第 K 部分

安全和 OTP 编程

目录

27 维护 IP 和设备准确性

在本章中

- xcore AES 模块
- AES 开发模块的启用
- 生产闪存编程流程
- 生产 OTP 编程流程

Xcore 设备包含一次性可编程（OTP）的片上存储器，内存可在装置测试期间或之后展开。可将 xcore AES 模块分散到 OTP 设备，允许程序被加密存储在闪存设备。这有助于提供：

- 保密性

加密的程序难以被反向编程。

- 程序真实性

AES 装载机将无法加载被篡改的程序或其他第三方程序。

- 设备的真实性

无法使用由第三方提供的 XCore 设备复制由您的密钥加密的程序。

一旦 AES 模块编程完成，OTP 安全位将被断开，每个区块生成一个“安全孤岛”所有计算、内存、I / O 和通信将接受相应区块上运行代码的单独控制。在设置时，此类位置：

- 将 boot 强制通过 OTP 引导以防止被绕过，
- 禁用 JTAG 访问区块以防止按键被读取, 并禁止任何其他 OTP 写入以防止更新。

AES 模块针对普通黑客提供了强有力的安全保护，因此很有必要实现。但重要的是要意识到，不存在绝对牢不可破的安全性，如果对方决心已定且具备所需能力，任何方法都无法阻止对方获得您的密匙。

Shiqiang Xiao 13-11-11 10:14 PM
已删除: 停止

27.1 XCore AES 模块

XCore 编辑 AES 模块验证并解密 SPI 闪存设备中的程序。如图 47 所示，将其编程至设备后可启用以下安全开机程序。

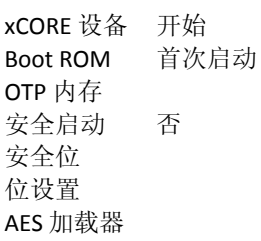


图 47:
使用 AES 模块的
安全开机程序

128 位认证密钥
128 位认证密钥
闪存设备
闪存加载器
0
原厂镜像
1
升级镜像
是
标准引导
（图中未示出）
安全加载
启动加载器
加载和认证
闪存加载器
加载和
认证
加密镜像

解密并加载程序段。
执行程序

1. 该装置主要由 ROM 引导程序加载，所检测到的安全启动位设置在 OTP 然后加载并执行 AES 模块。
2. AES 模块 SPI Flash 的代码加载至内存中
3. AES 模块使用 cmac-aes-128 算法和 128 位密钥验证闪存装载机。如果认证失败，则启动停止。
4. AES 模块将认证密钥和解密密钥存放于寄存器并跳转到闪存装载机。

Shiqiang Xiao 13-11-11 10:15 PM
已删除: 加载内存至

Shiqiang Xiao 13-11-11 10:15 PM
已删除: 装载机

闪存装载机执行下列操作：
1. 选择验证其 CRC 数字最大的镜像
2. 对选定的镜像使用 CMAC 标签和认证密钥。如果认证失败，启动停止。
3. 验证、解密和负载程序/数据段到内存中的表。如果镜像没有认证，启动停止。
4. 开始执行程序。
多节点系统，AES 模块写入某区块的 OTP，和从 xConnect 链路协议安全启动编程到所有其他区块。

27.2 启用 AES 开发模块
您可以在开发或设备制造过程的任何时间激活 AES 模块。在开发环境中，您可以离开没有设置的安全位并激活模块，从而使：
- XFlash 使用设备程序加载到闪存，
- xgdb 调试程序上运行的装置以及
- Xburn 编写额外的 OTP 位保护装置。

在生产环境中必须保护装置，防止 OTP 按键被最终用户读出。
如需将 AES 模块编程至开发板中的 XCore 设备，启动命令行工具（参见第 3.2 节）并输入以下命令：
1. xburn --genkey keyfile
XBURN 将两行随机的 128 位密钥写入 keyfile。第一行为上述认证密钥，第二行为解密密钥。
可使用开放源码库 crypto++生成加密密钥。如愿意，您也可创建该文件并提供自有钥匙。
2. xburn -l
XBURN 打印所列所有连接至您的电脑的 JTAG 适配器列表以及 JTAG 链中的每个装置，相应形式为：
ID - NAME (ADAPTER-SERIAL-NUMBER)

3. xburn --id ID --lock keyfile --target-file target.xn --enable-jtag
--disable-master-lock
XBURN 将 AES 模块和安全密钥写入连接至目标设备的 OTP 存储器并设定其安全开机位。用于引导的 SPI Flash 的端口将从 XN 文件中引出（见 XM-000929-PC）。

如需加密程序并将其写入闪存，请输入命令： • xflash --id ID bin.xe --key keyfile
要保护的 XCore 设备并防止任何进一步开发，请输入以下命令：
• xburn --id ID --target-file target.xn --disable-jtag --lock keyfile

27.3 生产闪存编程流程

在生产制造环境下，相同的程序通常被编程至多个 SPI 设备。
要生成一个加密 XCore 闪存格式的镜像，启动命令行工具（见§3.2）并输入以下命令：
• xflash prog.xe -key keyfile -o image-file
使用第三方闪存编程可将此镜像直接写入闪存，也可使用 XFLASH 将其写入：（通过 XCore 编辑设备）。如需使用 XFLASH 写入，请输入以下命令：
1. xflash -l
XFLASH 打印所列所有连接至您的电脑的 JTAG 适配器的枚举列表以及 JTAG 链中的每个装置，相应形式为：
ID - NAME (ADAPTER-SERIAL-NUMBER)
2. xflash --id ID --target-file platform.xn --write-all image-file
XFLASH 生成 XCore 闪存格式的镜像，其中包含第一阶段的加载器和原厂镜像以及来自您编译程序的二进制和数据段。随后 XFLASH 将使用 XCore 设备写入该镜像至闪存。

XN 文件必须定义一个 SPI flash 以及其连接的 XCore 设备的四个指定端口（见 XM-000929-PC）。

Shiqiang Xiao 13-11-11 10:19 PM
已删除: 闪存设备

27.4 生产 OTP 编程流程

在生产制造环境下，相同的密匙通常被编程至多个 XCore 设备。

如需生成包含 AES 模块和安全密钥并将被写入 OTP 的镜像，启动命令行工具（见第 3.2 节）并输入以下命令：

1. xburn --genkey keyfile

XBURN 将两行随机的 128 位密钥写入 keyfile。第一行为上述认证密钥，第二行为解密密钥。

可使用开放源码库 crypto++生成加密密钥。如愿意，您也可创建该文件并提供自有钥匙。

2. xburn --target-file target.xn --lock keyfile -o aes-image.otp

XBURN 生成包含 AES 模块、安全密钥的安全位取值的镜像。

该镜像包含密匙，不得泄密。

如需在生产环境中将 AES 模块和安全位写入设备，输入以下命令：

1. xburn -l

XBURN 打印所列所有连接至主机的 JTAG 适配器枚举列表以及 JTAG 链中的每个装置，相应形式为：

ID - NAME (ADAPTER-SERIAL-NUMBER)

2. xburn --id ID --target-file target.xn aes-image.otp

XBURN 在设备中加载将 AES 模块和安全密钥写入 OTP 的程序并设定其安全开机位。用于引导的 SPI 端口将从 XN 文件中引出（见 XM-000929-PC）。XBURN 返回 0 为成功，非零为失败。

28 XBURN 命令行手册

在本章中
- 整体选项
- 安全寄存器选项
- 目标选项
- 编程选项

XBURN 创建 OTP 镜像并将镜像编入 XCore 编辑设备的 OTP 存储器之中。

28.1 整体选项

下列选项被用于指定 OTP 镜像和安全寄存器的内容。

xe-file 指定通过 XE-文件和一组默认的安全位在装入段构造的开机镜像（见图 48）。

otp-file 指定 OTP 文件内的 OTP 分段，其中需包括安全寄存器的取值。

--lock keyfile
指定 XCore 编辑 AES 启动模块（见第 27.1 节）和默认安全位的集合（见图 48）。

--genkey keyfile
将两个用于认证和加密的 128 位密钥输出至 keyfile。使用开放源码库 crypto++生成密钥。
在--burn or --lock 下此选项是无效的
--mac-address mac
MAC 地址写入 OTP 的结尾。MAC 地址应该以 12:34:56:78:9A:BC 的形式指定。可以通过多次指定 - MAC 地址选项写入多个 MAC 地址。 MAC 地址写入至 OTP 的顺序与选项出现的顺序相同。
--serial-number 序列号
将某个 32 位的序列号写入 OTP 的结尾。
--read 打印 OTP 的全部内容。
--help 打印所支持的命令行选项的描述。
--version 显示版本号和版权。

Shiqiang Xiao 13-11-11 10:21 PM
已删除:

Shiqiang Xiao 13-11-11 10:21 PM
已删除:：

28.2 安全寄存器选项

下列选项用于指定 OTP 安全寄存器的内容并覆盖图 48 所示 BURN XE 镜像、OTP 镜像和 AES 模块的默认选项。

图 48：由 XBURN 写入的默认安全位

安全位	XE 镜像	OTP 镜像	AES 模块 (--lock)
OTP Boot	启用	对应每个 OTP 镜像文件	启用
JTAG Access	启用	禁用	
Plink Access	启用	启用	
Global Debug	启用	禁用	
Master Lock	禁用	启用	

--enable-otp-boot

使能从 OTP 启动。

--disable-jtag

禁用 JTAG 访问。一旦禁用，将不可再次获得设备调试访问权限或读取 OTP。此选项不禁用边界扫描。

--disable-plink-access

禁止通过其他区块访问 plink 寄存器。禁用伪链接访问将限制所有通过每个 plink 本地区块对该 plink 寄存器的所有存取。

--disable-global-debug

防止设备加入全局调试。禁用全局调试将阻止区块使用全局调试引脚进入调试。

--enable-master-lock

启用 OTP 主锁。即不允许对 OTP 进行任何进一步修改。

Shiqiang Xiao 13-11-11 10:24 PM
已删除: 允许从

Shiqiang Xiao 13-11-11 10:24 PM
已删除: 权力

28.3 目标选项

下列选项用于指定目标硬件平台。

--list-devices

-l 打印所列全部 JTAG 适配器列表，相应适配器应通过各个 JTAG 链按如下形式连接至主机和设备：

ID - NAME (ADAPTER-SERIAL-NUMBER)

应按照其序列号订购此类适配器。

--id ID 指定连接到目标硬件的适配器。

--adapter-id ADAPTER-SERIAL-NUMBER
指定连接到目标硬件的适配器的序列号。
--jtag-speed n
设置 JTAG 时钟分频器为 n。相应 JTAG 时钟速度是 6 /（n +1）MHz。默认值是 0（6MHz）。

Shiqiang Xiao 13-11-11 10:25 PM
已删除: 兆赫

--spi-div n 设置 AES 模块所用分压器的的 SPI 时钟至 n。相应的 SPI 时钟速度被设置为 100 /（2n）MHz。默认值为 20（2.5MHz）。
在--lock 下此选项才为有效。

Shiqiang Xiao 13-11-11 10:25 PM
已删除: 兆赫

--target-file xn-file
指定 XN-文件为目标平台。

--target platform
指定一处目标平台。在文件 platform.xn 中必须指定该平台的配置，可通过由 XCC_DEVICE_PATH 环境变量（参见第 9.8 节）中指定的路径搜索该本件。

28.4 编程选项
默认情况下，XBURN 将指定的 OTP 镜像写入目标平台。 -o otp-file 将输出至 otp-file, 从而禁用编程。
--make-exec xe-file
将一个可执行文件放入 xe-file，于 XCore 编辑设备运行时执行规定的 OTP 烧录操作并禁用编程。
也可随后使用 XRUN 运行该 XE 文件。

--force
-f 禁止在提示后写入 OTP。此并非默认值。
--size-limit n
限制写入 OTP 前 n 个字节的 OTP 存储器数量。如果镜像不符合规定的限制将出现一次错误。

第 L 部分

C/ XC 语言编程

目录

29 C / C ++和 XC 之间的调用

在本章中
•XC 至 C/ C + +的参数传递
•C / C++至 XC 参数传递

在某些情况下，可将 XC 语言的某类型参数转移至 C / C ++语言的非同类型函数参数，反之亦然。
为简化 C / C ++和 XC 之间常用函数的说明任务，系统头文件 xccompat.h 的包含多种类型的定义和宏定义。请参阅头文件相关文档。

29.1 XC 至 C/ C + +的参数传递

带有 unsigned int 类型的参数 C / C++ 语言定义函数可使用 XC 语言阐明，相应参数的类型可为端口、通道传输结束或计时器。
带有“指向 T”类型参数 C / C++ 语言定义函数可使用 XC 语言阐明，相应参数的类型为“参考 T”或“可为空的参考 T”。

带有“指向 T”类型参数 C / C++ 语言定义函数可使用 XC 语言阐明，相应参数的类型为“T 阵列”。

29.2 C / C++至 XC 参数传递

带有端口、通道传输结束或定时器类型参数 XC 语言定义函数可使用 C/C++语言阐明，相应参数的类型为 unsigned int。
带有“参考 T”或“可为空的参考 T”类型参数 XC 语言定义函数可使用 C/C++语言阐明，相应参数的类型为“指向 T”。
带有“T 阵列”类型参数 XC 语言定义函数可使用 C/C++语言阐明，相应参数的类型为“指向 T 类型”。在这种情况下，链接程序插入代码 XCore 编辑将添加一个等于无符号整形类型最大值的隐式数组边界参数。

Shiqiang Xiao 13-11-11 10:26 PM
已删除: 计时器

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

30 XC 语言实现定义的行为

需要一致性 XC 应用才可记录其被指定执行定义的语言规范的所有部分的行为选项。在以下章节中，所有依赖于外部定义的应用程序二进制接口的选项将被列为“符合 ABI 定义”，且其记录应符合应用程序二进制接口规范（参见第 40 节）。

- 多字符常量取值（第 1.5.2 节）。

多字符常量取值应与其首个字符取值相同，所有其他字符将被忽略。

- 相同字符串的文字是否不同对此没有影响（第 1.6 节）。

如相同字符串的文字不同；其实现方式应存在单一的内存位置。

- 可被存储至某 char 的可用取值范围以及是否带有签名（第 3.2 节）。

Char 的容量为 8 位。存储至 char 的变量值是否带有签名由 ABI 决定。

- 连接至某个端口并用于与环境进行通信的管脚数；不带外部说明其非显式初始化某个端口或时钟取值（第 3.2，7.7 节）。

连接至某端口用于环境通信的针脚数目的定义由显式初始化程序或其说明符决定。如不存在初始化程序，编译器会生成一份错误消息。

- 某端口的理论传输类型、某端口的理论计数器类型以及某计时器的理论计数器类型（第 3.2 节）。

ABI 将决定理论类型。

- 某整形值转换为带符号类型的情况应为其取值无法使用新类型表示（第 5.2 节）。

当任何整形值转换为带符号类型且其取值无法使用新类型表示时，其数值将被截短至新类型的宽度和符号扩展。

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

- 溢出处理、除法校验和其他异常情况的表达式评估（第 6 节）。

导致未定义的行为的所有条件（零除、零模组以及符号除法/模组溢出）。

- 对齐（第 6.3.4 节）的概念。

2n 的校准可保证最显著 n 位内存地址为零。上述特定校准类型由 ABI 确定。

• 运算符结果的取值和类型（第 6.4.6 节）。
运算符操作符结果的取值由 ABI 决定。结果类型为不带符号的 int。
•试运行时间的零除数（第 6.6 节）。
试运行时间的零除数将导致自陷
• 使用 inline 函数说明符创建暗示的程度应为有效的（第 7.3 节）。
inline 函数说明符被作为内联函数的某个提示。编译器将尝试在所有-O0 以上的优化级别内内联函数。
• 使用寄存器存储类说明符创建暗示的程度应为有效的（第 7.7.4 节）
寄存器存储说明符可使寄存器分配者尝试将变量置于函数中的寄存器内。但分配者并非一定将其置于某个寄存器之中。
• 支持输入操作所谓词函数（第 8.3 节）。
所支持的谓词函数集被记录于第 39.8 节中
• 输入和输出端口（第 8.3.2 节）的涵义。
输入和输出端口映射的是输入和输出端口的资源指示，其中的行为定义可参见 XS1 接口规范（见 X1373）。
• 达到底层通信协议根据交易通信需要优化的程度（第 8.9 节）。
通信协议由 ABI 决定。
• 无论是否由于通信发生后，相应输出的字节数不等于输入的子节数以及通信的取值而导致交易无效（第 11 节）。
所有情况下皆由 ABI 决定。
• 无效操作的行为（第 12 节）。
除下文所述以外，所有无效操作应被报告为编译错误或将在运行时导致自陷。
• 无效主交易说明的行为并不确定；无效从属交易将始终自陷。
• 不安全编译指示（见第 8 节）可以用来禁用特定的安全检查，这将导致无效操作的未定义行为。

31 C 语言实现定义的行为

在本章中
- 环境
- 标识符
- 字符
- 浮点运算
- 提示
- 预处理指令
- 库函数
- 环境特定的行为
- C 和 C++语言参考

需要一致性 C99 应用才可记录其被指定执行定义的语言规范的所有部分的行为选项。除如下所列选项以外，xTIMEcomposer 实现定义的行为应与 GCC4.2.15 相匹配。
下一章节的标题是指 C99 规范中的相应部分（见第 32.1 节），所有依赖于外部定义的应用程序二进制接口的选项将被列为“符合 ABI 定义”，且其记录应符合应用程序二进制接口规范（参见第 40 节）。
只有被支持的 C99 函数有相关记录。

31.1 环境

- 当程序在一个独立的环境启动时所调用函数的名称和类型（5.1.2.1）。

已提供某种托管环境。

- 其中主要函数可以被定义的另一种方式（5.1.2.2.1）。

没有可以定义主要函数的替代方式。

- 字符串的给定值由 argv 参数指向主要函数（5.1.2.2.1）。

argc 值等于零。argv[0]是一个零指示字。不存在其他数组成员。

- 某个互动装置的构成部分（5.1.2.3）。

所有数据流皆适用于交互式设备。

- 对应计算异常其除 SIGFPE、SIGILL 和 SIGSEGV 之外的其他信号值（7.14.1.1）。

不存在其他对应计算异常的信号值。

5 http://www.xmos.com/references/gcc-4.2.1-c-implementation

• 相当于该信号（sig，SIG_IGN）的信号值应在程序启动执行（7.14.1.1）。
在程序启动时将为所有信号执行信号（SIG，SIG_DFL）的等价值。
• 环境名称的组合以及更改环境列表中所使用 getenv 函数（7.20.4.5）的方法。
环境名称的组合为空。不存在可改变 getenv 函数所使用环境列表的方法。
• getenv 函数所使用的系统函数执行字符串的方式（7.20.4.6）。
相应情况由执行环境决定。

31.2　标识符
•单个标识符内重要的初始字符数（5.2.4.1，6.4.1）。
　　标识符中的所有字符（带或不带外部联动）皆为重要的。

31.3　字符
• 执行字符集（5.2.1）组成部分的取值。
　　应由 ASCII 字符集决定。
• 为各标准字母转义序列而产生的执行字符集组成部分的独特取值（5.2.2）。
应由 ASCII 字符集决定。
• 某个字符对象的值，其中已经存储了任何不为基本执行集组成部分的字符（6.2.5）。
任何不为基本执行集组成部分的字符的值由 ASCII 字符集决定。
•源代码字符集组成部分（字符常量和字符串文字）对执行字符集（6.4.4.4，5.1.1.2）组成部分的映射。
源代码字符集必须使用 ASCII 字符集。源代码字符集内部每个字符都将被映射至执行字符集内的相同字符。
• 包含多个字符或包含一个并未映射至某单字节执行字符的字符或转义序列的字符常量的取值。
包含多个字符的字符常量的取值等于该字符常量内最后一个字符的取值。包含一个并未映射至某单字节执行字符的字符或转义序列的字符常量的取值等于减去 modulo 2n 的取值，以达到相应的字符类型范围，其中 n 为单个字符的字节数。

•含有一个以上的多字节字符或包含多字节字符或转义的多字节字符又或包含扩展执行字符集并未表明的多字节字符或转义序列的宽字符常量的值（6.4.4.4）。
宽字符常量不能包含多字节字符。
• 用于转换宽字符常量的当前语言环境，该常量所包含的某单一多字节字符将扩展执行字符集的组成部分映射至对应的宽字符代码（6.4.4.4）。
宽字符常量不能包含多字节字符。
• 包含扩展执行字符集（6.4.5）并未表明的多字节字符或转义序列的某个字符串常量的值。
字符串字面值不可包含多字节字符。如果某个字符串字面值中使用了非执行字符集表达的转义序列，字符串中相应字符的值应等同于将会赋予作为转义序列组成部分的字符常量的值。

31.4 浮点运算

• 浮点运算以及返回<math.h>和<complex.h>浮点结果（5.2.4.2.2）的库函数的准确性。
故意未保存相关记录。
• 其他浮点异常、舍入模式、环境和分类的宏名称（7.6，7.12）。
没有其他被定义的浮点异常、舍入模式、环境或分类。

31.5 提示

▼ 使用 register 存储类说明符提出建议的程度是有效的（6.7.1）。

除非用作 register 变量扩展的一部分，否则忽略 register 说明符。

31.6 预处理指令

▼ 每个已识别的非 STDC #pragma 指令的行为（6.10.6）。

记录在 §8 中。

31.7 库函数

▼ 独立式程序可用的任何库工具，子句 4 要求的最低设置除外（5.1.2.1）。

提供主机环境。

▼ 断言宏输出的诊断格式（7.2.1.1）。

断言宏采用的格式为："断言失败：*expression*，文件 *filename*、行 *line number*、函数：*function*。"，其中 *expression* 是指参数文本，*filename* 是__FILE__的值，*line number* 是__LINE__的值，而 *function* 则是流函数的名称。如果无法确定流函数的名称，则省略消息的该部分。

▼ 由 fegetexceptflag 函数存储的浮点状态标志的表示（7.6.2.2）。

不支持函数 fegetexceptflag。

▼ 除了"上溢"或"下溢"浮点异常之外，feraiseexcept 函数是否会产生"不精确"浮点异常（7.6.2.3）。

不支持函数 fegetexceptflag。

▼ 除了"C"和""之外可以作为第二个参数传递给 setlocale 函数的其他字符串（7.11.1.1）。

其他字符串不可作为第二个参数传递给 setlocale 函数。

▼ 当 FLT_EVAL_METHOD 宏的值小于 0 或大于 2 时针对 float_t 和 double_t 定义的类型（7.12）。

不支持 FLT_EVAL_METHOD 宏的其他值。

▼ 数学函数的域错误，与此国际标准的要求不同（7.12.1）。

故意留白。

▼ 发生下溢范围错误时数学函数返回的值，当整形表达式 math_errhandling & MATH_ERRNO 为非零时，是否将 errno 设置为宏 ERANGE 的值，以及当整形表达式 math_errhandling & MATH_ERRNO 为非零时，是否产生"下溢"浮点异常。（7.12.1）。

Shiqiang Xiao 13-11-13 1:47 PM
已删除：整数

Shiqiang Xiao 13-11-13 1:47 PM
已删除：整数

故意留白。

▼ 当一个 fmod 函数具有一个等于零的第二参数时，是发生域错误还是返回零（7.12.10.1）。

当一个 fmod 函数具有一个等于零的第二参数时，发生域错误。

▼ 在减小商时，remquo 函数所用的以 2 为基数的模数对数（7.12.10.3）。

商是减小后的模数 2^7。

▼ 在调用信号处理程序之前是否执行 signal(sig, SIG_DFL); 的等效函数，如果不执行，则阻塞执行的信号（7.14.1.1）。

在调用信号处理程序之前执行 signal(sig, SIG_DFL); 的等效函数。

▼ 宏 NULL 扩展到的空指针常量（7.17）。

NULL 被定义为（（void *）0）。

▼ 文本流的最后一行是否需要一个终止换行符（7.19.2）。

由执行环境确定。

▼ 写出到换行符前面的文本流中的空格字符在读入时是否出现（7.19.2）。

由执行环境确定。

▼ 可附加至写入二进制流的数据的空字符数（7.19.2）。

由执行环境确定。

▼ 附加模式流的文件位置指示符最初是位于文件开头还是文件结尾（7.19.3）。

由执行环境确定。

▼ 对文本流的写操作是否会导致关联文件在超过该点后被截断（7.19.3）。

由执行环境确定。

▼ 文件缓冲的特征（7.19.3）。

缓冲的输出流保存字符，直至缓冲区满了为止，然后将所述字符作为块进行写入。行缓冲的输出流保存字符，直至行完成或缓冲满了为止，然后将所述字符作为块进行写入。未缓冲的输出流立即将字符写入目标文件。

▼ 零长度文件是否确实存在（7.19.3）。

由执行环境确定。

▼ 书写有效文件名的规则（7.19.3）。

由执行环境确定。

▼ 同一文件是否可以同时打开多次（7.19.3）。

由执行环境确定。

▼ 文件中多字节字符所用编码的性质和选项（7.19.3）。

执行字符集不得包含多字节字符。

▼ remove 函数对打开的文件的作用（7.19.4.1）。

由执行环境确定。

▼ 在调用 rename 函数之前存在一个具有新名称的文件时的作用（7.19.4.2）。

由执行环境确定。

▼ 程序异常终止后是否删除打开的临时文件（7.19.4.3）。

程序异常终止后，不删除临时文件。

▼ 允许模式进行哪些更改（如果有），以及在什么情况下允许更改（7.19.5.4）。

不得赋予文件更自由的存取模式（比如，在只读文件描述符上，“w”模式将失效），但可以改变状态如附加或二进制模式等。如果不能更改，将发生故障。

▼ 用于输出无穷大或 *NaN* 的样式，以及为 *NaN* 输出的任何 n-char 或 n-wchar 序列的含义（7.19.6.1，7.24.2.1）。

代表无穷大的双参数在样式[-] inf 中转换。代表 *NaN* 的双参数在 as nan 样式中转换。

▼ 在 fprintf 或 fwprintf 函数中%p 转换的输出（7.19.6.1，7.24.2.1）。

指针的值被转换为 *dddd* 样式的无符号十六进制计数；利用字母 abcdef 进行转换。精度规定出现的最少数字位数；如果被转换的值可以用较少数字表示，则用前导零加以扩展。默认精度为 1。将字符 0x 添加到输出之前。

不支持 fwprintf 函数。

▼ 在 fscanf 或 fwscanf 函数中 %[转换的扫描列表中，当 - 字符既不是第一个字符也不是最后一个字符，又不是 ^ 字符为第一个字符的情况下的第二个字符时，此时对 - 字符的解释（7.19.6.2，7.24.2.1）。

如果在 – 字符之后的字符在数值上比在 – 字符之前的字符大，则认为 – 字符限定了一个范围。否则 – 字符本身应被加入扫描列表。

不支持 fwscanf 函数。

▼ fscanf 或 fwscanf 函数中%p 转换匹配的序列集合以及相应输入项的解释（7.19.6.2，7.24.2.2）。

%p 与%x 匹配同样的格式。相应的输入项被转换为一个指针。不支持 fwscanf 函数。

▼ 在表示由 strtod、strtof、strtold、wcstod、wcstof 或 wcstold 函数转换的 *NaN* 的字符串中的任一 n-char 或 n-wchar 序列的含义(7.20.1.3，7.24.4.1.1)。

不支持 wcstod、wcstof 和 wcstold 函数。在表示 *NaN* 的字符串中的一个 n-char 序列应以十六进制的形式扫描。忽略所有非十六进制数位的字符。

▼ 发生下溢时，strtod、strtof、strtold、wcstod、wcstof 或 wcstold 函数是否将 errno 设置为 ERANGE（7.20.1.3，7.24.4.1.1）。

不支持 wcstod、wcstof 和 wcstold 函数。发生下溢时，函数 strtod、strtof 和 strotold 不会将 errno 设置为 ERANGE 并归零。

▼ 当请求的大小为零时，calloc、malloc 和 realloc 函数是返回空指针还是返回指针至一个已分配对象（7.20.3）。

当请求的大小为零时，函数 calloc、malloc 和 realloc 均返回空指针至一个已分配对象。

▼ 调用 abort 或_Exit 函数时，是刷新具有未写入缓冲数据的开放流、关闭开放流还是删除临时文件（7.20.4.1，7.20.4.3，7.20.4.4）。

调用 abort 函数或_Exit 函数时，删除临时文件，不刷新缓冲文件，也不关闭开放流。

▼ abort、exit 或_Exit 函数返回给主机环境的终止状态（7.20.3）。

函数 abort 导致软件异常发生。函数 exit 和_Exit 返回给主机环境的中止状态由执行环境确定。

▼ system 函数在其参数不是空指针时返回的值（7.20.4.6）。

由执行环境确定。

▼ 可以用 clock_t 和 time_t 表示的时间的范围和精度（7.23.1）。

用 time_t 表示的时间精度由执行环境界定。time_t 指定一个无符号长整形。用 time_t 表示的实际时间范围由执行环境界定。

宏 CLOCKS_PER_SEC 被界定为 1000。clock_t 指定一个无符号长整形。

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

▼ clock 函数的年代（7.23.2.1）。

clock 函数总是返回值（clock_t）（-1），表明所使用的处理机时间不可用。

▼ “C”语言环境中 strftime 和 wcsftime 函数的%Z 说明符的替换字符串（7.23.3.5，7.24.5.1）。

%Z 说明符用字符串“GMT”替换。

31.8 语言环境特定的行为

▼ 基本字符集之外的源代码字符集和执行字符集的附加成员（5.2.1）。

源代码字符集和执行字符集均包括 ASCII 字符集的所有成员。

▼ 基本字符集之外的执行字符集中附加多字节字符的存在、含义和表示（5.2.1.2）。

执行字符集中不存在多字节字符。

▼ 用于多字节字符编码的移位状态（5.2.1.2）。

源代码字符集和执行字符集中均不存在多字节字符。

▼ 连续输出字符的写入方向（5.2.2）。

字符从左到右输出。

▼ 小数点字符（7.1.1）。

小数点字符为‘.’。

▼ 输出字符集（7.4，7.25.2）。

由 ASCII 字符集确定。

▼ 控制字符集（7.4，7.25.2）。

由 ASCII 字符集确定。

▼ 由 isalpha、isblank、islower、ispunct、isspace、isupper、iswalpha、iswblank、iswlower、iswpunct、iswspace **或** iswupper **函数测试的字符集（7.4.1.2、7.4.1.3、7.4.1.7、7.4.1.9、7.4.1.10、7.4.1.11、7.25.2.1.2、7.25.2.1.3、7.25.2.1.7、7.25.2.1.9、7.25.2.1.10、7.25.2.1.11）。**

不支持函数 isblank、iswalpha、iswblank、iswlower、iswpunct、iswspace 和 iswupper。

函数 islower 测试字符‘a’至‘z’。函数 isupper 测试字符‘A’至‘Z’。函数 isspace 测试字符‘’，‘\f’，‘\n’，‘\r’，‘\t’和‘\v’。函数 isalpha 测试大写字符和小写字符。函数 ispunct 测试除空格字符和字母数字字符之外的所有可印刷字符。

▼ 本地环境（7.11.1.1）。

本地环境与 C 翻译的最小环境相同。

▼ 数值转换函数接受的附加主题序列（7.20.1，7.24.4.1）。

数值转换函数不接受任何附加主题序列。

▼ 执行字符集的整理序列（7.21.4.3，7.24.4.4.2）。

函数 strcoll 执行的比较与函数 strcmp 执行的比较相同。

▼ strerror **函数设置的错误消息字符串的内容（7.21.4.3，7.24.4.4.2）。**

错误消息字符串的内容见图 49。

▼ iswctype **函数支持的字符分类（7.25.1）。**

iswctype 支持的字符分类见图 50。

图 49：错误信息字符串

值	字符串
EPERM	非所有人
ENOENT	没有这样的文件或目录
EINTR	中断系统调用
EIO	输入/输出故障
ENXIO	没有这样的设备或地址
EBADF	文件号不符合要求
EAGAIN	没有更多过程
ENOMEM	没有足够空间
EACCES	拒绝访问
EFAULT	错误的地址
EBUSY	设备或资源忙
EEXIST	文件存在
EXDEV	文件存在交叉链接
ENODEV	没有这样的设备
ENOTDIR	不是一个目录
EISDIR	是一个目录
EINVAL	非法参数
ENFILE	系统打开的文件太多
EMFILE	打开的文件太多
ETXTBSY	文本文件占用
EFBIG	文件过大
ENOSPC	设备空间不足
ESPIPE	非法查询
EROFS	只读文件系统
EMLINK	链接过多
EPIPE	传输失败
EDOM	数学参数
ERANGE	结果太大
ENAMETOOLONG	文件或路径名过长
ENOSYS	函数未执行
ENOTEMPTY	目录非空
ELOOP	符号链接过多

Shiqiang Xiao 13-11-12 1:54 PM
已删除：管破裂

图 50：宽字符映射

值	描述
WCT_TOLOWER	转换为小写
WCT_TOUPPER	转换为大写

32 C 和 C++语言参考

在本章中

由于已存在高质量的文件，因此 XMOS 不制备与 C 和 C++标准语言特征相关的文件。

32.1 标准

▼ ISO/IEC 9899:1989：编程语言 — C.（C89）。国际标准组织。

▼ ISO/IEC 9899:1999：编程语言 — C.（C99）。国际标准组织。

▼ ISO/IEC 14882:2011：编程语言 — C++（C++标准）。国际标准组织。

32.2 书籍

▼ C 编程语言（第二版），Brian W. Kernighan 和 Dennis M. Ritchie 著，Prentice-Hall, Upper Saddle River, NJ, USA 于 1988 年出版，ISBN-10: 0131103628。

32.3 在线

▼ comp.lang.c 常见问题：http://c-faq.com/

第 M 部分

用汇编语言编程

目录

33 内联汇编

可利用 asm 陈述将用汇编语言写成的编码嵌入一个 C 或 XC 函数中。比如，加法指令可以写成：

```
asm("add %0, %1, %2" : "=r"(result) : "r"(a), "r"(b));
```

用双点隔开汇编程序的模板、输出操作码和输入操作码。用逗号隔开一组中的操作码。用一个操作码约束字符串描述各操作码，后接用括号括起来的表达式。操作码约束字符串中的“r”表示该操作码必须位于一个寄存器中。操作码约束字符串中的“=”表示该操作码是写成的。

各输出操作码表达式必须为一个左值，其约束中必须存在“=”。

可利用格式%num（其中 num 表示操作码编号）的一个转义序列指出操作码在汇编程序中的位置。可利用转义序列“%=”发出一个编号，该编号对一个 asm 陈述的各个表达式而言是唯一的。贴本地标签时这很有用。如需生成文字“%”，必须写入“%%”。

如果编码重写特定的寄存器，则用跟在输入操作码后面的第三个双点加以描述，后接 clobbered 寄存器的名称，作为一个由逗号分隔的字符串列表。比如：

```
asm ("getid r11\n\tmov %0, r11"
     : "=r"(result)
     : /* no inputs */
     : "r11");
```

编译器确保 clobbered 寄存器中无输入或输出操作码。

如果一个 asm 陈述具有输出操作码，则编译器假设该陈述除了写输出操作码外没有其它副作用。如果未使用 asm 陈述写的值，则编译器可删除该 asm 陈述。为了将一个 asm 陈述标记为具有副作用，应在 asm 后面加上关键字 volatile。比如：

```
asm volatile("in res[%1], %0" : "=r"(result) : "r"(lock));
```

如果 asm 陈述访问存储器，则在 clobber 寄存器列表中加入“memory”。比如：

```
asm volatile("stw %0, dp[0]"
     : /* no outputs */
     : "r"(value));
```

这能防止编译器将存储器值缓存至 asm 陈述周围的寄存器中。

可利用 earlyclobber 约束编译器“&”规定，在所有输入操作码被消耗完之前修改一个输出操作码。这能防止编译器将该操作码与其它输入操作码放入同一寄存器中。比如：

```
asm("or %0, %1, %2\n"
    "or %0, %0, %3\n"
    : "=&r"(result)
    : "r"(a), "r"(b), "r"(c));
```

不支持从一个 asm 陈述转移到另一个 asm 陈述。不得将 asm 陈述用来修改任一资源的事件启用状态。

34 编写与 XMOS XS1 ABI 兼容的汇编程序

在本章中

XMOS XS1 应用程序二进制接口（ABI）定义了用于由 C/C++、XC 和汇编码编译而来的对象的链接接口。本指南说明了如何写能够链接到 XMOS 编译器生成的对象的汇编码函数。

34.1 符号

由于汇编程序解析一个汇编文件，因此它能保持一个当前地址，每次当它分配存储区时，地址都会增加。

符号用来关联名称与地址。符号可在指令和指示中被引用，一旦值被计算出来，连接程序就会修补相应的地址。

以下程序定义了一个名为 f 的符号，指的是当前地址的值。它还让该符号全局可见，从其它文件也可查看，并且通过名称就能引用该符号。

```
# Give the symbol f the value of the current address.
f:
# Mark the symbol f as global.
  .globl f
```

通过写符号名称再加一点双点即可定义该符号。.globl 指令使符号从文件外部也可以看到。

34.2 对齐

XS1 ABI 规定了针对编码和数据的最低对齐要求。函数必须以对齐的 2 字节开头，数据必须逐字对齐。在一个符号的定义前加入.align 指令可使地址对齐。

以下程序定义了一个符号 f，确定其为下一个 2 字节对齐地址。

```
# Force 2 byte alignment of the next address.
  .align 2
f:
```

34.3 章节

各目标文件可能包含多个章节。当通过连接程序结合时，各目标文件中名称相同的章节将一起被放在连续的地址上。这样，在最终的可执行文件中，不同类型的编码或数据就可以分在一组。

XS1 ABI 要求将函数放在.text 章，将只读数据放在.cp.rodata 章，将可写数据放在.dp.data 章。默认章节为.text，可通过以下指令中的任意一个改变当前章节。

Shiqiang Xiao 13-11-12 2:09 PM
已删除: 章

图 51：章节用于指令

XMOS 连接程序支持的章节			
.text	可执行编码	.text	
.dp.data	可写数据	.section .dp.data, “awd”, @progbits	
.cp.data	只读数据	.section .cp.rodata, “ac”, @progbits	

34.3.1 数据

以下示例程序定义了一个用值 5 初始化、在一个 4 字节边界对齐的 4 字节可写对象。

```
  .section .dp.data, "awd", @progbits
  .align 4
x:
  .word 5
```

你可以利用以下指令发出不同类型的数据。

图 52：指令描述

指令	描述
.byte	发出一个 1 字节的值
.short	发出一个 2 字节的值
.word	发出一个 4 字节的值
.space	发出一个零初始化存储的 *n* 字节数组，其中 *n* 是指指

不同类型的数据的指令	令的参数	
	.asciiz	发出一个空结尾 ASCII 字符串
	.ascii	发出一个 ASCII 字符串（无隐含的终止字符）

34.3.2 数组

以下程序定义了一个大小为 42 字节的全局数组。

```
  .section .dp.data, "awd", @progbits
  .globl a
  .align 4
a:
  .space 42
  .globl a.globound
  .set a.globound, 42
```

XS1 ABI 要求每个全局数组 f 都具有一个相应的全局符号 f.globound，该符号用数组第一维度的元素数初始化。你可以利用.set 指令实施初始化。需要注意的是，如果该变量被一个 XC 函数使用，则将这个值用于数组界检查。

34.4 函数

XS1 ABI 规定了在函数之间传递参数和返回值的规则。此外，它还确定了用于规定函数所需硬件资源数量的符号。

34.4.1 参数和返回值

至多 32 位的标量值被当作 32 位值传递。最初的四个参数在寄存器 *r0*、*r1*、*r2* 和 *r3* 中传递，附加参数在堆栈中传递。同样，最初的四个返回值也在寄存器 *r0*、*r1*、*r2* 和 *r3* 中返回，附加值则在堆栈中返回。

在下列 XC 函数原型中，参数 a 和 b 在寄存器 *r0* 和 *r1* 中传递，返回值也一样。

```
{int, int} swap(int a, int b);
```

此函数的一个汇编实现如下所示：

```
  .globl swap
  .align 2
swap:
  mov r2, r0
  mov r0, r1
  mov r1, r2
  retsp 0
```

34.4.2 调用者和被调用者保存寄存器

XS1 ABI 规定寄存器 *r0*、*r1*、*r2*、*r3* 和 *r11* 为调用者保存寄存器，其它所有寄存器则为被调用者保存寄存器。

调用一个函数之前，必须保存其值在调用完成后还需用到的所有调用者保存寄存器的内容。从一个函数返回后，所有被调用者保存寄存器的内容必须与其进入函数前的内容相同。

以下示例显示了一个采用被调用者保存寄存器 *r4*、*r5* 和 *r6* 的函数的序言和尾声。序言复制寄存器值至堆栈，尾声则将值从堆栈归还至寄存器。

```
# Prologue
  entsp 4
  stw r4, sp[1]
  stw r5, sp[2]
  stw r6, sp[3]

# Main body of function goes here
# ...

# Epilogue
  ldw r4, sp[1]
  ldw r5, sp[2]
  ldw r6, sp[3]
  retsp 4
```

34.4.3 资源使用

连接程序尝试计算各函数所需的资源数，包括其存储要求、线程数、通道传输结束及其使用的计时器等。这使连接程序能够检查确定最终可执行文件的资源使用不超过目标设备上可得的资源。

对函数 f，XS1 ABI 定义的资源使用符号如下所示：

图 53： XS1 ABI 定义的资源使用符号

符号	描述
f.nstackwords	堆栈大小
f.maxthreads	所分配的最大线程数，包括当前线程
f.maxchanends	所分配的最大通道传输结束数量
f.maxtimers	所分配的最大计时器数量

你可以利用.linkset 指令定义资源使用符号。如果是全局函数，你也可以将资源使用符号做成全局的。

以下示例程序定义了函数 f 的资源使用符号，该函数 f 采用 4 字堆栈，2 条线程，0 个计时器和 2 个通道传输结束。

```
.globl f
.globl f.nstackwords
.linkset f.nstackwords, 5
.globl f.maxthreads
.linkset f.maxthreads, 2
.globl f.maxtimers
.linkset f.maxtimers, 0
.globl f.maxchanends
.linkset f.maxchanends, 2
```

在更为复杂的情况下，你可以利用最大（$M）和附加（+）运算符生成资源使用表达式，由连接程序求值。如果依次调用两个方程，那么你应该计算两个方程的最大值，但如果是并行调用，你则应该计算两个方程的总和。

以下示例程序定义了函数 f 的资源使用符号，该函数 f 将堆栈扩展至 10 字，分配两个计时器，并在释放计时器和返回之前依次调用函数 g 和函数 h。

```
.globl f
.globl f.nstackwords
.linkset f.nstackwords, 10 + (g.nstackwords $M h.nstackwords)
.globl f.maxthreads
.linkset f.maxthreads, 1 + ((g.maxthreads-1) $M (h.maxthreads-1))
.globl f.maxtimers
.linkset f.maxtimers, 2 + (g.maxtimers $M h.maxtimers)
.globl f.maxchanends
.linkset f.maxchanends, 0 + (g. maxchanends $M h.maxchanends)
```

如果某资源使用符号的值未知，那么你可以省略其定义，例如，如果函数通过一个函数指针进行间接调用的情况。但是，如果需要符号的值来满足程序中的重新安置，则程序将无法链接。

34.4.4 副作用

XC 语言要求在 select 陈述中用作布尔保护信息的函数无任何副作用。此外，它还要求从某交易陈述内部调用的函数不得宣布通道传输。除非你将下列符号明确设定为零，否则默认情况下，假设函数 f 有副作用，并且会宣布通道传输。

图 54：	符号	描述
指示		
副作用	f.locnoside	规定函数是否有副作用
的符号	f.locnochandec	规定函数是否分配通道传输结束

34.5 删除块

连接程序可以删除未使用的编码和数据。待删除的编码和数据必须放在删除块中。在最终链接时间，如果某删除块内的所有符号均未被引用，则将该删除块从最终图像中删除。

以下示例程序宣布了一个删除块中的一个符号。

```
  .cc_top f.function, f
f:
  .cc_bottom f.function
```

.cc_top 指令和.cc_bottom 指令的首个参数是删除块的名称。.cc_top 指令具有一个附加参数，即作为块删除的基础的一个符号。如果参考了该符号，则不删除块。

各删除块都必须有一个名称，该名称在汇编文件中是唯一的。

34.6 类型字符串

一个类型字符串是指用来描述一个函数或变量的类型的字符串。类型信息编码为一个类型字符串是由 XS1 ABI 规定的。利用以下指令关联一个类型字符串和一个符号。

联编	指令
图 55： 类型字符串 指令	全局 .globl name, “typestring” 外部 .extern name, “typestring” 局部 .locl name, “typestring”

当来自某目标文件的一个符号与另一目标中名称相同的一个符号匹配时，连接程序检查类型字符串是否兼容。如果类型字符串兼容，链接照常进行。如果类型字符串是函数类型，区别仅在于是否存在数组界参数，则连接程序将生成一个形实转换程序，并用这个形实转换程序替换符号的使用，以解释参数中的区别。出现其它任何类型字符串不

匹配，连接程序均错误。这确保了根据多个文件编制的程序与根据单个文件编制的程序一样健壮。

如果你不能为一个符号发出一个类型字符串，则假设与该符号的对比兼容。如果你正在执行一个带有未知大小数组的函数，则你应当发出一个类型字符串，以便从 C 和 XC 调用它。在其它情况下，类型字符串可以忽略，但不进行错误检查。

34.7 示例

以下程序将文字“Hello world”输出为标准输出。

```
const char str[] = "Hello world";

int main() {
  printf(str);
  return 0;
}
```

以下汇编实现符合 XS1 ABI。

```
  .extern printf, "f{si}(p(c:uc),va)"
  .section .cp.rodata, "ac", @progbits
  .globl str, "a(12:c:uc)"
  .cc_top str.data, str
  .align 4
str :
  .asciiz "Hello world"
.cc_bottom str.data
  .globl str.globound
  .set str.globound, 12

  .text
  .globl main, "f{si}(0)"
.cc_top main.function, main
  .align 2
main:
  entsp 1
  ldaw r11, cp[str]
  mov r0, r11
  bl printf
  ldc r0, 0
  retsp 0
.cc_bottom main.function
  .globl main.nstackwords
  .linkset main .nstackwords, 1 + printf.nstackwords
  .globl main.maxthreads
  .linkset main.maxthreads, printf.maxthreads
  .globl main.maxtimers
  .linkset main.maxtimers, 0 + printf.maxtimers
  .globl main.maxchanends
  .linkset main.maxchanends, 0 + printf.maxchanends
  .linkset main.locnochandec, 1
  .linkset main.locnoside, 1
```

通过定义资源使用的符号，连接程序可以检查程序是否适合某目标设备。通过提供类型字符串，连接程序可以在链接不同的目标文件时检查类型的兼容性。连接程序可以删除位于删除块中的未使用的编码和数据。

35 汇编语言编程手册

在本章中

XMOS 汇编语言支持对象在带有 DWARF 3[7] 调试信息的可执行和与可链接格式（ELF）[6] 中形成。至 ELF 格式的扩展记录在 XMOS 应用程序二进制接口中（见 §40）。

35.1 词法约定

一共有六种类型的指令：符号名称、指令、常量、运算符、指令助记符及其它分隔符。忽略空格、制表符、换页符及注释，但它们是单独指令的情况除外。

Shiqiang Xiao 13-11-12 3:03 PM
已删除: 令牌

Shiqiang Xiao 13-11-12 3:03 PM
已删除: 令牌

35.1.1 注释

字符#引出一个注释，该注释由换行符终止。注释不存在于字符串常量中。

35.1.2 符号名称

符号名称以一个字母或字符‘.’、‘-’或‘$’中的任意一个开头，后接一串字母、数字、句号、下划线和美元符号（任选）。大写字母和小写字母不同。

35.1.3 指令

指令以‘.’开头，后接一个或多个字母。指令指示汇编程序实施某些行为（见 §35.6）。

---

[6] http://www.xmos.com/references/elf
[7] http://www.xmos.com/references/dwarf3

35.1.4 常数

常数为一个整形，字符常数或字符串常量。

·二进制整形为 0b 或 0B，后跟零个或多个数字 01。

·八进制整形为 0，后跟零个或多个数字 01234567。

·十进制整形为非零位，后跟零个或多个数字 0123456789。

·十六进制整形为 0x 或 0X，后跟一个或多个数字和字母 0123456789abcdefABCDEF。

·字符常数是由单引号包围的字符序列。

·字符串常量是由双引号包围的字符序列。

C 转义序列可用于指定某些字符。

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

35.2 区段和重定位

命名 ELF 区段由命令（见§35.6.10）指定。此外，有一个唯一的未命名“绝对”区段和一个唯一的未命名“未定义”区段。记号{secname X}是指“offset X into section secname.”

绝对区段中的符号值不受重定位影响。例如，地址{absolute 0}“重定位”至运行时间地址 0。未定义区段中的符号值未设定。

汇编器保持着当前区段的轨迹。当前区段最开始设定为文本区段。命令可用于改变当前区段。可重定位存储区的汇编指令和命令在当前区段中发出。对于每个区段，汇编器维护在区段中保存当前偏移量的位置计数器。*active location counter* 是指当前区段的位置计数器。

35.3 符号

每个符号只有一个名称。汇编程序中的每个名称即是一个符号。本地符号是以字符“.L”开头的任何符号。本地符号可由连接器在连接不再需要时丢弃。

35.3.1 属性

每个符号都有一个 *value*，一个相关区段和一个 *binding*。符号通过 *set* 或 *linkset* 命令（见§35.6.12），或通过其在标签（见§35.4）中的应用赋值。符号在未定义区段的默认绑定为 *global*；所有其他符号的默认绑定则为 *local*。

35.4 标签

标签是后面紧跟冒号（：）的一个符号名。符号的值设为现用位置计数器的当前值。符号的区段设为当前区段。符号名不能出现在一个以上标签中。

35.5 表达式

表达式特指一个地址或值。一个表达式的结果必须为绝对数或特定区段内的偏移量。如果表达式的所有符号均被定义且其求值结果为一个常数，则该表达式为*constant expression*。如果表达式为一个常数表达式，一个符号或一个符号+-一个常数之一，则该表达式为简单表达式。表达式可在 ELF-扩展表达式区段中进行编码且由连接器计算其值（见§35.6.12）；编码方案由 ABI 确定。表达式的语法为：

```
expression    ::= unary-exp
                | expression infix-op unary-exp
                | unary-exp ? unary-exp $: unary-exp

unary-exp     ::= argument
                | prefix-op unary-exp

argument      ::= symbol
                | constant
                | ( expression )

infix-op      ::= one of
                  + - < > <= >= || << >> * $M $A & /

prefix-op     ::= one of
                  - ~ $D
```

当 *section* 是命名区段，绝对区段或未定义区段之一时，符号的求值结果为{section x }，且 x 是有符号的二进制补码为 32 位的整形。

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

插入运算符有与 C 相同的优先级和行为，且有相同优先级的运算符从左至右执行，此外，$M 运算符有最低优先级，而$A 运算符有最高优先级。

对于+和-运算符，图 56 中提供有效运算和结果的集。对于$D 运算符，参数必须为符号；结果为 1，当符号被定义时；否则为 0。

?运算符用于从符号中选择：如果第一操作数为非零，则结果为第二操作数，否则结果为第三操作数。

运算符 左操作数 右操作数 结果

Op	Left Operand	Right Operand	Result
+	{*section* x}	{absolute y}	{*section* x+y}
+	{absolute x}	{*section* y}	{*section* x+y}
+	{absolute x}	{absolute y}	{absolute x+y}
-	{*section* x}	{*section* y}	{absolute x-y}
-	{*section* x}	{absolute y}	{*section* x-y}
-	{absolute x}	{absolute y}	{absolute x-y}

图 56: + 和 -运算符的有效运算

对于所有其他运算符，两个参数必须为绝对数且结果也为绝对数。$M 运算符返回两个操作数的最大值，$A 运算符则返回与第二操作数对齐的第一操作数的值。当需要绝对表达式时，如果省略，则假定{absolute 0}。

35.6 命令

命令指示汇编器执行某一操作。该区段提供支持命令。

35.6.1 align

align 命令将现用位置计数器填补至指定存储边界，其语法为：

align-directive ::= .align expression

表达式必须为常数表达式，其值必须为 2 的幂。该值指定以字节的单位的所需对齐。

35.6.2 ascii, asciiz

ascii 命令将每个字符串汇编至连续地址中。asciiz 命令也一样，除非每个字符串后跟一个空字节。

ascii-directive ::= .ascii *string-list*
| .asciiz *string-list*

string-list ::= *string-list* , *string*
| .asciiz *string*

35.6.3 byte, short, int, long, word

这些命令为每个表达式发出一个数字，即在运行时间该表达式的值。字节顺序由目标架构的字节顺序决定。用字类命令发出的数字的大小由目标架构上自然字类的大小决定。用其他命令发出的数字的大小由 ABI 中相应类型的大小决定。

```
value-directive   ::= value-size exp-list

value-size        ::= .byte
                    | .short
                    | .int
                    | .long
                    | .word

exp-list          ::= exp-list , expression
                    | expression
```

35.6.4 file

file 命令有两种形式。

```
file-directive    ::= .file string
                    | .file constant string
```

当与一个参数一起使用时，file 命令通过类型 *STT_FILE* 和指定字符串值创建一个 ELF 符号表条目。该条目确保为字符表中的第一条目。

当与两个参数一起使用时，file 命令添加一个条目至 DWARF 3.debug_line 文件名表格中。第一个参数为唯一的正整形，用作表中条目的索引。第二个参数为文件名。

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

35.6.5 loc

.loc 命令添加一行至 DWARF 3 .debug_line 行号矩阵中。

```
loc-directive     ::= constant constant constant_opt
                    | constant constant constant (loc-option)*

loc-option        ::= basic_block
                    | prologue_end
                    | epilogue_begin
                    | is_stmt constant
                    | isa constant
```

地址寄存器设为现用位置计数器。开始两个参数分别设定文件和行寄存器。可选的第三个参数设定列寄存器。其他参数设定.debug_line 状态机中的更多寄存器。
basic_block
将 basic_block 设置为真
prologue_end
将 prologue_end 设置为真
epilogue_begin
将 epilogue_begin 设置为真
is_stmt
将 is_stmt 设置为规定值，必须是 0 或 1。
isa
将 isa 设置为规定值

35.6.6 weak

weak 命令设定指定符号上的弱属性。

```
weak-directive    ::= .weak symbol
```

35.6.7 globl, global, extern, locl, local

globl 命令使指定符号在连接期间可被其他对象看见。extern 命令特指符号在另一个对象中定义。locl 命令特指符号具有本地绑定，

```
visibility        ::= .globl
                  |   .extern
                  |   .locl
                  |   .global
                  |   .extern
                  |   .local

vis-directive     ::= visibility symbol
                  |   visibility symbol , string
```

如果提供可选字符串，在包含符号和字符串表格索引的 ELF 扩展型区段中创建 SHT_TYPEINFO 条目，所述字符串表格的条目包含指定字符串。（如果字符串已不存在于字符串表格中，则其被插入。）该字符串的意义由 ABI 决定。
global 和 local 命令为 globl 和 locl 命令的同义词，其用于兼容其他汇编器。

35.6.8 typestring

typestring 添加一个 SHT_TYPEINFO 条目至包含符号和字符表格索引的 ELF 扩展型区段中，所述字符表格索引的条目包含指定字符串。（如果字符串已不存在于字符串表格中，则其被插入。）该字符串的意义由 ABI 决定。

typestring-directive ::= .typestring symbol , string

35.6.9 ident, core, corerev

这些命令每个都创建一个 ELF 备注区段，命名为“.xmos_note.”

info-directive ::= .ident string
| .core string
| .corerev string

该区段的内容为一个（名称，类型，值）三元组：名称为 xmos；类型为 IDENT, CORE 或 COREREV；值为指定字符串。

35.6.10 section, pushsection, popsection

区段命令改变当前 ELF 区段（见§35.2）。

section-directive ::= sec-or-push name
| sec-or-push name , flags sec-typeopt
| .popsection
sec-or-push ::= .section
| .pushsection
flags ::= string
sec-type ::= type
| type , flag-args
type ::= @progbits
| @nobits
flag-args ::= string

紧跟 section 或 pushsection 命令的编码汇编且增补至命名区段。可选标记可包含下列字符的任意组合。
a 区段为可分配的
c 区段位于全局常数存储库中
d 区段位于全局数据区中
w 区段为可写入的
x 区段为可执行的
M 区段为可合并的
S 区段包含以零结尾的字符串

可选类型参数 progbits 特指包含数据的区段；nobits 特指不包含的区段。
如果 M 符号指定为标记，类型参数必须指定且必须提供一个整形作为标记特定参数。标记特定参数代表区段中数据条目的实体尺寸。例如：
.section .cp.const4, "M", @progbits, 4
有 M 标记但无 S 标记的区段必须包含固定大小的常数，每个为 flag-args 字节长。
有 M 和 S 标记的区段必须包含以零结尾的字符串，每个字符为 flag-args 字节长。
连接器可除去区段内有相同名称，实体大小和标记的重复条目。
名称相同的每个区段必须有相同类型和标记。section 命令用指定区段替代当前区段。pushsection 命令推动当前区段至 section stack 的顶部，然后用指定区段取代当前区段。popsection 命令用区段堆叠顶部的区段替代当前区段，然后从堆叠中取出该区段。

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

35.6.11 text
text 命令将当前 ELF 区段变为.text 区段。区段类型和属性由 ABI 决定。
text-directive ::= .text
35.6.12 set, linkset
符号通过 set 或 linkset 命令赋值。
set-directive ::= set-type symbol , expression
set-type ::= .set
| .linkset

set 命令用表达式的值定义命名符号。表达式必须为一个常数或一个符号；如果表达式是一个常数，符号定义在绝对区段中；如果表达式是一个符号，定义符号继承该符号其区段信息和其他属性。
linkset 命令也一样，除非表达式未确定值；反而，在 ELF 扩展表达式区段中创建一个或多个 SHT_EXPR 条目，共同形成一个表达式树表示。
汇编码中所用的任何符号可能是 SHT_EXPR 条目的目标，在这种情况下，其值由连接器在表达式中所有其他符号的值已知时通过确定表达式的值来计算。这可在任何增量连接阶段发生。一旦值已知，则可设定符号的值且将表达式条目从连接对象中除去。

35.6.13 cc_top, cc_bottom

cc_top 和 cc_bottom 命令用于标记消除块的开始和结尾。

```
cc-top-directive    ::= .cc_top name , predicate
                    |   .cc_top name

cc-directive        ::= cc-top-directive
                    |   .cc_bottom name

name                ::= symbol

predicate           ::= symbol
```

名称相同的 cc_top 和 cc_bottom 命令是指相同的消除块。一个消除块必须只有一个 cc_top 命令和一个 cc_bottom 命令。消除块的顶部和尾部必须在相同区段内。消除块包含该区段内 cc_top 和 cc_bottom 命令见的数据和标签。消除块不可相交。消除块重叠是非法的。
如果下列之一为真，消除块最终仍然是可执行的：
·消除块内的标签引用自消除块外的位置。
·消除块内的标签引用自未消除的消除块。
·谓词符号定义在消除块外或包含在未消除消除块内。
如果这些条件均不为真，则消除块从最终可执行文件中删除。

35.6.14 scheduling

scheduling 命令启用或禁用指令排程。当启用排程时，汇编器可重新排列指令以最小化 FNOP 的数目。默认排程模式由命令行选项-fschedule (见 §9.4)决定。

```
scheduling-directive    ::= .scheduling

scheduling-mode         ::= on
                          | off
                          | default
```

35.6.15 syntax

syntax 命令改变当前语法模式。在每个模式中如何指定汇编指令详见§35.7。

```
syntax-directive    ::= .syntax syntax

syntax              ::= default
                      | architectural
```

35.6.16 assert

assert-directive ::= .assert constant , symbol , string

assert 命令在产生可执行对象前需要一个待测试断言：如果符号有非零值，断言失败。如果常数为 0，失败应报告为警告；如果常数为 1，失败应报告为错误。字符串是汇编器或连接器发出的失败消息。

35.6.17 语言命令

语言命令在 ELF 扩展表达式区段中创建条目；编码由 ABI 确定。

```
xc-directive    ::= globdir symbol , string
                  | globdir symbol , symbol , range-args , string
                  | .globpassesref symbol , symbol , string
                  | .call symbol , symbol
                  | .par symbol , symbol , string

range-args      ::= expression , expression
```

```
globdir     ::= .globread
              | .globwrite
              | .parwrite
              | .globpassesref
```

对于每个命令，字符串是汇编器或连接器在遇到由命令引起的错误时显示的消息。

call

两个符号必须有函数类型。该命令设置属性，即第一函数可调用第二函数。

par

两个符号必须有函数类型。该命令设置属性，即第一函数与第二函数并行调用。

globread

第一符号必须有函数类型且第二命令必须有对象类型。该命令设置属性，即函数可读取对象。当指定一范围，第一表达式是读取地址的变量字节的开始偏移量，而第二表达式则是读取字节的大小。

globwrite

第一符号必须有函数类型且第二命令必须有对象类型。该命令设置属性，即函数可写入对象。当指定一范围，第一表达式是写入地址的变量字节的开始偏移量，而第二表达式则是写入字节的大小。

parwrite

第一符号必须有函数类型且第二命令必须有对象类型。该命令设置属性，即在写入和功能调用的求值顺序未定义时写入对象的表达式中调用函数。当指定一范围，第一表达式是写入地址的变量字节的开始偏移量，而第二表达式则是写入字节的大小。

globpassesref

第一符号必须有函数类型且第二命令必须有对象类型。该命令设置属性，即可引用函数来传递对象。

35.6.18 XMOS 定时分析器命令

Shiqiang Xiao 13-11-12 9:31 PM
已删除: 计时

XMOS 计时分析器命令添加计时数据库至 ELF 区段中。

xta-directive ::= .xtabranch *exp-list*$_{opt}$
| .xtaendpoint *string* , *source-location*
| .xtacall *string* , *source-location*
| .xtalabel *string* , *source-location*
| .xtathreadstart
| .xtathreadstop
| .xtaloop *constant*
| .*xtacommand string* , *source-location*

source-location ::= *string* , *string* , *constant*

源位置的第一字符串为编译目录。第二字符串为文件路径。路径可指定为自编译目录起的相对路径或绝对路径。第三参数为行号。

·xtabranch 特指可从当前位置分支或分支至当前位置的位置逗号分隔列表。

·xtaendpoint 用指定标签将当前位置标记为终点。

·xtacall 用指定标签将当前位置标记为函数调用。

·xtathreadstart 特指可初始化以在当前位置开始执行的线程。

·xtathreadstop 特指在当前位置执行指令的线程将不执行任何进一步指令。

·xtaloop 特指包含当前位置的最内层循环执行指定的次数。

·xtacommand 特指在分析可执行性时待执行的 XTA 命令。

35.6.19 uleb128, sleb128

下列命令为表达式的逗号分隔列表中的每个表达式发出一个值，编码有符号的或无符号的 DWARF 小端基 128 数。

leb-directive ::= .uleb128 exp-list
| .sleb128 exp-list

35.6.20 space, skip

Space 命令发出字节序列，由第一表达式指定，有填充值的每个字节序列由第二表达式指定。两个表达式必须为常数表达式。

```
space-or-skip     ::= .space
                  |  .skip

space-directive   ::= space-or-skip expression
                  |  space-or-skip expression , expression
```

skip 命令是空格命令的同义词，其用于兼容其他汇编器。

35.6.21 类型

type 命令特指符号类型，为函数类型或对象类型。

```
type-directive    ::= .type symbol , symbol-type

symbol-type       ::= @function
                  |  @object
```

35.6.22 尺寸

尺寸命令特指与符号相关的尺寸。

size-directive ::= .size symbol , expression

35.6.23 jmptable, jmptable32

Jmptable 和 jmptable32 命令生成一个无条件分支指令表。每个分支指令的目标为列表中的下一个标签。每个分支指令的尺寸为 jmptable 命令 16 位，jmptable32 命令 32 位。

jmptable-directive ::= .jmptable jmptable-listopt
| .jmptable32 jmptable-listopt
jmptable-list ::= symbol
| jmptable-list symbol

每个符号必须为一个标签。最多可指定 32 个标签。如果无条件分支距离不适于 16 位分支指令，分支可作为表格末尾的弹簧，执行到目标标签的分支。

35.7 指令

汇编指令通常被插入 ELF 代码段。指令语法为：

instruction ::= *mnemonic instruction-args*$_{opt}$

instruction-args ::= *instruction-args , instruction-arg*
| *instruction-arg*

instruction-arg ::= *symbol [expression]*
| *symbol [expression] : symbol*
| *expression*

汇编指令总结如下，引用 XS1 架构手册（见 X7879)。本册中，构架格式为记录型。语法指令用来
切换到此编码模式。
使用以下符号：

bitp 1, 2, 3, 4, 5, 6, 7, 8, 16, 24 及 32 其中之一
b 以寄存器为基址
c 以寄存器为条件运算元
d, e 以寄存器为目的运算元
i 以寄存器为索引运算元
r 以寄存器为资源标示
s 以寄存器为源运算元
t 以寄存器为线程标示符
us 小的无符号常量从 0 至 11
ux 无符号常量 0...(2x -1)
v, w, x, y 寄存器用于两个及以上源运算元
寄存器为 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, sp, dp, cp 和 lr 其中之一. 该指令决定了哪一个指令被允许。

只要有指令格式可选择，汇编器选择尺寸最小的格式。以强制一个特定格式、指定 INSTRUCTION_format 形式的助记符与架构手册中描述的指令与格式名称一致。例如，LDWCP_ru6 助记符指定 LDWCP 指令的 ru6 格式。
下表为 XS1 架构手册中指令的页面数（见 X7879)。

335.7.1	数据存取			
助记符	运算元	规格	语义	页数
ld16s	d, b[i]	16	加载符号 16 位	121
ld8u	d, b[i]	16	加载无符号 8 位	122
lda16	d, b[i]	32	增加 16-bit 地址	123
lda16	d, b[-i]	32	减去 16-bit 地址	124
ldap	r11, u20	16/32	加载 pc-relative 地址	125
ldap	r11, -u20	16/32	加载 pc-relative 地址	126
ldaw	d, b[i]	32	增加一个字地址	131
ldaw	d, b[-i]	32	删减一个字地址	127
ldaw	d, b[us]	32	即刻增加一个字地址	132
ldaw	d, b[-us]	32	即刻删减一个字地址	128
ldaw	r11, cp[u16]	16/32	在常量池加载字地址	129
ldaw	d, dp[u16]	16/32	在数据池加载字地址	130
ldaw	d, sp[u16]	16/32	在堆栈加载字地址	133
ldw	et, sp[4]	16	从堆栈加载 ET	135
ldw	sed, sp[3]	16	从堆栈加载 SED	137
ldw	spc, sp[1]	16	从堆栈加载 SPC	138
ldw	ssr, sp[2]	16	从堆栈加载 SSR	139
ldw	d, b[i]	16	加载字	140
ldw	d, b[us]	16	即刻加载字	141
ldw	d, cp[u16]	16/32	从常量池加载字	142
ldw	r11, cp[u20]	16/32	从常量池加载字	143
ldw	d, dp[u16]	16/32	从数据池加载字	144
ldw	d, sp[u16]	16/32	从堆栈加载字	145
set	cp, s	16	设置常量池	175
set	dp, s	16	设置数据指示器	177
set	sp, s	16	设置堆栈指示器	185
st16	s, b[i]	32	16-bit 存储	196
st8	s, b[i]	32	8-bit 存储	197
stw	sed, sp[3]	16	在堆栈存储 SED	199
stw	et, sp[4]	16	在堆栈存储 ET	198
stw	spc, sp[1]	16	在堆栈存储 SPC	200
stw	ssr, sp[2]	16	在堆栈存储 SSR	201
stw	s, b[i]	32	存储字	202
stw	s, b[us]	16	即刻存储字	203
stw	s, dp[u16]	16/32	在数据池存储字	204
stw	s, sp[u16]	16/32	在堆栈存储字	205

35.7.2 分支、跳跃及调用

助记符	运算元	规格	语义	页数
bau	s	16	绝对无条件分支	53
bf	c, u16	16/32	如为否，相对分支	63
bf	c, -u16	16/32	如为否，相对分支	60
bl	u20	16/32	相对分支与连接	59
bl	0	16/32	相对分支与连接	58
bla	s	16	通过寄存器绝对分支与连接	55
bla	cp[u20]	16/32	通过 CP 绝对分支与连接	56
blat	u16	16/32	通过表格绝对分支与连接	57
bru	s	16	通过寄存器无条件相对分支	66
bt	c, u16	16/32	如为是，相对分支	64
bt	c, -u16	16/32	如为是，相对分支	61
bu	u16	16/32	无条件相对分支	65
bu	0	16/32	无条件相对分支	62
entsp	u16	16/32	调整堆栈并储存连接寄存器	90
extdp	u16	16/32	扩展数据指示器	93
extsp	u16	16/32	扩展堆栈指示器	94
retsp	u16	16/32	返回	169

35.7.3 数据处理

助记符	运算元	规格	语义	页数
add	d, x, y	16	增加	47
add	d, x, us	16	即刻增加	48
and	d, x, y	16	位元 与	49
andnot	d, s	16	与非	50
ashr	d, x, y	32	算术右移	51
ashr	d, x, bitp	32	即刻算术右移	52
bitrev	d, s	32	位倒序	54
byterev	d, s	32	字节倒序	67
clz	d, s	32	增加了前导零计数	73
crc32	d, r, p	32	字的 CRC 校验	74
crc8	r, o, d, p	32	8 步 CRC 校验	75
divs	d, x, y	32	符号除法	79
divu	d, x, y	32	无符号除法	80
eq	c, x, y	16	均等	91
eq	c, x, us	16	即刻均等	92
ladd	e, d, x, y, v	32	长的无符号进位添加	120

（接下页）

Shiqiang Xiao 13-11-13 11:27 AM
已删除: 支化
Shiqiang Xiao 13-11-13 11:28 AM
已删除: 支化
Shiqiang Xiao 13-11-13 11:28 AM
已删除: 支化
Shiqiang Xiao 13-11-13 11:28 AM
已删除: 支化
Shiqiang Xiao 13-11-13 11:28 AM
已删除: 支化
Shiqiang Xiao 13-11-13 11:28 AM
已删除: 支化
Shiqiang Xiao 13-11-13 11:28 AM
已删除: 支化
Shiqiang Xiao 13-11-13 11:28 AM
已删除: 支化
Shiqiang Xiao 13-11-13 11:28 AM
已删除: 支化
Shiqiang Xiao 13-11-13 11:28 AM
已删除: 支化
Shiqiang Xiao 13-11-13 11:28 AM
已删除: 支化
Shiqiang Xiao 13-11-13 11:28 AM
已删除: 支化
Shiqiang Xiao 13-11-13 11:28 AM
已删除: 支化
Shiqiang Xiao 13-11-13 11:28 AM
已删除: 支化
Shiqiang Xiao 13-11-13 11:28 AM
已删除: 返还
Shiqiang Xiao 13-11-13 11:30 AM
已删除: and
Shiqiang Xiao 13-11-13 11:31 AM
已删除: 字
Shiqiang Xiao 13-11-13 11:31 AM
已删除: 8-step

助记符	运算元	规格	语义	页数
ldc	d, u16	16/32		134
ldivu	d, e, v, x,y	32	长的无符号划分	136
lmul	d, e, x, y,v, w	32	长的叠加	146
lss	c, x, y	16	少于符号	147
lsu	c, x, y	16	少于无符号	148
lsub	e, d, x, y,v	32	长的无符号差集	149
maccs	d, e, x, y	32	叠加积累符号	150
maccu	d, e, x, y	32	叠加积累无符号	151
mkmsk	d, s	16	制作掩码	153
mkmsk	d, bitp	16	立即制作掩码	154
mul	d, x, y	32	叠加	156
neg	d, s	16	二进制补码无效	157
not	d, s	16	按位否	158
or	d, x, y	16	按位或	159
rems	d, x, y	32	符号剩余	167

35.7.4 并发及线程同步

助记符	运算元	大小	语义	页数
freet		16	免费同步线程	96
get	r11, id	16	获取线程 ID	100
getst	d, res[r]	16	获取同步线程	108
mjoin	res[r]	16	主同步并加入	152
msync	res[r]	16	主同步	155
ssync		16	从同步	195
init	t[r]:cp, s	16	初始化线程的 CP	212
init	t[r]:dp, s	16	初始化线程的 DP	213
init	t[r]:lr, s	32	初始化线程的 LR	214

（接下页）

Shiqiang Xiao 13-11-13 11:32 AM
已删除: 规格

Shiqiang Xiao 13-11-13 11:32 AM
已删除: 96

Shiqiang Xiao 13-11-13 11:33 AM
已删除: 16

Shiqiang Xiao 13-11-13 11:32 AM
已删除: 免费同步线程

Shiqiang Xiao 13-11-13 11:33 AM
已删除: 195

Shiqiang Xiao 13-11-13 11:33 AM
已删除: 16

Shiqiang Xiao 13-11-13 11:33 AM
已删除: 从同步

助记符	运算元	大小	语义	页数
init	t[r]:pc, s	16	初始化线程的 PC	215
init	t[r]:sp, s	16	初始化线程的 SP	216
set	t[r]:d, s	16	在线程中设置储存器	218
start	t[r]	16	启动线程	219
tsetmr	d, s	16	主线程中设置储存器	217

Shiqiang Xiao 13-11-13 11:34 AM
已删除: 规格

35.7.5 通信

助记符	运算元	大小	语义	页数
chkct	res[r], s	16	控制符记测试	68
chkct	res[r], us	16	立即控制符记测试	69
getn	d, res[r]	32	获取网络	103
in	d, res[r]	16	输入数据	110
inct	d, res[r]	16	输入控制符记	111
int	d, res[r]	16	输入数据标记	114
out	res[r], s	16	输出数据	160
outct	res[r], s	16	输出控制符记	161
outct	res[r], us	16	立即输出控制标记	162
outt	res[r], s	16	输出数据符记	165
setn	res[r], s	32	设置网络	180
testlcl	d, res[r]	32	测试本地	209
testct	d, res[r]	16	控制符记测试	210
testwct	d, res[r]	16	控制符记位置测试	211

Shiqiang Xiao 13-11-13 11:34 AM
已删除: 规格

35.7.6 资源操作

助记符	运算元	大小	语义	页数
clrpt	res[r]	16	明确端口时间	71
endin	d, res[r]	16	结束当前输入	89
freer	res[r]	16	释放资源	95
getd	d, res[r]	32	获取资源数据	97
getr	d, us	16	分配资源	105
getts	d, res[r]	16	获取端口时间标记	109
in	d, res[r]	16	输入数据	110
inpw	d, res[r], bitp	32	输入部分字	112
inshr	d, res[r]	16	输入及右移	113
out	res[r], s	16	输出数据	160
outpw	res[r], s, bitp	32	输出部分字	163
outshr	res[r], s	16	输出数据并移位	164

Shiqiang Xiao 13-11-13 11:34 AM
已删除: 规格

（接下页）

助记符	运算元	大小	语义	页数
peek	d, res[r]	16	查看端口数据	166
setc	res[r], s	32	设置资源控制位	172
setc	res[r], u16	16/32	立即设置资源控制位	170
setclk	res[r], s	32	为资源置位时钟	174
setd	res[r], s	16	设置数据	176
setev	res[r], r11	16	设置环境矢量	178
setpsc	res[r], s	16	设置端口移数	182
setpt	res[r], s	16	设置端口时间	183
setrdy	res[r], s	32	设置端口准备输入	184
settw	res[r], s	32	设置端口传输端口	187
setv	res[r], r11	16	设置项目向量	188
syncr	res[r]	16	同步资源	208

Shiqiang Xiao 13-11-13 11:33 AM
已删除: 规格

35.7.7 事件处理

助记符	运算元	大小	语义	页数
clre		16	清除所有项目	70
clrsr	u16	16/32	清除 SR 中的位	72
edu	res[r]	16	禁用项目	85
eef	d, res[r]	16	如为否，启用项目	86
eet	d, res[r]	16	如为是，启用项目	87
eeu	res[r]	16	启用项目	88
getsr	r11, u16	16/32	从 SR 中获取位	107
setsr	u16	16/32	在 SR 中设置位	186
waitef	c	16	如为否，延期执行项目	220
waitet	c	16	若为是，延期执行项目	221
waiteu		16	延期执行项目	222

Shiqiang Xiao 13-11-13 11:33 AM
已删除: 规格

Shiqiang Xiao 13-11-13 11:33 AM
已删除: 70

Shiqiang Xiao 13-11-13 11:33 AM
已删除: 16

Shiqiang Xiao 13-11-13 11:33 AM
已删除: 清除所有项目

35.7.8 中断、异常及内核调用

助记符	运算元	大小	语义	页数
clrsr	u16	16/32	清除 SR 中的位	72
ecallf	c	16	如为否，引发异常	83
ecallt	c	16	如为是，引发异常	84
get	r11, ed	16	将 ED 纳入 r11	98
get	r11, et	16	将 ET 纳入 r11	99
get	r11, kep	16	获取内核入口点	101
get	r11, ksp	16	获取内核堆栈储存器	102
getsr	r11, u16	16/32	从 SR 获取 bits	107

Shiqiang Xiao 13-11-13 11:34 AM
已删除: 222

Shiqiang Xiao 13-11-13 11:34 AM
已删除: 16

Shiqiang Xiao 13-11-13 11:34 AM
已删除: 延期执行项目

Shiqiang Xiao 13-11-13 11:35 AM
已删除: 规格

（接下表）

助记符	运算元	大小	语义	页数
kcall	s	16	内核调用	115
kcall	u16	16/32	立即内核调用	116
kentsp	u16	16/32	切换至内核堆栈	117
krestsp	u16	16/32	从内核堆栈恢复堆栈指针	118
kret		16	内核返还	119
set	kep, r11	16	设置内核入口点	179
setsr	u16	16/32	在 SR 中设置 bits	186

35.7.9 调试

助记符	运算元	大小	语义	页数
dcall		16	导致调试中断	76
dentsp		16	保存并修改堆栈储存器进行调试	77
dgetreg	s	16	调试另一个线程储存器的读取	78
drestsp		16	恢复非调试堆栈指示器	81
dret		16	从调试中断返还	82
get	d, ps[r]	32	获取处理器状态	104
set	ps[r], s	32	设置处理器状态	181

35.7.10 伪指令

默认语法模式中，汇编器支持一小组伪指令。此类指令不存在于处理器上，但会经汇编器翻译进入 XS1 指令。

助记符　　运算元　　释义

　　add d, s, 0

nop　　r0, r0, 0

mov	运算元	定义
dcall	d, s	加入 d, S, 0
nop		r0, r0, 0

Shiqiang Xiao 13-11-13 11:35 AM
已删除: 规格

Shiqiang Xiao 13-11-13 11:35 AM
已删除: 119

Shiqiang Xiao 13-11-13 11:35 AM
已删除: 16

Shiqiang Xiao 13-11-13 11:35 AM
已删除: 内核返还

Shiqiang Xiao 13-11-13 11:35 AM
已删除: 规格

Shiqiang Xiao 13-11-13 11:35 AM
已删除: 76

Shiqiang Xiao 13-11-13 11:35 AM
已删除: 16

Shiqiang Xiao 13-11-13 11:35 AM
已删除: 导致调试中断

Shiqiang Xiao 13-11-13 11:36 AM
已删除: 16

Shiqiang Xiao 13-11-13 11:36 AM
已删除: 保存并修改堆栈储存器进行调试

Shiqiang Xiao 13-11-13 11:36 AM
已删除: 77

Shiqiang Xiao 13-11-13 11:36 AM
已删除: 81

Shiqiang Xiao 13-11-13 11:36 AM
已删除: 16

Shiqiang Xiao 13-11-13 11:36 AM
已删除: 恢复非调试堆栈指示器

Shiqiang Xiao 13-11-13 11:36 AM
已删除: 82

Shiqiang Xiao 13-11-13 11:36 AM
已删除: 16

Shiqiang Xiao 13-11-13 11:36 AM
已删除: 从调试中断返还

35.8 汇编程序

汇编程序由语句序列组成。

program ::= ⟨*statement*⟩*

statement ::= *label-list*$_{opt}$ *dir-or-inst*$_{opt}$ *separator*

label-list ::= *label*
| *label-list label*

dir-or-inst ::= *directive*
| *instruction*

separator ::= *newline*
| ;

directive ::= *align-directive*
| *ascii-directive*
| *value-directive*
| *file-directive*
| *loc-directive*
| *weak-directive*
| *vis-directive*
| *text-directive*
| *set-directive*
| *cc-directive*
| *scheduling-directive*
| *syntax-directive*
| *assert-directive*
| *xc-directive*
| *xta-directive*
| *space-directive*
| *type-directive*
| *size-directive*
| *jmptable-directive*

第N部分 XS1设备编程

内容

36 XS1设备的XCC目标依赖型行为

本章内容
-时钟模块支持
-端口支持
-通道通讯

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

本章描述了适用于 XS1 架构的 XMOS 编译器套装行为。

36.1 时钟模块支持

XS1设备提供单个基准时钟，以外部振荡器产生的频率运转。XC要求系统设计员保证基准时钟以100MHz的频率运转，以确保计时器正常工作。
xCORE模块提供了一整套可编程时钟模块，可用来产生时钟信号。

名为< xs1.h >的头文件对时钟类型进行了界定。extern中没有相关声明时，必须采用时钟模块表达式对时钟变量进行初始化，例如：

```
clock c = XS1_CLKBLK_1;
```

设备数据表中列出了可用的时钟模块数。其名称如上述声明，从1开始按顺序对其编号。

XC中时钟为高精度时钟，其附加规则如下：

- 结构中描述了这类时钟的构成部分，只能通过外部声明来描述带该类型时钟的结构变量。
- 变量声明加前缀“on”可以表示该类型时钟对象，该类型时钟不能描述自动变量。

36.2 端口支持

XC端口说明

XC端口说明端口p描述了源端口。在XS1设备中，所有用于数据输入输出的端口都通过一个100MHz的基准时钟（见§36.1）计时，并使用单入口缓冲器，即使端口说明并不符合缓冲关键词。

图57中表格根据XC说明是否符合缓冲关键词，来确定XS1端口上支持哪些输入/输出操作。

图57：XS1端口上支持的输入、输出操作	模式	操作		
		序列化	选通	@ 时间
	不符合	×	×	×
	缓冲	√	√	√

编译器监测并发布下列各项中出现的故障：

- 序列化：使用与端口宽度不同的输送宽度来描述不符合缓冲的端口。
- 选通：配置不符合缓冲的端口，使用即入即出信号。
- 同时使用@和when的输入语句：输入语句不符合缓冲时，将同时使用这两种运算符

36.2.1 序列化

```
par {
  p @ t <: x;
  q @ t :> y;
}
```

注意使用序列化时，计时输入语句规定的时间记录了最后一比特数据取样时间。由于该结构使q上的输入完成时立即开始p上的输出，因此使用序列化时会产生出人意料的结果。

为并行输入、输出该数据，将传输宽度除以端口宽度得到的数值在软件中补偿输入时间。

36.2.2时间戳

输入语句记录的时间戳一般出现在数据取样之后，这是因为XS1为数据输入和时间戳输入分别提供不同的指令，所以下一数据取样之后才能输入时间戳。这一问题同样影响输出，但不会影响在选择语句保护下执行的输入。输入或输出指令执行之后，编译器会立即输入时间戳，因此在实际中，本行为很少看到。

36.2.3 缓冲端口方向改变

试图改变符合缓冲的端口的方向会产生以下行为。

36.3 通道通讯

在一些版本的XS1结构中，从选择语句防护下的数据流通道中输入32位以下的数据是不可能的。

- 编译XS1-G结构时，编译器不允许在XC数据流通道中选择长度短于1个字长的输入语句。命令行选项-fsubword-select放松了这一限制，但将导致即使通道上数据可用，却不会发生具有这类函数的案例。

- 编译XS1-G结构时，inuchar_byref、inct_byref和testct函数不得用于XC选择语句。命令行选项-fsubword-select放松了这一限制，但将导致即使通道上数据可用，却不会发生具有这类函数的案例。

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Zhang Wilson 13-11-26 4:36 PM
已设置格式: 缩进: 首行缩进: 2 字符

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Zhang Wilson 13-11-26 4:36 PM
已设置格式: 缩进: 首行缩进: 2 字符

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

37 XS1数据类型

语言中没有规定C和XC数据类型的尺寸和对正，这可以将整形尺寸设定为目标设备的自然字长，确保计算中的高性能。这同样可以对数据调整设定，使其足够宽以保证有效的内存负荷和存储。图58描绘了xCORE应用二进制界面（见§40）所规定的数据类型尺寸和对正，为C和XC编译对象的链接提供了标准界面。

数据类型	尺寸（比特）	调整（比特）	支持		含义
			XC	C	
Char	8	8	√	√	字符类型
Short	16	16	√	√	短整形
Int	32	32	√	√	初始整形
Long	32	32	√	√	长整形
Long long	64	32	×	√	长长整形
Float	32	32	×	√	32 位 IEEE 浮点
Double	64	32	×	√	64 位 IEEE 浮点
Long double	64	32	×	√	数据指示器
Void *	32	32	√	×	端口
Port	32	32	√	×	计时器
Timer	32	32	√	×	通道端口
chanend	32	32			

图 58 XS1 设备上的数据类型尺寸与对正

另外：

- Char类型默认无符号。
- 可于位字段语句中规定char、short和int数据类型。
- 零宽度位字段会一直填充，直到下一位偏移与位字段说明类型调整。
- 端口概念传输类型为无符号整形（32位）。
- 端口概念计数器类型为无符号短整形（16位）。
- 计时器概念计数器类型为无符号整形（32位）。

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

38 XS1端脚映射

XS1设备中，通过端口将引脚与外部组件相连接，从而组成通向其他设备的链路，并于链路上建立通道。端口分多路传输，可配置引脚用于不同宽度的端口。图59中显示了XS1端脚的映射，解释如下：

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

- 各引脚以XnDpq的格式命名，其中n为设备的有效xCORE数据块，而pq存在于表格中。引脚的物理位置取决于程序包，列在设备数据表中。
- 每条链路以字母A-D标识；链路线通过上标数字0-4来标识。
- 每个端口使用其宽度（第一个数值1、4、8、16或32）和区别同一宽度多个端口的字母（A-P）来标识。这些名称对应于头文件<xs1.h>中的端口指示器（例如端口1A对应于指示器XS1_PORT_1A）。端口各数据位使用上标数字0-31来标识。
- 表格分为6列（或行）。前四行提供了1位、4位和8位数据端口，而后两行则启用单一31位端口。不同封装选项可以输出不同行的数据；16位和32位端口不适用于小设备。

程序使用的端口由一组XC端口说明来确定。例如，端口p = XS1_PORT_1A中数据块[0]：于xCORE数据块0上使用1位端口1A，其中，xCORE数据块0与X0D00引脚相连。

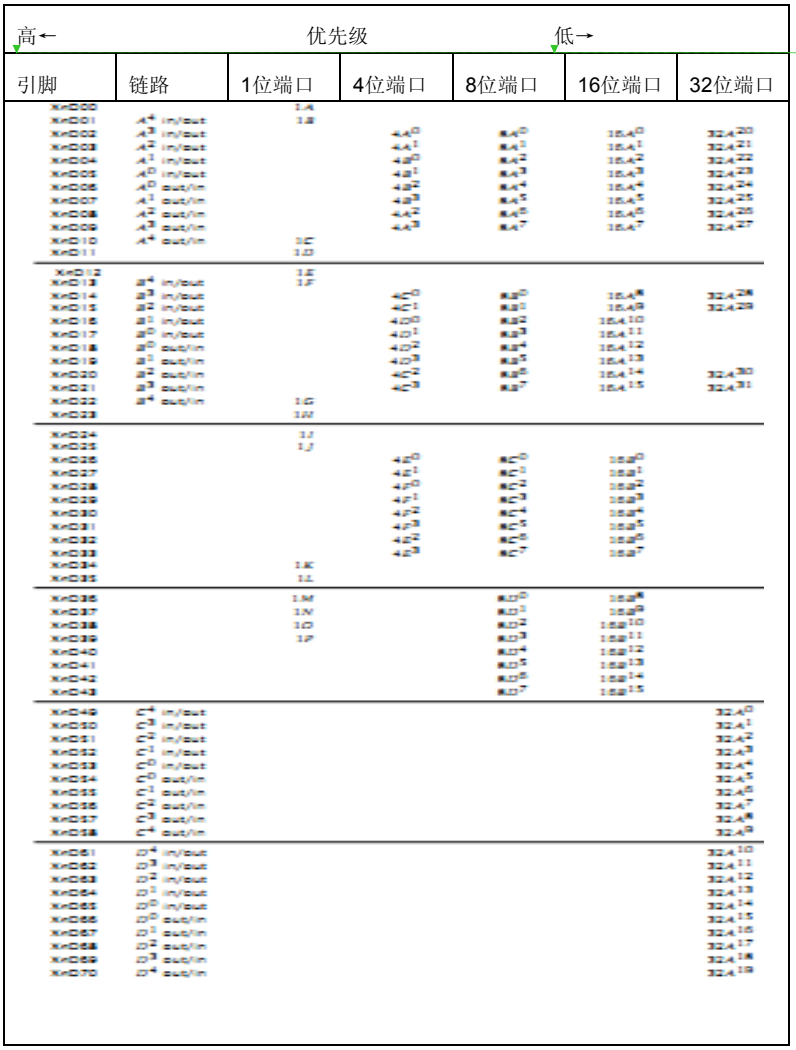
通常设计人员应确保说明端口引脚之间没有重叠，但必须根据需要设定优先级，当较窄端口重叠时，应使用较宽的端口。表格左侧的端口优先于右侧的端口。如果两个说明的端口共用一个引脚，则较窄的端口为先，例如：

```
on tile[2] : out port p1 = XS1_PORT_32A;
on tile[2] : out port p2 = XS1_PORT_8B;
on tile[2] : out port p3 = XS1_PORT_4C;
```

在本案例中：

- 端口p1上的输入/输出使用X2D02至X2D09和X2D49至X2D70。
- 端口p2上的输入/输出使用引脚X2D16至X2D19；自p2输入会于数位0、1、6、7上产生未定义值。
- 端口p3上的输入/输出使用X2D14、X2D15、X2D20和X2D21；自p1输入会于数位28-31上产生未定义值，此时这些数位数据输出时无驱动。

图59：各引脚上可用的端口和链路



Shiqiang Xiao 13-11-13 11:43 AM
已删除: 最

Shiqiang Xiao 13-11-13 11:43 AM
已删除: 最

39 XS1库

本章内容：

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

39.1 数据类型

clock 时钟精度类型。
时钟作为全局变量，使用时钟模块的一个精度标识符启动。运行状态下，时钟向使用该时钟配置的端口提供升降边缘。

39.2 端口配置函数

```
void configure_in_port_handshake(void port p,
in port readyin,
out port readyout, clock clk)
```

在握手模式下配置缓冲端口，作为计时的输入端口。

即入或即出端口不是1位端口则会引发例外情况。端口缓冲未完成时，通过时钟下降边缘来确定即出端口位置。即入、即出端口位置确定后，端口从时钟取样边缘抽取引脚样本。

默认情况下，端口取样边缘为时钟上升边缘。这可以通过set_port_sample_delay()函数来改变。

此函数具有以下参数：

p 需配置的缓冲端口

readyin 用于即入信号的1位端口。

readyout 用于即出信号的1位端口。

clk 用于端口配置的时钟。

```
void configure_out_port_handshake(void port p,
in port readyin,
out port readyout, clock clk,
unsigned initial)
```

配置缓冲端口，作为握手模式下的计时输出端口。

即入或即出端口不是1位端口则会引发例外情况。端口驱动引脚上的初始值，直到输出语句改变驱动值。即入端口可于缓冲端口的取样边缘读取。

自即入端口读取的数值较高时，在时钟下降边缘，如需输出数据，则会将即出端口升高，否则则将即出端口降低。默认情况下，端口取样边为时钟上升边缘。这可以通过函数set_port_sample_delay()来改变。

此函数具有以下参数：

p 需配置的缓冲端口

readyin 用于即入信号的1位端口。

readyout 用于即出信号的1位端口。

clk 用于端口配置的时钟。

initial 于端口输出的初始值。

```
void configure_in_port_strobed_master(void port p,
out port readyout, const clock clk)
```

配置缓冲端口，作为选通主模式下的计时输入端口。

即出端口不是1位端口则会引发例外情况。如端口缓冲未完成时，通过时钟下降边缘来确定即出端口位置。即出端口位置确定后，端口从时钟取样边缘抽取引脚样本。

默认情况下，端口取样边缘为时钟上升边缘。这可以通过set_port_sample_delay()函数来改变。

此函数具有以下参数：

p 需配置的缓冲端口

readyout 用于即出信号的1位端口。

clk 用于端口配置的时钟。

```
void configure_out_port_strobed_master(void port p,
out port readyout,
const clock clk,
unsigned initial)
```

配置缓冲端口，作为选通主模式下的计时输入端口。

即出端口不是1位端口则会引发例外情况。端口驱动引脚上的初始值，直到输出语句改变驱动值。在时钟下降边缘，驱动输出端口。如需输出数据，则会将即出端口升高，否则则将即出端口降低。

此函数具有以下参数：

p 需配置的缓冲端口

readyout 用于即出信号的1位端口。

clk 用于端口配置的时钟。

initial 于端口上输出的初始值。

```
void configure_in_port_strobed_slave(void port p,
in port readyin, clock clk)
```

配置缓冲端口，作为选通主模式下的计时输入端口。

即入端口不是1位端口则会引发例外情况。即入信号较高时，端口从引脚取样边缘取样。默认情况下，端口取样边缘为时钟上升边缘。这可以通过函数set_port_sample_delay()来改变。

此函数具有以下参数：

p 需配置的缓冲端口

readyin 用于即入信号的1位端口。

clk 用于端口配置的时钟。

void configure_out_port_strobed_slave(void port p,
in port readyin, clock clk,
unsigned initial) ·
配置缓冲端口，作为选通主模式下的计时输出端口。

即入端口不是1位端口则会引发例外情况。端口驱动引脚上的初始值，直到输出语句改变驱动值。即入端口可于缓冲端口的取样边缘读取。自即入端口读取的数值较高时，在时钟下降边缘，如需输出数据，则会升高即出端口，否则则降低即出端口。默认情况下，端口取样边为时钟上升边缘。这可以通过函数set_port_sample _delay()来改变。

此函数具有以下参数：

p 需配置的缓冲端口

readyin 用于即入信号的1位端口。

clk 用于端口配置的时钟。

initial 于端口上输出的初始值。

void configure_in_port(void port p, const clock clk)
配置端口，作为不含就绪信号的计时输入端口。

这是端口的默认模式，端口自取样边缘取样，如果端口无缓冲，则可通过输出来改变端口方向。这种改变发生在时钟下一下降边缘。此后，端口作为不含就绪信号的输出端口。

默认情况下，端口取样边为时钟上升边缘。这可以通过函数set_port_sample_delay()来改变。

此函数具有以下参数：

p 需配置的端口，可以为缓冲型也可以为非缓冲型。

clk 用于端口配置的时钟。

void configure_in_port_no_ready(void port p, const clock clk)
为configure_in_port()的别名。

void configure_out_port(void port p, const clock clk, unsigned initial)
配置端口，作为不含就绪信号的计时输入端口。

Shiqiang Xiao 13-11-13 11:45 AM
已删除：

端口驱动引脚上的初始值，直到输入或输出语句改变驱动值。于时钟下一下降边缘驱动输出，每个时钟周期需保持一个端口宽度的数据位。如果未缓冲端口，则可通过执行输入来改变端口方向。这种改变发生在输出暂停一个时钟周期之后时钟的下降边缘。此后，端口作为不含就绪信号的输入端口。

此函数具有以下参数：

p 需配置的端口，可以为缓冲型也可为非缓冲型。

clk 用于端口配置的时钟。

initial 于端口上输出的初始值。

void configure_out_port_no_ready(void port p, const clock clk, unsigned initial)

configure_out_port()的别名。

void configure_port_clock_output(void port p, const clock clk)

配置1位端口以输出时钟信号。

端口不是1位端口则会引发例外情况。在该模式下于端口上执行输入或输出时会产生下列未定义行为。

此函数具有以下参数：

p 需配置的1位端口。

clk 用于输出的时钟。

void set_port_no_sample_delay(void port p)

将端口设定为无抽样延迟模式。

使得端口于时钟上升边缘对输入数据取样，这是端口的默认状态。

此函数具有以下参数：

p 需配置的端口。

void set_port_sample_delay(void port p)

将端口设定为取样延迟模式。

这使端口在时钟下降边缘对输入数据进行抽样。

此函数具有以下参数：

p 需配置的端口。

void set_port_clock(void port p, const clock clk)

在端口附加时钟。

这相当于于端口上使用SETCLK指令。采用时钟边缘进行数据取样和输出。通常优先使用配置_*_port_* 函数，因其可确保所需模式要求的端口配置按正确顺序变化。

此函数具有以下参数：

p 需配置的端口。

clk 需附加的时钟。

void set_port_ready_src(void port p, void port ready)

设定1位端口作为另一即出端口。

这相当于于端口上使用SETRDY指令。即出端口不是1位端口时则会引发例外情况。在该模式下于端口上执行输入或输出时会产生下列未定义行为。即出端口用于表示端口准备就绪可用于传输数据。通常优先使用配置 _*_port_* 函数，因其可确保所需模式要求的端口配置按正确顺序变化。

此函数具有以下参数：

p 需配置的端口。

ready 1位端口就绪，可用于传输即出信号。

void set_port_use_on(void port p)

开启端口。

端口状态初始化到此类型端口的默认状态。如果此端口已开启，则将其状态复位到默认态。

此函数具有以下参数：

p 需打开的端口。

void set_port_use_off(void port p)

关闭端口。

端口关闭后不执行任何行动。使用关闭后的端口会引发例外情况。

此函数具有以下参数：

p 需关闭的端口。

void set_port_mode_data(void port p)

配置端口作为数据端口。

这是端口的默认状态；在端口上执行输出操作，以控制输出信号。

此函数具有以下参数：

p 需配置的端口。

void set_port_mode_clock(void port p)

配置1位端口作为时钟输出端口。

端口将输出与端口相连的时钟。端口不是1位端口则会引发例外情况。可使用set_port_mode_data() 函数，将端口复位至默认态。

此函数具有以下参数：

p 需配置的端口。

void set_port_mode_ready(void port p)

配置1位端口用作信号即出端口。

本端口将输出与set_port_ready_src()连接的即出信号。端口不是1位端口则会引发例外情况。可使用set_port_mode_data()函数将端口复位至默认状态。通常优先使用配置_*_port_* 函数，因其可确保所需模式要求的端口配置按正确顺序变化。

此函数具有以下参数：

p 需配置的端口。

void set_port_drive(void port p)

在驱动模式下配置端口。

输出至端口的数值通过引脚来驱动，这是端口的默认驱动态。调用set_port_drive()函数，可禁用端口上拉或下拉电阻。

此函数具有以下参数：

p 需配置的端口。

void set_port_drive_low(void port p)

在低驱动模式下配置端口。

1位端口输出0时，会将引脚驱动到较低的位置；输出为1时，不会驱动任何值。如果端口不为1位端口，则输出至端口的结果未界定。XS1-G设备调用set_port_drive_low()时，会同时启用端口内部上拉电阻。XS1-L设备调用set_port_drive_low()时，会同时启用端口内部下拉电阻。

此函数具有以下参数：

p 需配置的端口。

void set_port_pull_up(void port p)

启用端口内部上拉电阻。

不驱动引脚时，上拉电阻确保自端口的取样值为1，然而上拉电阻不足以保证确定的外部值。XS1-G设备调用set_port_pull_up()时，会同时在低驱动模式下驱动端口。XS1-L设备上无上拉电阻时调用set_port_pull_up()函数，则可能引发例外情况。

此函数具有以下参数：

p 需配置的端口。

void set_port_pull_down(void port p)

启用端口内部下拉电阻。

不驱动引脚时，下拉电阻确保端口取样值为0，然而下拉电阻不足以保证确定的外部值。XS1-G设备上无下拉电阻时调用set_port_pull_down()函数，则可能引发例外情况。XS1-L设备上调用set_port_pull_down()函数，则会在低驱动模式下配置端口。

此函数具有以下参数：

p 需配置的端口。

void set_port_pull_none(void port p)

禁用端口的上拉或下拉电阻。

这可以在驱动模式下配置端口。

此函数具有以下参数：

p 需配置的端口。

void set_port_master(void port p)

将端口设定为主模式。

这相当于于值为XS1_SETC_MS_MASTER的端口上使用SETC指令。通常优先使用configure_in_port_strobed_master()和configure_out_port_strobed_master()函数，因其可确保所需模式要求的端口配置按正确顺序变化。

此函数具有以下参数：

p 需配置的端口。

void set_port_slave(void port p)

将端口设定为从属模式。

这相当于于值为XS1_SETC_MS_SLAVE的端口上使用SETC指令。通常优先使用configure_in_port_strobed_slave() 和configure_out_port_strobed_slave()函数，因其可确保所需模式要求的端口配置按正确顺序变化。

此函数具有以下参数：

p 需配置的端口。

void set_port_no_ready(void port p)

配置端口不可使用就绪信号。

这相当于于值为XS1_SETC_RDY_NOREADY的端口上使用SETC指令。通常优先使用configure_in_port()和configure_out_port()函数，因其可确保所需模式要求的端口配置按正确顺序变化。
此函数具有以下参数：

p 需配置的端口。

void set_port_strobed(void port p)

将端口设定为选通模式。

这相当于于值为XS1_SETC_RDY_STROBED的端口上使用SETC指令。通常优先使用_*_port_strobed_*函数，因其可确保所需模式要求的端口配置按正确顺序变化。
此函数具有以下参数：

p 需配置的端口。

void set_port_handshake(void port p)

将端口设定为握手模式。

这相当于于值为XS1_SETC_RDY_HANDSHAKE的端口上使用SETC指令。通常优先使用_*_port_handshake函数，因其可确保所需模式要求的端口配置按正确顺序变化。

此函数具有以下参数：

p 需配置的端口。

void set_port_no_inv(void port p)

配置端口，不得转换已经取样且于引脚上驱动的数据。

这是端口的默认状态。

此函数具有以下参数：

p 需配置的端口。

void set_port_inv(void port p)

配置端口，不得转换已经取样且于引脚上驱动的数据。
端口不是1位端口则会引发例外情况。如果端口作为时钟源，则设定该模式会交换时钟的上升脉冲和下降边缘。

此函数具有以下参数：

p 需配置的1位端口。

void set_port_shift_count(void port p, unsigned n)

设置端口的移位计数。

这与SETPSC指令相对应，新的移位计数必须小于端口的移位宽度，大于零且大于端口宽度的数倍，否则将引发例外情况。一定量的数据移位时，在端口处输入该函数将使下一输入就绪。捕获数据有效位最多时，下一输入将会返回传输宽度位的数据。在端口处输出该函数将导致下一输出移出相应数位。通常优先使用partin()和partout()函数，而非setpsc()函数，因前两种函数可同时执行所需配置及输入或输出。

此函数具有以下参数：

p 需配置的缓冲端口。

n 新移位计数。

void set_pad_delay(void port p, unsigned n)

在与端口相连的引脚上设定延迟。

该端口引脚上取样的输入信号，于端口上发现之前，会产生一个等同于处理器时钟周期的延迟，引脚上默认延迟为0，而延迟必须在0-5之内设定。如果多个启用端口连接至同一引脚，则通过优先级最高的端口设定该引脚的延迟。

此函数具有以下参数：

p 需配置的端口。

n 输入信号延迟的处理器时钟周期数。

39.3 时钟配置函数

void configure_clock_src(clock clk, void port p)

配置时钟将1位端口作为时钟源。

这使得端口上的输入、输出操作与外部时钟信号同步。端口不是1位端口则会引发例外情况。

此函数具有以下参数：

clk 需配置的时钟。

p 用作时钟源的1位端口。

void configure_clock_ref(clock clk, unsigned char divide)

配置时钟以基准时钟作为时钟源。

如果时钟分频设定为0，则使用基准时钟频率，否则，将基准时钟频率除以2作为分频使用。默认情况下，基准时钟频率配置为100MHz。

此函数具有以下参数：

clk 需配置的时钟。

divide 时钟分频

void configure_clock_rate(clock clk, unsigned a, unsigned b)

配置时钟以(a/b) MHz的频率运行。

如果硬件不支持规定频率，则会引发例外情况，硬件支持频率MHz和频率形式（MHz为基准时钟频率，数值为1-255之间）。

此函数具有以下参数：

clk 需配置的时钟。

a 所需频率的被除数。

b 所需频率的除数。

void configure_clock_rate_at_least(clock clk, unsigned a, unsigned b)

配置时钟以硬件支持的最低频率即大于等于(a/b) MHz运行。

频率不符合该要求时会引发例外情况。此函数具有以下参数：

clk 需配置的时钟。

a 所需频率的被除数。

b 所需频率的除数。

void configure_clock_rate_at_most(clock clk, unsigned a, unsigned b)

配置时钟以硬件支持的最快非零频率即大于等于(a/b) MHz运行。

频率不符合该要求时会引发例外情况。此函数具有以下参数：

clk 需配置的时钟。

a 所需频率的被除数。

b 所需频率的除数。

void set_clock_src(clock clk, void port p)

于1位端口上配置时钟源。

这与时钟上使用SETCLK指令相对应。端口不是1位端口时则会引发例外情况，同时时钟先前设定过非零分频时，也会引发例外情况。通常使用通常优先使用configure_clock_src()函数，因其不会产生这类问题。

此函数具有以下参数：

clk 需配置的时钟。

p 作为时钟源的1位端口。

void set_clock_ref(clock clk)

为基准时钟设计时钟源。

这与时钟上使用SETCLK指令相对应，时钟分频保持不变。

此函数具有以下参数：

clk 需配置的时钟。

void set_clock_div(clock clk, unsigned char div)

设计时钟分频。

这与时钟上使用SETD指令相对应。必须为基准时钟设计时钟源，否则会引发例外情况。如果时钟分频设定为0，则源频率保持不变，否则将原频率除以分频的2倍。

此函数具有以下参数：

clk 需配置的时钟。

div 使用的分频。

void set_clock_rise_delay(clock clk, unsigned n)

为时钟上升边缘设定延迟。

与时钟相连的端口上发现时钟上升边缘之前，会产生一个等同于处理器时钟周期的延迟。默认上升边缘延迟为0，而延迟必须在0-512之内设定。如果时钟边缘会延迟多个时钟周期，则与时钟相连的端口上无法看到时钟上升边缘。

此函数具有以下参数：

clk 需配置的时钟。

n 时钟上升边缘延迟的处理器时钟周期数。

void set_clock_fall_delay(clock clk, unsigned n)

设计时钟下降边缘的延迟。

在与时钟相连的端口上看见时钟下降边缘之前，会产生一个等同于处理器时钟周期的延迟。默认下降边缘延迟为0，而延迟必须在0-512之内设定。如果时钟边缘会延迟多个时钟周期，则与时钟相连的端口上无法看到时钟下降边缘。

此函数具有以下参数：

clk 需配置的时钟。

n 时钟下降边缘延迟的处理器时钟周期数。

void set_clock_ready_src(clock clk, void port ready)

设置时钟将1位端口用于即入信号。

这相当于于时钟上使用SETRDY指令。端口不是1位端口时则会引发例外情况。在该模式下于端口上执行输入或输出时会产生下列未定义行为。即出端口用于表示端口准备就绪可用于传输数据。通常优先使用配置_*_port_* 函数，因其可确保所需模式要求的端口配置按正确顺序变化。

此函数具有以下参数：

clk 需配置的时钟。

ready 用于即入信号的1位端口。

void set_clock_on(clock clk)

打开时钟。

时钟状态初始化到此类型时钟的初始状态。如果此时钟开启，则其状态复位到默认态。

此函数具有以下参数：

clk 需打开的时钟。

void set_clock_off(clock clk)

关闭时钟。

时钟关闭后不执行行动。使用关闭后的端口会引发例外情况。

此函数具有以下参数：

clk 需关闭的时钟。

39.4 端口处理函数

void start_port(void port p)

启动端口。

清除端口所用的缓冲。

此函数具有以下参数：

p 需启动的端口。

void stop_port(void port p)

停用端口。

端口复位至非就绪端口。

此函数具有以下参数：

p 需停用的端口。

unsigned peek(void port p)

指示端口于引脚上对现值进行取样。

端口立即向处理器提供取样的端口宽度位数据，而不管传输宽度、时钟、就绪信号和缓冲。输入不会对端口上后续输入、输出产生影响。

此函数具有以下参数：

p 达到峰值的端口。

函数返回：

引脚上的取样值。

void clearbuf(void port p)

清除端口使用的缓冲。

未经处理器输入的数据，不得对其取样。由处理器输出但未经端口驱动的数据，不得对其取样。连续输出过程中的端口必须立即终止。如果暂停输出可能改变端口方向，则方向不会改变。如果调用clearbuf()函数于引脚上驱动数值，则会持续驱动该值直到输出语句改变驱动数值。

此函数具有以下参数：

p 需清除缓冲的端口。

void sync(void port p)

等待直到端口完成所有暂停输出。

等待直到端口暂停输出全部完成，引脚上最后一个端口宽度的数据保持一个时钟周期。

此函数具有以下参数：

p 等候中的端口。

unsigned endin(void port p)

于缓冲端口上终止当前输入。

返回经端口取样但未经处理器输入的位数。该计数包括反序列化所用的传输寄存器和移位寄存器中的数据。端口后续输入返回传输宽度位的数据，直到所剩数据位小于一个传输宽度。只需进一步输入就可以读取剩余数据，当捕获数据有效位最多时，该输入将会返回传输宽度位的数据。

此函数具有以下参数：

p 终止当前输入的端口。

此函数返回：

剩余数据位数。

unsigned partin(void port p, unsigned n)

于缓冲端口上执行规定宽度的数据输入。

宽度必须小于端口的传输宽度，大于零且大于端口宽度的数倍，否则将引发例外情况。端口移位寄存器中位数大于或等于规定宽度值时，返回值未定。

此函数具有以下参数：

p 用于输入的缓冲端口。

n 待输入的数据位数。

此函数返回：

输入值。

void partout(void port p, unsigned n, unsigned val)

于缓冲端口上执行规定宽度的数据输出。

宽度必须小于端口的传输宽度，大于零且大于端口宽度的数倍，否则将引发例外情况。输出最低n有效位的数值。

此函数具有以下参数：

p 用于输出的缓冲端口。
n 待输出的数据位数。
val 待输出的数值。

unsigned partout_timed(void port p, unsigned n, unsigned val, unsigned t)

端口计数等于规计时间时，于缓冲端口上执行规定宽度的数据输出。

宽度必须小于端口的传输宽度，大于零且大于端口宽度的数倍，否则将引发例外情况。输出最低n有效位的数值。

此函数具有以下参数：

p 用于输出的缓冲端口。

n 待输出的数据位数。

val 待输出的数值。

t 待输出的端口计数值。

{unsigned /* value */, unsigned /* timestamp */} p partin_timestamped(void port p, unsigned n)

于缓冲端口上执行规定宽度的输入，并添加输入时间戳。

宽度必须小于端口的传输宽度，大于零且大于端口宽度的数倍，否则将引发例外情况。端口移位寄存器中位数大于或等于规定宽度值时，返回值未定。

此函数具有以下参数：

p 用于输入的缓冲端口。

n 待输入的数据位数。

函数返回：

输入值及时间戳。

unsigned partout_timestamped(void port p, unsigned n, unsigned val)

于缓冲端口上执行规定宽度的数据输出，并添加输出时间戳。

宽度必须小于端口的传输宽度，大于零且大于端口宽度的数倍，否则将引发例外情况。输出最低n有效位的数值。

此函数具有以下参数：

p 用于输出的缓冲端口。

n 待输出的数据位数。

val 待输出的数值。

此函数返回：

输出时间戳。

39.5 时钟处理函数

void start_clock(clock clk)

时钟处于运行状态。

时钟只有处于运行状态才会产生边缘。所有与时钟相连的端口计数器都复位为0。

此函数具有以下参数：

clk 处于运行状态的时钟。

void stop_clock(clock clk)

等待直到时钟较低，而后将其置于停滞状态。

在停滞状态时钟不会产生边缘。

此函数具有以下参数：

clk 将置于停滞状态的时钟。

39.6 逻辑核心/数据块控制函数

void set_core_fast_mode_on(void)

将当前逻辑核心设定至快速运行模式。

排程器为快速运行模式下的逻辑核心预留一个槽，而不管核心是等待输入还是等待完成选择。这就降低了暂停输入或选择完成时状态变化所带来的最坏情况延迟。然而，将核心置于最快模式意味着核心处于等待状态时，其他逻辑核心都不能使用附加槽。除此之外，将逻辑核心设定为快速运行模式还可能增加耗电量。

void set_core_fast_mode_off(void)

将当前逻辑核心设定至正常执行模式。

如果先前将核心设定为快速模式，则使用set_core_fast_mode_on()函数将执行模式恢复至默认状态。

unsigned getps(unsigned reg)

获取处理器状态寄存器值。

这与GETPS指令相对应。如果参数不是合法处理器的状态寄存器，则会引发例外情况。

此函数具有以下参数：

reg 需读取的处理器状态寄存器。

此函数返回：

处理器状态寄存器值。

void setps(unsigned reg, unsigned value)

设定处理器状态寄存器值。

这与GETPS指令相对应。如果不是合法处理器状态寄存器，则会引发例外情况。

此函数具有以下参数：

reg 需写入的处理器状态寄存器。

value 处理器状态寄存器设定值。

int read_pswitch_reg(unsigned tileid, unsigned reg, unsigned &data)

读取处理器开关寄存器值。

该读数为规定 编号的处理器开关读数。成功时会返回1，此时为数据赋予寄存器值。收到故障应答，或寄存器值或数据块标识符太大而与读取包不匹配时，返回0.

此函数具有以下参数：

tileid 数据块标识符。

reg 寄存器号。

data 自寄存器读取数值。

此函数返回：

读数是否成功。

int write_pswitch_reg(unsigned tileid, unsigned reg, unsigned data)

向处理器开关寄存器写入数值。

该写入为规定 编号的处理器开关写入值。成功时会返回1，此时向数据分配寄存器值。收到故障应答，或寄存器值或数据块标识符太大而与写入包不匹配时，返回0。

此函数具有以下参数：

tileid 数据块标识符。

reg 寄存器号。

data 写入寄存器的数值。

此函数返回：

写入是否成功。

int write_pswitch_reg_no_ack(unsigned tileid, unsigned reg, unsigned data)

向处理器开关寄存器写入值而无需确认。

该写入为规定 编号的处理器开关读数。与write_pswitch_reg()不同，该函数无需一直等候写入执行。寄存器值或数据块标识符太大而与写入不匹配时，返回0，否则返回1。未对故障确认作出规定，因此返回值反映不出写入是否成功。

此函数具有以下参数：

tileid 数据块标识符。

reg 寄存器号。

data 写入寄存器的值。

此函数返回：

参数是否有效。

int read_sswitch_reg(unsigned tileid, unsigned reg, unsigned &data)

读取系统开关寄存器值。

该读数为规定 编号的处理器开关读数。成功时会返回1，此时为数据赋予寄存器值。收到故障应答，或寄存器值或数据块标识符太大而与读取包不匹配时，返回0。

此函数具有以下参数：

tileid 数据块标识符。

reg 寄存器号。

data 写入寄存器的值。

此函数返回：

读数是否成功。

int write_sswitch_reg(unsigned tileid, unsigned reg, unsigned data)

向系统开关寄存器写入值。

该写入是向规定编号的系统开关写入。收到成功应答返回1，收到故障应答，或寄存器值或数据块标识符太大而与写入包不匹配时，返回0。

此函数具有以下参数：

tileid 数据块标识符。

reg 寄存器号。

data 写入寄存器的值。

此函数返回：

写入是否成功。

int write_sswitch_reg_no_ack(unsigned tileid, unsigned reg, unsigned data)

向系统开关寄存器写入值，无需确认。

该写入是向规定编号的系统开关写入。收到成功应答返回1，收到故障应答，或寄存器值或数据块标识符太大而与写入包不匹配时，返回0，否则返回1。由于无需确认，返回值不能反映是否成功。

此函数具有以下参数：

tileid 数据块标识符。

reg 寄存器号。

data 写入寄存器的值。

此函数返回：

参数是否有效。

int read_node_config_reg(tileref tile, unsigned reg, unsigned &data)

读取节点配置寄存器值。

读取含规定数据块的节点值，读取成功时会返回1且为数据赋予寄存器值。如果收到故障确认或寄存器值太大而与读取包不匹配时，返回0。

此函数具有以下参数：

tile 数据块。

reg 寄存器号码。

data 自寄存器读取的数值。

此函数返回：

读取是否成功。

int write_node_config_reg(tileref tile, unsigned reg, unsigned data)

向节点配置寄存器写入值。

写入含规定数据块的节点值，读取成功时会返回1。如果收到故障确认或寄存器值太大而与写入包不匹配时，返回0。

此函数具有以下参数：

tile 数据块。

reg 寄存器号码。

data 向寄存器写入值。

此函数返回：

读取是否成功。

int write_node_config_reg_no_ack(tileref tile, unsigned reg, unsigned data)

向节点配置寄存器写入值，无需确认。

写入含规定数据块的节点值，与write_node_config_reg() 不同，此函数在写入执行之前无需等待。寄存器值太大而与写入包不匹配时返回0，否则返回1。由于不需确认，因此返回值不能反映写入是否成功。

此函数具有以下参数：

tile 数据块。

reg 寄存器号码。

data 向寄存器写入值。

此函数返回：

参数是否有效。

int read_periph_8(tileref tile,
unsigned peripheral,
unsigned base_address,
unsigned size,
unsigned char data[])

从以指定基地址为起点的外围处读取字节。

外围必须为带8位界面的外围，成功时返回1，以读取到的数值填充数据，失败时返回0。
此函数具有以下参数：

tile 数据块。
peripheral 外围号码。
base_address 基地址。
size 需读取的8位数值。
data 自外围读取的数值。
此函数返回：
读取是否成功。

int write_periph_8(tileref tile,
unsigned peripheral,
unsigned base_address,
unsigned size,
const unsigned char data[])

向以指定基地址为起点的外围处写入字节。

外围必须为带8位界面的外围，成功时返回1，失败时返回0。

此函数具有以下参数：

tile 数据块。
peripheral 外围号码。
base_address 基地址。
size 需写入的8位数值。
data 向外围写入的数值。
此函数返回：
写入是否成功

int write_periph_8_no_ack(tileref tile,

unsigned peripheral,
unsigned base_address, unsigned size,
const unsigned char data[])

向以指定基地址为起点的外围处写入字节，无需确认。

外围必须为带8位界面的外围，与write_periph_8()不同，此函数与写入执行之前无需等待。由于不需确认，因此返回值不能反映写入是否成功。

此函数具有以下参数：

tile 数据块。
peripheral 外围号码。
base_address 基地址。
size 需写入的8位数值。
data 向外围写入的数值。
此函数返回：
参数是否有效。

int read_periph_32(tileref tile,
unsigned peripheral,
unsigned base_address,
unsigned size,
unsigned data[])

自以指定基地址为起点的外围处读取32位字节。

外围必须为带8位界面的外围，成功时返回1，以读取到的数值填充数据，失败时返回0。读取带8位界面的外围时，各字有效字节为最小地址的字节（大端字节顺序）。

此函数具有以下参数：
tile 数据块。
peripheral 外围号码。
base_address 基地址。

size 需读取的32位字符数。
data 自外围读取的数值。
此函数返回：
读取是否成功。

int write_periph_32(tileref tile,
unsigned peripheral,
unsigned base_address,
unsigned size,
const unsigned data[])

向以指定基地址为起点的外围处写入32位字节。

成功时返回1，失败时返回0。向带8位界面的外围写入数据时，将传递给该函数的最有效字节写入最小地址的字节（大端字节顺序）。

此函数具有以下参数：
tile 数据块。
peripheral 外围号码。
base_address 基地址。
size 需写入的32位字符数。
data 向外围写入的数值。
此函数返回：
写入是否成功。

int write_periph_32_no_ack(tileref tile,
unsigned peripheral,
unsigned base_address, unsigned size,
const unsigned data[])

向以指定基地址为起点的外围处写入32位字节，无需确认。

与write_periph_32()不同，此函数与写入执行之前无需等待。由于不需确认，因此返回值不能反映写入是否成功。向带8位界面的外围写入数据时，将传递给该函数的最有效字节写入最小地址的字节（大端字节顺序）。

此函数具有以下参数：
tile 数据块。
peripheral 外围号码。
base_address 基地址。

size 需写入的32位字符数。

data 向外围写入的数值。
此函数返回：

参数是否有效。

unsigned get_local_tile_id(void)
返回调用器所运行的数据块标识符。
标识符唯一鉴别网络中的数据块。
此函数返回：
数据块标识符。

unsigned get_logical_core_id(void)
返回调用器所运行的逻辑核心标识符。
标识符唯一鉴别当前数据块上的逻辑核心。
此函数返回：
逻辑核心标识符。

39.7 通道函数

void start_streaming_master(chanend c)
启动通道上的流式通讯。

调用该函数时必须同时调用通道另一端的start_streaming_slave()函数。开通两个通道端点的路径，以使用流式输入、输出函数进行非同步通讯。使用stop_streaming_master()函数关闭之前，本路径一直保持开启状态。注意，如果网络中两点之间开启的通道数等于这两点之间所有可能的路径数，则这两点之前不存在其他通道通讯，这将使程序陷入死循环。

此函数具有以下参数：
c 数据流开始的通道端

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

void stop_streaming_master(chanend c)

终止通道中的流式通讯。

调用该函数时必须同时调用通道另一端的stop_streaming_slave()函数。关闭先前通过start_streaming_master()函数开启的路径，使路径可用于其他通道通讯。

此函数具有以下参数：

c 数据流终止的通道端

void start_streaming_slave(chanend c)

启动通道中的流式通讯。

调用该函数时必须同时调用通道另一端的start_streaming_master()函数。开通两个通道端点的路径，以使用流式输入、输出函数进行非同步通讯。使用stop_streaming_slave()函数关闭之前，本路径一直保持开启状态。注意，如果网络中两点之间开启的通道数等于这两点之间所有可能的路径数，则这两点之前不存在其他通道通讯，这将使程序陷入死循环。选择语句下可调用start_streaming_slave()函数，使通道另一端调用start_streaming_master()函数后，此函数处于就绪状态。

此函数具有以下参数：

c 数据流开始的通道端。

void stop_streaming_slave(chanend c)

终止通道中的流式通讯。

调用该函数时必须同时调用通道另一端的stop_streaming_master()函数。关闭先前通过start_streaming_slave()函数开启的路径，使路径可用于其他通道通讯。

此函数具有以下参数：

c 数据流终止的通道端

void outuchar(chanend c, unsigned char val)

流式输出一个值作为通道端的无符号字符。

所使用的协议与输入(:>)和输出(<:)操作符使用的协议不兼容。

此函数具有以下参数：

c 流式输出数据的目的通道端。

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

val 输出的数值。

void outuint(chanend c, unsigned val)

流式输出一个值作为通道端的无符号字符。

所使用的协议与输入(:>)和输出(<:)操作符使用的协议不兼容。

此函数具有以下参数：

c 流式输出数据的目的通道端。

val 输出的数值。

unsigned char inuchar(chanend c)

自通道端流式输入无符号字符。

如果通道中的下一符号为控制符，则会引发例外情况。所使用的协议与输入(:>)和输出(<:)操作符使用的协议不兼容。

此函数具有以下参数：

c 流式输入数据的目的通道端。

此函数返回：

收到的值。

unsigned inuint(chanend c)

自通道端流式输入无符号字符。

如果通道中的下一字节包含控制符，则会引发例外情况。所使用的协议与输入(:>)和输出(<:)操作符使用的协议不兼容。

此函数具有以下参数：

c 流式输入数据的目的通道端。

此函数返回：

收到的值。

void inuchar_byref(chanend c, unsigned char &val)

自通道端流式输入无符号字符。

向val写入输入值。如果通道中的下一符号为控制符，则会引发例外情况。所使用的协议与输入(:>)和输出(<:)操作符使用的协议不兼容。

此函数具有以下参数：

c 流式输入数据的目的通道端。

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

val 设定接收值的变量。

void inuint_byref(chanend c, unsigned &val)

自通道端流式输入无符号整形。

向val写入输入值。选择语句下可调用start_streaming_slave()函数，在这种语句只要数据可用于通道时，则此函数就处于就绪状态。所使用的协议与输入(:>)和输出(<:)操作符使用的协议不兼容。

此函数具有以下参数：

c 流式输入数据的目的通道端。

val 设定接收值的变量。

void outct(chanend c, unsigned char val)

于通道端流式输出控制符。

试图输出硬件控制符将引发意外情况。

此函数具有以下参数：

c 流式输出数据的目的通道端。

val 将输出的控制符值。

void chkct(chanend c, unsigned char val)

检查规定值的控制符。

通道中下一字节为控制符，且与期望值相符，则输入该控制符而后删除，否则将引发例外情况。

此函数具有以下参数：

c通道端。

val 期望控制符值。

unsigned char inct(chanend c)

于通道端流式输入控制符。

通道中下一字节不为控制符时，则会引发例外情况，否则返回控制符值。

此函数具有以下参数：

c 流式输入数据的通道端。

此函数返回

接收到的控制符。

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

void inct_byref(chanend c, unsigned char &val)

于通道端流式输入控制符。

向val写入输入值。通道中下一字节不为控制符时，则会引发例外情况

此函数具有以下参数：

c 流式输入数据的通道端。

val 设定接收值的变量。

int testct(chanend c)

测试通道端下一字节是否为控制符。

不从通道中删除控制符，因此控制符仍可用于输入。

此函数具有以下参数：

c 执行测试的通道端。

此函数返回：

下一字节为控制符返回1，否则返回0。

int testwct(chanend c)

测试通道端下一字节是否包含控制符。

字符中包含控制符，则返回字符中控制符的位置。不从通道中删除任何数据。

此函数具有以下参数：

c 执行测试的通道端。

此函数返回：

字符中首个控制符的位置（1-4）；字符中不包含控制符时返回0。

void soutct(streaming chanend c, unsigned char val)

于流式通道端输出控制符。

试图输出硬件控制符将会引发例外情况。试图输出控制符CT_END或 CT_PAUSE是无效的行为。

此函数具有以下参数：

c 流式输出数据的目的通道端。

val 输出的控制符值。

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

void schkct(streaming chanend c, unsigned char val)

于流式通道端检查给定值的控制符。

通道中下一字节为控制符，且与期望值相符，则输入该控制符而后删除，否则将引发例外情况。

此函数具有以下参数：

c 流式通道端。

val 期望控制符值。

unsigned char sinct(streaming chanend c)

于流式通道端输入控制符。

通道中下一字节不为控制符时，则会引发例外情况，否则返回控制符值。

此函数具有以下参数：

c 流式输入数据的通道端。

此函数返回：

收到的控制符。

void sinct_byref(streaming chanend c, unsigned char &val)

于流式通道端输入控制符。

向val写入输入值。如果通道中的下一符号不为控制符，则会引发例外情况。

此函数具有以下参数：

c 流式输入数据的流式通道端。

val 设定接收值的变量。

int stestct(streaming chanend c)

测试通道端下一字节是否为控制符。

不从通道中删除控制符，因此控制符仍可用于输入。

此函数具有以下参数：

c 执行测试的通道端。

此函数返回：

下一字节为控制符返回1，否则返回0.

int stestwct(streaming chanend c)

测试通道端下一字节是否包含控制符。

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Zhang Wilson 13-11-26 4:37 PM
已设置格式: 缩进: 首行缩进: 7 字符

Zhang Wilson 13-11-26 4:37 PM
已设置格式: 缩进: 首行缩进: 7 字符

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Zhang Wilson 13-11-26 4:37 PM
已设置格式: 缩进: 首行缩进: 7 字符

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Zhang Wilson 13-11-26 4:37 PM
已设置格式: 缩进: 首行缩进: 7 字符

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Zhang Wilson 13-11-26 4:37 PM
已设置格式: 缩进: 首行缩进: 7 字符

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

字符中包含控制符，则返回字符中控制符的位置。不从通道中删除任何数据。

此函数具有以下参数：

c 执行测试的流式通道端。

此函数返回：

字符中首个控制符的位置（1-4）；字符中不包含控制符时返回0。

transaction out_char_array(chanend c, const char src[], unsigned size)

通过通道输出数据块。

于通道端输出数据的全部字节。调用out_char_array()函数时必须同时调用通道另一端的in_char_array()函数。输出字节数必须与输入字节数匹配。

此函数具有以下参数：

c 用于输出的通道端。

src 发送的值数组。

size 输出的字节数。

transaction in_char_array(chanend c, char src[], unsigned size)

自通道输入数据块。

于通道端输入数据的全部字节并存入数组中。调用in_char_array()函数时必须同时调用通道另一端的out_char_array()函数。输入字节数必须与输出字节数匹配。

此函数具有以下参数：

c 用于输入的通道端。

src 存放自通道输入数值的数组。

size 输入字节数。

39.8 判定函数

void pinseq(unsigned val)

等待直到端口引脚数值与规定值相等。

必须调用该函数作为端口上输入的表达式。端口引脚上的数值与val端口宽度的有效位数相等时，输入就绪。

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Zhang Wilson 13-11-26 4:37 PM
已设置格式: 缩进: 首行缩进: 7 字符

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Zhang Wilson 13-11-26 4:37 PM
已设置格式: 缩进: 首行缩进: 7 字符

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

此函数具有以下参数：

val 相比对的数值。

void pinsneq(unsigned val)

等待直到端口引脚数值与规定值不相等。

必须调用该函数作为端口上输入的表达式。端口引脚上的数值与val端口宽度的有效位数相等时，输入就绪。

此函数具有以下参数：

val 相比对的数值。

void pinseq_at(unsigned val, unsigned time)

等待直到端口引脚数值与规定值相等，且端口计数与规计时间相等。

必须调用该函数作为非缓冲端口上输入的表达式。端口引脚上的数值与val端口宽度的有效位数相等且端口计数器与规计时间相等时，输入就绪。

此函数具有以下参数：

val 相比对的数值。

time 比对的时间

void pinsneq_at(unsigned val, unsigned time)

等待直到端口引脚数值与规定值不相等，而端口计数器与规计时间相等。

必须调用该函数作为非缓冲端口上输入的表达式。端口引脚上的数值与val端口宽度的有效位数相等且端口计数器与规计时间时，输入就绪。

此函数具有以下参数：

val 相比对的数值。

time 比对的时间

void timerafter(unsigned val)

等待直到计时器时间与规定值相等。

必须调用该函数作为计时器上输入的表达式。计时器计数通过晚于规定值即晚于给定值来解释后，输入就绪。表达式为真时，时间A即视为晚于时间B。

此函数具有以下参数：

val 相比对的时间。

39.9 XS1-S函数

此函数用于控制XS1-S设备上的模拟数字转换器。

void enable_xs1_su_adc_input(unsigned number, chanend c)

启用由数字规定的模拟数字转换器输入。

将样本送至通道端 C。

此函数具有以下参数：

number 模拟数字转换器输入号码。

c 与XS1-SU模拟数字转换器相连的通道。

void enable_xs1_su_adc_input_streaming(unsigned number, streaming chanend c)

启用由数字规定的模拟数字转换器输入。

将样本送至通道端 C。

此函数具有以下参数：

number 模拟数字转换器输入号码。

c 与XS1-SU模拟数字转换器相连的通道。

void disable_xs1_su_adc_input(unsigned number, chanend c)

禁用由数字规定的模拟数字转换器输入。

此函数具有以下参数：

number 模拟数字转换器输入号码。

c 与XS1-SU模拟数字转换器相连的通道。

void disable_xs1_su_adc_input_streaming(unsigned number, streaming chanend c)

禁用由数字规定的模拟数字转换器输入。

此函数具有以下参数：

number 模拟数字转换器输入号码。

c 与XS1-SU模拟数字转换器相连的通道。

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

39.10 辅助函数

void crc32(unsigned &checksum, unsigned data, unsigned poly)
于循环冗余检查和中加入字节。

执行下列计算：

```
for (int i = 0; i < 32; i++) {
  int xorBit = (crc & 1);

  checksum  = (checksum >> 1) | ((data & 1) << 31);
  data = data >> 1;

  if (xorBit)
    checksum = checksum ^ poly;
}
```

此函数具有以下参数：
checksum 检查和的初始值，随最新检查和一起更新。

data 循环冗余检查的数据。

poly 循环冗余检查计算时所用的多项式。

unsigned crc8shr(unsigned &checksum, unsigned data, unsigned poly)
于循环冗余检查和内并入8位字节。

于8位以上有效数据位上计算循环冗余检查，数据右移8位返回。执行的计算如下：

```
for (int i = 0; i < 8; i++) {
  int xorBit = (crc & 1);

  checksum  = (checksum >> 1) | ((data & 1) << 31);
  data = data >> 1;

  if (xorBit)
    checksum = checksum ^ poly;
}
```

此函数具有以下参数：

checksum 检查和的初始值，随最新检查和一起更新。

data 数据

poly 循环冗余检查计算时所用的多项式。
此函数返回：

数据右移8位。

{unsigned, unsigned} l lmul(unsigned a, unsigned b, unsigned c, unsigned d)

乘以2个字节生成一个双字，并加入两个单字。

返回结果的高字和低字。此乘法无符号，不能超值。执行的计算如下：

```
(uint64_t)a * (uint64_t)b + (uint64_t)c + (uint64_t)d
```

此函数返回：

计算出的高值和低值。

{unsigned, unsigned} m mac(unsigned a, unsigned b, unsigned c, unsigned d)

乘以2个字节生成一个双字，并加入一个双字。

返回结果的高字和低字。执行的计算如下：

```
(uint64_t)a * (uint64_t)b + (uint64_t)c<<32 + (uint64_t)d
```

此函数返回：

计算出的高值和低值。

{signed, unsigned} m macs(signed a, signed b, signed c, unsigned d)

乘以2个带符号字节并添加一个双字，生成一个双字。

返回结果的高字和低字。执行的计算如下：

```
(int64_t)a * (int64_t)b + (int64_t)c<<32 + (int64_t)d
```

此函数返回：

计算出的高值和低值。

signed sext(unsigned a, unsigned b)

符号扩展输入。

第一个参数为符号扩展值，第二个参数包含一个数位位置。将较高或等同位置的所有数位值设置为较低一位的值。实际上，较低的数位b通过带符号的整形来解释。如果b小于1或大于32，则该结果与参数a相等。

Shiqiang Xiao 13-11-13 1:47 PM

已删除: 整数

此函数返回：

符号扩展值。

unsigned zext(unsigned a, unsigned b)

零扩展输入。

第一个参数为零扩展值，第二个参数包含一个数位位置。将较高或等同位置的所有数位值设置为0。实际上，较低的b数位通过不带符号的整形来解释。如果b小于1或大于32，则该结果与参数a相等。

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

此函数返回：

零扩展值。

40 xCORE 32位应用二进制界面

工具开发指南[8]中包含了XS1 32位应用二进制界面、XE文件格式和系统调用界面。

[8] http://www.xmos.com/docnum/X9114

第O部分平台配置

内容

41描述目标平台

本章内容

通过XN描述硬件平台。XN文件向XMOS编译器工具链提供关于目标硬件如XMOS设备、端口、闪存和振荡器的资料。

XMOS工具使用XN数据产生名为<platform.h>规定平台的头文件，编译、启动并调试多节点程序。

41.1 支持的网络拓扑

工具支持下列网络拓扑。

图60：支持的网络拓扑

网络拓扑	支持的配置
线形	仅适用XS1-L设备，最多16个节点
超立方	2级（两个节点） 3级（4个节点一圈） 3级（8个节点一个立方） 4级（16个节点典型立方）

Shiqiang Xiao 13-11-13 12:02 PM
已设置格式: 缩进: 首行缩进: 1.78 cm, 空格 段前: 9.2 pt, 行距: 固定 9.75 pt

Shiqiang Xiao 13-11-13 12:02 PM
已删除:

41.2 带两个程序包的板

图61描述了带两个成一直线XMOS L1设备的板，下文描述了一个相匹配的XN。

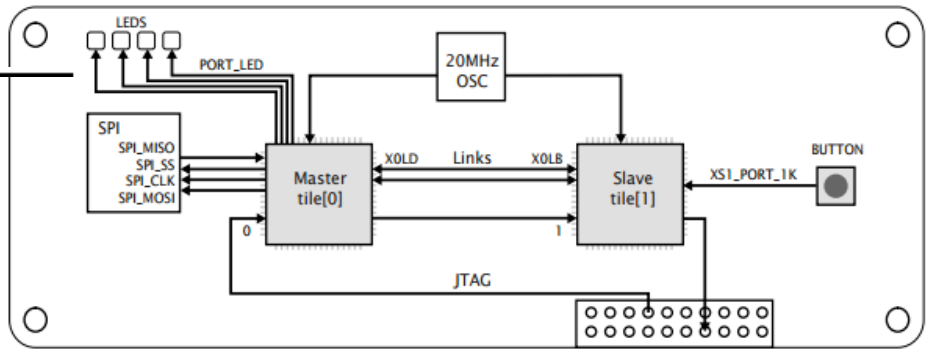
XN文件以XML声明开始。

```
<?xml version="1.0" encoding="UTF-8"?>
```

下列代码可启动网络。

```
<Network xmlns="http://www.xmos.com"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.xmos.com http://www.xmos.com">
```

图61示例硬件平台



下列代码描述了两个xCORE数据块，声明“tileref tile[2];”输出至名为<platform.h>的头文件。

```
<Declarations>
  <Declaration>tileref tile[2]</Declaration>
</Declarations>
```

下列代码描述了名为P1的程序包，包括一个名为Master的单节点。

```
<Packages>
  <Package Id="P1" Type="XS1-L1A-TQ128">
    <Nodes>
      <Node Id="Master" Type="XS1-L1A" InPackageId="0"
            Oscillator="20MHz" SystemFrequency="400MHz">
        <Boot>
          <Source Location="SPI:bootFlash"/>
          <Bootee NodeId="Slave" Tile="0"/>
        </Boot>
        <Tile Number="0" Reference="tile[0]">
          <Port Location="XS1_PORT_1A" Name="PORT_SPI_MISO"/>
          <Port Location="XS1_PORT_1B" Name="PORT_SPI_SS"/>
          <Port Location="XS1_PORT_1C" Name="PORT_SPI_CLK"/>
          <Port Location="XS1_PORT_1D" Name="PORT_SPI_MOSI"/>
          <Port Location="XS1_PORT_4A" Name="PORT_LED"/>
        </Tile>
      </Node>
    </Nodes>
  </Package>
```

Master节点为TQ128程序包中的一个400MHz XS1-L1A设备，通过20MHz振荡器计时。从名为“bootFlash”的SPI设备中启动，该SPI设备等级为“SPIFlash”。

数据块“0”的说明与数据块【0】相关，而端口1A、1B、1C、 1D和4A为规定符号名称，这些说明输出至名为<platform.h>的头文件。

下列代码描述了名为P2的程序包，包括一个名为Slave的单节点。

```
  <Package Id="P2" Type="XS1-L1A-TQ128">
    <Nodes>
      <Node Id="Slave" Type="XS1-L1A" InPackageId="0"
            Oscillator="20Mhz" SystemFrequency="400MHz">
        <Boot>
          <Source Location="LINK"/>
        </Boot>
        <Tile Number="0" Reference="tile[1]">
          <Port Location="XS1_PORT_1K" Name="PORT_BUTTON"/>
        </Tile>
      </Node>
    </Nodes>
  </Package>
</Packages>
```

Slave节点为TQ128程序包中的一个400MHz XS1-L1A设备，通过20MHz振荡器计时。从Master节点处通过xConnect Link 启动。

下面代码定义了一个2线xConnect链路，在Master节点上与链路X0LD连接，在Slave节点上与链路X0LB连接。

```
<Links>
  <Link Encoding="2wire" Delays="4,4">
    <LinkEndpoint NodeId="Master" Link="X0LD"/>
    <LinkEndpoint NodeId="Slave" Link="X0LB"/>
  </Link>
</Links>
```

链路符号内和符号间的迟延为4个时钟周期。
下列编码描述了与XMOS设备相连板上的部件清单。

```
<ExternalDevices>
  <Device NodeId="Master" Tile="0" Name="bootFlash"
          Class="SPIFlash" Type="AT25FS010">
     <Attribute Name="PORT_SPI_MISO" Value="PORT_SPI_MISO"/>
     <Attribute Name="PORT_SPI_SS"   Value="PORT_SPI_SS"/>
     <Attribute Name="PORT_SPI_CLK"  Value="PORT_SPI_CLK"/>
     <Attribute Name="PORT_SPI_MOSI" Value="PORT_SPI_MOSI"/>
   </Device>
 </ExternalDevices>
```

名为bootFlash设备于节点Master上与xCORE数据块0连接，并赋予属性——将设备上四个SPI引脚与端口连接（通过XFLASH识别级别XFLASH）。

下列代码描述了JTAG扫描链。

```
  <JTAGChain>
    <JTAGDevice NodeId="Master" Position="0"/>
    <JTAGDevice NodeId="Slave" Position="1"/>
  </JTAGChain>

</Network>
```

42 XN规范

本章内容

42.1 网络单元

xTIMEcomposer支持单一XN文件，该文件包含一个单一网络定义，此网络定义如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<Network xmlns="http://www.xmos.com"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.xmos.com http://www.xmos.com">
```

图62中列出了网络单元的XN层次。

42.2 声明

声明单元为一个或多个xCORE数据块提供符号名称。单一名称或多个名称通过下列形式支持：

tileref标识符

tileref标识符【常量表达】

同等声明输出至名为<platform.h>的头文件，用于XC程序。Tileref声明通过数据块单元（见§42.4.1）参考属性，与xCORE物理数据块相关。

示例

```
<Declaration>tileref master</Declaration>
<Declaration>tileref tile[8]</Declaration>
```

节点	编号	描述	章节
网络	1	xCORE网络	
声明	0+		
声明	1+	xCORE数据块声明	§??
程序包	1+		
程序包	1+	设备程序包	§??
节点	1		
节点	1+	节点声明	§??
数据块	1+	xCORE数据块	§??
端口	0+	xCORE符号型端口名	§??
启动	0 或 1	启动方式	§??
信号源	1	二进制定位	§??
启动对象	0+	启动的节点	§??
服务	0+	服务声明	§??
通道端	1+	通道端参数	§??
链路	0 或 1		
链路	1+	xConnect 链路声明	§??
链路端点	2	xConnect 链路端点	§??
外部设备	0 或 1		
设备	1+	外部设备	§??
属性	0+	设备属性	§??
JTAG链	0 或 1		
JTAG设备	1+	JTAG链中的设备	§??

图62： 单元的XN层次

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

42.3 程序包

程序包单元指一个程序包文件，描述来自xCORE端口和链路，至程序包引脚上的映射。

图63：XN程序包	属性	必要性	类型	描述
	标识符	必要	字符串	程序包名称，网络中所有程序包名称必须是惟一的。
	类型	必要	字符串	XML程序包名称，于XCC_DEVICE_PATH规定路径中搜索.pkg类型文件的工具。

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

示例

```
<Package id="L2" Type="XS1-L2A-QF124">
```

文件XS1-L2A-QF124.xml中描述了名为L2的程序包。

42.4 节点

节点单元描述了网络中一组xCORE数据块，所有数据块与单个开关相连。XMOS设备如G4或L1都为节点示例。

属性	必要性	类型	描述
标识符	不必要	字符串	节点名称，网络中所有节点名称必须是唯一的。
类型	必要	字符串	如果类型为periph:XS1-SU，则节点为XS1-SU外围节点。否则，类型则为描述节点的XML文件名。于XCC_DEVICE_PATH规定路径中搜索.pkg类型文件的工具。
基准	必要	字符串	将节点与声明中规定的xCORE数据块标识符相连。该属性适用于与类型为periph:XS1-SU的节点。
InPackageId	必要	字符串	将节点映射至程序包文件中的单元。
振荡器	不必要	字符串	PLL振荡器输入频率通过数字加单位MHz、KHz或Hz的形式规定。
OscillatorSrc	不必要	字符串	提供PLL振荡器输入的节点名称。
SystemFrequency	不必要	字符串	系统频率通过数字加单位MHz、KHz或Hz的形式规定，未设置时默认为400MHz。
PllFeedbackDivMin	不必要	整形	最小容许的PLL反馈分频器，未设置时默认为1.
ReferenceFrequency	不必要	字符串	基准时钟频率，通过数字加单位MHz、KHz或Hz的形式规定，未设置时默认为100MHz
PllDividerStageOneReg	不必要	整形	PLL 分频器 1 级寄存器值。
PllMultiplierStageReg	不必要	整形	PLL倍增器1级寄存器值。
PllDividerStageTwoReg	不必要	整形	PLL分频器2级寄存器值。
RefDiv	不必要	整形	SystemFrequency / RefDiv =ReferenceFrequency

图64：XN节点

可使用SystemFrequency、PllFeedbackDivMin和ReferenceFrequency属性自动配置PLL寄存器，或使用PllDividerStageOneReg、PllMultiplierStageReg、PllDividerStageTwoReg和RefDiv手动配置寄存器。如果提供了前三个属性，则无需提供后四个属性，反之亦然。

PLL振荡器输入频率可通过Oscillator或OscillatorSrc属性规定，如果提供了Oscillator属性，则无需提供OscillatorSrc属性，反之亦然。

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

使用手动配置时，必须提供PllDividerStageOneReg、PllMultiplierStageReg、PllDividerStageTwoReg和RefDiv属性，此时必须规定PLL振荡器的输入频率。工具使用这些值来设定PLL寄存器和基准时钟分频器。可于XS1-G 处理器（见X3221）和XS1-L 处理器（见X1433）的xCORE 频率控制文档中查看PLL分频器上的信息。

如果振荡器频率已定且未提供手动PLL属性，则使用自动配置。工具对PLL寄存器进行编程，从而达到目标系统频率，PLL反馈分频器大于等于最小值，此时达到目标基准时钟频率。如果任何一种限制条件不满足，则工具会发出警报并报告实际使用值。

如果未规定振荡器频率，则工具不会对PLL进行配置，节点引脚所确定的PLL寄存器初始值保持不变。

一个网络可以含XS1-L设备或XS1-G设备中的任意一个，但不同时拥有两个。

示例

```
<Node Id="MyL1" Type="XS1-L1A" Oscillator="20Mhz"
      SystemFrequency="410MHz" ReferenceFrequency="98.5Mhz">
```

如文件config_XS1-L1A.xml所述，名为MyL1的节点是L1设备。

42.4.1 数据块

数据块单元描述了单个xCORE数据块的性质。

属性	必要性	类型	描述
号码	必要	整形	节点中数据块的唯一号码，必须为0至n-1之间的任意值，其中，n为节点XML文件中的数据块数。
基准	不必要	字符串	将数据块与声明中tile[n]形式的标识符相关联。一个数据块最多能与一个标识符相关联。

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

图65 XN Tileref单元

示例

```
<Tile Number="0" Reference="tile[0]">
```

42.4.2 端口

端口单元提供了端口的符号型名称。
示例

属性	必要性	类型	描述
位置	必要	字符串	名为<xs1.h>标准头文件中定义的端口标识符。XC操作手册（见X1009）中描述了这些端口。
名称	必要	字符串	有效的C处理器标识符；网络中说明的所有端口名称必须是惟一的。

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

图66：XN端口单元

```
<Port Location="XS1_PORT_1I" Name="PORT_UART_TX"/>
<Port Location="XS1_PORT_1J" Name="PORT_UART_RX"/>
```

42.4.3 启动

Boot单元定义了节点是如何启动的，它包含一个"源"单元（见§42.4.4）及零或更多个从xConnect 链路启动的Bootee单元（见§42.4.5）。如果Source单元规定了xConnect链路，则无需规定Bootee单元。在XS1-L设备路线中，启动对象可根据自SPI启动的设备进行配置。

XMOS工具要求Boot单元可以从闪存中启动程序（见§22.1）。

42.4.4 源

Source单元规定了节点启动的位置，具有以下属性。

图67：XN源单元

属性	必要性	类型	描述
定位	必要	字符串	形式为SPI或LINK，一组设备单元中包含了设备名称。

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

只能配置XMOS XS1-L设备，通过xConnect链路启动。

示例

```
<Source Location="SPI:bootFlash"/>
```

42.4.5 启动对象

Bootee单元陈述了系统中通过xConnect链路启动的另一节点。如果该节点与其一个启动对象（见§42.5和§42.5.1）之间的xConnect链路超过一条，则工具会选择其中的一条用于启动。

示例

图 68 : XN Bootee单元	属性	必要性	类型	描述
	节点标识符	必要	字符串	可用于另一节点的标识符

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

```
<Bootee NodeId="Slave">
```

42.4.6 服务

Service单元规定了节点提供的XC服务函数。

图 69 : XN Service单元	属性	必要性	类型	描述
	原型	必要	字符串	服务函数原型，不包括服务关键字。本原型输出至名为<platform.h>的头文件，用于XC程序。

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

示例

```
<Service Proto="service_function(chanend c1, chanend c2)">
```

42.4.7 通道端

Chanend单元描述了XC服务函数所用的通道端参数。

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

图 70 : XN Service单元	属性	必要性	类型	描述
	标识符	必要	字符串	服务函数原型中通道端参数识别器
	端点	必要	整形	当前节点上的通道端号码。
	远程	必要	整形	与当前节点通道端相连的远程通道端号码。

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

示例

```
<Chanend Identifier="c" end="23" remote="5"/>
```

42.5 链路

系统规范文档（XS1-G: X7507，XS1-L: X1151）和链路性能文档（XS1-G: X7561, XS1-L: X2999）中描述了xConnect链路。

Link单元描述了xConnect链路的特征，必须确切包含两个LinkEndpoint产物（见§42.5.1）。

图71：XN链路单元

属性	必要性	类型	描述
编码	必要	字符串	必须为两线或五线
延迟	必要	字符串	为x、y型，其中，x规定了端点的外部迟延，y规定了端点的内部迟延。省略y值，则使用x，1；两值皆省略，则使用1.
旗标	必要	字符串	规定了链路的附加属性，使用XSCOPE值来确定使用的链路，从而向XScope发送追踪信息。

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

示例

```
<Link Encoding="2wire" Delays="4,4">
```

42.5.1 LinkEndpoint

LinkEndpoint描述了xConnect链路的一个端点，详情见系统规范文档（XS1-G: X7507、XS1-L: X1151）。每个端点将节点标识符与物理xConnect链路相关联。

图72：XN LinkEndpoint单元

属性	必要性	类型	描述
节点标识符	不必要	字符串	有效的节点标识符。
链路	不必要	字符串	链路标识符为XnLm型，其中，n代表数据块号，m为链接信。关于可用链路引脚分配，请看相应程序包数据表。
路由标识符	不必要	整形	xConnect链路网络上的路由标识符。
通道端	不必要	整形	通道端

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

Shiqiang Xiao 13-11-13 11:40 AM
已删除: 信道

端点通常为节点标识符与链路标识符的组合。在流式调试链路中，任意一个端点都必须为路由标识符与通道端的组合。

示例

```
<LinkEndpoint NodeId="0" Link="XOLD"/>
<LinkEndpoint RoutingId="0x8000" Chanend="1">
```

42.6 设备

Device单元描述与xCORE数据块相连但不直接与xConnect链路相连的设备

图 73 : XN Device单元	属性	必要性	类型	描述
	名称	必要	字符串	对设备命名的标识符。
	节点标识符	必要	字符串	与设备相连节点的标识符。
	数据块	必要	整形	与设备相连节点的数据块。
	等级	必要	字符串	设备等级
	类型	不必要	字符串	设备类型（与等级相关）

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

Shiqiang Xiao 13-11-13 1:47 PM
已删除: 整数

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

xTIMEcomposer识别SPIFlash级，并使用类型属性来区分闪存设备型号。

42.6.1 属性

属性单元描述设备的某一方面特质（见§42.6）。

图 74 : XN Attribute单元	属性	必要性	类型	描述
	名称	必要	字符串	规定设备的属性
	值	必要	字符串	规定与属性相关的值

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

Shiqiang Xiao 13-11-13 1:46 PM
已删除: 字串

XTIMEcomposer支持SPIFlash级别的下列设备属性名称：

PORT_SPI_MISO

SPI主进从出信号

PORT_SPI_SS

SPISlave选择信号

PORT_SPI_CLK

SPI时钟信号

PORT_SPI_MOSI

SPI主出从进信号

Shiqiang Xiao 13-11-13 1:46 PM
已删除:

示例

```
<Attribute Name="PORT_SPI_MISO" Value="PORT_SPI_MISO"/>
```

42.7 JTAGDevice

xTIMEcomposer使用JTAG于目标硬件上加载、调试程序。JTAGChain单元描述JTAG链路中的设备。这些单元的顺序决定了它们于JTAG链路中的顺序。

图 75 : XN JTAGDevice单元	属性	必要性	类型	描述
	节点标识符	必要	字符串	有效的节点标识符

示例

```
<!-- N1 comes before N2 in the JTAG chain -->
<JTAGDevice NodeId="N1">
<JTAGDevice NodeId="N2">
```