (Tab)、10 表示换行符 (Line Feed) 和 13 表示回车符 (Carriage Return)。这些字符代码调用动作,并不等同于静态字符定义。因此,它们的字符定义已用于特殊的四色字符。这些四色字符用于在运行时绘制 3-D 框边缘,并以 16 x 16 像素单元实现,而不是正常的 16 x 32 像素单元。它们占用偶数-奇数字符对 0-1、8-

当 cog 启动时,位置 0 (\$000) 至 495 (\$1EF) 会按顺序从主 RAM/ROM 加载,其特殊用途位置 496 (\$1F0) 至 511 (\$1FF) 会清除为零。每个特殊用途寄存器都可以通过其物理地址、预定义名称或间接在 Spin 中通过寄存器数组变量 SPR 访问,索引为 0 到 15,即寄存器地址的最后四位。

9、10-11 和 12-13。

表 15:Cog RAM 特殊用途寄存器

Cog RAM 地图	地址	名称	类型	描述
\$000	\$1F0	帕尔	只读1	启动参数
General Purpose Registers (496 x 32)	\$1F1	胡纳米管	只读1	系统计数器
	\$1F2	伊纳	只读1	P31 - P0 的输入状态
	\$1F3	阿隆美辛	只读1	P63 - P323 的输入状态
	\$1F4	欧塔	读/写	P31 - P0 的输出状态
	\$1F5	输出端	读/写	P63 - P323 的输出状态
	\$1F6	爱尔兰投票局	读/写	P31 - P0 的方向状态
	\$1F7	迪尔比	读/写	P63 - P323 的方向状态
	\$1F8	通过时间的命令	读/写	计数器A控制
	\$1F9	计算时间的控制	读/写	反B控制
	\$1FA	质量保证	读/写	计数器A频率
\$1EF	\$1FB	*****	读/写	计数器B频率
\$1F0	\$1FC	磷酸二氢酶	读/写2	反击A阶段:
\$1FF	1金融时报	磷化氢	读/写2	反击B阶段
	\$1FE	氯丙三利酒馆	读/写	视频配置
	\$1FF	氯丙三利酒馆	读/写	视频比例

注 1:仅可作为源寄存器访问 (即 MOV Dest, Source)。

注 2:仅可作为源寄存器读取 (即 MOV Dest, Source);不可作为目标寄存器进行读取-修改-写入。

注 3:保留以供将来使用。

6.0 编程语言

Propeller 芯片使用两种专门为设计的语言进行编程:1) Spin,一种高级基于对象的语言;2) Propeller Assembly,一种低级、高度优化的汇编语言。Propeller Assembly 中有许多基于硬件的命令,在 Spin 语言中都有直接对应命令。

Spin 语言由 Propeller 工具软件编译为 token,并在运行时由 Propeller 芯片内置的 Spin 解释器进行解释。Propeller 汇编语言由 Propeller 工具汇编为纯机器码,并在运行时以纯机器码形式执行。

Propeller 对象可以完全用 Spin 编写,也可以使用 Spin 和 Propeller Assembly 的各种组合。几乎完全用 Propeller Assembly 编写对象通常很有优势,但至少需要两行 Spin 代码才能启动最终应用程序。

6.1. 保留字列表

列出的所有字始终是保留的,无论是在 Spin 还是在 Propeller Assembly 中编程。截至 Propeller Tool v1.05:

表 16:保留字列表

_CLK频率	认知功能	如果	锁定新	北约帕	重复	已验证
_CLK模式	资讯	如果 C_AND_Za	锁定	禁止	雷萨	截断
FREEs	停止	如果	锁具	核受体	结果	直到
堆栈	缺点	如果	多头	对象	雷塔	增值转售商
_XINFRREQs	常数	如果 C_OR_NZa	长填方	ONESa#	返回	真空光纤光缆
中止	抗逆转录病毒药物	否则	长途移动	奥德	雷瓦	电压门限
丙烯腈	癌基因	IF_Ea	向下查看	有机镓	罗拉	等待时间
ABSNEGA	数据表	干扰素	LOOKDOWNZs	其他的	辐射剂量	等待时间
腺苷酸	迪拉德	如果 NC 和 NZa	查找	OUT广告	回合	等待
附加信息表	DIRBd#	如果_NC_AND_Za	查找区域	输出端口#	萨拉	等待视频
抗干扰能力	DJNZa	否则	最大	帕尔	硫脲	钨钙
亚硝基	其他	否则	最大额	肺动脉高压	舒张功能受体	时间
差分信号	其它条件语句	IF_NEa	甲基钠	苯并噻嗪	战败绩效指标	字
和	否则	IF_NEVERa	明萨	病因	步骤	填词
安得拉	恩卡#	IF_NZa	金属氧化物半导体	PLL1X	STRCOMP	单词移动
字节	已假	如果新西兰和加拿大		PLL2X	字符串	韦拉
字节填充	文件	如果 NZ_AND_NCa	摩维亚	PLL4X	字符串大小	写字符
字节移动	国际互联	IF_NZ_OR_Ca	金属氧化物半导体场	PLL8X系列	苏巴	沃龙加
卡拉	浮点数	如果是 NZ 或 NCa		锁相环	苏巴布萨	抗水抗盐
案例	来自	IF_Za	穆萨#	POSXd	亚基底	双方
希普弗	频次质控	如果_Z_AND_Ca	多路复用器	优先股	SUBSXa	信输入
时钟频率	频率质量分数	如果_Z_AND_NCa	多路复用	公共事业单位	亚基	异或
时钟模式	胡波帕	IF_Z_EQ_Ca	多路复用		苏木钙	晶圆
时钟设置	因果关系	IF_Z_NE_Ca	多路复用器	退出	苏门答腊岛	晶圆
碳钢	干扰素	IF_Z_OR_Ca	负离子	射频场	萨姆扎	晶圆
甲基丙烯酸甲酯	IF_Aa	否则	内皮细胞	雷克萨	萨姆扎	
复合膜结构	IF_AEa	腺病毒	心房纤颤源	RC低速	萨姆扎	
CMPSXa	IF_ALWAYSa	干扰素#		RDBYTEa	测试	
甲基异丙基苯	IF_Ba	压力等级	恩扎	测试Na	测试	
碳纳米管	IF_BEa	甲基异戊酸酯	负	罗加	天津市	
COGIDd	钙	锁定	尼格扎	读字长	特杰扎	
			近端串扰	重启	任务	

a = 组装元件; s = 自旋元件; d = 对偶 (两种语言均可用); # = 保留供将来使用

6.1.1. 为将来使用而保留的字

- DIRB,INB 和 OUTB:保留用于将来可能的 64 I/O 引脚模型。与 P8X32A 一起使用时,这些标签可用于访问这些位置的 Cog RAM 以供通用使用。
- ENC,MUL,MULS,ONES:与当前 P8X32A 架构一起使用会产生不确定的结果。

6.2. 数学和逻辑运算符

表 17:数学和逻辑运算符

级别1	操作员		持续的表达式3		是一元的	描述		
	普通的	分配2	表达式3					
			整数	漂浮				
最高 (0)	--	总是			预减	(-X)或后减 (X--)。 预增 (++X)或后增 (X++)。 将位 7		
	++	总是			符号扩展	(~X)或后清零为 0 (X~)。 将位 15 符号扩展 (~~X)或后设		
	~	总是			置为 -1	(X~~)。 随机数正向 (X?)或反向 (X?)。 符号地址。 对象		
	~~	总是			地址加符号。	正数 (+X) ;加法的一元形式。 负数 (-X) ;减法的一元形		
	?	总是			式。	平方根。 绝对值。		
	@	绝不						
	@@	绝不						
1	+	绝不						
	-	如果单独						
	^^	如果单独						
		如果单独						
	<	如果单独			按位	将 0 – 31 解码为带单个高位的长整型。 按位:将长整型编码		
	>	如果单独			为 0 – 32;高位优先。			
	!	如果单独			按位	非。		
2	<-	<=			按位	左循环旋转。		
	->	->=			按位	右旋转。		
	<<	<<=			按位	左移。		
	>>	>>=			按位	右移。		
	~>	~>=				将算术右移。		
	><	><=			按位	反转。		
3	&	&=			按位	:AND。		
4		=			按位	:或。		
	^	^=			按位	:异或。		
5	*	*=				相乘并返回低 32 位 (有符号)。		
	**	**=				相乘并返回高 32 位 (有符号)。		
	/	/=				除以 (有符号)。		
	//	//=				模数 (有符号)。		
6	+	+=				添加。		
	-	-=				减去。		
7	#>	#>=				限制最小值 (有符号)。		
	<#	<#=				限制最大值 (有符号)。		
8	<	<=				布尔值:小于 (有符号)。		
	>	>=				布尔值:大于 (有符号)。		
	<>	<>=				布尔值:不等于。		
	==	==				布尔值:相等。		
	=<	=<=				布尔值:等于或小于 (有符号)。		
	=>	=>=				布尔值:等于或大于 (有符号)。 布尔值:非 (将		
9	不是	如果单独			非 0 提升为 -1)。			
10	和	并且=				布尔值:AND (将非 0 提升为 -1)。		
11	或者	或=				布尔值:OR (将非 0 提升为 -1)。		
最低 (12)	=	总是	不适用	无		常量赋值 (CON 块)。		
	:=	优先级始	不适用	无		变量分配 (PUB/PRI 块)。		

终为1:高级别运算符先于低级别运算符求值。同级别的运算符是可交换的;求值顺序无关紧要。

2二元 (非一元)运算符的赋值形式具有最低优先级 (第 12 级)。

3常量表达式中不允许使用运算符的赋值形式。

6.3. Spin 语言汇总表

旋转命令	返回 价值	描述
ABORT 值 BYTE		使用中止状态退出 PUB/PRI 方法,并带有可选的返回值。
符号 [计数] 符号 BYTE		在 VAR 块中声明字节大小的符号。
数据 [计数] BYTE [基址] [偏移]		在 DAT 块中声明字节对齐/或字节大小的数据。
量]符号.BYTE [偏移量]	读/写主存储器的字节。	
	读取/写入字/长整型变量的字节大小部分。	
BYTEFILL (起始地址,值,计数)		用一个值填充主内存的字节。
BYTMOVE (目标地址,源地址,计数)		将字节从主内存中的一个区域复制到另一个区域。
CASE CaseExpression MatchExpression : 语 句 匹配表达式： 声明 (多份) 其 他： 声明		将表达式与匹配的表达式进行比较,如果匹配则执行代码块。 MatchExpression可以包含单个表达式或多个逗号分隔的表达式。表达式可以是单个值 (例如:10)或一系列值 (例如:10..15)。
奇普维尔		Propeller 芯片的版本号 (位于\$FFFF 处的字节)
时钟频率		当前系统时钟频率,单位为 Hz (Long at \$0000)
时钟模式		当前时钟模式设置 (位于 \$0004 的字节)
CLKSET (模式,频率)		在运行时设置时钟模式和系统时钟频率。
碳纳米管		当前 32 位系统计数器值。
慢性胃炎		当前齿轮的 ID 号 0-7。
COGINIT (CogID, SpinMethod (参数列表) ,堆栈指针)		通过 ID 启动或重新启动 cog 来运行 Spin 代码。
COGINIT (CogID、AsmAddress、参数)		通过 ID 启动或重新启动 cog 来运行 Propeller Assembly 代码。
COGNEW (SpinMethod (参数列表) , StackPointer)	为 Spin 代码启动新的齿轮并获取齿轮 ID;0-7 = 成功, -1 = 失败。	
COGNEW (AsmAddress,参数)		启动螺旋桨装配代码的新齿轮并获取齿轮 ID;0-7 = 成功, -1 = 失败。
COGSTOP (CogID)		根据 ID 停止齿轮。
反对 符号= Expr ((, :)) 符号= Expr ...		声明符号、全局常量。
反对 #Expr ((, :)) 符号 [偏移] ((, :)) 符号 [偏移] ...		声明全局枚举 (增加符号常量)。
碳 在 符号 [偏移] ((, :)) 符号 [偏移] ...		声明全局枚举 (增加符号常量)。
CONSTANT (常量表达式)		声明内联常量表达式在编译时完全解析。
数据对齐命令		计数器 A 控制寄存器。
计算时间控制		计数器 B 控制寄存器。
数据保护 符号对齐大小 数据 [数量] ,大小数据 [数量] ...		声明数据表,按指定方式对齐和调整大小。
数据保护 符号 条件指令效果		表示螺旋桨组装说明。
DIRA [Pin(s)]	32	位端口 A 的方向寄存器。cog 启动时默认为 0 (输入)。
FILE 文件名		导入外部文件作为 DAT 块中的数据。
FLOAT (整型常数)		将任意块中的整数常量表达式转换为编译时浮点值。计数器 A 频率寄存器。
质量保证		
线程保护命令		计数器 B 频率寄存器。

旋转命令	返回 价值	描述
((IF IFNOT)条件 If语句 ELSEIF条件 Elseif 语句 ... ELSEIFNOT条件 Elseif 语句 ... 其他 Else语句 INA		测试条件,如果有效则执行代码块。 IF 和 ELSEIF 分别测试是否为 TRUE。IFNOT 和 ELSEIFNOT 分别测试是否为 FALSE。
[Pin(s)]	✓ 32 位端口 A 的输入寄存器。	
锁定清除 (ID)		将信号量清除为假并获取其之前的状态;TRUE 或 FALSE。
锁新		检查新的信号量并获取其 ID;0-7,如果没有可用的,则获取 -1。
锁扣(ID)		将信号量返回到信号量池,释放它以供将来的 LOCKNEW 请求使用。
锁具 (ID)		将信号量设置为真并获取其先前的状态;TRUE 或 FALSE。
LONG符号 [计数] 符号		在 VAR 块中声明长尺寸符号。
LONG数据 [计数]		在 DAT 块中声明长对齐和/或长尺寸数据。
LONG [基地址] [偏移量]	读/写主存的长整型。	
LONGFILL (起始地址,值,计数)		用一个值填充主内存的长整型数。
LONGMOVE (目标地址,源地址,计数)		将 long 从主内存中的一个区域复制到另一个区域。
LOOKDOWN (值:表达式列表)		获取列表中某个值的基于一的索引。
LOOKDOWNZ (值:表达式列表)		获取列表中某个值的从零开始的索引。
LOOKUP (索引:表达式列表)		从列表的基于一的索引位置获取值。
LOOKUPZ (索引:表达式列表)		从列表的从零开始的索引位置获取值。
下一个		跳过 REPEAT 循环的剩余语句并继续下一次循环迭代。
对象 符号 [计数] : “对象” 符号 [计数] : “对象” ...		声明符号对象引用。
OUTA [引脚]	32 位端口 A 的输出寄存器。cog 启动时默认为 0 (接地)。	
帕尔	Cog 启动参数寄存器。	
磷酰二氯酶	计数器 A 锁相环 (PLL) 寄存器。	
磷化氢	计数器 B 锁相环 (PLL) 寄存器。	
PRI名称 (Par ,Par ...) :RVal LVar [Cnt] ,LVar [Cnt] ... 源代码语句		声明具有可选参数、返回值和局部变量的私有方法。
PUB名称 (Par ,Par ...) :RVal LVar [Cnt] ,LVar [Cnt] ... 源代码语句		声明具有可选参数、返回值和局部变量的公共方法。
辞职		立即退出 REPEAT 循环。
重启		重置 Propeller 芯片。
重复计数 语句		重复执行代码块,可以无限次执行,也可以进行有限次数的迭代。
重复从开始到结束 STEP Delta 语句		重复执行代码块,进行有限次数的迭代。
重复((直到 当))条件 声明		重复执行代码块,零到多条件迭代。
重复 声明 ((直到 当))条件		重复执行代码块,一对多条件迭代。
结果	PUB/PRI 方法的返回值变量。	
返回价值	退出 PUB/PRI 方法,并返回可选值。	
ROUND (浮点常数)		在任何块中,在编译时将浮点常数四舍五入为最接近的整数。
SPR [指数]	特殊用途寄存器阵列。	
STRCOMP (字符串地址 1,字符串地址 2)		比较两个字符串是否相等。
STRING (字符串表达式)		声明内联字符串常量并获取其地址。

旋转命令	返回 价值	描述
STRSIZE (字符串地址)		获取零终止字符串的大小（以字节为单位）。
TRUNC (浮点常数)		在编译时从任何块中的浮点常量中删除小数部分。
声明全局 大小 符号 [计数] ((, 大小) 符号 [计数] ...)		声明符号全局变量。
视频配置寄存器		
视频比例寄存器		
WAITCNT (值)		暂时暂停 cog 的执行。
WAITPEQ (状态,掩码,端口)		暂停 cog 的执行,直到 I/O 引脚与指定状态匹配。
WAITPNE (州,掩码,端口)		暂停 cog 的执行,直到 I/O 引脚与指定状态不匹配。
WAITVID (颜色,像素)		暂停 cog 的执行直到其视频生成器可用于像素数据。
WORD符号 [计数] 符号		在 VAR 块中声明字大小的符号。
WORD数据 [计数] WORD [基地]		在 DAT 块中声明字对齐和/或字大小的数据。
址] [偏移量]符号.WORD [偏移量]	读/写主存储器的字。	
	读取/写入长整型变量的字长部分。	
WORDFILL (起始地址,值,计数)		用一个值填充主存储器的字。
WORDMOVE (目标地址,源地址,计数)		将单词从主内存中的一个区域复制到另一个区域。

6.3.1. 常量

常量 (预定义)		
常数1	描述	
_CLK频率	可在顶部对象文件中设置以指定系统时钟频率。	
_CLK模式	可在顶部对象文件中设置,以指定应用程序的时钟模式。	
_XINFREQ	可在顶部对象文件中设置以指定外部晶体频率。	
_自由的	可在顶部对象文件中设置以指定应用程序的可用空间。	
_堆	可在顶部对象文件中设置,以指定应用程序的堆栈空间。	
真的	逻辑正确:	-1 (\$FFFFFF)
错误的	逻辑错误:	0 (\$00000000)
POSX	最大正整数:	2,147,483,647 (\$7FFFFFF)
负相关	最大负整数:	-2,147,483,648 (\$80000000)
PI	浮点 PI:	≈3.141593 (\$40490FDB)
射频快速接头	内部快速振荡器:	\$00000001 (%000000000001)
慢速	内部慢速振荡器:	\$00000002 (%000000000010)
新输入	外部时钟/振荡器:	\$00000004 (%00000000100)
晶振1	外部低速晶振:	\$00000008 (%00000001000)
晶振	外部中速晶振:	\$00000010 (%00000010000)
晶振3	外部高速晶振:	\$00000020 (%00000100000)
PLL1X	外部频率乘以1:	\$00000040 (%00001000000)
PLL2X	外部频率乘以2:	\$00000080 (%00010000000)
PLL4X	外部频率乘以4:	\$00000100 (%00100000000)
PLL8X	外部频率乘以8:	\$00000200 (%01000000000)
PLL16X	外部频率乘以16:	\$00000400 (%10000000000)

1 “可设置”常量在 Top Object File 的 CON 块中定义。请参阅 _CLKMODE 的有效时钟模式。其他可设置常量使用整数。

6.4. Propeller 汇编指令表

Propeller 汇编指令表列出了指令的 32 位操作码、输出和时钟周期数。操作码由指令位(iiisi)、Z 标志的“效果”状态、C 标志、结果和间接/立即状态(zcri)、条件执行位(cccc)以及目标和源位(dddddddsssss)。Z 和 C 标志 (如果有) 的含义显示在 Z 结果和 C 结果字段中;表示这些标志中 1 的含义。结果字段(R)显示指令的默认行为,即写入 (1) 或不写入 (0) 指令的结果值。时钟字段显示执行指令所需的时钟数。

01 零 (0) 和一 (1) 表示二进制 0 和 1。
 我 小写 “i” 表示受即时状态影响的位。
 ds ? 小写 “d” 和 “s” 表示目标位和源位。
 --- 问号表示由编译器动态设置的位。
 --- 连字符表示不适用或不重要的项目。
 .. 双点表示代表一系列连续的值。

iiisi zcri cccc ddddddsssss 指令		描述	Z 结果	C 结果	R 时钟
000000 000i 1111 ddddddsssss WRBYTE D,S 将 D[7..0] 写入主存储器字节 S[15..0]			-	-	0.8, 23
000000 001i 1111 ddddddsssss RDBYTE D,S		将主存储器字节 S[15..0] 读入 D (0-延伸)	结果 = 0	-	1.8, 23 *
000001 000i 1111 ddddddsssss WRWORD D,S 将 D[15..0] 写入主存储器字 S[15..1]			-	-	0.8, 23
000001 001i 1111 ddddddsssss 读字 D,S		将主存储器字 S[15..1] 读入 D (0-扩展)	结果 = 0	-	1.8, 23 *
000010 000i 1111 ddddddsssss WRLONG D,S 将 D 写入主存储器 long S[15..2] 000010 001i 1111 ddddddsssss			-	-	0.8, 23
ssssssss RDLONG D,S 将主存储器 long S[15..2] 读入 D 000011 000i 1111 ddddddsssss HUBOP D,S 根据 S 执			结果 = 0	-	1.8, 23 *
行集线器操作 将全局 CLK 寄存器设置为 D[7..0]			结果 = 0	-	0.8, 23
000011 0001 1111 ddddddsssss -----000 时钟设置 D			-	-	0.8, 23
000011 0011 1111 ddddddsssss -----001 COGID D		将此齿轮号 (0..7) 放入 D	ID = 0	0	1.8, 23
000011 0001 1111 ddddddsssss -----010 COGINIT D		根据 D 停止齿轮号 D[2..0] 初始化齿	ID = 0	无齿轮	0.8, 23
000011 0001 1111 ddddddsssss -----011 COGSTOP D		轮	停止 ID = 0 无齿轮	自由	0.8, 23
000011 0011 1111 ddddddsssss -----100 LOCKNEW D		将新的锁号 (0..7) 签入 D 返回锁号 D[2..0]	ID = 0	无锁	1.8, 23
000011 0001 1111 ddddddsssss -----101 锁定 D			ID = 0	无锁	0.8, 23
000011 0001 1111 ddddddsssss -----110 锁具 D		设置锁号 D[2..0]	ID = 0	先前锁定状态	0.8, 23
000011 0001 1111 ddddddsssss -----111 LOCKCLR D		清除锁号 D[2..0]	ID = 0	先前锁定状态	0.8, 23
000100 001i 1111 ddddddsssss MUL	D,S 将	无符号 D[15..0] 乘以 S[15..0]	结果 = 0	-	1 未来
000101 001i 1111 ddddddsssss 穆尔斯	D,S 将	有符号的 D[15..0] 乘以 S[15..0]	结果 = 0	-	1 未来
000110 001i 1111 ddddddsssss ENC	D,S 将	S 的幅度编码到 D 中, 结果 = 0..31 D,S 获取 S 中 1	结果 = 0	-	1 未来
000111 001i 1111 ddddddsssss 个位	D,S 将	的数目到 D 中, 结果 = 0..31 D,S 将 D 右移 S[4..0] 位 D,S	结果 = 0	-	1 未来
001000 001i 1111 ddddddsssss ROR	将 D 左移 S[4..0] 位 D,S 将 D 右移		结果 = 0	D[0]	1 4
001001 001i 1111 ddddddsssss 罗尔	S[4..0] 位, 将新的 MSB 设置为 0 D,S		结果 = 0	德[31]	1 4
001010 001i 1111 ddddddsssss SHR	将 D 左移 S[4..0] 位, 将新的 LSB 设置为 0		结果 = 0	D[0]	1 4
001011 001i 1111 ddddddsssss SHL			结果 = 0	德[31]	1 4
001100 001i 1111 ddddddsssss RCR	D,S 将	进位循环右移 S[4..0] 位, 放入 D 中 D,S 将进	结果 = 0	D[0]	1 4
001101 001i 1111 ddddddsssss RCL	位循环左移 S[4..0] 位, 放入 D 中 D,S 将 D 算术右		结果 = 0	德[31]	1 4
001110 001i 1111 ddddddsssss SAR	移 S[4..0] 位		结果 = 0	D[0]	1 4
001111 001i 1111 ddddddsssss REV	d,s	反转 32-S[4..0] 底部位在 D 和 0-延长	结果 = 0	D[0]	1 4
010000 001i 1111 ddddddsssss 分钟	D,S 如果	有符号则将 D 设置为 S (D < S)	S = 0	有符号 (D < S)	1 4
010001 001i 1111 ddddddsssss 最大	D,S 如果	有符号则将 D 设置为 S (D => S)	S = 0	有符号 (D < S)	1 4
010010 001i 1111 ddddddsssss 最小值	D,S 如果	无符号 (D < S), 则将 D 设置为 S	S = 0	无符号 (D < S)	1 4
010011 001i 1111 ddddddsssss 最大	D,S 如果	无符号, 则将 D 设置为 S (D => S)	S = 0	无符号 (D < S) 1	4
010100 001i 1111 ddddddsssss MOVS	D,S 将	S[8..0] 插入到 D[8..0] 中	结果 = 0	-	1 4
010101 001i 1111 ddddddsssss MOVD	D,S 将	S[8..0] 插入到 D[17..9] 中	结果 = 0	-	1 4
010110 001i 1111 ddddddsssss MOVI	D,S 将	S[8..0] 插入到 D[31..23] 中	结果 = 0	-	1 4
010111 001i 1111 ddddddsssss JMPRET D,S 将	PC+1 插入 D[8..0] 并将 PC 设置为 S[8..0]		结果 = 0	-	1 4
010111 000i 1111 -----ssssssss JMP	=	将 PC 设置为 S[8..0]	结果 = 0	-	0 4

iiiiii zcri cccc dddddddd ssssssss 指令		描述	Z 结果	C 结果	R时钟
010111 0011 1111 ??????? 呼叫	#S 与	JMPRET 类似,但汇编器处理细节	结果 = 0	-	1 4
010111 0001 1111 ----- RET		类似于 JMP,但汇编器处理细节	结果 = 0	-	0 4
011000 000i 1111 dddddddd ssssssss 测试	D, S AND S 与 D 仅影响标志	011001 000i	D = 0	结果奇偶性	0 4
1111 dddddddd ssssssss TESTN D, S AND IS 到 D 仅影响标志			结果 = 0	结果奇偶性	0 4
011000 001i 1111 dddddddd ssssssss 和	D, S 与 S 合并为 D		结果 = 0	结果奇偶性	1 4
011001 001i 1111 dddddddd ssssssss ANDN	D, S 与 IS 合并为 D		结果 = 0	结果奇偶性	1 4
011010 001i 1111 dddddddd ssssssss 或	D, S 或 S 变为 D		结果 = 0	结果奇偶性	1 4
011011 001i 1111 dddddddd ssssssss 异或	D, S 将 S 与 D 进行异或		结果 = 0	结果奇偶性	1 4
011100 001i 1111 dddddddd ssssssss MUXC	D, S 使用 S 作为掩码将 C 复制到 D 中的位 011101		结果 = 0	结果奇偶性	1 4
001i 1111 dddddddd ssssssss MUXNC D, S 使用 S 作为掩码将 IC 复制到 D 中的位, D, S 使用 S 作为掩码将 Z 复制到 D 中的位			结果 = 0	结果奇偶性	1 4
011110 001i 1111 dddddddd ssssssss MUXZ	位 01111001i 1111 dddddddd ssssssss		结果 = 0	结果奇偶性	1 4
MUXNZ D, S 使用 S 作为掩码将 Z 复制到 D 中的位			结果 = 0	结果奇偶性	1 4
100000 001i 1111 dddddddd ssssssss 添加	D, S 将 S 添加到 D 中		D + S = 0	无符号进位	1 4
100001 001i 1111 dddddddd ssssssss SUB	D, S 从 D 中减去 S		D - S = 0	未签名借位 1	4
100001 000i 1111 dddddddd ssssssss CMP	D, S 将 D 与 S 进行比较		D = S	无符号 (D < S)	0 4
100010 001i 1111 dddddddd ssssssss ADDABS D, S 将绝对 S 添加到 D 中	0010011001i 1111 dddddddd		D + S = 0 无符号进位 1 1		4
sssssssss SUBABS D, S 从 D 中减去绝对 S			D - S = 0	未签名借位 2 1	4
100100 001i 1111 dddddddd ssssssss SUMC	D, S 将 -S (如果 C) 或 -S (如果 IC) 加到 D 中		D ± S = 0	有符号溢出 1	4
100101 001i 1111 dddddddd ssssssss SUMNC D, S 将 -S (如果 C) 或 -S (如果 IC) 加到 D 中			D ± S = 0	有符号溢出 1	4
100110 001i 1111 dddddddd ssssssss SUMZ	D, S 将 Z 时为 -S 或 IZ 时为 S 加到 D 中		D ± S = 0	有符号溢出 1	4
100111 001i 1111 dddddddd ssssssss SUMNZ D, S 将 S (如果 Z) 或 -S (如果 IZ) 加到 D 中			D ± S = 0	有符号溢出 1	4
101000 001i 1111 dddddddd ssssssss MOV	D, S 将 D 设置为 S		结果 = 0	编辑[31]	1 4
101001 001i 1111 dddddddd ssssssss 负	D, S 将 D 设置为 -S		结果 = 0	编辑[31]	1 4
101010 001i 1111 dddddddd ssssssss ABS	D, S 将 D 设置为绝对 S		结果 = 0	编辑[31]	1 4
101011 001i 1111 dddddddd ssssssss ABSNEG D, S 将 D 设置为 -absolute S			结果 = 0	编辑[31]	1 4
101100 001i 1111 dddddddd ssssssss NEGC	D, S 如果为 C, 则将 D 设置为 -S, 如果为 IC, 则将 D 设置为 S		结果 = 0	编辑[31]	1 4
101101 001i 1111 dddddddd ssssssss NEGNC D, S 如果为 C, 则将 D 设置为 S, 如果为 IC, 则将 D 设置为 -S			结果 = 0	编辑[31]	1 4
101110 001i 1111 dddddddd ssssssss NEGZ	D, S 将 D 设置为如果为 Z 则为 -S 或如果为 IZ 则为 -S		结果 = 0	编辑[31]	1 4
Z 则为 S 101111 001i 1111 dddddddd ssssssss NEGNZ	D, S 将 D 设置为如果为 Z 则为 S 或如果为 IZ 则为 -S		结果 = 0	编辑[31]	1 4
110000 000i 1111 dddddddd ssssssss CMPS	D, S 将有符号的 D 与 S 进行比较		D = S	有符号 (D < S)	0 4
110001 000i 1111 dddddddd ssssssss CMPSX D, S 将有符号扩展的 D 与 S+C 进行比较			Z & (D = S+C) 有符号 (D < S+C) 0		4
110010 001i 1111 dddddddd ssssssss ADDX	D, S 将 S+C 扩展至 D		Z & (D+S+C = 0) 无符号进位		1 4
110011 001i 1111 dddddddd ssssssss SUBX	D, S 从 D 中减去扩展的 S+C		Z & (D-(S+C)=0) 无符号借位 1		4
110011 000i 1111 dddddddd ssssssss CMPX	D, S 比较扩展 D 到 S+C		Z & (D = S+C) 有符号 (D < S+C) 0		4
110100 001i 1111 dddddddd ssssssss 添加	D, S 将有符号的 S 添加到 D 中		D + S = 0	有符号溢出 1	4
110101 001i 1111 dddddddd ssssssss SUBS	D, S 从 D 中减去有符号的 S		D - S = 0	有符号溢出 1	4
110110 001i 1111 dddddddd ssssssss ADDSX D, S 将有符号扩展的 S+C 添加到 D			Z & (D+(S+C) = 0) 有符号溢出 1		4
110111 001i 1111 dddddddd ssssssss SUBSX D, S 从 D 中减去有符号扩展的 S+C 111000 001i 1111 dddddddd ssssssss			Z & (D-(S+C)=0) 有符号溢出 1		4
CMPSUB D, S 如果 D => S, 则从 D 中减去 S			D = S	无符号 (D => S) 1	4
111001 001i 1111 dddddddd ssssssss DJNZ	d, s	减去 D, 如果不为零则跳转到 S (不跳转 = 8 个时钟)	结果 = 0	无符号借位 1 4 或 8	
111010 000i 1111 dddddddd ssssssss 新西兰	d, s	测试 D, 如果不为零则跳转到 S (无跳转 = 8 个时钟)	D = 0	0	0 4 或 8
111011 000i 1111 dddddddd ssssssss TJZ	D, S 测试 D, 若为零则跳转到 S (无跳转 = 8 个时钟)		D = 0	0	0 4 或 8
111100 000i 1111 dddddddd ssssssss WAITPEQ D, S 等待引脚相等 - (INA & S) = D 111101 000i 1111 dddddddd			-	-	0 6+
sssssssss WAITPNE D, S 等待引脚不相等 - (INA & S) != D 1111001i 1111 dddddddd ssssssss WAITCNT D, S 等待 CNT =			-	-	0 6+
D, 然后将 S 添加到 D 中 111111 000i 1111 dddddddd ssssssss WAITVID D, S 等待视频外设抓取 D 和 S			-	无符号进位	1 6+
----- 0000 ----- NOP		无需操作, 仅经过 4 个时钟	-	-	0 4+3

*请参阅 Hub, 第 7 页第 4.4 节。

1. ADDABS C 输出: 如果 S 为负, 则 C = 无符号借位的倒数 (对于 DS)。

2. SUBABS C 输出: 如果 S 为负, 则 C = 无符号进位的倒数 (对于 D+S)。

3. WAITVID 本身消耗 4 个时钟; 但是, 完成数据切换需要帧 7 个时钟 (某些频率下为 6 个)。

CTRPLL 频率和 VSCL FrameClocks 的组合必须提供有效的 7 个 (或 6 个) 系统时钟。

6.4.1. 装配条件

健康)状况	指令执行
如果始终	总是
如果_不	绝不
如果	如果相等 (Z)
如果	如果不等于 (!Z)
如果	如果高于 (IC & !Z)
如果_B	如果低于 (C)
IF_AE	如果大于/等于 (IC)
如果是	如果低于/等于 (C Z)
如果	如果 C 集合
如果未指定	如果 C 清楚
IF_Z	如果 Z 设定
新西兰	如果 Z 清除
如果是 C_EQ_Z	如果 C 等于 Z
如果	如果 C 不等于 Z
如果 C 与 Z	如果 C 设置且 Z 设置
如果 C 和 NZ	如果 C 设置并且 Z 清除
如果_NC_AND_Z	如果 C 清除并且 Z 设置
如果是 NC 和 NZ	如果 C 清晰且 Z 清晰
如果 C_OR_Z	如果 C 设置或 Z 设置
如果 C_OR_NZ	如果 C 设置或 Z 清除
如果是NC或Z	如果 C 清除或者 Z 设置
如果是 NC 或 NZ	如果 C 清除或 Z 清除
如果是 Z 的话	如果 Z 等于 C
如果	如果 Z 不等于 C
如果是 Z 则返回	如果 Z 设置且 C 设置
如果是 Z 则返回 NC	如果 Z 设置并且 C 清除
如果是 NZ_AND_C	如果 Z 清除并且 C 设置
如果是 NZ_AND_NC	如果 Z 清晰且 C 清晰
否则	如果 Z 设置或 C 设置
否则	如果 Z 设置或 C 清除
否则	如果 Z 清除或者 C 设置
如果是 NZ_OR_NC	如果 Z 清除或 C 清除

6.4.2. 汇编指令

指示	描述
FIT 地址	验证先前的指令/数据是否适合地址下方。
ORG地址	调整编译时 cog 地址指针。
符号 RES计数	为符号保留下一个长整型。

6.4.3. 组装效果

效果结果	
厕所	C 标志已修改
无锡	Z 标志已修改
西弗	目标寄存器已修改
天然橡胶	目标寄存器未修改

6.4.4. 汇编运算符

Propeller 汇编代码可以包含常量表达式,可以使用常量表达式中允许的任何运算符。下表 (表 17 的子集)列出了 Propeller 汇编中允许的运算符。

操作符描述	
+	添加
+	正数 (+X);Add 的一元形式
-	减去
-	取反 (-X);减法的一元形式
*	相乘并返回低 32 位 (有符号)
**	相乘并返回高 32 位 (有符号)
/	除以 (有符号)
//	模数 (有符号)
#>	限制最小值 (有符号)
<#	限制最大值 (有符号)
^^	平方根;一元
	绝对值;一元
~>	算术右移
<	按位:将值 (0-31)解码为单高位长整型;一元
>	按位:将长整型编码为值 (0 - 32) ,作为高位优先级;一元
<<	按位:左移
>>	按位:右移
<-	按位:左旋转
->	按位:右移
><	按位:反转
&	按位:AND
	按位:或
^	按位:XOR
!	按位:非;一元
和	布尔值:AND (将非 0 提升为 -1)
或者	布尔值:OR (将非 0 提升为 -1)
不是	布尔值:NOT (将非 0 提升为 -1) ;一元
==	布尔值:相等
<>	布尔值:不等于
<	布尔值:小于 (有符号)
>	布尔值:大于 (有符号)
=<	布尔值:等于或小于 (有符号)
=>	布尔值:等于或大于 (有符号)
@	符号地址;一元

7.0 电气特性

7.1. 绝对最大额定值超过绝对最大额定值的应用

力可能会对器件造成永久性损坏。这些只是绝对应力额定值。在这些或任何其他超过第 0 节其余部分给出的条件的情况下,器件的功能操作并不隐含在内。长时间暴露在绝对最大额定值下可能会对器件的可靠性产生不利影响。

表 18:绝对最大额定值	
偏置下的环境温度	-55 °C 至 +125 °C
存储温度	-65 °C 至 +150 °C
Vdd相对于Vss的电压	-0.3 V 至 +4.0 V
所有其他引脚相对于Vss 的电压	-0.3 V 至(Vdd + 0.3 V)
总功率耗散	1 瓦
Vss引脚的最大电流	300 毫安
流入Vdd引脚的最大电流	300 毫安
带内部保护二极管正向偏置的输入引脚的最大直流电流	±500 μA
每个 I/O 引脚的最大允许电流	40 毫安
ESD (人体模型)电源引脚	3千伏
ESD (人体模型)所有非电源引脚	8千伏

*注意:如果没有超过内部保护二极管正向偏置电流,则相对于 V_{SS} 的 I/O 引脚电压可能会超过上限。

7.2. 直流特性

(工作温度范围: $-55^{\circ}\text{C} < \text{Ta} < +125^{\circ}\text{C}$ 除非另有说明)

象征	范围	状况	分钟	类型*	最大限度	单位
电压	电源电压		2.7	-	3.6	五
威,威	逻辑高 逻辑低		0.6电压 电压		电压 0.3电压	五 五
输入	输入漏电流	$Vin = Vdd$ 或 Vss	-1.0		+1.0	安培
沃	输出高电压	$Ioh = 10$ mA, $Vdd = 3.3$ V	2.85			五
卷	输出低电压	$Iol = 10$ 毫安, $Vdd = 3.3$ 伏			0.4	五
欠压检测器	欠压检测器电流			3.8		安培
·	静态电流	$RESn = 0V$, $BOEn = Vdd$, $P0-P31=0V$		600		毫安

*注:除非另有说明,典型值(“Typ”列中的数据为Ta = 25 °C)。

7.3. 交流特性 (工作温度范围: -55°C

$< T_a < +125^\circ\text{C}$ 除非另有说明)

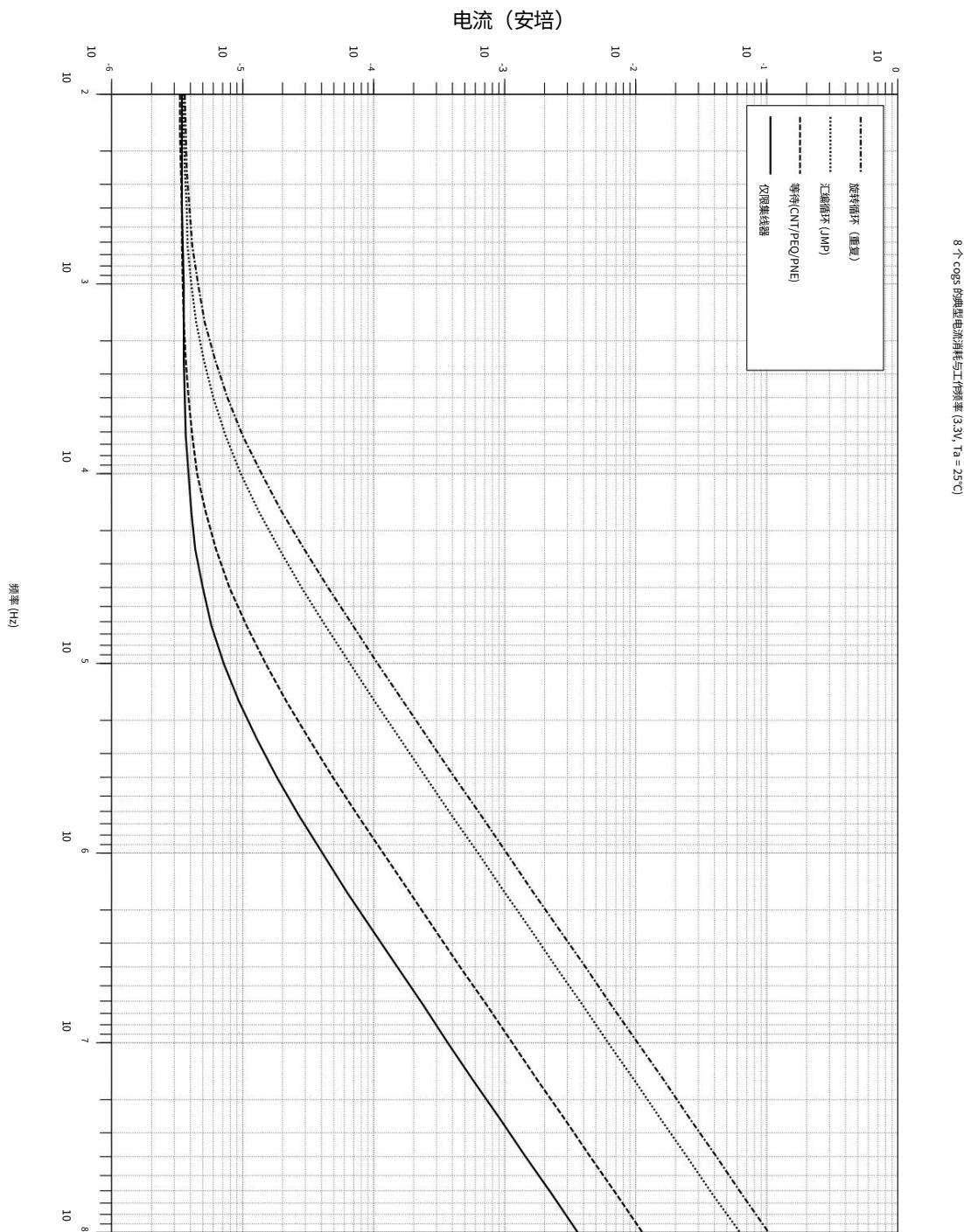
象征	范围	分钟	类型*	最大限度	单位	健康)状况
福斯克	外部 XI 频率	直流	-	80	MHz	
	振荡器频率	直流 13 8 4	- 20 12 -	80 33 20 8	兆赫 千赫 MHz MHz	直接驱动 (无 PLL) 慢速 射频快速接头 采用PLL的晶体
辛	输入电容		6	-	毫微法	

*注:除非另有说明,典型值(“Typ”列中的数据为 $T_a = 25^\circ\text{C}$.)

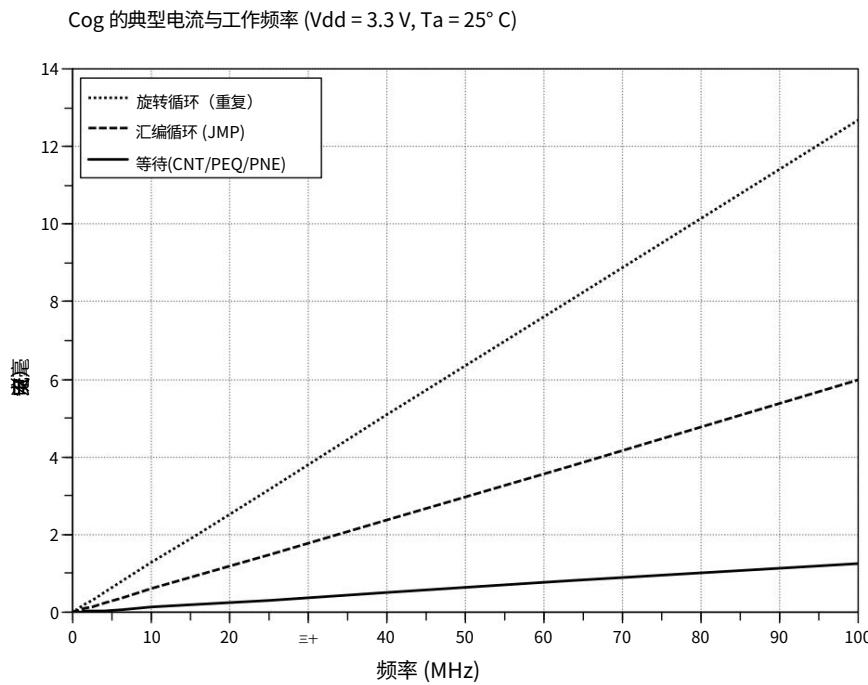
8.0 电流消耗特性

8.1. 8 个齿轮的典型电流消耗该图显示了螺旋桨在所有齿轮上

重复的各种操作条件下的典型电流消耗。在测试期间,电压不足电路和锁相环被禁用。电流消耗在工作温度范围内基本恒定。

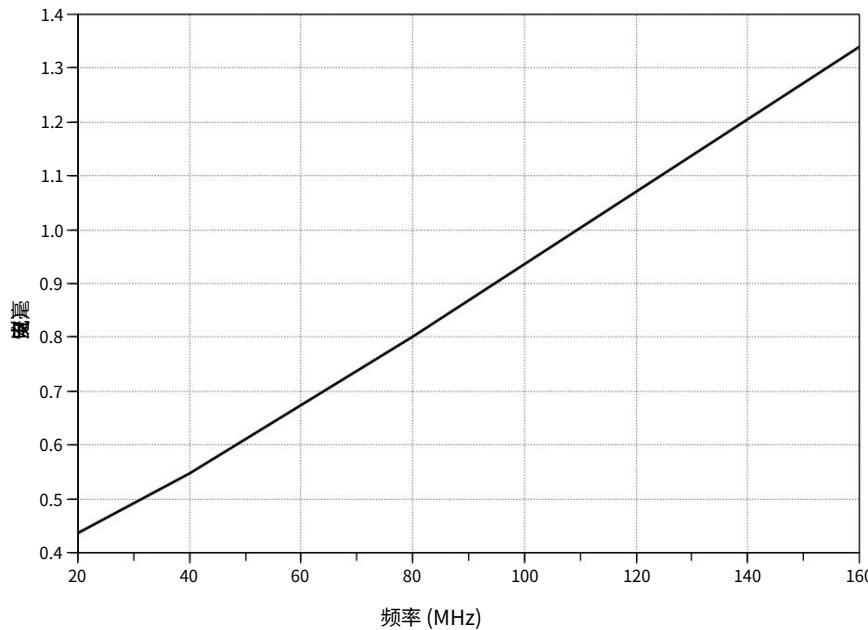


8.2. Cog 的典型电流与工作频率该图显示了在各种条件下,与 Propeller 芯片内的其他电流源隔离的 Cog 的典型电流消耗。



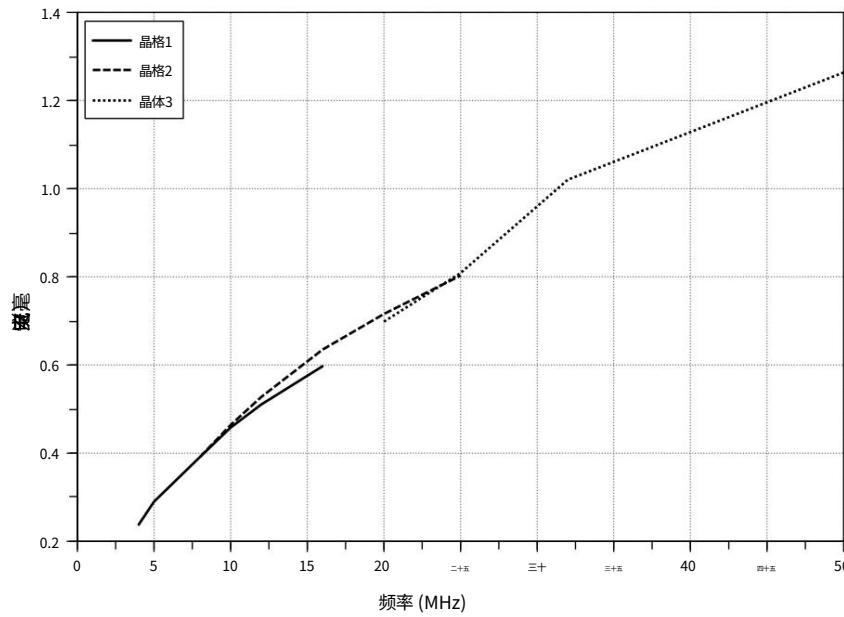
8.3. 典型 PLL 电流与 VCO 频率该图显示了锁相环所消耗的典型电流量，它是压控振荡器频率的函数，该频率是输入时钟频率的 16 倍。

典型 PLL 电流与 VCO 频率的关系 ($Vdd = 3.3$ V, $Ta = 25^\circ$ C)



8.4. 典型晶体驱动电流该图显示了晶体驱动器在一定

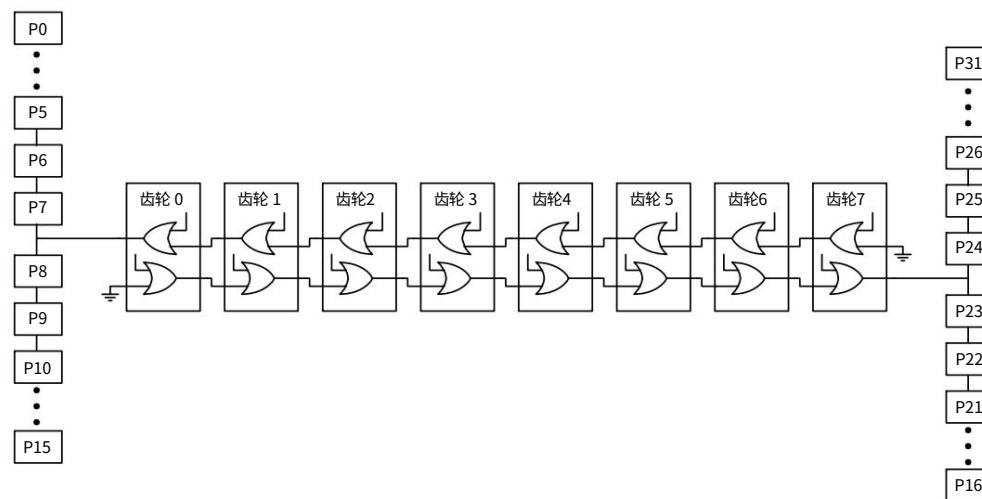
范围的晶体频率和晶体设置下的电流消耗,所有高于 25 MHz 的数据点都是通过使用谐振器获得的,因为驱动器不执行高于 25 MHz 的晶体所需的第一三谐波泛音驱动。

典型晶体驱动电流 ($V_{dd} = 3.3$ V, $T_a = 25^\circ$ C)

8.5. Cog 和 I/O 引脚关系下图说明了 cog 和 I/O 引脚

之间的物理关系。虽然最短路径和最长路径之间的输出转换可能存在 1 到 1.5 ns 的传播延迟,但该图的目的是说明引线的长度及其相关的寄生电容。此电容增加了转换引脚状态所需的能量,因此增加了切换引脚的电流消耗。因此,Cog 7 在 20 MHz 下切换 P0 所消耗的电流将大于 Cog 0 在 20 MHz 下切换 P7 所消耗的电流。转换引脚状态所消耗的电流量取决于许多因素,包括:温度、转换频率、外部负载和内部负载。

如上所述,内部负载取决于所用的齿轮和引脚。对于采用 DIP 封装的 Propeller,在 20 MHz 下切换引脚时,室温下的内部负载电流约为 300 μ A。



8.6. 各种启动条件下的电流分布下图显示了 Propeller 芯片在存在 EEPROM 和 PC 的情况下各种启动条件下的电流分布。

图 9

启动顺序当前配置文件

没有 PC 也没有 EEPROM (P31 保持低位且 P29 未连接
(与保持低位相同))。

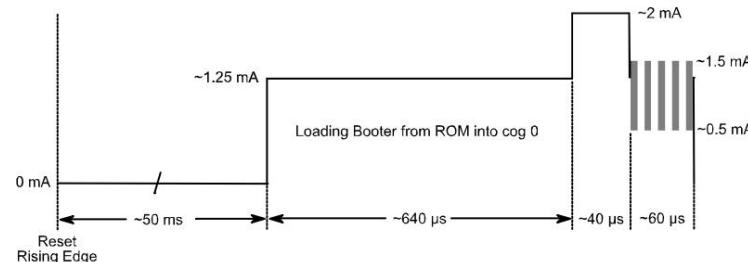


图 10

启动顺序当前配置文件
PC (已连接但空闲)且无
EEPROM。 (P31 保持高电平,
P29未连接)。

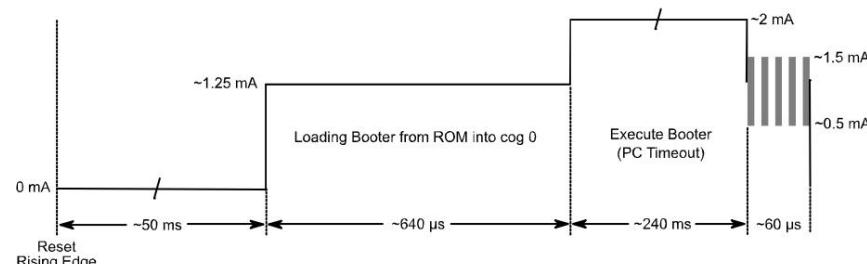


图 11

启动顺序当前配置文件
没有 PC 也没有 EEPROM (P31
保持低位而 P29 保持高位)。

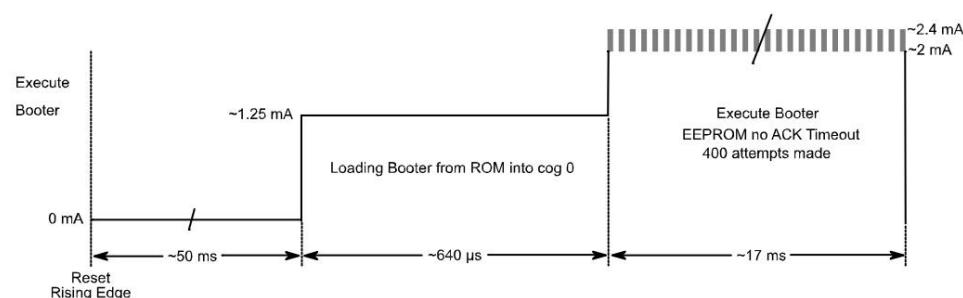


图 12

启动顺序当前配置文件
没有 PC 和 EEPROM (P31 保持
低且 P29 连接至 EEPROM SDA)。

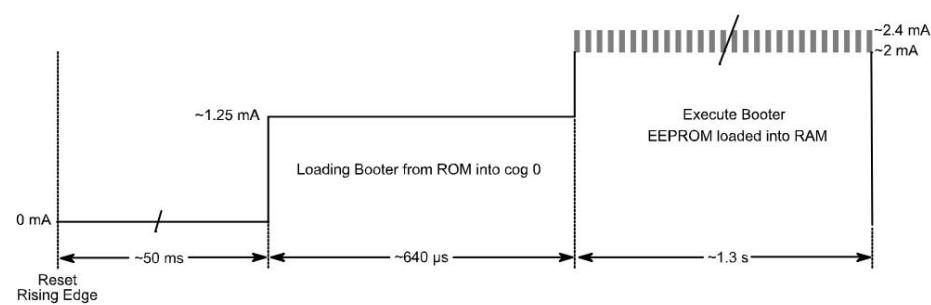
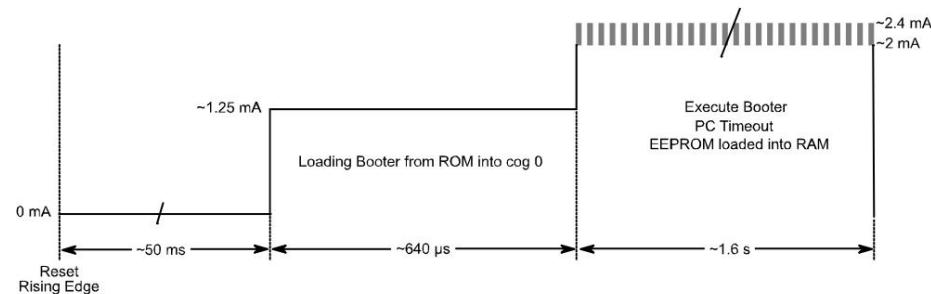


图 13

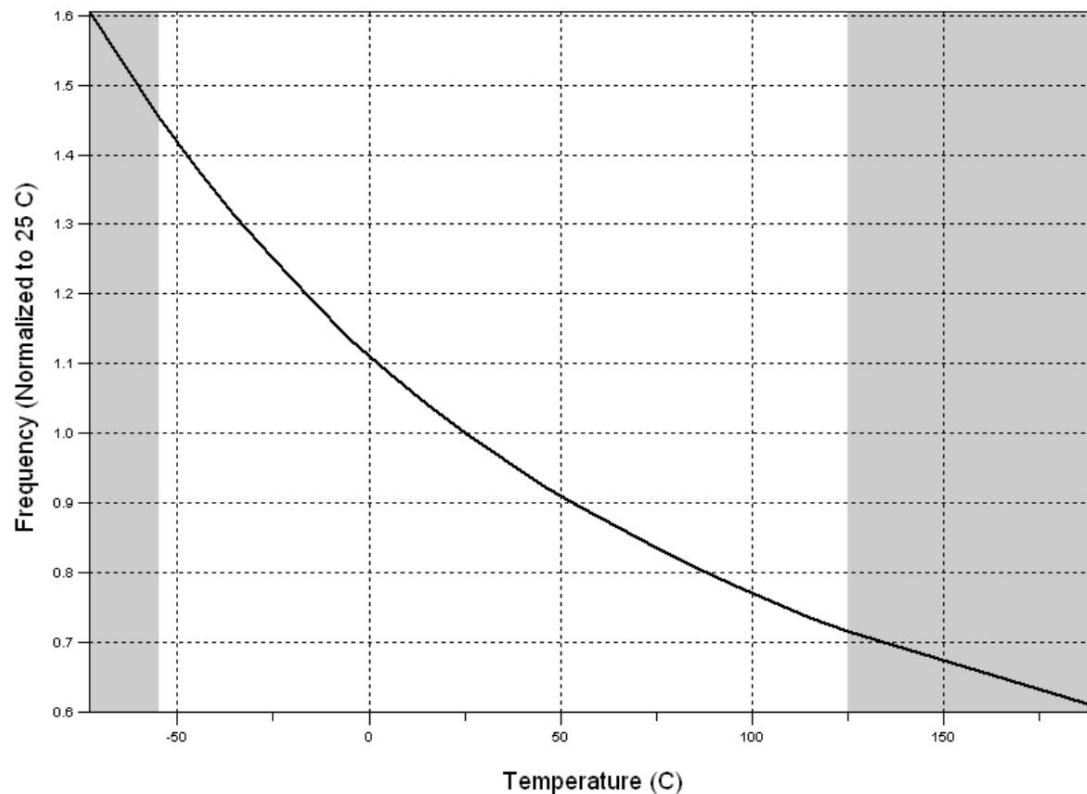
启动顺序当前配置文件
PC (已连接但空闲)和
EEPROM (P31 保持高位,P29
连接至EEPROM SDA)。



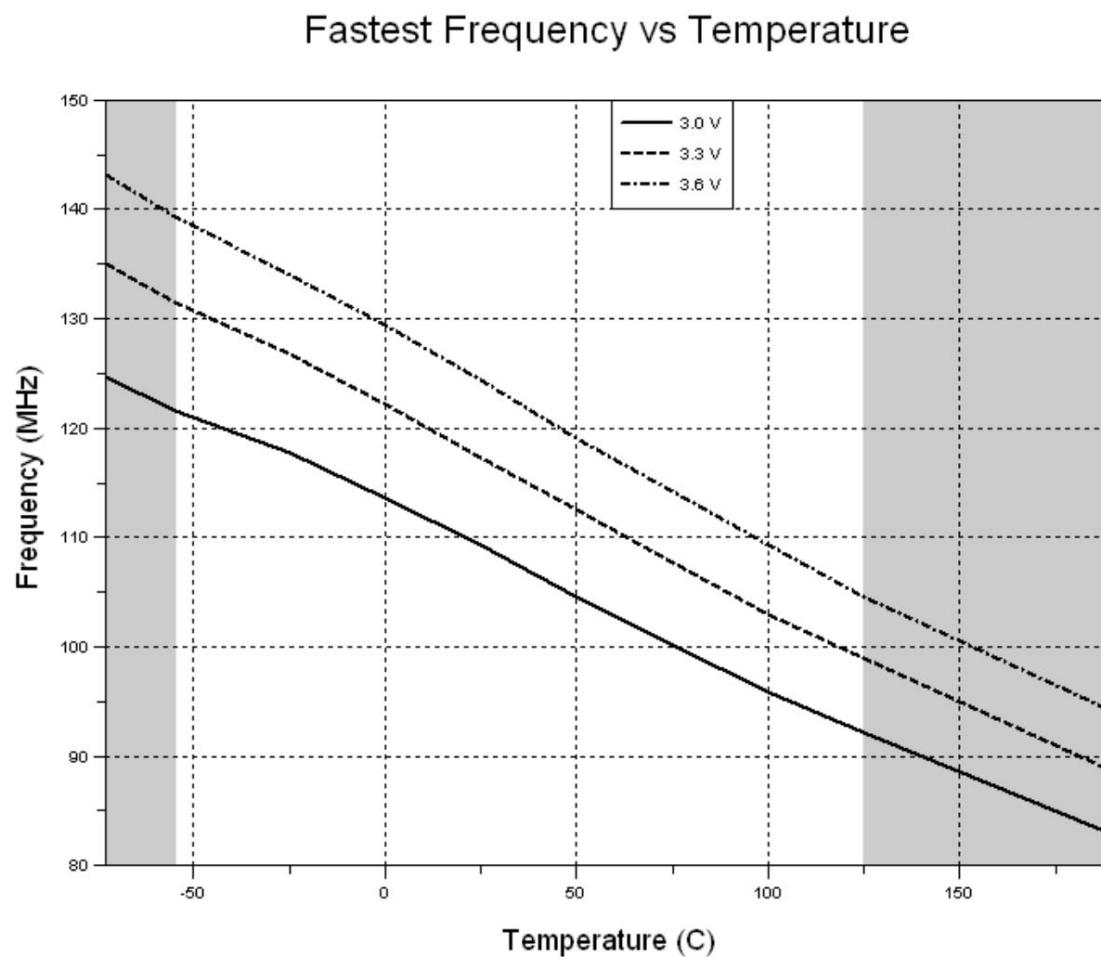
9.0 温度特性

9.1. 内部振荡器频率与温度的关系
虽然内部振荡器频率会因工艺变化而变化,但标准化后的温度函数变化率提供了芯片不变比率,可用于计算环境温度不是 25 °C (图形标准化的温度)时的振荡频率。25 °C 时的绝对频率在样本集中从 13.26 到 13.75 MHz 不等。图形中具有白色背景的部分是军用温度范围;灰色部分表示超出军用温度规范的数据。

RCFAST Normalized Frequency vs Temperature

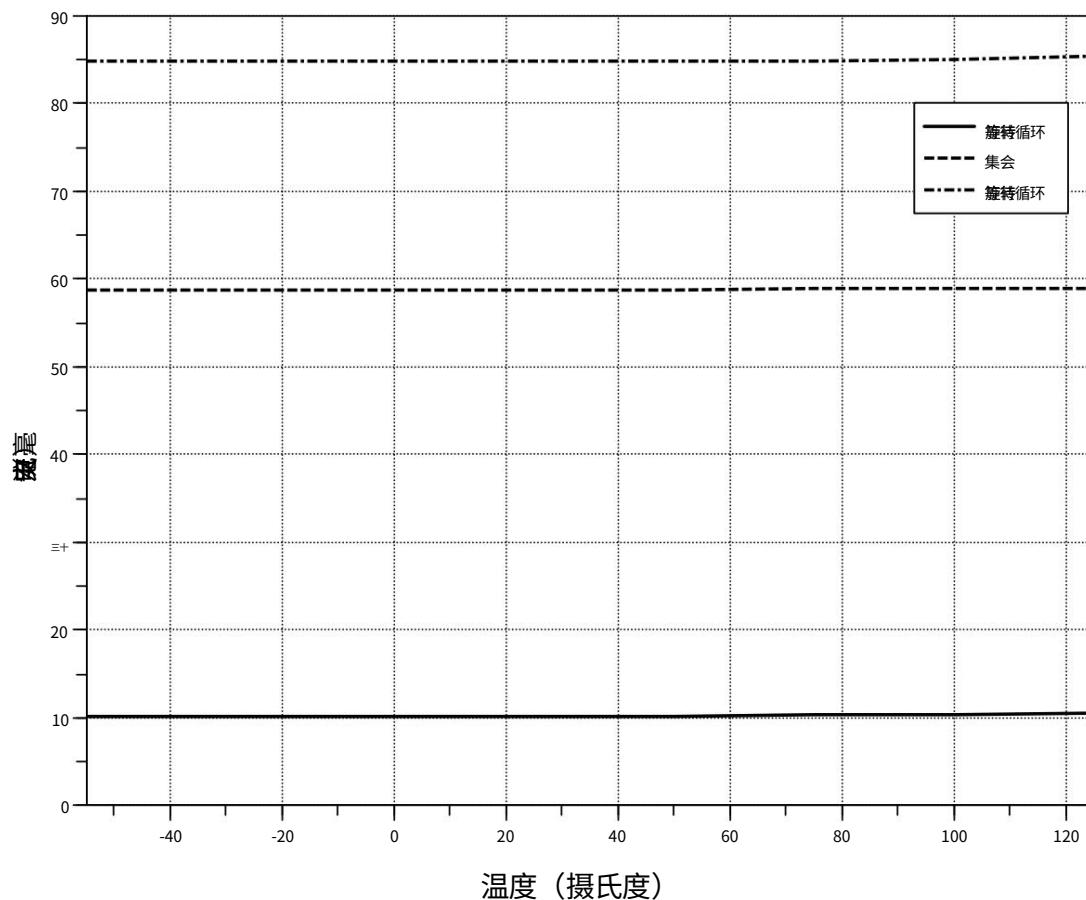


9.2. 最快工作频率与温度的关系下图表示 Propeller 芯片最快工作范围的小样本平均值。测试在强制空气室中进行,使用在所有八个齿轮、多个视频发生器和计数器模块上运行的代码。如果演示无故障运行一分钟,则认为频率成功。曲线表示激进的测试程序 (平均、强制空气、一分钟时间限制);因此,设计人员必须降低曲线以获得特定应用的稳定频率。同样,灰色区域表示超出军用温度范围的温度。



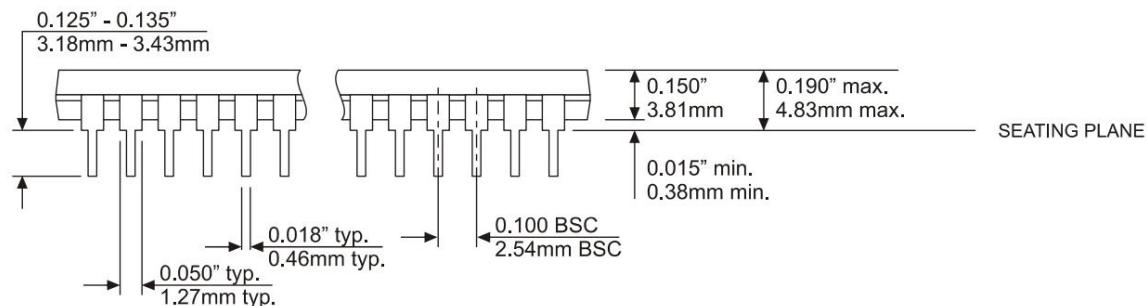
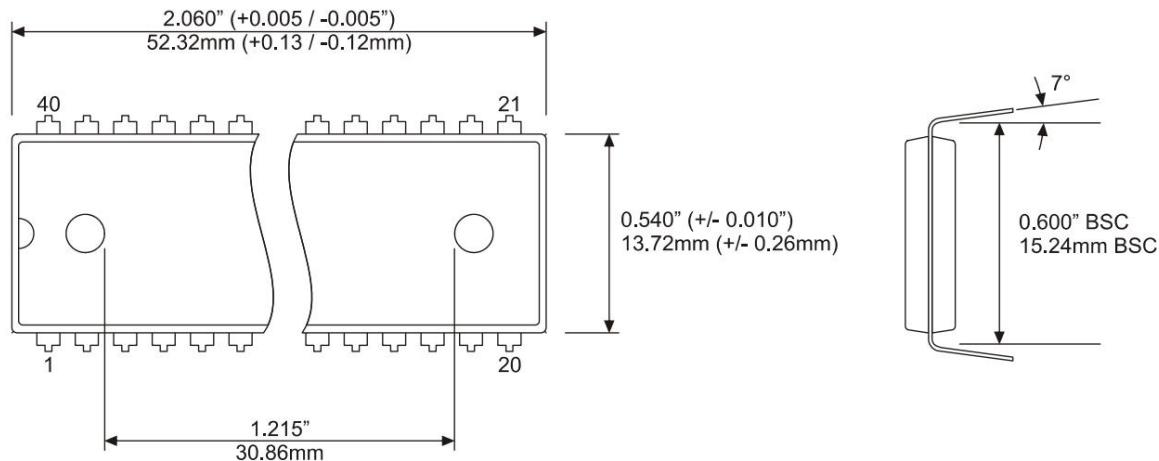
9.3. 电流消耗与温度的关系
下图显示了 Propeller 的电流消耗与温度的关系。从图中可以清楚地看出,在整个军用温度范围内,电流消耗几乎与温度无关。

电流消耗与温度

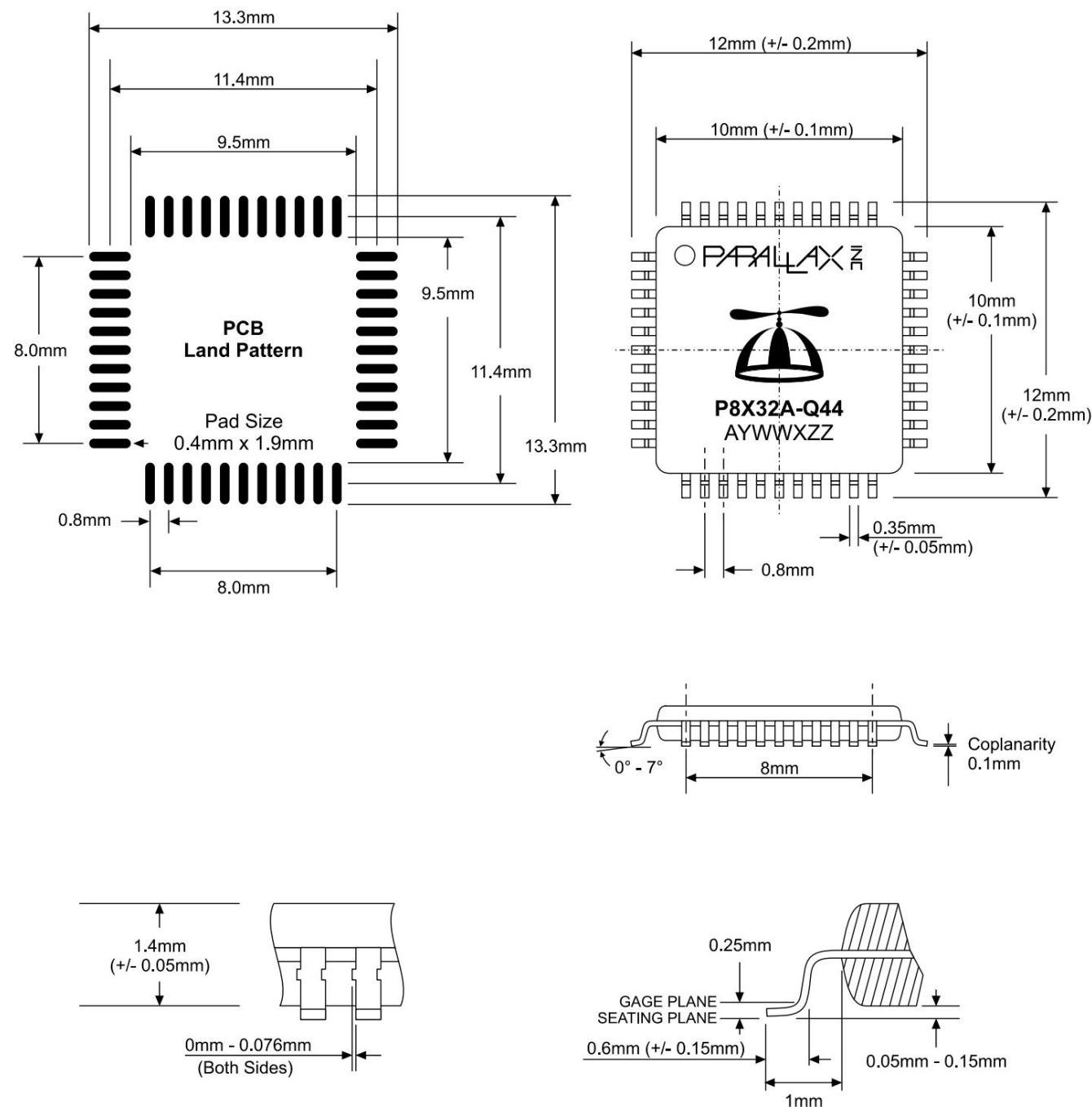


10.0 包装尺寸

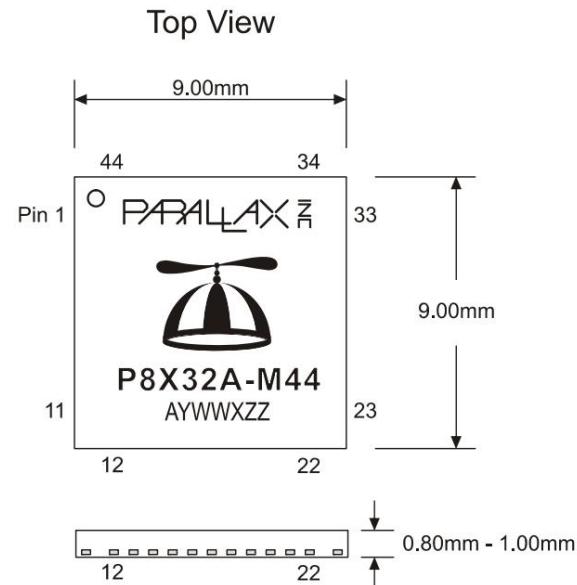
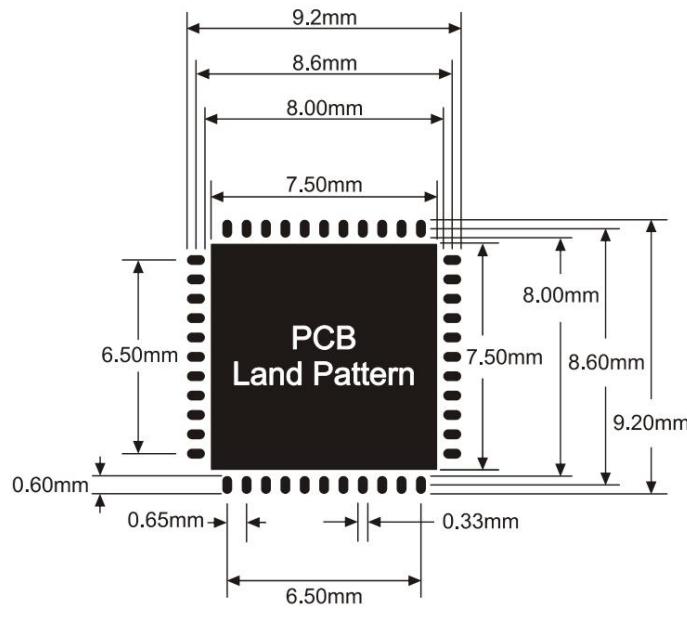
10.1. P8X32A-D40 (40 针 DIP)



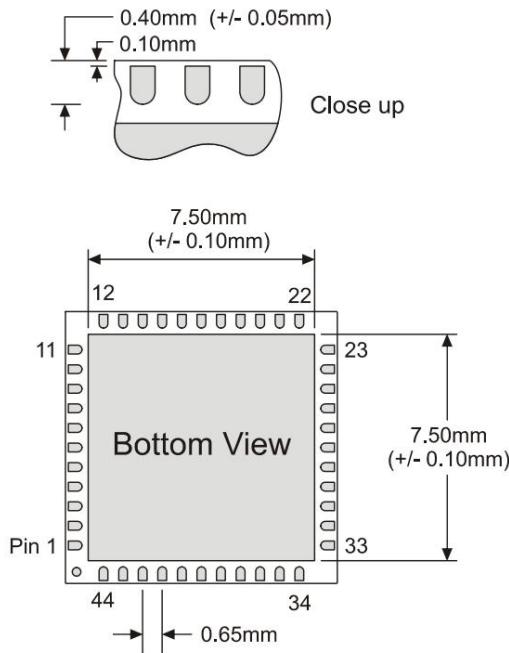
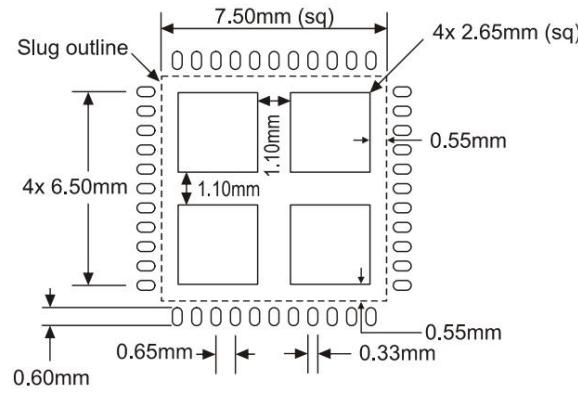
10.2. P8X32A-Q44 (44 引脚 LQFP)



10.3. P8X32A-M44 (44 引脚 QFN)



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



11.0 制造信息

11.1. 回流峰值温度

封装类型	回流峰值温度
蘸	255+5/-0 摄氏度
限制双面扁平无引线	255+5/-0 摄氏度
扁平无引线	255+5/-0 摄氏度

11.2. 绿色/RoHS 合规性所有 Parallax

Semiconductor Propeller P8X32A 芯片型号均经过绿色/RoHS 认证。RoHS、绿色和 ISO 证书可在 www.parallaxsemiconductor.com 上在线获取。

12.0 修订历史

12.1.1. 版本 1.1 的变更:第 10.3 节:

P8X32A-M44 (44 引脚 QFN)。图片已替换,以添加模板图案图。插入新章节:4.8 汇编指令执行阶段。

联系信息已更新。

12.1.2. 版本 1.2 的变更:第 6.4 节:修改了

ADD、ADDABS、ADDs、ADDsX、ADDx、CMP、CMPS、CMPSX、CMpX、COGID、COGINIT、COGSTOP、LOCKCLR、LCOKEW、LOCKRET、LOCKSET、MAX、MAXs、MIN、MINS、SUB、SUBABS、SUBs、SUBSX、SUBX、SUMC、SUMNC、SUMNZ、SUMZ、TEST、TJNZ、TJZ 的表格条目。第 4.5 节

已更新。第 5.1 节:在段落末尾添加了新句子。第 5.2 节:在第一段末尾添加了新句子。

12.1.3. 版本 1.3 的变更全文:更新了

Parallax Inc. (商名为 Parallax Semiconductor) 的徽标和联系信息。第 7.1 节:表 18:绝对最大额定值中添加了脚注。

12.1.4. 版本 1.4 的变更1.0 节变更: 1.3:

主要功能和优点

修订;删除了以前的 1.4 1.6 节。第 4.4 节:更新了所有有关集线器时序的参考资料,并替换了两个时序图。第 4.8 节:更新了有关集线器时序的参考资料。第 6.4 节:修订了集线器指令和 WAITxxx 指令的时序。以前的第 7.0 节:删除了螺旋桨演示板原理图。

Parallax Semiconductor 联系信息

视差半导体

门洛大道 599 号

罗克林,加利福尼亚州 95765

美国

电话:(916)632-4664

传真:(916) 624-8003

sales@parallaxsemiconductor.com

support@parallaxsemiconductor.com

www.parallaxsemiconductor.com

<http://obex.parallax.com>

Parallax, Inc. (dba Parallax Semiconductor) 不对其产品是否适用于任何特定用途作任何保证、陈述或担保,也不承担因应用或使用任何产品而产生的任何责任,并明确否认任何及所有责任,包括但不限于间接或附带损害,即使 Parallax, Inc. (dba Parallax Semiconductor) 已被告知此类损害的可能性。未经 Parallax, Inc. (dba Parallax Semiconductor) 事先书面同意,禁止全部或部分复制本文件。

版权所有 © 2011 Parallax, Inc. dba Parallax Semiconductor。保留所有权利。

Propeller 和 Parallax Semiconductor 是 Parallax, Inc. 的商标。