TwinCAT 控制任务的 CPU 耗时统计与分析

Version 1.0

倍福广州 陈利君 2018-06-01

适用范围:

常常有人问，CX9020 到底能带几个轴？

有人推荐 4 个轴以内，也有人说用过 CX9020 带了 8 个轴。同样的问题也可以针对任何一款倍福控制器。每次我都会这样解释：

如果带“轴”只要求能控制伺服或者步进电机动起来，实际上只用 PLC 也可以做到，因为驱动器本身就有定位功能，PLC 只要告诉它走若干个脉冲当量，并触发它运动就可以了。这个意义上说，任何一款 CX 控制器都可以带“无限”轴：借助 EtherCAT，一个驱动器就是一个从站，一条 EtherCAT 网络可以带 65535 个从站。此外，一台控制器上可以有多条 EtherCAT 网络，如果控制器集成的是千兆网卡，还可以利用 CU2508 倍增器，驱动 8 条 EtherCAT！

狭义上讲，工程师说起“带轴”的能力，通常是指 Motion Control 这个范畴。那么使用 Motion Control，CX9020 到底能带几个轴呢？倍福每一款 CX 控制器分别能带几个轴呢？

理论上，任何一台带 NC 功能的倍福控制器都可以配置 255 个轴。

但是能配置不等于能运行！实际上，能带几个轴并且正常工作，需要考虑的因素包括：

1) CPU 的运算能力，以及单核还是多核
2) 操作系统：CE 还是 WES
3) TwinCAT 版本：TC2 还是 TC3
4) NC 任务的控制周期
5) PLC 任务的运算量，除了 PID 运算，是否大量读写文件或者 Novram
6) 控制器上是否运行 TwinCAT PLC HMI
7) 控制器上是否运行第三方应用程序
8) 轴与轴之前的联动关系：速度耦合还是位置耦合，还是组成插补通道，有无坐标变换
9) 系统中有无时间敏感的任务，比如低于 1ms 的任务，且个别从站之间要求时钟同步
10) 系统中有无某种后台运行的 Server，以及来自 Client 的访问量如何，比如：
    TcpIp Server，OPC Server，Modbus Tcp Server 等

既使这 10 个因素都考虑到了，也只有定性的趋势而没有定量的公式来计算某个控制器到底能带多少轴。最后还是需要有经验的销售或者技术人员，通常其它项目经验和本项目的实际需求，以及控制器性能对照表，估算一款适合的控制器。通常样机开发的时候估算会偏保守，就是根据经验选中一款控制器之后，再上浮一个 CPU 主频级别。等样机试制完成，再根据实际的 CPU 利用率，重新选择适合的控制器。

为什么倍福运动控制器的可带轴数量这么难以计算呢？

这是由 PC-Based 控制的技术路线决定的，因为倍福的运动控制器——TwinCAT NC 并不是一个独立的硬件，而是与 PLC、HMI 和其它应用程序共享一个 CPU。虽然基于分时多任务的技术，NC 的优先级比其它任务要高，但是如果只有 NC 正常运行而 PLC 没有足够的资源来执行，或者控制器上的 HMI 画面反应迟钝，用户都不会接受。而 PLC 的程序量有多大、控制器上装了多少第三方应用都是不可限量的。NC 的控制周期本身，针对不同的应用也可以设置为 250us 到 5ms 之间的值，变化倍数达到 20 倍，所以倍福控制器到底能带多少轴，永远是个 case by case 的问题。

对比第三方的运动板卡或者运动控制器，因为是独立的硬件——包括 CPU、内存和 IO 接口，

其繁忙程度与 PLC 和 HMI 互不影响。另外，运动控制的最大计算量是基本恒定的，所以一套硬件最多带几个轴是确定的，厂商就可以在后台软件里固定某个板卡或者运动控制模块最多能带几个轴。

对倍福用户来说，有意义的问题不是它的运动控制系统能带几个轴，而是整套系统——包括 PLC、NC、HMI 和所有后台服务——带几个轴能正常运行，并且满足项目的生产速度和精度。所以有经验的工程师在选型的时候，会考虑前面所讲到的 10 个因素。

为了更加准确地估算合适的 CPU，或者说更准确地预测 CPU 利用率，本文基于 CX 控制器和 EtherCAT 的大量测试数据，统计不同 CX 控制器上的 TwinCAT 任务消耗 CPU 时间，分析这些时间的规律，结合 TwinCAT 和 EtherCAT 的底层机制，试图破解 CPU 时间消耗的秘密。

1. TwinCAT 的 CPU 利用率构成

TwinCAT 的 CPU 利用率，取决于它的每个任务消耗的 CPU 时间和任务周期。假定系统有两个任务——PLC 任务和 NC 任务

平均的 CPU 利用率＝PLC 任务耗时/PLC 任务周期+NC 任务耗时/NC 任务周期

最大 CPU 利用率＝（PLC 任务耗时+ NC 任务耗时）/NC 任务周期

最小 CPU 利用率＝ NC 任务耗时 / NC 任务周期

虽然低优先级的任务不会影响到高优先级的任务，但是在最大 CPU 利用率发生的瞬间，低优先级的任务和其它非实时任务可能会受到影响，特别要注意那些驱动分布时钟的任务。

2. Task 消耗 CPU 时间的构成

Task 消耗的 CPU 时间包括“计算”和“IO 刷新”

比如：

IO Additional Task 消耗的时间，只有 IO 刷新。

PLC Task 消耗的时间，包括 PLC Task 中显示的 Last Execute time，和 IO 刷新时间

NC Task 消耗的时间，包括 NC 发生器及单位换算、总线协议处理，和 IO 刷新的时间

因为 IO 刷新时由任务触发的，所以不同任务的 IO 刷新时间是独立的。以 EtherCAT 为例，通常情况下，多个任务会触发多个 EtherCAT Frame，每个 Frame 都要花费独立的 CPU 处理时间。

3. IO Additional Task 的耗时统计

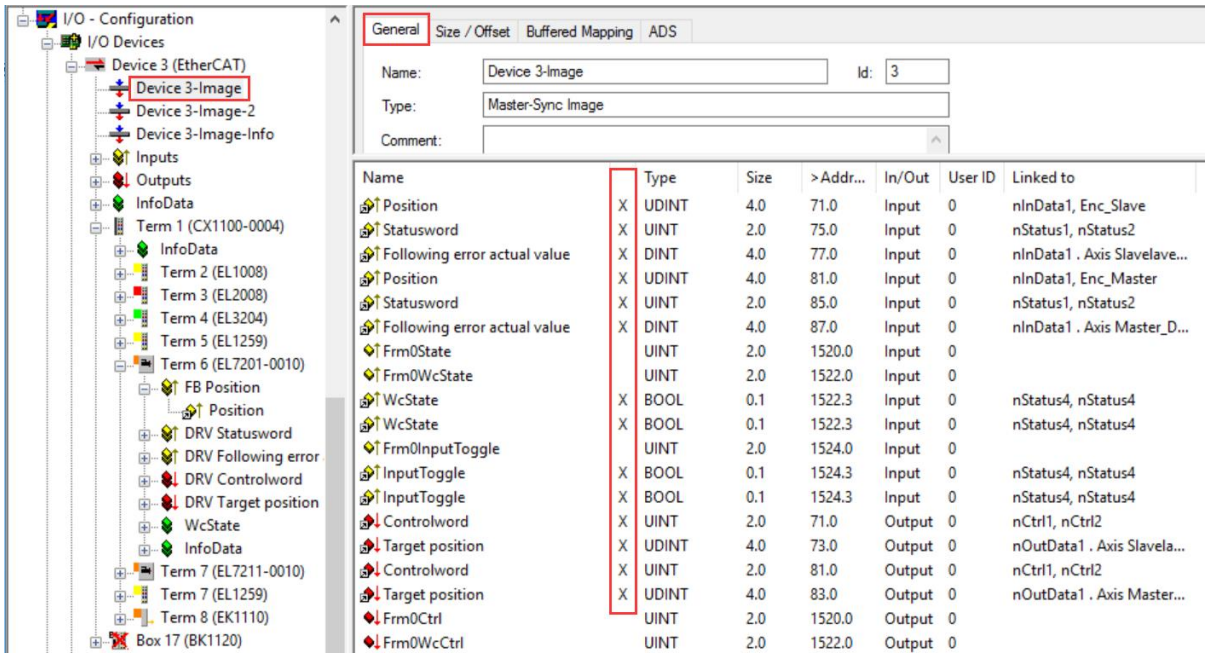
IO 任务所消耗的时间，都用来刷新 EtherCAT。刷新 EtherCAT 时，CPU 会把 TwinCAT 任务的输出变量的地址中的值写入到控制器的内存中分配给 EtherCAT 网卡专用的地址区间，并从该区间某些地址读取数据赋给相应的输入变量。

接下来，EtherCAT 网卡中的 DMA(直接内存访问)芯片检测到发送触发信号后会把这些数据组合进规则的 EtherCAT 数据帧并发送出去。但这一部份工作不消耗控制器的 CPU。

3.1. CPU 内存复制的依据

CPU 内存复制的依据是 TwinCAT 配置时指定的映射关系。

任务刷新 IO 的时候，重要的是这两个内存地址的对应，这就是变量映射。类似的变量映射还有很多，所有的映射都可以从 EtherCAT 的 Image 中观察：



内存复制的规则

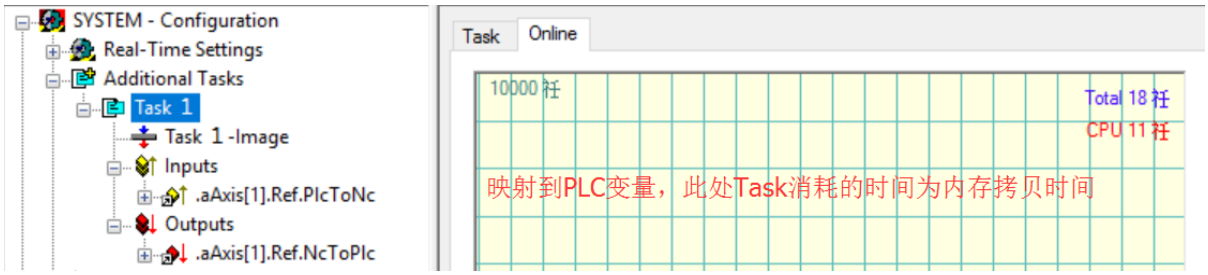
规则 1：如果任务中没有变量映射到 EtherCAT 地址，则 CPU 不需要内存复制，该任务不会消耗 IO 刷新时间。

规则 2：如果所有任务中只有 1 个变量映射到 EtherCAT 地址，EtherCAT 都会创建 Frame，并在该 Frame 中包含所有 IO 从站的数据。这个任务消耗的 IO 刷新时间就是刷新所有从站的时间。

规则 3：如果第二个任务有变量映射到某个从站，则创建另一个 Frame，并在该 Frame 中包含所映射从站的数据。

3.2. IO 刷新时间的统计

Frame 的长度与 IO 数据大小正相关，测试数据时只记录 Frame 长度、控制器与 IO 刷新时间的关系。记录的数据取自以下界面：



由于 IO 刷新时间，也是影响 CPU 利用率的一个重要部分，同系列的控制器，IO 刷新时间直接正比于 CPU 主频。所以实际项目中可以根据下面的数据来估算 IO 刷新时间，并根据任务周期计算出 CPU 利用率。

AdditionalTask IO 刷新	EL3702 等超采样模块，数据量大，但节点数不多			
	数据量/时间	数据量/时间	数据量/时间	数据量/时间
CX2020,TC3,WES7 EK1200， EBus	43Byte/12us 100Byte/11.5us			1250Byte/14.5us

CX2020,TC3, 2 条 EtherCAT			748Byte+100Byte 20us	
CX2020,TC3, 前面的网口		543Byte/15us	748Byte/15us	1217Byte/16us 末站丢失: 16-19us
CX5130,TC3,WES7 EK1200， Ebus	63Byte/29-43us 107Byte/28-37us			1217Byte/34- 50us
CX5130,TC3,WES7 2 条 EtherCAT		543Byte/37.4- 54.5us	748Byte+100Byte 记漏	
CX5130,TC3,WES7 前面的网口	CX5130 普遍数据波动，可能是 PLC 或者 NC 任务没有清空，重试		748Byte/36-55us	1217Byte/38.3us 末站丢失: 39-55us
CX1020,TC2,CE 内部 Ebus	43Byte/75us	341Byte/100us	681Byte/110us	1077Byte/135us
CX1010,TC2,CE 内部 Ebus	43Byte/150us	341Byte/192us	689Byte/206us	1089Byte/230us
CX5020,TC2,CE 前面的网口	82Byte/95us	334Byte/103us	736Byte/121us	1072Byte/131us
CX9020,TC2,CE 内部 Ebus	82Byte/91us	334Byte/104us	736Byte/140us	1072Byte/160us

3.3. IO 刷新时间估算示例

假定：CX2030 的 PLC 任务周期 10ms，Frame 长度 200Byte，NC 任务周期 2ms，Frame 长度 500Byte.

估算：
依据 CX2020，543Byte，15us 来估算：
平均 CPU 利用率为：15/2000+15/10000＝1.5%+0.15%=1.65%
CX2020 的主频 1.46G ，CX2030 的主频 1.5G，所以：
CX2030 的 CPU 利用率为：1.65/1.5*1.46=1.61%

假定：CX9020 带这些 EtherCAT 从站，
估算：
依据 CX9020，按 334 Byte，104us 来计算
平均 CPU 利用率为：104/2000+104/10000＝5.2%+1.04 %=6.24%
如果 NC 周期 1ms，则
平均 CPU 利用率为：104/1000+104/10000＝10.4%+1.04 %=11.44%

为了同系列 CPU 的利用率互相转换，特摘录各系列的 CX 控制器时钟主频如下：

Technical data	CX2020	CX2030	CX2040
Processor	Intel® Celeron® 827E 1.4 GHz, 1 core (TC3: 50)	Intel® Core™ i7 2610UE 1.5 GHz, 2 cores (TC3: 60)	Intel® Core™ i7 2715QE 2.1 GHz, 4 cores (TC3: 70)

Technical data	CX5120	CX5130	CX5140
Processor	Intel® Atom™ E3815, 1.46 GHz, 1 core (TC3: 40)	Intel® Atom™ E3827, 1.75 GHz, 2 cores (TC3: 40)	Intel® Atom™ E3845, 1.91 GHz, 4 cores (TC3: 50)

CX1010: Celeron M ULV 500 MHz Clock Frequency
CX1020: Celeron M ULV 1 GHz Clock Frequency
CX1030: Pentium M 1.8 GHz Clock Frequency

3.4. IO 刷新时间的分析

统计数据除了用于估算 CPU 利用之外，对比分析还可以发现以下规律：

1) 控制器带两条 EtherCAT

一个任务驱动同样总数的 IO 模块，放到 1 条 EtherCAT 网络，比放 2 条 EtherCAT 网络节约时间。比如 CX2020 用一条 EtherCAT 带 1250 字节的 Frame 需要耗时 15us，而一条 EtherCAT 带 748 字节 Frame，另一条带 100 字节的 Frame，同样的 Task 需要耗时 20us。

2) 两个任务带一条 EtherCAT

两个任务驱动同样总数的 IO 模块，比同一个任务刷新这些 IO 的时间几乎翻倍。所以 TC3 中允许任务合并是个优化，假如 PLC 有个任务是 2ms，NC 中有个任务也是 2ms，在 TC3 中就可以合并为同一个任务。

3) 一个任务带一条 EtherCAT，数据量超过 1 个 Frame

一个任务驱动的 IO 模块太多，一个 Frame 放不下，要放到两个 Frame 时，消耗的时间并没有明显增加。只与数据的总量有关系，是否与自动增加 Frame 无关。CX1020 带两包数据分别 918 和 721 字节，耗时 180us；带 1 包数据 1473 字节耗时 151us。

4) 刷新 IO 的时间取决于配置的从站数量及过程数据

IO 刷新需要的时间，与配置的从站多少相关，而与实际从站多少无关。所以可以带上一个从站，而配置 70 个从站，来实测带满 70 个从站需要的 IO 刷新时间。配置了从站多而实际少，与所有从站正常通讯所需要的时间，反而会增加。这是由于需要增加额外的时间来处理异常。

4. PLC 任务的耗时统计

在项目完成之前无法预知 PLC 代码量，更无法预估其 CPU 占用率。得益于 PC 控制的仿真运行功能，如果客户已经有类似规模的程序，可以下载到与目标控制器同系列的硬件 CPU 上运行，不带任何物理 IO，查看实际的 CPU 利用率，再乘以测试 CPU 与目标控制器的 CPU 主频之比，就可以推算出在目标控制器中运行这套 PLC 程序占用的 CPU 利用率。

但是了解一些常见的耗时较多的 PLC 命令在不同控制器上运行时所花费的时间，对于预防潜在的问题会有所帮助。

4.1. 写文件

以下数据为向本地存储卡写入 500kByte 数据到文件，需要花费的时间

	CX9020	CX5020	CX1020	CX5130	CX2020
创建文件并写入数据	8ms	4ms	2.1ms		
向已有文件写入数据	4.8ms	1.8ms			

实际测试结果显示，如果写文件的代码位于最低优先级，则其它任务不受其影响。如果不将写文件的代码与其它正常逻辑分开到不同的任务，则写文件的周期，正常逻辑会受影响，严重时可能隔一个周期才能执行。所以：**应将写文件的代码位于最低优先级。**

4.2. Novram

	CX9020		CX1020		CX2020
映射方式，1kBytes	279us				
映射方式，6kBytes	1493us		1472us		
Write 方式，1kBytes	580 us		PLC 执行时间：180us		
Write 方式，6kBytes	2060 us		PLC 执行时间：539us		
Write 方式，10kBytes	3140 us		PLC 执行时间：1033us		

实际测试显示，

使用映射的式，时间花在 IO 刷新上。如果 Novram 变量忘记拖放到慢的任务下，高优先级任务就会受影响，比如原本任何周期从 2ms，实际要 4ms 才执行一次。

使用 Write 方式，CPU 时间花在 PLC 执行上。从 TaskInfo 可以看到在写数据的那个周期执行时间特别长，但并不影响高优先级任务。

所以：使用 Write 方式，Novram 的写入频率更加可控，且占用的 CPU 资源更少。比如用来计数的变量，可以数据改变了才写入 Novram 区。而不用象映射方式每个任务周期都写 Novram，浪费 CPU 资源。但是要注意，写 Novram 的代码要放到低优先级且周期较长的任务中。

4.3. 温控 PID

		CX9020	CX1020	CX5130	CX2020

4.4. 同一个 CPU 的指令执行时间的对比

了解 PLC 指令的执行时间并进行对比，可以在优化代码节约 CPU 资源时更加胸有成竹。下面是使用指令 GetCpuCounter 测试得到的常见指令执行时间对比：

MEMCPY 耗时为普通赋值语句的 3 倍。

赋值、取反再赋值、判断的运算时间相当，称之为 PLC 基本指令时间。

整数加减运算是 PLC 基本指令时间的 2-2.5 倍

整数乘除运行是 PLC 基本指令时间的 1.5-2 倍

测试条件：CX1020 和笔记本电脑上的 32 位虚拟机。

测试数据：

时段	测试内容	CX1020 1000 次	虚拟机 1000 次	虚拟机 4000 次
1	(*读取 CpuCounter 的语句*)，单次	4	3	3
2	(*MEMCPY 指令耗时测量*) ，多次	405	97	381
3	(*判断语句*) ，多次	99	40	150
4	(*取反运算并赋值*) ，多次	135	38	154

5	(*赋值语句*) ，多次	131	37	144
6	(*减法运算并赋值*) ，多次	355	67	261
7	(*乘法运算并赋值*) ，多次	253	48	195

基于以上测试结果，之前常用的每个 PLC 周期执行的批量赋值语句，是非常消耗 CPU 资源的做法。在小项目中这种方式可以短平快地开发程序，在大型项目中这种方式就有优化的空间。比如令地址连续，然后用 MEMCPY 内存复制。在 TwinCAT 3 中可以用 Union 实现内存共享，项目开发之初详细规划数据存储地址和流向，可以提升程序执行的效率。

4.5. Display Flow Control 对 CPU 执行时间的影响

当前高亮显示的代码行，会增加 CPU 执行时间。不高亮显示的代码行不受影响。
从 SystemTaskInfoArr 看出，把高亮显示代码行的窗体最小化后，LastExecTime 恢复正常。
影响的程度：LAN 连接的影响最大，本地虚拟机影响小 10 倍，虽然用 ping 指令响应都 <1ms.
只有 Display Flow Control 才会影响代码执行，并且影响的程度与 PLC 周期无关

4.6. Login 的影响指令执行时间的分析

PLC Control 对程序的 Login，不会影响代码执行。只有 Display Flow Control 才会。
与不带 IO 映射且直接 Login 到 801 查看结果相比，PLC 执行时间没有区别。
*CPU 利用率可能有区别，因为消耗的是 IO 刷新的时间，不是 CPU 运算的时间。

5. NC 任务的耗时统计

NC Task 消耗的时间，包括 NC 发生器及单位换算、总线协议处理，和 IO 刷新的时间。
NC 发生器的时间，使用虚轴即可测量。
单位换算和总线协议处理，只有带上物理轴才能测量出来。
IO 刷新时间的计算遵循第 3 节中描述的规则。
因此，本节重点测试的是虚轴运行时间和少量实轴的转换时间。

5.1. 虚轴运行时间统计

虚轴运行的时间，在轴处于断使能、上使能、走定位运动、速度耦合运动、位置耦合运动等状态时，需要消耗的 CPU 资源是不同的。一个系统中实际上不可能所有的轴都同时做最耗资源的位置耦合运动，为了测试方便，本测试中的数据都是所有轴同时做相同运动时的数据。为了便于对比分析，所有控制器上运行同一套 PLC 程序和轴配置：40 个虚轴，其中 1 个主轴 39 个从轴。即使是 CX9020 也是如此，但为了不至于系统崩溃，把 CX9020 上的 NC 周期设置为了 10ms。

实测的 SAF 最大任务执行时间如下：

40 个虚轴	实测（SAF ） 任务最大耗时
CX2020 （TC3 单核）	SAF:145-154us SVB:3.5-3.8us
CX5130 （TC3 单核）	SAF:389-426us SVB:7us

CX5130 （TC3 双核）	SAF:382-437us SVB:6.2-7.8us
CX1020,	SAF:591us PLC:925us
CX5020,	SAF:591us
CX9020	SAF:2723us SVB:42us

根据以上 CPU 消耗时间，加上任务周期就可以计算出需要的 CPU 利用率。

5.2. 虚轴运行时间的分析

5.2.1.凸轮到底有多耗资源

有一个常识，是电子凸轮比电子齿轮运动更耗资源，实测数据也显示确实如此。在单轴定位、凸轮、齿轮运动中，最节约资源的就是齿轮了。但是电子凸轮与单轴定位相比，到底哪个更耗资源呢？不同的控制器，测试结果并不相同。

从 40 个虚轴的同时定位运动和 39 对凸轮运动的 CPU 耗时来看，

CX9020：定位比凸轮的 CPU 耗时低 8%

CX5020：定位比凸轮的 CPU 耗时高 **40**%（此数据可能有误，需要重新测试）

CX1020：定位比凸轮的 CPU 耗时低 10%

CX5130：定位比凸轮的 CPU 耗时高 20%

CX2020：定位比凸轮的 CPU 耗时低 7%

ARM 和 x86 平台的控制器，定位运动节约资源，而 ATOM 的控制器 CX5xxx 系列，则凸轮更耗资源。但是 CX5020 的数据中，走凸轮的耗时甚至比使能后静止的耗时更少，这是不可能的。

以凸轮最耗资源的数据 CX1020 来看，也仅比定位运动高 10%。所以预留 CPU 利用率时，可以直接以全部走凸轮运动这种最坏的情况来考虑。

另外，测试还发现，Fix Table 与 Motion Function 的凸轮表，对 CPU 消耗并无明显区别。执执效果在虚轴上测试也看不出区别。

5.3. 带不同类型的实轴时的 SAF 耗时

理论上，步进轴只有速度环，位置环需要在 NC 里完成。而 CSP 伺服的位置环在驱动器里面完成，所以 NC 带步进轴应该比带 CSP 伺服轴更耗资源。

但在用 EL7041（步进轴，速度模式）和 EL7211（伺服轴，位置模式）做对比测试时，发现结果并无明显差别。原因可能是步进电机并没有外部编码器来做闭环控制，而是以模块发出的脉冲数做为虚拟的位置反馈值。

5.3.1.带不同轴时 SAF 耗时统计

NC 运算(36 个虚轴，3 个实轴，NC：2ms，PLC:10ms， 1 个步进，2 个伺服，无 PLC 程序， 完全在 System Mangaer 中控制轴的运动		
	使能	轴定位

CX1020,只配 1 实轴（步进为实）	SAF:380-382us	SAF:391-392us
CX1020,只配 2 实轴 1 个伺服和步进为实	SAF:384-385	SAF:397-398us
CX1020,配 3 实轴 3 实轴同时动	SAF:398-400	SAF:418-420us
CX1020,实轴不动 3 虚轴同时动	SAF:398-400	SAF:418-420us
CX1020,配 3 实轴 但禁用后两个模块		SAF:423-425us

5.3.2.SAF 耗时数据分析

1) 只要配置成了实轴，不论虚轴动作还是实轴动作，IO 刷新时间不变，位置发生器耗时不变，协议转换单位换算的时间都不变，所以 SAF 的耗时不变。
2) 分别配置 1 个、2 个、3 个实轴时，SAF 耗时是递增的，这表示只有配置成实轴，NC 才会启用单位换算的和协议转换。
3) 增加一个步进实轴和增加一个伺服实轴，SAF 的耗时增加是不同的。可以推论，NC 驱动一个内置反馈的步进轴比驱动一个伺服轴要节约 CPU 资源。原因可能是，伺服轴除了状态字控制字目标位置实际位置之外，过程数据更多，需要做的转换工作也更多。到于 NC 带步进模块时需要增加的位置环控制，因为没有真正的外置编码器，所以简化了。

5.3.3.关于全闭环的 NC 轴耗时

做全闭环的时候，NC 的 SAF 耗时应该增加，增加的具体时间未能测试。

5.4. 带实轴的 NC 任务耗时估算实例

5.4.1.测试数据

测试条件：CX1020，两个实轴都是 EL72xx 伺服模块，等效于 COE 伺服。数据如下：

	40 个轴全部走定位运动
CX1020,0 实轴	SAF:542us
CX1020,1 实轴	SAF:652us
CX1020,2 实轴	SAF:658us

这是由于不带实轴时，SAF 任务完全不触发 EtherCAT 通讯。而只要带了 1 个实轴，那么同一个 Frame 中所有的 IO 从站数据都要由 SAF 任务去刷新。所以实轴数量从 0 到 1，任务耗时增加了 90us，而从 1 到 2 只增加了 6us。增加的这 6us 用于第 2 个实轴的单位换算、协议转换等处理。

可以推算，第 1 个实轴的 652us 中，有 542 用于虚轴运动，6us 用于单位换算协议转换，还有 104us 用于 EtherCAT 通讯的 IO 刷新。

5.4.2.估算带 40 个实轴的 NC 任务耗时

因为根据 3.2 节的 IO 刷新时间统计，
数据包在 1200 字节时，刷新时间为 135us，
数据包在 681 字节时，刷新时间为 110us，

如果 40 个轴都是实轴，以每个伺服 20 个字节计算，Frame 会略大于 800，所以取 IO 刷新时间取 135us 应该足够。所以 SAF 耗时为：
虚轴运动 542+IO 刷新 135+单位换算及协议转换 6*40＝646+240=917us。
如果 NC 任务周期 2ms，则对 CPU 利用率的贡献为：917/2000=45.8%
显然，再加上 PLC 任务，控制器最终的 CPU 利用率很可能超过 60%，所以 CX1020 以 2ms 的任务周期带不动 40 根轴。

5.4.3.估算带 20 个实轴的 NC 任务耗时

虚轴运行时间和实轴的单位换算时间，与轴的数量直接成正比，但是 NC 的 IO 刷新时间并不是与轴成正比的。所以要根据实际配置的 EtherCAT Frame 大小，估算 IO 刷新时间。

根据 3.2 节的 IO 刷新时间统计，
数据包在 1200 字节时，刷新时间为 135us，
数据包在 681 字节时，刷新时间为 110us，

如果 20 个轴都是实轴，以每个伺服 20 个字节计算，Frame 会略大于 400，所以取 IO 刷新时间取 110us 应该足够。所以 SAF 耗时为：
虚轴运动 542/2+IO 刷新 110+单位换算及协议转换 6*20＝271+110+120=501us。
如果 NC 任务周期 2ms，则对 CPU 利用率的贡献为：501/2000=25%
再加上 10ms 的 PLC 任务，控制器最终的 CPU 利用率也很难超过 50%，所以 CX1020 以 2ms 的任务周期可以带动 20 根轴。

5.4.4.特别说明

增加一个实轴所增加的 CPU 开销，需要多几个实轴测试才能准确。也可以平时在项目中积累数据，到新项目估算时才能尽量准确。
另外，上一节的测量数据也表明，带不同类型的轴 CPU 消耗不同，所以带几十个轴时也要考虑是否这些轴都是同一类型。比如不同品牌的伺服，可能过程数据就不同。是否启用探针功能、外部跟随误差等等，也会影响到 NC 需要处理的数据量。总之，积累的数据越丰富，估算时的依据就越完整，估算结果就越接近实际。

6. 什么是安全的 CPU 利用率

所谓安全，是指程序能稳定工作，跟外围的编程调试、通讯接口能正常响应，本机如果有 HMI 等应用程序也要能正常响应。从前面的描述可以知，CPU 利用率总是在不停地波动，所以在平均 CPU 利用率之外，还要考虑它的波动范围。

实际测试的结果：

CX1020，无 HMI 或者第三方应用程序，在 CPU 利用率超过 70%以后，就相继出现以下问题：

Scope 电子示波器先是卡慢，继而卡死。

System Manager 变得很慢，继而 TimeOut

PLC Control 中创建引导程序失败，PLC 停止都不行，必须断电重启。

所以，对于 CX1020，CPU 利用率应控制在 60%以内。这些都是经验数据，不可一概而论。如果控制器上运行的第三方应用程序或者服务较多，CPU 利用率还应该再降低。从趋势上来说，越是主频低的控制器，安全的 CPU 利用率越低。因为在线编程调试的资源是必须预留的。越是主频高的控制器，或者是多核 CPU，其安全 CPU 利用率越高。

至于 CX51、CX20 系列的安全 CPU 利用率，需要再做测试。即测试 CPU 繁忙的极限，直到出现 ADS 通讯卡死等问题。

总之，还是要具体问题具体分析，根据每个应用项目选择安全的 CPU 利用率。