



UG643 ngày 2 tháng 12 năm 2009

Bộ sưu tập tài liệu về hệ điều hành và thư viện

Tổng quan

Nhiều gói phần mềm Xilinx® được cung cấp cùng với Bộ phát triển nhúng (EDK), bao gồm trình điều khiển, thư viện, gói hỗ trợ bo mạch và hệ điều hành hoàn chỉnh để giúp bạn phát triển nền tảng phần mềm. Bộ sưu tập tài liệu này cung cấp thông tin về những điều này và về sự phát triển của Gói Hỗ trợ Hội đồng quản trị cho các hệ điều hành VxWorks, Linux 2.4 (Monta Vista Linux 3.1) và Linux 2.6. Tài liệu đầy đủ cho các hệ điều hành khác có thể được tìm thấy trong hướng dẫn tham khảo cụ thể cho từng hệ điều hành. Trình điều khiển thiết bị được ghi lại cùng với tài liệu ngoại vi tương ứng.

Các tài liệu trong bảng sau được bao gồm trong bộ sưu tập này. Để xem một tài liệu, hãy nhấp vào tên của nó.

Bảng 1: Nội dung thu thập tài liệu về hệ điều hành và thư viện

Tên tài liệu	Bản tóm tắt
Thư viện C chuẩn LibXil	Mô tả các thư viện phần mềm có sẵn cho các bộ xử lý nhúng.
Độc lập (v2.00.a)	Nền tảng Độc lập là một nền tảng hệ điều hành (OS) đơn luồng, đơn giản, cung cấp lớp thấp nhất của các mô-đun phần mềm được sử dụng để truy cập các chức năng dành riêng cho bộ xử lý. Một số chức năng điển hình được cung cấp bởi nền tảng Độc lập bao gồm thiết lập hệ thống ngắt và ngoại lệ, định cấu hình bộ nhớ đệm và các chức năng cụ thể khác của phần cứng.
Xilkernel (v4.00.a)	Xilkernel là một nhân xử lý nhúng đơn giản có thể được tùy chỉnh ở mức độ lớn cho một hệ thống nhất định. Xilkernel có các tính năng chính của một hạt nhân nhúng như đa tác vụ, lập lịch trù ớc theo hướng ưu tiên, giao tiếp giữa các quá trình, cơ sở đồng bộ hóa và xử lý ngắn. Xilkernel nhỏ, mô-đun, người dùng có thể tùy chỉnh và nó có thể được sử dụng trong các cấu hình hệ thống khác nhau. Các ứng dụng liên kết tĩnh với hạt nhân để tạo thành một tập thực thi duy nhất.
Hệ thống tệp LibXil FAT (FATFS) (v1.00.a)	Thư viện truy cập hệ thống tệp XilFATFS FAT cung cấp quyền truy cập đọc và ghi vào các tệp được lưu trữ trên thiết bị flash hoặc microdrive nhỏ gọn Xilinx System ACE™.
Hệ thống tệp bộ nhớ LibXil (MFS) (v1.00.a)	Mô tả một hệ thống tệp dựa trên bộ nhớ, đơn giản có thể nằm trong RAM, ROM hoặc bộ nhớ Flash.
Thư viện lwIP (lwIP130.v1.00.b)	Mô tả cổng EDK của thư viện mạng bên thứ ba, Light Weight IP (lwIP) phiên bản v1.30.b, dành cho bộ xử lý nhúng.
LibXil Flash (v1.03a)	Mô tả chức năng được cung cấp trong thư viện lập trình flash. Thư viện này cung cấp quyền truy cập vào các thiết bị bộ nhớ flash tuân theo tiêu chuẩn Giao diện Flash chung (CFI). Thiết bị Intel và AMD CFI cho một số bộ phận cụ thể hiện đang được hỗ trợ.

© 2009 Xilinx, Inc. XILINX, biểu trưng Xilinx, Virtex, Spartan, ISE và các nhãn hiệu được chỉ định khác có trong tài liệu này là nhãn hiệu của Xilinx tại Hoa Kỳ và các quốc gia khác. Tất cả các nhãn hiệu khác là tài sản của chủ sở hữu tương ứng của họ.

Xilinx sẽ tiết lộ hướng dẫn sử dụng, hướng dẫn sử dụng, ghi chú phát hành và / hoặc thông số kỹ thuật này ("Tài liệu") cho bạn chỉ để sử dụng trong việc phát triển các thiết kế để vận hành với các thiết bị phần cứng của Xilinx. Bạn không được phép sao chép, phân phối, tái xuất bản, tái xuất bản, hiển thị, đăng hoặc truyền Tài liệu dưới bất kỳ hình thức nào hoặc bằng bất kỳ phương tiện nào bao gồm như ng không giới hạn ở Xilinx. Xilinx từ chối rõ ràng bất kỳ trách nhiệm pháp lý nào phát sinh từ việc bạn sử dụng Tài liệu. Xilinx bảo lưu quyền, theo quyết định riêng của mình, thay đổi Tài liệu mà không cần thông báo bất kỳ lúc nào. Xilinx không có nghĩa vụ phải sửa bất kỳ lỗi nào có trong Tài liệu hoặc thông báo cho bạn về bất kỳ chỉnh sửa hoặc cập nhật nào. Xilinx từ chối rõ ràng bất kỳ trách nhiệm pháp lý nào liên quan đến hỗ trợ kỹ thuật hoặc hỗ trợ có thể được cung cấp cho bạn liên quan đến Thông tin.

TÀI LIỆU ĐƯỢC CÔNG BỐ CHO BẠN "NGUYÊN TRẠNG" KHÔNG ĐƯỢC BẢO HÀNH BẤT KỲ HÌNH THỨC NÀO. XILINX KHÔNG CÓ BẢO ĐÁM KHẮC, DÙ THỂ HIỆN, NGƯ Ý HOẶC TRUYỀN NGHĨA, LIÊN QUAN ĐẾN TÀI LIỆU, BAO GỒM BẤT KỲ BẢO ĐÁM NÀO VỀ TÍNH KHẢ NĂNG, PHÙ HỢP VỚI MỤC ĐÍCH CỤ THỂ HOẶC KHÔNG ĐÁM BẢO QUYỀN CỦA BÊN THỨ BA. TRONG MỌI TRƯỜNG HỢP XILINX SẼ CHỊU TRÁCH NHIỆM PHÁP LÝ ĐỐI VỚI BẤT KỲ THIẾT HẠI HẬU QUẢ, DÙNG, BẤT CỨ, ĐẶC BIỆT HOẶC SỰ CÓ NÀO, BAO GỒM BẤT CỨ MẤT DỮ LIỆU HOẶC MẤT LỢI NHUẬN NÀO PHÁT SINH TỪ VIỆC BẠN SỬ DỤNG TÀI LIỆU.

Bảng 1: Nội dung thu thập tài liệu về hệ điều hành và thư viện (Tiếp theo)

Tên tài liệu	Bản tóm tắt
LibXil Isf (v1.00.a)	Mô tả thư viện phần cứng trong hệ thống Flash, cho phép phần mềm lớp cao hơn (chẳng hạn như một ứng dụng) giao tiếp với Isf. LibXil Isf hỗ trợ Xilinx Flash trong hệ thống và bộ nhớ Flash nối tiếp bên ngoài của Atmel (AT45XXXD), Intel (S33) và ST Microelectronics (STM) (M25PXX).
Gói hỗ trợ bo mạch tự động tạo ra Tornado 2.x (VxWorks 5.x)	Mô tả sự phát triển của Gói hỗ trợ hội đồng quản trị Tornado 2.x (VxWorks 5.x) (BSP).
Tự động tạo các gói hỗ trợ bo mạch của Wind River VxWorks 6.3	Mô tả sự phát triển của Wind River VxWorks 6.3 BSP.
Tự động tạo gói hỗ trợ bo mạch Wind River VxWorks 6.5	Mô tả sự phát triển của Wind River VxWorks 6.5 BSP.
Tự động tạo ra các gói hỗ trợ bo mạch của Wind River VxWorks 6.7	Mô tả sự phát triển của Wind River VxWorks 6.7 BSPs.
Tự động tạo gói hỗ trợ bo mạch Linux 2.6	Mô tả sự phát triển của Linux 2.6 BSP.

Về Thư viện

Thư viện hỗ trợ C chuẩn bao gồm newlib, libc, chứa các hàm C chuẩn như stdio, stdlib và các quy trình chuỗi. Thư viện toán học là một cải tiến so với thư viện toán học newlib, libm, và cung cấp các quy trình toán học tiêu chuẩn.

Các thư viện LibXil bao gồm:

- Trình điều khiển LibXil (Trình điều khiển thiết bị Xilinx)
- LibXil MFS (hệ thống tệp bộ nhớ Xilinx)
- LibXil Flash (thư viện lập trình flash song song)
- LibXil Isf (một thư viện lập trình flash nối tiếp)

Có hai tùy chọn hệ điều hành được cung cấp trong gói phần mềm Xilinx: Nền tảng độc lập và Xilkernel.

Hầu hết các quy trình trong thư viện được viết bằng C và có thể được chuyển sang bất kỳ nền tảng nào. Trình tạo Thư viện (Libgen) định cấu hình các thư viện cho một bộ xử lý nhúng, sử dụng các thuộc tính được xác định trong tệp Đặc tả Phần mềm Vi xử lý (MSS).

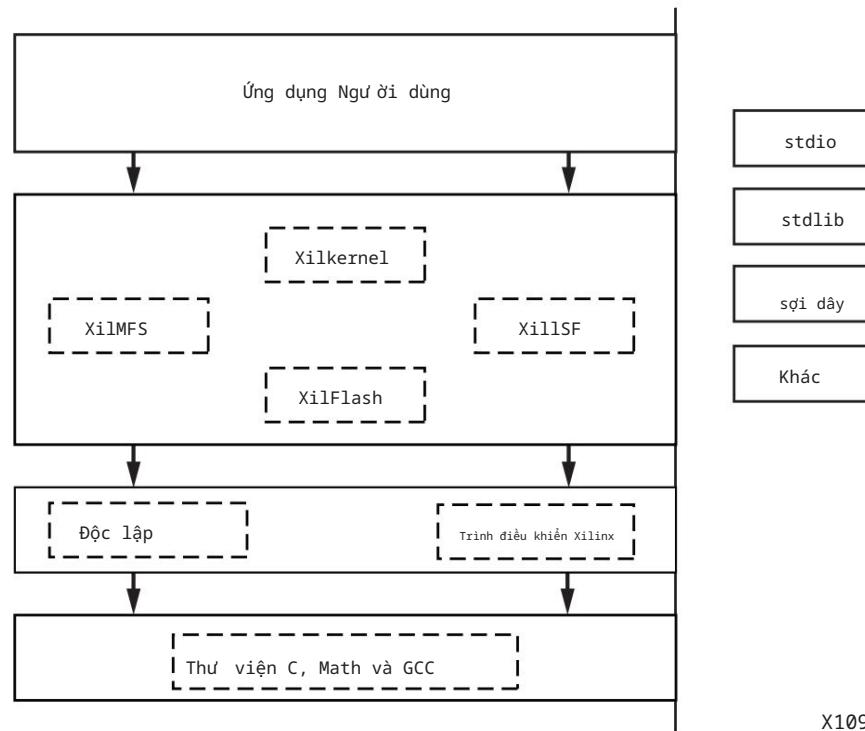
Các ứng dụng của người dùng phải bao gồm các tiêu đề thích hợp và liên kết với các thư viện cần thiết để biên dịch thích hợp và đưa vào các chức năng cần thiết. Các thư viện này và các tệp bao gồm tương ứng của chúng được tạo trong thư mục bộ xử lý \ lib và \ bao gồm tương ứng trong dự án hiện tại. Các tùy chọn -I và -L của trình biên dịch đang được sử dụng nên được tận dụng để thêm các thư mục này vào các đường dẫn tìm kiếm. Libgen điều chỉnh việc biên dịch của từng thành phần phần mềm. Tham khảo chương "Libgen" và "Đặc điểm kỹ thuật phần mềm vi xử lý" trong Sổ tay Tham khảo Công cụ Hệ thống Nhúng để biết thêm thông tin.

Thư viện Cơ quan

Tổ chức của các thư viện được minh họa trong hình bên dưới. Như được hiển thị, ứng dụng của bạn có thể giao diện với các thành phần theo nhiều cách khác nhau. Các thư viện độc lập với nhau, ngoại trừ một số tương tác. Ví dụ: Xilkernel sử dụng nền tảng Độc lập trong nội bộ. Trình điều khiển LibXil và Trình điều khiển độc lập tạo thành lớp trung gian phần cứng thấp nhất. Thư viện và các thành phần hệ điều hành dựa trên các thành phần thư viện C tiêu chuẩn. Thư viện toán học, libm.a cũng có sẵn để liên kết với các ứng dụng người dùng.

Lưu ý: "Trình điều khiển LibXil" là trình điều khiển thiết bị được bao gồm trong nền tảng phần mềm để cung cấp giao diện cho các thiết bị ngoại vi trong hệ thống. Các trình điều khiển này được cung cấp cùng với EDK và được cấu hình bởi Libgen. Bộ sưu tập tài liệu này chứa một phần thảo luận ngắn gọn về khái niệm trình điều khiển thiết bị và cách chúng tích hợp với nền tảng phần mềm trong EDK.

Có tính đến một số hạn chế và hàm ý, được mô tả trong hướng dẫn tham khảo cho từng thành phần, bạn có thể trộn và kết hợp các thư viện thành phần.



X10968_102908

Hình 1: Tổ chức Thư viện



Thư viện LibXil Standard C

UG 645 ngày 15 tháng 4 năm 2009

Bản tóm tắt

Tài liệu này mô tả các thư viện phần mềm có sẵn cho các bộ xử lý nhúng. Tài liệu bao gồm các phần sau:

- “Tổng quan”
- “Tài nguyên bổ sung”
- “Thư viện C chuẩn (libc.a)”
- “Thư viện Xilinx C (libxil.a)”
- “Chức năng Đầu vào / Đầu ra”
- “Chức năng quản lý bộ nhớ”
- “Phép toán số học”
- “An toàn chuỗi”

Tổng quan

Các thư viện và trình điều khiển thiết bị Xilinx® Embedded Development Kit (EDK) cung cấp các chức năng thư viện tiêu chuẩn, cũng như các chức năng để truy cập thiết bị ngoại vi. Các thư viện EDK được Libgen tự động định cấu hình cho mọi dự án dựa trên tệp Đặc tả Phần mềm Vì xử lý (MSS). Các thư viện này và các tệp bao gồm được lưu trong lib của dự án hiện tại và bao gồm các thư mục, tư ơng ứng. Tùy chọn -I và -L của mb-gcc được sử dụng để thêm các thư mục này vào đường dẫn tìm kiếm thư viện của nó.

Thêm vào Tài nguyên

- Hướng dẫn Tham khảo Bộ xử lý MicroBlaze™
[http://www.xilinx.com/ise/embedded\(mb_ref_guide.pdf](http://www.xilinx.com/ise/embedded(mb_ref_guide.pdf)
- Tài liệu tham khảo về công cụ hệ thống nhúng
http://www.xilinx.com/ise/embedded_design_prod/platform_studio.htm

Thư viện C chuẩn (libc.a)

Thư viện C chuẩn, libc.a, chứa các hàm C chuẩn được biên dịch cho bộ xử lý MicroBlaze hoặc bộ xử lý PowerPC®. Bạn có thể tìm thấy các tệp tiêu đề tư ơng ứng với các hàm tiêu chuẩn C này trong <XILINX_EDK> / gnu / <processor> / <platform> / <processor lib> / include, trong đó:

- <XILINX_EDK> là <Thư mục cài đặt>
- <processor> là powerpc-eabi hoặc microblaze

© 2009 Xilinx, Inc. XILINX, biểu trưng Xilinx, Virtex, Spartan, ISE và các nhãn hiệu được chỉ định khác có trong tài liệu này là nhãn hiệu của Xilinx tại Hoa Kỳ và các quốc gia khác. Tất cả các nhãn hiệu khác là tài sản của chủ sở hữu tư ơng ứng của họ.

Xilinx sẽ tiết lộ hướng dẫn sử dụng, hướng dẫn sử dụng, ghi chú phát hành và / hoặc thông số kỹ thuật này ("Tài liệu") cho bạn chỉ để sử dụng trong việc phát triển các thiết kế để vận hành với các thiết bị phần cứng của Xilinx. Bạn không được phép sao chép, phân phối, tái xuất bản, tái xuất, hiển thị, đăng hoặc truyền Tài liệu dưới bất kỳ hình thức nào hoặc bằng bất kỳ phư ơng tiện nào bao gồm như ng không giới hạn ở Xilinx. Xilinx từ chối rõ ràng bất kỳ trách nhiệm pháp lý nào phát sinh từ việc bạn sử dụng Tài liệu. Xilinx bảo lưu quyền, theo quyết định riêng của mình, thay đổi Tài liệu mà không cần thông báo bất kỳ lúc nào. Xilinx không có nghĩa vụ phải sửa bất kỳ lỗi nào có trong Tài liệu hoặc thông báo cho bạn về bất kỳ chỉnh sửa hoặc cập nhật nào. Xilinx từ chối rõ ràng bất kỳ trách nhiệm pháp lý nào liên quan đến hỗ trợ kỹ thuật hoặc hỗ trợ có thể được cung cấp cho bạn liên quan đến Thông tin.

TÀI LIỆU ĐƯỢC CÔNG BỐ CHO BẠN "NGUYÊN TRẠNG" KHÔNG ĐƯỢC BẢO HÀNH BẤT KỲ HÌNH THỨC NÀO. XILINX KHÔNG CÓ BẢO ĐÁM KHẮC, DÙ THẾ HIỆN, NGƯ Ý HOẶC TRUYỀN NGHĨA, LIÊN QUAN ĐẾN TÀI LIỆU, BAO GỒM BẤT KỲ BẢO ĐÁM NÀO VỀ TÍNH KHẢ NĂNG, PHÙ HỢP VỚI MỤC ĐÍCH CỤ THỂ HOẶC KHÔNG ĐÁM BẢO QUYỀN CỦA BÊN THỨ BA. TRONG MỌI TRƯỜNG HỢP XILINX SẼ CHỊU TRÁCH NHIỆM PHÁP LÝ ĐỐI VỚI BẤT KỲ THIẾT HẠI HẬU QUẢ, DÙNG, BẤT CỨ, ĐẶC BIỆT HOẶC SỰ CÓ NÀO, BAO GỒM BẤT CỨ MẤT DỮ LIỆU HOẶC MẤT LỢI NHUẬN NÀO PHÁT SINH TỪ VIỆC BẠN SỬ DỤNG TÀI LIỆU.

- <platform> là sol, nt hoặc lin
- <processor-lib> là powerpc-eabi hoặc microblaze-xilinx-elf

Các thư mục và chức năng lib.c là:

_ansi.h	máy fastmath.h /	reent.h	stdlib.h	utime.h
_syslist.h fcntl.h		malloc.h	regdef.h	string.h
ar.h	float.h	toán học.h	setjmp.h	sys /
khẳng định.h	grp.h	đường dẫn.h	signal.h	termios.h
ctype.h	ieeefp.h	process.h stdarg.h		time.h
Dirent.h	giới hạn.h	pthread.h stddef.h		unctrl.h
errno.h	locale.h	pwd.h	stdio.h	unistd.h

Các chương trình truy cập các hàm thư viện C chuẩn phải được biên dịch như sau:

Đối với bộ xử lý MicroBlaze:

mb-gcc <C tệp>

Đối với bộ xử lý PowerPC:

powerpc-eabi-gcc <C files>

Thư viện libc được đưa vào tự động.

Đối với các chương trình truy cập các hàm toán học libm, hãy chỉ định tùy chọn lm .

Tham khảo phần “[Giao diện nhị phân ứng dụng MicroBlaze \(ABI\)](#)” trong Hướng dẫn Tham khảo Bộ xử lý MicroBlaze để biết thông tin về Thư viện thời gian chạy C. “[Tài nguyên bổ sung](#)”, [trang 1](#) chứa một liên kết đến tài liệu.

Thư viện Xilinx C (libxil.a)

Thư viện Xilinx C, libxil.a, chứa các tệp đối tượng sau cho bộ xử lý nhúng bộ xử lý MicroBlaze:

```
_exception_handler.o
_interrupt_handler.o
_program_clean.o
_program_init.o
```

Các trình xử lý ngoại lệ và ngắt mặc định được cung cấp. Thư viện libxil.a được đưa vào tự động.

Các chương trình truy cập các hàm thư viện Xilinx C phải được biên dịch như sau:

mb-gcc <C tệp>

Đầu ra đầu vào Chức năng

Các thư viện EDK chứa các hàm C tiêu chuẩn cho I / O, chẳng hạn như printf và scanf. Các chức năng này lớn và có thể không phù hợp với bộ xử lý nhúng.

Nguyên mẫu cho các hàm này có trong stdio.h.

Lưu ý: Các quy trình I / O tiêu chuẩn C như printf, scanf, vfprintf, theo mặc định, dòng được lưu vào bộ đệm. Để thay đổi lưu trữ dữ liệu thành không đệm, bạn phải gọi setvbuf một cách thích hợp. Ví dụ:

```
setvbuf (stdout, NULL, _IONBF, 0);
```

Các quy trình Đầu vào / Đầu ra này yêu cầu một dòng mới được kết thúc bằng cả CR và LF. Đảm bảo rằng hành vi CR / LF đầu cuối của bạn tương ứng với yêu cầu này.

Tham khảo chương "Đặc điểm kỹ thuật phần mềm vi xử lý (MSS)" trong Sổ tay Tham khảo Công cụ Hệ thống Nhúng để biết thông tin về cách thiết lập thiết bị đầu vào và đầu ra tiêu chuẩn cho hệ thống. "Tài nguyên bổ sung", trang 1 chứa một liên kết đến tài liệu.

Ngoài các chức năng C tiêu chuẩn, bộ xử lý EDK (bộ xử lý MicroBlaze và Thư viện bộ xử lý PowerPC 405) cung cấp các chức năng I / O nhỏ hơn sau:

`void print (char *)`

Hàm này in một chuỗi tới thiết bị ngoại vi được chỉ định làm đầu ra tiêu chuẩn trong tệp Đặc tả Phần mềm Vi xử lý (MSS). Hàm này xuất ra chuỗi được truyền như hiện tại và không có diễn giải về chuỗi được truyền. Ví dụ: "\ n" được truyền vào được hiểu là một ký tự dòng mới chứ không phải là ký tự xuống dòng và một dòng mới như trường hợp của hàm ANSI C printf.

`void putnum (int)`

Hàm này chuyển đổi một số nguyên thành một chuỗi thập lục phân và in nó ra thiết bị ngoại vi được chỉ định làm đầu ra tiêu chuẩn trong tệp MSS.

`void xil_printf (const * char ctrl1, ...)`

xil_printf là một triển khai trọng lượng nhẹ của printf. Nó có kích thước nhỏ hơn nhiều (chỉ 1 kB). Nó không có hỗ trợ cho số dấu phẩy động. xil_printf cũng không hỗ trợ in các số dài (chẳng hạn như 64-bit).

Lưu ý: Giới thiệu về hỗ trợ chuỗi định dạng:

Chuỗi định dạng bao gồm không hoặc nhiều chỉ thị: các ký tự thông thường (không phải%), được sao chép không thay đổi vào luồng đầu ra; và thông số kỹ thuật chuyển đổi, mỗi đặc điểm trong số đó dẫn đến việc tìm nạp không hoặc nhiều đối số tiếp theo. Mỗi đặc tả chuyển đổi được giới thiệu bởi ký tự% và kết thúc bằng một thông số chuyển đổi.

Ở giữa có thể có (theo thứ tự) không hoặc nhiều cờ, độ rộng trựòng tối thiểu tùy chọn và độ chính xác tùy chọn. Các ký tự cờ được hỗ trợ là:

Ký tự% được sau bởi không hoặc nhiều cờ sau:

Ø Giá trị không được đệm. Đổi với chuyển đổi d, x, giá trị được chuyển đổi được đệm ở bên trái bằng các số không thay vì khoảng trắng.

Nếu cả hai cờ Ø và - đều xuất hiện, thì cờ Ø sẽ bị bỏ qua.

- Giá trị được chuyển đổi sẽ được điều chỉnh để lại trên ranh giới thực địa.

(Mặc định là biến minh bên phải.) Ngoại trừ n chuyển đổi, giá trị được chuyển đổi được đệm ở bên phải với khoảng trắng, thay vì ở bên trái với khoảng trắng hoặc số không. A - ghi đè Ø nếu cả hai đều được cho trựòng.

Lưu ý: Về Độ rộng Trựòng được Hỗ trợ:

Độ rộng trựòng được biểu thị bằng một chuỗi chữ số thập phân tùy chọn (với một chữ số khác ở chữ số đầu tiên) chỉ định độ rộng trựòng tối thiểu. Nếu giá trị được chuyển đổi có ít ký tự hơn chiều rộng trựòng, nó sẽ được đệm bằng khoảng trắng ở bên trái (hoặc bên phải, nếu cờ điều chỉnh bên trái đã được đưa ra). Các chỉ số chuyển đổi được hỗ trợ là:

d Đổi số int được chuyển đổi thành ký hiệu thập phân có dấu.

l Đổi số int được chuyển đổi thành ký hiệu dài có dấu.

x Đổi số int không dấu được chuyển đổi thành ký hiệu thập lục phân không dấu. Các chữ cái abcdef được sử dụng cho các chuyển đổi x.

c Đổi số int được chuyển đổi thành một ký tự không dấu và ký tự kết quả là bằng văn bản.

s Đổi số const char * được mong đợi là một con trỏ đến một mảng ký tự gõ (con trỏ tới một chuỗi).

Các ký tự từ mảng được ghi tối đa (nhưng không bao gồm) một NULL kết thúc tính cách; nếu một độ chính xác được chỉ định, thì không được ghi nhiều hơn số đã chỉ định.

Nếu một độ chính xác được cung cấp, không cần ký tự null; nếu độ chính xác không được chỉ định hoặc lớn hơn kích thước của mảng, thì mảng phải chứa ký tự NULL kết thúc.

Kỉ niệm Ban quản lý Chức năng

Các thư viện C của bộ xử lý MicroBlaze và bộ xử lý PowerPC hỗ trợ các chức năng quản lý bộ nhớ tiêu chuẩn như malloc (), calloc () và free (). Cấp phát bộ nhớ động cung cấp bộ nhớ từ đồng chương trình. Con trỏ heap bắt đầu ở bộ nhớ thấp và phát triển về bộ nhớ cao. Không thể tăng kích thước của đồng trong thời gian chạy. Do đó, một giá trị thích hợp phải được cung cấp cho kích thước heap tại thời điểm biên dịch. Hàm malloc () yêu cầu heap phải có kích thước ít nhất 128 byte để có thể cấp phát bộ nhớ động (ngay cả khi yêu cầu động nhỏ hơn 128 byte). Giá trị trả về của malloc phải luôn được kiểm tra để đảm bảo rằng nó thực sự có thể cấp phát bộ nhớ được yêu cầu.

Môn số học Hoạt động

Phần mềm triển khai số học nguyên và dấu phẩy động có sẵn dưới dạng các quy trình thư viện trong libgcc.a cho cả hai bộ xử lý. Trình biên dịch cho cả hai bộ xử lý sẽ chèn các lệnh gọi đến các quy trình này trong mã được tạo ra, trong trường hợp phần cứng không hỗ trợ nguyên thủy số học với một lệnh.

Bộ xử lý MicroBlaze

Số học Số nguyên

Theo mặc định, phép nhân số nguyên được thực hiện trong phần mềm bằng cách sử dụng hàm thư viện __mulsi3. Phép nhân số nguyên được thực hiện trong phần cứng nếu tùy chọn mb-gcc, -mno-xl-soft-mul, được chỉ định.

Phép toán chia số nguyên và mod được thực hiện trong phần mềm bằng cách sử dụng các hàm thư viện __divsi3 và __modsi3. Bộ xử lý MicroBlaze cũng có thể được tùy chỉnh để sử dụng dài phân cách cứng, trong trường hợp đó, lệnh div được sử dụng thay cho quy trình thư viện __divsi3.

Các hàm nhân, chia và mod chính xác kép lần lượt được thực hiện bởi các hàm thư viện __muldi3, __divdi3 và __moddi3.

Phiên bản chua ký của các thao tác này tương ứng với phiên bản đã ký được mô tả ở trên, nhưng có tiền tố là __u thay vì __.

Số học Dấu phẩy động

Tất cả các phép cộng, trừ, nhân, chia và chuyển đổi dấu phẩy động đều được thực hiện bằng các hàm phần mềm trong thư viện C.

Bộ xử lý PowerPC

Số học Số nguyên

Các phép toán cộng và trừ số nguyên được cung cấp trong phần cứng; không có thư viện phần mềm cụ thể nào cho bộ xử lý PowerPC.

Số học Dấu phẩy động

Bộ xử lý PowerPC hỗ trợ tất cả số học dấu phẩy động được thực hiện trong thư viện C tiêu chuẩn.

An toàn chủ đề

Thư viện C tiêu chuẩn được cung cấp với EDK không được xây dựng cho môi trường đa luồng. Các hàm STDO như `printf()`, `scanf()` và các hàm quản lý bộ nhớ như `malloc()` và `free()` là những ví dụ phổ biến về các hàm không an toàn cho luồng. Khi sử dụng thư viện C trong môi trường đa luồng, các kỹ thuật loại trừ lẫn nhau thích hợp phải được sử dụng để bảo vệ các chức năng không an toàn của luồng.



UG 647 ngày 15 tháng 4 năm 2009

Độc lập (v.2.00.a)

Bản tóm tắt

Độc lập là lớp thấp nhất của mô-đun phần mềm được sử dụng để truy cập các chức năng cụ thể của bộ xử lý. Độc lập được sử dụng khi một ứng dụng truy cập trực tiếp vào các tính năng của bo mạch / bộ xử lý và nằm bên dưới lớp hệ điều hành.

Tài liệu này bao gồm các phần sau:

- “[Tài nguyên bổ sung](#)”
- “[API bộ xử lý MicroBlaze™](#)”
- “[API bộ xử lý PowerPC 405™](#)”
- “[API bộ xử lý PowerPC 440™](#)”
- “[Hồ sơ chương trình](#)”
- “[Định cấu hình độc lập](#)”

Thêm vào Tài nguyên

[Hướng dẫn Tham khảo Bộ xử lý MicroBlaze™](#)

[Hướng dẫn Tham khảo Bộ xử lý PowerPC®](#)

Tài liệu Tham khảo Công cụ Hệ thống Nhúng

http://www.xilinx.com/support/documentation/dt_edk.htm

© 2009 Xilinx, Inc. XILINX, biểu trưng Xilinx, Virtex, Spartan, ISE và các nhãn hiệu được chỉ định khác có trong tài liệu này là nhãn hiệu của Xilinx tại Hoa Kỳ và các quốc gia khác. Tất cả các nhãn hiệu khác là tài sản của chủ sở hữu tương ứng của họ.

Xilinx sẽ tiết lộ hướng dẫn sử dụng, hướng dẫn sử dụng, ghi chú phát hành và / hoặc thông số kỹ thuật này (“Tài liệu”) cho bạn chỉ để sử dụng trong việc phát triển các thiết kế để vận hành với các thiết bị phần cứng của Xilinx. Bạn không được phép sao chép, phân phối, tái xuất bản, tái xuất bản, hiển thị, đăng hoặc truyền Tài liệu dưới bất kỳ hình thức nào hoặc bằng bất kỳ phương tiện nào bao gồm như ng không giới hạn ở Xilinx. Xilinx từ chối rõ ràng bất kỳ trách nhiệm pháp lý nào phát sinh từ việc bạn sử dụng Tài liệu. Xilinx bảo lưu quyền, theo quyết định riêng của mình, thay đổi Tài liệu mà không cần thông báo bất kỳ lúc nào. Xilinx không có nghĩa vụ phải sửa bất kỳ lỗi nào có trong Tài liệu hoặc thông báo cho bạn về bất kỳ chỉnh sửa hoặc cập nhật nào. Xilinx từ chối rõ ràng bất kỳ trách nhiệm pháp lý nào liên quan đến hỗ trợ kỹ thuật hoặc hỗ trợ có thể được cung cấp cho bạn liên quan đến Thông tin.

TÀI LIỆU ĐƯỢC CÔNG BỐ CHO BẠN “NGUYÊN TRẠNG” KHÔNG ĐƯỢC BẢO HÀNH BẤT KỲ HÌNH THỨC NÀO. XILINX KHÔNG CÓ BẢO ĐÁM KHÁC, DÙ THỂ HIỆN, NGỦ Ý HOẶC TRUYỀN NGHĨA, LIÊN QUAN ĐẾN TÀI LIỆU, BAO GỒM BẤT KỲ BẢO ĐÁM NÀO VỀ TÍNH KHẢ NĂNG, PHÙ HỢP VỚI MỤC ĐÍCH CỤ THỂ HOẶC KHÔNG ĐÁM BẢO QUYỀN CỦA BÊN THỨ BA. TRONG MỌI TRƯỜNG HỢP XILINX SẼ CHỊU TRÁCH NHIỆM PHÁP LÝ ĐỐI VỚI BẤT KỲ THIẾT HẠI HẬU QUẢ, DÙNG, BẤT CỨ, ĐẶC BIỆT HOẶC SỰ CÓ NÀO, BAO GỒM BẤT CỨ MẤT DỮ LIỆU HOẶC MẤT LỢI NHUẬN NÀO PHÁT SINH TỪ VIỆC BẠN SỬ DỤNG TÀI LIỆU.

MicroBlaze API bộ xử lý

Danh sách sau đây là tóm tắt về các phần API bộ xử lý MicroBlaze. Bạn có thể nhấp vào một liên kết để chuyển trực tiếp đến phần chức năng.

- "Xử lý ngắt bộ xử lý MicroBlaze"
- "Xử lý ngoại lệ bộ xử lý MicroBlaze"
- "Xử lý bộ nhớ cache hư hỏng dẫn bộ xử lý MicroBlaze"
- "Xử lý bộ nhớ đệm dữ liệu của bộ xử lý MicroBlaze"
- "Macro giao diện Fast Simplex Link (FSL) của Bộ xử lý MicroBlaze"
- "Cờ Macro FSL của Bộ xử lý MicroBlaze"
- "Tóm tắt Macro Pseudo-asm của Bộ xử lý MicroBlaze"
- "Quy trình truy cập đăng ký phiên bản bộ xử lý MicroBlaze (PVR) và macro"
- "Xử lý tệp của bộ xử lý MicroBlaze"
- "Bộ xử lý MicroBlaze Errno"

Xử lý ngắt bộ xử lý MicroBlaze

Các chức năng xử lý ngắt giúp quản lý việc xử lý ngắt trên các thiết bị xử lý MicroBlaze. Để sử dụng các chức năng này, bạn phải bao gồm tệp tiêu đề mb_interface.h trong mã nguồn của mình.

Tóm tắt chức năng xử lý ngắt của bộ xử lý MicroBlaze

Bảng sau đây cung cấp một bản tóm tắt về các chức năng có sẵn. Nhấp vào một liên kết để đi đến mô tả chức năng.

```
void microblaze_enable_interrupts (void)
void microblaze_disable_interrupts (void)
void microblaze_register_handler (Trình xử lý XInterruptHandler, void * DataPtr)
```

Mô tả chức năng xử lý ngắt của bộ xử lý MicroBlaze

void microblaze_enable_interrupts (void)

Bật ngắt trên bộ xử lý MicroBlaze. Khi bộ xử lý MicroBlaze khởi động, các ngắt sẽ bị vô hiệu hóa. Ngắt phải được bật rõ ràng bằng chức năng này.

void microblaze_disable_interrupts (void)

Tắt ngắt trên bộ xử lý MicroBlaze. Hàm này có thể được gọi khi nhập một đoạn mã quan trọng mà không mong muốn chuyển đổi ngữ cảnh.

**void microblaze_register_handler (XInterruptHandler
Xử lý, void * DataPtr)**

Đăng ký trình xử lý ngắt cho bộ xử lý MicroBlaze. Trình xử lý này được gọi lần lượt bởi trình xử lý ngắt cấp đầu tiên có mặt trong Độc lập.

Trình xử lý ngắt mức đầu tiên lưu và khôi phục các thanh ghi, khi cần thiết để xử lý ngắt, để chức năng bạn đăng ký với trình xử lý này có thể được dành riêng cho các khía cạnh khác của xử lý ngắt mà không cần lưu và khôi phục các thanh ghi.

Xử lý ngoại lệ của bộ xử lý MicroBlaze

Phần này mô tả chức năng xử lý ngoại lệ có sẵn trên bộ xử lý MicroBlaze. Tính năng này và các giao diện tương ứng không khả dụng trên các phiên bản của bộ xử lý MicroBlaze cũ hơn v3.00.a.

Lưu ý: Các chức năng này chỉ hoạt động chính xác khi các tham số xác định việc xử lý ngoại lệ phần cứng được định cấu hình thích hợp trong khái niệm MicroBlaze Microprocessor Hardware Specification (MHS). Ví dụ: bạn có thể đăng ký một trình xử lý để chia cho 0 ngoại lệ chỉ khi phần cứng chia cho 0 ngoại lệ được bật trên bộ xử lý MicroBlaze. Tham khảo Hướng dẫn Tham khảo Bộ xử lý MicroBlaze để biết thông tin về cách định cấu hình các thông số bộ đệm này. Bạn có thể tìm thấy liên kết đến tài liệu đó trong "[Tài nguyên bổ sung](#)", trang 1.

Tóm tắt chức năng xử lý ngoại lệ của bộ xử lý MicroBlaze

Sau đây là tóm tắt về các chức năng của Trình xử lý ngoại lệ cho bộ xử lý MicroBlaze. Để chuyển đến phần mô tả, hãy nhấp vào tên chức năng.

```
void microblaze_disable_exceptions (void)
void microblaze_enable_exceptions (void)
void microblaze_register_exception_handler (Xuint8 ExceptionId, XExceptionHandler Handler, void
* DataPtr)
```

Mô tả chức năng xử lý ngoại lệ của bộ xử lý MicroBlaze

Các chức năng sau đây giúp quản lý các ngoại lệ trên bộ xử lý MicroBlaze. Bạn phải bao gồm tệp tiêu đề `mb_interface.h` trong mã nguồn của mình để sử dụng các chức năng này.

`void microblaze_disable_exceptions (void)`

Tắt các ngoại lệ phần cứng khỏi bộ xử lý MicroBlaze. Quy trình này xóa bit "cho phép ngoại lệ" thích hợp trong thanh ghi dành riêng cho mô hình (MSR) của bộ xử lý.

`void microblaze_enable_exceptions (void)`

Bật ngoại lệ phần cứng từ bộ xử lý MicroBlaze. Quy trình này đặt bit "kích hoạt ngoại lệ" thích hợp trong MSR của bộ xử lý.

`void microblaze_register_exception_handler (Xuint8 ExceptionId, XExceptionHandler Handler, void * DataPtr)`

Đăng ký một trình xử lý cho loại ngoại lệ được chỉ định. Xử lý là hàm xử lý ngoại lệ được chỉ định. DataPtr là một giá trị dữ liệu gọi lại được chuyển đến trình xử lý ngoại lệ tại thời điểm chạy. Theo mặc định, ID ngoại lệ của ngoại lệ tương ứng được chuyển cho trình xử lý.

Bảng bên dưới mô tả các ID ngoại lệ hợp lệ, được xác định trong tệp `microblaze_exceptions_i.h`.

Bảng 1: ID ngoại lệ hợp lệ

ID ngoại lệ	Giá trị	Sự mô tả
XEXC_ID_FSL	0	Các ngoại lệ về xe buýt FSL.
XEXC_ID_UNALIGNED_ACCESS	1	Các ngoại lệ truy cập chưa định chỉ định.

Bảng 1: ID ngoại lệ hợp lệ (Tiếp theo)

ID ngoại lệ	Giá trị	Sự mô tả
XEXC_ID_<BUS>_EXCEPTION (1)	2	Ngoại lệ do hết thời gian chờ từ bus hệ thống bên Hư hỏng dẫn.
XEXC_ID_ILLEGAL_OPCODE	3	Ngoại lệ do cố gắng thực thi một mã opcode bất hợp pháp.
XEXC_ID_D <BUS>_EXCEPTION (1)	4	Ngoại lệ do hết thời gian chờ trên xe buýt hệ thống phía Dữ liệu.
XEXC_ID_DIV_BY_ZERO	5	Chia cho 0 ngoại lệ từ phép chia phần cứng.
XEXC_ID_FPU	6	Các ngoại lệ từ đơn vị dấu chấm động trên bộ xử lý MicroBlaze. Lưu ý: Ngoại lệ này chỉ có hiệu lực trên v4.00.a và các phiên bản mới hơn của bộ xử lý MicroBlaze.
XEXC_ID_MMU	7	Ngoại lệ từ MMU của bộ xử lý MicroBlaze. Tất cả các truy cập ngoại lệ MMU có thể có đều được lưu vào cùng một trình xử lý. Lưu ý: Ngoại lệ này chỉ có hiệu lực trên v7.00.a và các phiên bản mới hơn của bộ xử lý MicroBlaze.

Ghi chú:

1. BUS có thể là OPB hoặc PLB

Theo mặc định, Standalone cung cấp các trình xử lý trống, no-op cho tất cả các truy cập hợp ngoại lệ ngoại trừ các truy cập ngoại lệ không được đánh dấu. Trình xử lý ngoại lệ truy cập mặc định, nhanh, không dấu được cung cấp bởi Standalone.

Một ngoại lệ không được căn chỉnh có thể được xử lý bằng cách thực hiện quyền truy cập được căn chỉnh tự ứng vào các byte thích hợp trong bộ nhớ. Truy cập không được chỉ định được xử lý minh bạch bởi trình xử lý mặc định. Tuy nhiên, phần mềm tạo ra một lưỡng đáng kể các truy cập không được chỉ định sẽ thấy các tác động hiệu suất của việc này trong thời gian chạy. Điều này là do trình xử lý ngoại lệ phần mềm mất nhiều thời gian hơn để đáp ứng yêu cầu truy cập không được căn chỉnh so với truy cập được căn chỉnh.

Trong một số truy cập hợp, bạn có thể muốn sử dụng điều khoản cho các ngoại lệ không liên quan để chỉ bẫy ngoại lệ và để biết phần mềm nào đang gây ra ngoại lệ. Trong truy cập hợp này, bạn nên đặt các điểm ngắt ở trình xử lý ngoại lệ không dấu, để bẫy sự xuất hiện động của một ngoại lệ như vậy hoặc đăng ký trình xử lý tùy chỉnh của riêng bạn cho các ngoại lệ không dấu.

Lưu ý: Lớp xử lý ngoại lệ thấp nhất, luôn được cung cấp bởi Standalone, lưu trữ các thanh ghi để bay hơi và tạm thời trên ngăn xếp; do đó, trình xử lý tùy chỉnh của bạn cho các ngoại lệ phải xem xét rằng trình xử lý ngoại lệ cấp đầu tiên sẽ lưu một số trạng thái trên ngăn xếp, trước khi gọi trình xử lý của bạn.

Bộ xử lý MicroBlaze cho phép các ngoại lệ lồng nhau. Trình xử lý ngoại lệ, trong phần mở đầu của nó, cho phép lại các ngoại lệ. Do đó, các ngoại lệ trong các trình xử lý ngoại lệ được cho phép và xử lý.

Khi tham số predecode_fpu_exceptions đư ợc đặt thành true, nó khiến trình xử lý ngoại lệ cấp thấp:

- Giải mã lệnh dấu phẩy động lỗi
- Xác định các thanh ghi toán hạng
- Lưu trữ các giá trị của chúng thành hai biến toàn cục

Bạn có thể đăng ký một trình xử lý cho các ngoại lệ dấu phẩy động và truy xuất giá trị của các toán hạng từ các biến toàn cục. Bạn có thể sử dụng macro `microblaze_getfpex_operand_a()` và `microblaze_getfpex_operand_b()`.

Lưu ý: Các macro này trả về giá trị toán hạng của ngoại lệ dấu chấm động (FP) cuối cùng. Nếu có các ngoại lệ lồng nhau, bạn không thể truy xuất giá trị của các ngoại lệ bên ngoài. Một lệnh FP có thể có một trong các thanh ghi nguồn giống với toán hạng đích. Trong trường hợp này, lệnh lỗi sẽ ghi đè giá trị toán hạng đầu vào và nó không thể khôi phục lại đư ợc.

Xử lý bộ nhớ đệm hư hỏng dẫn bộ xử lý MicroBlaze

Các chức năng sau đây giúp quản lý bộ nhớ đệm hư hỏng dẫn trên bộ xử lý MicroBlaze. Bạn phải bao gồm tệp tiêu đề `mb_interface.h` trong mã nguồn của mình để sử dụng các chức năng này.

Lưu ý: Các chức năng này chỉ hoạt động chính xác khi các tham số xác định hệ thống bộ nhớ đệm đư ợc định cấu hình thích hợp trong khái phần cứng MicroBlaze Microprocessor Hardware Specification (MHS). Tham khảo Hư hỏng dẫn Tham khảo MicroBlaze để biết thông tin về cách định cấu hình các thông số bộ đệm này. ["Tài nguyên bổ sung", trang 1](#) chứa một liên kết đến tài liệu này.

Tóm tắt chức năng xử lý bộ nhớ đệm hư hỏng dẫn bộ xử lý MicroBlaze

Sau đây là các liên kết đến các mô tả chức năng. Bấm vào tên để chuyển đến chức năng đó.

```
void microblaze_enable_icache (void)
void microblaze_disable_icache (void)
void microblaze_invalidate_icache ()
void microblaze_invalidate_icache_range (unsigned int cache_addr, unsigned int cache_size)
```

Mô tả chức năng xử lý bộ nhớ cache của bộ xử lý MicroBlaze

`void microblaze_enable_icache (void)`

Kích hoạt bộ đệm chỉ dẫn trên bộ xử lý MicroBlaze. Khi bộ xử lý MicroBlaze khởi động, bộ đệm lệnh bị vô hiệu hóa. Bộ đệm chỉ dẫn phải đư ợc bật rõ ràng bằng chức năng này.

`void microblaze_disable_icache (void)`

Tắt bộ đệm lệnh trên bộ xử lý MicroBlaze.

`void microblaze_invalidate_icache ()`

Vô hiệu hóa icache hư hỏng dẫn.

Lưu ý: Đối với bộ xử lý MicroBlaze trước phiên bản v7.20.a:

Bộ đệm và ngắt bị vô hiệu hóa trước khi bắt đầu hết hiệu lực và đư ợc khôi phục về trạng thái trước đó của chúng sau khi hết hiệu lực.

```
void microblaze_invalidate_icache_range ( int không dấu
                                         cache_addr, unsigned int cache_size)
```

Làm mất hiệu lực phạm vi được chỉ định trong icache hứa ứng dãy. Hàm này có thể được sử dụng để làm mất hiệu lực của tất cả hoặc một phần của icache hứa ứng dãy. Tham số cache_addr cho biết vị trí bắt đầu của bộ nhớ cache bị vô hiệu. Cache_size đại diện cho số byte từ cache_addr để làm mất hiệu lực. Lưu ý rằng các dòng bộ đệm bị vô hiệu bắt đầu từ dòng bộ đệm mà cache_addr thuộc về và kết thúc ở dòng bộ đệm chứa địa chỉ (cache_addr + cache_size - 1).

Ví dụ: microblaze_invalidate_icache_range (0x00000300, 0x100) làm mất hiệu lực vùng bộ đệm chỉ lệnh từ 0x300 đến 0x3ff (0x100 byte bộ nhớ đệm bị xóa bắt đầu từ 0x300).

Lưu ý: Đối với bộ xử lý MicroBlaze trư ớc phiên bản v7.20.a:

Bộ đệm và ngắn bị vô hiệu hóa trư ớc khi bắt đầu hết hiệu lực và được khôi phục về trạng thái trư ớc đó của chúng sau khi hết hiệu lực.

Xử lý bộ nhớ đệm dữ liệu của bộ xử lý MicroBlaze

Các chức năng sau đây giúp quản lý bộ nhớ đệm dữ liệu trên bộ xử lý MicroBlaze. Bạn phải bao gồm tệp tiêu đề mb_interface.h trong mã nguồn của mình để sử dụng các chức năng này.

Lưu ý: Các chức năng này chỉ hoạt động chính xác khi các tham số xác định hệ thống bộ nhớ đệm được cấu hình thích hợp trong khái phần cứng MicroBlaze MHS. Tham khảo Hứa ứng dãy Tham khảo Bộ xử lý MicroBlaze để biết thông tin về cách định cấu hình các thông số bộ đệm này. ["Tài nguyên bổ sung"](#), [trang 1](#) chứa một liên kết đến tài liệu này.

Tóm tắt chức năng xử lý bộ nhớ đệm dữ liệu của bộ xử lý MicroBlaze

Sau đây là các liên kết đến các mô tả chức năng. Bấm vào tên để chuyển đến chức năng đó.

```
void microblaze_enable_dcache (void)
void microblaze_disable_dcache (void)
void microblaze_flush_dcache ()
void microblaze_flush_dcache_range (unsigned int cache_addr, unsigned int cache_len)
void microblaze_invalidate_dcache ()
void microblaze_invalidate_dcache_range (unsigned int cache_addr, unsigned int cache_size)
```

Chức năng xử lý bộ nhớ đệm dữ liệu

`void microblaze_enable_dcache (void)`

Bật bộ đệm dữ liệu trên bộ xử lý MicroBlaze. Khi bộ xử lý MicroBlaze khởi động, bộ đệm dữ liệu sẽ bị vô hiệu hóa. Bộ nhớ cache dữ liệu phải được bật rõ ràng bằng chức năng này.

`void microblaze_disable_dcache (void)`

Tắt bộ đệm dữ liệu trên bộ xử lý MicroBlaze. Nếu bộ nhớ đệm ghi lại được bật trong phần cứng bộ xử lý MicroBlaze, chức năng này cũng sẽ chuyển dữ liệu bẩn trong bộ đệm trở lại bộ nhớ ngoài và làm mất hiệu lực bộ đệm. Để ghi qua bộ đệm, chức năng này không thực hiện thêm bất kỳ xử lý nào ngoài việc tắt bộ đệm.

`void microblaze_flush_dcache ()`

Xóa toàn bộ bộ nhớ cache dữ liệu. Chức năng này có thể được sử dụng khi bộ nhớ đệm ghi lại được bật trong phần cứng bộ xử lý MicroBlaze. Việc thực thi chức năng này đảm bảo rằng dữ liệu bẩn trong bộ đệm sẽ được ghi trở lại bộ nhớ ngoài và nội dung sẽ bị vô hiệu.

- Bộ nhớ đệm bị tắt trước khi quá trình xả bắt đầu và được khôi phục về trạng thái trước đó sau khi xả là hoàn tất.
- Ngắt bị vô hiệu hóa trong khi bộ đệm được xóa và khôi phục về trạng thái sau khi xả xong.

`void microblaze_flush_dcache_range (int không dấu
cache_addr, unsigned int cache_len)`

Xóa phạm vi bộ nhớ cache dữ liệu được chỉ định. Chức năng này có thể được sử dụng khi bộ nhớ đệm ghi lại được bật trong phần cứng bộ xử lý MicroBlaze. Việc thực thi chức năng này đảm bảo rằng dữ liệu bẩn trong phạm vi bộ nhớ cache được ghi trở lại bộ nhớ ngoài và nội dung của phạm vi bộ nhớ cache bị vô hiệu. Lưu ý rằng các dòng cache sẽ được xóa bắt đầu từ dòng cache mà `cache_addr` thuộc về và kết thúc tại dòng cache chứa địa chỉ (`cache_addr + cache_size - 1`).

Ví dụ: `microblaze_flush_dcache_range (0x00000300, 0x100)` xóa vùng bộ nhớ đệm dữ liệu từ `0x300` đến `0x3ff` (`0x100` byte bộ nhớ đệm được xóa bắt đầu từ `0x300`).

`void microblaze_invalidate_dcache ()`

Làm mất hiệu lực bộ đệm dữ liệu hưu ơng dãnh.

Lưu ý: Đối với bộ xử lý MicroBlaze trước phiên bản v7.20.a:

Bộ đệm và ngắt bị vô hiệu hóa trước khi bắt đầu hết hiệu lực và được khôi phục về trạng thái trước đó của chúng sau khi hết hiệu lực.

```
void microblaze_invalidate_dcache_range ( int không dấu
                                         cache_addr, unsigned int cache_size)
```

Làm mất hiệu lực bộ nhớ cache dữ liệu. Chức năng này có thể được sử dụng để làm mất hiệu lực của tất cả hoặc một phần của bộ đệm dữ liệu. Tham số cache_addr cho biết vị trí bắt đầu của bộ đệm và cache_size biểu thị kích thước từ cache_addr đến vô hiệu.

Lưu ý rằng các dòng bộ đệm sẽ bị vô hiệu hóa bắt đầu từ dòng bộ đệm mà cache_addr đến thuộc và kết thúc tại dòng cache chứa địa chỉ (cache_addr + cache_size - 1).

Lưu ý: Đối với bộ xử lý MicroBlaze trư ớc phiên bản v7.20.a:

Bộ đệm và ngắt bị vô hiệu hóa trư ớc khi bắt đầu hết hiệu lực và được khôi phục về trạng thái trư ớc đó của chúng sau khi hết hiệu lực.

Ví dụ: `microblaze_invalidate_dcache_range (0x000000300, 0x100)` làm mất hiệu lực vùng bộ đệm dữ liệu từ 0x300 đến 0x3ff (0x100 byte bộ nhớ đệm bị xóa bắt đầu từ 0x300).

Trình tự phần mềm để khởi tạo lệnh và bộ nhớ cache dữ liệu

Thông thường, trư ớc khi sử dụng bộ đệm, chương trình của bạn phải thực hiện một chuỗi hoạt động cụ thể của bộ đệm để đảm bảo rằng dữ liệu không hợp lệ / bẩn trong bộ đệm không được bộ xử lý sử dụng. Điều này thường xảy ra trong quá trình tải xuống và thực thi chương trình lặp đi lặp lại.

Các đoạn mã ví dụ sau đây cho thấy trình tự phần mềm cần thiết để khởi tạo hưng dẫn và bộ nhớ đệm dữ liệu trong chương trình của bạn.

```
/ * Khởi tạo ICache * /
microblaze_invalidate_icache ();
microblaze_enable_icache ();

/ * Khởi tạo DCache * /
microblaze_invalidate_dcache ();
microblaze_enable_dcache ();

Vào cuối chương trình của bạn, bạn cũng nên đặt một trình tự tương tự như đoạn mã ví dụ bên dưới.
Điều này đảm bảo rằng bộ nhớ cache và bộ nhớ ngoài được duy trì ở trạng thái hợp lệ và sạch sẽ.

/ * Dọn dẹp DCache. Đối với bộ nhớ đệm ghi lại, quy trình disable_dcache
nội bộ không tuân ra và làm mất hiệu lực. Để ghi qua bộ nhớ đệm,
một sự vô hiệu rõ ràng phải được thực hiện trên toàn bộ bộ nhớ cache. * /

#ifndef XPAR_MICROBLAZE_DCACHE_USE_WRITEBACK == 0
microblaze_invalidate_dcache ();
#endif

microblaze_disable_dcache ();

/ * Dọn dẹp ICache * /
microblaze_invalidate_icache ();
microblaze_disable_icache ();
```

Macro giao diện Fast Simplex Link (FSL) của bộ xử lý MicroBlaze

Độc lập bao gồm các macro để cung cấp khả năng truy cập thuận tiện vào các trình tăng tốc đư ợc kết nối với Giao diện MicroBlaze Fast Simplex Link (FSL).

Tóm tắt Macro giao diện Fast Simplex Link (FSL) của Bộ xử lý MicroBlaze

Sau đây là danh sách các macro có sẵn. Bạn có thể nhấp vào tên macro trong bảng để chuyển đến mô tả của macro đang hoạt động.

<code>getfslx (val, id, flags)</code>	<code>putfslx (val, id, flags)</code>
<code>putfslx (val, id, flags)</code>	<code>tgetfslx (val, id, flags)</code>
<code>tgetfslx (val, id, flags)</code>	<code>tputfslx (val, id, flags)</code>
<code>tputfslx (val, id, flags)</code>	<code>fsl_isinvalid (không hợp lệ)</code>
<code>getd_fslx (val, id, flags)</code>	<code>fsl_iserror (lỗi)</code>

Mô tả Macro FSL của Bộ xử lý MicroBlaze

Các macro sau cung cấp quyền truy cập vào tất cả các chức năng của tính năng MicroBlaze FSL trong một giao diện đơn giản và đư ợc tham số hóa. Một số khả năng chỉ có sẵn trên MicroBlaze v7.00.a và mới hơn, như đã nêu trong phần mô tả.

Trong mô tả macro, val đe cập đến một biến trong chương trình của bạn có thể là nguồn hoặc phần chìm của hoạt động FSL.

Lưu ý: id phải là một chữ số nguyên trong các phiên bản cơ bản của macro (getfslx, putfslx, tgetfslx, tputfslx) và có thể là một chữ số nguyên hoặc một biến số nguyên trong các phiên bản động của macro (getdfslx, putdfslx, tgetdfslx, tputdfslx.)

Bạn phải bao gồm fsl.h trong các tệp nguồn của mình để cung cấp các macro này.

`getfslx (val, id, flags)`

Thực hiện chức năng lấy trên FSL đầu vào của bộ xử lý MicroBlaze; id là mã định danh FSL và là một ký tự trong phạm vi từ 0 đến 7 (0 đến 15 cho MicroBlaze v7.00.a trở lên). Ngữ nghĩa của lệnh đư ợc xác định bởi các cờ macro FSL hợp lệ, đư ợc liệt kê trong [Bảng 2, trang 11](#).

`putfslx (val, id, flags)`

Thực hiện chức năng đặt trên FSL đầu vào của bộ xử lý MicroBlaze; id là mã định danh FSL và là một ký tự trong phạm vi từ 0 đến 7 (0 đến 15 cho bộ xử lý MicroBlaze v7.00.a trở lên). Ngữ nghĩa của lệnh đư ợc xác định bởi các cờ macro FSL hợp lệ, đư ợc liệt kê trong [Bảng 2, trang 11](#).

`tgetfslx (val, id, flags)`

Thực hiện chức năng kiểm tra trên FSL đầu vào của bộ xử lý MicroBlaze; id là mã định danh FSL và là một chữ trong phạm vi từ 0 đến 7 (0 đến 15 cho MicroBlaze v7.00.a trở lên). Macro này có thể đư ợc sử dụng để kiểm tra việc ghi một giá trị vào FSL. Ngữ nghĩa của lệnh đư ợc xác định bởi các cờ macro FSL hợp lệ, đư ợc liệt kê trong [Bảng 2, trang 11](#).

tputfslx (val, id, flags)

Thực hiện chức năng đặt trên FSL đầu vào của bộ xử lý MicroBlaze; id là mã định danh FSL và là một ký tự trong phạm vi từ 0 đến 7 (0 đến 15 cho bộ xử lý MicroBlaze v7.00.a trở lên). Macro này có thể được sử dụng để kiểm tra việc nhận một giá trị đơn lẻ từ FSL. Ngữ nghĩa của lệnh put được xác định bởi các cờ macro FSL hợp lệ, được liệt kê trong [Bảng 2, trang 11](#).

getd fslx (val, id, flags)

Thực hiện chức năng lấy trên FSL đầu vào của bộ xử lý MicroBlaze; id là mã định danh FSL và là giá trị số nguyên hoặc biến trong phạm vi từ 0 đến 15. Ngữ nghĩa của lệnh được xác định bởi các cờ macro FSL hợp lệ, được liệt kê trong [Bảng 2, trang 11](#). Macro này có sẵn trên MicroBlaze chỉ bộ xử lý v7.00.a và mới hơn.

putdfslx (val, id, flags)

Thực hiện chức năng đặt trên FSL đầu vào của bộ xử lý MicroBlaze; id là mã định danh FSL và là giá trị số nguyên hoặc biến trong phạm vi từ 0 đến 15. Ngữ nghĩa của lệnh được xác định bởi các cờ macro FSL hợp lệ, được liệt kê trong [Bảng 2, trang 11](#). Macro này có sẵn trên MicroBlaze chỉ bộ xử lý v7.00.a và mới hơn.

tgetdfslx (val, id, flags)

Thực hiện chức năng lấy kiểm tra trên FSL đầu vào của bộ xử lý MicroBlaze; id là mã định danh FSL và là một số nguyên hoặc biến trong phạm vi từ 0 đến 15. Macro này có thể được sử dụng để kiểm tra việc ghi một giá trị duy nhất vào FSL. Ngữ nghĩa của lệnh được xác định bởi các cờ macro FSL hợp lệ, được liệt kê trong [Bảng 2](#). Macro này chỉ có sẵn trên bộ xử lý MicroBlaze v7.00.a và mới hơn.

tputdfslx (val, id, flags)

Thực hiện chức năng đặt trên FSL đầu vào của bộ xử lý MicroBlaze; id là mã định danh FSL và là một số nguyên hoặc biến trong phạm vi từ 0 đến 15. Macro này có thể được sử dụng để kiểm tra việc nhận một giá trị duy nhất từ FSL. Ngữ nghĩa của lệnh được xác định bởi các cờ macro FSL hợp lệ, được liệt kê trong [Bảng 2](#). Macro này chỉ khả dụng trên bộ xử lý MicroBlaze v7.00.a trở lên.

fsl_isinvalid (không hợp lệ)

Kiểm tra xem hoạt động FSL cuối cùng có trả lại dữ liệu hợp lệ hay không. Macro này có thể áp dụng sau khi gọi lệnh đặt hoặc nhận FSL không chặn. Nếu không có dữ liệu trên kênh FSL khi đặt hoặc nếu kênh FSL đã đầy khi đặt thì không hợp lệ được đặt thành 1; nếu không, nó được đặt thành 0.

fsl_iserror (lỗi)

Macro này được sử dụng để kiểm tra xem hoạt động FSL cuối cùng có đặt cờ lỗi hay không. Macro này có thể áp dụng sau khi gọi một FSL điều khiển đặt hoặc nhận lệnh. Nếu bit điều khiển được thiết lập, lỗi được đặt thành 1; nếu không, nó được đặt thành 0.

Cờ Macro FSL của Bộ xử lý MicroBlaze

Bảng sau liệt kê các cờ Macro FSL có sẵn.

Bảng 2: Cờ Macro FSL

Lá cờ	Sự mô tả
FSL_DEFAULT	Chặn ngữ nghĩa (trên bộ xử lý MicroBlaze v7.00.a trở lên, chế độ này có thể ngắt đư ợc).
FSL_NONBLOCKING (2)	Ngữ nghĩa không chặn.
FSL_EXCEPTION (1)	Tạo ngoại lệ về sự không khớp bit điều khiển.
FSL_CONTROL	Kiểm soát ngữ nghĩa.
FSL_ATOMIC (1)	Ngữ nghĩa nguyên tử. Một chuỗi hướng dẫn FSL không thể bị gián đoạn.
FSL_NONBLOCKING_EXCEPTION (1)	Kết hợp ngữ nghĩa không chặn và ngoại lệ.
FSL_NONBLOCKING_CONTROL	Kết hợp ngữ nghĩa không chặn và kiểm soát.
FSL_NONBLOCKING_ATOMIC (1)	Kết hợp ngữ nghĩa không chặn và nguyên tử.
FSL_EXCEPTION_CONTROL (1)	Kết hợp ngữ nghĩa ngoại lệ và điều khiển.
FSL_EXCEPTION_ATOMIC (1)	Kết hợp ngoại lệ và ngữ nghĩa nguyên tử.
FSL_CONTROL_ATOMIC (1)	Kết hợp điều khiển và ngữ nghĩa nguyên tử.
FSL_NONBLOCKING_EXCEPTION_CONTROL (1)	Kết hợp ngữ nghĩa không chặn, ngoại lệ và điều khiển.
FSL_NONBLOCKING_EXCEPTION_ATOMIC (1)	Kết hợp ngữ nghĩa không chặn, ngoại lệ và nguyên tử.
FSL_NONBLOCKING_CONTROL_ATOMIC (1)	Kết hợp ngữ nghĩa không chặn, nguyên tử và điều khiển.
FSL_EXCEPTION_CONTROL_ATOMIC (1)	Kết hợp ngữ nghĩa ngoại lệ, nguyên tử và điều khiển.
FSL_NONBLOCKING_EXCEPTION_CONTROL_ATOMIC (1)	Kết hợp ngữ nghĩa không chặn, ngoại lệ, kiểm soát và nguyên tử.

Ghi chú:

1. Tổ hợp cờ này chỉ có sẵn trên bộ xử lý MicroBlaze v7.00.a và các phiên bản mới hơn.
2. Khi ngữ nghĩa không chặn không đư ợc áp dụng, ngữ nghĩa chặn đư ợc ngụ ý.

Macro Fast Simplex Link (FSL) của Bộ xử lý Microblaze không được dùng nữa

Các macro sau không được dùng nữa:

getfsl (val, id) (không dùng nữa)

Thực hiện chức năng lấy dữ liệu chặn trên FSL đầu vào của bộ xử lý MicroBlaze; id là mã định danh FSL trong phạm vi từ 0 đến 7. Macro này là liên tục.

putfsl (val, id) (không dùng nữa)

Thực hiện chức năng đặt dữ liệu chặn trên FSL đầu ra của bộ xử lý MicroBlaze; id là mã định danh FSL trong phạm vi từ 0 đến 7. Macro này là liên tục.

ngetfsl (val, id) (không dùng nữa)

Thực hiện chức năng lấy dữ liệu không chặn trên FSL đầu vào của bộ xử lý MicroBlaze; id là số nhận dạng FSL trong phạm vi từ 0 đến 7.

nputfsl (val, id) (không dùng nữa)

Thực hiện chức năng đưa dữ liệu không chặn vào FSL đầu ra của bộ xử lý MicroBlaze; id là số nhận dạng FSL trong phạm vi từ 0 đến 7.

cgetfsl (val, id) (không dùng nữa)

Thực hiện chức năng lấy điều khiển chặn trên FSL đầu vào của bộ xử lý MicroBlaze; id là mã định danh FSL trong phạm vi từ 0 đến 7. Macro này là liên tục.

cputfsl (val, id) (không dùng nữa)

Thực hiện chức năng đặt kiểm soát chặn trên FSL đầu ra của bộ xử lý MicroBlaze; id là mã định danh FSL trong phạm vi từ 0 đến 7. Macro này là liên tục.

ncgetfsl (val, id) (không dùng nữa)

Thực hiện chức năng nhận điều khiển không chặn trên FSL đầu vào của bộ xử lý MicroBlaze; id là số nhận dạng FSL trong phạm vi từ 0 đến 7.

ncputfsl (val, id) (không dùng nữa)

Thực hiện chức năng đặt điều khiển không chặn trên FSL đầu ra của bộ xử lý MicroBlaze; Tôi là mã định danh FSL trong phạm vi từ 0 đến 7.

getfsl_interruptible (val, id) (không dùng nữa)

Thực hiện lặp lại các hoạt động lấy dữ liệu không chặn trên FSL đầu vào của bộ xử lý MicroBlaze cho đến khi dữ liệu hợp lệ thực sự được tìm nạp; id là số nhận dạng FSL trong phạm vi từ 0 đến 7. Vì quyền truy cập FSL là không bị chặn nên bộ xử lý sẽ xử lý các ngắt.

putfsl_interruptible (val, id) (không dùng nữa)

Thực hiện lặp lại các hoạt động đặt dữ liệu không chặn trên FSL đầu ra của bộ xử lý MicroBlaze cho đến khi dữ liệu hợp lệ được gửi đi; id là mã định danh FSL trong phạm vi từ 0 đến 7. Bởi vì truy cập FSL là không chặn, các ngắt sẽ được xử lý bởi bộ xử lý.

cgetfsl_interruptible (val, id) (không được dùng nữa)

Thực hiện lặp đi lặp lại kiểm soát không chặn các hoạt động trên FSL đầu vào của bộ xử lý MicroBlaze cho đến khi dữ liệu hợp lệ thực sự được tìm nạp; id là số nhận dạng FSL trong phạm vi từ 0 đến 7. Vì quyền truy cập FSL là không bị chặn, nên bộ xử lý sẽ xử lý các ngắt.

cputfsl_interruptible (val, id) (không dùng nữa)

Thực hiện lặp đi lặp lại các hoạt động kiểm soát không chặn trên FSL đầu ra của bộ xử lý MicroBlaze cho đến khi dữ liệu hợp lệ được gửi đi; id là mã định danh FSL trong phạm vi từ 0 đến 7. Bởi vì truy cập FSL là không chặn, các ngắt được phục vụ bởi bộ xử lý.

Vì xử lý MicroBlaze Macro giả asm

Độc lập bao gồm các macro để cung cấp khả năng truy cập thuận tiện vào các thanh ghi khác nhau trong bộ xử lý MicroBlaze. Một số macro này rất hữu ích trong các trình xử lý ngoại lệ để truy xuất thông tin về ngoại lệ. Để sử dụng các macro này, bạn phải bao gồm tệp tiêu đề `mb_interface.h` trong mã nguồn của mình.

Tóm tắt Macro Pseudo-asm của Bộ xử lý MicroBlaze

Sau đây là tóm tắt về macro giả asm của bộ xử lý MicroBlaze. Nhấp vào tên macro để chuyển đến mô tả.

```
mfgpr (xn)
mfmsr ()
mfesr ()
mfear ()
mffsr ()
mtmsr (v)
mtgpr (xn, v)
microblaze_getfpex_operand_a ()
microblaze_getfpex_operand_b ()
```

Mô tả Macro Pseudo-asm của Bộ xử lý MicroBlaze

mfgpr (rn)

Giá trị trả về từ thanh ghi mục đích chung (GPR) rn.

mfmsr ()

Trả về giá trị hiện tại của MSR.

mfesr ()

Trả về giá trị hiện tại của Thanh ghi Trạng thái Ngoại lệ (ESR).

mfear ()

Trả về giá trị hiện tại của Thanh ghi Địa chỉ Ngoại lệ (EAR).

mffsr ()

Trả về giá trị hiện tại của Trạng thái dấu chấm động (FPS).

mtmsr (v)

Đi chuyển giá trị v sang MSR.

mtgpr (rn, v)

Đi chuyển giá trị v sang GPR rn.

microblaze_getfpex_operand_a ()

Trả về giá trị đã lưu của toán hạng A của lệnh dấu chấm động lõi cuối cùng.

microblaze_getfpex_operand_b ()

Trả về giá trị đã lưu của toán hạng B của lệnh dấu chấm động lõi cuối cùng.

Lưu ý: Do cách viết một số macro này, chúng không thể được sử dụng làm tham số để gọi hàm và các cấu trúc khác như vậy.

Quy trình Truy cập Đăng ký Phiên bản Bộ xử lý MicroBlaze (PVR) và Macro

Bộ xử lý MicroBlaze v5.00.a và các phiên bản mới hơn có Thanh ghi phiên bản bộ xử lý (PVR) có thể định cấu hình. Nội dung của PVR được ghi lại bằng cách sử dụng cấu trúc dữ liệu pvr_t , được định nghĩa là một mảng các từ 32 bit, với mỗi từ tương ứng với một thanh ghi PVR trên phần cứng. Số lượng từ PVR được xác định bởi số lượng PVR được cấu hình trong phần cứng. Bạn không nên cố gắng truy cập các thanh ghi PVR không có trong phần cứng, vì cấu trúc dữ liệu pvr_t được thay đổi kích thước để chỉ chứa nhiều PVR như hiện có trong phần cứng.

Để truy cập thông tin trong PVR:

1. Sử dụng hàm microblaze_get_pvr () để điền dữ liệu PVR vào dữ liệu pvr_t kết cấu.
2. Trong các bước tiếp theo, bạn có thể sử dụng bất kỳ một trong danh sách macro truy cập PVR để nhận từng dữ liệu được lưu trữ trong PVR.

Lưu ý: Các macro truy cập PVR nhận một tham số, tham số này phải thuộc loại pvr_t.

Quy trình truy cập PVR

Quy trình sau đây được sử dụng để truy cập PVR. Bạn phải bao gồm tệp pvr.h để làm cho quy trình này khả dụng.

```
int_microblaze_get_pvr (pvr_t * pvr)
```

Điền cấu trúc dữ liệu PVR mà pvr trả đến với các giá trị của thanh ghi PVR phần cứng. Quy trình này chỉ điền vào số lượng PVR hiện có trong phần cứng và phần còn lại là 0. Quy trình này không khả dụng nếu C_PVR được đặt thành NONE trong phần cứng.

Macro PVR

Các macro bộ xử lý sau được sử dụng để truy cập PVR. Bạn phải bao gồm tệp pvr.h để cung cấp các macro này.

Bảng sau liệt kê các macro và mô tả PVR của bộ xử lý MicroBlaze.

Bảng 3: Macro truy cập PVR

Macro	Sự mô tả
MICROBLAZE_PVR_IS_FULL (pvr)	Trả về số nguyên khác 0 nếu PVR thuộc loại FULL, 0 nếu cơ bản.
MICROBLAZE_PVR_USE_BARREL (pvr)	Trả về số nguyên khác 0 nếu có bộ dịch chuyển thẳng phần cứng.
MICROBLAZE_PVR_USE_DIV (pvr)	Trả về số nguyên khác 0 nếu có bộ chia phần cứng.
MICROBLAZE_PVR_USE_HW_MUL (pvr)	Trả về số nguyên khác 0 nếu có hệ số phần cứng.
MICROBLAZE_PVR_USE_FPU (pvr)	Trả về số nguyên khác 0 nếu có đơn vị dấu phẩy động phần cứng (FPU).
MICROBLAZE_PVR_USE_FPU2 (pvr)	Trả về số nguyên khác 0 nếu có hư hỏng dẫn chuyển đổi dấu phẩy động phần cứng và căn bậc hai.
MICROBLAZE_PVR_USE_ICACHE (pvr)	Trả về số nguyên khác 0 nếu có I-cache.
MICROBLAZE_PVR_USE_DCACHE (pvr)	Trả về số nguyên khác 0 nếu có D-cache.
MICROBLAZE_PVR_MICROBLAZE_VERSION (pvr)	Trả về mã hóa phiên bản bộ xử lý MicroBlaze. Tham khảo Hình ảnh Tham khảo Bộ xử lý MicroBlaze để biết ánh xạ từ mã hóa sang phiên bản phần cứng thực tế. "Tài nguyên bổ sung" , trang 1 chứa một liên kết đến tài liệu này.
MICROBLAZE_PVR_USER1 (pvr)	Trả lại truthorng USER1 được lưu trữ trong PVR.
MICROBLAZE_PVR_USER2 (pvr)	Trả lại truthorng USER2 được lưu trữ trong PVR.

Bảng 3: Macro truy cập PVR (Tiếp theo)

Macro	Sự mô tả
MICROBLAZE_PVR_INTERCONNECT (pvr)	Trả về khác 0 nếu bộ xử lý MicroBlaze có kết nối PLB; nếu không thì trả về số không.
MICROBLAZE_PVR_D_PLB (pvr)	Trả về số nguyên khác 0 nếu có giao diện Data Side PLB.
MICROBLAZE_PVR_D_OPB (pvr)	Trả về số nguyên khác 0 nếu có giao diện Bus ngoại vi trên chip (OPB) bên dữ liệu.
MICROBLAZE_PVR_D_LMB (pvr)	Trả về số nguyên khác 0 nếu có giao diện Bus bộ nhớ cục bộ phía dữ liệu (LMB).
MICROBLAZE_PVR_I_PLB (pvr)	Trả về số nguyên khác 0 nếu có giao diện PLB bên hư ứng dẫn.
MICROBLAZE_PVR_I_OPB (pvr)	Trả về số nguyên khác 0 nếu có giao diện OPB bên hư ứng dẫn.
MICROBLAZE_PVR_I_LMB (pvr)	Trả về số nguyên khác 0 nếu có giao diện LMB bên hư ứng dẫn.
MICROBLAZE_PVR_INTERRUPT_IS_EDGE (pvr)	Trả về số nguyên khác 0 nếu ngắt đư ợc định cấu hình là kích hoạt cạnh.
MICROBLAZE_PVR_EDGE_IS_POSITIVE (pvr)	Trả về số nguyên khác 0 nếu ngắt đư ợc định cấu hình là kích hoạt dư ơng.
MICROBLAZE_PVR_USE_MUL64 (pvr)	Trả về số nguyên khác 0 nếu bộ xử lý MicroBlaze hỗ trợ các sản phẩm 64-bit cho phép nhân.
MICROBLAZE_PVR_OPCODE_0x0_ILLEGAL (pvr)	Trả về số nguyên khác 0 nếu opcode 0x0 đư ợc coi là opcode bất hợp pháp.
MICROBLAZE_PVR_UNALIGNED_EXCEPTION (pvr)	Trả về số nguyên khác 0 nếu các ngoại lệ không dấu đư ợc hỗ trợ.
MICROBLAZE_PVR_ILL_OPCODE_EXCEPTION (tr vr)	Trả về số nguyên khác 0 nếu các ngoại lệ opcode bất hợp pháp đư ợc hỗ trợ.
MICROBLAZE_PVR_IOPB_EXCEPTION (pvr)	Trả về số nguyên khác 0 nếu các ngoại lệ I-OPB đư ợc hỗ trợ.
MICROBLAZE_PVR_DOPB_EXCEPTION (pvr)	Trả về số nguyên khác 0 nếu các ngoại lệ D-OPB đư ợc hỗ trợ.
MICROBLAZE_PVR_IPLB_EXCEPTION (pvr)	Trả về số nguyên khác 0 nếu các ngoại lệ I-PLB đư ợc hỗ trợ.
MICROBLAZE_PVR_DPLB_EXCEPTION (pvr)	Trả về số nguyên khác 0 nếu các ngoại lệ D-PLB đư ợc hỗ trợ.
MICROBLAZE_PVR_DIV_ZERO_EXCEPTION (pvr)	Trả về số nguyên khác 0 nếu phép chia cho 0 ngoại lệ đư ợc hỗ trợ.
MICROBLAZE_PVR_FPU_EXCEPTION (pvr)	Trả về số nguyên khác 0 nếu các ngoại lệ FPU đư ợc hỗ trợ.
MICROBLAZE_PVR_FSL_EXCEPTION (pvr)	Trả về số nguyên khác 0 nếu có ngoại lệ FSL.
MICROBLAZE_PVR_DEBUG_ENABLED (pvr)	Trả về số nguyên khác 0 nếu gõ lỗi đư ợc bật.
MICROBLAZE_PVR_NUM_PC_BRK (pvr)	Trả lại số lư ợng điểm ngắt PC phần cứng có sẵn.
MICROBLAZE_PVR_NUM_RD_ADDR_BRK (pvr)	Trả về số lư ợng điểm xem phần cứng địa chỉ đọc đư ợc hỗ trợ.
MICROBLAZE_PVR_NUM_WR_ADDR_BRK (pvr)	Trả về số lư ợng điểm giám sát phần cứng địa chỉ ghi đư ợc hỗ trợ.
MICROBLAZE_PVR_FSL_LINKS (pvr)	Trả lại số lư ợng liên kết FSL hiện có.
MICROBLAZE_PVR_ICACHE_BASEADDR (pvr)	Trả lại địa chỉ cơ sở của I-cache.
MICROBLAZE_PVR_ICACHE_HIGHADDR (pvr)	Trả lại địa chỉ cao của I-cache.
MICROBLAZE_PVR_ICACHE_ADDR_TAG_BITS (tr vr)	Trả về số bit thẻ địa chỉ cho I-cache.
MICROBLAZE_PVR_ICACHE_USE_FSL (pvr)	Trả về khác 0 nếu I-cache sử dụng liên kết FSL.
MICROBLAZE_PVR_ICACHE_ALLOW_WR (pvr)	Trả về khác 0 nếu cho phép ghi vào I-cache.

Bảng 3: Macro truy cập PVR (Tiếp theo)

Macro	Sự mô tả
MICROBLAZE_PVR_ICACHE_LINE_LEN (pvr)	Trả về độ dài của mỗi dòng I-cache tính bằng byte.
MICROBLAZE_PVR_ICACHE_BYTE_SIZE (pvr)	Trả về kích thước của D-cache tính bằng byte.
MICROBLAZE_PVR_DCACHE_BASEADDR (pvr)	Trả lại địa chỉ cơ sở của D-cache.
MICROBLAZE_PVR_DCACHE_HIGHADDR (pvr)	Trả lại địa chỉ cao của D-cache.
MICROBLAZE_PVR_DCACHE_ADDR_TAG_BITS (trvr)	Trả về số bit thẻ địa chỉ cho D-cache.
MICROBLAZE_PVR_DCACHE_USE_FSL (pvr)	Trả về khác 0 nếu D-cache sử dụng liên kết FSL.
MICROBLAZE_PVR_DCACHE_ALLOW_WR (pvr)	Trả về khác 0 nếu cho phép ghi vào D-cache.
MICROBLAZE_PVR_DCACHE_LINE_LEN (pvr)	Trả về độ dài của mỗi dòng trong D-cache tính bằng byte.
MICROBLAZE_PVR_DCACHE_BYTE_SIZE (pvr)	Trả về kích thước của D-cache tính bằng byte.
MICROBLAZE_PVR_TARGET_FAMILY (pvr)	Trả lại giá trị nhận dạng họ đích được mã hóa.
MICROBLAZE_PVR_MSR_RESET_VALUE	Tham khảo Hướng dẫn Tham khảo Bộ xử lý MicroBlaze để ánh xạ từ các mã hóa đến các chuỗi họ đích . "Tài nguyên bổ sung" , trang 1 chứa một liên kết đến tài liệu này.
MICROBLAZE_PVR_MMU_TYPE (pvr)	Trả về giá trị của C_USE_MMU. Tham khảo Hướng dẫn Tham khảo Bộ xử lý MicroBlaze để biết ánh xạ từ các giá trị kiểu MMU sang hàm MMU . "Tài nguyên bổ sung" , trang 1 chứa một liên kết đến tài liệu này.

Xử lý tệp của bộ xử lý MicroBlaze

Quy trình sau được bao gồm để xử lý tệp:

```
int_fcntl (int fd, int cmd, long arg);
```

Một triển khai giả của fcntl (), luôn trả về 0, được cung cấp. fcntl nhằm mục đích thao tác các bộ mô tả tệp theo lệnh được chỉ định bởi cmd. Bởi vì Đặc lập không cung cấp hệ thống tệp, chức năng này chỉ được bao gồm để hoàn thiện.

Bộ xử lý MicroBlaze Errno

Quy trình sau cung cấp giá trị số lỗi:

```
int_errno ( );
```

Trả về giá trị toàn cục của errno như được đặt bởi lệnh gọi thư viện C cuối cùng.

PowerPC 405**API bộ xử lý**

Độc lập cho bộ xử lý PowerPC 405 chứa mã khởi động, bộ nhớ đệm, quản lý tệp và bộ nhớ, cấu hình, xử lý ngoại lệ, bao gồm thời gian và các chức năng dành riêng cho bộ xử lý.

Sau đây là danh sách các phần API bộ xử lý PowerPC 405. Để đi đến mô tả chức năng, hãy nhấp vào tên chức năng trong phần tóm tắt.

- "Mã khởi động bộ xử lý PowerPC 405"
- "Chức năng bộ nhớ đệm của bộ xử lý PowerPC 405"
- "Tóm tắt chức năng xử lý ngoại lệ của bộ xử lý PowerPC 405"
- "Tệp bộ xử lý PowerPC 405"
- "Errno Bộ xử lý PowerPC 405"
- "Quản lý bộ nhớ bộ xử lý PowerPC 405"
- "Chức năng xử lý PowerPC 405"
- "Tệp bao gồm dành riêng cho bộ xử lý PowerPC 405"
- "Chức năng thời gian của bộ xử lý PowerPC 405"
- "Bộ xử lý PowerPC 405 Macro giao diện liên kết đơn giản nhanh chóng"
- "Tóm tắt Macro Pseudo-asm Bộ xử lý PowerPC 405"
- "Macro PowerPC 405 cho hướng dẫn APU FCM do người dùng xác định"

Mã khởi động bộ xử lý PowerPC 405

Tệp boot.S chứa một bộ mã tối thiểu để chuyển quyền điều khiển từ vị trí đặt lại của bộ xử lý đến nơi khởi động ứng dụng. Code trong boot.S bao gồm hai phần boot và boot0. Phần khởi động chỉ chứa một lệnh được gắn nhãn _boot.

Trong quá trình liên kết, hướng dẫn này được ánh xạ tới vectơ đặt lại và nhãn _boot đánh dấu điểm vào của ứng dụng. Lệnh khởi động là một bức nhảy đến nhãn _boot0. Nhãn _boot0 phải nằm trong không gian địa chỉ ± 23-bit của nhãn _boot. Nó được định nghĩa trong phần boot0. Đoạn mã trong phần boot0 tính toán địa chỉ 32-bit của nhãn _start và chuyển đến địa chỉ đó.

Chức năng bộ nhớ đệm của bộ xử lý PowerPC 405

Tệp xcache_l.c và tệp bao gồm xcache_l.h tương ứng cung cấp quyền truy cập vào bộ đệm sau và các hoạt động liên quan đến bộ đệm

Tóm tắt chức năng bộ nhớ đệm của bộ xử lý PowerPC 405

Sau đây là các liên kết đến các mô tả chức năng. Bấm vào tên để chuyển đến chức năng đó.

```
void XCache_WriteCCR0 (unsigned int val)
void XCache_EnableDCache (vùng int không dấu)
void XCache_DisableDCache (void)
void XCache_FlushDCacheLine (unsigned int adr)
void XCache_InvalidateDCacheLine (unsigned int adr)
void XCache_FlushDCacheRange (unsigned int adr, unsigned len)
void XCache_InvalidateDCacheRange (unsigned int adr, unsigned len)
void XCache_StoreDCacheLine (unsigned int adr);
void XCache_EnableICache (vùng int không dấu);
void XCache_DisableICache (void);
void XCache_InvalidateICache (void);
void XCache_InvalidateICacheLine (unsigned int adr)
```

Mô tả chức năng bộ nhớ đệm của bộ xử lý PowerPC 405

```
void XCache_WriteCCR0 (unsigned int val)
```

Ghi một giá trị số nguyên vào thanh ghi CCR0. Dưới đây là một chuỗi mã mẫu. Trước khi ghi vào thanh ghi này, bộ đệm lệnh phải được kích hoạt để ngăn chặn việc khóa lỗi bộ xử lý. Sau khi ghi CCR0, bộ đệm lệnh có thể bị vô hiệu hóa nếu không cần thiết.

```
XCache_EnableICache (0x80000000) /* kích hoạt bộ nhớ cache hứa sẵn cho 128 đầu tiên
Vùng bộ nhớ MB */
XCache_WriteCCR0 (0x2700E00) /* cho phép tìm nạp trước 8 từ */
XCache_DisableICache () /* vô hiệu hóa bộ đệm chỉ dẫn */
```

```
void XCache_EnableDCache (vùng int không dấu)
```

Bật bộ đệm dữ liệu cho một vùng bộ nhớ cụ thể. Mỗi bit trong tham số vùng đại diện cho 128 MB bộ nhớ.

Giá trị 0x80000000 bật bộ đệm dữ liệu cho 128 MB bộ nhớ đầu tiên (0 - 0x07FFFFFF).

Giá trị 0x1 bật bộ đệm dữ liệu cho 128 MB bộ nhớ cuối cùng (0xF8000000 - 0xFFFFFFFF).

```
void XCache_DisableDCache (void)
```

Tắt bộ đệm dữ liệu cho tất cả các vùng bộ nhớ.

```
void XCache_FlushDCacheLine (unsigned int adr)
```

Xả và làm mất hiệu lực dòng bộ đệm dữ liệu có chứa địa chỉ được chỉ định bởi adr tham số. Một lần truy cập dữ liệu tiếp theo vào địa chỉ này dẫn đến việc bỏ lỡ bộ nhớ cache và làm đầy dòng bộ nhớ cache.

```
void XCache_InvalidateDCacheLine (unsigned int adr)
```

Làm mất hiệu lực dòng bộ đệm dữ liệu chứa địa chỉ được chỉ định bởi tham số adr. Nếu dòng bộ đệm hiện bị bắn, nội dung đã sửa đổi sẽ bị mất và không được ghi vào bộ nhớ hệ thống. Một lần truy cập dữ liệu tiếp theo vào địa chỉ này dẫn đến việc bỏ lỡ bộ nhớ cache và làm đầy dòng bộ nhớ cache.

```
void XCache_FlushDCacheRange (unsigned int adr, unsigned len)
```

Xả và làm mất hiệu lực các dòng bộ đệm dữ liệu được mô tả bằng dài địa chỉ bắt đầu từ byte adr và len dài. Một lần truy cập dữ liệu tiếp theo vào bất kỳ địa chỉ nào trong phạm vi này dẫn đến việc bỏ lỡ bộ nhớ cache và làm đầy dòng bộ nhớ cache.

```
void XCache_InvalidateDCacheRange (unsigned int adr, unsigned len)
```

Làm mất hiệu lực các dòng trong bộ đệm dữ liệu được mô tả bởi dài địa chỉ bắt đầu từ byte dài adr và len. Nếu một dòng bộ nhớ cache hiện đang bị bắn, nội dung đã sửa đổi sẽ bị mất và không được ghi vào bộ nhớ hệ thống. Một lần truy cập dữ liệu tiếp theo vào bất kỳ địa chỉ nào trong phạm vi này dẫn đến việc bỏ lỡ bộ nhớ cache và làm đầy dòng bộ nhớ cache.

```
void XCache_StoreDCacheLine (unsigned int adr);
```

Lưu trữ trong bộ nhớ dòng bộ đệm dữ liệu có chứa địa chỉ được chỉ định bởi adr tham số. Một lần truy cập dữ liệu tiếp theo vào địa chỉ này dẫn đến một lần truy cập vào bộ nhớ cache nếu địa chỉ đã được lưu trữ trong bộ nhớ cache; nếu không, nó dẫn đến việc bỏ lỡ bộ nhớ cache và lập đầy dòng bộ nhớ cache.

```
void XCache_EnableICache ( vùng int không dấu);
```

Bật bộ đệm lệnh cho một vùng bộ nhớ cụ thể. Mỗi bit trong tham số vùng đại diện cho 128 MB bộ nhớ.

Giá trị 0x80000000 kích hoạt bộ đệm lệnh cho 128 MB bộ nhớ đầu tiên (0 - 0x07FFFFFF). Giá trị 0x1 bật bộ đệm chỉ lệnh cho 128 MB bộ nhớ cuối cùng (0xF8000000 - 0xFFFFFFFF).

```
void XCache_DisableICache (void);
```

Tắt bộ đệm chỉ lệnh cho tất cả các vùng bộ nhớ.

```
void XCache_InvalidateICache (void);
```

Làm mất hiệu lực toàn bộ bộ đệm chỉ dẫn. Các hướng dẫn tiếp theo tạo ra các lần bỏ lỡ bộ nhớ cache và các dòng nạp vào bộ nhớ cache.

```
void XCache_InvalidateICacheLine (unsigned int adr)
```

Làm mất hiệu lực dòng bộ đệm chỉ dẫn có chứa địa chỉ được chỉ định bởi tham số adr .
Một lệnh tiếp theo tới địa chỉ này tạo ra lỗi bộ nhớ cache và một dòng lấp đầy bộ nhớ cache.

Xử lý ngoại lệ bộ xử lý PowerPC 405

Một API xử lý ngoại lệ được cung cấp trong Độc lập. Để có giải thích chuyên sâu về cách hoạt động của các ngoại lệ và ngắt trên bộ xử lý PowerPC, hãy tham khảo chương “Ngoại lệ và ngắt” trong Hướng dẫn Tham khảo Bộ xử lý PowerPC. Một liên kết đến tài liệu này được cung cấp trong “Tài nguyên bổ sung”, trang 1.

Lưu ý: Trình xử lý ngoại lệ không tự động đặt lại (vô hiệu hóa) bit kích hoạt trạng thái chờ trong MSR khi quay lại mã nguồn dùng. Bạn có thể buộc các trình xử lý ngoại lệ đặt lại bit Wait-Enable về 0 khi trả lại tất cả các ngoại lệ bằng cách biên dịch Độc lập với ký hiệu tiền xử lý PPC405_RESET_WE_ON_RFI được xác định. Bạn có thể thêm điều này vào cờ trình biên dịch được liên kết với các thư viện. Định nghĩa tiền xử lý này sẽ bắt hành vi.

API xử lý ngoại lệ bao gồm một tập hợp các tệp xvectors.S, xexception_l.c và tệp tiêu đề tư ứng xexception_l.h.

Để biết thêm thông tin về xử lý gián đoạn, hãy xem Trợ giúp XPS và phụ lục “Quản lý ngắt” trong Sổ tay Tham khảo Công cụ Hệ thống Nhúng (có sẵn trong thư mục / doc của cài đặt EDK của bạn).

Tóm tắt chức năng xử lý ngoại lệ của bộ xử lý PowerPC 405

Sau đây là các liên kết đến các mô tả chức năng. Bấm vào tên để chuyển đến chức năng đó.

```
void XExc_Init (void)
void XExc_RegisterHandler (Xuint8 ExceptionId, XExceptionHandler Handler, void * DataPtr)
void XExc_RemoveHandler (Xuint8 ExceptionId)
void XExc_mEnableExceptions (EnableMask)
void XExc_mDisableExceptions (DisableMask)
```

Mô tả chức năng xử lý ngoại lệ của bộ xử lý PowerPC 405

```
void XExc_Init (void)
```

Thiết lập bảng vectơ ngắt và đăng ký một chức năng “không làm gì cả” cho mọi ngoại lệ. Hàm này không có tham số và không trả về giá trị.

Hàm này phải được gọi trước khi đăng ký bất kỳ trình xử lý ngoại lệ nào hoặc cho phép bất kỳ ngắt nào. Khi sử dụng API xử lý ngoại lệ, hàm này phải được gọi ở đầu quy trình main () của bạn.

QUAN TRỌNG: Nếu bạn không sử dụng tập lệnh trình liên kết mặc định, bạn cần dành không gian bộ nhớ để lưu bảng vectơ trong tập lệnh trình liên kết của mình. Không gian bộ nhớ phải bắt đầu trên ranh giới 64 k.

Mục nhập tập lệnh của trình liên kết sẽ giống như ví dụ sau:

```
.vectors:
{
    . = ALIGN (64k);
    * (. vecto)
}
```

Để biết thêm thông tin về các tập lệnh của trình liên kết, hãy tham khảo tài liệu về Trình liên kết.

```
void XExc_RegisterHandler (Xuint8 ExceptionId,
                           Trình xử lý XExceptionHandler, void * DataPtr)
```

Đăng ký một trình xử lý ngoại lệ cho một ngoại lệ cụ thể; không trả về giá trị. Tham khảo bảng sau để biết danh sách các loại ngoại lệ và giá trị của chúng.

Các thông số là:

- ExceptionId có kiểu tham số Xuint8 và là ngoại lệ mà trình xử lý này nên đư ợc đăng ký. Loại và các giá trị đư ợc xác định trong xexception_l.h tập tin tiêu đề. Bảng sau liệt kê các loại ngoại lệ và các giá trị có thể có.
- Handler là một tham số XExceptionHandler là con trỏ đến hàm xử lý ngoại lệ.
- DataPtr có kiểu tham số void * và là giá trị ngư ời dùng đư ợc chuyển khi hàm xử lý đư ợc gọi.

Bảng 4: Các loại và giá trị ngoại lệ đã đăng ký

Loại ngoại lệ	Giá trị
XEXC_ID_MACHINE_CHECK	1
XEXC_ID_CRITICAL_INT	2
XEXC_ID_DATA_STORAGE_INT	3
XEXC_ID_INSTRUCTION_STORAGE_INT	4
XEXC_ID_NON_CRITICAL_INT	5
XEXC_ID_ALIGNMENT_INT	6
XEXC_ID_PROGRAM_INT	7
XEXC_ID_FPU_UNAVAILABLE_INT	8
XEXC_ID_SYSTEM_CALL	9
XEXC_ID_APU_AVAILABLE	10
XEXC_ID_PIT_INT	11
XEXC_ID_FIT_INT	12
XEXC_ID_WATCHDOG_TIMER_INT	13
XEXC_ID_DATA_TLB_MISS_INT	14
XEXC_ID_INSTRUCTION_TLB_MISS_INT	15
XEXC_ID_DEBUG_INT	16

Hàm được cung cấp dưới dạng tham số Xử lý phải có nguyên mẫu hàm sau:

```
typedef void (* XExceptionHandler) (void * DataPtr);
```

Nguyên mẫu này được khai báo trong tệp tiêu đề xexception_l.h.

Khi hàm xử lý ngoại lệ này được gọi, tham số DataPtr chứa cùng giá trị như bạn đã cung cấp khi đăng ký trình xử lý.

void XExc_RemoveHandler (Xuint8 ExceptionId)

Hủy đăng ký một hàm xử lý cho một ngoại lệ nhất định. Để biết các giá trị có thể có của tham số ExceptionId, hãy tham khảo [Bảng 7, trang 42](#).

void XExc_mEnableExceptions (EnableMask)

Bật ngoại lệ. Macro này phải được gọi sau khi khởi tạo bảng vectơ với hàm exception_Init và đăng ký trình xử lý ngoại lệ với hàm XExc_RegisterHandler. Tham số EnableMask là một mảng nạp bit cho các trường hợp ngoại lệ được kích hoạt. Tham số EnableMask có thể có các giá trị XEXC_CRITICAL, XEXC_NON_CRITICAL hoặc XEXC_ALL.

void XExc_mDisableExceptions (DisableMask)

Tắt các ngoại lệ. Tham số DisableMask là một mảng nạp bit để tắt các ngoại lệ. Tham số DisableMask có thể có các giá trị XEXC_CRITICAL, XEXC_NON_CRITICAL hoặc XEXC_ALL.

Tệp bộ xử lý PowerPC 405

Hỗ trợ tệp được giới hạn cho các luồng stdin và stdout; do đó, các chức năng sau là không cần thiết:

- open () (trong open.c)
- close () (in close.c)
- fstat () (trong fstat.c)
- unlink () (trong unlink.c)
- lseek () (trong lseek.c)

Các tệp này được bao gồm để hoàn thiện và vì chúng được tham chiếu bởi thư viện C.

int read (int fd, char * buf, int nbytes)

Hàm read () trong read.c đọc nbyte byte từ đầu vào chuẩn bằng cách gọi inbyte (). Nó chặn cho đến khi tất cả các ký tự có sẵn hoặc ký tự cuối dòng được đọc. Hàm read () trả về số ký tự được đọc. Tham số fd bị bỏ qua.

int write (int fd, char * buf, int nbytes)

Ghi nbyte byte vào đầu ra tiêu chuẩn bằng cách gọi outbyte (). Nó chặn cho đến khi tất cả các ký tự đã được viết. Hàm write () trả về số ký tự được viết. Fd _ tham số bị bỏ qua.

```
int isatty (int fd)
```

Báo cáo nếu một tệp được kết nối với một tty. Hàm này luôn trả về 1, Bởi vì chỉ có stdin và các luồng stdout được hỗ trợ.

```
int fcntl (int fd, int cmd, long arg);
```

Một triển khai giả của fcntl, luôn trả về 0. fcntl nhằm mục đích thao tác các bộ mô tả tệp theo lệnh được chỉ định bởi cmd. Vì Standalone không cung cấp hệ thống tệp nên chức năng này không được sử dụng.

Bộ xử lý PowerPC 405 Errno

```
int errno ()
```

Trả về giá trị toàn cục của errno như được đặt bởi lệnh gọi thư viện C cuối cùng.

Quản lý bộ nhớ bộ xử lý PowerPC 405

```
char * sbrk (int nbytes)
```

Phân bổ nbyte của heap và trả về một con trỏ đến phần bộ nhớ đó. Hàm này được gọi từ các hàm cấp phát bộ nhớ của thư viện C.

Chức năng xử lý PowerPC 405

Các hàm getpid () trong getpid.c và kill () trong kill.c được bao gồm để hoàn thiện và vì chúng được tham chiếu bởi thư viện C.

Tệp bao gồm bộ xử lý PowerPC 405 dành riêng cho bộ xử lý

Tệp bao gồm xreg405.h chứa các số đăng ký và các bit đăng ký cho bộ xử lý PowerPC 405.

Tệp bao gồm xpseudo-asm.h chứa các định nghĩa cho các hướng dẫn trình hợp dịch nội tuyến thư ờng được sử dụng nhất, có sẵn dưới dạng macro. Chúng có thể rất hữu ích cho các tác vụ như thiết lập hoặc nhận các thanh ghi mục đích đặc biệt, đồng bộ hóa hoặc thao tác bộ nhớ cache.

Các hướng dẫn trình hợp dịch nội tuyến này có thể được sử dụng từ các trình điều khiển và ứng dụng người dùng được viết bằng C.

Chức năng thời gian của bộ xử lý PowerPC 405

Tệp xtime_l.c và tệp bao gồm xtime_l.h tư ơng ứng cung cấp quyền truy cập vào bộ đếm cơ sở thời gian 64-bit bên trong lõi PowerPC. Bộ đếm tăng một ở mỗi chu kỳ xử lý.

Tệp sleep.c và sleep.h tư ơng ứng bao gồm tệp thực hiện các chức năng ngủ. Các chức năng ngủ được thực hiện dưới dạng vòng lặp bận

Tóm tắt chức năng thời gian của bộ xử lý PowerPC 405 Sau đây

là các liên kết đến mô tả chức năng. Bấm vào tên để chuyển đến chức năng đó.

```
typedef unsigned long long
XTime void XTime_SetTime
(XTime xtime) void
XTime_GetTime (XTime * xtime) void
XTime_TSRClearStatusBits (unsigned long
Bitmask) void XTime_PITSetInterval (unsigned
long gap) void XTimenable_PITEITimeable (void
XTimeelLoadAnterTimeable void
XTime_PITClearInterrupt (void) void
XTime_FITEnableInterrupt (void) void
XTime_FITDisableInterrupt (void) void
XTime_FITClearInterrupt (void) void
XTime_FITSetPeriod (unsigned long period)
void XTime_WDTEnable ) void XTime_WDTResetControl
(unsigned long ControlVal) void
XTime_WDTEnableNextWatchdog (void) void
XTime_WDTClearResetStatus (void) unsigned int
usleep (unsigned int _useconds) unsigned int
sleep (unsigned int _seconds) int nanosleep
(const struct timespec * rpq)
```

Mô tả chức năng thời gian của bộ xử lý PowerPC 405

typedef không dấu XTime dài dài

Kiểu XTime trong xtime_l.h đại diện cho thanh ghi Cơ sở thời gian. Cấu trúc này bao gồm các thanh ghi Time Base Low (TBL) và Time Base High (TBH), mỗi thanh ghi là một thanh ghi rộng 32 bit.

Định nghĩa của XTime như sau:

```
typedef không dấu dài dài XTime;
```

```
void XTime_SetTime (XTime xtime)
```

Đặt thanh ghi cơ sở thời gian thành giá trị trong xtime.

```
void XTime_GetTime (XTime * xtime)
```

Ghi giá trị hiện tại của thanh ghi cơ sở thời gian vào biến xtime.

```
void XTime_TSRClearStatusBits (Bitmask dài không dấu)
```

Xóa các bit trong thanh ghi trạng thái bộ định thời (TSR). Tham số Bitmask chỉ định các bit cần xóa. Giá trị 1 ở bất kỳ vị trí nào của tham số Bitmask sẽ xóa bit tương ứng trong TSR. Hàm này không trả về giá trị.

Thí dụ:

```
XTime_TSRClearStatusBits (TSR_CLEAR_ALL);
```

Bảng sau chứa các giá trị cho các tham số Bitmask được chỉ định trong tệp tiêu đề xreg405.h.

Bảng 5: Giá trị tham số Bitmask

Tên	Giá trị	Sự mô tả
XREG_TSR_WDT_ENABLE_NEXT_WATCHDOG	0x80000000	Xóa bit này sẽ vô hiệu hóa sự kiện bộ đếm thời gian của cơ quan giám sát.
XREG_TSR_WDT_INTERRUPT_STATUS	0x40000000	Xóa bit Trạng thái ngắt của bộ định thời gian giám sát. Bit này được thiết lập sau khi xảy ra ngắt cơ quan giám sát.
XREG_TSR_WDT_RESET_STATUS_11	0x30000000	Xóa các bit Trạng thái Đặt lại Bộ định thời của Cơ quan giám sát. Các bit này chỉ định kiểu đặt lại đã xảy ra do sự kiện bộ đếm thời gian của cơ quan giám sát.
XREG_TSR_PIT_INTERRUPT_STATUS	0x08000000	Xóa bộ hẹn giờ khoảng thời gian có thể lập trình (PIT) Trạng thái bit. Bit này được đặt sau một ngắt PIT tần suất xảy ra
XREG_TSR_FIT_INTERRUPT_STATUS	0x04000000	Xóa bit Trạng thái Bộ hẹn giờ Khoảng thời gian Có định (FIT). Bit này được đặt sau khi xảy ra ngắt FIT
XREG_TSR_CLEAR_ALL	0xFFFFFFFF	Xóa tất cả các bit trong TSR. Sau khi Đặt lại, nội dung của TSR không được chỉ định. Sử dụng Bitmask này để xóa tất cả các bit trong TSR

```
void XTime_PITSetInterval ( khoảng thời gian dài không dấu)
```

Nạp một giá trị mới vào thanh ghi bộ hẹn giờ có thể lập trình được. Thanh ghi này là một bộ đếm giảm dần 32 bit có cùng tần số với thanh ghi cơ sở thời gian. Tùy thuộc vào cài đặt Tự động tải lại, PIT được tự động tải lại với giá trị được ghi cuối cùng hoặc phải được tải lại theo cách thủ công. Hàm này không trả về giá trị.

Thí dụ:

```
XTime_PITSetInterval (0x00ffff);
```

```
void XTime_PITEnableInterrupt (void)
```

Cho phép tạo ra các ngắt PIT. Ngắt xảy ra khi thanh ghi PIT chưa giá trị 1, và sau đó được giảm dần. Hàm này không trả về giá trị. XExc_Init () phải được gọi, trình xử lý ngắt PIT phải được đăng ký, và các ngoại lệ phải được kích hoạt trước khi gọi hàm này.

Thí dụ:

```
XTime_PITEnableInterrupt ();
```

```
void XTime_PITDisableInterrupt (void)
```

Tắt tạo ngắt PIT. Nó không trả về một giá trị.

Thí dụ:

```
XTime_PITDisableInterrupt ();
```

```
void XTime_PITEnableAutoReload (void)
```

Bật chức năng tự động tải lại Số đăng ký thuê TNCN. Khi bật tính năng tự động tải lại, Số đăng ký PIT sẽ tự động được tải lại với giá trị cuối cùng được tải bằng cách gọi hàm XTime_PITSetInterval () khi Số đăng ký PIT có giá trị là 1 và được giảm dần. Khi bật tính năng tự động tải lại, Số đăng ký PIT không bao giờ chứa giá trị bằng 0. Chức năng này không trả về giá trị.

Thí dụ:

```
XTime_PITEnableAutoReload ();
```

```
void XTime_PITDisableAutoReload (void)
```

Tắt tính năng tự động tải lại của Số đăng ký thuê TNCN. Khi tắt tính năng tự động tải lại, PIT sẽ giảm từ 1 xuống 0. Nếu chứa giá trị 0, nó sẽ ngừng giảm cho đến khi được tải bằng giá trị khác 0. Hàm này không trả về giá trị.

Thí dụ:

```
XTime_PITDisableAutoReload ();
```

void XTime_PITClearInterrupt (void)

Xóa bit PIT-Ngắt-Trạng thái trong Thanh ghi Trạng thái-Bộ định thời. Bit này chỉ định xem có xảy ra ngắt PIT hay không. Bạn phải gọi hàm này trong trình xử lý ngắt của mình để xóa bit Trạng thái, nếu không thì một ngắt PIT khác xảy ra ngay sau khi thoát khỏi chức năng xử lý ngắt. Hàm này không trả về giá trị. Gọi hàm này tương đương với gọi XTime_TSRClearStatusBits (XREG_TSR_PIT_INTERRUPT_STATUS).

Thí dụ:

```
XTime_PITClearInterrupt();
```

void XTime_FITEnableInterrupt (void)

Bật ngắt bộ hẹn giờ khoảng thời gian cố định (FIT).

Thí dụ:

```
XTime_FITEnableInterrupt();
```

void XTime_FITDisableInterrupt (void)

Vô hiệu hóa ngắt bộ hẹn giờ khoảng thời gian cố định (FIT).

Thí dụ:

```
XTime_FITDisableInterrupt();
```

void XTime_FITClearInterrupt (void)

Xóa bit trạng thái ngắt của Bộ định thời gian cố định (FIT). Chức năng này tương đương với việc gọi XTime_TSRClearStatusBits (XREG_TSR_FIT_INTERRUPT_STATUS).

Thí dụ:

```
XTime_FITDisableInterrupt();
```

void XTime_FITSetPeriod (Khoảng thời gian dài chia k)

Đặt giá trị Khoảng thời gian cố định khoảng thời gian (FIT) . Giá trị này có thể là một trong những giá trị sau:

- XREG_TCR_FIT_PERIOD_11 (2 ^ 21 đồng hồ)
- XREG_TCR_FIT_PERIOD_10 (2 ^ 17 đồng hồ)
- XREG_TCR_FIT_PERIOD_01 (2 ^ 13 đồng hồ)
- XREG_TCR_FIT_PERIOD_00 (2 ^ 9 đồng hồ)

Các giá trị này được xác định trong xreg405.h

Thí dụ:

```
XTime_FITSetPeriod (XREG_TCR_FIT_PERIOD_11);
```

```
void XTime_WDTEnableInterrupt (void)
Bật ngắt bộ định thời gian giám sát (WDT).
```

Thí dụ:

```
XTime_WDTEnableInterrupt ();
```

```
void XTime_WDTDisableInterrupt (void)
Tắt các ngắt của Watchdog Timer (WDT).
```

Thí dụ:

```
XTime_WDTDisableInterrupt ();
```

```
void XTime_WDTClearInterrupt (vô hiệu)
```

Xóa bit trạng thái ngắt Watchdog Timer (WDT). Gọi hàm này tương ứng với gọi XTime_TSRClearStatusBits (XREG_TSR_WDT_INTERRUPT_STATUS).

Thí dụ:

```
XTime_WDTClearInterrupt ();
```

```
void XTime_WDTSetPeriod (Đầu chấm dài chia a ký)
```

Đặt khoảng thời gian cho sự kiện Bộ hẹn giờ cơ quan giám sát (WDT).

Thí dụ:

```
XTime_WDTSetPeriod (0x10000);
```

```
void XTime_WDTResetControl ( ControlVal dài không dấu)
```

Chỉ định loại thiết lập lại xảy ra do sự kiện Bộ định thời gian giám sát (WDT).

Giá trị điều khiển có thể là một trong những giá trị sau:

- XREG_WDT_RESET_CONTROL_11 (Đặt lại hệ thống)
- XREG_WDT_RESET_CONTROL_10 (Đặt lại chip)
- XREG_WDT_RESET_CONTROL_01 (đặt lại bộ xử lý)
- XREG_WDT_RESET_CONTROL_00 (không đặt lại)

Các giá trị này được xác định trong xreg405.h

Thí dụ:

```
XTime_WDTResetControl (XREG_WDT_RESET_CONTROL_11);
```

```
void XTime_WDTEnableNextWatchdog (void)
```

Bật sự kiện Bộ hẹn giờ cơ quan giám sát (WDT).

Thí dụ:

```
XTime_WDTEnableNextWatchdog ();
```

```
void XTime_WDTClearResetStatus (void)
```

Xóa các bit trạng thái đặt lại Watchdog Timer (WDT).

Thí dụ:

```
XTime_WDTClearResetStatus ();
```

```
unsigned int usleep (unsigned int _useconds)
```

Trì hoãn việc thực thi chương trình trong `_useconds` micro giây. Nó luôn trả về số không. Chức năng này yêu cầu tần số bộ xử lý (tính bằng Hz) được xác định. Giá trị mặc định của biến này là 400 MHz. Giá trị này có thể được ghi đè trong tệp Thông số kỹ thuật phần mềm vi xử lý (MSS) như sau:

```
BỘ XỬ LÝ BEGIN
PARAMETER HW_INSTANCE = PPC405_i
PARAMETER DRIVER_NAME = cpu_ppc405
PARAMETER DRIVER_VER = 1,00.a
PARAMETER CORE_CLOCK_FREQ_HZ = 20000000
CHẨM DỨT
```

Tệp `xparameters.h` cũng có thể được sửa đổi với giá trị chính xác, như sau:

```
#define XPAR_CPU_PPC405_CORE_CLOCK_FREQ_HZ 20000000
```

```
unsigned int sleep (unsigned int _seconds)
```

Trì hoãn việc thực thi một chương trình bởi những gì được chỉ định trong `_seconds`. Nó luôn trả về 0. Hàm này yêu cầu tần số của bộ xử lý (tính bằng Hz) được xác định. Giá trị mặc định của biến này là 400 MHz. Giá trị này có thể được ghi đè trong tệp Thông số kỹ thuật phần mềm vi xử lý (MSS) như sau:

```
BỘ XỬ LÝ BEGIN
PARAMETER HW_INSTANCE = PPC405_i
PARAMETER DRIVER_NAME = cpu_ppc405
PARAMETER DRIVER_VER = 1,00.a
PARAMETER CORE_CLOCK_FREQ_HZ = 20000000
CHẨM DỨT
```

Tệp `xparameters.h` cũng có thể được sửa đổi với giá trị chính xác, như sau:

```
#define XPAR_CPU_PPC405_CORE_CLOCK_FREQ_HZ 20000000
```

```
int nanosleep (const struct timespec * rqtp, struct timespec
* rmtp)
```

Hàm `nanosleep ()` trong `sleep.c` không được triển khai. Nó là một trình giữ chỗ để liên kết các ứng dụng với thư viện C và trả về số không.

Bộ xử lý PowerPC 405 Macro giao diện liên kết đơn giản nhanh chóng

Độc lập bao gồm các macro để cung cấp khả năng truy cập thuận tiện vào các bộ gia tốc được kết nối với Bộ xử lý Phụ trợ (APU) của bộ xử lý PowerPC 405 qua các giao diện FSL.

Bộ xử lý PowerPC 405 Giao diện liên kết nhanh Simplex Tóm tắt Macro

Sau đây là danh sách liên kết các macro; nhấp vào tên macro để chuyển đến mô tả.

<code>getfsl (val, id)</code>	<code>ncputfsl (val, id)</code>
<code>putfsl (val, id)</code>	<code>getfsl_interruptible (val, id)</code>
<code>ngetfsl (val, id)</code>	<code>putfsl_interruptible (val, id)</code>
<code>nputfsl (val, id)</code>	<code>cgetfsl_interruptible (val, id)</code>
<code>cgetfsl (val, id)</code>	<code>cputfsl_interruptible (val, id)</code>
<code>cputfsl (val, id)</code>	<code>fsl_isinvalid (không hợp lệ)</code>
<code>ncgetfsl (val, id)</code>	<code>fsl_iserror (lỗi)</code>

Bộ xử lý PowerPC 405 Mô tả macro giao diện FSL

Trong macro, val đề cập đến một biến trong chương trình của bạn có thể là nguồn hoặc phần chìm của hoạt động FSL. Bạn phải bao gồm tệp tiêu đề fsl.h trong tệp nguồn của mình để cung cấp các macro này.

`getfsl (val, id)`

Thực hiện chức năng lấy dữ liệu chặn trên giao diện FSL đầu vào; id là mã định danh FSL trong phạm vi từ 0 đến 31. Macro này có thể ngắn đư ợc.

`putfsl (val, id)`

Thực hiện chức năng đặt dữ liệu chặn trên giao diện FSL đầu ra; id là mã định danh FSL trong phạm vi từ 0 đến 31. Macro này có thể ngắn đư ợc.

`ngetfsl (val, id)`

Thực hiện chức năng lấy dữ liệu không chặn trên giao diện FSL đầu vào; id là số nhận dạng FSL trong phạm vi từ 0 đến 31.

`nputfsl (val, id)`

Thực hiện chức năng đặt dữ liệu không chặn trên giao diện FSL đầu ra; id là số nhận dạng FSL trong phạm vi từ 0 đến 31.

`cgetfsl (val, id)`

Thực hiện chức năng lấy kiểm soát chặn trên giao diện FSL đầu vào; id là mã định danh FSL trong phạm vi từ 0 đến 31. Macro này có thể ngắn đư ợc.

cputfsl (val, id)

Thực hiện chức năng đặt kiểm soát chặn trên giao diện FSL đầu ra; id là mã định danh FSL trong phạm vi từ 0 đến 31. Macro này có thể ngắn đư ợc.

ncgetfsl (val, id)

Thực hiện chức năng nhận điều khiển không chặn trên giao diện FSL đầu vào; id là số nhận dạng FSL trong phạm vi từ 0 đến 31.

ncputfsl (val, id)

Macro này thực hiện chức năng kiểm soát dữ liệu không chặn trên giao diện FSL đầu ra; id là số nhận dạng FSL trong phạm vi từ 0 đến 31.

getfsl_interruptible (val, id)

Macro này đư ợc đặt bí danh là getfsl (val, id).

putfsl_interruptible (val, id)

Macro này đư ợc đặt bí danh là putfsl (val, id).

cgetfsl_interruptible (val, id)

Macro này đư ợc đặt bí danh là cgetfsl (val, id).

cputfsl_interruptible (val, id)

Macro này đư ợc đặt bí danh là cputfsl (val, id).

fsl_isinvalid (không hợp lệ)

Kiểm tra để xác định xem hoạt động FSL cuối cùng có trả lại dữ liệu hợp lệ hay không. Macro này có thể áp dụng sau khi gọi lệnh đặt hoặc nhận FSL không chặn. Nếu không có dữ liệu trên kênh FSL khi đặt hoặc nếu kênh FSL đã đầy khi đặt, thì không hợp lệ đư ợc đặt thành 1; nếu không, không hợp lệ đư ợc đặt thành 0.

fsl_iserror (lỗi)

Kiểm tra để xác định xem hoạt động FSL cuối cùng có đặt cờ lỗi hay không. Macro này có thể áp dụng sau khi gọi một FSL điều khiển đặt hoặc nhận lệnh. Nếu bit điều khiển đư ợc thiết lập, lỗi đư ợc đặt thành 1; nếu không, nó đư ợc đặt thành 0.

Bộ xử lý PowerPC 405 Macro giả asm

Độc lập bao gồm các macro để cung cấp khả năng truy cập thuận tiện vào các thanh ghi khác nhau trên bộ xử lý PowerPC 405. Bạn phải bao gồm tệp tiêu đề xpseudo_asm.h trong mã nguồn của mình để sử dụng các API này.

Tóm tắt Macro Pseudo-asm Bộ xử lý PowerPC 405 Sau đây là danh sách được liên kết của Macro Pseudo-asm; nhấp vào tên macro để chuyển đến mô tả.

<code>mfgpr (rn)</code>	<code>icbi (adr)</code>	<code>lbz (adr)</code>
<code>mfsprr (rn)</code>	<code>icbt (adr)</code>	<code>lhz (adr)</code>
<code>mfmsr ()</code>	<code>isync dcccii</code>	<code>lwz (adr)</code>
<code>mfdcr (rn)</code>	<code>(adr) dcibi</code>	<code>stb (adr, val) sth</code>
<code>mtdcr (rn, v)</code>	<code>(adr) dcbst</code>	<code>(adr, val) stw</code>
<code>mtevpr (addr) mtspr</code>	<code>(adr) dcibf</code>	<code>(adr, val) lhbrx</code>
<code>(rn, v) mtgpr (rn,</code>	<code>(adr) dcread</code>	<code>(adr) lwbrx (adr)</code>
<code>v)</code>	<code>(adr)</code>	<code>sthbrx (adr, val)</code>
<code>iccci</code>	<code>eieio</code>	<code>stwbrx (adr, val)</code>

đóng bộ hóa

Mô tả macro Pseudo-asm của bộ xử lý PowerPC 405

`mfgpr (rn)`

Giá trị trả về từ GPR rn.

`mfsprr (rn)`

Trả về giá trị hiện tại của thanh ghi mục đích đặc biệt (SPR) rn.

`mfmsr ()`

Giá trị trả về từ MSR.

`mfdcr (rn)`

Trả về giá trị từ thanh ghi điều khiển thiết bị (DCR) rn.

`mtdcr (rn, v)`

Di chuyển giá trị v sang DCR rn.

`mtevpr (addr)`

Di chuyển bộ cộng giá trị vào thanh ghi tiền tố vectơ ngoại lệ (EVPR) .

`mtspr (rn, v)`

Di chuyển giá trị v thành SPR rn.

mtgpr (rn, v)

Đi chuyển giá trị v sang GPR rn.

iccci

Làm mất hiệu lực lớp đồng thời của bộ đệm chỉ dẫn (tất cả bộ đệm).

icbi (adr)

Làm mất hiệu lực khỏi bộ nhớ cache hư hỏng dẫn tại địa chỉ hiệu quả adr.

icbt (adr)

Chạm vào khỏi bộ nhớ cache hư hỏng dẫn tại adr địa chỉ hiệu quả.

isync

Thực thi lệnh isync .

dccci (adr)

Làm mất hiệu lực lớp ứng ứng của bộ đệm dữ liệu được đại diện bởi adr địa chỉ hiệu quả.

dcbi (adr)

Làm mất hiệu lực khỏi bộ nhớ cache dữ liệu tại địa chỉ hiệu quả adr.

dcbst (adr)

Lưu trữ khỏi bộ nhớ cache dữ liệu tại địa chỉ hiệu quả adr.

dcbf (adr)

Xóa khỏi bộ nhớ cache dữ liệu tại địa chỉ hiệu quả adr.

dcread (adr)

Đọc từ địa chỉ bộ nhớ cache dữ liệu adr.

eieio

Thực hiện lệnh eieio.

đồng bộ hóa

Thực hiện hư hỏng dẫn đồng bộ hóa.

lbz (adr)

Thực hiện tải và trả về giá trị byte từ địa chỉ adr.

lhz (adr)

Thực hiện tải và trả về giá trị nửa từ từ địa chỉ adr.

lwz (adr)

Thực hiện tải và trả về giá trị từ từ địa chỉ adr.

stb (adr, val)

Lưu trữ giá trị byte trong val vào địa chỉ adr.

sth (adr, val)

Lưu trữ giá trị nửa từ trong val vào địa chỉ adr.

stw (adr, val)

Lưu trữ giá trị từ trong val vào địa chỉ adr.

lhbrx (adr)

Thực hiện huống dẫn đư ợc lập chỉ mục theo kiểu nửa từ khóa đư ợc đảo ngược trên adr địa chỉ hiệu quả và trả về giá trị.

lwbrx (adr)

Thực hiện huống dẫn đư ợc Lập chỉ mục theo byte đư ợc đảo ngược Load Word trên địa chỉ hiệu quả adr và trả về giá trị.

sthbrx (adr, val)

Thực hiện huống dẫn Lập chỉ mục nửa từ theo byte đư ợc đảo ngược của cửa hàng trên adr địa chỉ hiệu quả, trên giá trị val.

stwbrx (adr, val)

Thực hiện huống dẫn Lập chỉ mục theo từng phân đoạn đư ợc đảo ngược của Store Word trên adr địa chỉ hiệu quả, trên giá trị val.

Macro PowerPC 405 cho APU FCM do ngư ời dùng xác định hư ớng dẫn

Macro được cung cấp để sử dụng các hư ớng dẫn do ngư ời dùng xác định được hỗ trợ bởi Mô-đun đồng xử lý vài PowerPC 405 APU (FCM). Có tổng cộng 16 cách ghi nhớ hư ớng dẫn do ngư ời dùng xác định được cung cấp: tám hư ớng dẫn sửa đổi Số đăng ký điều kiện (CR) và tám hư ớng dẫn không sửa đổi CR. Bởi vì ý nghĩa của các toán hạng mà các hư ớng dẫn này sử dụng có thể được xác định lại động, các macro được cung cấp cho tất cả các kết hợp của các toán hạng. Chuyển trình ngư ời dùng phải sử dụng các macro một cách thích hợp, cùng với luồng chương trình cấp cao hơn.

UDI <n> FCM (a, b, c, fmt)

Chèn ký hiệu cho lệnh fcm do ngư ời dùng xác định n (không sửa đổi CR) vào chương trình ngư ời dùng. Lệnh do ngư ời dùng định nghĩa, có a, b, c là các toán hạng của nó theo thứ tự đó. Cách các toán hạng được trình biên dịch giải thích, được xác định bởi trình định dạng do fmt đưa ra. Thông số định dạng được giải thích thêm bên dưới. n có thể nằm trong khoảng từ 0 đến 7. Phép ghi nhớ được chèn là, udi <n> fcm.

UDI <n> FCMCR (a, b, c, fmt)

Chèn ký hiệu cho lệnh fcm do ngư ời dùng xác định (sửa đổi CR) n vào chương trình ngư ời dùng. Lệnh do ngư ời dùng định nghĩa có a, b, c là các toán hạng của nó theo thứ tự đó. Cách các toán hạng được trình biên dịch giải thích, được xác định bởi trình định dạng fmt. Bảng sau liệt kê các mô tả và định dạng mã định dạng. Giá trị của <n> có phạm vi từ 0 đến 7. Cú pháp ghi nhớ là udi <n> fcm. (lưu ý dấu chấm ở cuối).

Bảng 6: Định dạng chỉ định cho hư ớng dẫn UDI

Định danh	Nghĩa
FMT_GPR_GPR_GPR	Toán hạng a, b và c là các thanh ghi mục đích chung
FMT_GPR_GPR_IMM	Toán hạng a và b là các thanh ghi mục đích chung. Toán hạng c là một giá trị tức thời đại diện cho một hằng số tức thời hoặc một thanh ghi FCM
FMT_GPR_IMM_IMM	Toán hạng a là một thanh ghi mục đích chung. Toán hạng b và c là các giá trị tức thời đại diện cho một hằng số tức thời hoặc một thanh ghi FCM
FMT_IMM_GPR_GPR	Toán hạng b và c là các thanh ghi mục đích chung. Toán hạng a là một giá trị tức thời đại diện cho một hằng số tức thời hoặc một thanh ghi FCM.
FMT_IMM_IMM_GPR	Toán hạng c là một thanh ghi mục đích chung. Toán hạng a và b là các giá trị tức thời đại diện cho một hằng số tức thời hoặc một thanh ghi FCM.
FMT_IMM_IMM_IMM	Cả ba toán hạng đều là các giá trị tức thời đại diện cho một hằng số tức thời hoặc một thanh ghi FCM.

PowerPC 440**API bộ xử lý**

Độc lập chứa mã khởi động, bộ nhớ cache, quản lý tệp và bộ nhớ, cấu hình, xử lý ngoại lệ, bao gồm thời gian và bộ xử lý cụ thể.

Sau đây liệt kê các phần API bộ xử lý PowerPC 440. Để chuyển đến phần chức năng, hãy nhấp vào tên.

- “[Mã khởi động bộ xử lý PowerPC 440](#)”
- “[Chức năng bộ nhớ đệm của bộ xử lý PowerPC 440](#)”
- “[Xử lý ngoại lệ bộ xử lý PowerPC 440](#)”
- “[Chức năng Errno của bộ xử lý PowerPC 440](#)”
- “[Quản lý bộ nhớ bộ xử lý PowerPC 440](#)”
- “[Chức năng Quy trình PowerPC 440](#)”
- “[Tệp bao gồm bộ xử lý PowerPC 440 dành riêng cho bộ xử lý](#)”
- “[Chức năng thời gian của bộ xử lý PowerPC 440](#)”

Các phần phụ sau đây mô tả các chức năng của bộ xử lý PowerPC 440 theo loại.

Mã khởi động bộ xử lý PowerPC 440

Tệp boot.S chứa một bộ mã tối thiểu để chuyển quyền điều khiển từ vị trí đặt lại của bộ xử lý đến nơi khởi động ứng dụng. Code trong boot.S bao gồm hai phần boot và boot0.

Phần khởi động chỉ chứa một lệnh được gắn nhãn _boot. Trong quá trình liên kết, lệnh này được ánh xạ tới vectơ đặt lại và nhãn _boot đánh dấu điểm vào của ứng dụng. Lệnh khởi động là một bước nhảy đến nhãn _boot0 và nó được định nghĩa trong phần boot0.

Khi đặt lại lõi 440, chỉ trang bộ nhớ chương trình 4 kB, nằm ở cuối vùng địa chỉ hiệu dụng 32-bit (bắt đầu từ 0xFFFFF000), được ánh xạ vào MMU của bộ xử lý.

Phần .boot0 chứa các hướng dẫn khởi tạo TLB trong MMU sao cho toàn bộ không gian địa chỉ 4 GB được ánh xạ một cách minh bạch cho cả phía I và D:

- Các mục nhập TLB phía I có bộ nhận dạng vùng địa chỉ được đặt thành 0.
- Các mục TLB phía D có bộ nhận dạng vùng địa chỉ được đặt thành 1.

Phần .boot0 nằm ở địa chỉ 0xFFFFF000 nằm trong vùng bộ nhớ được ánh xạ ban đầu.

Ngoài việc ánh xạ TLB, mã trong boot0 cũng làm mất hiệu lực của bộ nhớ đệm I và D. Các thanh ghi lõi khác như CCR01, CCR1 và MSR được khởi tạo. MSR [DS] được đặt thành 1 để phân vùng các bản dịch phía dữ liệu sang không gian địa chỉ 1. Cuối cùng, mã trong phần boot0 tính toán 32-bit của nhãn _start và nhảy đến địa chỉ đó.

Chức năng bộ nhớ đệm của bộ xử lý PowerPC 440

Tệp xcache_1.c và tệp bao gồm xcache_1.h tương ứng cung cấp quyền truy cập vào bộ đệm sau và các hoạt động liên quan đến bộ đệm.

Tóm tắt chức năng bộ nhớ đệm của bộ xử lý PowerPC 440

Sau đây là các liên kết đến các mô tả chức năng. Bấm vào tên để chuyển đến chức năng đó.

```
void XCache_WriteCCR0 (unsigned int val)
void XCache_EnableDCache (vùng int không dấu)
void XCache_DisableDCache (void)
void XCache_FlushDCacheLine (unsigned int adr)
void XCache_InvalidateDCacheLine (unsigned int adr)
void XCache_FlushDCacheRange (unsigned int adr, unsigned len)
void XCache_InvalidateDCacheRange (unsigned int adr, unsigned len)
void XCache_StoreDCacheLine (unsigned int adr)
void XCache_EnableICache (vùng int không dấu)
void XCache_DisableICache (void)
void XCache_InvalidateICache (void)
void XCache_InvalidateICacheLine (unsigned int adr)
void XCache_TouchICacheBlock (unsigned int adr)
```

Mô tả chức năng bộ nhớ đệm của bộ xử lý PowerPC 440

`void XCache_WriteCCR0 (unsigned int val)`

Ghi một giá trị số nguyên vào thanh ghi CCR0. Đây là một chuỗi mã mẫu. Trước khi ghi vào thanh ghi này, bộ đệm lệnh phải được kích hoạt để ngăn chặn việc khóa lỗi bộ xử lý. Sau khi ghi CCR0, bộ đệm lệnh có thể bị vô hiệu hóa nếu không cần thiết.

```
XCache_EnableICache (0x80000000) /* kích hoạt bộ nhớ cache hứa sẵn cho 256 đầu tiên
Vùng bộ nhớ MB */
XCache_WriteCCR0 (0x00100000) /* Tắt chương trình phát lệnh APU */
XCache_DisableICache () /* vô hiệu hóa bộ đệm chỉ sẵn */
```

`void XCache_EnableDCache (vùng int không dấu)`

Bật bộ đệm dữ liệu cho một vùng bộ nhớ cụ thể. Mỗi cặp bit liền kề trong tham số vùng đại diện cho 256 MB bộ nhớ. Đặt một trong hai bit trong cặp thành 1 sẽ cho phép lưu vào bộ nhớ đệm cho một vùng bộ nhớ 256 MB cụ thể.

Ví dụ:

- Giá trị 0x80000000 hoặc 0x40000000 hoặc 0xC0000000 sẽ bật bộ đệm dữ liệu cho 256 MB bộ nhớ đầu tiên (0 - 0x07FFFFFF).
- Giá trị 0x1 hoặc 0x2 hoặc 0x3 bật bộ đệm dữ liệu cho 256 MB bộ nhớ cuối cùng (0xF0000000 - 0xFFFFFFFF).

Lưu ý: nếu bạn đang di chuyển phần mềm từ thiết kế bộ xử lý PowerPC 405, hãy lưu ý rằng mỗi bit sẽ kích hoạt thêm 128 MB bộ nhớ cho bộ nhớ đệm.

```
void XCache_DisableDCache (void)
```

Tắt bộ đệm dữ liệu cho tất cả các vùng bộ nhớ.

```
void XCache_FlushDCacheLine (unsigned int adr)
```

Xả và làm mất hiệu lực dòng bộ đệm dữ liệu có chứa địa chỉ được chỉ định bởi adr tham số. Một lần truy cập dữ liệu tiếp theo vào địa chỉ này dẫn đến việc bỏ lỡ bộ nhớ cache và làm đầy dòng bộ nhớ cache.

```
void XCache_InvalidateDCacheLine (unsigned int adr)
```

Làm mất hiệu lực dòng bộ đệm dữ liệu chứa địa chỉ được chỉ định bởi tham số adr. Nếu dòng bộ đệm hiện bị bắn, nội dung đã sửa đổi sẽ bị mất và không được ghi vào bộ nhớ hệ thống. Một lần truy cập dữ liệu tiếp theo vào địa chỉ này dẫn đến việc bỏ lỡ bộ nhớ cache và làm đầy dòng bộ nhớ cache.

```
void XCache_FlushDCacheRange (unsigned int adr, unsigned len)
```

Xả và làm mất hiệu lực các dòng bộ đệm dữ liệu được mô tả bằng dài địa chỉ bắt đầu từ byte adr và len dài. Một lần truy cập dữ liệu tiếp theo vào bất kỳ địa chỉ nào trong phạm vi này dẫn đến việc bỏ lỡ bộ nhớ cache và làm đầy dòng bộ nhớ cache.

```
void XCache_InvalidateDCacheRange (unsigned int adr, unsigned len)
```

Làm mất hiệu lực các dòng trong bộ đệm dữ liệu được mô tả bởi dài địa chỉ bắt đầu từ byte dài adr và len. Nếu một dòng bộ nhớ cache hiện đang bị bắn, nội dung đã sửa đổi sẽ bị mất và không được ghi vào bộ nhớ hệ thống. Một lần truy cập dữ liệu tiếp theo vào bất kỳ địa chỉ nào trong phạm vi này dẫn đến việc bỏ lỡ bộ nhớ cache và làm đầy dòng bộ nhớ cache.

```
void XCache_StoreDCacheLine (unsigned int adr)
```

Lưu trữ trong bộ nhớ dòng bộ đệm dữ liệu có chứa địa chỉ được chỉ định bởi adr tham số. Một lần truy cập dữ liệu tiếp theo vào địa chỉ này dẫn đến một lần truy cập vào bộ nhớ cache nếu địa chỉ đã được lưu trữ trong bộ nhớ cache; nếu không, nó dẫn đến việc bỏ lỡ bộ nhớ cache và lắp đầy dòng bộ nhớ cache.

```
void XCache_EnableICache (vùng int không dấu)
```

Bật bộ đệm lệnh cho một vùng bộ nhớ cụ thể. Mỗi cặp bit liền kề trong tham số vùng đại diện cho 256 MB bộ nhớ. Đặt một trong hai bit trong cặp thành 1 sẽ cho phép lưu vào bộ nhớ đệm cho một vùng bộ nhớ 256 MB cụ thể. Ví dụ: giá trị 0x80000000 hoặc 0x40000000 hoặc 0xC0000000 sẽ bật lệnh cache cho 256 MB bộ nhớ đầu tiên (0 - 0xFFFFFFFF). Giá trị 0x1 hoặc 0x2 hoặc 0x3 bật bộ đệm lệnh cho 256 MB bộ nhớ cuối cùng (0xF0000000 - 0xFFFFFFFF).

Lưu ý: Nếu bạn đang di chuyển phần mềm từ PPC405, hãy lưu ý rằng mỗi bit sẽ kích hoạt thêm 128 MB bộ nhớ cho bộ nhớ đệm.

void XCache_DisableICache (void)

Tắt bộ đệm ẩn lệnh cho tất cả các vùng bộ nhớ.

void XCache_InvalidateICache (void)

Làm mất hiệu lực toàn bộ bộ đệm chỉ dẫn. Các hướng dẫn tiếp theo tạo ra các lần bỏ lỡ bộ nhớ cache và các dòng nạp vào bộ nhớ cache.

void XCache_InvalidateICacheLine (unsigned int adr)

Làm mất hiệu lực dòng bộ đệm chỉ dẫn có chứa địa chỉ được chỉ định bởi tham số adr .
Một lệnh tiếp theo tới địa chỉ này tạo ra lỗi bộ nhớ cache và một dòng lấp đầy bộ nhớ cache.

void XCache_TouchICacheBlock (unsigned int adr)

Tìm nạp khôi (dòng) bộ đệm chỉ lệnh vào bộ đệm, nếu địa chỉ đầu vào trả đến vùng lệnh có thể lưu trữ trong bộ đệm.

Xử lý ngoại lệ bộ xử lý PowerPC 440

Một API xử lý ngoại lệ được cung cấp trong Độc lập. Để có giải thích chuyên sâu về cách hoạt động của các ngoại lệ và ngắt trên bộ xử lý PowerPC 440, hãy tham khảo chương “Ngoại lệ và ngắt” trong Hướng dẫn Tham khảo Bộ xử lý PowerPC 440. Một liên kết đến tài liệu này được cung cấp trong “[Tài nguyên bổ sung](#)”, trang 1.

Lưu ý: Trình xử lý ngoại lệ không tự động đặt lại (vô hiệu hóa) bit kích hoạt trạng thái chờ trong MSR khi quay lại mã nguồn dùng. Bạn có thể buộc các trình xử lý ngoại lệ đặt lại bit Wait-Enable về 0 khi trả lại tất cả các ngoại lệ bằng cách biên dịch Độc lập với ký hiệu tiền xử lý PPC440_RESET_WE_ON_RFI được xác định. Bạn có thể thêm điều này vào cờ trình biên dịch được liên kết với các thư viện. Định nghĩa tiền xử lý này sẽ bật hành vi.

API xử lý ngoại lệ bao gồm một tập hợp các tệp `xvectors.S`, `xexception_l.c` và tệp tiêu đề tương ứng `xexception_l.h`.

Để biết thêm thông tin về xử lý gián đoạn, hãy xem Trợ giúp XPS và phụ lục “Quản lý ngắt” trong Sổ tay Tham khảo Công cụ Hệ thống Nhúng (có sẵn trong thư mục / doc của cài đặt EDK của bạn).

Tóm tắt chức năng xử lý ngoại lệ của bộ xử lý PowerPC 440

Bảng sau đây cung cấp tóm tắt về các chức năng xử lý ngoại lệ PowerPC 440.

Nhấp vào tên chức năng để chuyển đến phần mô tả.

```
void XExc_Init (void)
void XExc_RegisterHandler (Xuint8 ExceptionId, XExceptionHandler Handler, void * DataPtr)
void XExc_RemoveHandler (Xuint8 ExceptionId)
void XExc_mEnableExceptions (EnableMask)
void XExc_mDisableExceptions (DisableMask)
```

Mô tả chức năng xử lý ngoại lệ của bộ xử lý PowerPC 440

`void XExc_Init (void)`

Thiết lập bảng vectơ ngắt và đăng ký một chức năng “không làm gì cả” cho mỗi ngoại lệ. Hàm này không có tham số và không trả về giá trị.

Hàm này phải được gọi trước khi đăng ký bất kỳ trình xử lý ngoại lệ nào hoặc cho phép bất kỳ ngắt nào. Khi sử dụng API xử lý ngoại lệ, hàm này phải được gọi ở đầu quy trình `main()` của bạn.

```
void XExc_RegisterHandler (Xuint8 ExceptionId,
                           Trình xử lý XExceptionHandler, void * DataPtr)
```

Đăng ký một trình xử lý ngoại lệ cho một ngoại lệ cụ thể; không trả về giá trị. Tham khảo bảng sau để biết danh sách các loại ngoại lệ và giá trị của chúng. Các thông số như sau:

- ExceptionId có kiểu tham số Xuint8 và là ngoại lệ mà trình xử lý này nên được đăng ký.
Loại và các giá trị được xác định trong xexception_l.h
tập tin tiêu đề. [Bảng 7](#) liệt kê các loại ngoại lệ và các giá trị có thể có
- Handler là một tham số XExceptionHandler là con trỏ đến hàm xử lý ngoại lệ
- DataPtr có kiểu tham số void * và là giá trị người dùng được chuyển khi hàm xử lý được gọi

Bảng 7: Các loại và giá trị ngoại lệ đã đăng ký

Loại ngoại lệ	Giá trị
XEXC_ID_CRITICAL_INT	0
XEXC_ID_MACHINE_CHECK	1
XEXC_ID_DATA_STORAGE_INT	2
XEXC_ID_INSTRUCTION_STORAGE_INT	3
XEXC_ID_NON_CRITICAL_INT	4
XEXC_ID_ALIGNMENT_INT	5
XEXC_ID_PROGRAM_INT	6
XEXC_ID_FPU_UNAVAILABLE_INT	7
XEXC_ID_SYSTEM_CALL	8
XEXC_ID_APU_AVAILABLE	9
XEXC_ID_DEC_INT	10
XEXC_ID_FIT_INT	11
XEXC_ID_WATCHDOG_TIMER_INT	12
XEXC_ID_DATA_TLB_MISS_INT	13
XEXC_ID_INSTRUCTION_TLB_MISS_INT	14
XEXC_ID_DEBUG_INT	15

Hàm được cung cấp dưới dạng tham số Xử lý phải có nguyên mẫu hàm sau:

```
typedef void (* XExceptionHandler) (void * DataPtr);
```

Nguyên mẫu này được khai báo trong tập tin tiêu đề xexception_l.h.

Khi hàm xử lý ngoại lệ này được gọi, tham số DataPtr chứa cùng giá trị như bạn đã cung cấp khi đăng ký trình xử lý.

```
void XExc_RemoveHandler (Xuint8 ExceptionId)
```

Hủy đăng ký một hàm xử lý cho một ngoại lệ nhất định. Để biết các giá trị có thể có của tham số ExceptionId, hãy tham khảo [Bảng 7, trang 42](#).

void XExc_mEnableExceptions (EnableMask)

Bật ngoại lệ. Macro này phải được gọi sau khi khởi tạo bảng vectơ bằng hàm XExc_Init và đăng ký trình xử lý ngoại lệ với XExc_RegisterHandler hàm số.

Tham số EnableMask là một mặt nạ bit cho các trường hợp ngoại lệ được kích hoạt. EnableMask tham số có thể có các giá trị sau: XEXC_CRITICAL, XEXC_NON_CRITICAL, XEXC_DEBUG, XEXC_MACHINE_CHECK hoặc XEXC_ALL.

void XExc_mDisableExceptions (DisableMask)

Tắt các ngoại lệ. Tham số DisableMask là một mặt nạ bit để tắt các ngoại lệ. Tham số DisableMask có thể có các giá trị sau: XEXC_CRITICAL, XEXC_NON_CRITICAL, XEXC_DEBUG, XEXC_MACHINE_CHECK hoặc XEXC_ALL.

Hỗ trợ tệp bộ xử lý PowerPC 440

Hỗ trợ tệp được giới hạn cho các luồng stdin và stdout; do đó, các chức năng sau là không cần thiết:

- open () (trong open.c)
- close () (in close.c)
- fstat () (trong fstat.c)
- unlink () (trong unlink.c)
- lseek () (trong lseek.c)

Các tệp này được bao gồm để hoàn thiện và vì chúng được tham chiếu bởi thư viện C.

Mô tả chức năng hỗ trợ tệp bộ xử lý PowerPC 440**int read (int fd, char * buf, int nbytes)**

Hàm read () trong read.c đọc nbyte byte từ đầu vào chuẩn bằng cách gọi inbyte (). Nó chặn cho đến khi tất cả các ký tự có sẵn hoặc ký tự cuối dòng được đọc. Hàm read () trả về số ký tự được đọc. Tham số fd bị bỏ qua.

int write (int fd, char * buf, int nbytes)

Ghi nbyte byte vào đầu ra tiêu chuẩn bằng cách gọi outbyte (). Nó chặn cho đến khi tất cả các ký tự đã được viết. Hàm write () trả về số ký tự được viết. Fd _ tham số bị bỏ qua.

int isatty (int fd)

Báo cáo nếu một tệp được kết nối với một tty. Hàm này luôn trả về 1, Bởi vì chỉ có stdin và các luồng stdout được hỗ trợ.

```
int fcntl (int fd, int cmd, -long arg)
```

Một triển khai giả của fcntl, luôn trả về 0. fcntl nhằm điều khiển các bộ mô tả tệp theo lệnh được chỉ định bởi cmd. Vì Standalone không cung cấp hệ thống tệp nên chức năng này không được sử dụng.

Chức năng Errno của bộ xử lý PowerPC 440

```
int errno ( )
```

Trái về giá trị toàn cục của errno như được đặt bởi lệnh gọi thư viện C cuối cùng.

Quản lý bộ nhớ bộ xử lý PowerPC 440

```
char * sbrk (int nbytes)
```

Phân bổ nbyte của heap và trả về một con trỏ đến phần bộ nhớ đó. Hàm này được gọi từ các hàm cấp phát bộ nhớ của thư viện C.

Các chức năng của quy trình PowerPC 440

Các hàm getpid () trong getpid.c và kill () trong kill.c được bao gồm để hoàn thiện và vì chúng được tham chiếu bởi thư viện C.

Tệp bao gồm bộ xử lý PowerPC 440 cụ thể

Tệp bao gồm xreg440.h chứa các số đăng ký và các bit đăng ký cho bộ xử lý PowerPC 440.

Tệp bao gồm xpseudo-asm.h chứa các định nghĩa cho các hưng dẫn trình hợp dịch nội tuyến thường được sử dụng nhất, có sẵn dưới dạng macro. Chúng có thể rất hữu ích cho các tác vụ như thiết lập hoặc nhận các thanh ghi mục đích đặc biệt, đồng bộ hóa hoặc thao tác bộ nhớ cache.

Các hưng dẫn trình hợp dịch nội tuyến này có thể được sử dụng từ các trình điều khiển và ứng dụng người dùng được viết bằng C.

Chức năng thời gian của bộ xử lý PowerPC 440

Tệp xtime_l.c và tệp bao gồm xtime_l.h tư ơng ứng cung cấp quyền truy cập vào bộ đếm thời gian cơ sở 64-bit cũng như bộ định thời giảm dần, FIT và WDT bên trong lõi PowerPC 440. Bộ đếm cơ sở thời gian 64-bit tăng một ở mỗi chu kỳ bộ xử lý.

Tệp sleep.c và sleep.h tư ơng ứng bao gồm tệp thực hiện các chức năng ngủ. Các chức năng ngủ được thực hiện dưới dạng các vòng lặp bận.

Tóm tắt chức năng thời gian của bộ xử lý PowerPC 440 Các chức

năng thời gian của bộ xử lý PowerPC 440 được tóm tắt trong bảng sau. Nhấp vào tên chức năng để chuyển đến phần mô tả.

```
typedef unsigned long long XTime
void XTime_SetTime (XTime xtime)
void XTime_GetTime (XTime * xtime)
void XTime_TSRClearStatusBits (unsigned long Bitmask)
void XTime_DECSetInterval (khoảng dài không dấu); void
XTime_DECEnableInterrupt (void); void
XTime_DECDisableInterrupt(void) void
XTime_DECEnableAutoReload(void) void
XTime_DECDisableAutoReload(void) void
XTime_DECClearInterrupt(void) void
XTime_FITEnableInterrupt(void) void
XTime_FITDisableInterrupt(void) void
XTime_FITClearInterrupt(void) void
XTime_FITSetPeriod(unsigned long Period) void
XTime_WDTEnableInterrupt(void) void
XTime_WDTDisableInterrupt( void XTime_WDTClearInterrupt
(void) void XTime_WDTSetPeriod (unsigned long period)
void XTime_WDTResetControl (unsigned long ControlVal)
void XTime_WDTEnableNextWatchdog (void) void
XTime_WDTClearResetStatus (void) unsigned int nausleep
(unsigned int nausleep (unsigned int nausleep) const
struct timespec * rqtp, struct timespec * rmtp)
```

Mô tả chức năng thời gian của bộ xử lý PowerPC 440

typedef không dấu XTime dài dài

Kiểu XTime trong xtime_l.h đại diện cho thanh ghi Cơ sở thời gian. Cấu trúc này bao gồm các thanh ghi Time Base Low (TBL) và Time Base High (TBH), mỗi thanh ghi là một thanh ghi rộng 32 bit.

Định nghĩa của XTime như sau:

```
typedef không dấu dài dài XTime;
```

```
void XTime_SetTime (XTIME xtime)
```

Đặt thanh ghi cơ sở thời gian thành giá trị trong xtime.

```
void XTime_GetTime (XTIME * xtime)
```

Ghi giá trị hiện tại của thanh ghi cơ sở thời gian vào biến xtime.

```
void XTime_TSRClearStatusBits (Bitmask dài không dấu)
```

Xóa các bit trong thanh ghi trạng thái bộ định thời (TSR). Tham số Bitmask chỉ định các bit cần xóa. Giá trị 1 ở bất kỳ vị trí nào của tham số Bitmask sẽ xóa bit tương ứng trong TSR. Hàm này không trả về giá trị.

Thí dụ:

```
XTime_TSRClearStatusBits (XREG_TSR_CLEAR_ALL);
```

Bảng sau đây chứa các giá trị cho các tham số mặt nạ bit được chỉ định trong tệp tiêu đề xreg440.h.

Bảng 8: Giá trị tham số Bitmask

Tên	Giá trị	Sự mô tả
XREG_TSR_WDT_ENABLE_NEXT_WATCHDOG	0x80000000	Việc xóa bit này sẽ vô hiệu hóa sự kiện bộ đếm thời gian của cơ quan giám sát.
XREG_TSR_WDT_INTERRUPT_STATUS	0x40000000	Xóa bit Trạng thái ngắt của bộ định thời gian giám sát. Bit này được đặt sau khi xảy ra ngắt cơ quan giám sát.
XREG_TSR_WDT_RESET_STATUS_00	0x00000000	Xóa các bit Trạng thái Đặt lại Bộ hẹn giờ của Cơ quan giám sát. Sự kết hợp bit chỉ định kiểu thiết lập lại xảy ra do sự kiện bộ đếm thời gian của cơ quan giám sát.
XREG_TSR_WDT_RESET_STATUS_01	0x10000000	Xóa các bit Trạng thái Đặt lại Bộ định thời cho Cơ quan giám sát. Sự kết hợp bit chỉ định kiểu thiết lập lại xảy ra do sự kiện bộ đếm thời gian của cơ quan giám sát.
XREG_TSR_WDT_RESET_STATUS_10	0x20000000	Xóa các bit Trạng thái Đặt lại Bộ hẹn giờ của Cơ quan giám sát. Sự kết hợp bit chỉ định kiểu thiết lập lại xảy ra do sự kiện bộ đếm thời gian của cơ quan giám sát.
XREG_TSR_WDT_RESET_STATUS_11	0x30000000	Xóa các bit Trạng thái Đặt lại Bộ định thời của Cơ quan giám sát. Sự kết hợp bit chỉ định kiểu thiết lập lại xảy ra do sự kiện bộ đếm thời gian của cơ quan giám sát.
XREG_TSR_DEC_INTERRUPT_STATUS	0x08000000	Xóa bit trạng thái Decrementer (DEC). Bit này được đặt sau sự xuất hiện gián đoạn giảm dần.
XREG_TSR_FIT_INTERRUPT_STATUS	0x04000000	Xóa bit Trạng thái Bộ hẹn giờ Khoảng thời gian cố định (FIT). Bit này được đặt sau khi xảy ra ngắt FIT.
XREG_TSR_CLEAR_ALL	0xFFFFFFFF	Xóa tất cả các bit trong TSR. Sau khi Đặt lại, nội dung của TSR không được chỉ định. Sử dụng mặt nạ bit này để xóa tất cả các bit trong TSR.

```
void XTime_DECSetInterval ( khoảng thời gian dài không dấu);
```

Nạp một giá trị mới vào Thanh ghi Bộ giảm dần. Thanh ghi này là một bộ đếm giảm dần 32 bit có cùng tần số với thanh ghi cơ sở thời gian. Tùy thuộc vào cài đặt AutoReload, Decrementer được tự động tải lại với giá trị được ghi cuối cùng hoặc phải được tải lại theo cách thủ công. Hàm này không trả về giá trị.

Thí dụ:

```
XTime_DECSetInterval (0x00ffff);
```

```
void XTime_DECEnableInterrupt (void);
```

Cho phép tạo ngắt Decrementer. Một ngắt xảy ra khi thanh ghi DEC chứa giá trị 1, và sau đó được giảm dần. Hàm này không trả về giá trị.

XExc_Init () phải được gọi, trình xử lý ngắt Decrementer phải được đăng ký và các ngoại lệ phải được kích hoạt trước khi gọi hàm này.

Thí dụ:

```
XTime_DECEnableInterrupt ();
```

```
void XTime_DECDisableInterrupt (vô hiệu)
```

Tắt tạo ngắt Decrementer. Nó không trả về một giá trị.

Thí dụ:

```
XTime_DECDisableInterrupt ();
```

```
void XTime_DECEnableAutoReload (void)
```

Bật chức năng tự động tải lại của Thanh ghi Decrementer. Khi bật tính năng tự động tải lại, Thanh ghi Decrementer sẽ tự động được tải lại với giá trị cuối cùng được tải bằng cách gọi hàm XTime_DECSetInterval () khi Thanh ghi Decrementer chứa giá trị 1 và được giảm dần. Khi tính năng tự động tải lại được bật, Thanh ghi Decrementer không bao giờ chứa giá trị bằng 0. Hàm này không trả về giá trị.

Thí dụ:

```
XTime_DECEnableAutoReload ();
```

```
void XTime_DECDisableAutoReload (void)
```

Tắt tính năng tự động tải lại của Sổ đăng ký Decrementer. Khi tắt tự động tải lại, Bộ giảm dần giảm từ 1 xuống 0. Nếu chứa giá trị 0, nó sẽ ngừng giảm cho đến khi được tải bằng giá trị khác 0. Hàm này không trả về giá trị.

Thí dụ:

```
XTime_DECDisableAutoReload ();
```

void XTime_DECClearInterrupt (vô hiệu)

Xóa bit Trạng thái-ngắt của Decrementer trong Thanh ghi Trạng thái-Bộ định thời. Bit này chỉ định xem có xảy ra ngắt Decrementer hay không. Bạn phải gọi hàm này trong trình xử lý ngắt của mình để xóa bit Trạng thái, nếu không thì một ngắt Decrementer khác xảy ra ngay sau khi thoát khỏi chức năng xử lý ngắt. Hàm này không trả về giá trị. Gọi hàm này tương đương với gọi XTime_TSRClearStatusBits (XREG_TSR_DEC_INTERRUPT_STATUS).

Thí dụ:

```
XTime_DECClearInterrupt ();
```

void XTime_FITEnableInterrupt (void)

Bật ngắt bộ hẹn giờ khoảng thời gian cố định (FIT).

Thí dụ:

```
XTime_FITEnableInterrupt ();
```

void XTime_FITDisableInterrupt (void)

Vô hiệu hóa ngắt bộ hẹn giờ khoảng thời gian cố định (FIT).

Thí dụ:

```
XTime_FITDisableInterrupt ();
```

void XTime_FITClearInterrupt (void)

Xóa bit trạng thái ngắt của Bộ định thời gian cố định (FIT). Chức năng này tương đương với việc gọi XTime_TSRClearStatusBits (XREG_TSR_FIT_INTERRUPT_STATUS).

Thí dụ:

```
XTime_FITDisableInterrupt ();
```

void XTime_FITSetPeriod (Khoảng thời gian dài chia ký)

Đặt giá trị Khoảng thời gian cố định khoảng thời gian (FIT) . Giá trị này có thể là một trong những giá trị sau:

- XREG_TCR_FIT_PERIOD_11 (2 ^ 21 đồng hồ)
- XREG_TCR_FIT_PERIOD_10 (2 ^ 17 đồng hồ)
- XREG_TCR_FIT_PERIOD_01 (2 ^ 13 đồng hồ)
- XREG_TCR_FIT_PERIOD_00 (2 ^ 9 đồng hồ)

Các giá trị này được xác định trong xreg440.h

Thí dụ:

```
XTime_FITSetPeriod (XREG_TCR_FIT_PERIOD_11);
```

```
void XTime_WDTEnableInterrupt (void)
Bật ngắt bộ định thời gian giám sát (WDT).
```

Thí dụ:

```
XTime_WDTEnableInterrupt ();
```

```
void XTime_WDTDisableInterrupt (void)
Tắt các ngắt của Watchdog Timer (WDT).
```

Thí dụ:

```
XTime_WDTDisableInterrupt ();
```

```
void XTime_WDTClearInterrupt (vô hiệu)
```

Xóa bit trạng thái ngắt Watchdog Timer (WDT). Gọi hàm này tương ứng với gọi XTime_TSRClearStatusBits (XREG_TSR_WDT_INTERRUPT_STATUS).

Thí dụ:

```
XTime_WDTClearInterrupt ();
```

```
void XTime_WDTSetPeriod (Đầu chấm dài chia a ký)
```

Đặt khoảng thời gian cho sự kiện Bộ hẹn giờ cơ quan giám sát (WDT).

Thí dụ:

```
XTime_WDTSetPeriod (0x10000);
```

```
void XTime_WDTResetControl ( ControlVal dài không dấu)
```

Chỉ định loại thiết lập lại xảy ra do sự kiện Bộ định thời gian giám sát (WDT).

Giá trị điều khiển có thể là một trong những giá trị sau:

- XREG_WDT_RESET_CONTROL_11 (Đặt lại hệ thống)
- XREG_WDT_RESET_CONTROL_10 (Đặt lại chip)
- XREG_WDT_RESET_CONTROL_01 (đặt lại bộ xử lý)
- XREG_WDT_RESET_CONTROL_00 (không đặt lại)

Các giá trị này được xác định trong xreg440.h.

Thí dụ:

```
XTime_WDTResetControl (XREG_WDT_RESET_CONTROL_11);
```

```
void XTime_WDTEnableNextWatchdog (void)
```

Bật sự kiện Bộ hẹn giờ cơ quan giám sát (WDT).

Thí dụ:

```
XTime_WDTEnableNextWatchdog ();
```

```
void XTime_WDTClearResetStatus (void)
```

Xóa các bit trạng thái đặt lại Watchdog Timer (WDT).

Thí dụ:

```
XTime_WDTClearResetStatus ();
```

```
unsigned int usleep (unsigned int _useconds)
```

Trì hoãn việc thực thi chương trình trong `_useconds` micro giây. Nó luôn trả về số không. Chức năng này yêu cầu tần số bộ xử lý (tính bằng Hz) được xác định. Giá trị mặc định của biến này là 400 MHz. Giá trị này có thể được ghi đè trong tệp Thông số kỹ thuật phần mềm vi xử lý (MSS) như sau:

```
BỘ XỬ LÝ BEGIN
PARAMETER HW_INSTANCE = PPC440_i
PARAMETER DRIVER_NAME = cpu_ppc440
PARAMETER DRIVER_VER = 1,00.a
PARAMETER CORE_CLOCK_FREQ_HZ = 20000000
CHẤM DỨT
```

Tệp `xparameters.h` cũng có thể được sửa đổi với giá trị chính xác, như sau:

```
#define XPAR_CPU_PPC440_CORE_CLOCK_FREQ_HZ 20000000
```

```
unsigned int sleep (unsigned int _seconds)
```

Trì hoãn việc thực thi một chương trình bởi những gì được chỉ định trong `_seconds`. Nó luôn trả về 0. Hàm này yêu cầu tần số của bộ xử lý (tính bằng Hz) được xác định. Giá trị mặc định của biến này là 400 MHz.

Giá trị này có thể được ghi đè trong tệp Thông số kỹ thuật phần mềm vi xử lý (MSS) như sau:

```
BỘ XỬ LÝ BEGIN
PARAMETER HW_INSTANCE = PPC440_i
PARAMETER DRIVER_NAME = cpu_ppc440
PARAMETER DRIVER_VER = 1,00.a
PARAMETER CORE_CLOCK_FREQ_HZ = 20000000
CHẤM DỨT
```

Tệp `xparameters.h` cũng có thể được sửa đổi với giá trị chính xác, như sau:

```
#define XPAR_CPU_PPC440_CORE_CLOCK_FREQ_HZ 20000000
```

```
int nanosleep (const struct timespec * rqtp, struct timespec
* rmtp)
```

Hàm `nanosleep ()` trong `sleep.c` không được triển khai. Nó là một trình giữ chỗ để liên kết các ứng dụng với thư viện C và trả về số không.

Chương trình**Hồ sơ**

Standalone hỗ trợ lập hồ sơ chương trình kết hợp với các công cụ biên dịch GNU và Trình gỡ lỗi vi xử lý Xilinx® (XMD). Việc lập hồ sơ một chương trình đang chạy trên phần cứng (bo mạch) cung cấp thông tin chi tiết về việc thực thi chương trình và xác định nơi i dành thời gian thực thi. Tương tác của chương trình với bộ nhớ và các thiết bị ngoại vi khác có thể được ghi lại chính xác hơn.

Chương trình chạy trên mục tiêu phần cứng được cấu hình bằng phương pháp xâm nhập phần mềm. Trong phương pháp này, mã phần mềm cấu hình được thiết kế trong chương trình người dùng. Mã phần mềm tạo hồ sơ là một phần của thư viện libxil.a và được tạo khi bật tính năng lập hồ sơ xâm nhập phần mềm trong Độc lập. Để biết thêm chi tiết về quy trình Lập hồ sơ, hãy tham khảo phần "Thiết kế nhúng Hồ sơ" của Trợ giúp XPS.

Khi tùy chọn hồ sơ -pg được chỉ định cho trình biên dịch (mb-gcc hoặc powerpc-eabi gcc), các chức năng tạo hồ sơ được liên kết với ứng dụng thành hồ sơ tự động. Tệp thực thi được tạo chứa mã để tạo thông tin hồ sơ.

Khi thực hiện chương trình, chức năng cấu hình công cụ này lưu trữ thông tin trên phần cứng. Trình gỡ lỗi vi xử lý Xilinx (XMD) thu thập thông tin cấu hình và tạo tệp đầu ra, tệp này có thể được đọc bằng công cụ gprof GNU. Chức năng của chương trình vẫn không thay đổi như nó làm chậm quá trình thực thi.

Lưu ý: Các chức năng cấu hình không có bất kỳ API ứng dụng rõ ràng nào. Thư viện được liên kết do các lệnh gọi hồ sơ (_mcount) được giới thiệu bởi GCC để lập hồ sơ.

Yêu cầu về hồ sơ

- Hồ sơ xâm nhập phần mềm yêu cầu bộ nhớ để lưu trữ thông tin hồ sơ. Bạn có thể sử dụng bất kỳ bộ nhớ nào trong hệ thống để lập hồ sơ.
- Cần có bộ đếm thời gian cho địa chỉ lệnh lấy mẫu. Xps_timer hoặc opb_timer là bộ hẹn giờ hồ sơ được hỗ trợ. Đối với hệ thống bộ xử lý PowerPC, Bộ hẹn giờ ngắn có thể lập trình (PIT) cũng có thể được sử dụng làm bộ hẹn giờ cấu hình.

Chức năng lập hồ sơ**_profile_init**

Được gọi trước hàm main () của ứng dụng. Khởi tạo quy trình bộ hẹn giờ cấu hình và đăng ký bộ xử lý bộ hẹn giờ tương ứng, dựa trên bộ hẹn giờ được sử dụng, kết nối với bộ xử lý và khởi động bộ hẹn giờ. Quy trình Tcl của thư viện Độc lập xác định loại bộ đếm thời gian và kết nối với bộ xử lý, sau đó tạo #defines trong tệp profile_config.h.

Tham khảo chương "Định nghĩa Thư viện Bộ vi xử lý (MLD)" trong Sổ tay Tham khảo Công cụ Hệ thống Nhúng, có sẵn trong thư mục cài đặt. Một liên kết đến tài liệu này cũng được cung cấp trong "Tài nguyên bổ sung", trang 1.

_mcount

Được gọi bằng hàm _mcount, được chèn vào mỗi lần bắt đầu hàm bởi gcc. Ghi lại thông tin người gọi và người gọi (Địa chỉ lệnh), được sử dụng để tạo thông tin đồ thị cuộc gọi.

_profile_intr_handler

Bộ xử lý ngắn cho bộ định thời gian định hình. Bộ hẹn giờ được đặt để lấy mẫu ứng dụng đang thực thi cho các giá trị PC ở những khoảng thời gian cố định và tăng số lượng Thùng. Hàm này được sử dụng để tạo thông tin biểu đồ.

Định cấu hình Độc lập

Bạn có thể định cấu hình Độc lập bằng hộp thoại Cài đặt Nền tảng Phần mềm.

Bảng sau liệt kê các thông số có thể định cấu hình cho Độc lập.

Bảng 9: Thông số cấu hình

Tham số	Loại hình	Mặc định Giá trị	Sự mô tả
enable_sw_intrusive_profiling	Bool	false	Cho phép lập hồ sơ xâm nhập phần mềm chức năng. Chọn true để bật.
profile_timer	Ngoại vi Ví dụ	không	Chỉ định bộ đếm thời gian để sử dụng cho việc lập hồ sơ. Chọn xps_timer hoặc opb_timer từ danh sách các trường hợp được hiển thị. Đối với hệ thống PowerPC, chọn không có để sử dụng bộ hẹn giờ PIT tích hợp.
stdin	Ngoại vi Ví dụ	không	Chỉ định thiết bị ngoại vi STDIN từ danh sách thả xuống
stdout	Ngoại vi Ví dụ	không	Chỉ định thiết bị ngoại vi STDOUT từ danh sách thả xuống.
predecode_fpu_exception	Bool	sai	Tham số này chỉ hợp lệ cho bộ xử lý MicroBlaze khi các ngoại lệ FPU được bật trong phần cứng. Đặt điều này thành true sẽ bao gồm mã bổ sung để giải mã và lưu trữ các toán hạng lệnh FP bị lỗi trong các biến toàn cục.



Xilkernel (v4.00.a)

UG 646 ngày 16 tháng 9 năm 2009

Bản tóm tắt

Tài liệu này mô tả Xilkernel, một hạt nhân cho các bộ xử lý nhúng Xilinx®. Tài liệu bao gồm các phần sau:

- “[Tổng quan](#)”
- “[Tại sao lại sử dụng Kernel?](#)”
- “[Các tính năng chính](#)”
- “[Tài nguyên bổ sung](#)”
- “[Tổ chức Xilkernel](#)”
- “[Xây dựng ứng dụng Xilkernel](#)”
- “[Mô hình quy trình Xilkernel](#)”
- “[Mô hình lập lịch trình Xilkernel](#)”
- “[Giao diện POSIX](#)”
- “[Chức năng Xilkernel](#)”
- “[API Xilkernel](#)”
- “[Xử lý giản đoạn](#)”
- “[Xử lý ngoại lệ](#)”
- “[Bảo vệ bộ nhớ](#)”
- “[Giao diện khác](#)”
- “[Yêu cầu phần cứng](#)”
- “[Khởi tạo hệ thống](#)”
- “[An toàn chuỗi và sự hấp dẫn lại](#)”
- “[Hạn chế](#)”
- “[Tùy chỉnh hạt nhân](#)”
- “[Gỡ lỗi Xilkernel](#)”
- “[Dấu chân bộ nhớ](#)”
- “[Tổ chức tệp Xilkernel](#)”
- “[Các tính năng không được dùng nữa](#)”

© 2009 Xilinx, Inc. XILINX, biểu trưng Xilinx, Virtex, Spartan, ISE và các nhãn hiệu được chỉ định khác có trong tài liệu này là nhãn hiệu của Xilinx tại Hoa Kỳ và các quốc gia khác. Tất cả các nhãn hiệu khác là tài sản của chủ sở hữu tương ứng của họ.

Xilinx sẽ tiết lộ hướng dẫn sử dụng, hướng dẫn sử dụng, ghi chú phát hành và / hoặc thông số kỹ thuật này ("Tài liệu") cho bạn chỉ để sử dụng trong việc phát triển các thiết kế để vận hành với các thiết bị phần cứng của Xilinx. Bạn không được phép sao chép, phân phối, tái xuất bản, tái xuất, hiển thị, đăng hoặc truyền Tài liệu dưới bất kỳ hình thức nào hoặc bằng bất kỳ phương tiện nào bao gồm như ng không giới hạn ở Xilinx. Xilinx từ chối rõ ràng bất kỳ trách nhiệm pháp lý nào phát sinh từ việc bạn sử dụng Tài liệu. Xilinx bảo lưu quyền, theo quyết định riêng của mình, thay đổi Tài liệu mà không cần thông báo bất kỳ lúc nào. Xilinx không có nghĩa vụ phải sửa bất kỳ lỗi nào có trong Tài liệu hoặc thông báo cho bạn về bất kỳ chỉnh sửa hoặc cập nhật nào. Xilinx từ chối rõ ràng bất kỳ trách nhiệm pháp lý nào liên quan đến hỗ trợ kỹ thuật hoặc hỗ trợ có thể được cung cấp cho bạn liên quan đến Thông tin.

TÀI LIỆU ĐƯỢC CÔNG BỐ CHO BẠN "NGUYÊN TRẠNG" KHÔNG ĐƯỢC BẢO HÀNH BẤT KỲ HÌNH THỨC NÀO. XILINX KHÔNG CÓ BẢO ĐÁM KHÁC, DÙ THẾ HIỆN, NGƯ Ý HOẶC TRUYỀN NGHĨA, LIÊN QUAN ĐẾN TÀI LIỆU, BAO GỒM BẤT KỲ BẢO ĐÁM NÀO VỀ TÍNH KHẢ NĂNG, PHÙ HỢP VỚI MỤC ĐÍCH CỤ THỂ HOẶC KHÔNG ĐÁM BẢO QUYỀN CỦA BÊN THỨ BA. TRONG MỌI TRƯỜNG HỢP XILINX SẼ CHỊU TRÁCH NHIỆM PHÁP LÝ ĐỐI VỚI BẤT KỲ THIẾT HẠI HẬU QUẢ, ÁN ĐỘ, BẤT CỨ, ĐẶC BIỆT HOẶC BẤT CỨ SỰ CỐ NÀO, BAO GỒM BẤT KỲ VIỆC MẤT DỮ LIỆU HOẶC MẤT LỢI NHUẬN NÀO PHÁT SINH TỪ VIỆC BẠN SỬ DỤNG TÀI LIỆU.

Tổng quan

Xilkernel là một hạt nhân mô-đun nhỏ, mạnh mẽ và mạnh mẽ. Nó được tích hợp cao với khung Platform Studio và là thư viện phần mềm miễn phí mà bạn nhận được cùng với Bộ phát triển nhúng Xilinx (EDK). Xilkernel:

- Cho phép mức độ tùy biến rất cao, cho phép bạn điều chỉnh hạt nhân đến mức tối ưu cả về kích thước và chức năng.
- Hỗ trợ các tính năng cốt lõi cần thiết trong một hạt nhân nhúng nhẹ, với API POSIX.
- Hoạt động trên các bộ vi xử lý MicroBlaze™, PowerPC® 405 và PowerPC 440.

Các dịch vụ IPC của Xilkernel có thể được sử dụng để triển khai các dịch vụ cấp cao hơn (chẳng hạn như mạng, video và âm thanh) và sau đó chạy các ứng dụng sử dụng các dịch vụ này.

Tại sao lại sử dụng Kernel?

Sau đây là một số yêu tố quyết định có thể ảnh hưởng đến lựa chọn sử dụng hạt nhân làm nền tảng phần mềm cho dự án ứng dụng tiếp theo của bạn:

- Các ứng dụng điều khiển nhúng điển hình bao gồm các tác vụ khác nhau cần được thực hiện theo một trình tự hoặc lịch trình cụ thể. Khi số lượng nhiệm vụ kiểm soát liên quan tăng lên, việc sắp xếp các nhiệm vụ con theo cách thủ công và chia sẻ thời gian sẽ trở nên khó khăn hơn. Khả năng đáp ứng và khả năng của một ứng dụng như vậy giảm đáng kể khi độ phức tạp tăng lên.
- Việc chia nhỏ các tác vụ thành các ứng dụng riêng lẻ và triển khai chúng trên một hệ điều hành (OS) trực quan hơn nhiều.
- Một hạt nhân cho phép bạn viết mã ở mức trừu tượng, thay vì ở mức nhỏ, vì mã độc lập cấp bộ điều khiển.
- Nhiều ứng dụng thông thường và kế thừa dựa vào các dịch vụ hệ điều hành như hệ thống tệp, quản lý thời gian, v.v. Xilkernel là một thư viện mỏng cung cấp các dịch vụ thiết yếu này. Việc chuyển hoặc sử dụng các thư viện mã nguồn mở và phổ biến (chẳng hạn như đồ họa hoặc giao thức mạng) cũng có thể yêu cầu một số dạng của các dịch vụ hệ điều hành này.

Các tính năng chính

Xilkernel bao gồm các tính năng chính sau:

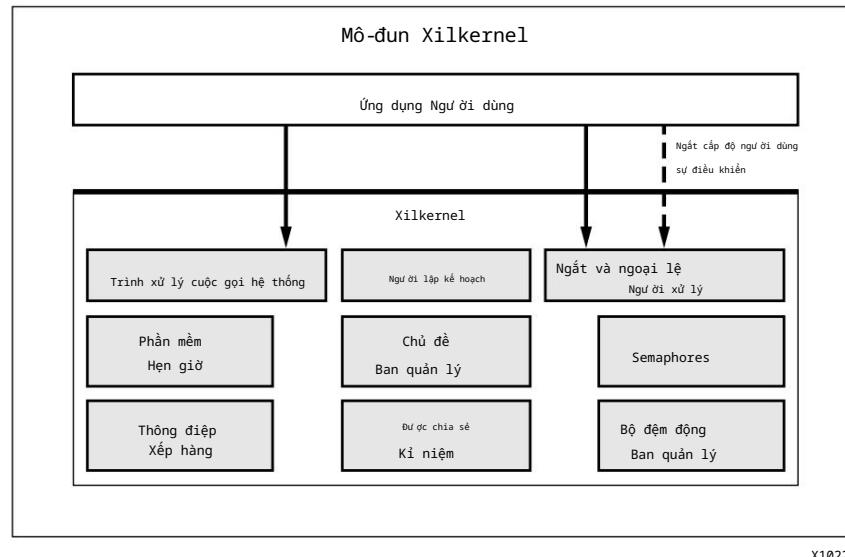
- Khả năng mở rộng cao vào một hệ thống nhất định thông qua việc bao gồm hoặc loại trừ chức năng như yêu cầu.
- Hoàn thành cấu hình hạt nhân và triển khai trong vòng vài phút từ bên trong Nền tảng Phòng thu.
- Tính mạnh mẽ của hạt nhân: các lệnh gọi hệ thống được bảo vệ bởi các kiểm tra tính hợp lệ của tham số và sự phù hợp trả về mã lỗi POSIX.
- Một API POSIX nhằm mục tiêu các hạt nhân nhúng, giành được các tính năng của hạt nhân cốt lõi như :
 - Các chủ đề với lịch trình ưu tiên vòng tròn hoặc nghiêm ngặt.
 - Dịch vụ đồng bộ hóa: khóa semaphores và mutex.
 - Dịch vụ IPC: hàng đợi tin nhắn và bộ nhớ dùng chung.
 - Cấp phát bộ nhớ vùng đệm động.
 - Phần mềm hẹn giờ.
 - Xử lý ngắt mức người dùng.
- Tạo luồng tĩnh khởi động với hạt nhân.
- Giao diện cuộc gọi hệ thống tới hạt nhân.
- Xử lý ngoại lệ cho bộ xử lý MicroBlaze.
- Bảo vệ bộ nhớ bằng cách sử dụng các tính năng Đơn vị Quản lý (Bảo vệ) Bộ nhớ MicroBlaze (MMU) khi có sẵn.

Thêm vào Tài nguyên

- Sổ tay Tham khảo Công cụ Hệ thống Nhúng:
http://www.xilinx.com/ise/embedded/edk_docs.htm
- Mẫu thiết kế dựa trên Xilkernel:
http://www.xilinx.com/ise/embedded/edk_examples.htm

Xilkernel Cơ quan

Kernel được thiết kế theo mô-đun cao. Bạn có thể chọn và tùy chỉnh các mô-đun nhân cần thiết cho ứng dụng trong tầm tay. Tùy chỉnh nhân được thảo luận chi tiết trong phần "[Tùy chỉnh nhân](#)" (1). Hình sau cho thấy các mô-đun khác nhau của Xilkernel:



X10226

Hình 1: Mô-đun Xilkernel

Tòa nhà Xilkernel Các Ứng dụng

Xilkernel được tổ chức dưới dạng một thư viện các hàm nhân. Điều này dẫn đến một mô hình liên kết nhân đơn giản. Để xây dựng Xilkernel, bạn phải đưa Xilkernel vào nền tảng phần mềm của mình, định cấu hình nó một cách thích hợp và chạy Libgen để tạo thư viện Xilkernel. Nguồn ứng dụng của bạn có thể được chỉnh sửa và phát triển riêng biệt, hoặc dưới dạng một dự án ứng dụng phần mềm từ bên trong XPS. Sau khi bạn đã phát triển ứng dụng của mình, bạn phải liên kết với thư viện Xilkernel, do đó kéo tất cả các chức năng của hạt nhân vào để xây dựng hình ảnh hạt nhân cuối cùng. Thư viện Xilkernel được tạo dưới dạng libxilkernel.a. [Hình 2, trang 5](#) cho thấy luồng phát triển này.

Bên trong, Xilkernel cũng hỗ trợ phương pháp liên kết giống hệ điều hành truyền thống, mạnh mẽ hơn nhiều và các tệp thực thi riêng biệt. Các hệ điều hành thông thường có hình ảnh hạt nhân là một tệp riêng biệt và mỗi ứng dụng thực thi trên hạt nhân là một tệp riêng biệt. Tuy nhiên, Xilinx khuyên bạn nên sử dụng chế độ liên kết thư viện đơn giản hơn và thanh lịch hơn. Chế độ này được hỗ trợ đầy đủ trong Platform Studio và mang lại sự dễ sử dụng tối đa. Nó cũng là chế độ ưu tiên để gỡ lỗi, tải xuống và tải khởi động. Chế độ thực thi riêng biệt chỉ được yêu cầu bởi những người có yêu cầu nâng cao ở dạng tệp thực thi riêng biệt.

Chế độ thực thi riêng biệt và những lưu ý của nó được ghi lại trong phần "[Tính năng không được dùng nữa](#)", [trang 55](#).

1. Một số tính năng này có thể không được hỗ trợ đầy đủ trong một bản phát hành nhất định của Xilkernel.

Sau đây là các bước cho chế độ liên kết hạt nhân của quá trình phát triển ứng dụng:

- Các tệp C nguồn ứng dụng nên bao gồm tệp xmk.h làm tệp đầu tiên trong số các tệp khác. Vì thí dụ:

```
#include "xmk.h"
```

Việc xác định cờ này cung cấp các định nghĩa và khai báo nhất định từ GNU bao gồm các tệp được yêu cầu bởi Xilkernel và các ứng dụng.

- Dự án phần mềm ứng dụng của bạn liên kết với thư viện libxil.a. Thư viện này chứa các hàm nhân thực tế được tạo ra. Ứng dụng của bạn liên kết với điều này và tạo thành hạt nhân cuối cùng và hình ảnh ứng dụng.

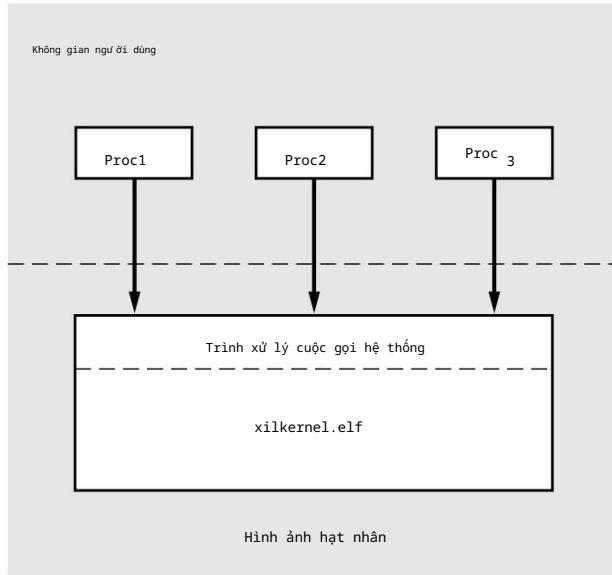
• Xilkernel chịu trách nhiệm về tất cả các ngắt mức đầu tiên và xử lý ngoại lệ trên cả Bộ vi xử lý MicroBlaze và PowerPC. Do đó, bạn không nên trực tiếp cố gắng sử dụng bất kỳ phươn pháp xử lý ngắt nào được ghi lại cho các chương trình độc lập. Thay vào đó, hãy tham khảo phần về xử lý ngắt để biết cách xử lý ngắt và ngoại lệ cấp nguồn dùng.

- Bạn có thể kiểm soát bản đồ bộ nhớ của hạt nhân bằng cách sử dụng tính năng tập lệnh trình liên kết của dự án ứng dụng phần mềm cuối cùng liên kết với hạt nhân. Tạo tập lệnh trình liên kết tự động giúp bạn ở đây.

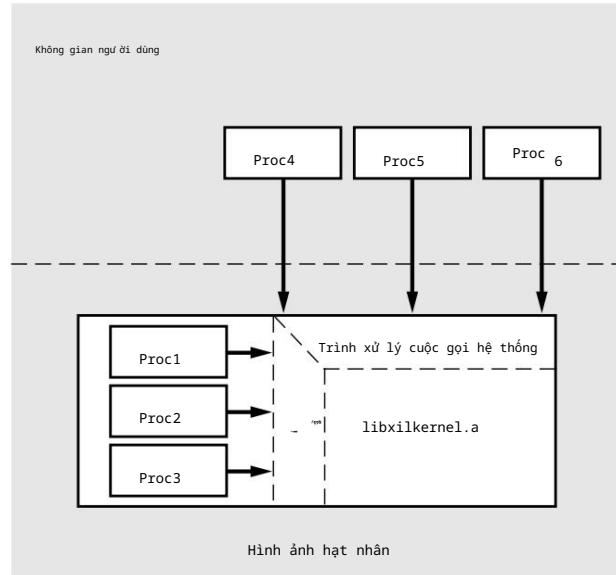
- Ứng dụng của bạn phải cung cấp một main () là điểm bắt đầu thực thi cho ảnh hạt nhân của bạn. Bên trong main (), bạn có thể thực hiện bất kỳ quá trình khởi tạo và thiết lập nào mà bạn cần. Kernel vẫn chưa khởi động và không hoạt động. Tại thời điểm thiết lập ứng dụng của bạn hoàn tất và bạn muốn hạt nhân khởi động, bạn phải gọi xilkernel_main () khởi động hạt nhân, cho phép ngắt và chuyển quyền kiểm soát đến các quy trình ứng dụng của bạn, như đã được định cấu hình. Một số tính năng cấp hệ thống có thể cần được bật trước khi gọi xilkernel_main (). Đây thường là các tính năng ở trạng thái máy như kích hoạt bộ nhớ cache, kích hoạt ngoại lệ phần cứng phải “luôn BẬT” ngay cả khi chuyển đổi ngữ cảnh từ ứng dụng này sang ứng dụng khác. Đảm bảo rằng bạn thiết lập trạng thái hệ thống như vậy trước khi gọi xilkernel_main (). Ngoài ra, bạn không được tự ý sửa đổi trạng thái hệ thống như vậy trong chuỗi ứng dụng của mình. Nếu một chuyển đổi ngữ cảnh xảy ra khi trạng thái hệ thống được sửa đổi, nó có thể dẫn đến các luồng tiếp theo thực thi mà trạng thái đó không được kích hoạt; do đó, bạn phải khóa các công tắc và ngắt ngữ cảnh trước khi sửa đổi trạng thái như vậy.

Lưu ý: Tập lệnh trình liên kết của bạn phải biết các yêu cầu của hạt nhân. Ví dụ: trên hệ thống PowerPC 405, có phần `a.vectors` chứa tất cả mã xử lý ngoại lệ cấp đầu tiên. Tập lệnh trình liên kết cuối cùng của bạn phải đảm bảo rằng phần này nhận được chỉ định bộ nhớ thích hợp.

Kịch bản chế độ thực thi riêng biệt thuần túy



Kịch bản chế độ thực thi gói hạt nhân



X10128

Hình 2: Luồng phát triển Xilkernel

Xilkernel

Mô hình quy trình

Các đơn vị thực thi trong Xilkernel được gọi là ngữ cảnh quy trình. Lập lịch trình được thực hiện ở cấp bối cảnh quy trình. Không có khái niệm về các nhóm luồng kết hợp để tạo thành, cái được gọi là quy trình. Thay vào đó, tất cả các chủ đề là ngang hàng và cạnh tranh bình đẳng về tài nguyên. API luồng POSIX là giao diện chính mà người dùng có thể nhìn thấy đối với các ngữ cảnh quy trình này. Có một số giao diện bổ sung hữu ích khác được cung cấp, không phải là một phần của POSIX. Các giao diện cho phép tạo, hủy và thao tác các luồng ứng dụng đã tạo. Các giao diện thực tế được mô tả chi tiết trong "[API Xilkernel](#)". Chủ đề được thao tác với các định danh chủ đề. Bối cảnh quy trình cơ bản được xác định bằng mã định danh quy trình `pid_t`.

Xilkernel**Lập lịch trình****Người mẫu**

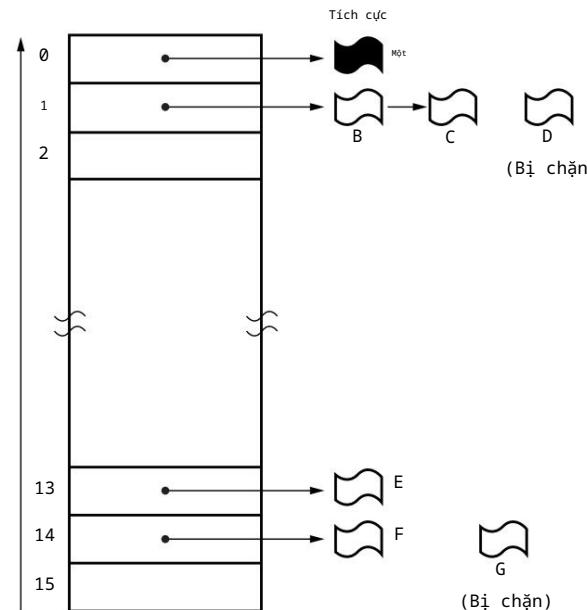
Xilkernel hỗ trợ lập lịch ưu tiên, định hướng ưu tiên với phân chia thời gian (SCHED_PRIO) hoặc lập lịch quay vòng đơn giản (SCHED_RR). Đây là chính sách lập lịch toàn cầu và không thể thay đổi trên cơ sở từng luồng. Điều này phải được cấu hình tĩnh tại thời điểm tạo hạt nhân.

Trong SCHED_RR, có một hàng đợi sẵn sàng duy nhất và mỗi ngữ cảnh quy trình thực thi trong một lát thời gian đã định cấu hình trước khi chuyển giao thực thi cho ngữ cảnh quy trình tiếp theo trong hàng đợi.

Trong SCHED_PRIO có bao nhiêu hàng đợi sẵn sàng cũng như có mức độ ưu tiên. Mức độ ưu tiên 0 là mức độ ưu tiên cao nhất trong hệ thống và các giá trị cao hơn có nghĩa là mức độ ưu tiên thấp hơn.

Như thể hiện trong hình sau, tiến trình ở đầu hàng đợi sẵn sàng có mức ưu tiên cao nhất luôn được lên lịch để thực thi tiếp theo. Trong cùng một mức độ ưu tiên, việc lập lịch biểu diễn ra theo vòng tròn và thời gian cắt ngắn. Nếu cấp hàng đợi sẵn sàng trống, nó sẽ bị bỏ qua và cấp độ hàng đợi sẵn sàng tiếp theo được kiểm tra để tìm các quy trình có thể lập lịch. Các quy trình bị chặn nằm ngoài hàng đợi sẵn sàng của chúng và nằm trong hàng đợi thích hợp của chúng. Số lượng mức độ ưu tiên có thể được định cấu hình cho SCHED_PRIO.

Đối với cả hai mô hình lập lịch, độ dài của hàng đợi sẵn sàng cũng có thể được định cấu hình. Nếu có hàng đợi bên trong hạt nhân (trong semaphores, hàng đợi tin nhắn, v.v.), chúng được cấu hình như hàng đợi ưu tiên nếu chế độ lập lịch là SCHED_PRIO. Nếu không, chúng được định cấu hình dưới dạng hàng đợi nhập trứớc - xuất trứớc (FIFO) đơn giản.



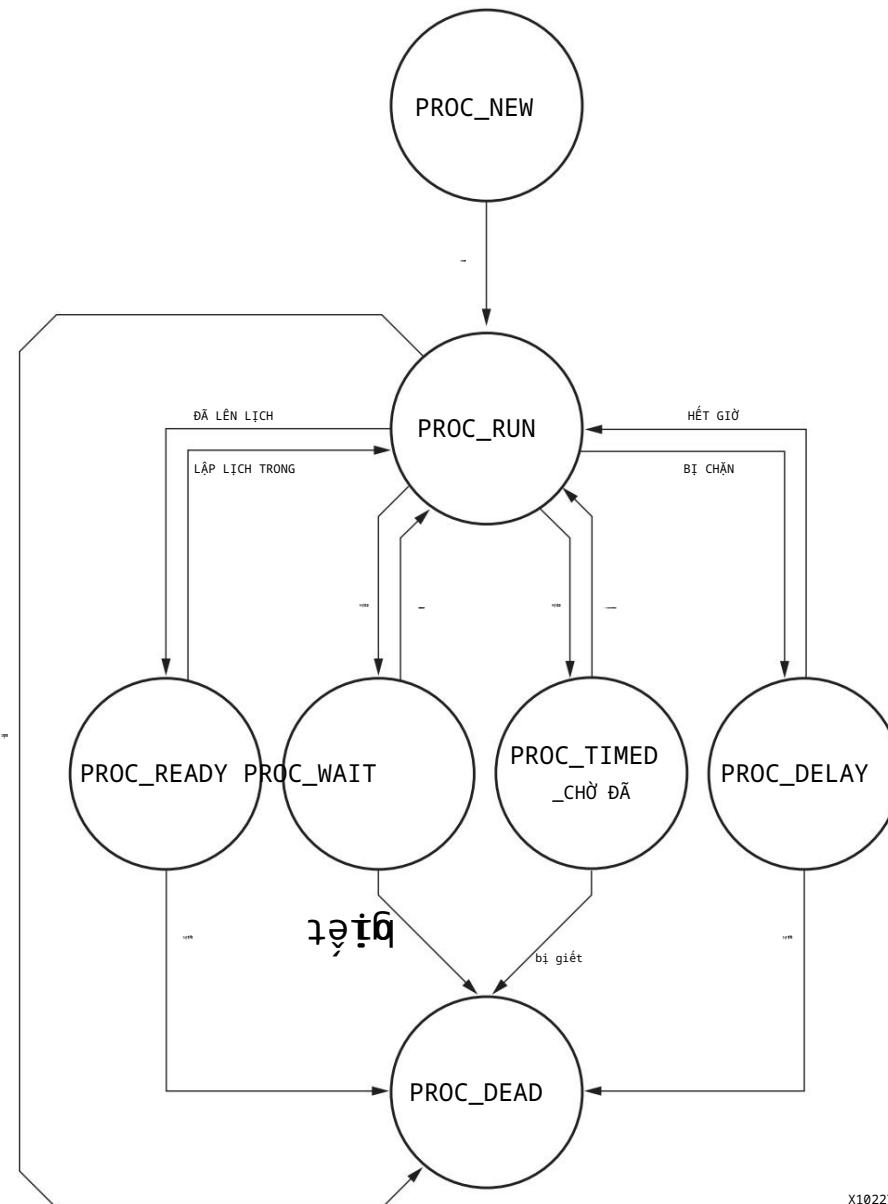
X10132

Hình 3: Lập lịch trình theo hướng ưu tiên

Mỗi ngõ cảnh quy trình ở bất kỳ trạng thái nào trong số sáu trạng thái sau:

- PROC_NEW - Một quy trình mới được tạo.
- PROC_READY - Một quá trình sẵn sàng để thực thi.
- PROC_RUN - Một tiến trình đang chạy.
- PROC_WAIT - Một quá trình bị chặn trên một tài nguyên.
- PROC_DELAY - Một quá trình đang chờ hết thời gian.
- PROC_TIMED_WAIT - Quá trình bị chặn trên một tài nguyên và có một hết giờ.

Khi một tiến trình kết thúc, nó đi vào trạng thái chết được gọi là PROC_DEAD. Biểu đồ trạng thái ngũ cảnh quy trình được hiển thị trong hình sau.



Hình 4: Các trạng thái bối cảnh quy trình

Giao diện POSIX Xilkernel cung cấp giao diện POSIX cho nhân. Không phải tất cả các khái niệm và giao diện đều được định nghĩa của POSIX đều có sẵn. Một tập hợp con bao gồm các giao diện và khái niệm hữu ích nhất được triển khai. Các chương trình Xilkernel có thể chạy gần như tương đương trên hệ điều hành máy tính để bàn của bạn, như Linux hoặc SunOS. Điều này giúp cho việc phát triển ứng dụng dễ dàng, tính di động và hỗ trợ phần mềm kế thừa. Mô hình lập trình hấp dẫn những người đã làm việc trên các giao diện POSIX tương đương trên các hệ điều hành truyền thống. Đối với những giao diện đã được cung cấp, POSIX được tuân thủ nghiêm ngặt trong hầu hết các trường hợp. Đối với các trường hợp có sự khác biệt, sự khác biệt được xác định rõ ràng. Tham khảo "["Xilkernel API"](#), để biết các giao diện thực tế và mô tả của chúng.

Xilkernel Chức năng

Nhập vào một mục bên dưới để xem tóm tắt và mô tả chức năng cho:

- Quản lý chuỗi
- Semaphores
- Hàng đợi tin nhắn
- Bộ nhớ dùng chung
- Mutex Locks
- Quản lý bộ nhớ đệm động
- Bộ hẹn giờ phần mềm
- API xử lý ngắn mức người dùng
- Quản lý quy trình ELF (Không được dùng nữa)

API Xilkernel

Quản lý chuỗi

Xilkernel hỗ trợ API luồng POSIX cơ bản. Việc tạo và thao tác luồng được thực hiện trong ký hiệu POSIX tiêu chuẩn. Chủ đề được xác định bằng một mã định danh chủ đề duy nhất. Định danh chủ đề thuộc loại pthread_t. Mã định danh luồng này xác định duy nhất một luồng cho một hoạt động. Các luồng được tạo trong hệ thống có một trình bao bọc hạt nhân mà chúng trả lại quyền kiểm soát khi chúng kết thúc. Do đó, một chức năng thoát cụ thể không được yêu cầu ở cuối mã của luồng.

Ngăn xếp luồng được cấp phát tự động thay mặt cho luồng từ một nhóm bộ nhớ Ký hiệu Bắt đầu Khối (BSS) được cấp phát tĩnh dựa trên số luồng hệ thống tối đa. Bạn cũng có thể gán một phần bộ nhớ tùy chỉnh làm ngăn xếp cho mỗi luồng để tạo động.

Toàn bộ mô-đun luồng là tùy chọn và có thể được cấu hình trong hoặc ngoài như một phần của đặc điểm kỹ thuật phần mềm. Xem "[Định cấu hình quản lý chuỗi](#)", trang 50 để biết thêm chi tiết về cách tùy chỉnh mô-đun này.

Tóm tắt chức năng quản lý luồng

Danh sách sau đây là bản tóm tắt được liên kết của các chức năng quản lý luồng trong Xilkernel.
Bấm vào một chức năng để xem mô tả chi tiết.

```
int pthread_create (pthread_t thread, pthread_attr_t * att, void * (* start_func) (void *), void *
param) void pthread_exit (void * value_ptr) int pthread_join (pthread_t thread, void ** value_ptr)
pthread_t pthread_self (void) (pthread_t target) int pthread_equal (pthread_t t1, pthread_t t2) int
pthread_getschedparam (pthread_t thread, int * policy, struct Sched_param * param) int
pthread_setschedparam (pthread_t thread, int policy, const struct _tread_param * param) int
pthread_attr_destroy (pthread_attr_t* attr) int pthread_attr_setdetachstate(pthread_attr_t* attr,
int dstate) int pthread_attr_getdetachstate(pthread_attr_t* attr, int *dstate) int
pthread_attr_setschedparam(pthread_attr_t* attr, struct sched_param *schedpar) int
pthread_attr_getschedparam(pthread_attr_t* attr, struct sched_param* Schedpar) int pthread_attr_setstack
(const pthread_attr_t * attr, _ void * stackaddr, size_t stacksize) int pthread_attr_getstack (const
pthread_attr_t * a_ttr, void ** stackaddr, size_t * stacksize) pid_t get_currentPID (void) int kill
(pid_tpid) int process_status (pid_t pid, p_stat * ps) int xmk_add_static_thread (void * (*
start_routine) (void *), int Sched_prighte (vô hiệu)
```

Mô tả chức năng quản lý luồng

Các mô tả sau đây là số nhận dạng giao diện quản lý luồng.

```
int pthread_create (pthread_t thread, pthread_attr_t * att,
                   void * (* start_func) (void *), void * param)
```

Thông số

luồng là vị trí để lưu trữ định danh của luồng đã tạo.

Lợi nhuận

attr là con trỏ đến cấu trúc thuộc tính tạo luồng.

start_func là địa chỉ bắt đầu của hàm mà từ đó luồng cần thực thi.

param là đối số con trỏ đến hàm luồng.

Sự mô tả

0 và mã định danh luồng của luồng đã tạo trong * luồng, khi thành công.

-1 nếu luồng để cập đến một vị trí không hợp lệ.

EINVAL nếu attr để cập đến các thuộc tính không hợp lệ.

EAGAIN nếu tài nguyên không có sẵn để tạo chuỗi.

Bao gồm

xmk.h, pthread.h

void pthread_exit (void * value_ptr)

Thông số

value_ptr là một con trỏ đến giá trị trả về của luồng.

Lợi nhuận

Không có.

Sự mô tả

Hàm pthread_exit () kết thúc chuỗi đang gọi và cung cấp giá trị value_ptr cho bất kỳ kết nối thành công nào với chuỗi kết thúc. Kết thúc luồng giải phóng tài nguyên ngữ cảnh quy trình bao gồm, nhưng không giới hạn, bộ nhớ và các thuộc tính. Một lệnh gọi ngầm định tới pthread_exit () được thực hiện khi một luồng quay trở lại từ quy trình bắt đầu tạo. Giá trị trả về của hàm đóng vai trò là trạng thái thoát của luồng. Do đó không có pthread_exit () rõ ràng

được yêu cầu ở cuối một chủ đề.

Bao gồm

xmk.h, pthread.h

```
int pthread_join ( chuỗi pthread_t, void ** value_ptr)
```

Thông số	value_ptr là một con trỏ đến giá trị trả về của luồng.
Lợi nhuận	0 về thành công. ESRCH nếu luồng đích không ở trạng thái có thể nối hoặc là một luồng không hợp lệ. EINVAL nếu chuỗi đích đã có người đang đợi tham gia với nó.
Sự mô tả	Hàm pthread_join () tạm ngừng thực thi luồng đang gọi cho đến khi pthread_t (luồng đích) kết thúc, trừ khi luồng đích đã kết thúc. Hàm pthread_exit () của chuỗi kết thúc được tạo sẵn ở vị trí được tham chiếu bởi value_ptr. Khi một pthread_join () trả về thành công, luồng đích đã được kết thúc. Kết quả của nhiều cuộc gọi đồng thời tới pthread_join () chỉ định cùng một luồng đích là chỉ một luồng thành công và những luồng khác không thành công với EINVAL.

Lưu ý: Không có phát hiện bê tắc nào được cung cấp.

Bao gồm **xmk.h, pthread.h**

```
pthread_t pthread_self (void)
```

Thông số	Không có.
Lợi nhuận	Khi thành công, trả về mã định danh luồng của luồng hiện tại. Hành vi lỗi không được xác định.
Sự mô tả	Hàm pthread_self () trả về ID luồng của luồng đang gọi.
Bao gồm	xmk.h, pthread.h

```
int pthread_detach (pthread_t target)
```

Thông số	target là chuỗi mục tiêu để tách ra.
Lợi nhuận	0 về thành công. ESRCH nếu không tìm thấy luồng đích.
Sự mô tả	Hàm pthread_detach () cho biết việc triển khai rằng có thể lấy lại dung lượng lưu trữ cho luồng khi luồng đó kết thúc. Nếu luồng chưa kết thúc, pthread_detach () không làm cho nó kết thúc. Hiệu ứng của nhiều pthread_detach () các cuộc gọi trên cùng một luồng đích là không xác định.
Bao gồm	xmk.h, pthread.h

```
int pthread_equal (pthread_t t1, pthread_t t2)
```

Thông số	t1 và t2 là hai định danh luồng để so sánh.
Lợi nhuận	1 nếu t1 và t2 tham chiếu đến các chủ đề bằng nhau. 0 nếu không.
Sự mô tả	Hàm pthread_equal () trả về giá trị khác 0 nếu t1 và t2 bằng nhau; nếu không, số không được trả về. Nếu t1 hoặc t2 không phải là ID luồng hợp lệ, số không được trả về.
Bao gồm	xmk.h, pthread.h

```
int pthread_getschedparam (pthread_t thread, int * policy,
                           struct Sched_param * param)
```

Thông số	luồng là định danh của luồng để thực hiện thao tác. chính sách là một con trỏ đến vị trí nơi lưu trữ chính sách lập lịch biểu toàn cầu. param là một con trỏ đến cấu trúc tham số lập lịch.
Lợi nhuận	0 về thành công. ESRCH nếu giá trị được chỉ định bởi luồng không tham chiếu đến một luồng hiện có. EINVAL nếu tham số hoặc chính sách tham chiếu đến bộ nhớ không hợp lệ.
Sự mô tả	Hàm pthread_getschedparam () nhận chính sách lập lịch và các tham số của một luồng riêng lẻ. Đối với SCHED_RR không có tham số lập lịch; do đó, quy trình này không được xác định cho SCHED_RR. Đối với SCHED_PRIO, thành viên bắt buộc duy nhất của cấu trúc Sched_param là lịch trình ưu tiên. Giá trị ưu tiên trả về là giá trị được chỉ định bởi lệnh gọi pthread_getschedparam () hoặc pthread_create () gần đây nhất ảnh hưởng đến luồng đích. Nó không phản ánh bất kỳ điều chỉnh tạm thời nào đối với mức độ ưu tiên của nó do kết quả của bất kỳ chức năng trần hoặc kế thừa ưu tiên nào. Quy trình này chỉ được xác định nếu loại lập lịch là SCHED_PRIO.
Lợi nhuận	xmk.h, pthread.h

```
int pthread_setschedparam ( luồng pthread_t, int chính sách,
                           const struct Sched_param * param)
```

Thông số luồng là định danh của luồng để thực hiện thao tác.

chính sách bị bỏ qua.

param là một con trỏ đến cấu trúc tham số lập lịch.

Lợi nhuận

0 về thành công.

ESRCH nếu luồng không tham chiếu đến một luồng hợp lệ.

EINVAL nếu các tham số lập lịch không hợp lệ.

Sự mô tả

Hàm pthread_setschedparam () đặt chính sách lập lịch và các tham số của các luồng riêng lẻ sẽ được truy xuất. Đối với SCHED_RR không có tham số lập lịch; do đó, quy trình này không được xác định cho SCHED_RR. Đối với SCHED_PRIO, thành viên bắt buộc duy nhất của cấu trúc Sched_param là lich_trinh_uu_tien. Giá trị ưu tiên phải là một giá trị hợp lệ như được định cấu hình trong các tham số lập lịch của hạt nhân. Tham số chính sách bị bỏ qua.

Lưu ý: Quy trình này chỉ được xác định nếu loại lập lịch là SCHED_PRIO.

Bao gồm

xmk.h, pthread.h

```
int pthread_attr_init (pthread_attr_t * attr)
```

Thông số attr là một con trỏ đến cấu trúc thuộc tính được khởi tạo.

Lợi nhuận

0 về thành công.

1 về thất bại.

EINVAL trên tham số attr không hợp lệ .

Sự mô tả

Hàm pthread_attr_init () khởi tạo đối tượng thuộc tính luồng với giá trị mặc định cho tất cả các thuộc tính riêng lẻ được sử dụng bởi một triển khai nhất định. Nội dung hàm được xác định trong tiêu đề sys / styles.h .

Lưu ý: Hàm này không thực hiện cuộc gọi tới hạt nhân.

Bao gồm

xmk.h, pthread.h

```
int pthread_attr_destroy (pthread_attr_t * attr)
```

Thông số	attr là một con trỏ đến các thuộc tính luồng cần phải bị hủy.
Lợi nhuận	0 về thành công. EINVAL về lỗi.
Sự mô tả	Hàm pthread_attr_destroy () hủy đổi tư ợng thuộc tính luồng và đặt attr thành giá trị không hợp lệ do triển khai xác định.
	Khởi tạo lại đổi tư ợng thuộc tính attr bị hủy bằng pthread_attr_init (); kết quả của việc tham chiếu khác đến đổi tư ợng sau khi nó bị phá hủy là không xác định. Lưu ý: Hàm này không thực hiện cuộc gọi tới hạt nhân.

Bao gồm xmk.h, pthread.h

```
int pthread_attr_setdetachstate (pthread_attr_t * attr, int
dstate)
```

Thông số	attr là cấu trúc thuộc tính mà hoạt động sẽ được thực hiện. dstate là yêu cầu tách rời.
Lợi nhuận	0 về thành công. EINVAL trên các tham số không hợp lệ.
Sự mô tả	Thuộc tính detachstate kiểm soát xem luồng có được tạo ở trạng thái tách rời hay không. Nếu luồng được tạo tách rời, thì khi luồng thoát ra, tài nguyên của luồng sẽ được tách ra mà không yêu cầu pthread_join () hoặc lệnh gọi pthread_detach (). Ứng dụng có thể đặt detachstate thành PTHREAD_CREATE_DETACHED hoặc PTHREAD_CREATE_JOINABLE.

Lưu ý: Thao tác này không thực hiện cuộc gọi vào hạt nhân.

Bao gồm xmk.h, pthread.h

```
int pthread_attr_getdetachstate (pthread_attr_t * attr, int
* dstate)
```

Thông số	attr là cấu trúc thuộc tính mà hoạt động sẽ được thực hiện.
Lợi nhuận	0 về thành công. EINVAL trên các tham số không hợp lệ.
Sự mô tả	Việc triển khai lưu trữ PTHREAD_CREATE_DETACHED hoặc PTHREAD_CREATE_JOINABLE trong dstate, nếu giá trị của detachstate là hợp lệ trong attr. Lưu ý: Thao tác này không thực hiện cuộc gọi vào hạt nhân.

Bao gồm xmk.h, pthread.h

```
int pthread_attr_setschedparam (pthread_attr_t * attr,
                               struct Sched_param * Schedpar)
```

Thông số attr là cấu trúc thuộc tính mà hoạt động sẽ được thực hiện.

Schedpar là vị trí của cấu trúc có chứa các tham số lập lịch.

Lợi nhuận Ø về thành công.

EINVAL trên các tham số không hợp lệ.

ENOTSUP cho các tham số lập lịch không hợp lệ.

Sự mô tả Hàm pthread_attr_setschedparam () đặt các thuộc tính tham số lập lịch trong đối số attr .

Nội dung của cấu trúc Sched_param được xác định trong tiêu đề Sched.h .

Lưu ý: Thao tác này không thực hiện cuộc gọi vào hạt nhân.

Bao gồm xmk.h, pthread.h

```
int pthread_attr_getschedparam (pthread_attr_t * attr,
                               struct Sched_param * Schedpar)
```

Thông số attr là cấu trúc thuộc tính mà hoạt động sẽ được thực hiện.

Schedpar là vị trí để lưu trữ Sched_param kết cấu.

Lợi nhuận Ø về thành công.

EINVAL trên các tham số không hợp lệ.

Sự mô tả Pthread_attr_getschedparam () nhận các thuộc tính tham số lập lịch trong đối số attr . Nội dung của cấu trúc tham số được xác định trong tiêu đề Sched.h .

Lưu ý: Thao tác này không thực hiện cuộc gọi tới hạt nhân.

Bao gồm xmk.h, pthread.h

```
int pthread_attr_setstack (const pthread_attr_t * attr, _  
    void * stackaddr, size_t stacksize)
```

Thông số attr là cấu trúc thuộc tính để thực hiện thao tác.

stackaddr là địa chỉ cơ sở của bộ nhớ ngăn xếp.

stacksize là kích thước của khối bộ nhớ tính bằng byte.

Lợi nhuận 0 về thành công.

EINVAL nếu tham số định kèm không hợp lệ hoặc nếu stackaddr không được cẩn chỉnh phù hợp.

Sự mô tả Hàm pthread_attr_setstack () sẽ đặt các thuộc tính ngăn xếp tạo luồng là stackaddr và stacksize trong attr sự vật.

Các thuộc tính ngăn xếp chỉ định khu vực lưu trữ sẽ được sử dụng cho ngăn xếp của luồng đã tạo. Cơ sở (byte địa chỉ thấp nhất) của bộ lưu trữ là stackaddr và kích thước của bộ lưu trữ là byte kích thước ngăn xếp.

Ngăn xếp phải được cẩn chỉnh thích hợp theo EABI của bộ xử lý, để được sử dụng như một ngăn xếp; ví dụ: pthread_attr_setstack () có thể không thành công với EINVAL nếu (stackaddr và 0x3) không phải là 0.

Đối với bộ xử lý PowerPC 405, cẩn chỉnh bắt buộc là 8 byte.

Đối với bộ xử lý MicroBlaze, cẩn chỉnh yêu cầu là 4 byte.

Bao gồm xmk.h, pthread.h

```
int pthread_attr_getstack (const pthread_attr_t * attr, void  
    ** stackaddr, size_t * stacksize)
```

Thông số attr là cấu trúc thuộc tính để thực hiện thao tác.

stackaddr là vị trí lưu trữ địa chỉ cơ sở của ngăn xếp
kí niêm.

stacksize là vị trí lưu trữ kích thước của khối bộ nhớ tính bằng byte.

Lợi nhuận 0 về thành công.

EINVAL trên tập tin định kèm không hợp lệ.

Sự mô tả Hàm pthread_attr_getstack () truy xuất các thuộc tính tạo luồng liên quan đến ngăn xếp của cấu trúc thuộc tính được chỉ định và lưu trữ nó trong stackaddr và stacksize.

Bao gồm xmk.h, pthread.h

pid_t get_currentPID (vô hiệu)

Thông số	Không có.
Lợi nhuận	Số nhận dạng quy trình được liên kết với luồng hiện tại hoặc quy trình elf.
Sự mô tả	Nhận mã định danh quy trình cơ bản của ngữ cảnh quy trình hiện đang thực thi. Mã định danh quy trình là cần thiết để thực hiện các hoạt động nhất định như kill () trên cả quy trình và luồng.
Bao gồm	xmk.h, sys / process.h

int kill (pid_tpid)

Thông số	pid là PID của quy trình.
Lợi nhuận	0 về thành công. -1 về thất bại.
Sự mô tả	Xóa bối cảnh quy trình được chỉ định bởi pid khỏi hệ thống. Nếu pid đề cập đến ngữ cảnh quy trình đang thực thi hiện tại, sau đó nó tương đương với ngữ cảnh quy trình hiện tại kết thúc. Một lệnh giết có thể được gọi trên các quá trình bị tạm dừng trên hàng đợi hoặc thời gian chờ. Không có dấu hiệu nào được đưa ra cho các quy trình khác phụ thuộc vào bối cảnh quy trình này. Lưu ý: Hàm này chỉ được xác định nếu CONFIG_KILL là true. Điều này có thể được cấu hình với danh mục tính năng nâng cao của hạt nhân.
Bao gồm	xmk.h, sys / process.h

int process_status (pid_t pid, p_stat * ps)

Thông số	pid là PID của quá trình. ps là bộ đệm nơi trạng thái quá trình được trả về.
Lợi nhuận	Trạng thái quy trình trong ps về thành công. NULL trong ps khi thất bại.
Sự mô tả	Nhận trạng thái của tiến trình hoặc luồng, có pid là pid. Trạng thái được trả về trong cấu trúc p_stat có các trường sau: • pid là ID quy trình. • trạng thái là trạng thái lập lịch hiện tại của tiến trình. Nội dung của p_stat được xác định trong tiêu đề sys / ktypes.h .
Bao gồm	xmk.h, sys / process.h

```
int xmk_add_static_thread (void * (* start_routine) (void *),
                           int Sched_priasty)
```

Thông số	<code>start_routine</code> là quy trình bắt đầu chuỗi. <code>Sched_priasty</code> là mức ưu tiên của luồng khi hạt nhân được cấu hình để lập lịch ưu tiên.
Lợi nhuận	0 khi thành công và -1 khi thất bại.
Sự mô tả	Hàm này cung cấp khả năng thêm một luồng vào danh sách các luồng khởi động hoặc tĩnh chạy khi khởi động hạt nhân, thông qua mã C. Hàm này phải được sử dụng trước khi <code>xilkernel_main()</code> được gọi.
Bao gồm	<code>xmk.h</code> , <code>sys / init.h</code>

int yi (vô hiệu)

Thông số	Không có.
Lợi nhuận	Không có.
Sự mô tả	Đưa bộ xử lý đến bối cảnh quy trình tiếp theo đã sẵn sàng để thực thi. Quy trình hiện tại được đưa trở lại hàng đợi sẵn sàng thích hợp. Lưu ý: Chức năng này là tùy chọn và chỉ được bao gồm khi <code>CONFIG_YIELD</code> được xác định. Điều này có thể được cấu hình với mục <code>tinh_nang_cao</code> của hạt nhân.
Bao gồm	<code>xmk.h</code> , <code>sys / process.h</code>

Semaphores

Xilkernel hỗ trợ các semaphores POSIX được cấp phát hạt nhân có thể được sử dụng để đồng bộ hóa. Các semaphores POSIX đang đếm các semaphores cũng đếm dưới 0 (một giá trị âm cho biết số lượng quá trình bị chặn trên semaphore). Xilkernel cũng hỗ trợ một vài giao diện để làm việc với các semaphores được đặt tên. Số lượng semaphores được cấp phát trong hạt nhân và độ dài của hàng đợi semaphore có thể được cấu hình trong quá trình khởi tạo hệ thống.
Tham khảo “[Định cấu hình Semaphores](#)”, trang 51 để biết thêm chi tiết. Mô-đun semaphore là tùy chọn và có thể được cấu hình trong hoặc ngoài trong quá trình khởi tạo hệ thống. Mô-đun hàng đợi thông báo, được mô tả ở phần sau của tài liệu này, sử dụng semaphores. Mô-đun này phải được bao gồm nếu bạn muốn sử dụng hàng đợi tin nhắn.

Tóm tắt hàm Semaphore

Danh sách sau cung cấp một bản tóm tắt được liên kết của các hàm semaphore trong Xilkernel. Bạn có thể nhấp vào một chức năng để chuyển đến phần mô tả.

```
int sem_init (sem_t * sem, int pshared, giá trị không dấu)
int sem_destroy (sem_t * sem)
int sem_getvalue (sem_t * sem, int * value)
int sem_wait (sem_t * sem)
int sem_trywait (sem_t * sem)
int sem_timedwait (sem_t * sem, unsigned_ms)
sem_t * sem_open (const char * name, int oflag, ...)
int sem_close (sem_t * sem)
int sem_post (sem_t * sem)
int sem_unlink (const char * name)
```

Mô tả hàm Semaphore

Sau đây là mô tả về các hàm semaphore của Xilkernel:

```
int sem_init (sem_t * sem, int pshared, giá trị không dấu)
```

Thông số

sem là vị trí để lưu định danh của semaphore đã tạo.

pshared cho biết trạng thái chia sẻ của semaphore, giữa các tiến trình.

giá trị là số lượng ban đầu của semaphore.

Lưu ý: pshared hiện chưa được sử dụng.

Lợi nhuận

0 về thành công.

-1 về lỗi và đặt lỗi thích hợp. Errno được đặt thành ENOSPC nếu không còn tài nguyên semaphore trong hệ thống.

Sự mô tả

Hàm sem_init () khởi tạo semaphore không tên được gọi bởi sem. Giá trị của semaphore được khởi tạo là giá trị. Sau một cuộc gọi thành công đến sem_init (), semaphore có thể được sử dụng trong các cuộc gọi tiếp theo sem_wait (), sem_trywait (), sem_post () và sem_destroy ().Semaphore này vẫn có thể sử dụng được cho đến khi semaphore bị phá hủy. Chỉ bản thân sem có thể được sử dụng để thực hiện đồng bộ hóa. Kết quả của việc tham chiếu đến các bản sao của sem trong các cuộc gọi tới sem_wait (), sem_trywait (), sem_post () và sem_destroy () là không xác định. Có gắng khởi tạo một semaphore đã được khởi tạo dẫn đến hành vi không xác định.

Bao gồm

xmk.h, semaphore.h

```
int sem_destroy (sem_t * sem)
```

Thông số

sem là semaphore sẽ bị hủy.

Lợi nhuận

0 về thành công.

-1 về lỗi và đặt lỗi thích hợp. Errno có thể được đặt thành:

- EINVAL nếu định danh semaphore không tham chiếu đến semaphore hợp lệ.
- EBUSY nếu semaphore hiện đang bị khóa và các quy trình bị chặn trên đó.

Sự mô tả

Hàm sem_destroy () phá hủy semaphore không tên được chỉ ra bởi sem. Chỉ một semaphore được tạo bằng sem_init () mới có thể bị hủy bằng sem_destroy (); hiệu quả của việc gọi sem_destroy () với một semaphore được đặt tên là không xác định. Hiệu quả của việc sử dụng tiếp theo của semaphore là không xác định cho đến khi sem được khởi tạo lại bằng một lời gọi khác tới sem_init () .

Bao gồm

xmk.h, semaphore.h

```
int sem_getvalue (sem_t * sem, int * value)
```

Thông số

sem là định danh semaphore.

value là vị trí lưu trữ giá trị semaphore.

Lợi nhuận

0 về thành công và giá trị được điền thích hợp.

-1 khi thất bại và đặt errno một cách thích hợp. Errno có thể được đặt thành EINVAL nếu định danh semaphore đe cập đến một semaphore không hợp lệ.

Sự mô tả

Hàm sem_getvalue () cập nhật vị trí được tham chiếu bởi đối số sval để có giá trị của semaphore được tham chiếu bởi sem mà không ảnh hưởng đến trạng thái của semaphore. Giá trị được cập nhật đại diện cho một giá trị semaphore thực tế đã xảy ra tại một số thời điểm không xác định trong quá trình gọi, nhưng nó không cần phải là giá trị thực của semaphore khi nó được trả về quá trình gọi.

Nếu sem bị khóa, thì đối tượng mà sval trả đến được đặt thành một số âm có giá trị tuyệt đối đại diện cho số lượng tiến trình đang chờ semaphore tại một số thời điểm không xác định trong khi gọi.

Bao gồm

xmk.h, semaphore.h

```
int sem_wait (sem_t * sem)
```

Thông số

sem là định danh semaphore.

Lợi nhuận

0 khi thành công và semaphore ở trạng thái bị khóa.

-1 về lỗi và lỗi được thiết lập thích hợp. Errno có thể được đặt thành:

- EINVAL nếu định danh semaphore không hợp lệ.
- EIDRM nếu semaphore bị loại bỏ cưỡng bức.

Sự mô tả

Hàm sem_wait () khóa semaphore được tham chiếu bởi sem bằng cách thực hiện thao tác khóa semaphore trên semaphore đó. Nếu giá trị semaphore hiện là 0, thì luồng đang gọi sẽ không trở lại từ cuộc gọi đến sem_wait () cho đến khi nó khóa semaphore hoặc semaphore bị phá hủy cưỡng bức.

Khi trả về thành công, trạng thái của semaphore bị khóa và vẫn bị khóa cho đến khi hàm sem_post () được thực thi và trả về thành công.

Lưu ý: Khi một tiến trình bị chặn trong lệnh gọi sem_wait , nó bị chặn do không có sẵn semaphore, semaphore có thể đã bị phá hủy cưỡng bức. Trong trường hợp này, -1 được trả về. Semaphores có thể bị hủy cưỡng bức do hủy các hàng đợi tin nhắn sử dụng semaphores nội bộ. Không có phát hiện bê tắc nào được cung cấp.

Bao gồm

xmk.h, semaphore.h

```
int sem_trywait (sem_t * sem)
```

Thông số

sem là định danh semaphore.

Lợi nhuận

0 về thành công.

-1 về lỗi và lỗi được thiết lập thích hợp. Errno có thể được đặt thành:

- EINVAL nếu định danh semaphore không hợp lệ.
- EAGAIN nếu không thể khóa semaphore ngay lập tức.

Sự mô tả

Hàm sem_trywait () khóa semaphore được tham chiếu bởi sem chỉ khi semaphore hiện không bị khóa; nghĩa là, nếu giá trị semaphore hiện là dương. Nếu không, nó không khóa semaphore và trả về -1.

Bao gồm

xmk.h, semaphore.h

```
int sem_timedwait (sem_t * sem, unsigned_ms)
```

Thông số

sem là định danh semaphore.

Lợi nhuận

0 khi thành công và semaphore ở trạng thái bị khóa.

-1 về lỗi và lỗi được thiết lập thích hợp. Errno có thể được đặt thành:

- EINVAL - Nếu định danh semaphore không tham chiếu đến semaphore hợp lệ.
- ETIMEDOUT - Không thể khóa semaphore trước khi hết thời gian quy định.
- EIDRM - Nếu semaphore bị buộc phải xóa khỏi hệ thống.

Sự mô tả

Hàm sem_timedwait () khóa semaphore được tham chiếu bởi sem bằng cách thực hiện thao tác khóa semaphore trên semaphore đó. Nếu giá trị semaphore hiện là 0, thì chuỗi đang gọi sẽ không trả về từ cuộc gọi đến sem_timedwait () cho đến khi một trong các điều kiện sau xảy ra:

- Nó khóa semaphore. •
- Semaphore bị phá hủy cưỡng bức.
- Thời gian chờ được chỉ định đã trôi qua.

Khi trả về thành công, trạng thái của semaphore bị khóa và vẫn bị khóa cho đến khi hàm sem_post () được thực thi và trả về thành công.

Lưu ý: Khi một tiến trình bị bỏ chặn trong lệnh gọi sem_wait , nó i nó bị chặn do không có sẵn semaphore, semaphore có thể đã bị phá hủy cưỡng bức. Trong trường hợp này, -1 được trả về. Semaphores có thể bị phá hủy cưỡng bức do phá hủy các hàng đợi tin nhắn sử dụng semaphores nội bộ. Không có phát hiện bối tắc nào được cung cấp.

Lưu ý: Quy trình này phụ thuộc vào việc hỗ trợ bộ hẹn giờ phần mềm có trong hạt nhân và chỉ được xác định nếu CONFIG_TIME là true.

Lưu ý: Quy trình này hơi khác so với quy trình tương ứng với POSIX. Phiên bản POSIX chỉ định thời gian chờ là thời gian đồng hồ treo tương tuyệt đối. Bởi vì không có khái niệm về thời gian tuyệt đối trong Xilkernel, chúng tôi sử dụng thời gian tương đối được chỉ định bằng mili giây.

Bao gồm

xmk.h, semaphore.h

```
sem_t * sem_open (const char * name, int oflag, ...)
```

Thông số

tên trả đến một chuỗi đặt tên cho một đối tượng semaphore.

oflag là cờ điều khiển việc tạo semaphore.

Lợi nhuận

Một con trỏ đến mã định danh semaphore đã tạo / hiện có.

SEM_FAILED về lỗi và khi lỗi được đặt thích hợp. Errno có thể được đặt thành:

- ENOSPC - Nếu hệ thống hết tài nguyên để tạo một semaphore (hoặc ánh xạ) mới.

- EEXIST - nếu O_EXCL đã được yêu cầu và semaphore được đặt tên đã tồn tại.

- EINVAL - nếu các tham số không hợp lệ.

Sự mô tả

Hàm sem_open () thiết lập một kết nối giữa một semaphore được đặt tên và một tiến trình. Đang theo lệnh gọi tới sem_open () với tên semaphore , tiến trình có thể tham chiếu đến semaphore được liên kết với tên bằng cách sử dụng địa chỉ được trả về từ cuộc gọi. Semaphore này có thể được sử dụng trong các cuộc gọi tiếp theo sem_wait (), sem_trywait (), sem_post () và sem_close (). Semaphore vẫn có thể sử dụng được bởi quá trình này cho đến khi semaphore được đóng bằng một lệnh gọi thành công đến sem_close (). Đối số oflag kiểm soát việc semaphore được tạo hay chỉ được truy cập bằng lệnh gọi tới sem_open (). Các bit có thể được đặt trong oflag là:

O_CREAT

Được sử dụng để tạo một semaphore nếu nó chưa tồn tại. Nếu O_CREAT được đặt và semaphore đã tồn tại, thì O_CREAT không có hiệu lực, ngoại trừ trừ được ghi chú trong O_EXCL.

Nếu không, sem_open () tạo ra một semaphore được đặt tên.

O_CREAT yêu cầu đối số thứ ba và thứ tư : mode, thuộc loại mode_t và giá trị, thuộc loại không có dấu. O_EXCL

Nếu O_EXCL và O_CREAT được đặt, sem_open () không thành công nếu tồn tại tên semaphore. Việc kiểm tra sự tồn tại của semaphore và việc tạo ra semaphore nếu nó không tồn tại là nguyên tử đối với các quy trình khác thực thi sem_open () với O_EXCL và O_CREAT set. Nếu O_EXCL được đặt và O_CREAT không được đặt, hiệu ứng là không xác định.

Lưu ý: Đối số chế độ hiện không được sử dụng. Giao diện này là tùy chọn và chỉ được xác định nếu CONFIG_NAMED_SEMA được đặt thành TRUE.

Lưu ý: Nếu các cờ không phải O_CREAT và O_EXCL được chỉ định trong tham số oflag , thì một lỗi sẽ được báo hiệu.

Semaphore được tạo với giá trị ban đầu là value.

Sau khi tên semaphore đã được tạo bởi sem_open () với cờ O_CREAT , các tiến trình khác có thể kết nối với semaphore bằng cách gọi sem_open () có cùng giá trị tên.

Nếu một tiến trình thực hiện nhiều cuộc gọi thành công đến sem_open () với cùng một giá trị cho tên, thì cùng một địa chỉ semaphore sẽ được trả về cho mỗi cuộc gọi thành công như vậy, giả sử rằng không có lệnh gọi nào đến sem_unlink () cho semaphore này.

Bao gồm

xmk.h, semaphore.h

int sem_close (sem_t * sem)

Thông số

sem là định danh semaphore.

Lợi nhuận

0 về thành công.

-1 về lỗi và đặt lỗi thích hợp. Errno có thể được đặt thành:

- EINVAL - Nếu định danh semaphore không hợp lệ.
- ENOTSUP - Nếu semaphore hiện đang bị khóa và / hoặc các tiến trình bị chặn trên semaphore.

Sự mô tả

Hàm sem_close () cho biết rằng quá trình gọi đã kết thúc bằng cách sử dụng sem có tên semaphore . Sem_close ()

hàm deallocates (có nghĩa là, có sẵn để sử dụng lại bởi sem_open () tiếp theo bởi quá trình này) bất kỳ tài nguyên hệ thống nào được hệ thống cấp phát để quá trình này sử dụng cho semaphore này. Hiệu quả của việc sử dụng tiếp theo của semaphore được chỉ ra bởi sem bởi quá trình này là không xác định. Ánh xạ tên cho semaphore có tên này cũng bị hủy. Cuộc gọi không thành công nếu semaphore hiện đang bị khóa.

Lưu ý: Giao diện này là tùy chọn và chỉ được xác định nếu CONFIG_NAMED_SEMA là true.

Bao gồm

xmk.h, semaphore.h

int sem_post (sem_t * sem)

Thông số

sem là định danh semaphore.

Lợi nhuận

0 về thành công.

-1 về lỗi và đặt lỗi thích hợp. Errno có thể được đặt thành EINVAL nếu định danh semaphore không hợp lệ.

Sự mô tả

Hàm sem_post () thực hiện thao tác mở khóa trên semaphore được tham chiếu bởi định danh sem .

Nếu giá trị semaphore tạo ra từ hoạt động này là dương, thì không có luồng nào bị chặn khi đợi semaphore được mở khóa và giá trị semaphore được tăng lên.

Nếu giá trị của semaphore tạo ra từ hoạt động này là 0 hoặc âm, thì một trong các luồng bị chặn đang chờ semaphore được phép trả về thành công từ lệnh gọi của nó tới sem_wait (). Đây là luồng đầu tiên trong hàng đợi, nếu chế độ lập lịch là SCHED_RR hoặc, nó là luồng có mức độ ưu tiên cao nhất trong hàng đợi, nếu chế độ lập lịch là SCHED_PRIO.

Lưu ý: Nếu thao tác hủy liên kết được yêu cầu trên semaphore, thì thao tác đăng sẽ thực hiện hủy liên kết khi không còn quá trình nào đang chờ trên semaphore.

Bao gồm

xmk.h, semaphore.h

```
int sem_unlink (const char * name)
```

Thông số	tên là tên đè cập đến semaphore.
Lợi nhuận	0 về thành công.
Sự mô tả	Hàm sem_unlink () loại bỏ semaphore được đặt tên bằng tên chuỗi. Nếu semaphore được đặt theo tên có các tiến trình bị chặn trên đó, thì sem_unlink () không có tác dụng ngay lập tức đến trạng thái của semaphore. Việc phá hủy semaphore được hoàn lại cho đến khi tắt cả các quá trình bị chặn và khóa từ bỏ semaphore. Các lệnh gọi đến sem_open () để tạo lại hoặc kết nối lại với semaphore tham chiếu đến một semaphore mới sau khi sem_unlink () được gọi. Lời gọi sem_unlink () không chặn cho đến khi tắt cả các tham chiếu từ bỏ semaphore; nó trả về ngay lập tức.

Lưu ý: Nếu thao tác hủy liên kết đã được yêu cầu trên semaphore, thì thao tác hủy liên kết được thực hiện trên thao tác đăng mà thấy rằng không có quá trình nào đang chờ trên semaphore. Giao diện này là tùy chọn và chỉ được xác định nếu CONFIG_NAMED_SEMA là true.

Bao gồm xmk.h, semaphore.h

Hàng đợi tin nhắn

Xilkernel hỗ trợ hàng đợi thông báo X / Open System Interface (XSI) được cấp phát hạt nhân. XSI là một tập hợp các giao diện tùy chọn trong POSIX. Hàng đợi tin nhắn có thể được sử dụng như một cơ chế IPC. Các hàng đợi tin nhắn có thể nhận các tin nhắn có kích thước tùy ý. Tuy nhiên, cấp phát bộ nhớ đệm phải được cấu hình thích hợp cho các khối bộ nhớ cần thiết cho các thông báo, như phần của quá trình khởi tạo cấp phát bộ nhớ đệm hệ thống. Trong quá trình khởi tạo hệ thống. Mô-đun hàng đợi tin nhắn là tùy chọn và có thể được cấu hình vào / ra.

Tham khảo “[Định cấu hình hàng đợi tin nhắn](#)”, trang 51 để biết thêm chi tiết. Mô-đun này phụ thuộc vào mô-đun semaphore và mô-đun cấp phát bộ nhớ đệm động hiện có trong hệ thống. Ngoài ra còn có một chức năng hàng đợi tin nhắn lớn hơn như `mq_getmsg()` có thể được định cấu hình nếu được yêu cầu. Khi giao diện hàng đợi tin nhắn nâng cao được chọn, thì `malloc` và miễn phí được sử dụng để phân bổ và giải phóng không gian cho các tin nhắn. Do đó, các thông báo có kích thước tùy ý có thể được truyền đi xung quanh mà không cần phải đảm bảo rằng các API cấp phát bộ nhớ đệm có thể xử lý các yêu cầu đối với kích thước tùy ý.

Lưu ý: Khi sử dụng tính năng hàng đợi thư nêu cao, bạn phải chọn kích thước đồng chung của mảng một cách cẩn thận, sao cho các yêu cầu về bộ nhớ heap từ giao diện hàng đợi thư được đáp ứng mà không có lỗi. Bạn cũng phải lưu ý về các vấn đề an toàn luồng khi sử dụng malloc(), free() trong mã của riêng bạn. Bộ nhớ hóng ngắt và chuyển mạch ngữ cảnh trัว ớc khi gọi các quy trình cấp phát bộ nhớ động. Bạn phải tuân theo các quy tắc tương tự khi sử dụng bất kỳ quy trình thư viện nào khác có thể sử dụng cấp phát bộ nhớ động nội bộ.

Tóm tắt chức năng hàng đợi tin nhắn

Danh sách sau đây cung cấp một bản tóm tắt được liên kết của các hàng đợi tin nhắn trong Xilkernel. Bạn có thể nhấp vào một chức năng để chuyển đến phần mô tả.

```
int msgget (key_t key, int msgflg)
int msgctl (int msqid, int cmd, struct msqid_ds * buf)
int msgsnd (int msqid, const void * msgp, size_t msgsiz, int msgflg)
ssize_t msgrcv (int msqid, void * msqp, size_t nbytes, long msgtyp, int msgflg)
```

Mô tả chức năng hàng đợi tin nhắn

Mô tả chức năng hàng đợi tin nhắn Xilkernel như sau:

```
int msgget (key_t key, int msgflg)
```

Thông số

key là một mã định danh duy nhất để tham chiếu đến hàng đợi tin nhắn.
msgflg chỉ định các tùy chọn tạo hàng đợi tin nhắn.

Lợi nhuận

Một mã định danh hàng đợi tin nhắn số nguyên không âm duy nhất.

-1 về lỗi và đặt lỗi thích hợp; errno có thể được đặt thành:

EEXIST - Nếu tồn tại mã định danh hàng đợi thông báo cho khóa đối số
nhưng ((msgflg và IPC_CREAT) và msgflg & IPC_EXCL) khác 0.

ENOENT - Mã định danh hàng đợi thông báo không tồn tại cho khóa
đối số và (msgflg & IPC_CREAT) là 0.

ENOSPC - Nếu tài nguyên hàng đợi tin nhắn đã cạn kiệt.

Sự mô tả

Hàm msgget () trả về mã định danh hàng đợi thông báo được liên
kết với khóa đối số. Mã định danh hàng đợi thông báo, hàng đợi
thông báo được liên kết và cấu trúc dữ liệu (xem sys / kmsg.h), được
tạo cho khóa đối số nếu khóa đối số chưa có mã định danh hàng đợi
thông báo được liên kết với nó và (msgflg và IPC_CREAT) là khác không.

Khi tạo, cấu trúc dữ liệu được liên kết với mã định danh hàng đợi
tin nhắn mới được khởi tạo như sau:

msg_qnum, msg_lspid, msg_lrid được đặt bằng 0.

msg_qbytes được đặt bằng giới hạn hệ thống
(MSGQ_MAX_BYTENS).

Hàm msgget () không thành công nếu tồn tại mã định danh hàng đợi thông
báo cho khóa đối số như ng ((msgflg và IPC_CREAT) và (msgflg &
IPC_EXCL)) khác 0.

IPC_PRIVATE không được hỗ trợ. Ngoài ra, các tin nhắn trong hàng đợi
tin nhắn không bắt buộc phải có dạng như hình dưới đây. Không có hỗ
trợ cho loại tin nhắn nhận và gửi tin nhắn trong quá trình triển khai này.

Sau đây là một đoạn mã mẫu:

```
struct mymsg {
    .. long mtype; /* Loại tin nhắn. */
    .. char mtext [some_size]; /* Tin nhắn văn bản. */
}
```

Bao gồm

xmk.h, sys / msg.h, sys / ipc.h

```
int msgctl (int msqid, int cmd, struct msqid_ds * buf)
```

Thông số

msqid là định danh hàng đợi tin nhắn.

cmd là lệnh.

buf là bộ đệm dữ liệu

Lợi nhuận

0 về thành công. Trạng thái được trả về trong buf cho IPC_STAT.

-1 về lỗi và đặt lỗi thích hợp. Lỗi có thể được đặt thành EINVAL nếu xảy ra bất kỳ điều kiện nào sau đây:

- Thông số msgstr đề cập đến một hàng đợi tin nhắn không hợp lệ.
- cmd không hợp lệ.
- buf chứa các tham số không hợp lệ.

Sự mô tả

Hàm msgctl () cung cấp các hoạt động điều khiển thông báo như được chỉ định bởi cmd. Các giá trị cho cmd và các hoạt động kiểm soát thông báo mà chúng chỉ định, là:

- IPC_STAT - Đặt giá trị hiện tại của từng thành viên trong Cấu trúc dữ liệu msqid_ds liên kết với msqid vào cấu trúc được buf trả tới. Nội dung của cấu trúc này được định nghĩa trong sys / msg.h.
- IPC_SET - Không được hỗ trợ.

- IPC_RMID - Xóa định danh hàng đợi tin nhắn do msqid chỉ định khỏi hệ thống và hủy hàng đợi tin nhắn và cấu trúc dữ liệu msqid_ds liên quan. Thao tác remove buộc phải phá hủy các semaphores được sử dụng nội bộ và bỏ chặn các tiến trình bị chặn trên semaphore. Nó cũng phân bổ bộ nhớ được cấp phát cho các thông điệp trong hàng đợi.

Bao gồm

xmk.h, sys / msg.h, sys / ipc.h

```
int msgsnd (int msqid, const void * msgp, size_t msgsz, int msgflg)
```

Thông số	<p>msqid là định danh hàng đợi tin nhắn.</p> <p>msgp là một con trỏ đến bộ đệm thư.</p> <p>msgsz là kích thước của tin nhắn.</p> <p>msgflg được sử dụng để chỉ định các tùy chọn gửi tin nhắn.</p>
Lợi nhuận	<p>0 về thành công.</p> <p>-1 về lỗi và đặt lỗi thích hợp. Errno có thể được đặt thành:</p> <ul style="list-style-type: none"> • EINVAL - Giá trị của msqid không phải là hàng đợi tin nhắn hợp lệ định danh. • ENOSPC - Hệ thống không thể cấp phát không gian cho thông báo. • EIDRM - Hàng đợi tin nhắn đã bị xóa khỏi hệ thống trong quá trình gửi.
Sự mô tả	<p>Hàm msgsnd () gửi thông báo đến hàng đợi được liên kết với định danh hàng đợi thông báo được chỉ định bởi msqid.</p> <p>Đối số msgflg chỉ định hành động sẽ được thực hiện nếu hàng đợi thông báo đã đầy:</p> <p>Nếu (msgflg và IPC_NOWAIT) khác 0, tin nhắn sẽ không được gửi và chuỗi cuộc gọi trả về ngay lập tức.</p> <p>Nếu (msgflg và IPC_NOWAIT) là 0, chuỗi gọi sẽ tạm ngừng thực thi cho đến khi một trong những điều sau xảy ra:</p> <ul style="list-style-type: none"> • Điều kiện chịu trách nhiệm cho việc tạm ngưng không còn tồn tại, trong tin nhắn được gửi trong trường hợp nào. • msqid nhận dạng hàng đợi tin nhắn bị xóa khỏi hệ thống; khi điều này xảy ra, -1 được trả về. <p>Gửi không thành công nếu nó không thể cấp phát bộ nhớ để lưu thông báo bên trong hạt nhân. Trên một hoạt động gửi thành công, msg_llspid và msg_qnum các thành viên của hàng đợi tin nhắn được thiết lập thích hợp.</p>
Bao gồm	xmk.h, sys / msg.h, sys / ipc.h

```
ssize_t msgrcv (int msqid, void * msgp, size_t nbytes, long
msgtyp, int msgflg)
```

Thông số

`msqid` là định danh hàng đợi tin nhắn.
`msgp` là bộ đệm nơi tin nhắn nhận được sẽ được sao chép.
`nbytes` chỉ định kích thước của thông báo mà bộ đệm có thể chứa.
`msgtyp` hiện không được hỗ trợ.
`msgflg` được sử dụng để điều khiển hoạt động nhận tin nhắn.

Lợi nhuận

Khi thành công, lưu trữ thông báo đã nhận trong bộ đệm của người dùng và trả về số byte dữ liệu đã nhận.
-1 về lỗi và đặt lỗi thích hợp. Errno có thể được đặt thành:
• `EINVAL` - Nếu `msgstr` không phải là một định danh hàng đợi tin nhắn hợp lệ.
• `EIDRM` - Nếu hàng đợi tin nhắn đã bị xóa khỏi hệ thống.
• `ENOMSG` - `msgsz` nhỏ hơn kích thước của thông báo trong xếp hàng.

Sự mô tả

Hàm `msgrcv` () đọc một thông báo từ hàng đợi được liên kết với mã định danh hàng đợi thông báo được chỉ định bởi `msqid` và đặt nó vào bộ đệm do người dùng xác định được chỉ đến bởi `msgp`.

Đối số `msgsz` chỉ định kích thước tính bằng byte của thông báo. Tin nhắn đã nhận được cắt ngắn thành byte `msgsz` nếu nó lớn hơn `msgsz` và (`msgflg` và `MSG_NOERROR`) khác 0. Phần bị cắt ngắn của thông báo bị mất và không có dấu hiệu nào về việc cắt ngắn được đưa ra cho quá trình gọi. Nếu `MSG_NOERROR` không được chỉ định và thông báo nhận được lớn hơn `nbyte`, thì -1 là lỗi báo hiệu trả về.

Đối số `msgflg` chỉ định hành động được thực hiện nếu thư không có trong hàng đợi. Những điều này như sau:

- Nếu (`msgflg` và `IPC_NOWAIT`) khác 0, chuỗi gọi trả về ngay lập tức với giá trị trả về -1.
- Nếu (`msgflg` và `IPC_NOWAIT`) là 0, luồng gọi sẽ tạm ngừng thực thi cho đến khi một trong những điều sau xảy ra:
 - Một tin nhắn được đặt trên hàng đợi
 - `msqid` nhận dạng hàng đợi tin nhắn bị xóa khỏi hệ thống; khi điều này xảy ra -1 được trả về

Sau khi hoàn thành thành công, các hành động sau được thực hiện đối với cấu trúc dữ liệu được liên kết với `msqid`:

- `msg_qnum` giảm đi 1.
- `msg_lpid` được đặt bằng ID tiến trình của tiến trình gọi.

Bao gồm

`xmk.h, sys / msg.h, sys / ipc.h`

Bộ nhớ dùng chung

Xilkernel hỗ trợ bộ nhớ chia sẻ XSI được cấp phát hạt nhân. XSI là X / Open System Interface, là một tập hợp các giao diện tùy chọn trong POSIX. Bộ nhớ dùng chung là một cơ chế IPC phổ biến, có độ trễ thấp. Các khôi bộ nhớ dùng chung được yêu cầu trong thời gian chạy phải được xác định và chỉ định trong quá trình cấu hình hệ thống. Từ đặc điểm kỹ thuật này, bộ nhớ đệm được cấp phát cho mỗi vùng bộ nhớ dùng chung. Bộ nhớ dùng chung hiện không được cấp phát động tại thời gian chạy.

Mô-đun này là tùy chọn và có thể được cấu hình trong hoặc ngoài trong quá trình đặc tả hệ thống. Tham khảo "[Định cấu hình bộ nhớ dùng chung](#)", [trang 51](#) để biết thêm chi tiết.

Tóm tắt chức năng bộ nhớ dùng chung

Danh sách sau đây cung cấp một bản tóm tắt được liên kết của các chức năng bộ nhớ dùng chung trong Xilkernel. Bạn có thể nhấp vào một chức năng để chuyển đến phần mô tả.

```
int shmget (key_t key, size_t size, int shmflg)
int shmctl (int shmid, int cmd, struct shmid_ds * buf)
void * shmat (int shmid, const void * shmaddr, int flag)
int shm_detach (void * shmaddr)
```

Mô tả chức năng bộ nhớ dùng chung

Giao diện bộ nhớ dùng chung Xilkernel được mô tả bên dưới.

Thận trọng! Các bộ đệm bộ nhớ được cấp phát bởi API bộ nhớ chia sẻ có thể không được căn chỉnh theo các xanh giới từ. Do đó, các cấu trúc không nên được ánh xạ tùy tiện vào các phân đoạn bộ nhớ dùng chung mà không kiểm tra xem các yêu cầu về căn chỉnh có được đáp ứng hay không.

int shmget (key_t key, size_t size, int shmflg)

Thông số

đư ợc sử dụng để xác định duy nhất vùng bộ nhớ dùng chung.
 kích thư ớc là kích thư ớc đư ợc yêu cầu của phân đoạn bộ nhớ dùng chung.
 shmflg chỉ định các tùy chọn tạo phân đoạn.

Lợi nhuận

Mã định danh bộ nhớ đư ợc chia sẻ không phủ định duy nhất khi thành công.
 -1 khi bị lỗi và đặt errno một cách thích hợp: errno có thể đư ợc đặt thành:

EEXIST - Mã định danh bộ nhớ dùng chung tồn tại cho khóa đối số như ng
 (shmflg và IPC_CREAT) và (shmflg và IPC_EXCL) thi không
 số không.

ENOTSUP - Shmflg không đư ợc hỗ trợ.

ENOENT - Mã định danh bộ nhớ dùng chung không tồn tại cho khóa đối số
 và (shmflg và IPC_CREAT) là 0.

Sự mô tả

Hàm shmget () trả về mã định danh bộ nhớ dùng chung đư ợc liên kết với khóa. Mã định danh bộ
 nhớ dùng chung, cấu trúc dữ liệu đư ợc liên kết và phân đoạn bộ nhớ dùng chung có kích thư ớc
 ít nhất là byte (xem sys / shm.h) đư ợc tạo cho khóa nếu một trong những điều sau là đúng:

khóa bằng IPC_PRIVATE.

khóa chưa có mã định danh bộ nhớ dùng chung đư ợc liên kết với nó và
 (shmflg và IPC_CREAT) khác 0.

Sau khi tạo, cấu trúc dữ liệu đư ợc liên kết với mã định danh bộ nhớ dùng
 chung mới sẽ đư ợc khởi tạo. Giá trị của shm_segsz đư ợc đặt bằng giá trị
 của kích thư ớc. Các giá trị của shm_lpid, shm_nattch, shm_cpid đều đư ợc
 khởi tạo thích hợp. Khi phân đoạn bộ nhớ chia sẻ đư ợc tạo, nó đư ợc khởi
 tạo với tất cả các giá trị bằng không. Ít nhất một trong các phân đoạn bộ
 nhớ dùng chung có sẵn trong hệ thống phải khớp chính xác với kích thư ớc
 đư ợc yêu cầu để cuộc gọi thành công. Khóa IPC_PRIVATE không đư ợc hỗ trợ.

Bao gồm

xmk.h, sys / shm.h, sys / ipc.h

int shmctl (int shmid, int cmd, struct shmid_ds * buf)

Thông số

shmid là định danh phân đoạn bộ nhớ dùng chung.
 cmd là lệnh cho chức năng điều khiển.
 buf là bộ đệm nơi i trạng thái đư ợc trả về.

Lợi nhuận

0 về thành công. Trạng thái đư ợc trả lại trong bộ đệm cho IPC_STAT.

-1 khi hỏng hóc và đặt errno một cách thích hợp: errno có thể đư ợc đặt thành EINVAL với
 các điều kiện sau:

- nếu shmid đê cập đến một phân đoạn bộ nhớ đư ợc chia sẻ không hợp lệ.
- nếu cmd hoặc các thông số khác không hợp lệ.

Sự mô tả

Hàm shmctl () cung cấp nhiều hoạt động điều khiển bộ nhớ dùng chung như đư ợc chỉ định
 bởi cmd. Các giá trị sau cho cmd có sẵn:

- IPC_STAT: đặt giá trị hiện tại của từng thành viên trong cấu trúc dữ liệu shmid_ds liên
 kết với shmid vào cấu trúc đư ợc buf trả tới.
Nội dung của cấu trúc đư ợc xác định trong sys / shm.h.
- IPC_SET không đư ợc hỗ trợ.
- IPC_RMID: xóa định danh bộ nhớ dùng chung do shmid chỉ định khỏi hệ thống và phá hủy phân
 đoạn bộ nhớ dùng chung và cấu trúc dữ liệu shmid_ds liên kết với nó. Không có thông báo
 nào đư ợc gửi đến các quy trình vẫn đư ợc đính kèm với phân đoạn.

Bao gồm

xmk.h, sys / shm.h, sys / ipc.h

```
void * shmat (int shmid, const void * shmaddr, int flag)
```

Thông số

shmid là định danh phân đoạn bộ nhớ dùng chung.

shmaddr được sử dụng để chỉ định vị trí, để định kèm phân đoạn bộ nhớ được chia sẻ. Điều này hiện không được sử dụng.

còn được sử dụng để chỉ định các tùy chọn đính kèm bộ nhớ dùng chung (SHM) .

Lợi nhuận

Địa chỉ bắt đầu của phân đoạn bộ nhớ được chia sẻ khi thành công.

NULL khi thất bại và đặt lỗi một cách thích hợp. errno có thể được đặt thành EINVAL nếu shmid đề cập đến một phân đoạn bộ nhớ được chia sẻ không hợp lệ

Sự mô tả

shmat () tăng giá trị của shm_nattch trong cấu trúc dữ liệu được liên kết với ID bộ nhớ dùng chung của phân đoạn bộ nhớ dùng chung được đính kèm và trả về địa chỉ bắt đầu của phân đoạn. shm_lpid cũng được thiết lập thích hợp.

Lưu ý: các đối số shmaddr và cờ không được sử dụng.

Bao gồm

xmk.h, sys / shm.h, sys / ipc.h

```
int shm_dt (void * shmaddr)
```

Thông số

shmaddr là địa chỉ phân đoạn bộ nhớ dùng chung sẽ được tách ra.

Lợi nhuận

0 về thành công.

-1 về lỗi và đặt lỗi thích hợp. Errno có thể được đặt thành EINVAL nếu shmaddr không nằm trong bất kỳ phân đoạn bộ nhớ dùng chung nào có sẵn.

Sự mô tả

Hàm shmdt () tách đoạn bộ nhớ được chia sẻ nằm tại địa chỉ được chỉ định bởi shmaddr khỏi không gian địa chỉ của quá trình gọi. Giá trị của shm_nattch cũng bị giảm. Đoạn bộ nhớ không bị xóa khỏi hệ thống và có thể được gắn vào lại.

Bao gồm

xmk.h, sys / shm.h, sys / ipc.h

Mutex Locks

Xilkernel cung cấp hỗ trợ cho các khóa mutex luồng POSIX được phân bổ hạt nhân. Cơ chế đồng bộ hóa này có thể được sử dụng cùng với API pthread_. Số lượng khóa mutex và độ dài của hàng đợi khóa mutex có thể được định cấu hình trong quá trình đặc tả hệ thống.

Khóa mutex loại PTHREAD_MUTEX_DEFAULT và PTHREAD_MUTEX_RECURSIVE được hỗ trợ. Mô-đun này cũng là tùy chọn và có thể được cấu hình trong hoặc ngoài trong quá trình đặc tả hệ thống. Tham khảo "[Định cấu hình bộ nhớ dùng chung](#)", [trang 51](#) để biết thêm chi tiết.

Tóm tắt chức năng khóa Mutex

Danh sách sau đây cung cấp một bản tóm tắt được liên kết về các khóa Mutex trong Xilkernel. Bạn có thể nhấp vào một chức năng để chuyển đến phần mô tả.

```
int pthread_mutex_init (pthread_mutex_t * mutex, const pthread_mutexattr_t * attr)
int pthread_mutex_destroy (pthread_mutex_t * mutex)
int pthread_mutex_lock (pthread_mutex_t * mutex)
int pthread_mutex_trylock (pthread_mutex_t * mutex)
int pthread_mutex_unlock (pthread_mutex_t * mutex)
int pthread_mutexattr_init (pthread_mutexattr_t * attr)
int pthread_mutexattr_destroy (pthread_mutexattr_t * attr)
int pthread_mutexattr_settype (pthread_mutexattr_t * attr, kiểu int)
int pthread_mutexattr_gettype (pthread_mutexattr_t * attr, int * type)
```

Mô tả chức năng khóa Mutex

Mô tả chức năng khóa Mutex như sau:

```
int pthread_mutex_init (pthread_mutex_t * mutex, const
pthread_mutexattr_t * attr)
```

Thông số

mutex là vị trí nơi i mã định danh của khóa mutex mới được tạo sẽ được lưu trữ.

attr là cấu trúc thuộc tính tạo mutex.

Lợi nhuận

0 về thành công và số nhận dạng mutex trong * mutex.

EAGAIN nếu hệ thống hết tài nguyên.

Sự mô tả

Hàm pthread_mutex_init () khởi tạo mutex được tham chiếu bởi mutex với các thuộc tính được chỉ định bởi attr. Nếu attr là NULL, các thuộc tính mutex mặc định được sử dụng; hiệu ứng giống như việc chuyển địa chỉ của một đối tượng thuộc tính mutex mặc định.

Hãy tham khảo quy trình pthread_mutexattr_, được ghi lại bắt đầu từ trang 36 để xác định loại thuộc tính tạo mutex nào có thể được thay đổi. Sau khi khởi tạo thành công, trạng thái của mutex sẽ được khởi tạo và mở khóa. Chỉ bắn thân mutex mới có thể được sử dụng để thực hiện đồng bộ hóa. Kết quả của việc tham chiếu đến các bản sao của mutex trong các lệnh gọi đến pthread_mutex_lock (), pthread_mutex_trylock (), pthread_mutex_unlock () và pthread_mutex_destroy () không định nghĩa được.

Cố gắng khởi tạo mutex đã được khởi tạo dẫn đến hành vi không xác định. Trong trường hợp các thuộc tính mutex mặc định là phù hợp, macro PTHREAD_MUTEX_INITIALIZER có thể được sử dụng để khởi tạo mutex được cấp phát tĩnh. Hiệu ứng này tương đương với việc khởi tạo động bằng một cuộc gọi tới pthread_mutex_init () với tham số được chỉ định là NULL, với ngoại lệ là không có kiểm tra lỗi nào được thực hiện.

Ví dụ:

```
static pthread_mutex_t foo_mutex =
PTHREAD_MUTEX_INITIALIZER;
```

Bao gồm

xmk.h, pthread.h

Lưu ý: Các khóa mutex được phân bổ bởi Xilkernel tuân theo ngữ nghĩa của PTHREAD_MUTEX_DEFAULT mutex khóa theo mặc định. Các hành động sau sẽ dẫn đến hành vi không xác định:

- Đang có găng khóa mutex một cách đệ quy.
- Đang có găng mở khóa mutex nếu nó không bị khóa bởi chuỗi cuộc gọi.
- Có găng mở khóa mutex nếu nó không bị khóa.

`int pthread_mutex_destroy (pthread_mutex_t * mutex)`

Thông số	mutex là định danh mutex.
Lợi nhuận	0 về thành công. EINVAL nếu mutex đề cập đến một số nhận dạng không hợp lệ.
Sự mô tả	Hàm pthread_mutex_destroy () hủy đổi tư ợng mutex được tham chiếu bởi mutex; trên thực tế, đổi tư ợng mutex trở nên chưa được khởi tạo. Đổi tư ợng mutex bị phá hủy có thể được khởi động lại bằng cách sử dụng pthread_mutex_init (); kết quả của việc tham chiếu khác đến đổi tư ợng sau khi nó đã bị phá hủy là không xác định.
	Lưu ý: Bỏ qua trạng thái khóa / mở khóa Mutex trong quá trình tiêu diệt. Không có sự xem xét nào được đưa ra cho các quá trình chờ đợi.

Bao gồm `xmk.h, pthread.h`

`int pthread_mutex_lock (pthread_mutex_t * mutex)`

Thông số	mutex là định danh mutex.
Lợi nhuận	0 khi thành công và mutex ở trạng thái bị khóa. EINVAL trên tham chiếu mutex không hợp lệ . -1 về các lỗi chưa được xử lý.
Sự mô tả	Đổi tư ợng mutex được tham chiếu bởi mutex bị khóa bởi luồng gọi pthread_mutex_lock (). Nếu mutex đã bị khóa, chuỗi gọi sẽ chặn cho đến khi mutex khả dụng.
	Nếu loại mutex là PTHREAD_MUTEX_RECURSIVE, thì mutex duy trì khái niệm về số lư ợng khóa. Khi một chuỗi có được mutex thành công lần đầu tiên, số lư ợng khóa được đặt thành một. Mỗi khi một chuỗi khóa lại mutex này, số lư ợng khóa sẽ tăng lên một.
	Mỗi lần chuỗi mở khóa mutex, số lư ợng khóa sẽ giảm đi một.
	Nếu loại mutex là PTHREAD_MUTEX_DEFAULT, việc cố gắng khóa đệ quy mutex dẫn đến hành vi không xác định. Nếu thành công, thao tác này trả về với đổi tư ợng mutex được tham chiếu bởi mutex ở trạng thái bị khóa.
Bao gồm	<code>xmk.h, pthread.h</code>

```
int pthread_mutex_trylock (pthread_mutex_t * mutex)
```

Thông số	mutex là định danh mutex.
Lợi nhuận	0 về thành công. mutex ở trạng thái bị khóa. EINVAL trên tham chiếu mutex không hợp lệ, EBUSY nếu mutex đã bị khóa. -1 về các lỗi chưa được xử lý.
Sự mô tả	Đối tượng mutex được tham chiếu bởi mutex bị khóa bởi luồng gọi pthread_mutex_trlock (). Nếu mutex đã bị khóa, chuỗi gọi trả về ngay lập tức với EBUSY. Nếu loại mutex là PTHREAD_MUTEX_RECURSIVE, thì mutex duy trì khái niệm về số lượng khóa.
	Khi một chuỗi có được mutex thành công lần đầu tiên, số lượng khóa được đặt thành một.
Bao gồm	Mỗi khi một chuỗi khóa lại mutex này, số lượng khóa sẽ tăng lên một. Mỗi lần chuỗi mở khóa mutex, số lượng khóa sẽ giảm đi một. Nếu loại mutex là PTHREAD_MUTEX_DEFAULT, việc cố gắng khóa để quy mutex dẫn đến hành vi không xác định. Nếu thành công, thao tác này trả về với đối tượng mutex được tham chiếu bởi mutex ở trạng thái bị khóa.

```
int pthread_mutex_unlock (pthread_mutex_t * mutex)
```

Thông số

mutex là định danh mutex.

Lợi nhuận

0 khi thành công, EINVAL trên tham chiếu mutex không hợp lệ.
-1 về lỗi không xác định.

Sự mô tả

Hàm pthread_mutex_unlock () giải phóng đối tượng mutex được tham chiếu bởi mutex. Nếu có các luồng bị chặn trên đối tượng mutex được tham chiếu bởi mutex khi pthread_mutex_unlock () được gọi, dẫn đến việc mutex trở nên khả dụng, chính sách lập lịch sẽ xác định luồng nào sẽ nhận được mutex. Nếu đó là SCHED_RR, thì luồng nằm ở đầu hàng đợi mutex sẽ được bỏ chặn và được phép khóa mutex.

Nếu loại mutex là PTHREAD_MUTEX_RECURSIVE, mutex duy trì khái niệm về số lượng khóa. Khi số lượng khóa đạt đến 0, mutex sẽ có sẵn cho các luồng khác lấy. Nếu một chuỗi có gắng mở khóa mutex mà nó chưa khóa hoặc mutex đã được mở khóa, thì sẽ trả về lỗi.

Nếu loại mutex là PTHREAD_MUTEX_DEFAULT, các hành động sau dẫn đến hành vi không xác định:

- Có gắng mở khóa mutex nếu nó không bị khóa bởi cuộc gọi chủ đề.
- Có gắng mở khóa mutex nếu nó không được khóa.

Nếu thành công, thao tác này trả về với đối tượng mutex được tham chiếu bởi mutex ở trạng thái mở khóa.

Bao gồm

xmk.h, pthread.h

```
int pthread_mutexattr_init (pthread_mutexattr_t * attr)
```

Thông số

attr là vị trí của cấu trúc thuộc tính.

Lợi nhuận

0 về thành công.

EINVAL nếu attr đề cập đến một vị trí không hợp lệ.

Sự mô tả

Hàm pthread_mutexattr_init () khởi tạo đối tượng thuộc tính mutex attr với giá trị mặc định cho tất cả các thuộc tính được xác định bởi quá trình triển khai.

Tham khảo sys / styles.h để biết nội dung của pthread_mutexattr kết cấu.

Lưu ý: Quy trình này không liên quan đến lệnh gọi vào hạt nhân.

Bao gồm

xmk.h, pthread.h

```
int pthread_mutexattr_destroy (pthread_mutexattr_t * attr)
```

Thông số	attr là vị trí của cấu trúc thuộc tính.
Lợi nhuận	0 về thành công. EINVAL nếu attr đề cập đến một vị trí không hợp lệ.
Sự mô tả	Hàm pthread_mutexattr_destroy () hủy một đối tượng thuộc tính mutex; đối tượng trở thành, trên thực tế, không được khởi tạo. Lưu ý: Quy trình này không liên quan đến lệnh gọi vào hạt nhân.
Bao gồm	xmk.h, pthread.h

```
int pthread_mutexattr_settype (pthread_mutexattr_t * attr,  
    kiểu int)
```

Thông số	attr là vị trí của cấu trúc thuộc tính.
Lợi nhuận	loại là loại để đặt mutex. 0 về thành công. EINVAL nếu attr đề cập đến một vị trí không hợp lệ hoặc nếu loại là một loại không được hỗ trợ.
Sự mô tả	Hàm pthread_mutexattr_settype () đặt kiểu mutex trong cấu trúc thuộc tính mutex thành kiểu được chỉ định. Chỉ PTHREAD_MUTEX_DEFAULT và PTHREAD_MUTEX_RECURSIVE được hỗ trợ. Lưu ý: Quy trình này không liên quan đến lệnh gọi vào hạt nhân.
Bao gồm	xmk.h, pthread.h

```
int pthread_mutexattr_gettype (pthread_mutexattr_t * attr,  
    kiểu int *)
```

Thông số	attr là vị trí của cấu trúc thuộc tính.
Lợi nhuận	type là một con trỏ đến vị trí để lưu trữ mutex. 0 về thành công. EINVAL nếu attr đề cập đến một vị trí không hợp lệ.
Sự mô tả	Hàm pthread_mutexattr_gettype () nhận loại mutex trong cấu trúc thuộc tính mutex và lưu trữ nó ở vị trí được trả đến theo loại.
Bao gồm	xmk.h, pthread.h

Quản lý bộ nhớ đệm động

Kernel cung cấp một lược đồ cấp phát bộ nhớ đệm, có thể được sử dụng bởi các ứng dụng cần cấp phát bộ nhớ động. Các giao diện này là lựa chọn thay thế cho các quy trình cấp phát bộ nhớ C tiêu chuẩn - malloc (), free () chậm hơn và lớn hơn nhiều, mặc dù mạnh hơn. Các quy trình cấp phát xử lý các phần bộ nhớ từ một nhóm bộ nhớ mà người dùng chuyển đến trình quản lý bộ nhớ đệm.

Trình quản lý bộ nhớ đệm quản lý nhóm bộ nhớ. Bạn có thể tự động tạo các nhóm bộ nhớ mới. Bạn cũng có thể chỉ định tinh các kích thước khối bộ nhớ khác nhau và số lượng khối bộ nhớ như vậy cần thiết cho các ứng dụng của bạn. Tham khảo “[Định cấu hình phân bổ bộ nhớ đệm](#)”, trang 52 để biết thêm chi tiết. Phương pháp quản lý bộ đệm này tương đối đơn giản, nhỏ và là cách cấp phát bộ nhớ nhanh chóng. Sau đây là các giao diện cấp phát bộ nhớ đệm. Mô-đun này là tùy chọn và có thể đưa vào trong quá trình khởi tạo hệ thống.

Tóm tắt chức năng quản lý bộ nhớ đệm động

Danh sách sau đây cung cấp một bản tóm tắt được liên kết về các chức năng quản lý bộ nhớ đệm động trong Xilkernel. Bạn có thể nhấp vào một chức năng để chuyển đến phần mô tả.

```
int bufcreate (membuf_t * mbuf, void * memptr, int nblks, size_t blksiz)
int bufdestroy (membuf_t mbuf)
void * bufmalloc (membuf_t mbuf, size_t siz)
void buffree (membuf_t mbuf, void * mem)
```

Thận trọng! API cấp phát bộ nhớ đệm nội bộ sử dụng nhóm bộ nhớ do người dùng cung cấp để lưu trữ tại chỗ danh sách trống trong nhóm bộ nhớ. Do đó, chỉ các kích thước bộ nhớ lớn hơn hoặc bằng 4 byte dài mới được hỗ trợ bởi các API cấp phát bộ nhớ đệm. Ngoài ra, bởi vì có một danh sách trống được xây dựng tại chỗ trong nhóm bộ nhớ, các yêu cầu trong đó kích thước khối bộ nhớ không phải là bội số của 4 byte gây ra sự không điều chỉnh trong thời gian chạy. Nếu nền tảng phần mềm của bạn có thể xử lý việc không căn chỉnh nguyên bản hoặc thông qua các trường hợp ngoại lệ thì điều này không có vấn đề gì. Các bộ đệm bộ nhớ được cấp phát và trả về bởi API cấp phát bộ nhớ đệm cũng có thể không được căn chỉnh theo các ranh giới từ. Do đó, ứng dụng của bạn không nên tự ý ánh xạ cấu trúc vào bộ nhớ được cấp phát theo cách này mà không kiểm tra trước xem các yêu cầu về căn chỉnh và đệm có được đáp ứng hay không.

Mô tả chức năng quản lý bộ nhớ đệm động

Mô tả chức năng quản lý bộ nhớ đệm động như sau:

```
int bufcreate (membuf_t * mbuf, void * memptr, int nblk, size_t
               blksiz)
```

Thông số

`mbuf` là vị trí lưu mã định danh của vùng bộ nhớ được tạo.

`memptr` là vùng bộ nhớ để sử dụng.

`nblk` là số khối bộ nhớ mà nhóm này sẽ hỗ trợ. `blksiz` là kích thước của mỗi khối bộ nhớ tính bằng byte.

Lợi nhuận

0 khi thành công và lưu trữ mã định danh của nhóm bộ nhớ đã tạo ở vị trí được chỉ ra bởi `mbuf`.

-1 về lỗi.

Sự mô tả

Tạo một nhóm bộ nhớ ngoài khối bộ nhớ được chỉ định trong `memptr`. `nblk` số lượng bộ nhớ được xác định trong nhóm, mỗi khối có kích thước . Do đó, `memptr` phải trả đến ít nhất (`nblk * blksiz`) byte bộ nhớ. `blksiz` phải lớn hơn hoặc bằng 4.

Bao gồm

xmk.h, sys / bufmalloc.h

int bufdestroy (membuf_t mbuf)

Thông số

`mbuf` là định danh của vùng nhớ cần hủy.

Lợi nhuận

0 về thành công.

-1 về lỗi.

Sự mô tả

Quy trình này sẽ phá hủy nhóm bộ nhớ được xác định bởi `mbuf`.

Bao gồm

xmk.h, sys / bufmalloc.h

void * bufmalloc (membuf_t mbuf, size_t siz)

Thông số

`mbuf` là định danh của vùng bộ nhớ để cấp phát bộ nhớ.

`siz` là kích thước của khối bộ nhớ được yêu cầu.

Lợi nhuận

Địa chỉ bắt đầu của khối bộ nhớ khi thành công.

NULL khi bị lỗi và đặt `errno` một cách thích hợp: `errno` được đặt thành:

- `EINVAL` nếu `mbuf` tham chiếu đến bộ đệm bộ nhớ không hợp lệ.
- `EAGAIN` nếu yêu cầu không được đáp ứng.

Sự mô tả

Phân bổ một phần bộ nhớ từ nhóm bộ nhớ được chỉ định bởi `mbuf`. Nếu `mbuf` là `MEMBUF_ANY`, thì tất cả các vùng nhớ có sẵn sẽ được tìm kiếm cho yêu cầu và vùng đầu tiên có một khối miễn phí có kích thước `siz`, được sử dụng và cấp phát từ đó.

Bao gồm

xmk.h, sys / bufmalloc.h

```
void buffree (membuf_t mbuf, void * mem)
```

Thông số	mbuf là định danh của vùng bộ nhớ.
Lợi nhuận	mem là địa chỉ của khối bộ nhớ.
Sự mô tả	Giải phóng bộ nhớ được cấp phát bởi một lệnh gọi tư ứng trả bufmalloc. Nếu mbuf là MEMBUF_ANY, hãy trả bộ nhớ về nhóm đáp ứng yêu cầu này.
	Nếu không, hãy trả bộ nhớ về nhóm đã chỉ định.
	Hành vi không được xác định nếu các giá trị tùy ý được chỉ định cho mem.
Bao gồm	xmk.h, sys / bufmalloc.h

Bộ hẹn giờ phần mềm

Xilkernel cung cấp chức năng hẹn giờ của phần mềm, để xử lý liên quan đến thời gian. Môđun này là tùy chọn và có thể được cấu hình trong hoặc ngoài. Tham khảo "[Định cấu hình bộ hẹn giờ phần mềm](#)", [trang 52](#) để biết thêm thông tin về cách tùy chỉnh môđun này. Các giao diện sau có sẵn với môđun bộ hẹn giờ phần mềm.

```
unsigned int xget_clock_ticks ( )
```

Thông số	Không có.
Lợi nhuận	Số lượng tích tắc hạt nhân đã trôi qua kể từ khi hạt nhân được khởi động.
Sự mô tả	Một lần đánh dấu được đếm, mỗi khi bộ đếm thời gian hạt nhân đưa ra một ngắt. Điều này được lưu trữ trong một số nguyên 32-bit và cuối cùng bị tràn. Lệnh gọi xget_clock_ticks () trả về thông tin đánh dấu này, mà không truyền tải các lỗi tràn đã xảy ra.
Bao gồm	xmk.h, sys / timer.h

```
time_t time (time_t * bộ đếm thời gian)
```

Thông số	bộ đếm thời gian chỉ đến vị trí bộ nhớ để lưu thông tin thời gian được yêu cầu.
Lợi nhuận	Số giây đã trôi qua kể từ khi hạt nhân được khởi động.
Sự mô tả	Thời gian thường lệ trôi qua kể từ khi hạt nhân bắt đầu tính bằng đơn vị giây. Điều này cũng có thể bị tràn.
Bao gồm	xmk.h, sys / timer.h

ngủ không dấu (unsigned int ms)

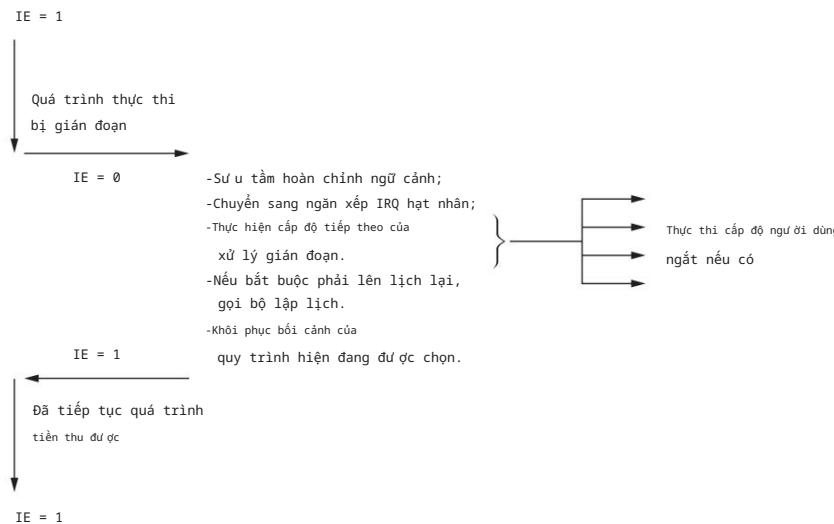
Thông số	ms là số mili giây để ngủ.
Lợi nhuận	Số giây giữa các lần ngủ.
	0 về thành công hoàn toàn.
Sự mô tả	Quy trình này khiến quy trình gọi chuyển sang trạng thái ngủ trong một số mili giây được chỉ định.
Bao gồm	xmk.h, sys / timer.h

Ngắt

Sự điều khiển

Xilkernel loại bỏ các yêu cầu xử lý ngắt chính từ ứng dụng người dùng.

Mặc dù hạt nhân hoạt động mà không có bất kỳ ngắt nào, hệ thống chỉ có ý nghĩa khi nó được điều khiển bởi ít nhất một ngắt bộ định thời để lập lịch. Kernel xử lý ngắt bộ đếm thời gian chính, sử dụng nó làm dấu chọn của hạt nhân để thực hiện việc lập lịch. Ngắt bộ định thời được khởi tạo và gắn với mã vectơ trong quá trình khởi tạo hệ thống. Xung hạt nhân này cũng cung cấp các phương tiện hẹn giờ phần mềm và các quy trình liên quan đến thời gian. Ngoài ra, Xilkernel có thể xử lý nhiều ngắt khi được kết nối thông qua bộ điều khiển ngắt và hoạt động với lối bộ điều khiển ngắt xps_intc. Hình dưới đây cho thấy một dịch vụ ngắt cơ bản trong Xilkernel.



Hình 4: Dịch vụ ngắt cơ bản trong Xilkernel

Kích bản xử lý ngắt được minh họa trong sơ đồ này. Khi bị gián đoạn:

- Ngữ cảnh của tiến trình đang thực thi được lưu vào vùng lưu ngữ cảnh. • Ngắt bị vô hiệu hóa từ thời điểm này trở đi, cho đến khi chúng được kích hoạt khi kết thúc quá trình xử lý ngắt.
- Điều này làm giảm bớt gánh nặng ngăn xếp của quy trình, vì việc thực thi trong thời gian gián đoạn, không sử dụng ngăn xếp ứng dụng người dùng.
- Ngữ cảnh ngắt này có thể được coi như một luồng nhân đặc biệt thực thi các trình xử lý ngắt theo thứ tự. Luồng này bắt đầu sử dụng không gian ngăn xếp thực thi riêng biệt của nó.
- Ngăn xếp thực thi hạt nhân riêng biệt có kích thước tối thiểu là 1 KB để cho phép nó xử lý các mức lồng sâu bên trong các trình xử lý ngắt. Ngăn xếp hạt nhân này cũng được tự động cấu hình để sử dụng kích thước ngăn xếp pthread do người dùng chọn, nếu nó lớn hơn 1 KB. Nếu bạn thấy truớc việc sử dụng ngăn xếp lớn trong các trình xử lý ngắt của mình, bạn sẽ cần chỉ định một giá trị lớn cho pthread_stack_size.

Điều này kết thúc mức xử lý ngắt đầu tiên của hạt nhân. Tại thời điểm này, hạt nhân chuyển quyền điều khiển đến trình xử lý ngắt mức thứ hai. Đây là quy trình xử lý ngắt chính của bộ điều khiển ngắt. Từ điểm này, trình xử lý cho bộ điều khiển ngắt gọi người dùng xử lý ngắt được chỉ định cho các thiết bị ngoại vi ngắt khác nhau.

Trong hạt nhân bộ xử lý MicroBlaze, nếu bộ đếm thời gian hệ thống được kết nối thông qua bộ điều khiển ngắt, thì hạt nhân xử lý vô hình ngắt bộ định thời chính (dánh dấu hạt nhân), bằng cách đăng ký chính nó làm bộ xử lý cho ngắt đó.

Các trình xử lý ngắt có thể thực hiện bất kỳ loại hành động xử lý ngắt nào được yêu cầu, bao gồm cả việc thực hiện các lệnh gọi hệ thống. Tuy nhiên, bộ xử lý không bao giờ được gọi các lệnh gọi hệ thống chặn, hoặc toàn bộ hạt nhân bị chặn và hệ thống ở trạng thái tạm ngừng. Sử dụng trình xử lý một cách khôn ngoan để thực hiện xử lý tối thiểu khi bị gián đoạn.

Thận trọng! Trình xử lý ngắt mức người dùng không được thực hiện các cuộc gọi hệ thống chặn. Các cuộc gọi hệ thống đã thực hiện, nếu có, sẽ không bị chặn.

Sau khi trình xử lý ngắt cấp người dùng được phục vụ, trình xử lý ngắt cấp đầu tiên trong nhân sẽ được điều khiển trở lại. Nó xác định xem việc xử lý ngắt trước đó có gây ra yêu cầu lặp lại trong hạt nhân hay không.

Nếu có một yêu cầu như vậy, nó sẽ gọi bộ lập lịch hạt nhân và thực hiện việc lặp lịch thích hợp. Sau khi bộ lập lịch đã xác định quy trình tiếp theo sẽ thực thi, ngữ cảnh của quy trình mới sẽ được khôi phục và các ngắt được kích hoạt trở lại.

Lưu ý: Hiện tại, Xilkernel chỉ hỗ trợ bộ điều khiển ngắt gắn với chân ngắt bên ngoài của bộ xử lý PowerPC 405. Nó không hỗ trợ bộ điều khiển ngắt gắn với chân đầu vào quan trọng của bộ xử lý.

Khi Xilkernel được sử dụng với nhiều ngắt trong hệ thống, API xử lý ngắt cấp người dùng Xilkernel sẽ khả dụng. Phần phụ sau liệt kê các API xử lý ngắt cấp người dùng.

API xử lý ngắt mức người dùng

Tóm tắt chức năng API xử lý ngắt mức người dùng

Danh sách sau cung cấp bản tóm tắt được liên kết về các API xử lý ngắt cấp người dùng trong Xilkernel. Bạn có thể nhấp vào một chức năng để chuyển đến phần mô tả.

```
unsigned int register_int_handler (id int_id_t, void * handler) (void *, void * callback)
void unregister_int_handler (id int_id_t)
void enable_interrupt (id int_id_t)
void disable_interrupt (id int_id_t)
void accept_interrupt (id int_id_t)
```

Mô tả chức năng API xử lý ngắt mức người dùng

Các mô tả API xử lý ngắt như sau:

```
unsigned int register_int_handler (id int_id_t , void
* xử lý) (void *), void * callback)
```

Thông số

id là id số dựa trên 0 của ngắt.

trình xử lý là trình xử lý cấp người dùng.

callback là một giá trị gọi lại có thể được phân phối tới trình xử lý cấp người dùng.

Lợi nhuận

XST_SUCCESS thành công.

mã lỗi được xác định trong xstatus.h.

Sự mô tả

Hàm register_int_handler () đăng ký trình xử lý ngắt mức người dùng để được chỉ định làm trình xử lý cho một ngắt được chỉ định. Quy trình cấp người dùng để được gọi không đồng bộ khi được phục vụ bởi bộ điều khiển ngắt trong hệ thống. Quy trình trả về lỗi trên hệ thống bộ xử lý MicroBlaze nếu id là mã định danh cho ngắt bộ hẹn giờ hệ thống.

Hệ thống bộ xử lý PowerPC có ngắt bộ định thời phần cứng chuyên dụng tồn tại riêng biệt với các ngắt khác trong hệ thống. Do đó, việc kiểm tra này không được thực hiện đối với hệ thống bộ xử lý PowerPC.

Bao gồm

xmk.h, sys / intr.h

void unregister_int_handler (id int_id_t)

Thông số

id là id số dựa trên 0 của ngắt.

Lợi nhuận

Không có.

Sự mô tả

Hàm unregister_int_handler () hủy đăng ký trình xử lý ngắt cấp người dùng đã đăng ký làm trình xử lý ngắt được chỉ định. Quy trình không làm gì cả và không hoạt động một cách âm thầm trên hệ thống bộ xử lý MicroBlaze nếu id là mã định danh cho ngắt bộ đếm thời gian hệ thống.

Bao gồm

xmk.h, sys / intr.h

void enable_interrupt (id int_id_t)

Thông số

id là id số dựa trên 0 của ngắt.

Lợi nhuận

Không có.

Sự mô tả

Hàm enable_interrupt () cho phép ngắt được chỉ định trong bộ điều khiển ngắt. Quy trình không làm gì cả và không hoạt động một cách âm thầm trên hệ thống bộ xử lý MicroBlaze, nếu id là mã định danh cho ngắt bộ đếm thời gian hệ thống.

Bao gồm

xmk.h, sys / intr.h

void disable_interrupt (id int_id_t)

Thông số	id là id số dựa trên 0 của ngắt.
Lợi nhuận	Không có.
Sự mô tả	Hàm disable_interrupt () vô hiệu hóa ngắt được chỉ định trong bộ điều khiển ngắt. Quy trình không làm gì cả và không hoạt động một cách âm thầm trên hệ thống bộ xử lý MicroBlaze nếu id là mã định danh cho ngắt bộ đếm thời gian hệ thống.
Bao gồm	xmk.h, sys / intr.h

void accept_interrupt (id int_id_t)

Thông số	id là định danh số dựa trên 0 của ngắt.
Lợi nhuận	Không có.
Sự mô tả	Hàm accept_interrupt () xác nhận việc xử lý ngắt được chỉ định cho bộ điều khiển ngắt. Quy trình không làm gì cả và không hoạt động một cách âm thầm trên hệ thống bộ xử lý MicroBlaze nếu id là mã định danh cho ngắt bộ đếm thời gian hệ thống.
Bao gồm	xmk.h, sys / intr.h

**Ngoại lệ
Sự điều khiển**

Xilkernel xử lý các ngoại lệ cho bộ xử lý MicroBlaze, coi chúng như các điều kiện lỗi do các quy trình / luồng đang thực thi. Xilkernel giết quá trình lỗi và báo cáo bằng cách sử dụng một thông báo tới bảng điều khiển (nếu chế độ tiết được bật) về bản chất của ngoại lệ. Bạn không thể đăng ký trình xử lý của riêng mình cho những ngoại lệ này và Xilkernel xử lý tất cả chúng theo cách nguyên bản.

Xilkernel không xử lý các truger hợp ngoại lệ cho bộ xử lý PowerPC. API và mô hình xử lý ngoại lệ có sẵn cho nền tảng Độc lập có sẵn cho Xilkernel. Bạn có thể muốn đăng ký trình xử lý hoặc đặt các điểm ngắt (trong khi gỡ lỗi) cho các truger hợp ngoại lệ mà bạn quan tâm.

**Kỉ niệm
Sự bảo vệ**

Bảo vệ bộ nhớ là một tính năng cực kỳ hữu ích có thể tăng cường độ mạnh mẽ, độ tin cậy và khả năng chịu lỗi cho ứng dụng dựa trên Xilkernel của bạn. Bảo vệ bộ nhớ yêu cầu hỗ trợ từ phần cứng. Xilkernel được thiết kế để sử dụng các tính năng của Đơn vị Quản lý (Bảo vệ) Bộ nhớ MicroBlaze (MMU) khi có sẵn. Điều này cho phép bạn xây dựng các ứng dụng không an toàn mà mỗi ứng dụng chạy trong hộp cát hợp lệ của hệ thống, được xác định bởi tệp thực thi và các thiết bị I / O khả dụng.

Lưu ý: Quản lý bộ nhớ ảo đầy đủ không được Xilkernel hỗ trợ. Ngay cả khi có đầy đủ MMU trên bộ xử lý MicroBlaze, chỉ các bản dịch bộ nhớ trong suốt mới được sử dụng và không có khái niệm phân trang nhu cầu.

Lưu ý: Xilkernel không hỗ trợ tính năng Bảo vệ Bộ nhớ trên bộ xử lý PowerPC.

Tổng quan về Bảo vệ Bộ nhớ

Khi tham số MicroBlaze C_USE_MMU được đặt thành $= 2$, hạt nhân sẽ tự động cấu hình bảo vệ bộ nhớ trong quá trình khởi động.

Lưu ý: Để tắt chức năng bảo vệ bộ nhớ trong hạt nhân, hãy thêm cờ trình biên dịch -D XILKERNEL_MB_MPUR_DISABLE, vào thư viện và bản dựng ứng dụng của bạn.

Kernel xác định ba loại vi phạm bảo vệ:

1. Vi phạm mã - xảy ra khi một luồng có gắng thực thi từ bộ nhớ không được xác định để chứa các hướng dẫn chương trình.

Lưu ý: Vì Xilkernel là một tệp thực thi duy nhất, tất cả các luồng đều có quyền truy cập vào tất cả các lệnh của chương trình và hạt nhân không thể bẫy các vi phạm trong đó một luồng bắt đầu thực thi trực tiếp mã nhân.

2. Vi phạm quyền truy cập dữ liệu - Xảy ra khi một chuỗi cố gắng đọc hoặc ghi dữ liệu đến hoặc từ bộ nhớ không được xác định là một phần của không gian dữ liệu chương trình. Tương tự, các phân đoạn dữ liệu chỉ đọc có thể được bảo vệ bằng cách truy cập ghi bởi tất cả các luồng.

Lưu ý: Bởi vì Xilkernel là một tệp thực thi duy nhất, tất cả các luồng đều có quyền truy cập như nhau vào tất cả dữ liệu cũng như cấu trúc dữ liệu hạt nhân. Kernel không thể bẫy các vi phạm trong đó một luồng truy cập dữ liệu mà nó không được chỉ định để xử lý.

3. Vi phạm I / O - xảy ra khi một luồng cố gắng đọc hoặc ghi từ không gian I / O ngoại vi được ánh xạ bộ nhớ không có trong hệ thống.

Xilkernel cố gắng xác định ba khu vực bảo vệ khái niệm này trong chương trình và hệ thống của bạn trong quá trình xây dựng hệ thống và thời gian khởi động hạt nhân một cách tự động. Hạt nhân cố gắng xác định mã và nhãn dữ liệu phân ranh giới mã và các phần dữ liệu trong tệp ELF thực thi của bạn. Các nhãn này thường được cung cấp bởi các tập lệnh của trình liên kết.

Ví dụ: các tập lệnh của trình liên kết MicroBlaze sử dụng các nhãn _ftext và _etext để chỉ ra phần đầu và phần cuối của phần .text tương ứng.

Bảng sau đây tóm tắt các phần hợp lý phải có trong tập lệnh trình liên kết, các yêu cầu về cẩn chỉnh của mỗi phần và các nhãn phân chia ranh giới.

Bảng 1: Các phần logic của tập lệnh liên kết

Tiết diện	Bắt đầu nhãn	Nhãn kết thúc	Sự mô tả
.chữ	_ftext	_etext	Các phần hướng dẫn thực thi
.dữ liệu	_fdata	_edata	Đọc-ghi các phần dữ liệu bao gồm các phần dữ liệu nhỏ
.rodata	_frodata	_erodata	Chỉ đọc các phần dữ liệu bao gồm các phần dữ liệu nhỏ
.cây rơm	_stack_end	_cây rơm	Ngăn xếp nhân với trang bảo vệ 1 KB ở trên và dưới
trang bảo vệ ngắn xếp (trên cùng)	_fstack_guard_top	_estack_guard_top	Trang bảo vệ ngắn xếp nhân trên cùng (1 KB)
trang bảo vệ ngắn xếp (dưới cùng)	_fstack_guard_bottom	_estack_guard_bottom	Trang bảo vệ ngắn xếp nhân dưới cùng (1 KB)

Mỗi phần phải được cẩn chỉnh ở ranh giới 1 KB và được phân ranh giới rõ ràng bằng các nhãn được chỉ định. Nếu không, Xilkernel sẽ bỏ qua các phần logic bị thiếu mà không có thông báo lỗi hoặc cảnh báo.

Thận trọng! Hành vi này có thể tự biểu hiện trong việc phần mềm của bạn không hoạt động như mong đợi, vì các mục đích MPU sẽ bị thiếu đối với các phần ELF quan trọng và bộ xử lý sẽ coi các yêu cầu hợp lệ là không hợp lệ.

Lưu ý: Mỗi phần thư ờng có một loại dữ liệu cụ thể dự kiến sẽ có mặt. Nếu logic của dữ liệu được chèn vào các phần bằng tập lệnh trình liên kết của bạn không phù hợp, thì khả năng bảo vệ do hạt nhân cung cấp có thể không chính xác hoặc mức độ bảo vệ có thể bị pha loãng.

Các phạm vi I / O được tự động liệt kê bởi các công cụ tạo thư viện và được cung cấp như một cấu trúc dữ liệu cho hạt nhân. Các phạm vi I / O ngoại vi này sẽ không bao gồm các vùng bộ nhớ đọc / ghi vì các điều khiển truy cập cho bộ nhớ được xác định tự động từ tệp ELF.

Trong quá trình khởi động hạt nhân, các phạm vi I / O liệt kê được đánh dấu là có thể đọc và ghi được bởi các luồng. Các truy cập bên ngoài phạm vi I / O đã xác định gây ra lỗi bảo vệ.

Bảo vệ do ngư ời dùng chỉ định

Ngoài việc thiết lập vùng bảo vệ và suy luận tự động do hạt nhân thực hiện, bạn có thể cung cấp các vùng bảo vệ của riêng mình bằng cách cung cấp cấu trúc dữ liệu như thể hiện trong ví dụ sau. Nếu tính năng này không được yêu cầu, các cấu trúc dữ liệu này có thể bị xóa khỏi mã ứng dụng.

```
#include <mpu.h>

int user_io_nranges = 2;
xilkernel_io_range_t user_io_range [1] = {{0x25004000, 0x25004fff,
MPU_PROT_READWRITE},
{0x44000000, 0x44001fff, MPU_PROT_NONE}};
```

Loại xilkernel_io_ranges_t được định nghĩa như sau:

```
typedef struct xilkernel_io_range_s {
    không dấu int baseaddr;
    không dấu int highaddr;
    cờ int không dấu;
} xilkernel_io_range_t;
```

Bảng sau liệt kê các cờ tru ờng hợp lệ xác định các tùy chọn bảo vệ quyền truy cập do ngư ời dùng chỉ định:

Bảng 2: Cờ tru ờng bảo vệ truy cập

Cờ tru ờng	Sự mô tả
MPU_PROT_EXEC	Hư ờng dẫn chương trình có thể thực thi (không có quyền đọc hoặc ghi)
MPU_PROT_READWRITE	Phần dữ liệu có thể đọc và có thể ghi (không có quyền thực thi)
MPU_PROT_READ	Phần dữ liệu chỉ đọc (không có quyền ghi / thực thi)
MPU_PROT_NONE	(Hiện tại không có trang nào có thể được bảo vệ khỏi cả ba quyền truy cập cùng một lúc. Cờ tru ờng này tương ứng với MPU_PROT_READ)

Đã sửa lỗi Hỗ trợ bộ đệm nhìn sang bên ngoài bản dịch thông nhất (TLB) trên Bộ xử lý MicroBlaze

Bộ xử lý MicroBlaze có Bộ đệm nhìn sang một bên bản dịch thông nhất 64 mục nhập cố định (TLB). Xilkernel chỉ có thể hỗ trợ số TLB tối đa này. Nếu vượt quá TLBs tối đa để kích hoạt bảo vệ cho một vùng nhất định, Xilkernel sẽ thông báo lỗi trong quá trình khởi tạo Bộ vi xử lý (MPU) và tiến hành khởi động hạt nhân mà không cần bảo vệ bộ nhớ. Không hỗ trợ quản lý TLB hoán đổi động để cung cấp một số vùng bảo vệ tùy ý.

Các giao diện khác Nội bộ, Xilkernel, phụ thuộc vào nền tảng Độc lập; do đó, các giao diện mà

Những món quà độc lập đư ợc kế thừa bởi Xilkernel. Tham khảo tài liệu "[Độc lập](#)" để biết thông tin về các giao diện có sẵn. Ví dụ: để thêm trình xử lý tùy chỉnh của riêng bạn cho các ngoại lệ khác nhau mà bộ xử lý PowerPC 405 hỗ trợ, bạn sẽ sử dụng giao diện xử lý ngoại lệ do Độc lập cung cấp cho bộ xử lý PowerPC 405.

Phần cứng

Yêu cầu

Xilkernel đã đư ợc thiết kế để hoạt động với luồng phần cứng và phần mềm EDK. Nó đư ợc tích hợp hoàn toàn với cấu hình nền tảng phần mềm và cơ chế tạo thư viện tự động. Do đó, một nền tảng phần mềm dựa trên Xilkernel có thể đư ợc định cấu hình và xây dựng chỉ trong vài phút. Tuy nhiên, một số dịch vụ trọng nhân yêu cầu hỗ trợ từ phần cứng.

Lập lịch và tất cả các tính năng phụ thuộc yêu cầu đánh dấu nhân định kỳ và thường một số loại bộ đếm thời gian đư ợc sử dụng. Xilkernel đã đư ợc thiết kế để hoạt động với lõi IP Xilinx fit_timer hoặc lõi IP xps_timer. Bằng cách chỉ định tên phiên bản của thiết bị hẹn giờ trong cấu hình nền tảng phần mềm, Xilkernel có thể tự động khởi tạo và sử dụng lõi bộ hẹn giờ cũng như các dịch vụ liên quan đến bộ hẹn giờ. Tham khảo "[Định cấu hình bộ hẹn giờ hệ thống](#)", trang 53 để biết thêm thông tin về cách chỉ định thiết bị hẹn giờ. Trên bộ xử lý PowerPC 405 và PowerPC 440, Xilkernel sử dụng bộ định thời có thể lập trình bên trong của bộ xử lý và do đó không cần lõi bộ định thời bên ngoài cho chức năng hạt nhân; Tuy nhiên, bạn phải chỉ định các giá trị cho tần số bộ đếm thời gian hệ thống và khoảng thời gian bộ đếm thời gian hệ thống.

Xilkernel cũng đã đư ợc thiết kế để hoạt động trong các tình huống liên quan đến các thiết bị ngoại vi nhiều ngắt. Lõi IP xps_intc xử lý các ngắt phần cứng và cung cấp một đường IRQ duy nhất từ bộ điều khiển đến bộ xử lý. Bằng cách chỉ định tên của thiết bị ngoại vi bộ điều khiển ngắt trong cấu hình nền tảng phần mềm, bạn sẽ nhận biết hạt nhân về nhiều ngắt. Xilkernel sẽ tự động khởi tạo lõi phần cứng, hệ thống ngắt và mức thứ hai của trình xử lý phần mềm như một phần của quá trình khởi động. Bạn không phải làm điều này theo cách thủ công. Xilkernel xử lý bộ điều khiển ngắt không phân tầng; bộ điều khiển ngắt theo tầng không đư ợc hỗ trợ.

Hệ thống Khởi tạo

Điểm vào của hạt nhân là quy trình xilkernel_main () đư ợc định nghĩa trong main.c. Bất kỳ khởi tạo người dùng nào phải đư ợc thực hiện đều có thể đư ợc thực hiện trước lệnh gọi xilkernel_main (). Điều này bao gồm bất kỳ tính năng nào trên toàn hệ thống có thể cần đư ợc bật trước khi gọi xilkernel_main (). Đây thường là các tính năng ở trạng thái máy như kích hoạt bộ nhớ cache hoặc kích hoạt ngoại lệ phần cứng phải "luôn BẬT" ngay cả khi chuyển đổi ngữ cảnh giữa các ứng dụng. Đảm bảo thiết lập các trạng thái hệ thống như vậy trước khi gọi xilkernel_main (). Về mặt khái niệm, quy trình xilkernel_main () thực hiện hai việc: nó khởi tạo kernel thông qua xilkernel_init () và sau đó khởi động kernel bằng xilkernel_start (). Hành động đầu tiên đư ợc thực hiện trong xilkernel_init () là khởi tạo phần cứng cụ thể của hạt nhân. Điều này bao gồm đăng ký trình xử lý ngắt và cấu hình bộ đếm thời gian hệ thống, cũng như khởi tạo bảo vệ bộ nhớ. Ngắt / ngoại lệ không đư ợc bật sau khi hoàn thành hw_init (). Tiếp theo, quy trình sys_init () đư ợc nhập vào. Quy trình này thực hiện khởi tạo từng mô-đun, chẳng hạn như quy trình và luồng, khởi tạo theo thứ tự sau:

1. Cấu trúc ngữ cảnh quy trình nội bộ
2. Hàng đợi sẵn sàng
3. mô-đun pthread
4. Mô-đun Semaphore
5. Mô-đun hàng đợi tin nhắn
6. Mô-đun bộ nhớ dùng chung
7. Mô-đun cấp phát bộ nhớ
8. Mô-đun hẹn giờ phần mềm

9. Tạo tác vụ nhàn rỗi

10. Tạo pthread tĩnh

Sau các bước này, `xilkernel_start()` được gọi nơi ngắn và ngoại lệ được kích hoạt. Hạt nhân lặp lại vô hạn trong tác vụ nhàn rỗi, cho phép bộ lập lịch bắt đầu các quá trình lập lịch.

An toàn và Re Entrancy

Xilkernel, theo định nghĩa, tạo ra một môi trường đa luồng. Nhiều thư viện và quy trình điều khiển có thể không được viết theo cách an toàn theo luồng hoặc theo cách đăng nhập lại. Ví dụ bao gồm các thư viện C như `printf()`, `sprintf()`, `malloc()`, `free()`. Khi sử dụng bất kỳ thư viện hoặc API trình điều khiển nào không phải là một phần của Xilkernel, bạn phải đảm bảo xem xét các tính năng an toàn luồng và hấp dẫn của quy trình. Một cách phổ biến để ngăn chặn hành vi không chính xác với các quy trình không an toàn là bảo vệ việc xâm nhập vào quy trình bằng các khóa hoặc semaphores.

Những hạn chế

- Không thể sử dụng các ứng dụng dấu chấm động với Xilkernel trên PPC440 và PPC405
bởi vì xử lý. Hạn chế này là do Xilkernel không chuyển đổi ngữ cảnh các thanh ghi dấu chấm động và thanh ghi trạng thái / điều khiển dấu chấm động trên các bộ xử lý này. Một bản phát hành trong tương lai sẽ nhằm bổ sung hỗ trợ này cho Xilkernel.
Các ứng dụng dấu chấm động có thể được sử dụng một cách an toàn với bộ xử lý MicroBlaze vì bộ xử lý MicroBlaze không có bộ đăng ký khác cho các giá trị dấu chấm động.
- Trình biên dịch bộ xử lý MicroBlaze hỗ trợ chuyển đổi -mxl-ngăn xếp-kiểm tra, có thể được sử dụng để bắt lỗi tràn ngăn xếp. Tuy nhiên, công tắc này chỉ hoạt động với các ứng dụng đơn luồng, vì vậy nó không thể được sử dụng trong Xilkernel.

Kernel Tùy biến

Xilkernel có khả năng tùy biến cao. Như đã mô tả trong các phần trước, bạn có thể thay đổi các mô-đun và các tham số riêng cho phù hợp với ứng dụng của mình. Xử lý nền tảng Xilinx (XPS) hộp thoại Cài đặt Nền tảng Phần mềm cung cấp một phương pháp cấu hình dễ dàng cho các tham số Xilkernel. Tham khảo chương "Tổng quan về Kiến trúc Hệ thống và Công cụ Nhúng" trong "Sổ tay Tham khảo Công cụ Hệ thống Nhúng" để biết thêm chi tiết (liên kết đến tài liệu có sẵn trong "Tài nguyên Bổ sung", trang 3). Để tùy chỉnh một mô-đun trong nhân, một tham số có tên của danh mục được đặt thành TRUE phải được xác định trong tệp Đặc tả Phần mềm Vi xử lý (MSS). Một ví dụ để tùy chỉnh pthread như sau:

```
tham số config_pthread_support = true
```

Nếu bạn không xác định tham số config_ có thể định cấu hình cho mô-đun, mô-đun đó không được triển khai. Bạn không phải nhập các thông số và giá trị này theo cách thủ công. Khi bạn nhập thông tin vào hộp thoại Cài đặt Nền tảng Phần mềm , XPS sẽ tự động tạo các mục nhập tệp Đặc tả Phần mềm Vi xử lý (MSS) tương ứng.

Sau đây là đoạn mã tệp MSS để định cấu hình OS Xilkernel cho hệ thống PowerPC. Các giá trị trong đoạn mã là các giá trị mẫu nhằm mục tiêu đến một bảng giả định:

```
BEGIN OS
PARAMETER OS_NAME = xilkernel
PARAMETER OS_VER = 3,00.a
THÔNG SỐ STDIN = RS232
THÔNG SỐ STDOUT = RS232
PARAMETER proc_instance = ppc405_0
PARAMETER config_debug_support = true
PARAMETER verbose = true
PARAMETER systmr_spec = true
PARAMETER systmr_freq = 100000000
PARAMETER systmr_interval = 80
PARAMETER sysintc_spec = system_intc
PARAMETER config_sched = true
PARAMETER Sched_type = SCHED_PRIO
PARAMETER n_prio = 6
PARAMETER max_readyq = 10
PARAMETER config_pthread_support = true
PARAMETER max_pthreads = 10
PARAMETER config_sema = true
PARAMETER max_sem = 4
PARAMETER max_sem_waitq = 10
PARAMETER config_msgq = true
PARAMETER num_msgqs = 1
PARAMETER msgq_capacity = 10
PARAMETER config_bufmalloc = true
PARAMETER config_pthread_mutex = true
PARAMETER config_time = true
PARAMETER max_tmrs = 10
PARAMETER Enhance_features = true
PARAMETER config_kill = true
PARAMETER mem_table = ((4,30), (8,20))
PARAMETER static_pthread_table = ((shell_main, 1))

CHẤM DỨT
```

Các tham số cấu hình trong đặc tả MSS ảnh hưởng đến bộ nhớ và kích thước mã của hình ảnh Xilkernel. Các cấu trúc được cấp phát hạt nhân mà số lượng có thể được cấu hình thông qua MSS phải được xem xét để đảm bảo rằng kích thước bộ nhớ và mã của bạn phù hợp với thiết kế của bạn.

Ví dụ, số lượng tối đa cấu trúc ngữ cảnh quy trình được phân bổ trong nhân được xác định bằng tổng của hai tham số; max_procs và max_pthreads. Nếu cấu trúc ngữ cảnh quy trình chiếm x byte bộ nhớ bss, thì tổng yêu cầu bộ nhớ bss cho ngữ cảnh quy trình là $(max_pthreads * x)$ byte. Do đó, các tham số như vậy phải được điều chỉnh cẩn thận và bạn cần kiểm tra hình ảnh hạt nhân cuối cùng bằng tiện ích kích thước GNU để đảm bảo rằng các yêu cầu bộ nhớ của bạn được đáp ứng. Để biết được đóng góp của mỗi cấu trúc được phân bổ hạt nhân đối với yêu cầu bộ nhớ, hãy xem lại tệp tiêu đề tương ứng. Đặc tả trong MSS được dịch bởi các tệp Libgen và Xilkernel Tcl thành các chỉ thị cấu hình ngôn ngữ C trong hai tệp tiêu đề: os_config.h và config_init.h. Xem lại hai tệp này, được tạo trong thư mục bao gồm bộ xử lý chính, để hiểu cách thông số kỹ thuật được dịch.

Cấu hình STDIN và STDOUT

Các thiết bị ngoại vi đầu vào và đầu ra tiêu chuẩn có thể được cấu hình cho Xilkernel. Xilkernel có thể hoạt động mà không cần đầu vào và đầu ra tiêu chuẩn. Các thiết bị ngoại vi này là mục tiêu của các API đầu vào-đầu ra

như print, outbyte và inbyte. Bảng sau cung cấp các mô tả thuộc tính, kiểu dữ liệu và giá trị mặc định.

Bảng 3: Thông số cấu hình STDIN / STDOUT

Thuộc tính	Sự mô tả	Loại hình	Mặc định
stdin	Tên phiên bản của thiết bị ngoại vi stdin.	sợi dây	không ai
stdout	Tên phiên bản của thiết bị ngoại vi stdout.	sợi dây	không ai

Định cấu hình lập lịch trình

Bạn có thể cấu hình chính sách lập lịch hạt nhân bằng cách cấu hình các tham số hiển thị trong bảng sau.

Bảng 4: Các thông số lập lịch trình

Thuộc tính	Sự mô tả	Loại hình	Mặc định
config_sched	Định cấu hình mô-đun bộ lập lịch.	boolean	thật
Sched_type	Loại Bộ lập lịch sẽ được sử dụng. Giá trị được phép: 2 - SCHED_RR 3 - SCHED_PRIO	enum	SCHED_RR
n_prio	Số mức độ ưu tiên nếu lập lịch là SCHED_PRIO.	số	32
max_readyq	Độ dài của mỗi hàng đợi sẵn sàng. Đây là số lượng quy trình tối đa có thể hoạt động trong một hàng đợi sẵn sàng bất kỳ lúc nào ngay lập tức.	số	10

Định cấu hình quản lý chuỗi

Luồng là cơ chế chính để tạo bối cảnh quy trình. Các tham số có thể cấu hình của mô-đun luồng được liệt kê trong bảng sau.

Bảng 5: Thông số mô-đun luồng

Thuộc tính	Sự mô tả	Loại hình	Mặc định
config_pthread_support	Cần mô-đun pthread.	boolean	thật
max_threads	Số lượng chủ đề tối đa có thể được phân bổ tại bất kỳ thời điểm nào.	số	10
pthread_stack_size	Kích thước ngăn xếp cho các luồng được tạo động (tính bằng byte).	số	1000
static(pthread_table	Định cấu hình tĩnh các luồng khởi động khi hạt nhân được khởi động. Đây được định nghĩa là một mảng với mỗi phần tử chứa các tham số pthread_start_addr và pthread_prio. Lưu ý: Nếu bạn đang chỉ định tên hàm cho pthread_start_addr, chúng phải là các hàm trong mã nguồn của bạn được biên dịch bằng phươn ngữ C. Chúng không thể là các hàm được biên dịch bằng phươn ngữ C++.	mảng 2-tuples	không ai

Bảng 5: Tham số mô-đun luồng (Tiếp theo)

Thuộc tính	Sự mô tả	Loại hình	Mặc định
pthread_start_addr	Địa chỉ bắt đầu chuỗi.	Tên hàm (chuỗi)	không ai
pthread_prio	Ưu tiên hàng đầu.	số	không ai

Định cấu hình Semaphores

Bạn có thể cấu hình mô-đun semaphores, số lượng semaphores tối đa và độ dài hàng đợi semaphore. Bảng sau đây cho thấy các thông số được sử dụng để cấu hình.

Bảng 6: Thông số mô-đun Semaphore

Thuộc tính	Sự mô tả	Loại hình	Mặc định
config_sema	Cần mô-đun Semaphore.	boolean	sai
max_sem	Số lượng Hội thảo tối đa.	số	10
max_sem_waitq	Độ dài Hàng đợi Semaphore.	số	10
config_name_sema	Định cấu hình hỗ trợ semaphore có tên trong hạt nhân.	boolean	sai

Định cấu hình hàng đợi tin nhắn

Theo tùy chọn, bạn có thể định cấu hình mô-đun hàng đợi tin nhắn, số lượng hàng đợi tin nhắn và kích thước của mỗi hàng đợi tin nhắn. Mô-đun hàng đợi thông báo phụ thuộc vào cả mô-đun semaphore và mô-đun cấp phát bộ nhớ đệm. Bảng sau đây cho thấy các định nghĩa tham số được sử dụng để cấu hình. Bộ nhớ cho thông báo phải được chỉ định rõ ràng trong tùy chỉnh malloc hoặc được tạo trong thời gian chạy.

Bảng 7: Thông số mô-đun hàng đợi thông báo

Thuộc tính	Sự mô tả	Loại hình	Mặc định
config_msgq	Cần mô-đun Hàng đợi Thư.	boolean	sai
num_msgqs	Số hàng đợi tin nhắn trong hệ thống.	số	10
msgq_capacity	Số lượng thư tối đa trong hàng đợi.	số	10
use_malloc	Cung cấp các hàng đợi tin nhắn mạnh mẽ hơn sử dụng malloc và miễn phí cấp phát bộ nhớ cho các tin nhắn.	boolean	sai

Định cấu hình bộ nhớ dùng chung

Theo tùy chọn, bạn có thể định cấu hình mô-đun bộ nhớ dùng chung và kích thước của từng đoạn bộ nhớ dùng chung. Tất cả các phân đoạn bộ nhớ chia sẻ cần thiết phải được chỉ định trong các tham số này. Bảng sau đây cho thấy các tham số được sử dụng để cấu hình.

Bảng 8: Thông số mô-đun bộ nhớ dùng chung

Thuộc tính	Sự mô tả	Loại hình	Mặc định
config_shm	Cần mô-đun bộ nhớ được chia sẻ.	boolean	sai
shm_table	Bảng bộ nhớ dùng chung. Được định nghĩa là một mảng với mỗi phần tử có shm_size tham số.	mảng 1 bộ	không ai

Bảng 8: Thông số mô-đun bộ nhớ dùng chung (Tiếp theo)

Thuộc tính	Sự mô tả	Loại hình	Mặc định
shm_size	Kích thước bộ nhớ dùng chung.	số	không ai
num_shm	Số bộ nhớ được chia sẻ được biểu thị bằng kích thước mảng shm_table .	số	không ai

Cấu hình Pthread Mutex Locks

Theo tùy chọn, bạn có thể chọn bao gồm mô-đun mutex pthread, số lượng khóa mutex và kích thước của hàng đợi cho các khóa mutex. Bảng sau đây cho thấy các thông số được sử dụng để cấu hình.

Bảng 9: Thông số mô-đun Pthread Mutex

Thuộc tính	Sự mô tả	Loại hình	Mặc định
config_pthread_mutex	Cần mô-đun mutex pthread.	boolean	sai
max_pthread_mutex	Số lượng khóa mutex pthread tối đa có sẵn trong hệ thống.	số	10
max_pthread_mutex_waitq	Độ dài của mỗi hàng đợi khóa mutex.	số	10

Định cấu hình phân bổ bộ nhớ đệm

Theo tùy chọn, bạn có thể định cấu hình mô-đun quản lý bộ nhớ đệm động, kích thước khối bộ nhớ và số khối bộ nhớ. Bảng sau đây cho thấy các thông số được sử dụng để cấu hình.

Bảng 10: Thông số mô-đun quản lý bộ nhớ

Thuộc tính	Sự mô tả	Loại hình	Mặc định
config_bufmalloc	Cần quản lý bộ nhớ đệm.	boolean	sai
max_bufs	Hạt nhân có thể quản lý số lượng vùng đệm tối đa bất kỳ lúc nào.	số	10
mem_table	Bảng khối bộ nhớ. Đây được định nghĩa là một mảng với mỗi phần tử có các tham số mem_bsize , mem_nblk .	mảng 2 bộ	không ai
mem_bsize	Kích thước khối bộ nhớ.	số	không ai
mem_nblk	Số lượng khối bộ nhớ có kích thước.	số	không ai

Cấu hình bộ hẹn giờ phần mềm

Theo tùy chọn, bạn có thể định cấu hình mô-đun bộ hẹn giờ phần mềm và số bộ hẹn giờ tối đa được hỗ trợ. Bảng sau đây cho thấy các thông số được sử dụng để cấu hình.

Bảng 11: Thông số mô-đun bộ hẹn giờ phần mềm

Thuộc tính	Sự mô tả	Loại hình	Mặc định
config_time	Cần phần mềm hẹn giờ và phân hệ quản lý thời gian.	boolean	sai
max_tmrs	Số bộ định thời phần mềm tối đa trong nhân.	số	10

Định cấu hình giao diện nâng cao

Theo tùy chọn, bạn có thể định cấu hình một số tính năng / giao diện nâng cao bằng cách sử dụng các tham số sau được hiển thị trong bảng sau.

Bảng 12: Các tính năng nâng cao

Thuộc tính	Sự mô tả	Loại hình	Mặc định
config_kill	Bao gồm khả năng hủy một tiến trình bằng hàm kill () .	boolean	sai
config_yield	Bao gồm giao diện gain () .	boolean	sai

Cấu hình bộ hẹn giờ hệ thống

Bạn có thể cấu hình thiết bị hẹn giờ trong hệ thống cho các hạt nhân bộ xử lý MicroBlaze. Ngoài ra, bạn có thể định cấu hình khoảng thời gian hẹn giờ cho các hệ thống vi xử lý MicroBlaze dựa trên bộ hẹn giờ PowerPC và PIT. Bảng sau đây cho thấy các thông số có sẵn.

Bảng 13: Các thuộc tính để sao chép tệp nguồn hạt nhân

Thuộc tính	Sự mô tả	Loại hình	Mặc định
systmr_dev1	Tên phiên bản của thiết bị ngoại vi hẹn giờ hệ thống.	sợi dây	không ai
systmr_freq	Chỉ định tần số đồng hồ của thiết bị hẹn giờ hệ thống: <ul style="list-style-type: none"> Đối với xps_timer, nó là tần số đồng hồ OPB. Đối với fit_timer, nó là đồng hồ được cấp cho fit_timer. Đối với bộ xử lý PowerPC 405, đó là tần số của PowerPC 405. 	số	100000000
systmr_interval	Khoảng thời gian cho mỗi lần ngắt bộ định thời hệ thống. Điều này được xác định tự động (và không thể thay đổi) cho fit_timer.	số (mili giây)	10

1. Chỉ MicroBlaze.

Định cấu hình xử lý ngắn

Bạn có thể cấu hình thiết bị điều khiển ngắn trong nhân hệ thống. Việc thêm tham số này sẽ tự động định cấu hình hỗ trợ nhiều ngắn và API xử lý ngắn cấp ngay lập tức dùng trong nhân. Điều này cũng làm cho hạt nhân tự động khởi tạo bộ điều khiển ngắn. Bảng sau đây cho thấy các tham số được triển khai.

Bảng 14: Các thuộc tính để sao chép tệp nguồn hạt nhân

Thuộc tính	Sự mô tả	Loại hình	Mặc định
sysintc_spec	Chỉ định tên phiên bản của thiết bị điều khiển ngắn được kết nối với cổng ngắn bên ngoài.	sợi dây	vô giá trị

Định cấu hình thông báo gỡ lỗi

Bạn có thể cấu hình để hạt nhân xuất ra các thông báo gỡ lỗi / chẩn đoán thông qua luồng thực thi của nó. Việc kích hoạt tham số trong bảng sau sẽ làm cho macro DBG_PRINT khả dụng và sau đó đầu ra của nó tới thiết bị đầu ra tiêu chuẩn: Bảng 15: Thuộc tính cho thông báo gỡ lỗi

Thuộc tính	Sự mô tả	Loại hình	Mặc định
chế độ kiểm tra sửa lỗi	Bật thông báo gỡ lỗi hạt nhân.	boolean	sai

Sao chép tệp nguồn hạt nhân

Bạn có thể sao chép các tệp nguồn nhân đã được định cấu hình vào kho lưu trữ của mình để chỉnh sửa thêm và sử dụng chúng để xây dựng nhân. Bảng sau đây cho thấy các tham số được triển khai: Bảng 16:

Các thuộc tính để sao chép tệp nguồn hạt nhân

Thuộc tính	Sự mô tả	Loại hình	Mặc định
copyoutfiles	Cần sao chép các tệp nguồn.	boolean	sai
copytodir	Thư mục kho lưu trữ ngưng chờ dùng. Đường dẫn liên quan đến project_directory / system_name / libsrc / xilkernel_v4_00_a / src_dir.	chuỗi đường dẫn ".../copyoflib"	

Gỡ lỗi Xilkernel

Toàn bộ hình ảnh hạt nhân là một tệp duy nhất có thể dùng làm mục tiêu để gỡ lỗi bằng cơ chế EDK GNU Debugger (GDB). Các ứng dụng ngưng chờ dùng và thư viện phải được biên dịch bằng -g. Tham khảo Sổ tay Tham khảo Công cụ Hệ thống Nhúng để biết tài liệu về cách gỡ lỗi ứng dụng với GDB. Liên kết đến tài liệu này có sẵn trong "[Tài nguyên bổ sung](#)", trang 3.

Lưu ý: Phương pháp gỡ lỗi này liên quan đến khả năng hiển thị lớn đối với hạt nhân và có thể xâm nhập. Ngoài ra, lợc đồ gỡ lỗi này không được ứng dụng nhân-ngưng chờ dùng biết.

Kỉ niệm Dấu chân

Kích thước của Xilkernel phụ thuộc vào cấu hình ngưng chờ dùng. Nó có kích thước nhỏ và có thể phù hợp với các cấu hình khác nhau. Bảng sau đây cho thấy đầu ra kích thước bộ nhớ từ tiện ích kích thước GNU cho hạt nhân. Xilkernel đã được thử nghiệm với cờ tối ưu hóa Bộ sưu tập trình biên dịch GNU (GCC) của -O2; các số trong bảng có cùng mức độ tối ưu hóa.

Bảng 17: Cấu hình ngưng chờ dùng và kích thước kênh Xilker

Cấu hình	MicroBlaze (tính bằng kb)	PowerPC (tính bằng kb)
Chức năng hạt nhân cơ bản chỉ với đa luồng.	7	16
Chức năng hạt nhân đầy đủ với lập lịch vòng lặp (không hỗ trợ nhiều ngắt và không có tính năng nâng cao).	16	26
Chức năng hạt nhân đầy đủ với lập lịch ưu tiên (không hỗ trợ nhiều ngắt và không có tính năng nâng cao).	16,5	26,5
Chức năng hạt nhân đầy đủ với tất cả các mô-đun (luồng, hỗ trợ cho cả quy trình ELF, lập lịch ưu tiên, IPC, cấu trúc đồng bộ hóa, bộ đệm malloc, xử lý ngắt nhiều cấp và ngưng chờ dùng, trình điều khiển cho bộ điều khiển ngắt và bộ đếm thời gian, các tính năng nâng cao).	22	32

Tệp Xilkernel

Các nguồn Xilkernel được sắp xếp như trong bảng dưới đây:

Cơ quan

Bảng 18: Tổ chức các nguồn Xilkernel

nguồn gốc/				Chứa các thư mục / data và / src .
	dữ liệu/			Chứa Định nghĩa Thư viện Bộ vi xử lý (MLD) và các tệp Tcl xác định cấu hình XilKernel.
	src /			Chứa tất cả các nguồn.
	bao gồm/			Chứa các tệp tiêu đề được tổ chức tự động như / src .
	src /			Tệp nguồn không có tiêu đề.
		các nguồn cụ thể về kiến trúc / kiến trúc.		
		sys /		Nguồn cấp hệ thống.
		ipc /		Các nguồn triển khai chức năng IPC.

Sửa đổi Xilkernel

Bạn có thể tùy chỉnh thêm Xilkernel bằng cách thay đổi cơ sở mã thực tế. Để làm việc với bản sao tùy chỉnh của Xilkernel, trước tiên bạn phải sao chép thư mục nguồn Xilkernel_v4_00_a từ cài đặt EDK và đặt nó vào kho phần mềm; ví dụ: <.... / mylibraries / bsp / xilkernel_v4_00_a>. Nếu đang dẫn kho lưu trữ được thêm vào các công cụ, Libgen sẽ chọn thư mục nguồn của Xilkernel để biên dịch.

Tham khảo "[Tổ chức tệp Xilkernel](#)", trang 55 để biết thêm thông tin về tổ chức của các nguồn Xilkernel. Các nguồn Xilkernel đã được viết theo phong cách cơ bản và trực quan và bao gồm các khái niệm xét phía trên mỗi chức năng quan trọng. Mỗi tệp nguồn cũng mang một khái niệm riêng cho biết vai trò của nó.

Không được chấp nhận Đặc trưng

Quản lý quy trình ELF (Không được dùng nữa)

Một tính năng không được dùng nữa của Xilkernel là hỗ trợ tạo bối cảnh thực thi từ các tệp được liên kết có thể thực thi (ELF) riêng biệt.

Bạn có thể muốn làm điều này nếu bạn cần tạo các quy trình từ các tệp thực thi nằm trên hệ thống tệp (ví dụ: XilFATFS hoặc XilMFS). Thông thường, cần phải có bộ tài, mà Xilkernel không cung cấp. Giả sử rằng ứng dụng của bạn có liên quan đến bộ tài, sau đó được cung cấp một điểm vào trong bộ nhớ cho tệp thực thi, Xilkernel sau đó có thể tạo một quy trình. Kernel không phân bổ một ngăn xếp riêng biệt cho các quá trình như vậy; ngăn xếp được thiết lập như một phần của CRT của tệp thực thi riêng biệt.

Lưu ý: Các tệp ELF thực thi riêng biệt như vậy, được thiết kế để chạy trên Xilkernel, phải được biên dịch với cờ trình biên dịch -xl-mode-xilkernel cho bộ xử lý MicroBlaze. Đối với bộ xử lý PowerPC, bạn phải sử dụng tập lệnh trình liên kết tùy chỉnh, không bao gồm các phần .boot và .vectors trong hình ảnh ELF cuối cùng. Lý do mà các sửa đổi này được yêu cầu là theo mặc định, bất kỳ chương trình nào được biên dịch với dòng công cụ EDK GNU, đều có thể chứa các phần ghi đè lên phần vecto ngắn, ngoại lệ và đặt lại quan trọng trong bộ nhớ. Xilkernel yêu cầu hình ảnh ELF của chính nó khởi tạo các phần này và chúng vẫn nguyên vẹn. Sử dụng các cờ biên dịch đặc biệt này và các tập lệnh trình liên kết, xóa các phần này khỏi hình ảnh đầu ra cho các ứng dụng.

Chế độ thực thi riêng biệt có những lưu ý sau:

- Tối ưu hóa con trỏ toàn cục không được hỗ trợ.

Lưu ý: Điều này được hỗ trợ trong chế độ liên kết hạt nhân mặc định. Nó không chỉ được hỗ trợ trong chế độ thực thi riêng biệt này.

- Xilkernel không có bộ nạp khi tạo các quy trình và luồng mới. Nó tạo ra các ngữ cảnh quy trình và luồng để bắt đầu từ các hình ảnh bộ nhớ giả định được khởi tạo.

Do đó, nếu bất kỳ tệp ELF nào phụ thuộc vào các phần dữ liệu được khởi tạo, thì lần tiếp theo cùng một hình ảnh bộ nhớ được sử dụng để tạo một quy trình, các phần đã khởi tạo sẽ không hợp lệ, trừ khi một số cơ chế bên ngoài được sử dụng để tải lại hình ảnh ELF trước khi tạo quy trình.

Lưu ý: Tính năng này không được dùng nữa và Xilinx khuyến khích sử dụng mô hình ứng dụng tệp thực thi đơn, tiêu chuẩn.

Tham khảo phần "[Định cấu hình Quản lý Quy trình ELF \(Không được dùng nữa\)](#)," trang 56 để biết thêm chi tiết. Quy trình ELF được tạo và xử lý bằng cách sử dụng các giao diện sau.

```
int elf_process_create (void * start_addr, int prio)
```

Thông số	start_addr là địa chỉ bắt đầu của quá trình. prio là mức ưu tiên bắt đầu của tiến trình trong hệ thống.
Lợi nhuận	<ul style="list-style-type: none"> PID của quy trình mới về thành công. -1 khi thất bại.
Sự mô tả	Tạo một quy trình mới. Phân bổ PID và Khối điều khiển quy trình (PCB) mới cho quy trình. Quy trình được đặt trong hàng đợi sẵn sàng thích hợp.
Bao gồm	xmk.h, sys / process.h

```
int elf_process_exit (void)
```

Thông số	Không có.
Lợi nhuận	Không có.
Sự mô tả	Xóa quy trình khỏi hệ thống. Thận trọng! Không sử dụng chức năng này để kết thúc một chuỗi.
Bao gồm	xmk.h, sys / process.h

Định cấu hình quản lý quy trình ELF (Không được dùng nữa)

Bạn có thể chọn số lượng quy trình tối đa trong hệ thống và các chức năng khác nhau cần thiết để xử lý các quy trình. Các quy trình và luồng có trong hệ thống khi khởi động hệ thống có thể được cấu hình tĩnh. Bảng sau cung cấp danh sách các tham số có sẵn:

Bảng 19: Các thông số quản lý quy trình

Thuộc tính	Sự mô tả	Loại hình	Mặc định
config_elf_process Cần quản lý quy trình ELF. Lưu ý: Việc sử dụng config_elf_process yêu cầu nâng cao_features = true trong cấu hình hạt nhân.		boolean	thật
max_procs Số lượng quy trình tối đa trong hệ thống.		số 10	
static_elf_process _bàn	Định cấu hình các quy trình khởi động là các tệp thực thi riêng biệt. Đây được định nghĩa là một mảng với mỗi phần tử chứa các tham số process_start và process_prio.	Mảng 2-tuples	không ai
process_start_addr Địa chỉ bắt đầu quy trình.		Không có địa chỉ	
process_prio Quá trình ưu tiên.		Không có số	



Hệ thống FATFile LibXil (FATFS)

UG 648 ngày 15 tháng 4 năm 2009

Bản tóm tắt

Tài liệu này mô tả thư viện truy cập Hệ thống FATFile XilFatfs. Thư viện này cung cấp quyền truy cập đọc / ghi vào các tệp được lưu trữ trên thiết bị ổ đĩa nhỏ gọn hoặc flash Xilinx® System ACE™.

Tài liệu bao gồm các phần sau:

- "Tổng quan"
- "Tài nguyên bổ sung"
- "Tóm tắt chức năng XilFATFS"
- "Mô tả chức năng XilFATFS"
- "Tùy chỉnh Libgen"

Tổng quan

Thư viện truy cập hệ thống tệp XilFATFS cung cấp quyền truy cập đọc / ghi vào các tệp được lưu trữ trên thiết bị flash nhỏ gọn Xilinx System ACE hoặc thiết bị ổ đĩa nhỏ của IBM. Thư viện này yêu cầu nền tảng phần cứng bên dưới phải chứa những thứ sau:

- Bộ điều khiển giao diện ACE hệ thống XPS - Môđun Logicore

- Bộ điều khiển ACE hệ thống và đầu nối CompactFlash
- Thẻ CompactFlash hoặc IBM Microdrive được định dạng bằng tệp FAT12, FAT16 hoặc FAT32 hệ thống

Thận trọng! FAT16 là cần thiết để System ACE cấu hình trực tiếp FPGA như ng thư viện XilFATFS có thể hoạt động với phần cứng System ACE để hỗ trợ cả FAT12 và FAT32.

Xem tài liệu về Bộ điều khiển Giao diện ACE của Hệ thống XPS trong Hướng dẫn Tham khảo IP của Bộ xử lý để biết thêm chi tiết về phần cứng.

Bạn có thể sao chép tệp dễ dàng vào thiết bị flash từ PC của mình bằng cách cắm flash hoặc microdrive vào bộ điều hợp USB phù hợp hoặc thiết bị tương tự.

Nếu flash compact hoặc microdrive có nhiều phân vùng, mỗi phân vùng được định dạng là hệ thống tệp FAT12, FAT16 hoặc FAT32, thì XilFATFS cho phép các phân vùng được truy cập bằng tên phân vùng. Phân vùng đầu tiên luôn được gọi là A:, phân vùng thứ hai luôn được gọi là B:, v.v. Như đã lưu ý trước đó, phân vùng đầu tiên phải là FAT16 để System ACE cấu hình trực tiếp FPGA.

© 2009 Xilinx, Inc. XILINX, biểu trưng Xilinx, Virtex, Spartan, ISE và các nhãn hiệu được chỉ định khác có trong tài liệu này là nhãn hiệu của Xilinx tại Hoa Kỳ và các quốc gia khác. Tất cả các nhãn hiệu khác là tài sản của chủ sở hữu tương ứng của họ.

Xilinx sẽ tiết lộ hướng dẫn sử dụng, hướng dẫn sử dụng, ghi chú phát hành và / hoặc thông số kỹ thuật này ("Tài liệu") cho bạn chỉ để sử dụng trong việc phát triển các thiết kế để vận hành với các thiết bị phần cứng của Xilinx. Bạn không được phép sao chép, phân phối, tái xuất bản, tái xuất bản, hiển thị, đăng hoặc truyền Tài liệu dưới bất kỳ hình thức nào hoặc bằng bất kỳ phương tiện nào bao gồm nhưng không giới hạn ở Xilinx. Xilinx từ chối rõ ràng bất kỳ trách nhiệm pháp lý nào phát sinh từ việc bạn sử dụng Tài liệu. Xilinx bảo lưu quyền, theo quyết định riêng của mình, thay đổi Tài liệu mà không cần thông báo bất kỳ lúc nào. Xilinx không có nghĩa vụ phải sửa bất kỳ lỗi nào có trong Tài liệu hoặc thông báo cho bạn về bất kỳ chỉnh sửa hoặc cập nhật nào. Xilinx từ chối rõ ràng bất kỳ trách nhiệm pháp lý nào liên quan đến hỗ trợ kỹ thuật hoặc hỗ trợ có thể được cung cấp cho bạn liên quan đến Thông tin.

TÀI LIỆU ĐƯỢC CÔNG BỐ CHO BẠN "NGUYÊN TRẠNG" KHÔNG ĐƯỢC BẢO HÀNH BẤT KỲ HÌNH THỨC NÀO. XILINX KHÔNG CÓ BẢO ĐÁM KHÁC, DÙ THẾ HIỆN, NGƯ Ý HOẶC TRUYỀN NGHĨA, LIÊN QUAN ĐẾN TÀI LIỆU, BAO GỒM BẤT KỲ BẢO ĐÁM NÀO VỀ TÍNH KHẢ NĂNG, PHÙ HỢP VỚI MỤC ĐÍCH CỤ THỂ HOẶC KHÔNG ĐÁM BẢO QUYỀN CỦA BÊN THỨ BA. TRONG MỌI TRƯỜNG HỢP XILINX SẼ CHỊU TRÁCH NHIỆM PHÁP LÝ ĐỐI VỚI BẤT KỲ THIẾT HẠI HẬU QUẢ, ĐÙNG, BẤT CỨ, ĐẶC BIỆT HOẶC SỰ CÓ NÀO, BAO GỒM BẤT CỨ MẤT DỮ LIỆU HOẶC MẤT LỢI NHUẬN NÀO PHÁT SINH TỪ VIỆC BẠN SỬ DỤNG TÀI LIỆU.

Thêm vào**Tài nguyên**

- Hướng dẫn Tham khảo IP của Bộ xử lý:
http://www.xilinx.com/support/documentation/dt_edk.htm
- Chức năng XilFATFS

Các phần sau đây cung cấp tóm tắt về các chức năng XilFATFS và mô tả chức năng.

XilFATFS**Hàm số****Bản tóm tắt**

Phần này cung cấp danh sách các chức năng được cung cấp bởi XilFATFS. Sau đây là danh sách liên kết nơi bạn có thể nhấp vào tên hàm để chuyển đến phần mô tả.

```
void * sysace_fopen (const char * file, const char * mode)
int sysace_fread (void * buffer, int_size, int count, void * file)
int sysace_fwrite (void * buffer, int size, int count, void * file)
int sysace_fclose (void * tệp)
int sysace_mkdir (const char * path)
int sysace_chdir (const char * path)
int sysace_remove_dir (const char * path)
int sysace_remove_file (const char * path)
```

XilFATFS**Hàm số****Mô tả**

void * sysace_fopen (const char * file, const char * mode)

Thông số

tệp là tên của tệp trên thiết bị flash.

chế độ là "r" hoặc "w".

Lợi nhuận

Một xử lý tệp khác 0 khi thành công.

0 cho sự thất bại.

Sự mô tả

Tên tệp phải tuân theo quy ước đặt tên Microsoft 8.3 của một tên tệp gồm tám ký tự, theo sau là dấu '.' và một phần mở rộng bao gồm tám ký tự. Ví dụ: test.txt.

Hàm này trả về một xử lý tệp phải được sử dụng cho các lệnh gọi tiếp theo để đọc, ghi hoặc đóng tệp.

Nếu chế độ là "r" và tệp được đặt tên không tồn tại trên thiết bị, thì giá trị 0 sẽ được trả về.

Bao gồm

sysace_stdio.h

```
int sysace_fread (void * buffer, int_size, int count, void * file)
```

Thông số

buffer là bộ đệm được cấp phát trựớc được chuyển vào thủ tục này và được sử dụng để trả về các ký tự đã đọc từ thiết bị.

kích thước được giới hạn ở 1.

count là số ký tự được đọc.

tệp là trình xử lý tệp được trả về bởi sysace_fopen.

Lợi nhuận

Không có số ký tự thực sự được đọc để thành công.

0 cho sự thất bại.

Sự mô tả

Bộ đệm được phân bổ trựớc chứa đầy các ký tự được đọc từ thiết bị. Giá trị trả về cho biết số ký tự thực tế được đọc, trong khi số đếm chỉ định số ký tự tối đa để đọc.

Kích thước bộ đệm ít nhất phải được đếm. luồng phải là một trình xử lý tệp hợp lệ được trả về bởi một cuộc gọi đến sysace_fopen.

Bao gồm

sysace_stdio.h

```
int sysace_fwrite (void * buffer, int size, int count, void * file)
```

Thông số

buffer là bộ đệm được cấp phát trựớc được chuyển vào thủ tục này và chứa các ký tự được ghi vào thiết bị.

kích thước được giới hạn ở 1.

count là số ký tự được viết.

tệp là trình xử lý tệp được trả về bởi sysace_fopen.

Lợi nhuận

Không có số ký tự thực sự được viết để thành công.

0 hoặc -1 nếu không thành công.

Sự mô tả

Bộ đệm được cấp phát trựớc được lắp đầy (bởi người gọi) với các ký tự sẽ được ghi vào thiết bị. Giá trị trả về cho biết thực tế

số ký tự được viết, trong khi số đếm chỉ định số ký tự tối đa để viết. Kích thước bộ đệm ít nhất phải được đếm. luồng phải là một trình xử lý tệp hợp lệ được trả về bởi một cuộc gọi đến sysace_fopen. Chức năng này có thể chỉ trả lại sau khi cập nhật bộ đệm đệm (xem CONFIG_BUFCACHE_SIZE). Để đảm bảo rằng dữ liệu được ghi vào thiết bị, hãy thực hiện lệnh gọi sysace_fclose .

Bao gồm

sysace_stdio.h

int sysace_fclose (tệp void *)

Thông số	tệp: Xử lý tệp được trả về bởi sysace_fopen.
Lợi nhuận	0 về thành công. -1 về thất bại.
Sự mô tả	Đóng một tệp đang mở. Chức năng này cũng đồng bộ bộ đệm cache vào bộ nhớ. Nếu bất kỳ tệp nào được ghi bằng sysace_fwrite, thì cần phải đồng bộ hóa dữ liệu vào đĩa bằng cách thực hiện sysace_fclose. Nếu điều này không được thực hiện, thì đĩa có thể bị hỏng.
Bao gồm	sysace_stdio.h

int sysace_mkdir (const char * path)

Thông số	đường dẫn là tên đường dẫn của thư mục mới.
Lợi nhuận	0 về thành công. -1 về thất bại.
Sự mô tả	Tạo một thư mục mới được chỉ định bởi đường dẫn. Tên thư mục có thể là tuyệt đối hoặc tương đối và phải tuân theo quy ước đặt tên tệp 8.3.
	Ví dụ: a: \\ dirname, a: \\ dirname.dir, a: \\ dir1 \\ dirnew, dirname, dirname.dir
	Nếu một đường dẫn tương đối được chỉ định và thư mục làm việc hiện tại chưa được đặt, thì thư mục làm việc hiện tại sẽ mặc định là thư mục gốc.
Bao gồm	sysace_stdio.h

int sysace_chdir (const char * path)

Thông số	đường dẫn là tên đường dẫn của thư mục mới
Lợi nhuận	0 về thành công -1 khi thất bại
Sự mô tả	Tạo một thư mục mới được chỉ định bởi đường dẫn. Tên thư mục có thể là tuyệt đối hoặc tương đối và phải tuân theo quy ước đặt tên tệp 8.3.
	Ví dụ: a: \\ dirname, a: \\ dirname.dir, a: \\ dir1 \\ dirnew, dirname, dirname.dir
	Nếu một đường dẫn tương đối được chỉ định và thư mục làm việc hiện tại chưa được đặt, thì thư mục làm việc hiện tại sẽ mặc định là thư mục gốc.
Bao gồm	sysace_stdio.h

int sysace_remove_dir (const char * path)

Thông số path là đường dẫn đầy đủ đến thư mục phải được xóa.

Lợi nhuận 0 về thành công

Số nguyên âm khi bị lỗi.

Sự mô tả Xóa tệp hoặc thư mục được chỉ định bởi đường dẫn. Các chức năng này chỉ khả dụng khi tham số CONFIG_WRITE thành XilFATFS được đặt.

Bao gồm sysace_stdio.h

int sysace_remove_file (const char * path)

Thông số đường dẫn là đường dẫn đầy đủ đến tệp phải được xóa.

Lợi nhuận 0 về thành công.

Số nguyên âm khi bị lỗi.

Sự mô tả Xóa tệp hoặc thư mục được chỉ định bởi đường dẫn. Các chức năng này chỉ khả dụng khi tham số CONFIG_WRITE thành XilFATFS được đặt.

Bao gồm sysace_stdio.h

Hệ thống tệp XilFATFS có thể được tích hợp với hệ thống bằng cách sử dụng đoạn mã sau trong Tệp thông số kỹ thuật phần mềm vi xử lý (MSS):

```
BẮT ĐẦU THƯ VIỆN
tham số LIBRARY_NAME = xilfatfs
tham số LIBRARY_VER = 1.00.a
tham số CONFIG_WRITE = true
tham số CONFIG_DIR_SUPPORT = false
tham số CONFIG_FAT12 = false
tham số CONFIG_MAXFILES = 5
tham số CONFIG_BUFCACHE_SIZE = 10240
tham số PROC_INSTANCE = powerpc_0

THƯ VIỆN KẾT THÚC
```

Mô tả về Thông Số:

- Khi CONFIG_WRITE được đặt thành true, khả năng ghi sẽ được thêm vào thư viện.
- Khi CONFIG_DIR_SUPPORT được đặt thành true, các hàm mkdir và chdir được thêm vào thư viện. Để hàm mkdir () hoạt động, CONFIG_WRITE cần được bật.
- Khi CONFIG_FAT12 được đặt thành true, thư viện được cấu hình để hoạt động với hệ thống tệp FAT12. Nếu không, thư viện hoạt động với cả hệ thống tệp FAT16 và FAT32.
- CONFIG_MAXFILES giới hạn số lượng tệp tối đa có thể mở. Điều này ảnh hưởng đến lượng bộ nhớ được phân bổ tĩnh bởi XilFATFS.
- CONFIG_BUFCACHE_SIZE: xác định dung lượng bộ nhớ (tính bằng byte) được thư viện sử dụng để đệm các lệnh đọc và ghi tới Hệ thống ACE. Điều này cải thiện hiệu suất của cả sysace_fread và sysace_fwrite bằng cách đệm dữ liệu trong bộ nhớ và tránh các lệnh gọi không cần thiết để đọc thiết bị CF. Bộ đệm chỉ được đồng bộ hóa với thiết bị khi gọi sysace_fclose; do đó, điều cần thiết là phải thực hiện sysace_fclose nếu bất kỳ tệp nào bị sửa đổi.
- Tham số PROC_INST không cần thiết cho các hệ thống đơn xử lý. Trong một hệ thống đa xử lý, đặt PROC_INST thành tên bộ xử lý mà thư viện phải được biên dịch. Thiết bị ngoại vi System ACE phải có thể kết nối được từ bộ xử lý này.



Hệ thống tệp bộ nhớ LibXil (MFS)

UG 649 ngày 15 tháng 4 năm 2009

Bản tóm tắt

Tài liệu này mô tả Hệ thống tệp bộ nhớ Xilinx® (MFS). Hệ thống tệp này nằm trong bộ nhớ RAM / ROM / Flash và có thể được truy cập trực tiếp hoặc thông qua các cuộc gọi MFS. MFS được tích hợp với một hệ thống sử dụng Trình tạo thư viện, Libgen. Tài liệu bao gồm các phần sau:

- “[Tổng quan](#)”
- “[Chức năng MFS](#)”
- “[Chức năng Tiện ích](#)”
- “[Tiện ích bổ sung](#)”
- “[Tùy chỉnh Libgen](#)”

Tổng quan

LibXil MFS cung cấp khả năng quản lý bộ nhớ chưƠng trình dưới dạng các tệp xử lý. Bạn có thể tạo thư mục và có tệp trong mỗi thư mục. Hệ thống tệp có thể được truy cập từ ngôn ngữ C cấp cao thông qua các lệnh gọi hàm dành riêng cho hệ thống tệp.

Các chức năng MFS Phần này cung cấp một bản tóm tắt được liên kết và các mô tả về các chức năng MFS.

© 2009 Xilinx, Inc. XILINX, biểu trưng Xilinx, Virtex, Spartan, ISE và các nhãn hiệu được chỉ định khác có trong tài liệu này là nhãn hiệu của Xilinx tại Hoa Kỳ và các quốc gia khác. Tất cả các nhãn hiệu khác là tài sản của chủ sở hữu tương ứng của họ.

Xilinx sẽ tiết lộ hướng dẫn sử dụng, hướng dẫn sử dụng, ghi chú phát hành và / hoặc thông số kỹ thuật này ("Tài liệu") cho bạn chỉ để sử dụng trong việc phát triển các thiết kế để vận hành với các thiết bị phần cứng của Xilinx. Bạn không được phép sao chép, phân phối, tái xuất bản, tái xuất, hiển thị, đăng hoặc truyền Tài liệu dưới bất kỳ hình thức nào hoặc bất kỳ phương tiện nào bao gồm nhưng không giới hạn ở Xilinx. Xilinx từ chối rõ ràng bất kỳ trách nhiệm pháp lý nào phát sinh từ việc bạn sử dụng Tài liệu. Xilinx bảo lưu quyền, theo quyết định riêng của mình, thay đổi Tài liệu mà không cần thông báo bất kỳ lúc nào. Xilinx không có nghĩa vụ phải sửa bất kỳ lỗi nào có trong Tài liệu hoặc thông báo cho bạn về bất kỳ chỉnh sửa hoặc cập nhật nào. Xilinx từ chối rõ ràng bất kỳ trách nhiệm pháp lý nào liên quan đến hỗ trợ kỹ thuật hoặc hỗ trợ có thể được cung cấp cho bạn liên quan đến Thông tin.

TÀI LIỆU ĐƯỢC CÔNG BỐ CHO BẠN "NGUYÊN TRẠNG" KHÔNG ĐƯỢC BẢO HÀNH BẤT KỲ HÌNH THỨC NÀO. XILINX KHÔNG CÓ BẢO ĐÁM KHẮC, DÙ THỂ HIỆN, NGƯ Ý HOẶC TRUYỀN NGHĨA, LIÊN QUAN ĐẾN TÀI LIỆU, BAO GỒM BẤT KỲ BẢO ĐÁM NÀO VỀ TÍNH KHẢ NĂNG, PHÙ HỢP VỚI MỤC ĐÍCH CỤ THỂ HOẶC KHÔNG ĐÁM BẢO QUYỀN CỦA BÊN THỨ BA. TRONG MỌI TRƯỜNG HỢP XILINX SẼ CHỊU TRÁCH NHIỆM PHÁP LÝ ĐỐI VỚI BẤT KỲ THIẾT HẠI HẬU QUẢ, ĐÙNG, BẤT CỨ, ĐẶC BIỆT HOẶC SỰ CỐ NÀO, BAO GỒM BẤT CỨ MẤT DỮ LIỆU HOẶC MẤT LỢI NHUẬN NÀO PHÁT SINH TỪ VIỆC BẠN SỬ DỤNG TÀI LIỆU.

Tóm tắt chức năng MFS

Danh sách sau đây là bản tóm tắt đư ợc liên kết của các chức năng MFS đư ợc hỗ trợ. Mô tả các chức năng đư ợc cung cấp sau bảng tóm tắt. Bạn có thể nhấp vào một chức năng trong danh sách tóm tắt để chuyển đến phần mô tả.

```
void mfs_init_fs (int numbytes, _char_ * address, _int init_type)
void mfs_init_genimage (int numbytes, char * address, int init_type)
int mfs_change_dir (char_ * newdir)
int mfs_create_dir (char * newdir)
int mfs_delete_dir (char * dirname)
int mfs_get_current_dir_name (char * dirname)
int mfs_delete_file (char * tên tệp)
int mfs_rename_file (char * from_file, char * to_file)
int mfs_exists_file (char * tên tệp)
int mfs_get_usage (int * num_blocks_used, int * num_blocks_free)
int mfs_dir_open (char * dirname)
int mfs_dir_close (int fd)
int mfs_dir_read (int fd, char _ ** filename, int * filesize, int * filetype)
int mfs_file_open (char * tên tệp, chế độ int)
int mfs_file_read (int fd, char * buf, int buflen)
int mfs_file_write (int fd, char * buf, int buflen)
int mfs_file_close (int fd)
long mfs_file_lseek (int fd, long offset, int whence)
```

Mô tả chức năng MFS

```
void mfs_init_fs (int numbytes, _char_ * address, _int
init_type)
```

Tham số numbyte là số byte bộ nhớ có sẵn cho hệ thống tệp.

địa chỉ là địa chỉ bắt đầu (cơ sở) của bộ nhớ hệ thống tệp.

init_type là MFSINIT_NEW, MFSINIT_IMAGE hoặc MFSINIT_ROM_IMAGE.

Sự mô tả

Khởi tạo hệ thống tệp bộ nhớ. Hàm này phải đư ợc gọi trước bất kỳ hoạt động nào của hệ thống tệp. Sử dụng mfs_init_genimage thay vì hàm này nếu hệ thống tệp đang đư ợc khởi tạo bằng hình ảnh do mfsgen tạo ra. Tham số trạng thái / chế độ xác định các thuộc tính hệ thống tệp nhất định:

- MFSINIT_NEW tạo một hệ thống tệp trống mới để đọc / ghi.
- MFSINIT_IMAGE khởi tạo hệ thống tệp có dữ liệu trước đó đư ợc tải vào bộ nhớ tại địa chỉ cơ sở.
- MFSINIT_ROM_IMAGE khởi tạo hệ thống tệp Chỉ đọc có dữ liệu trước đó đã đư ợc tải vào bộ nhớ tại địa chỉ cơ sở.

Bao gồm

xilmfs.h

```
void mfs_init_genimage (int numbytes, char * address, int
init_type)
```

Thông số

numbyte là số byte bộ nhớ trong hình ảnh do công cụ mfsgen tạo ra. Con số này bằng với dung lư ợng bộ nhớ có sẵn cho hệ thống tệp, cộng với 4.

địa chỉ là địa chỉ bắt đầu (cơ sở) của hình ảnh.

init_type là MFSINIT_IMAGE hoặc MFSINIT_ROM_IMAGE

Sự mô tả

Khởi tạo hệ thống tệp bộ nhớ bằng một hình ảnh đã được tải trước đó vào bộ nhớ tại địa chỉ cơ sở.

Hàm này phải được gọi trước bất kỳ hoạt động nào của hệ thống tệp. Tham số trạng thái / chế độ xác định các thuộc tính hệ thống tệp nhất định:

- MFSINIT_IMAGE khởi tạo hệ thống tệp tin có dữ liệu đã được tải trước đó vào bộ nhớ tại địa chỉ cơ sở.
- MFSINIT_ROM_IMAGE khởi tạo hệ thống tệp Chỉ đọc có dữ liệu đã được tải trước đó vào bộ nhớ tại địa chỉ cơ sở.

Bao gồm

xilmfs.h

```
int mfs_change_dir (char * newdir)
```

Thông số

newdir là đích chdir.

Lợi nhuận

1 về thành công.

0 khi thất bại.

Sự mô tả

Nếu newdir tồn tại, hãy đặt nó thành thư mục hiện tại của MFS. Thư mục hiện tại không được sửa đổi trong trường hợp bị lỗi.

Bao gồm

xilmfs.h

```
int mfs_create_dir (char * newdir)
```

Thông số

newdir là tên thư mục sẽ được tạo.

Lợi nhuận

Chỉ mục của thư mục mới trong hệ thống tệp về thành công.

0 khi thất bại.

Sự mô tả

Tạo một thư mục trống mới có tên là newdir bên trong thư mục hiện tại.

Bao gồm

xilmfs.h

```
int mfs_delete_dir (char * dirname)
```

Thông số

dirname là thư mục sẽ bị xóa.

Lợi nhuận

Chỉ mục của thư mục mới trong hệ thống tệp về thành công.

0 khi thất bại.

Sự mô tả

Xóa tên thư mục, nếu nó tồn tại và trống.

Bao gồm

xilmfs.h

int mfs_get_current_dir_name (char * dirname)

Thông số	dirname là tên thư mục hiện tại.
Lợi nhuận	1 về thành công. 0 khi thất bại.
Sự mô tả	Trả lại tên của thư mục hiện tại trong bộ đệm được phân bổ trước, tên dirname, gồm ít nhất 16 ký tự. Nó không trả về tên đường dẫn tuyệt đối của thư mục hiện tại mà chỉ là tên của thư mục hiện tại.
Bao gồm	xilmfs.h

int mfs_delete_file (char * tên tệp)

Thông số	tên tệp là tệp sẽ bị xóa.
Lợi nhuận	1 về thành công. 0 khi thất bại.
Sự mô tả	Xóa tên tệp khỏi thư mục.
Bao gồm	xilmfs.h
Thận trọng! Chức năng này không hoàn toàn giải phóng không gian thư mục được sử dụng bởi tệp. Các lệnh gọi tạo và xóa tệp lặp đi lặp lại có thể khiến hệ thống tệp hết dung lượng.	

int mfs_rename_file (char * from_file, char * to_file)

Thông số	from_file là tên tệp gốc. to_file là tên tệp mới.
Lợi nhuận	1 về thành công. 0 khi thất bại.
Sự mô tả	Đổi tên from_file thành to_file. Đổi tên hoạt động cho các thư mục cũng như các tệp. Hành không thành công nếu to_file đã tồn tại.
Bao gồm	xilmfs.h

int mfs_exists_file (char * tên tệp)

Thông số	tên tệp là tệp hoặc thư mục được kiểm tra sự tồn tại.
Lợi nhuận	0 nếu tên tệp không tồn tại. 1 nếu tên tệp là một tệp. 2 nếu tên tệp là một thư mục.
Sự mô tả	Kiểm tra xem tệp / thư mục có trong thư mục hiện tại hay không.
Bao gồm	xilmfs.h

```
int mfs_get_usage (int * num_blocks_used, int
* num_blocks_free)
```

Thông số

`num_blocks_used` là số khối được sử dụng.
`num_blocks_free` là số khối miễn phí.

Lợi nhuận

1 về thành công.
0 khi thất bại.

Sự mô tả

Nhận số khối đã sử dụng và số khối miễn phí trong hệ thống tệp thông qua con trỏ.

Bao gồm

xilmfs.h

```
int mfs_dir_open (char * dirname)
```

Thông số

`dirname` là thư mục được mở để đọc.

Lợi nhuận

Chỉ mục của tên `dirname` trong mảng các tệp mở khi thành công.
-1 về thất bại.

Sự mô tả

Mở tên thư mục để đọc. Việc đọc một thư mục được thực hiện bằng `mfs_dir_read()`.

Bao gồm

xilmfs.h

```
int mfs_dir_close (int fd)
```

Thông số

`fd` là bộ mô tả tệp trả về bằng cách mở.

Lợi nhuận

1 về thành công.
0 khi thất bại.

Sự mô tả

Đóng dir được trả bởi `fd`. Hệ thống tệp lấy lại `fd` và sử dụng nó cho các tệp mới.

Bao gồm

xilmfs.h

```
int mfs_dir_read (int fd, char ** tên tệp,
                  int * filesize, int * filetype)
```

Thông số

fd là bộ mô tả tệp trả về bằng cách mở; được chuyển đến chức năng này bởi người gọi.

tên tệp là con trỏ đến tên tệp ở vị trí hiện tại trong thư mục trong MFS; giá trị này được điều bởi hàm này.

filesize là con trỏ đến một giá trị được điều bởi hàm này: Kích thước tính bằng byte của tên tệp, nếu đó là tệp thông thường; Số mục nhập thư mục nếu tên tệp là một thư mục.

filetype là con trỏ đến một giá trị được điều bởi hàm này:
MFS_BLOCK_TYPE_FILE nếu tên tệp là một tệp thông thường;
MFS_BLOCK_TYPE_DIR nếu tên tệp là một thư mục.

Lợi nhuận

1 về thành công.

0 khi thất bại.

Sự mô tả

Đọc mục nhập thư mục hiện tại và chuyển con trỏ nội bộ đến mục nhập thư mục tiếp theo. tên tệp, loại tệp và kích thước tệp là các con trỏ đến các giá trị được lưu trữ trong mục nhập thư mục hiện tại.

Bao gồm

xilmfs.h

```
int mfs_file_open (char * tên tệp, chế độ int)
```

Thông số

tên tệp là tệp sẽ được mở.

chế độ là Đọc / Viết hoặc Tạo.

Lợi nhuận

Chỉ mục của tên tệp trong mảng tệp mở khi thành công.

-1 về thất bại.

Sự mô tả

Mở tên tệp với chế độ nhất định. Hàm nên được sử dụng cho tệp chứ không phải thư mục:

- **MODE_READ**, không có kiểm tra lỗi nào được thực hiện (nếu tệp hoặc thư mục).
- **MODE_CREATE** tạo một tệp chứ không phải một thư mục. •
- MODE_WRITE** không thành công nếu tệp được chỉ định là DIR.

Bao gồm

xilmfs.h

```
int mfs_file_read (int fd, char * buf, int buflen)
```

Thông số

fd là bộ mô tả tệp trả về khi mở.

buf là bộ đệm đích cho việc đọc.

buflen là chiều dài của bộ đệm.

Lợi nhuận

Số byte được đọc khi thành công.

0 khi thất bại.

Sự mô tả

Đọc byte số buflen và đặt nó trong buf. fd phải là một chỉ mục hợp lệ trong mảng "tệp đang mở", trả đến một tệp chứ không phải một thư mục. buf phải là bộ đệm được phân bổ trước có kích thước buflen trả lên. Nếu có ít hơn các ký tự buflen thì chỉ có nhiều ký tự đó được đọc.

Bao gồm

xilmfs.h

```
int mfs_file_write (int fd, char * buf, int buflen)
```

Thông số

fd là bộ mô tả tệp trả về khi mở.

buf là bộ đệm nguồn từ nơi dữ liệu được đọc.

buflen là chiều dài của bộ đệm.

Lợi nhuận

1 về thành công.

0 khi thất bại.

Sự mô tả

Ghi số byte buflen từ buf vào tệp. fd phải là một chỉ mục hợp lệ trong mảng open_files. buf phải là bộ đệm được cấp phát trước có kích thước từ buflen trở lên.

Thận trọng! Không hỗ trợ ghi vào các vị trí không phải là cuối tệp.

Sử dụng mfs_file_lseek () đi đến một số vị trí khác trong tệp sau đó gọi mfs_file_write () không được hỗ trợ

Bao gồm

xilmfs.h

```
int mfs_file_close (int fd)
```

Thông số

fd là bộ mô tả tệp trả về khi mở.

Lợi nhuận

1 về thành công.

0 khi thất bại.

Sự mô tả

Đóng tệp được trả bởi fd. Hệ thống tệp lấy lại fd và sử dụng nó cho các tệp mới.

Bao gồm

xilmfs.h

```
long mfs_file_lseek (int fd, long offset, int whence)
```

Thông số

fd là bộ mô tả tệp trả về khi mở.

offset là số byte cần tìm.

khi nào là chế độ phụ thuộc vào hệ thống tệp:

- MFS_SEEK_END, thì bù đắp có thể là 0 hoặc âm, nếu không bù là không âm.
- MFS_SEEK_CURR, sau đó bù đắp được tính từ vị trí hiện tại.
- MFS_SEEK_SET, thì bù đắp được tính từ đầu tệp.

Lợi nhuận

Trả về offset từ đầu tệp đến vị trí hiện tại trên thành công.

-1 khi thất bại: vị trí hiện tại không được sửa đổi.

Sự mô tả

Tìm kiếm một độ lệch đã cho trong tệp tại vị trí fd trong mảng open_files.

Thận trọng! Đó là một lỗi khi tìm kiếm trước khi bắt đầu tệp hoặc sau khi kết thúc tệp.

Thận trọng! Không hỗ trợ ghi vào các vị trí không phải là cuối tệp. Sử dụng hàm mfs_file_lseek () hoặc đi đến một số vị trí khác trong tệp sau đó gọi mfs_file_write () không được hỗ trợ.

Bao gồm

xilmfs.h

Tính thiết thực

Chức năng

Các phần phụ sau đây cung cấp bản tóm tắt và mô tả về các chức năng tiện ích có thể được sử dụng cùng với MFS. Các hàm này được định nghĩa trong `mfs_filesys_util.c` và được khai báo trong `xilmfs.h`.

Tóm tắt chức năng tiện ích

Danh sách sau đây là bản tóm tắt được liên kết của các chức năng MFS Utility được hỗ trợ. Mô tả các chức năng được cung cấp sau bằng tóm tắt. Bạn có thể nhấp vào một chức năng trong danh sách tóm tắt để chuyển đến phần mô tả.

```
int mfs_ls (void)
int mfs_ls_r (int đệ quy)
int mfs_cat (char * tên tệp)
int mfs_copy_stdin_to_file (char * tên tệp)
int mfs_file_copy (char * from_file, char * to_file)
```

Mô tả chức năng tiện ích

`int mfs_ls (void)`

Thông số	Không có.
Lợi nhuận	1 về thành công. 0 khi thất bại.
Sự mô tả	Liệt kê nội dung của thư mục hiện tại trên STDOUT.
Bao gồm	<code>xilmfs.h</code>

`int mfs_ls_r (int đệ quy)`

Các tham số đệ quy kiểm soát số lượng đệ quy:

- 0 liệt kê nội dung của thư mục hiện tại và dừng lại.
- > 0 liệt kê nội dung của thư mục hiện tại và bắt kỳ thư mục con nào cho đến độ sâu của đệ quy.
- = -1 hoàn thành danh sách thư mục đệ quy không có giới hạn về độ sâu đệ quy.

Lợi nhuận	1 về thành công. 0 khi thất bại.
-----------	-------------------------------------

Sự mô tả	Liệt kê nội dung của thư mục hiện tại trên STDOUT.
----------	--

Bao gồm	<code>xilmfs.h</code>
---------	-----------------------

```
int mfs_cat (char * tên tệp)

Thông số          tên tệp là tệp sẽ được hiển thị.

Lợi nhuận        1 về thành công.
                  0 khi thất bại.

Sự mô tả        In tệp thành STDOUT.

Bao gồm          xilmfs.h
```

```
int mfs_copy_stdin_to_file (char * tên tệp)

Tên tệp tham số là tệp đích.

Lợi nhuận        1 về thành công.
                  0 khi thất bại.

Sự mô tả        Sao chép từ STDIN sang tệp đã đặt tên. Một ký tự cuối tệp (EOF) phải được gửi từ STDIN để cho phép
                  hàm trả về 1.

Bao gồm          xilmfs.h
```

```
int mfs_file_copy (char * from_file, char * to_file)

Tham số from_file là tệp nguồn.
                  to_file là tệp đích.

Lợi nhuận        1 về thành công.
                  0 khi thất bại.

Mô tả Sao chép from_file sang to_file. Sao chép không thành công nếu to_file đã tồn tại hoặc từ hoặc đến vị trí không
                  thẻ mở được.

Bao gồm          xilmfs.h
```

Thêm vào Tiện ích

Chương trình mfsgen được cung cấp cùng với thư viện MFS. Bạn có thể sử dụng mfsgen để tạo hình ảnh bộ nhớ MFS trên hệ thống máy chủ mà sau đó có thể tải xuống bộ nhớ hệ thống nhúng. Mfsgen liên kết đến LibXil MFS và được biên dịch để chạy trên máy chủ chứ không phải hệ thống vi xử lý MicroBlaze™ hoặc PowerPC® đích. Về mặt khái niệm, điều này tương tự như các chương trình zip hoặc tar quen thuộc.

Toàn bộ hệ thống phân cấp thư mục trên hệ thống máy chủ có thể được sao chép sang ảnh tệp MFS cục bộ bằng cách sử dụng mfsgen. Sau đó, hình ảnh tệp này có thể được tải xuống bộ nhớ của hệ thống nhúng để tạo hệ thống tệp được tải trước.

Các chương trình thử nghiệm được đưa vào để minh họa quá trình này. Để biết thêm thông tin, hãy xem tệp readme.txt trong thư mục con utils.

Cách sử dụng: mfsgen - {c filelist | t | x} vsb num_blocks f mfs_filename

c: chỉ định chính xác một trong các chế độ c, t hoặc x

t: tạo một hình ảnh hệ thống tệp mfs bằng cách sử dụng danh sách các tệp được chỉ định trên dòng lệnh (các thư mục được chỉ định trong danh sách này được duyệt đệ quy).

t: liệt kê các tệp trong hình ảnh hệ thống tệp mfs

x: trích xuất hệ thống tệp mfs từ hình ảnh sang hệ thống tệp máy chủ

v: là chế độ tiết

s: chuyển đổi endianness

b: liệt kê số lượng khối (num_blocks) phải nhiều hơn 2

- Nếu tùy chọn b được chỉ định, giá trị num_blocks phải được chỉ định

- Nếu tùy chọn b bị bỏ qua, giá trị mặc định của num_blocks là 5000

- Tùy chọn b chỉ có ý nghĩa khi được sử dụng cùng với tùy chọn c

f: chỉ định tên tệp máy chủ (mfs_filename) nơi lưu trữ hình ảnh hệ thống tệp mfs

- Nếu tùy chọn f được chỉ định, tên tệp mfs phải được chỉ định

- Nếu tùy chọn f bị bỏ qua, tên tệp mặc định là filesystem.mfs

Hệ thống tệp bộ nhớ có thể được tích hợp với hệ thống bằng đoạn mã sau trong tệp Thông số kỹ thuật phần mềm vi xử lý (MSS).

```
BẮT ĐẦU THƯ VIỆN
tham số LIBRARY_NAME = xilmfs
tham số LIBRARY_VER = 1.00.a
tham số numbytes = 50000
tham số base_address = 0xffe00000
tham số init_type = MFSINIT_NEW
tham số need_utils = false
CHẤM Dứt
```

Hệ thống tệp bộ nhớ phải được khởi tạo với tên xilmfs. Bảng sau liệt kê các thuộc tính được Libgen sử dụng.

Bảng 1: Các thuộc tính để bao gồm hệ thống tệp bộ nhớ

Thuộc tính	Sự mô tả
numbyte	Số byte được phân bổ cho hệ thống tệp.
base_address	Địa chỉ bắt đầu cho bộ nhớ hệ thống tệp.
init_type	Các tùy chọn là: <ul style="list-style-type: none"> MFSINIT_NEW (mặc định) tạo một hệ thống tệp mới, trống. MFSINIT_ROM_IMAGE tạo hệ thống tệp dựa trên hình ảnh bộ nhớ được tải trước được tải trong bộ nhớ có kích thước số byte tại địa chỉ bắt đầu base_address. Bộ nhớ này được coi là chỉ đọc và không được phép sửa đổi hệ thống tệp. <ul style="list-style-type: none"> MFS_INIT_IMAGE tương tự như tùy chọn tru桔 đó ngoại trừ việc hệ thống tệp có thể được sửa đổi và bộ nhớ có thể đọc và ghi được.
need_utils	true hoặc false (default = false) Nếu đúng, điều này khiến stdio.h được bao gồm từ mfs_config.h. Các chức năng được mô tả trong "Chức năng Tiện ích", trang 8 yêu cầu bạn phải xác định stdin hoặc stdout. Việc đặt need_utils thành true sẽ dẫn đến việc bao gồm stdio.h. Thận trọng! Nền tảng phần mềm và phần cứng bên dưới phải hỗ trợ các thiết bị ngoại vi stdin và stdout để các chức năng tiện ích này có thể biên dịch và liên kết một cách chính xác.



Thư viện lwIP 1.3.0 (v1.00.a)

UG 650 ngày 15 tháng 4 năm 2009

Bản tóm tắt

Tài liệu này mô tả công Bộ phát triển nhúng (EDK) của ngăn xếp TCP / IP trọng lư ợng nhẹ (lwIP) mã nguồn mở. LwIP cung cấp một cách dễ dàng để thêm khả năng kết nối mạng dựa trên TCP / IP vào hệ thống nhúng.

Lwip130_v1_00_a cung cấp bộ điều hợp cho xps_etherenetlite và xps_ll_temac

Các lõi MAC Xilinx® Ethernet và dựa trên ngăn xếp lwIP phiên bản 1.3.0. Tài liệu này mô tả cách sử dụng lwip130_v1_00_a để thêm khả năng kết nối mạng vào phần mềm nhúng.

Nó bao gồm các phần sau:

- “[Tổng quan](#)”
- “[Tính năng](#)”
- “[Tài nguyên bổ sung](#)”
- “[Sử dụng lwIP](#)”
- “[Thiết lập Hệ thống Phần cứng](#)”
- “[Thiết lập Hệ thống Phần mềm](#)”
- “[Hiệu suất lwIP](#)”
- “[Các vấn đề và hạn chế đã biết](#)”
- “[Di chuyển từ lwip_v3_00_a sang lwip130_v1_00_a](#)”
- “[Ví dụ về API](#)”

Tổng quan

LwIP là một bộ giao thức TCP / IP mã nguồn mở có sẵn theo giấy phép BSD. LwIP là một ngăn xếp độc lập; không có phụ thuộc hệ điều hành, mặc dù nó có thể được sử dụng cùng với hệ điều hành. LwIP cung cấp hai API để các ứng dụng sử dụng:

- API RAW: Cung cấp quyền truy cập vào ngăn xếp lwIP lõi.
- API Socket: Cung cấp giao diện kiểu ồ cǎm BSD cho ngăn xếp.

Lwip130_v1_00_a là thư viện EDK được xây dựng trên thư viện lwIP mã nguồn mở phiên bản 1.3.0. Thư viện lwip130_v1_00_a cung cấp bộ điều hợp cho các lõi Ethernetlite (xps_etherenetlite) và TEMAC (xps_ll_temac) Xilinx EMAC. Thư viện có thể chạy trên bộ vi xử lý MicroBlaze™, PowerPC® 405 hoặc PowerPC 440.

© 2009 Xilinx, Inc. XILINX, biểu trưng Xilinx, Virtex, Spartan, ISE và các nhãn hiệu được chỉ định khác có trong tài liệu này là nhãn hiệu của Xilinx tại Hoa Kỳ và các quốc gia khác. Tất cả các nhãn hiệu khác là tài sản của chủ sở hữu tương ứng của họ.

Xilinx sẽ tiết lộ hứa hẹn dẫn sử dụng, hứa hẹn dẫn sử dụng, ghi chú phát hành và / hoặc thông số kỹ thuật này ("Tài liệu") cho bạn chỉ để sử dụng trong việc phát triển các thiết kế để vận hành với các thiết bị phần cứng của Xilinx. Bạn không được phép sao chép, phân phối, tái xuất bản, tái xuất bản, hiển thị, đăng hoặc truyền Tài liệu dưới bất kỳ hình thức nào hoặc bằng bất kỳ phương tiện nào bao gồm như ng không giới hạn ở Xilinx. Xilinx từ chối rõ ràng bất kỳ trách nhiệm pháp lý nào phát sinh từ việc bạn sử dụng Tài liệu. Xilinx bảo lưu quyền, theo quyết định riêng của mình, thay đổi Tài liệu mà không cần thông báo bất kỳ lúc nào. Xilinx không có nghĩa vụ phải sửa bất kỳ lỗi nào có trong Tài liệu hoặc thông báo cho bạn về bất kỳ chỉnh sửa hoặc cập nhật nào. Xilinx từ chối rõ ràng bất kỳ trách nhiệm pháp lý nào liên quan đến hỗ trợ kỹ thuật hoặc hỗ trợ có thể được cung cấp cho bạn liên quan đến Thông tin.

TÀI LIỆU ĐƯỢC CÔNG BỐ CHO BẠN "NGUYÊN TRẠNG" KHÔNG ĐƯỢC BẢO HÀNH BẤT KỲ HÌNH THỨC NÀO. XILINX KHÔNG CÓ BẢO ĐÁM KHÁC, DÙ THỂ HIỆN, NGƯ Ý HOẶC TRUYỀN NGHĨA, LIÊN QUAN ĐẾN TÀI LIỆU, BAO GỒM BẤT KỲ BẢO ĐÁM NÀO VỀ TÍNH KHẢ NĂNG, PHÙ HỢP VỚI MỤC ĐÍCH CỤ THỂ HOẶC KHÔNG ĐÁM BẢO QUYỀN CỦA BÊN THỨ BA. TRONG MỌI TRƯỜNG HỢP XILINX SẼ CHỊU TRÁCH NHIỆM PHÁP LÝ ĐỐI VỚI BẤT KỲ THIẾT HẠI HẬU QUẢ, DÙNG, BẤT CỨ, ĐẶC BIỆT HOẶC SỰ CÓ NÀO, BAO GỒM BẤT CỨ MẤT DỮ LIỆU HOẶC MẤT LỢI NHUẬN NÀO PHÁT SINH TỪ VIỆC BẠN SỬ DỤNG TÀI LIỆU.

Đặc trưng

LwIP cung cấp hỗ trợ cho các giao thức sau:

- Giao thức Internet (IP)
- Giao thức tin nhắn điều khiển Internet (ICMP)
- Giao thức sơ đồ ngưỜI DÙNG (UDP)
- TCP (Giao thức điều khiển truyền (TCP))
- Giao thức phân giải địa chỉ (ARP)
- Giao thức cấu hình máy chủ động (DHCP)

Thêm vào Tài nguyên

- lwIP wiki: <http://lwip.scribblewiki.com>
- Các mẫu thiết kế và ứng dụng Xilinx lwIP: http://www.xilinx.com/support/documentation/application_notes/xapp1026.pdf
- Ví dụ về lwIP sử dụng RAW và API Socket: <http://savannah.nongnu.org/projects/lwip/>
- Bảng dữ liệu Bộ điều khiển bộ nhớ nhiều cổng (MPMC): có sẵn trong thư mục cài đặt phần mềm sau: EDK \ hw \ XilinxProcessorIPLib \ pcores \ mpmc_v * _00_a

Sử dụng lwIP

Các phần sau trình bày chi tiết các bước phần cứng và phần mềm để sử dụng lwIP cho mạng trong hệ thống phân phối EDK. Các bước quan trọng là:

1. Tạo một hệ thống phần cứng chứa bộ xử lý, lõi ethernet và bộ đếm thời gian. Bộ hẹn giờ và ngắt ethernet phải được kết nối với bộ xử lý bằng bộ điều khiển ngắn.
2. Định cấu hình lwip130_v1_00_a để trở thành một phần của nền tảng phần mềm. Đổi với API socket lwIP, thư viện Xilkernel là điều kiện tiên quyết.

Thiết lập Phần cứng

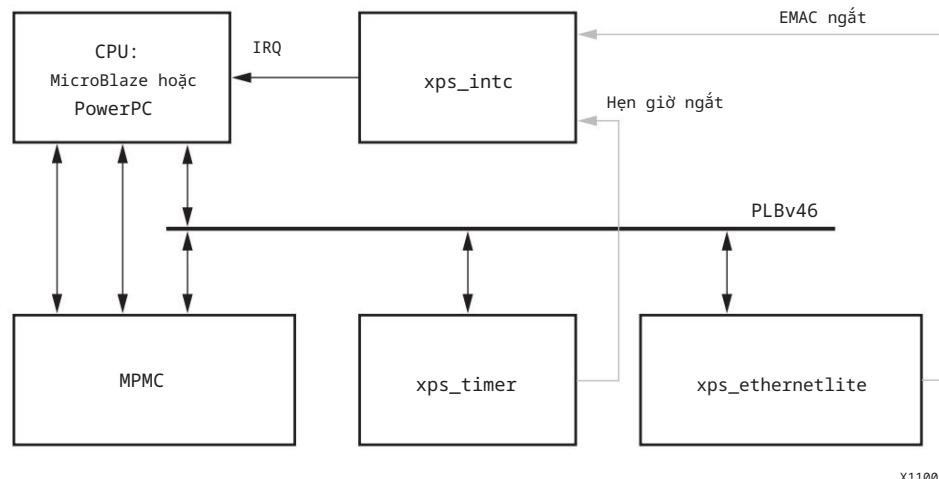
Hệ thống

Phần này mô tả các cấu hình phần cứng được hỗ trợ bởi lwIP. Các thành phần chính của hệ thống phần cứng bao gồm:

- Bộ xử lý: PowerPC (bộ xử lý 405 hoặc 440) hoặc bộ xử lý MicroBlaze.
- EMAC: lwIP hỗ trợ các lõi xps_etherenetlite và xps_ll_temac EMAC
- Bộ định thời: để duy trì bộ định thời TCP, lwIP yêu cầu một số chức năng nhất định được gọi theo các khoảng thời gian định kỳ bởi ứng dụng. Một ứng dụng có thể làm điều này bằng cách đăng ký một trình xử lý ngắn với một bộ đếm thời gian.
- DMA: lõi xps_ll_temac có thể được định cấu hình bằng Bộ nhớ trực tiếp mềm tùy chọn Công cụ truy cập (SDMA)

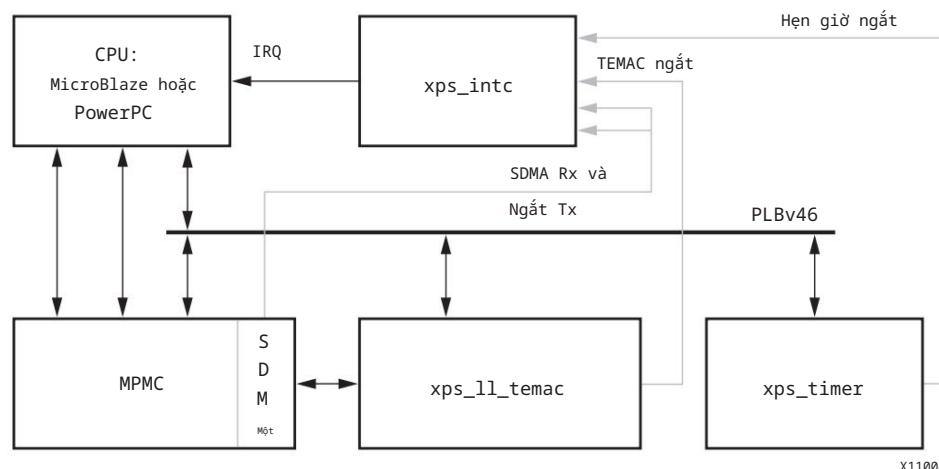
Hình dưới đây cho thấy một kiến trúc hệ thống trong đó hệ thống đang sử dụng một lõi xps_etherenetlite.

Hệ thống có bộ xử lý được kết nối với Bộ điều khiển bộ nhớ nhiều cổng (MPMC) với các thiết bị ngoại vi cần thiết khác (bộ đếm thời gian và etherenetlite) trên xe buýt PLB v4.6. Các ngắt từ cả bộ định thời và etherenetlite là bắt buộc, vì vậy các ngắt được kết nối với bộ điều khiển ngắn.



Hình 1: Kiến trúc hệ thống sử dụng xps_etherenetlite Core

Khi sử dụng TEMAC, kiến trúc hệ thống thay đổi tùy thuộc vào việc có yêu cầu DMA hay không. Nếu DMA là bắt buộc, một cổng thứ tư (thuộc loại SDMA), cung cấp kết nối trực tiếp giữa TEMAC (xps_ll_temac) và bộ điều khiển bộ nhớ (MPMC), sẽ được thêm vào bộ điều khiển bộ nhớ. Hình sau cho thấy kiến trúc hệ thống này.

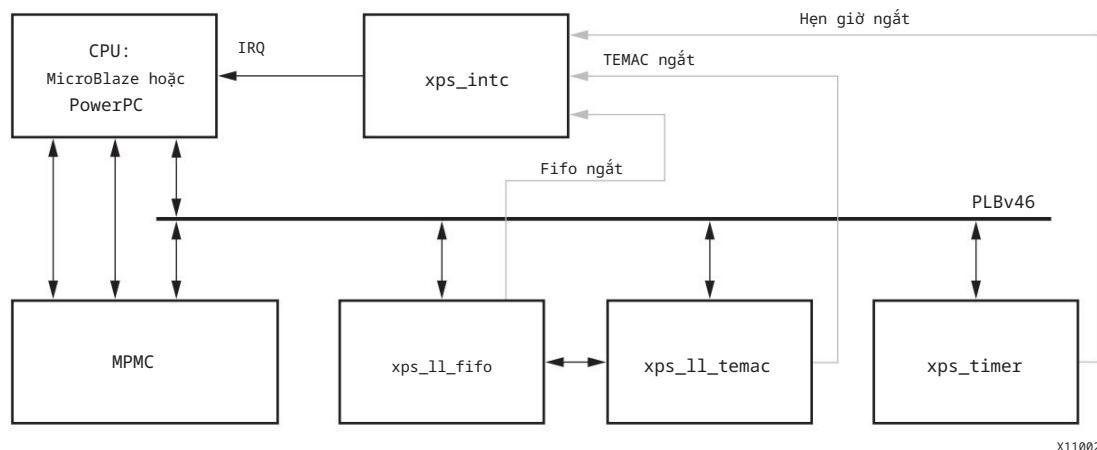


Hình 2: Kiến trúc hệ thống sử dụng xps_ll_temac Core (với DMA)

Lưu ý: Có bốn ngắt cần thiết trong trường hợp này: ngắt bộ định thời, ngắt TEMAC và ngắt SDMA RX và TX. Ngắt SDMA là từ Bộ điều khiển bộ nhớ đa cổng (MPMC).

Mô-đun giao diện tính cách SDMA (PIM). Tham khảo Bảng dữ liệu Bộ điều khiển bộ nhớ đa cổng (MPMC) để biết thêm thông tin.

Nếu TEMAC được sử dụng mà không có DMA, FIFO (xps_ll_fifo) được sử dụng để giao tiếp với TEMAC. Kiến trúc hệ thống trong trang hợp này được thể hiện trong hình sau.



X11002

Hình 3: Kiến trúc hệ thống sử dụng TEMAC với xps_ll_fifo (không có DMA)

Thiết lập Phần mềm Hệ thống

Để sử dụng lwIP trong một ứng dụng phần mềm, trước tiên bạn phải biên dịch thư viện như một phần của nền tảng phần mềm ứng dụng của mình. Để thiết lập thư viện lwIP trong XPS:

1. Mở hộp thoại Cài đặt Nền tảng Phần mềm .
2. Bật lwIP trong tab Cài đặt Thư viện / Hệ điều hành . (Đối với API Socket, Xilkernel phải là Hệ điều hành, được định cấu hình với semaphores, mutexes và chức năng lợi nhuận có sẵn).
3. Chọn Tạo thư viện và BSP để tạo lại thư viện.
4. Liên kết ứng dụng với cờ trình liên kết -l lwip4 . (Đối với API socket, hãy thêm -l xilkernel.)

Cấu hình các tùy chọn lwIP

LwIP cung cấp các tham số có thể định cấu hình. Các giá trị cho các tham số này có thể được thay đổi bằng cách sử dụng hộp thoại Cài đặt Nền tảng Phần mềm . Có hai danh mục chính của các tùy chọn có thể định cấu hình:

- Các tùy chọn Bộ điều hợp Xilinx sang lwIP: Các tùy chọn này kiểm soát các cài đặt được sử dụng bởi bộ điều hợp Xilinx cho lõi xps_ethernetlite và xps_ll_temac.
- Tùy chọn lwIP cơ sở: Các tùy chọn này là một phần của chính thư viện lwIP và bao gồm các tham số cho TCP, UDP, IP và các giao thức khác được hỗ trợ bởi lwIP.

Các phần sau đây mô tả các tùy chọn có thể định cấu hình lwIP.

Tùy chỉnh Chế độ API lwIP

Lwip130_v1_00_a hỗ trợ cả API thô và API ẩn:

- API thô được tùy chỉnh để có hiệu suất cao và chi phí bộ nhớ thấp hơn. Hạn chế của API thô là nó dựa trên cuộc gọi lại và do đó không cung cấp khả năng di động cho các ngăn xếp TCP khác.
- API socket cung cấp giao diện kiểu socket BSD và rất linh hoạt; tuy nhiên, chế độ này không hiệu quả cả về hiệu suất và yêu cầu bộ nhớ.

Lwip130_v1_00_a cũng cung cấp khả năng đặt mức độ ưu tiên trên TCP / IP và các luồng ứng dụng lwIP khác. Bảng sau cung cấp các chế độ API thư viện lwIP.

Bảng 1: Các tùy chọn và mô tả chế độ API

Thuộc tính / Tùy chọn	Sự mô tả	Loại hình	Mặc định
api_mode {RAW_API SOCKET_API}	Chế độ hoạt động của thư viện lwIP	enum RAW_API	
socket_mode_thread_prio số nguyên	Mức độ ưu tiên của luồng lwIP TCP / IP và tất cả các luồng ứng dụng lwIP. Cài đặt này chỉ áp dụng khi Xilkernel được sử dụng ở chế độ ưu tiên. Khuyến nghị rằng tất cả các luồng sử dụng lwIP chạy ở cùng một mức độ ưu tiên.	số nguyên	1

Định cấu hình các tùy chọn bộ điều hợp Xilinx

Bộ điều hợp Xilinx cho các lõi xps_etherenetlite và xps_ll_temac EMAC có thể định cấu hình.

Tùy chọn Bộ điều hợp Ethernetlite

Bảng sau cung cấp các tham số cấu hình cho xps_etherenetlite
bộ chuyển đổi.

Bảng 2: Tùy chọn Bộ điều hợp xps_etherenetlite

Thuộc tính	Sự mô tả	Loại hình	Mặc định
sw_rx_fifo_size Kích thước	Độ đệm phần mềm tính bằng byte nhận dữ liệu giữa EMAC và bộ xử lý	số nguyên	8192
sw_tx_fifo_size Kích thước	Độ đệm Phần mềm tính bằng byte của truyền dữ liệu giữa bộ xử lý và EMAC	số nguyên	8192

Tùy chọn Bộ điều hợp TEMAC

Bảng sau cung cấp các tham số cấu hình cho bộ điều hợp xps_ll_temac.

Bảng 3: Bộ điều hợp xps_LL_temac

Thuộc tính	Sự mô tả	Loại hình	Mặc định
phy_link_speed {CONFIG_LINKSPEED10 CONFIG_LINKSPEED100 CONFIG_LINKSPEED1000 CONFIG_LINKSPEED_AUTODETECT}	Tốc độ liên kết do PHY tự động thương lượng. lwIP định cấu hình TEMAC cho cài đặt tốc độ này. Cài đặt này phải chính xác để TEMAC truyền hoặc nhận gói tin. Lưu ý: Cài đặt, CONFIG_LINKSPEED_AUTODETECT, có gắng phát hiện tốc độ liên kết chính xác bằng cách đọc các thanh ghi PHY; tuy nhiên, điều này phụ thuộc vào PHY và đã được thử nghiệm với các PHY Marvell có mặt trên bảng phát triển Xilinx. Đối với các PHY khác, nên chọn tốc độ chính xác.	số nguyên	CONFIG_LINKSPEED_TU ĐỘNG PHÁT HIỆN
n_tx_descriptors	Số bộ mô tả bộ đệm TX được sử dụng trong Chế độ SDMA	số nguyên	32
n_rx_descriptors	Số lượng bộ mô tả bộ đệm RX được sử dụng trong Chế độ SDMA	số nguyên	32
n_tx_coalesce	Cài đặt liên kết ngắt TX cho TEMAC	số nguyên	1
n_rx_coalesce	Cài đặt liên kết ngắt RX cho TEMAC	số nguyên	1
tcp_tx_csum_offload	TX cho phép giảm tải tổng kiểm tra	số nguyên	1
tcp_rx_csum_offload	RX cho phép giảm tải tổng kiểm tra	số nguyên	1

Định cấu hình các tùy chọn bộ nhớ

Ngăn xếp lwIP cung cấp các loại ký ức khác nhau. Các tùy chọn bộ nhớ có thể định cấu hình được cung cấp như một danh mục riêng biệt. Các giá trị mặc định hoạt động tốt trừ khi cần điều chỉnh ứng dụng. Các tùy chọn tham số bộ nhớ khác nhau được cung cấp trong bảng sau:

Bảng 4: Các tùy chọn tham số cấu hình bộ nhớ

Thuộc tính	Sự mô tả	Loại hình	Mặc định
mem_size	Kích thước của bộ nhớ heap tính bằng byte. Đặt giá trị này cao nếu ứng dụng gửi dữ liệu lớn.	int	8192
mem_num_pbuf	Số lượng mem struct pbufs. Đặt giá trị này cao nếu ứng dụng gửi nhiều dữ liệu ra khỏi ROM hoặc bộ nhớ tĩnh.	int	16
mem_num_udp_pcb	Số khối điều khiển giao thức UDP đang hoạt động. Một trên mỗi kết nối UDP đang hoạt động.	int	5
mem_num_tcp_pcb	Số khối điều khiển giao thức TCP đang hoạt động. Một cho mỗi kết nối TCP đang hoạt động.	int	5
mem_num_tcp_pcb_listen	Số lượng kết nối TCP đang nghe.	int	5
mem_num_tcp_seg	Số lượng phân đoạn TCP được xếp hàng đồng thời.	int	255
mem_num_sys_timeout	Số lần ngắt hoạt động đồng thời.	int	3

Định cấu hình các tùy chọn bộ nhớ ồ cǎm

Chế độ API Sockets có các tùy chọn bộ nhớ. Các tùy chọn bộ nhớ ồ cǎm có thể định cấu hình được cung cấp như một danh mục riêng biệt. Các giá trị mặc định hoạt động tốt trừ khi cần điều chỉnh ứng dụng. Bảng sau cung cấp các thông số cho các tùy chọn bộ nhớ ồ cǎm.

Bảng 5: Thông số cấu hình tùy chọn bộ nhớ ồ cǎm

Thuộc tính	Sự mô tả	Loại hình	Mặc định
memp_num_netbuf	Số lượng cấu trúc netbufs. Điều này chuyển thành một trên mỗi ồ cǎm.	int	5
memp_num_netconn	Số lượng cấu trúc netconns. Điều này chuyển thành một trên mỗi ồ cǎm.	int	5
memp_num_api_msg	Số lượng struct api_msg. Được sử dụng để giao tiếp giữa ngăn xếp TCP / IP và ứng dụng.	int	16
memp_num_tcpip_msg	Số lượng struct tcpip_msg. Được sử dụng cho giao tiếp API tuần tự và các gói đến.	int	16

Lưu ý: Vì hỗ trợ Sockets Mode sử dụng các dịch vụ Xilkernel, số lượng semaphores được chọn trong cấu hình Xilkernel phải tính đến giá trị được đặt cho tham số memp_num_netbuf .

Cấu hình tùy chọn bộ nhớ đệm gói (Pbuf)

Bộ đệm gói (Pbufs) mang các gói qua các lớp khác nhau của ngăn xếp TCP / IP. Sau đây là các tùy chọn bộ nhớ pbuf được cung cấp bởi ngăn xếp lwIP. Các giá trị mặc định hoạt động tốt trừ khi cần điều chỉnh ứng dụng. Bảng sau cung cấp các thông số cho tùy chọn bộ nhớ Pbuf: Bảng 6: Thông số cấu hình tùy chọn bộ nhớ Pbuf

Thuộc tính	Sự mô tả	Loại hình	Mặc định
pbuf_pool_size	Số bộ đệm trong pbuf pool.	int	512
pbuf_pool_bufsize	Kích thước tính bằng byte của mỗi pbuf trong pbuf pool.	int	1536

Định cấu hình các tùy chọn ARP

Bảng sau cung cấp các tham số cho các tùy chọn ARP. Các giá trị mặc định hoạt động tốt trừ khi cần điều chỉnh ứng dụng.

Bảng 7: Tham số cấu hình tùy chọn ARP

Thuộc tính	Sự mô tả	Loại hình	Mặc định
arp_table_size	Số lưu lượng địa chỉ phần cứng đang hoạt động, cặp địa chỉ IP được lưu trong bộ nhớ đệm.	int	10
arp_queueing	Khi được bật, (mặc định (1)), các gói gửi đi được xếp hàng đợi trong quá trình phân giải địa chỉ phần cứng.	int	1
arp_queue_first	Khi được bật, gói đầu tiên được xếp hàng đợi sẽ không bị các gói sau ghi đè. Giá trị mặc định (0), bị vô hiệu hóa, được khuyến nghị.	int	0
etharp_always_insert	Khi được đặt thành 1, các mục nhập trong bộ nhớ cache được cập nhật hoặc thêm cho mọi lưu lưu ARP. Tùy chọn này được khuyến nghị cho các bộ định tuyến. Khi được đặt thành 0, chỉ các mục nhập bộ nhớ cache hiện có mới được cập nhật. Các mục nhập sẽ được thêm vào khi lwIP gửi đến chúng. Được đề xuất cho các thiết bị nhúng.	int	0

Định cấu hình các tùy chọn IP

Bảng sau cung cấp các tùy chọn tham số IP. Các giá trị mặc định hoạt động tốt trừ khi cần điều chỉnh ứng dụng.

Bảng 8: Các tùy chọn tham số cấu hình IP

Thuộc tính	Sự mô tả	Loại hình	Mặc định
ip_osystem	Đặt thành 1 để cho phép khả năng chuyển tiếp các gói IP qua các giao diện mạng. Nếu chạy lwIP trên một giao diện mạng, hãy đặt 0.	int	0
ip_reassembly	Lắp ráp lại các gói IP bị phân mảnh đến.	int	1
ip_frag	Phân mảnh các gói IP gửi đi nếu kích thước của chúng vượt quá MTU.	int	1
ip_options	Khi được đặt thành 1, các tùy chọn IP được cho phép (nhưng không được phân tích cú pháp). Khi được đặt thành 0, tất cả các gói có tùy chọn IP sẽ bị loại bỏ.	int	0

Định cấu hình các tùy chọn ICMP

Bảng sau cung cấp tham số cho tùy chọn giao thức ICMP. Các giá trị mặc định hoạt động tốt trừ khi cần điều chỉnh ứng dụng.

Bảng 9: Tùy chọn tham số cấu hình ICMP

Thuộc tính	Sự mô tả	Loại hình	Mặc định
icmp_ttl	Giá trị TTL ICMP.	int	255

Định cấu hình các tùy chọn UDP

Bảng sau cung cấp các tùy chọn giao thức UDP. Các giá trị mặc định hoạt động tốt trừ khi cần điều chỉnh ứng dụng.

Bảng 10: Các tùy chọn tham số cấu hình UDP

Thuộc tính	Sự mô tả	Loại hình	Mặc định
lwip_udp	Chi định xem UDP có được yêu cầu hay không.	bool	thật
udp_ttl	Giá trị TTL UDP.	int	255

Cấu hình các tùy chọn TCP

Bảng sau cung cấp các tùy chọn giao thức TCP. Các giá trị mặc định hoạt động tốt trừ khi cần điều chỉnh ứng dụng.

Bảng 11: Tham số cấu hình tùy chọn TCP

Thuộc tính	Sự mô tả	Loại hình	Mặc định
lwip_tcp	Yêu cầu TCP.	bool	thật
tcp_ttl	Giá trị TTL TCP.	int	255
tcp_wnd	Kích thư ớc cửa sổ TCP tinh bằng byte.	int	16384
tcp_maxrtx	TCP Giá trị truyền lại tối đa.	int	12
tcp_synmaxrtx	TCP Giá trị truyền lại SYN tối đa.	int	4
tcp_queue_ooseq	Chấp nhận các phân đoạn hàng đợi TCP không theo thứ tự. Đặt thành 0 nếu thiết bị của bạn sắp hết bộ nhớ.	int	1
tcp_mss	TCP Kích thư ớc phân đoạn tối đa.	int	1476
tcp_snd_buf	Không gian đệm của người gửi TCP tinh bằng byte.	int	32768

Định cấu hình các tùy chọn gỡ lỗi

ngăn xếp lwIP có thông tin gỡ lỗi. Ché độ gỡ lỗi có thể được bật để kết xuất thông báo gỡ lỗi vào STDOUT. Tùy chọn sau, khi được đặt thành true, sẽ in thông báo gỡ lỗi.

Bảng 12: Tham số cấu hình tùy chọn gỡ lỗi

Thuộc tính	Sự mô tả	Loại hình	Mặc định
lwip_debug	Bật Gỡ lỗi lwIP	bool	sai

Định cấu hình tùy chọn thống kê

ngăn xếp lwIP đã được viết để thu thập số liệu thống kê, chẳng hạn như số lượng kết nối được sử dụng; dung lượng bộ nhớ được sử dụng; và số lượng semaphores được sử dụng cho ứng dụng. Thư viện cung cấp API stats_display () để kết xuất các thống kê có liên quan đến ngữ cảnh mà lệnh gọi được sử dụng. Tùy chọn thống kê có thể được bật để cho phép thu thập và hiển thị thông tin thống kê khi API stats_display được gọi từ mã người dùng. Sử dụng tùy chọn sau để cho phép thu thập thông tin thống kê cho ứng dụng.

Bảng 13: Tham số cấu hình tùy chọn thống kê

Thuộc tính	Sự mô tả	Loại hình	Mặc định
lwip_stats	Bật thống kê lwIP	int	0

API phần mềm

lwIP cung cấp hai API khác nhau: chế độ RAW và chế độ Socket.

API thô

API thô dựa trên lệnh gọi lại. Các ứng dụng có quyền truy cập trực tiếp vào ngăn xếp TCP và ngược lại. Do đó, không có việc sao chép dữ liệu không cần thiết và sử dụng API thô mang lại hiệu suất tuyệt vời với mức giá tương thích với các ngăn xếp TCP khác.

Yêu cầu bộ điều hợp Xilinx khi sử dụng API RAW

Ngoài API lwIP RAW, các bộ điều hợp Xilinx cung cấp chức năng tiện ích xemacif_input để nhận các gói tin. Hành này phải được gọi theo các khoảng thời gian thường xuyên để di chuyển các gói đã nhận từ các trình xử lý ngắn sang ngăn xếp lwIP. Tùy thuộc vào loại gói nhận được, lwIP sau đó gọi các cuộc gọi lại ứng dụng đã đăng ký.

Tệp API thô

Tệp \$ XILINX_EDK / sw / ThirdParty / sw_services / lwip130_v1_00_a / src / lwip 1.3.0 / doc / rawapi.txt mô tả API thô lwIP.

API ồ cǎm

API ồ cǎm lwIP cung cấp một API kiểu ồ cǎm BSD cho các chương trình. API này cung cấp một mô hình thực thi là một mô hình chặn, mở-đọc-ghi-đóng.

Yêu cầu của bộ điều hợp Xilinx khi sử dụng API Socket

Các ứng dụng sử dụng API Socket với bộ điều hợp Xilinx cần tạo ra một luồng riêng có tên là xemacif_input_thread. Luồng này đảm nhận việc di chuyển các gói đã nhận từ trình xử lý ngắn đến tcpip_thread của lwIP. Các chuỗi ứng dụng sử dụng lwIP phải được tạo bằng API lwIP sys_thread_new. Bên trong, hành này sử dụng hàm pthread_create () trong Xilkernel để tạo một luồng mới. Nó cũng khởi tạo cấu trúc thời gian chờ cụ thể cho mỗi luồng cần thiết cho hoạt động của lwIP.

Chính sách lập lịch Xilkernel khi sử dụng API Socket

lwIP ở chế độ socket yêu cầu sử dụng Xilkernel, cung cấp hai chính sách cho việc lập lịch luồng: theo vòng và dựa trên mức độ ưu tiên:

Không có yêu cầu đặc biệt nào khi chính sách lập lịch quay vòng được sử dụng vì tất cả các luồng đều nhận được lượng tử thời gian giống nhau.

Với lập lịch ưu tiên, phải cẩn thận để đảm bảo rằng các luồng lwIP không bị chép đói. lwIP khởi chạy nội bộ tất cả các luồng ở mức ưu tiên được chỉ định trong socket_mode_thread_prio.

Ngoài ra, các chuỗi ứng dụng phải khởi chạy xemacif_input_thread. Mức độ ưu tiên của cả luồng xemacif_input_thread và luồng nội bộ lwIP (socket_mode_thread_prio) phải đủ cao so với các luồng ứng dụng khác để chúng không bị bỏ sót.

Sử dụng các chức năng của trình trợ giúp bộ điều hợp Xilinx

Bộ điều hợp Xilinx cung cấp các chức năng trợ giúp sau đây để đơn giản hóa việc sử dụng các API lwIP.

```
void lwip_init ()
```

Hàm này cung cấp một hàm khởi tạo duy nhất cho cấu trúc dữ liệu lwIP. Điều này thay thế các lệnh gọi cụ thể để khởi tạo các lớp thống kê, hệ thống, bộ nhớ, pbufs, ARP, IP, UDP và TCP.

```
struct netif * xemac_add (struct netif * netif, struct ip_addr *  
    ipaddr, struct ip_addr * netmask, struct ip_addr * gw, unsigned  
    char * mac_etherenet_address unsigned mac_baseaddr)
```

Hàm xemac_add cung cấp một giao diện thống nhất để thêm bất kỳ IP Xilinx EMAC nào. Hàm này là một trình bao bọc xung quanh hàm lwIP netif_add khởi tạo giao diện mạng 'netif' với địa chỉ IP ipaddr, netmask, địa chỉ IP của cổng, gw, địa chỉ ethernet 6 byte mac_etherenet_address và địa chỉ cơ sở, mac_baseaddr, của lõi MAC xps_ethernetlite hoặc xps_ll_temac.

```
void xemacif_input (struct netif * netif)
```

(Chỉ chế độ RAW)

Bộ điều hợp Xilinx lwIP hoạt động ở chế độ ngắn. Bộ xử lý ngắn nhận dữ liệu gói từ EMAC và lưu trữ chúng trong một hàng đợi. Hàm xemacif_input lấy các gói đã nhận từ hàng đợi và chuyển chúng đến lwIP; do đó, chức năng này được yêu cầu cho hoạt động lwIP ở chế độ RAW. Một ứng dụng lwIP ở chế độ RAW phải có cấu trúc như sau:

```
trong khi (1) {  
    /* nhận gói tin */  
    xemacif_input (netif);  
  
    /* xử lý ứng dụng cụ thể */  
}
```

Chương trình được thông báo về dữ liệu nhận được thông qua các cuộc gọi lại.

```
void xemacif_input_thread (struct netif * netif)
```

(Chỉ chế độ ỗ cắm)

Trong chế độ socket, luồng ứng dụng phải khởi chạy một luồng riêng để nhận các gói đầu vào. Điều này thực hiện công việc tương tự như chức năng chế độ RAW, xemacif_input, ngoại trừ việc nó nằm trong luồng riêng biệt của riêng nó; do đó, bất kỳ ứng dụng chế độ ỗ cắm lwIP nào đều được yêu cầu phải có mã tương tự như sau trong luồng chính của nó:

```
sys_thread_new ("xemacif_input_thread",  
    xemacif_input_thread, netif, THREAD_STACK_SIZE, DEFAULT_THREAD_PRIO);
```

Sau đó, ứng dụng có thể tiếp tục khởi chạy các luồng riêng biệt để thực hiện các tác vụ cụ thể của ứng dụng. Xemacif_input_thread nhận dữ liệu được xử lý bởi trình xử lý ngắn và chuyển chúng đến lwIP tcpip_thread.

lwIP

Phần này cung cấp tổng quan ngắn gọn về hiệu suất mong đợi khi sử dụng lwIP với Xilinx Ethernet MAC.

Màn biểu diễn

Bảng sau cung cấp thông tin TCP tối đa có thể đạt được bằng FPGA, CPU, EMAC và tần số hệ thống ở chế độ RAW và Socket. Các ứng dụng yêu cầu hiệu suất cao nên sử dụng API RAW.

Bảng 14: Hiệu suất Thư viện

FPGA	CPU	EMAC	Hệ thống Frequency	Thông tin TCP tối đa	
				Chế độ RAW	Chế độ ô cắm
Virtex®	PPC405 xps_ll_temac		100 MHz	140 Mb / giây	40 Mb / giây
Virtex	Microblaze xps_ll_temac		125 MHz	100 Mbps	30 Mb / giây
Spartan®	Microblaze xps_ll_temac		66 MHz	35 Mbps	10 Mb / giây
Spartan	Microblaze xps_ethernetlite	66 MHz	15 Mbps		7 Mb / giây

Các vấn đề đã biết và**Những hạn chế****Di chuyển từ lwip_v3_00_a sang lwip130_v1_00_a**

Lwip130_v1_00_a không hỗ trợ nhiều hơn một TEMAC trong một phiên bản xps_ll_temac. Ví dụ: lwip130_v1_00_a không hỗ trợ TEMAC được bật bằng cách đặt C_TEMAC1_ENABLED = 1 trong xps_ll_temac.

Các ứng dụng được viết để hoạt động với lwip_v3_00_a phải thực hiện các thay đổi sau để hoạt động với lwip130_v1_00_a:

- API cho hàm sys_thread_new đã thay đổi từ lwIP 1.2.0 thành lwIP 1.3.0. Sử dụng API mới như sau:


```
sys_thread_t sys_thread_new (char * name, void (* thread) (void * arg), void * arg, int stacksize, int prio);
```
- Định cấu hình Xilkernel để bao gồm chức năng lợi nhuận.
- Các chức năng gọi lại chế độ UDP RAW nhận một con trỏ đến địa chỉ IP của người gửi như một trong các tham số. Không chuyển lại tham số này cho bất kỳ hàm UDP nào khác làm đối số. Thay vào đó, hãy tạo một bản sao và chuyển một con trỏ tới bản sao.

Ví dụ về API

Các ứng dụng mẫu sử dụng API RAW và API Socket có sẵn trên trang web Xilinx.
Phần này cung cấp mã giả minh họa cấu trúc mã điển hình.

API RAW

Các ứng dụng sử dụng API RAW là một luồng và có cấu trúc rộng sau:

```
int main ()
{
    struct netif * netif, server_netif;
    struct ip_addr ipaddr, netmask, gw;

    /* địa chỉ MAC của bảng.
     * Điều này phải là duy nhất trên mỗi bảng / PHY */
    unsigned char mac_ethernet_address [] =
        {0x00, 0xa, 0x35, 0x00, 0x01, 0x02};

    lwip_init ();

    /* Thêm giao diện mạng vào netif_list,
     * và đặt nó làm mặc định */
    if (! xemac_add (netif, & ipaddr, & netmask, & gw,
                     mac_ethernet_address, EMAC_BASEADDR)) {

        printf ("Lỗi khi thêm giao diện N / W \ n \ r");
        trả về -1;
    }
    netif_set_default (netif);

    /* bây giờ cho phép ngắn */
    platform_enable_interrupts ();

    /* chỉ định rằng mạng nếu được lên */
    netif_set_up (netif);

    /* khởi động ứng dụng, thiết lập các cuộc gọi lại */
    start_application ();

    /* nhận và xử lý gói tin */
    trong khi (1) {
        xemacif_input (netif);
        /* chức năng cụ thể của ứng dụng */
        truyền tải dữ liệu();
    }
}
```

API RAW hoạt động chủ yếu bằng cách sử dụng các lệnh gọi lại Gửi và Nhận không đồng bộ.

API ỗ cám

Trong chế độ ỗ cám, các ứng dụng chỉ định một danh sách tĩnh các chủ đề mà Xilkernel tạo ra khi khởi động trong cài đặt nền tảng phần mềm Xilkernel. Giả sử rằng main_thread () là một luồng được chỉ định để khởi chạy bởi Xilkernel, thì đoạn mã giả sau đây minh họa cấu trúc chương trình chế độ ỗ cám điển hình

```
void network_thread (void * p)
{
    struct netif * netif;
    struct ip_addr ipaddr, netmask, gw;

    /* Địa chỉ MAC của bảng.
     * Điều này phải là duy nhất trên mỗi bảng / PHY */
    unsigned char mac_ethernet_address [] =
        {0x00, 0x0a, 0x35, 0x00, 0x01, 0x02};

    netif = & server_netif;

    /* Khởi tạo địa chỉ IP sẽ được sử dụng */
    IP4_ADDR (& ipaddr, 192,168,1,10);
    IP4_ADDR (& netmask, 255,255,255,0);
    IP4_ADDR (& gw, 192,168,1,1);

    /* Thêm giao diện mạng vào netif_list,
     * và đặt nó làm mặc định */
    if (! xemac_add (netif, & ipaddr, & netmask,
                     & gw, mac_ethernet_address,
                     EMAC_BASEADDR)) {
        printf ("Lỗi khi thêm giao diện N / W \ n \ r");
        trả về;
    }
    netif_set_default (netif);

    /* chỉ định rằng mạng nếu được lên */
    netif_set_up (netif);

    /* bắt đầu chuỗi nhận gói
     * cần thiết cho hoạt động lwIP */
    sys_thread_new ("xemacif_input_thread", xemacif_input_thread,
                    netif,
                    THREAD_STACKSIZE, DEFAULT_THREAD_PRIO);

    /* bây giờ chúng ta có thể bắt đầu các chuỗi ứng dụng */
    /* bắt đầu chuỗi máy chủ web (ví dụ) */
    sys_thread_new ("httpd" web_application_thread, 0,
                    THREAD_STACKSIZE DEFAULT_THREAD_PRIO);
}

int main_thread ()
{
    /* khởi tạo lwIP trước khi gọi sys_thread_new */
    lwip_init ();

    /* bắt kỳ chủ đề nào sử dụng lwIP nên được tạo bằng
     * sys_thread_new () */
    sys_thread_new ("network_thread" network_thread, NULL,
                    THREAD_STACKSIZE DEFAULT_THREAD_PRIO);

    trả về 0;
}
```



LibXil Flash

UG651 ngày 16 tháng 9 năm 2009

Bản tóm tắt

Tài liệu này mô tả thư viện XilFlash dành cho các thiết bị flash song song tuân thủ CFI. Thư viện này cung cấp chức năng đọc / ghi / xóa / khóa / mở khóa và các chức năng cụ thể của thiết bị cho thiết bị flash.

Tài liệu bao gồm các phần sau:

- "Tổng quan"
- "API thư viện XilFlash"
- "Tùy chỉnh Libgen"

Tổng quan

Thư viện XilFlash cung cấp các tính năng đọc / ghi / xóa / khóa / mở khóa để truy cập thiết bị flash song song. Thư viện cũng hỗ trợ các chức năng cụ thể của họ thiết bị Flash. Thư viện này yêu cầu nền tảng phần cứng bên dưới phải chứa những thứ sau: • xps_mch_emc hoặc lõi tư ơng tự để truy cập flash.

Thư viện này triển khai chức năng cho các thiết bị bộ nhớ flash tuân theo tiêu chuẩn "Giao diện Flash chung" (CFI). CFI cho phép sử dụng một thư viện flash duy nhất cho toàn bộ nhóm bộ phận. Thư viện này hỗ trợ các thiết bị bộ nhớ flash tương thích với Intel và AMD CFI.

Tất cả các lệnh gọi trong thư viện về bản chất là bị chặn trong đó điều khiển chỉ được trả lại cho người dùng sau khi hoàn thành hoạt động hiện tại một cách thành công hoặc một lỗi được báo cáo.

Các API phổ biến sau được hỗ trợ cho tất cả các thiết bị flash:

- Khởi tạo
- Đọc
- Viết
- Tẩy xóa
- Khoá
- Mở khóa
- Sẵn sàng
- Cài lại
- Kiểm soát thiết bị cụ thể

Người dùng phải gọi API "`int XFlash_Initialize (XFlash * InstancePtr)`" trước khi gọi bất kỳ API nào khác.

Thư viện XilFlash API

Phần này cung cấp bản tóm tắt đư ợc liên kết của các API thư viện LibXil Flash và mô tả chi tiết về các API đó.

Tóm tắt API

Sau đây là danh sách tóm tắt các API đư ợc cung cấp bởi thư viện LibXil Flash. Danh sách đư ợc liên kết với mô tả API. Nhấp vào tên API để chuyển đến phần mô tả.

```
int XFlash_Initialize (XFlash * InstancePtr)
int XFlash_Reset (XFlash * InstancePtr)
int XFlash_Read (XFlash * InstancePtr, u32 Offset, u32 Byte, void * DestPtr)
int XFlash_Write (XFlash * InstancePtr, u32 Offset, u32 Byte, void * SrcPtr)
int XFlash_Erase (XFlash * InstancePtr, u32 Offset, u32 Byte)
int XFlash_Lock (XFlash * InstancePtr, u32 Offset, u32 Byte)
int XFlash_UnLock (XFlash * InstancePtr, u32 Offset, u32 Byte)
int XFlash_DeviceControl (XFlash * InstancePtr, u32 Command, DeviceControl * Tham số)
int XFlash_IsReady (XFlash * InstancePtr)
```

Mô tả API thư viện XilFlash

`int XFlash_Initialize (XFlash * InstancePtr)`

Thông số	InstancePtr là một con trỏ đến XFlash Instance.
Lợi nhuận	XST_SUCCESS nếu thành công. XFLASH_PART_NOT_SUPPORTED nếu thuật toán đặt lệnh hoặc bộ cục không được hỗ trợ bởi bất kỳ họ flash nào được biên dịch vào hệ thống.
	XFLASH_CFI_QUERY_ERROR nếu thiết bị không vào chế độ truy vấn CFI. Thiết bị không hỗ trợ CFI hoặc tồn tại bộ phận không được hỗ trợ hoặc có sự cố phần cứng.
Sự mô tả	Khởi tạo một phiên bản XFlash cụ thể. Quá trình khởi tạo đòi hỏi: <ul style="list-style-type: none"> • Kiểm tra loại Gia đình thiết bị • Ban hành lệnh truy vấn CFI • Đặt các tùy chọn mặc định cho phiên bản • Thiết lập VTable • Khởi tạo Xilinx Platform Flash XL sang chế độ Không đồng bộ nếu người dùng chọn sử dụng Platform Flash XL trong MLD. Platform Flash XL là thiết bị khiếu nại CFI của Intel.
Bao gồm	xilflash.h xilflash_cfi.h xilflash_intel.h xilflash_amd.h

`int XFlash_Reset (XFlash * InstancePtr)`

Thông số	InstancePtr là một con trỏ đến XFlash Instance.
Lợi nhuận	XST_SUCCESS nếu Thành công. XFLASH_BUSY nếu thiết bị flash đang trong quá trình hoạt động và không thể đặt lại.
	XFLASH_ERROR nếu thiết bị đã gặp lỗi nội bộ trong quá trình hoạt động. XFlash_DeviceControl () phải được sử dụng để truy cập nguyên nhân của tình trạng lỗi cụ thể của thiết bị.
Sự mô tả	API này đặt lại thiết bị flash và đặt nó ở chế độ đọc.
Bao gồm	xilflash.h xilflash_cfi.h xilflash_intel.h xilflash_amd.h

```
int XFlash_Read (XFlash * InstancePtr, u32 Offset, u32
Byte, void * DestPtr)
```

Thông số

InstancePtr là một con trỏ đến XFlash Instance.

Offset là phần bù vào không gian địa chỉ thiết bị để đọc.

Byte là số byte cần đọc.

DestPtr là Địa chỉ đích để sao chép dữ liệu vào.

Lợi nhuận

XST_SUCCESS nếu thành công.

XFLASH_ADDRESS_ERROR nếu địa chỉ nguồn không bắt đầu trong vùng có thể định địa chỉ của thiết bị.

Sự mô tả

API này đọc dữ liệu từ thiết bị flash và sao chép nó vào bộ đệm ngay khi dùng được chỉ định. Địa chỉ nguồn và địa chỉ đích có thể nằm trên bất kỳ sự liên kết nào được bộ xử lý hỗ trợ.

Bao gồm

xilflash.h

xilflash_cfi.h

xilflash_intel.h

xilflash_amd.h

```
int XFlash_Write (XFlash * InstancePtr, u32 Offset, u32
Byte, void * SrcPtr)
```

Thông số

InstancePtr là một con trỏ đến XFlash Instance.

Offset là phần bù vào không gian địa chỉ thiết bị để bắt đầu lập trình.

Byte là số byte trong Chương trình.

SrcPtr là Địa chỉ nguồn chứa dữ liệu được lập trình.

Lợi nhuận

XST_SUCCESS nếu Thành công.

XFLASH_ERROR nếu xảy ra lỗi ghi. Lỗi thường là của thiết bị cụ thể. Sử dụng XFlash_DeviceControl () để truy xuất các điều kiện lỗi cụ thể. Khi lỗi này được trả về, có thể dài địa chỉ đích chỉ được lập trình một phần.

Sự mô tả

API này lập trình thiết bị flash với dữ liệu được chỉ định trong bộ đệm ngay khi dùng. Địa chỉ nguồn và địa chỉ đích phải được căn chỉnh theo chiều rộng của bus dữ liệu flash.

Bao gồm

xilflash.h

xilflash_cfi.h

xilflash_intel.h

xilflash_amd.h

```
int XFlash_Erase (XFlash * InstancePtr, u32 Offset, u32
Byte)
```

Thông số

InstancePtr là một con trỏ đến Xflash Instance.

Offset là phần bù vào không gian địa chỉ thiết bị mà từ đó bắt đầu xóa.

Byte là số byte cần Xóa.

Lợi nhuận

XST_SUCCESS nếu thành công.

XFLASH_ADDRESS_ERROR nếu dải địa chỉ đích không hoàn toàn nằm trong vùng có thể định địa chỉ của thiết bị.

Sự mô tả

API này xóa phạm vi địa chỉ được chỉ định trong thiết bị flash.

Số byte cần xóa có thể là bất kỳ số nào miễn là nó nằm trong giới hạn của thiết bị.

Bao gồm

xilflash.h

xilflash_cfi.h

xilflash_intel.h

xilflash_amd.h

```
int XFlash_Lock (XFlash * InstancePtr, u32 Offset, u32
Byte)
```

Thông số

InstancePtr là một con trỏ đến Xflash Instance.

Offset là phần bù của địa chỉ khôi vào vùng địa chỉ thiết bị cần được khóa.

Byte là số byte được khóa.

Lợi nhuận

XST_SUCCESS nếu thành công.

XFLASH_ADDRESS_ERROR nếu dải địa chỉ đích không hoàn toàn nằm trong vùng có thể định địa chỉ của thiết bị.

Sự mô tả

API này khóa một khôi trong thiết bị flash.

Bao gồm

xilflash.h

xilflash_cfi.h

xilflash_intel.h

xilflash_amd.h

```
int XFlash_UnLock (XFlash * InstancePtr, u32 Offset, u32
Byte)
```

Thông số

InstancePtr là một con trỏ đến XFlash Instance.

Offset là phần bù của địa chỉ khôi vào không gian địa chỉ thiết bị cần
đư ợc mở khóa.

Byte là số byte đư ợc mở khóa.

Lợi nhuận

XST_SUCCESS nếu thành công.

XFLASH_ADDRESS_ERROR nếu dải địa chỉ đích không hoàn toàn nằm trong vùng
có thẻ định địa chỉ của thiết bị.

Sự mô tả

API này mở khóa các khôi đã khóa trước đó đã bị khóa.

Bao gồm

xilflash.h

xilflash_cfi.h

xilflash_intel.h

xilflash_amd.h

```
int XFlash_DeviceControl (XFlash * InstancePtr, u32
Command, DeviceControl * Tham số)
```

Thông số

InstancePtr là một con trỏ đến XFlash Instance.

Lệnh là lệnh thiết bị cụ thể để phát hành.

Tham số chỉ định các đối số đư ợc truyền cho chức năng điều khiển thiết
bị.

Lợi nhuận

XST_SUCCESS nếu thành công.

XFLASH_NOT_SUPPORTED nếu lệnh không đư ợc thiết bị hỗ trợ.

Sự mô tả

API này đư ợc sử dụng để thực thi các lệnh cụ thể của thiết bị.

Bao gồm

xilflash.h

xilflash_cfi.h

xilflash_intel.h

xilflash_amd.h

```
int XFlash_IsReady (XFlash * InstancePtr)
```

Thông số

InstancePtr là một con trỏ đến XFlash Instance.

Lợi nhuận

TRUE nếu thiết bị đã đư ợc khởi tạo; ngược lại, FALSE.

Sự mô tả

API này kiểm tra tính sẵn sàng của thiết bị, có nghĩa là nó đã đư ợc khởi
tạo thành công.

Bao gồm

xilflash.h

xilflash_cfi.h

xilflash_intel.h

xilflash_amd.h

Thư viện XilFlash có thể được tích hợp với một hệ thống bằng cách sử dụng đoạn mã sau trong Tệp thông số kỹ thuật phần mềm vi xử lý (MSS):

```
BẮT ĐẦU THU VIÊN
tham số LIBRARY_NAME = xilflash
tham số LIBRARY_VER = 1.03.a
tham số PROC_INSTANCE = ppc405_0
tham số flash_family = 1
tham số part_width = 2
tham số num_parts = 2
tham số part_mode = 2
tham số base_address = 0x80800000
tham số platform_flash = 0x0
```

CHẤM DỨT

Mô tả về Thông Số:

- **flash_family** chỉ định họ thuộc về thiết bị flash, nó phải được đặt thành 1 cho các thiết bị flash tuân thủ Intel CFI và nên được đặt thành 2 để hỗ trợ các thiết bị flash tuân thủ AMD CFI.
- **part_width** chỉ định độ rộng bus dữ liệu được hỗ trợ bởi phần flash.
- **num_part** chỉ định số lượng bộ phận thiết bị flash trong mảng tạo thành đèn flash kí niêm.
- **part_mode** chỉ định độ rộng bus dữ liệu của phần flash thực sự được sử dụng.
- **base_address** chỉ định địa chỉ cơ sở flash.
- **platform_flash** chỉ định xem thiết bị Flash có phải là thiết bị Xilinx Platform Flash XL hay không, nó nên được đặt thành 1 nếu Flash là thiết bị Xilinx Platform Flash XL, nếu không, nó phải được đặt thành 0.



LibXil Isf

EDK 10.1, Gói dịch vụ 2, ngày
25 tháng 4 năm 2008

Bản tóm tắt

Thư viện Xilinx® In-system và Serial Flash (XilIsf) hỗ trợ Xilinx In-System Flash và Serial Flash Memories bên ngoài từ Atmel (AT45XXXD), Intel (S33) và ST Microelectronics (STM) (M25PXX). Thư viện cho phép phần mềm lớp cao hơn (chẳng hạn như một ứng dụng) giao tiếp với Serial Flash.

Tài liệu này bao gồm các phần sau:

- [Tổng quan về Thư viện LibXil Isf](#)
- [Các API thư viện LibXil Isf](#)
- [Tùy chỉnh Libgen](#)
- [Tài nguyên bổ sung](#)

LibXil Isf

Thư viện LibXil Isf:

Thư viện Tổng quan

- Cho phép người dùng Viết, Đọc và Xóa Flash nối tiếp. • Cho phép bảo vệ dữ liệu được lưu trữ trong Serial Flash khỏi sự sửa đổi không chính đáng bằng cách bật tính năng Bảo vệ khu vực.
- Hỗ trợ Flash trong hệ thống Xilinx và bộ nhớ flash nối tiếp bên ngoài của Atmel (AT45XXXD), Intel (S33) và ST Microelectronics (STM) (M25PXX).
- Hỗ trợ nhiều phiên bản Serial Flash cùng một lúc, miễn là chúng thuộc cùng một họ thiết bị (Atmel, Intel hoặc STM) khi họ thiết bị được chọn tại thời điểm biên dịch.
- Cho phép ứng dụng người dùng thực hiện các hoạt động Điều khiển khác nhau trên Intel và STM Serial Tốc biến.
- Yêu cầu nền tảng phần cứng bên dưới chứa thiết bị xps_spi để truy cập Đèn flash nối tiếp.
- Sử dụng trình điều khiển Xilinx XSpi ở chế độ điều khiển gián đoạn hoặc chế độ được thăm dò để giao tiếp với Serial Flash. Trong chế độ ngắn, ứng dụng người dùng phải xác nhận mọi ngắt liên quan từ Bộ điều khiển ngắt.
- Yêu cầu ứng dụng người dùng theo dõi trạng thái của các hoạt động bắt đầu khi ở chế độ ngắn; quá trình chuyển giao được bắt đầu và quyền kiểm soát được trao lại cho ứng dụng người dùng.

© Bản quyền 2002 - 2008 Xilinx, Inc. Mọi quyền được bảo lưu. XILINX, biểu trưng Xilinx, Cửa sổ thư mục và các thương hiệu được chỉ định khác có trong tài liệu này là các thương hiệu của Xilinx, Inc. Tên và logo PowerPC là thương hiệu đã đăng ký của IBM Corp. và được sử dụng theo giấy phép. Tất cả các nhãn hiệu khác là tài sản của chủ sở hữu tương ứng của họ.

Tuyên bố tự chịu trách nhiệm: Xilinx tiết lộ hướng dẫn sử dụng, số tay, ghi chú phát hành và / hoặc thông số kỹ thuật ("Tài liệu") này cho bạn chỉ để sử dụng trong việc phát triển các thiết kế để vận hành với các thiết bị phần cứng Xilinx. Bạn không được sao chép, phân phối, tái xuất bản, tái xuống, hiển thị, đăng hoặc truyền Tài liệu dưới bất kỳ hình thức nào hoặc bằng bất kỳ phương tiện nào bao gồm, nhưng không giới hạn ở, điện tử, cơ khí, photocopy, ghi âm, hoặc cách khác, mà không có sự đồng ý trước bằng văn bản của Xilinx. Xilinx từ chối rõ ràng bất kỳ trách nhiệm pháp lý nào phát sinh từ sử dụng Tài liệu. Xilinx bảo lưu quyền, theo quyết định riêng của mình, thay đổi Tài liệu mà không cần thông báo bất kỳ lúc nào. Xilinx không có nghĩa vụ sửa chữa bất kỳ các lỗi có trong Tài liệu hoặc để thông báo cho bạn về bất kỳ sửa chữa hoặc cập nhật nào. Xilinx từ chối rõ ràng bất kỳ trách nhiệm pháp lý nào liên quan đến hỗ trợ hoặc hỗ trợ kỹ thuật có thể được cung cấp cho bạn liên quan đến Thông tin. GIẤY TỜ ĐƯỢC CÔNG BỐ CHO BẠN "NGUYỄN TRANG" KHÔNG ĐƯỢC BÁO HÀNH BẤT KỲ HÌNH THỨC NÀO. XILINX KHÔNG CÓ BẢO ĐÁM KHÁC NÀO KHÁC RÕ Ràng, NGƯỜI HOẶC KHÔNG PHÂN BỐ QUYỀN CỦA BÊN THỨ BA. TRONG KHÔNG SỰ KIỆN XILINX SẼ CHỊU TRÁCH NHIỆM PHÁP LÝ ĐỐI VỚI BẤT KỲ SỰ KIỆN NÀO CÁC THIẾT HẠI HÀU QUẢ, CHỈ ĐỊNH, NGOẠI LỆ, ĐẶC BIỆT HOẶC BẤT CỨ SỰ CỐ, BAO GỒM BẤT CỨ MẤT DỮ LIỆU HOẶC MẤT LỢI NHUẬN NÀO DO VIỆC SỬ DỤNG CỦA BẠN CỦA TÀI LIỆU.

LibXil Isf

Phần này cung cấp bản tóm tắt được liên kết về các API thư viện LibXil Isf và mô tả chi tiết về các API đó.

API thư viện

Tóm tắt API

Sau đây là danh sách tóm tắt các API được cung cấp bởi thư viện LibXil Isf. Danh sách được liên kết với mô tả API. Nhấp vào tên API để chuyển đến phần mô tả.

```
int XIsf_Initialize (XIIsf * InstancePtr, XSpi * SpiInstPtr, u32 SlaveSelect, u8 * WritePtr)
int XIsf_GetStatus (XIIsf * InstancePtr, u8 * ReadPtr)
int XIsf_GetDeviceInfo (XIIsf * InstancePtr, u8 * ReadPtr)
int XIsf_Read (XIIsf * InstancePtr, XIIsf_ReadOperation Hoạt động, void * OpParamPtr)
int XIsf_Write (XIIsf * InstancePtr, XIIsf_WriteOperation Hoạt động, void * OpParamPtr)
int XIsf_Erase (XIIsf * InstancePtr, XIIsf_EraseOperation Hoạt động, Địa chỉ u32)
int XIsf_SectorProtect (XIIsf * InstancePtr, XIIsf_SpOperation Hoạt động, u8 * BufferPtr)
int XIsf_WriteEnable (XIIsf * InstancePtr, u8 WriteEnable)
int XIsf_Ioctl (XIIsf * InstancePtr, XIIsf_IoctlOperation Hoạt động)
```

Mô tả API LibXil Isf

```
int XIIsf_Initialize (XIIsf * InstancePtr, XSpi * SpiInstPtr,
                      u32 SlaveSelect, u8 * WritePtr)
```

Tham số InstancePtr là một con trỏ đến thẻ hiện XIIsf.

SpiInstPtr là một con trỏ tới thẻ hiện XSpi sẽ được làm việc.

SlaveSelect là một mặt nạ 32 bit với 1 ở vị trí bit của nô lệ được chọn. Mỗi lần chỉ có thể chọn một nô lệ.

WritePtr là một con trỏ tới bộ đệm do ngư ời dùng cấp phát để được sử dụng bởi In system và Serial Flash Library để thực hiện bất kỳ thao tác đọc / ghi nào trên thiết bị Serial Flash.

Ứng dụng ngư ời dùng phải khởi tạo thư viện Isf bằng cách chuyển địa chỉ của vùng đệm này tới API khởi tạo.

Đối với các hoạt động Viết:

- Có thể ghi tối thiểu một byte và tối đa ISF_PAGE_SIZE byte vào Serial Flash, thông qua một thao tác Ghi.
- Kích thước bộ đệm phải bằng số byte được ghi vào Serial Flash + XISF_CMD_MAX_EXTRA_BYTES và phải đủ lớn để sử dụng trên các ứng dụng sử dụng phiên bản chung của Serial Flash.

Đối với các hoạt động Không ghi:

- Kích thước bộ đệm phải bằng XISF_CMD_MAX_EXTRA_BYTES.

Lợi nhuận

XST_SUCCESS khi thành công.

XST_DEVICE_IS_STOPPED nếu thiết bị phải được khởi động trước khi chuyển dữ liệu.

XST_FAILURE khi bị lỗi.

Mô tả Hình dạng của Flash nối tiếp bên dưới được xác định bằng cách đọc

Thông tin thiết bị của Hội đồng kỹ thuật thiết bị điện tử chung (JEDEC) và Số đăng ký trạng thái đèn nháy nối tiếp.

API này phải được gọi trước khi sử dụng bất kỳ API nào khác trong thư viện này.

Lưu ý: API XIIsf_Initialize () là lệnh gọi chặn (đối với cả chế độ đã tham dò và chế độ ngắn của trình điều khiển SPI). Nó đọc thông tin JEDEC của thiết bị và đợi cho đến khi quá trình chuyển hoàn tất trước khi kiểm tra xem thông tin có hợp lệ hay không.

Thư viện này có thể hỗ trợ nhiều phiên bản Serial Flash cùng một lúc, miễn là chúng thuộc cùng một họ thiết bị (Atmel, Intel hoặc STM) khi họ thiết bị được chọn tại thời điểm biên dịch.

Bao gồm

xilisf.h

```
int XIsf_GetStatus (XIsf * InstancePtr, u8 * ReadPtr)
```

Thông số	InstancePtr là một con trỏ đến thẻ hiện XIsf.
	ReadPtr là một con trỏ tới bộ nhớ nơi i nội dung thanh ghi trạng thái đư ợc sao chép.
Lợi nhuận	XST_SUCCESS khi thành công
	XST_FAILURE khi thất bại
Sự mô tả	API này đọc Đăng ký trạng thái Flash nối tiếp. Lưu ý: Nội dung của thanh ghi trạng thái đư ợc lưu trữ ở byte thứ hai đư ợc trả bởi ReadPtr.
Bao gồm	xilisf.h

```
int XIsf_GetDeviceInfo (XIsf * InstancePtr, u8 * ReadPtr)
```

Thông số	InstancePtr là một con trỏ đến thẻ hiện XIsf.
	ReadPtr là một con trỏ tới bộ nhớ nơi i thông tin Thiết bị đư ợc sao chép.
Lợi nhuận	XST_SUCCESS khi thành công.
	XST_FAILURE khi bị lỗi.
Sự mô tả	API này đọc thông tin JEDEC của Serial Flash. Lưu ý: Thông tin thiết bị đư ợc lưu trữ ở byte thứ hai đư ợc trả bởi ReadPtr.
Bao gồm	xilisf.h

```
int XIIsf_Read (XIIsf * InstancePtr, XIIsf_ReadOperation
    Hoạt động, void * OpParamPtr)
```

Thông số

InstancePtr là một con trỏ đến thẻ hiện XIIsf.

Thao tác là loại thao tác đọc được thực hiện trên Serial Flash.

Các tùy chọn Hoạt động là:

XISF_READ: Đọc bình thường

XISF_FAST_READ: Đọc nhanh

XISF_PAGE_TO_BUF_TRANS: Chuyển từ trang sang bộ đệm

XISF_BUFFER_READ: Đọc bộ đệm

XISF_FAST_BUFFER_READ: Đọc bộ đệm nhanh

XISF OTP_READ: Đọc vùng lập trình một lần (OTP).

OpParamPtr là con trỏ đến biến cấu trúc chứa tham số hoạt động của Hoạt động được chỉ định. Loại tham số này phụ thuộc vào loại Hoạt động sẽ được thực hiện.

Khi chỉ định Đọc bình thường (XISF_READ), Đọc nhanh (XISF_FAST_READ) và Đọc vùng có thể lập trình một lần (XISF_OTP_READ):

- OpParamPtr phải có kiểu struct XIIsf_ReadParam.
- OpParamPtr-> Address là địa chỉ bắt đầu trong Serial Flash.
- OpParamPtr-> ReadPtr là một con trỏ tới bộ nhớ nơi i dữ liệu đọc từ Serial Flash được lưu trữ.
- OpParamPtr-> NumBytes là số byte cần đọc.

Các hoạt động Đọc bình thường và Đọc nhanh được hỗ trợ cho Atmel, Intel và STM Serial Flash.

Hoạt động Đọc OTP chỉ được hỗ trợ trong Intel Serial Flash.

Khi chỉ định chuyển trang sang bộ đệm (XISF_PAGE_TO_BUF_TRANS):

- OpParamPtr phải thuộc loại struct XIIsf_FlashToBufTransferParam.
- OpParamPtr-> BufferNum chỉ định Bộ đệm SRAM bên trong của Flash nối tiếp. Các giá trị hợp lệ là XISF_PAGE_BUFFER1 hoặc XISF_PAGE_BUFFER2. XISF_PAGE_BUFFER2 không hợp lệ trong trự ờng hợp AT45DB011D Flash vì nó chứa một bộ đệm duy nhất.
- OpParamPtr-> Address là địa chỉ bắt đầu trong Serial Flash.

Thao tác này chỉ được hỗ trợ trong Atmel Serial Flash.

XIIsf_Read (tiếp theo)

Thông số	Khi chỉ định Đọc bộ đệm (XISF_BUFFER_READ) và Đọc bộ đệm nhanh (XISF_FAST_BUFFER_READ): <ul style="list-style-type: none"> • OpParamPtr phải có kiểu struct XIIsf_BufferReadParam. • OpParamPtr-> BufferNum chỉ định Bộ đệm SRAM bên trong của Flash nối tiếp. Các giá trị hợp lệ là XISF_PAGE_BUFFER1 hoặc XISF_PAGE_BUFFER2. XISF_PAGE_BUFFER2 không hợp lệ trong trường hợp AT45DB011D Flash vì nó chứa một bộ đệm duy nhất. • OpParamPtr-> ReadPtr là con trỏ đến bộ nhớ nơi dữ liệu đọc từ bộ đệm SRAM sẽ được lưu trữ. • OpParamPtr-> ByteOffset là byte offset trong bộ đệm SRAM từ nơi byte đầu tiên được đọc. • OpParamPtr-> NumBytes là số byte được đọc từ Bộ đệm. <p>Các thao tác này chỉ được hỗ trợ trong Atmel Serial Flash.</p>
Lợi nhuận	XST_SUCCESS khi thành công. XST_FAILURE khi bị lỗi.
Sự mô tả	API này đọc dữ liệu từ Serial Flash. Lưu ý: Ứng dụng phải điền vào các phần tử cấu trúc của đối số thứ ba và chuyên con trả của nó bằng cách nhập nó với con trả void. Dữ liệu hợp lệ có sẵn từ vị trí thứ tư được ReadPtr trả đến cho các hoạt động Đọc bình thường và Đọc bộ đệm. Dữ liệu hợp lệ có sẵn từ vị trí thứ năm được chỉ ra bởi ReadPtr cho các hoạt động Đọc nhanh, Đọc bộ đệm nhanh và Đọc OTP.
Bao gồm	xilisf.h

```
int XIIsf_Write (XIIsf * InstancePtr, XIIsf_WriteOperation
    Hoạt động, void * OpParamPtr)
```

Thông số

InstancePtr là một con trỏ đến thẻ hiện XIIsf.

Thao tác là loại thao tác ghi được thực hiện trên Flash nối tiếp.

Các tùy chọn Hoạt động là:

- XISF_WRITE: Viết bình thường
- XISF_AUTO_PAGE_WRITE: Viết trang tự động
- XISF_BUFFER_WRITE: Ghi vào bộ đệm
- XISF_BUF_TO_PAGE_WRITE_WITH_ERASE: Bộ đệm vào Trang Chuyển bằng Erase
- XISF_BUF_TO_PAGE_WRITE_WITHOUT_ERASE: Bộ đệm vào Chuyển trang mà không cần xóa
- XISF_WRITE_STATUS_REG: Ghi Trạng thái Đăng ký
- XISF OTP_WRITE: Ghi OTP.

OpParamPtr là con trỏ đến một biến cấu trúc chứa các tham số hoạt động của hoạt động được chỉ định.

Loại tham số này phụ thuộc vào giá trị của đối số đầu tiên (Hoạt động).

Khi chỉ định Ghi bình thường (XISF_WRITE):

- OpParamPtr phải có kiểu struct XIIsf_WriteParam.
- OpParamPtr-> Address là địa chỉ bắt đầu trong Serial Flash.
- OpParamPtr-> WritePtr là một con trỏ đến dữ liệu được ghi vào Serial Flash.
- OpParamPtr-> NumBytes là số byte được ghi vào Serial Flash.

Thao tác này được hỗ trợ cho Atmel, Intel và STM Serial Flash.

Khi chỉ định Ghi trang tự động
(XISF_AUTO_PAGE_WRITE):

- OpParamPtr phải là biến số nguyên không dấu 32 bit. Đây là địa chỉ của số trang trong Serial Flash sẽ được làm mới.

Thao tác này chỉ được hỗ trợ trong Atmel Serial Flash.

Khi chỉ định Ghi bộ đệm (XISF_BUFFER_WRITE): • OpParamPtr phải thuộc loại struct XIIsf_BufferWriteParam.

- OpParamPtr-> BufferNum chỉ định Bộ đệm SRAM bên trong của Flash nối tiếp. Các giá trị hợp lệ là XISF_PAGE_BUFFER1 hoặc XISF_PAGE_BUFFER2. XISF_PAGE_BUFFER2 không hợp lệ trong trự ờng hợp AT45DB011D Flash vì nó chứa một bộ đệm duy nhất.
- OpParamPtr-> WritePtr là một con trỏ đến dữ liệu được ghi vào Bộ đệm SRAM Flash nối tiếp.
- OpParamPtr-> ByteOffset là byte offset trong bộ đệm từ nơi dữ liệu sẽ được ghi.
- OpParamPtr-> NumBytes là số byte được ghi vào bộ đệm.

Thao tác này chỉ được hỗ trợ cho Atmel Serial Flash.

XIif_Write (tiếp theo)

Thông số

Khi chỉ định Bộ đệm vào Bộ nhớ Ghi bằng Xóa (XISF_BUF_TO_PAGE_WRITE_WITH_ERASE) hoặc Bộ đệm vào Ghi vào bộ nhớ mà không cần xóa (XISF_BUF_TO_PAGE_WRITE_WITHOUT_ERASE):

- OpParamPtr phải thuộc loại struct XIif_BufferToFlashWriteParam.
- OpParamPtr-> BufferNum chỉ định Bộ đệm SRAM bên trong của Flash nối tiếp. Các giá trị hợp lệ là XISF_PAGE_BUFFER1 hoặc XISF_PAGE_BUFFER2. XISF_PAGE_BUFFER2 không hợp lệ trong trường hợp AT45DB011D Flash vì nó chứa một bộ đệm duy nhất.
- OpParamPtr-> Address là địa chỉ bắt đầu trong bộ nhớ Serial Flash từ nơ i dữ liệu sẽ được ghi.

Các thao tác này chỉ được hỗ trợ trong Atmel Serial Flash.

Khi chỉ định Thanh ghi trạng thái ghi (XISF_WRITE_STATUS_REG), OpParamPtr phải là một biến số nguyên không dấu 8 bit. Đây là giá trị được ghi vào Thanh ghi trạng thái.

Thao tác này chỉ được hỗ trợ trong Intel và STM Serial Flash.

Khi chỉ định Ghi vùng có thể lập trình một lần (XISF_OTP_WRITE):

- OpParamPtr phải có kiểu struct XIif_WriteParam.
- OpParamPtr-> Địa chỉ là địa chỉ trong SRAM Buffer của Flash nối tiếp mà dữ liệu sẽ được ghi vào.
- OpParamPtr-> WritePtr là một con trỏ tới dữ liệu được ghi vào Serial Flash.
- OpParamPtr-> NumBytes phải được đặt thành 1 khi thực hiện Hoạt động ghi OTP.

Thao tác này chỉ được hỗ trợ trong Intel Serial Flash.

Lợi nhuận

XST_SUCCESS khi thành công.

XST_FAILURE khi bị lỗi.

Sự mô tả

API này ghi dữ liệu vào Serial Flash.

Lưu ý: Ứng dụng phải điền vào các phần tử cấu trúc của đối số thứ ba và chuyển con trỏ của nó bằng cách nhập nó với con trỏ void.

Đối với Intel và STM Serial Flash, ứng dụng người dùng phải gọi API XIif_WriteEnable () bằng cách chuyển XISF_WRITE_ENABLE làm đối số trước khi gọi API XIif_Write ().

Bao gồm

xilisf.h

```
int XIIsf_Erase (XIIsf * InstancePtr, XIIsf_EraseOperation  
    Hoạt động, Địa chỉ u32)
```

Thông số

InstancePtr là một con trỏ đến thẻ hiện XIIsf.

Thao tác là loại thao tác Xóa được thực hiện trên Flash nối tiếp.

Các hoạt động khác nhau là

- XISF_PAGE_ERASE: Xóa trang
- XISF_BLOCK_ERASE: Xóa khối
- XISF_SECTOR_ERASE: Xóa khu vực
- XISF_BULK_ERASE: Xóa hàng loạt

Địa chỉ là địa chỉ của Trang / Khối / Ngành cần xóa.

Địa chỉ có thể là địa chỉ Trang, địa chỉ Khối hoặc địa chỉ Khu vực dựa trên thao tác Xóa sẽ được thực hiện.

Lợi nhuận

XST_SUCCESS khi thành công.

XST_FAILURE khi bị lỗi.

Sự mô tả

API này xóa nội dung của bộ nhớ được chỉ định trong Serial Flash.

Lưu ý: Các byte bị xóa sẽ đọc là 0xFF.

Đối với Intel và STM Serial Flash, ứng dụng người dùng phải gọi API XIIsf_WriteEnable () bằng cách chuyển XISF_WRITE_ENABLE làm đối số trước khi gọi API XIIsf_Erase ().

Atmel Serial Flash hỗ trợ Các thao tác xóa trang / khối / khu vực.

Hỗ trợ Intel Serial Flash hoạt động Sector / Block / Bulk Erase.

Hỗ trợ STM Serial Flash Hoạt động xóa hàng loạt / Khu vực.

Bao gồm

xilisf.h

```
int XIIsf_SectorProtect (XIIsf * InstancePtr, XIIsf_SpOperation
    Hoạt động, u8 * BufferPtr)
```

Thông số

InstancePtr là một con trỏ đến thẻ hiện XIIsf.

Hoạt động là loại hoạt động Bảo vệ khu vực được thực hiện trên Flash nối tiếp.

Các tùy chọn Hoạt động là

- XISF_SPR_READ: Đọc thanh ghi bảo vệ khu vực
- XISF_SPR_WRITE: Ghi sổ đăng ký bảo vệ khu vực
- XISF_SPR_ERASE: Xóa sổ đăng ký bảo vệ khu vực
- XISF_SP_ENABLE: Bật Bảo vệ khu vực
- XISF_SP_DISABLE: Tắt bảo vệ khu vực

BufferPtr là một con trỏ tới bộ nhớ nơi nội dung SPR được đọc / ghi từ đó. Đối số này có thể là NULL nếu Thao tác là SpErase, SpEnable và SpDisable.

Lợi nhuận

XST_SUCCESS khi thành công.

XST_FAILURE khi bị lỗi.

Sự mô tả

API này được sử dụng để thực hiện các hoạt động liên quan đến Bảo vệ khu vực.

Lưu ý: Nội dung SPR được lưu trữ tại vị trí thứ tự được chỉ ra bởi BufferPtr khi thực hiện thao tác XISF_SPR_READ.

Đối với Intel và STM Serial Flash, ứng dụng người dùng phải gọi API XIIsf_WriteEnable () bằng cách chuyển XISF_WRITE_ENABLE làm đối số, trước khi gọi API XIIsf_SectorProtect (), đối với hoạt động Việt (XISF_SPR_WRITE) của Sector Protect Register.

Atmel Flash hỗ trợ tắt cả các hoạt động Sector Protect này.

Intel và STM Flash chỉ hỗ trợ Sector Protect Read và Sector Protect Write các hoạt động.

Bao gồm

xilisf.h

```
int XIIsf_WriteEnable (XIIsf * InstancePtr, u8 WriteEnable)
```

Thông số

InstancePtr là một con trỏ đến thẻ hiện XIIsf.

WriteEnable chỉ định Bật (XISF_CMD_ENABLE_WRITE) hay Tắt (XISF_CMD_DISABLE_WRITE) ghi vào Flash nối tiếp.

Lợi nhuận

XST_SUCCESS khi thành công.

XST_FAILURE khi bị lỗi.

Sự mô tả

API này Bật / Tắt ghi vào Flash nối tiếp Intel và STM.

Lưu ý: API này chỉ hoạt động với Intel và STM Serial Flash. Nếu API này được gọi cho Atmel Flash, thì XST_FAILURE sẽ được trả về.

Bao gồm

xilisf.h

```
int XIssf_Ioctl (XIssf * InstancePtr, XIssf_IoctlOperation
    Hoạt động)
```

Thông số

InstancePtr là một con trỏ đến thẻ hiện XIssf.

Hoạt động là loại hoạt động Điều khiển được thực hiện trên Đèn flash nối tiếp.

Các tùy chọn Hoạt động kiểm soát là:

- XISF_RELEASE_DPD: Giải phóng khỏi Deep Power Down (DPD)
Cách thức
- XISF_ENTER_DPD: Vào Chế độ DPD
- XISF_CLEAR_SR_FAIL_FLAGS: Xóa Đăng ký Trạng thái Không thành công Cờ.

Lợi nhuận

XST_SUCCESS khi thành công.

XST_FAILURE khi bị lỗi.

Sự mô tả

API này định cấu hình và điều khiển Intel và STM Serial Flash.

Lưu ý: Atmel Serial Flash không hỗ trợ bất kỳ thao tác nào trong số này.

Hỗ trợ Intel Serial Flash Nhập / Phát hành từ Chế độ DPD và Xóa trạng thái Đăng ký Cờ lỗi.

Hỗ trợ STM Serial Flash Nhập / Phát hành từ Chế độ DPD.

Bao gồm

xilisf.h

Libgen Tùy biến

Thư viện LibXil Isf có thể được tích hợp với hệ thống bằng cách sử dụng đoạn mã sau trong Tệp Thông số kỹ thuật phần mềm vi xử lý (MSS) cho Xilinx Flash trong hệ thống hoặc Atmel bên ngoài Đèn flash nối tiếp:

```
BẮT ĐẦU THƯ VIỆN
tham số LIBRARY_NAME = xilisf
tham số LIBRARY_VER = 1.00.a
tham số PROC_INSTANCE = microblaze_0
tham số serial_flash_family = 1
CHẤM DỨT
```

Lưu ý: Giá trị serial_flash_family tham số phải được đặt như sau:

- 1 để hỗ trợ Flash trong hệ thống Xilinx hoặc Flash nối tiếp Atmel
- 2 để hỗ trợ Intel Serial Flash
- 3 để hỗ trợ STM Serial Flash

Thêm vào Tài nguyên

- Hướng dẫn sử dụng Flash trong hệ thống Spartan-3AN (UG333)
 http://www.xilinx.com/support/documentation/user_guides/ug333.pdf
- Bộ nhớ Flash nối tiếp Atmel (AT45XXXD)
 http://www.atmel.com/dyn/products/devices.asp?family_id=616#1802
- Bộ nhớ flash nối tiếp Intel (S33)
 http://www.intel.com/design/f1comp/prodbref/s33.htm?iid=ipp_embed+flash_s33&
- Bộ nhớ Flash nối tiếp STM
 [\(M25PXX\)](http://www.st.com/stoneline/products/liteosystem/ds/10027.pdf)
<http://www.st.com/stoneline/products/liteosystem/ds/10027.pdf>



Gói hỗ trợ bo mạch tự động tạo ra Tornado 2.x (VxWorks 5.x)

Bản tóm tắt

Tài liệu này mô tả việc tạo tự động Gói hỗ trợ bo mạch aTornado 2.x (BSP) sử dụng Bộ phát triển phần mềm Xilinx (SDK) (1). Tài liệu gồm các phần sau.

- "Tổng quan"
- "Tạo ra Tornado BSP"
- "Cơ n lốc xoáy 2.x BSP"
- "Khởi động VxWorks"

Tổng quan

Một trong những hoạt động phát triển hệ thống nhúng chính là phát triển BSP. Việc tạo ra một BSP có thể là một quá trình kéo dài và tẻ nhạt phải phát sinh mỗi khi tổ hợp bộ vi xử lý (bộ vi xử lý cộng với các thiết bị ngoại vi liên quan) thay đổi. Trong khi việc quản lý những thay đổi này áp dụng cho bất kỳ dự án dựa trên bộ vi xử lý nào, những thay đổi có thể xảy ra nhanh hơn bao giờ hết với sự ra đời của phần cứng Hệ thống trên chip (SoC) có thể lập trình được.

Tài liệu này mô tả việc tạo tự động Tornado 2.x BSP (dành cho Tornado 2.0.2, 2.2 và 2.2.1) cho bộ vi xử lý IBM PowerPC™ 405 và các thiết bị ngoại vi của nó như được định nghĩa trong Xilinx® FPGA. BSP được tạo tự động cho phép các nhà thiết kế hệ thống nhúng:

- Giảm đáng kể chu kỳ phát triển (giảm thời gian đưa ra thị trường)
- Tạo BSP phù hợp với phần cứng và ứng dụng (BSP tùy chỉnh)
- Loại bỏ lỗi thiết kế BSP (được tạo tự động dựa trên các thành phần được chứng nhận)
- Cho phép phát triển phần mềm ứng dụng (không phải đợi phát triển BSP)

Tornado 2.x BSP được tạo từ SDK, một IDE được phân phối như một phần của Bộ phát triển nhúng Xilinx (EDK) hoặc có sẵn riêng từ Xilinx. SDK được sử dụng để tạo các ứng dụng phần mềm cho các hệ thống nhúng trong Xilinx FPGA. Tornado BSP chứa tất cả phần mềm hỗ trợ cần thiết cho một hệ thống, bao gồm mã khởi động, trình điều khiển thiết bị và khởi tạo RTOS. BSP được tùy chỉnh dựa trên các thiết bị ngoại vi được người dùng chọn và cấu hình cho hệ thống nhúng dựa trên FPGA.

1. Xilinx SDK được sử dụng làm môi trường phát triển phần mềm chính cho Bộ phát triển nhúng Xilinx (EDK) người dùng của EDK 11.1i. Khả năng phát triển phần mềm của Xilinx Platform Studio (XPS) hiện không được dùng nữa và sẽ bị xóa khỏi XPS trong các bản phát hành sau. Các luồng được mô tả trong tài liệu này liên quan đến SDK, mặc dù chúng vẫn có thể áp dụng chung cho XPS trong khi các tính năng đó vẫn còn trong công cụ

© 2005-2009 Xilinx, Inc. XILINX, biểu trưng Xilinx, Virtex, Spartan, ISE và các nhãn hiệu được chỉ định khác có trong tài liệu này là nhãn hiệu của Xilinx tại Hoa Kỳ và các quốc gia khác. Tất cả các nhãn hiệu khác là tài sản của chủ sở hữu tương ứng của họ.

Xilinx sẽ tiết lộ hướng dẫn sử dụng, hướng dẫn sử dụng, ghi chú phát hành và / hoặc thông số kỹ thuật này ("Tài liệu") cho bạn chỉ để sử dụng trong việc phát triển các thiết kế để vận hành với các thiết bị phần cứng của Xilinx. Bạn không được phép sao chép, phân phối, tái xuất bản, tái xuất bản, hiển thị, đăng hoặc truyền Tài liệu dưới bất kỳ hình thức nào hoặc bằng bất kỳ phương tiện nào bao gồm nhưng không giới hạn ở Xilinx. Xilinx từ chối rõ ràng bất kỳ trách nhiệm pháp lý nào phát sinh từ việc bạn sử dụng Tài liệu. Xilinx bảo lưu quyền, theo quyết định riêng của mình, thay đổi Tài liệu mà không cần thông báo bất kỳ lúc nào.

Xilinx không có nghĩa vụ phải sửa bất kỳ lỗi nào có trong Tài liệu hoặc thông báo cho bạn về bất kỳ chính sửa hoặc cập nhật nào. Xilinx từ chối rõ ràng bất kỳ trách nhiệm pháp lý nào liên quan đến hỗ trợ kỹ thuật hoặc hỗ trợ có thể được cung cấp cho bạn liên quan đến Thông tin.

TÀI LIỆU ĐƯỢC CÔNG BỐ CHO BẠN "NGUYÊN TRẠNG" KHÔNG ĐƯỢC BẢO HÀNH BẤT KỲ HÌNH THỨC NÀO. XILINX KHÔNG CÓ BẢO ĐÀM KHÁC, DÙ THÌ HIỆN RỘ RẰNG, NGƯỜI DÙNG TRUYỀN NGHĨA, LIÊN QUAN ĐẾN TÀI LIỆU, BAO GỒM BẤT KỲ BẢO ĐÀM NÀO VỀ TÍNH KHÁ NẮNG, PHÙ HỢP VỚI MỤC ĐÍCH CỤ THỂ HOẶC KHÔNG ĐÀM BẢO QUYỀN CỦA BÊN THỨ BA. TRONG MỌI SỰ KIỆN XILINX SẼ CHỊU TRÁCH NHIỆM PHÁP LÝ ĐỐI VỚI BẤT KỲ THIỆT HẠI HẬU QUẢ, CHỈ ĐỊNH, BẤT CỨ, ĐẶC BIỆT HOẶC BẤT CỨ SỰ CỐ NÀO, BAO GỒM BẤT CỨ MẤT DỮ LIỆU HOẶC MẤT LỢI NHUẬN NÀO PHÁT SINH TỪ VIỆC BẠN SỬ DỤNG TÀI LIỆU

Các nhà thiết kế BSP có kinh nghiệm sẽ gặp ít khó khăn khi tích hợp BSP đã tạo vào hệ thống mục tiêu của họ. Người mới có thể gặp khó khăn vì mặc dù SDK có thể tạo BSP hoạt động cho một tập hợp phần cứng IP nhất định, nhưng sẽ luôn có một số câu hỏi và điều chỉnh bổ sung cần thiết để đạt được hiệu suất tốt nhất từ hệ thống mục tiêu. Người dùng rất nên có trong tay Bộ công cụ dành cho nhà phát triển Tornado BSP cho Hume dãy sử dụng VxWorks và Hume dãy dành cho người lập trình VxWorks có sẵn từ Wind River. Wind River cũng cung cấp các lớp học về thiết kế BSP với một khoản chi phí bổ sung.

Yêu cầu

Người dùng phải cài đặt bộ phát triển Tornado 2.0.2 hoặc Tornado 2.2 hoặc 2.2.1 trên máy chủ của họ. Người dùng nên sử dụng Tornado 2.2.1 do có nhiều bản sửa lỗi và cài đặt dễ dàng hơn. Máy tính chủ không bắt buộc phải cài đặt cả Xilinx SDK và Tornado vì SDK tạo ra các BSP có thể di dời được, được biên dịch và định cấu hình bên ngoài môi trường SDK.

Tornado 2.0.2

Các bản vá bắt buộc có thể tìm thấy tại trang web hỗ trợ kỹ thuật LưỚt ván buồm của Wind River:

- Bản vá tích lũy SPR67953
- Hỗ trợ hệ thống tệp DOS DosFs 2.0

Lưu ý rằng một số bản vá lỗi này có thể không còn nữa. Liên hệ với Wind River FAE của bạn để biết thêm thông tin.

Lốc xoáy 2.2

Đây là bản phát hành Tornado cập nhật với nhiều bản sửa lỗi cho môi trường phát triển. Không cần bản vá.

BUFFER ĐƠN

Trình gỡ lỗi SingleStep phải biết bộ xử lý PowerPC® 405 bên trong Virtex-II Pro hoặc Virtex-4 FPGA. Trình gỡ lỗi này hoạt động cùng với nhóm gỡ lỗi VisionProbe.

Định nghĩa thư viện bộ vi xử lý

SDK hỗ trợ giao diện trình cắm thêm cho các thư viện và hệ điều hành của bên thứ 3 thông qua giao diện Định nghĩa Thư viện Bộ vi xử lý (MLD). Điều này cho phép các nhà cung cấp bên thứ 3 cung cấp phần mềm của họ cho người dùng SDK và cung cấp cho các nhà cung cấp phương tiện để điều chỉnh thư viện hoặc BSP của họ cho phù hợp với hệ thống nhúng dựa trên FPGA. Vì hệ thống có thể thay đổi dễ dàng, khả năng này là tối quan trọng để hỗ trợ đúng cách các hệ thống nhúng trong FPGA.

Xilinx phát triển và duy trì VxWorks 5.x MLD trong các bản phát hành SDK của mình. MLD được sử dụng để tự động tạo Tornado 2.x BSP.

Phương pháp Tiếp cận Dựa trên Mẫu

Một tập hợp các tệp mẫu Tornado BSP được phát hành cùng với SDK. Các tệp mẫu này được sử dụng trong quá trình tạo BSP tự động và các sửa đổi thích hợp được thực hiện dựa trên cấu trúc của hệ thống nhúng dựa trên FPGA.

Nếu người dùng chọn không tự động tạo BSP, các tệp mẫu này có thể được sử dụng làm tài liệu tham khảo để xây dựng BSP từ đầu.

Trình điều khiển thiết bị

Một tập hợp các tệp nguồn trình điều khiển thiết bị được phát hành cùng với SDK và nằm trong thư mục cài đặt. Trong quá trình tạo BSP tùy chỉnh, mã nguồn trình điều khiển thiết bị được sao chép từ thư mục cài đặt này sang thư mục BSP. Chỉ mã nguồn liên quan đến các thiết bị được tích hợp trong hệ thống nhúng dựa trên FPGA mới được sao chép. Bản sao này cung cấp cho người dùng một BSP độc lập, độc lập

thư mục mà người dùng có thể sửa đổi nếu cần và / hoặc di dời nếu cần. Nếu người dùng thực hiện các thay đổi đối với mã nguồn trình điều khiển thiết bị cho BSP này và đôi khi muốn khôi phục những thay đổi đó, người dùng có thể sử dụng các công cụ SDK để tạo lại BSP. Các tệp nguồn của trình điều khiển thiết bị sau đó được mở lại từ thư mục cài đặt vào BSP.

Tạo ra Tornado BSP

Sử dụng SDK

SDK có sẵn dưới dạng công cụ được cài đặt riêng hoặc trong EDK và là môi trường phát triển phần mềm để phát triển phần mềm nhưng xung quanh hệ thống nhưng dựa trên Xilinx PowerPC 405 hoặc MicroBlaze™. Phần này mô tả các bước cần thiết để tạo BSP Tornado 2.x bằng SDK.

Các bước này có thể áp dụng khi sử dụng các công cụ Xilinx 11.1i trở lên.

Giả định rằng một thiết kế phần cứng hợp lệ đã được tạo và xuất sang SDK, và SDK đã được mở và trả đến thiết kế phần cứng.

1. Sử dụng File-> New, tạo một dự án Gói hỗ trợ hội đồng quản trị mới. Trong hộp thoại, nhập tên dự án và chọn vxworks5_5 làm Loại gói hỗ trợ hội đồng quản trị. Lưu ý rằng SDK có thể quản lý nhiều dự án thuộc các loại BSP khác nhau.

Các bước còn lại liên quan đến hộp thoại Tools-> Board Support Package Settings ..., hộp thoại này sẽ tự động được hiển thị sau bước trên.

2. Định cấu hình thiết bị giao diện điều khiển VxWorks

Nếu một thiết bị nối tiếp như Uart được dự định sử dụng làm bảng điều khiển VxWorks, hãy chọn hoặc nhập tên phiên bản của thiết bị nối tiếp làm thiết bị ngoại vi STDIN / STDOUT trong hộp thoại Cài đặt Gói Hỗ trợ Bảng. Điều quan trọng là phải nhập cùng một thiết bị cho cả STDIN và STDOUT.

Hiện tại, chỉ có các thiết bị Uart 16550/16450 và UartLite được hỗ trợ làm thiết bị VxWorks console.

3. Tích hợp trình điều khiển thiết bị

một. Kết nối với VxWorks

Có một hộp thoại kết nối_periph có sẵn trong hộp thoại Thiết đặt gói hỗ trợ hội đồng quản trị Các thiết bị ngoại vi đã được điều chỉnh sẵn để thuận tiện cho người dùng. Sử dụng hộp thoại này để sửa đổi các thiết bị ngoại vi đó để được tích hợp chặt chẽ với Hệ điều hành, bao gồm thiết bị đã được chọn làm thiết bị ngoại vi STDIN / STDOUT. Xem phần "[Tích hợp thiết bị](#)" để biết thêm chi tiết về tích hợp chặt chẽ các thiết bị.

4. Tạo Tornado 2.x BSP

Bấm OK trên hộp thoại Cài đặt Gói Hỗ trợ Ban ... để tạo BSP. Đầu ra của lệnh gọi này được hiển thị trong cửa sổ bảng điều khiển SDK. Sau khi hoàn tất, Tornado BSP kết quả sẽ tồn tại trong không gian làm việc SDK của bạn, dưới tên thư mục dự án bạn đã tạo ở bước 1 ở trên, trong thư mục con phiên bản PowerPC 405. Ví dụ: nếu trong thiết kế phần cứng, người dùng đã đặt tên cho phiên bản PowerPC 405 là ppc405_0, BSP sẽ nằm tại <SDK workspace> / <SDK project name> / ppc405_0 / bsp_ppc405_0.

Sao lưu

Nếu vị trí thư mục của BSP chứa các tệp hiện có, các tệp này được sao chép vào thư mục sao lưu trước khi bị ghi đè. Điều này ngăn ngừa việc vô tình làm mất các thay đổi do người dùng thực hiện đối với tệp nguồn BSP. Thư mục sao lưu nằm trong thư mục BSP và được đặt tên là backup <timestamp>, trong đó <timestamp> đại diện cho ngày và giờ hiện tại. Lưu ý rằng BSP do SDK tạo ra có thể định vị lại, vì vậy, bạn nên di chuyển BSP từ thư mục dự án SDK sang thư mục phát triển BSP thích hợp ngay khi nền tảng phần cứng ổn định.

Tornado 2.x BSP

Phần này giả định người đọc đã quen thuộc với Tornado 2.0.2 hoặc 2.2.x IDE của Wind River. Nó mô tả đầu ra Tornado BSP của SDK.

BSP được tạo tự động đưa vào tích hợp vào Tornado IDE và cơ sở Dự án. BSP có thể được biên dịch từ dòng lệnh bằng cách sử dụng các công cụ tạo Tornado hoặc từ cơ sở Dự án Tornado (còn được gọi là Tornado IDE). Khi BSP đã được tạo, người dùng có thể chỉ cần gõ make vxWorks từ dòng lệnh để biên dịch hình ảnh RAM có thể khởi động. Điều này giả định rằng môi trường Tornado đã được thiết lập trước đó, có thể được thực hiện thông qua dòng lệnh sử dụng host / x86-

tập lệnh win32 / bin / torVars.bat nếu trên nền tảng Windows. Nếu sử dụng cơ sở Dự án Tornado, người dùng có thể tạo một dự án dựa trên BSP mới được tạo, sau đó sử dụng môi trường xây dựng đưa cung cấp thông qua IDE để biên dịch BSP.

Trong Tornado 2.2.x, trình biên dịch diab được hỗ trợ ngoài trình biên dịch gnu . Tornado BSP được tạo bởi SDK có một Makefile có thể được sửa đổi bởi người dùng dòng lệnh để sử dụng trình biên dịch diab thay vì trình biên dịch gnu. Tìm biến make có tên TOOLS và đặt giá trị thành "diab" thay vì "gnu". Nếu sử dụng cơ sở Dự án Tornado, người dùng có thể chọn công cụ mong muốn khi dự án được tạo lần đầu tiên.

Tổ chức tài xế

Phần này thảo luận ngắn gọn về cách trình điều khiển Xilinx được biên dịch và liên kết và cuối cùng được sử dụng bởi các câu lệnh Tornado để đưa vào hình ảnh VxWorks.

Trình điều khiển Xilinx được triển khai bằng "C" và có thể được phân phối giữa một số tệp nguồn không giống như trình điều khiển VxWorks truyền thống, bao gồm các tệp triển khai và tiêu đề "C" duy nhất.

Có tám phần cho trình điều khiển Xilinx:

- Bao gồm nguồn trình điều khiển.
- Triển khai độc lập hệ điều hành
- Triển khai phụ thuộc vào hệ điều hành (tùy chọn).

"Bao gồm nguồn trình điều khiển" để cập đến cách trình điều khiển Xilinx được biên dịch. Đối với mọi trình điều khiển, có một tệp có tên là <procname>_drv_<dev>_<version>.c. # Tệp này bao gồm mỗi (các) tệp nguồn trình điều khiển (*.c) cho thiết bị nhất định.

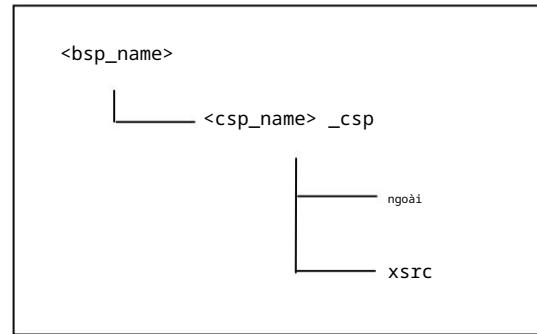
Quá trình này tương tự như cách nguồn sysLib.c # include của VxWorks dành cho các trình điều khiển được cung cấp bởi Wind River. Lý do tại sao các tệp Xilinx không chỉ đơn giản được đưa vào sysLib.c như phần còn lại của các trình điều khiển là do xung động không gian tên và các vấn đề về khả năng bảo trì. Nêu tắt cả các tệp Xilinx là một phần của một đơn vị biên dịch duy nhất, thì các hàm và dữ liệu tĩnh không còn là riêng tư nữa. Điều này đặt ra những hạn chế đối với trình điều khiển thiết bị và sẽ phủ nhận tính độc lập của hệ điều hành của chúng.

Phần độc lập hệ điều hành của trình điều khiển được thiết kế để sử dụng với bất kỳ hệ điều hành hoặc bất kỳ bộ xử lý nào. Nó cung cấp một API sử dụng chức năng của phần cứng bên dưới. Phần phụ thuộc vào hệ điều hành của trình điều khiển sẽ điều chỉnh trình điều khiển để sử dụng với VxWorks. Các ví dụ như vậy là trình điều khiển SIO cho các cổng nối tiếp hoặc trình điều khiển END cho bộ điều hợp ethernet. Không phải tắt cả các trình điều khiển đều yêu cầu trình điều khiển phụ thuộc vào hệ điều hành, cũng như không bắt buộc phải bao gồm phần phụ thuộc vào hệ điều hành của trình điều khiển trong bản dựng VxWorks.

Vị trí trình điều khiển thiết bị

BSP được tạo tự động giống với hầu hết các BSP Tornado khác ngoại trừ việc đặt mã trình điều khiển thiết bị. Mã trình điều khiển thiết bị có sẵn được phân phối với Tornado IDE thư mục target / src / drv trong thư mục phân phối Tornado. Mã trình điều khiển thiết bị cho BSP được tạo tự động nằm trong chính thư mục BSP. Sự sai lệch nhỏ này là do bản chất động của hệ thống nhưng dựa trên FPGA. Vì hệ thống nhưng dựa trên FPGA có thể được lập trình lại với IP mới hoặc thay đổi, cấu hình trình điều khiển thiết bị có thể thay đổi, yêu cầu vị trí năng động hơn của các tệp nguồn trình điều khiển thiết bị.

Cây thư mục cho BSP được tạo tự động đưa ra thể hiện trong [Hình 1-1](#).



Hình 1-1: Vị trí thư mục trình điều khiển

Thư mục cấp cao nhất được đặt tên theo tên của phiên bản bộ xử lý trong dự án thiết kế phần cứng. Các tệp nguồn BSP tùy chỉnh nằm trong thư mục này. Có một thư mục con trong thư mục BSP được đặt tên theo phiên bản bộ xử lý với `_drv_csp` làm hậu tố. Thư mục trình điều khiển chứa hai thư mục con. Thư mục `xsrc` chứa tất cả các tệp nguồn liên quan đến trình điều khiển thiết bị. Thư mục `out` được tạo trong quá trình xây dựng và chỉ tồn tại nếu xây dựng từ dòng lệnh.

Nó chứa các tệp được tạo trong quá trình biên dịch hoặc xây dựng (ví dụ: tệp `.o` cho mỗi tệp nguồn trình điều khiển). Nếu xây dựng từ cơ sở Dự án Tornado, các tệp được tạo trong quá trình xây dựng nằm ở `$PRJ_DIR / $BUILD_SPEC`.

Cấu hình

BSP do SDK tạo ra được định cấu hình giống như bất kỳ BSP Tornado 2.x nào khác. Có rất ít khả năng cấu hình đối với trình điều khiển Xilinx vì phần cứng IP đã được cấu hình trước. Cấu hình duy nhất có sẵn nói chung là liệu trình điều khiển có được bao gồm trong bản dựng VxWorks hay không. Làm thế nào để bao gồm / loại trừ trình điều khiển phụ thuộc vào việc cơ sở Dự án hoặc phương thức dòng lệnh đang được sử dụng để thực hiện các hoạt động cấu hình.

Lưu ý rằng chỉ cần bao gồm trình điều khiển thiết bị Xilinx không có nghĩa là trình điều khiển đó sẽ tự động được sử dụng. Hầu hết các trình điều khiển với bộ điều hợp VxWorks đều có mã khởi tạo. Trong một số trường hợp, người dùng có thể được yêu cầu thêm các lệnh gọi hàm khởi tạo trình điều khiển thích hợp vào BSP.

Khi sử dụng SDK để tạo BSP, các tệp BSP kết quả có thể chứa các nhận xét “VIỆC CẦN LÀM”. Những nhận xét này, nhiều trong số đó bắt nguồn từ mẫu PowerPC™ 405 BSP do Wind River cung cấp, cung cấp gợi ý về những gì người dùng phải cung cấp để định cấu hình BSP cho bảng mục tiêu của họ. Bộ công cụ dành cho nhà phát triển Tornado BSP cho Hướng dẫn sử dụng VxWorks và Hướng dẫn cho người lập trình VxWorks là những tài nguyên rất tốt cho cấu hình BSP.

Bao gồm / Loại trừ Trình điều khiển Dòng lệnh

Trong BSP, một tập hợp các hằng số (một cho mỗi trình điều khiển) được xác định trong `<procname>_drv_config.h` và tuân theo định dạng:

```
#define INCLUDE_ <XDRIVER>
```

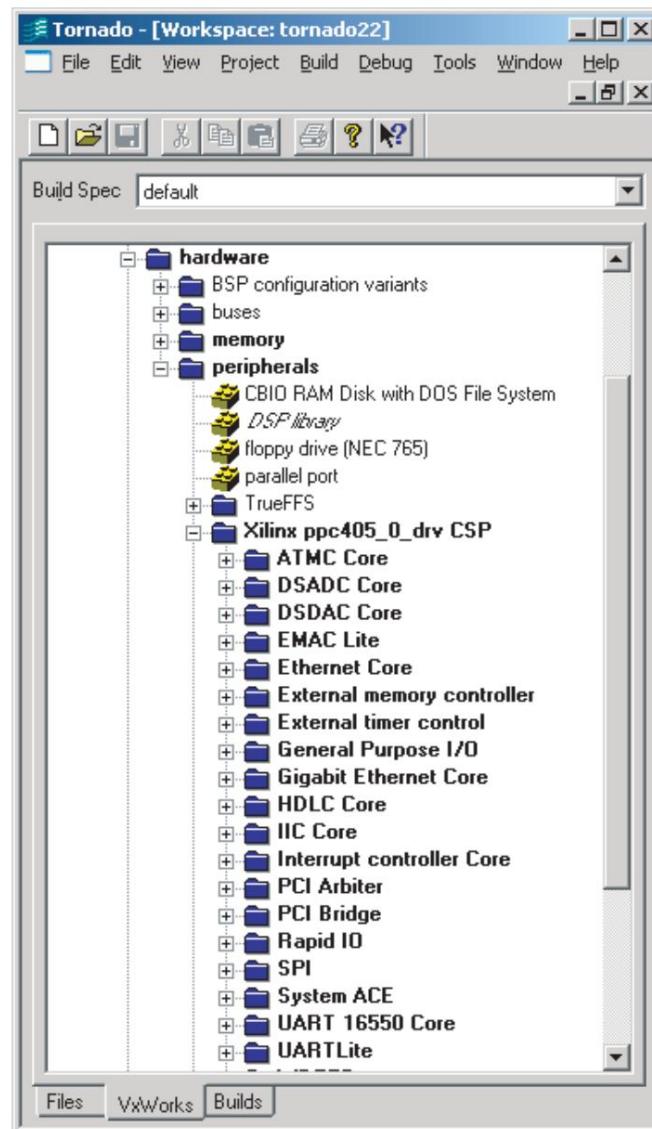
Tệp này được bao gồm gần đầu config.h. Theo mặc định, tất cả các trình điều khiển được bao gồm trong bản dựng. Để loại trừ trình điều khiển, hãy thêm dòng sau vào config.h sau khi bao gồm tệp tiêu đề `<procname>_drv_config.h`.

```
#undef INCLUDE_ <XDRIVER>
```

Điều này sẽ ngăn trình điều khiển được biên dịch và liên kết với bản dựng. Để cài đặt lại trình điều khiển, hãy xóa dòng `#undef` khỏi config.h. Một số cần thận là cần thiết cho một số trình điều khiển. Ví dụ, Ethernet có thể yêu cầu phải có trình điều khiển DMA. Việc hủy xác định trình điều khiển DMA sẽ khiến bản dựng không thành công.

Bao gồm / Loại trừ Trình điều khiển Cơ sở Dự án

Tệp 50 <csp_name>.cdf nằm trong thư mục BSP và được điều chỉnh trong quá trình tạo BSP. Tệp này tích hợp trình điều khiển thiết bị Xilinx vào Tornado IDE. Trình điều khiển thiết bị Xilinx được nối vào IDE tại thư mục con phần cứng / thiết bị ngoại vi của tab VxWorks. Dưới đây là các thư mục trình điều khiển thiết bị riêng lẻ. [Hình 1-2](#) cho thấy một ví dụ về GUI với trình điều khiển Xilinx. Để thêm / xóa trình điều khiển Xilinx, chỉ cần bao gồm hoặc loại trừ các thành phần trình điều khiển giống như bạn làm với bất kỳ thành phần VxWorks nào khác.



Hình 1-2: Dự án Tornado 2.x IDE - VxWorks

Lưu ý rằng bất kỳ cấu hình nào đã được chỉ định trong <procname>_drv_config.h và config.h sẽ bị ghi đè bởi cơ sở dự án

Xây dựng VxWorks

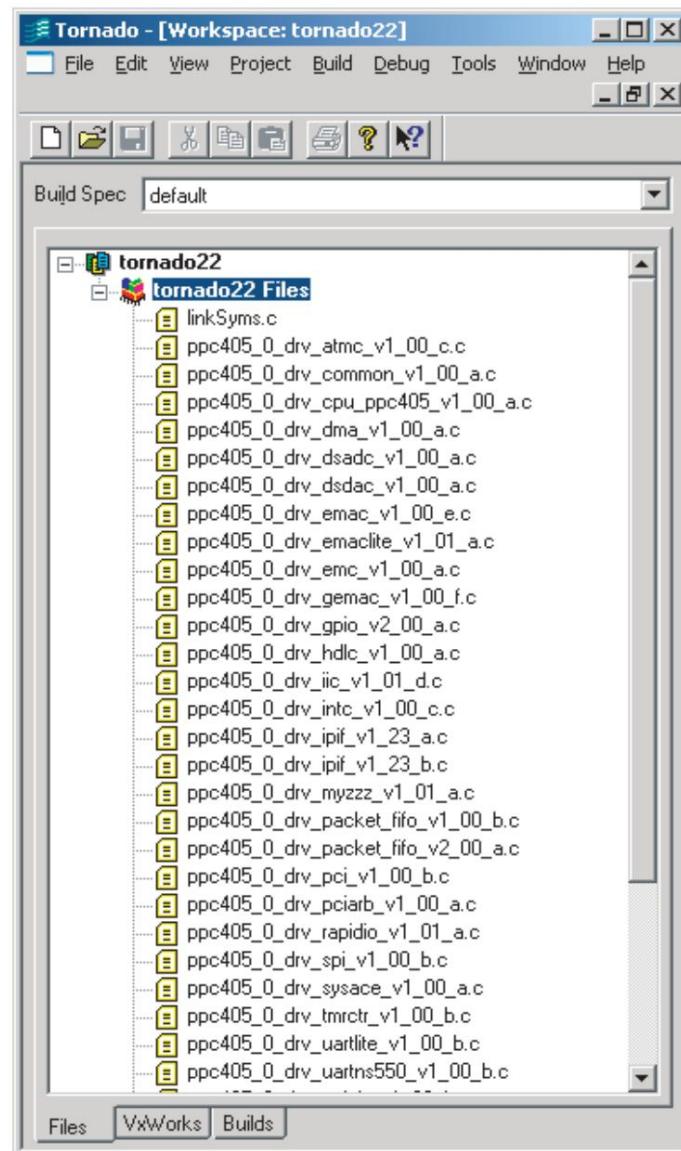
Các BSP được tạo tự động tuân theo các quy ước tiêu chuẩn của Tornado khi tạo hình ảnh VxWorks. Tham khảo tài liệu Tornado về cách tạo hình ảnh VxWorks.

Tiện ích mở rộng bản dựng BSP dòng lệnh

Các trình điều khiển Xilinx được biên dịch / liên kết với cùng một chuỗi công cụ mà VxWorks được xây dựng. Các bổ sung nhỏ cho Makefile được yêu cầu để giúp Tornado tìm thấy vị trí của các tệp mã nguồn trình điều khiển.

Project BSP Build Extensions

Tab Tệp của Tornado Project IDE cũng sẽ hiển thị một số tệp mới được sử dụng để tích hợp trình điều khiển thiết bị Xilinx vào quy trình xây dựng Tornado. Một lần nữa, các tệp này được tạo tự động bởi SDK. Người dùng chỉ cần biết rằng các tệp tồn tại. Các tệp này có tiền tố là tên của phiên bản bộ xử lý trong dự án thiết kế phần cứng. [Hình 1-3](#) cho thấy một ví dụ về các tệp xây dựng trình điều khiển.



Hình 1-3: Tornado 2.x Project IDE - Tệp

Tích hợp thiết bị

Các thiết bị trong hệ thống nhúng dựa trên FPGA có các mức độ tích hợp khác nhau với hệ điều hành VxWorks. Người dùng SDK có thể chọn mức độ tích hợp trong hộp thoại Thiết bị ngoại vi được kết nối của tab Tham số Thư viện / Hệ điều hành. Dưới đây là danh sách các thiết bị hiện được hỗ trợ và mức độ tích hợp của chúng.

- Một hoặc nhiều thiết bị UART 16450/16550 / Lite có thể được tích hợp vào giao diện VxWorks Serial I / O (SIO). Điều này làm cho một UART khả dụng cho I / O tệp và printf / stdio. Chỉ một thiết bị UART có thể được chọn làm bảng điều khiển, nơi I / O tiêu chuẩn (stdin, stdout và stderr) đều hướng đến. Một thiết bị UART, khi được tích hợp vào giao diện SIO, phải có khả năng tạo ra một ngắt. Tham khảo tệp sysSerial.c của BSP để xem chi tiết về tích hợp này.
- Các thiết bị MAC Ethernet 10/100, Ethernet Lite 10/100 và Gigabit Ethernet MAC có thể được tích hợp vào giao diện VxWorks Enhanced Network Driver (END). Điều này làm cho thiết bị khả dụng với ngăn xếp mạng VxWorks và do đó là các ứng dụng cấp ô cắm. Một thiết bị Ethernet, khi được tích hợp vào giao diện END, phải có khả năng tạo ra các ngắt. Tham khảo tệp configNet.h và sysNet.c của BSP để xem chi tiết về tích hợp này. Ngoài ra, người dùng có thể cần sửa đổi các giá trị đóng khởi động mặc định trong config.h để thiết bị Ethernet được sử dụng làm thiết bị khởi động.
- Bộ điều khiển ngắt có thể được kết nối với xử lý ngoại lệ VxWorks intLib và Chân ngắt không quan trọng bên ngoài PowerPC 405. BSP được tạo hiện không xử lý tích hợp bộ điều khiển ngắt cho chân ngắt quan trọng của PowerPC™ 405, cũng như không hỗ trợ kết nối trực tiếp của một thiết bị ngắt đơn (không phải intc) với bộ xử lý. Tuy nhiên, người dùng luôn có thể tự do thêm tích hợp này theo cách thủ công trong tệp sysInterrupt.c của BSP.
- Bộ điều khiển System ACE™ có thể được kết nối với VxWorks như một thiết bị khồi, cho phép người dùng gắn hệ thống tập tin vào thiết bị CompactFlash được kết nối với bộ điều khiển System ACE. Người dùng phải gọi các hàm BSP theo cách thủ công để khởi tạo Hệ thống ACE / CompactFlash dưới dạng thiết bị khồi và gắn nó vào hệ điều hành DOS. Các chức năng hiện có sẵn cho người dùng là: sysSystemAceInitFS () và sysSystemAceMount (). Bộ điều khiển ACE hệ thống, khi được tích hợp vào giao diện thiết bị khồi, phải có khả năng tạo ra ngắt. Tham khảo tệp sysSystemAce.c trong BSP để biết thêm chi tiết. BSP sẽ gắn CF làm phân vùng đĩa DOS FAT bằng tiện ích bổ sung DosFs2.0 của Wind River. Để đưa các thư viện VxWorks cần thiết vào hình ảnh, các gói sau phải được định nghĩa trong config.h hoặc bởi Project Facility:

- INCLUDE_DOSFS_MAIN
- INCLUDE_DOSFS_FAT
- INCLUDE_DISK_CACHE
- INCLUDE_DISK_PART
- INCLUDE_DOSFS_DIR_FIXED
- INCLUDE_DOSFS_DIR_VFAT
- INCLUDE_CPIO

Theo chương trình, một ứng dụng có thể gắn kết hệ thống tệp DOS bằng cách sử dụng các lệnh gọi API sau:

TẬP TIN * fp;

```
sysSystemAceInitFS ();
if (sysSystemAceMount ("/ cf0", 1)! = OK)
{
    /* xử lý lỗi */
}

fp = fopen ("/ cf0 / myfile.dat", "r");
```

- Một cầu PCI có thể được khởi tạo và cung cấp cho trình điều khiển VxWorks PCI tiêu chuẩn và các chức năng cầu hình. Người dùng được khuyến khích chỉnh sửa các tệp config.h và sysBusPci.c BSP để điều chỉnh các địa chỉ và cầu hình bộ nhớ PCI cho hệ thống đích của họ. Lưu ý rằng ngắt PCI không được tích hợp tự động vào BSP.
- Tất cả các thiết bị khác và trình điều khiển thiết bị liên quan không được tích hợp chặc chẽ vào giao diện VxWorks. Thay vào đó, chúng được tích hợp lỏng lẻo và khả năng truy cập vào các thiết bị này bằng cách truy cập trực tiếp vào trình điều khiển thiết bị được liên kết từ ứng dụng của người dùng.

- Các lõi ngư ời dùng và trình điều khiển thiết bị liên quan, nếu được bao gồm trong dự án EDK, được hỗ trợ thông qua quy trình tạo BSP. Trình điều khiển thiết bị lõi của ngư ời dùng sẽ được sao chép vào BSP giống như cách sao chép trình điều khiển thiết bị Xilinx. Điều này giả định cấu trúc thư mục của trình điều khiển thiết bị lõi ngư ời dùng khớp với cấu trúc của trình điều khiển thiết bị Xilinx. Thư mục con / data và / build của trình điều khiển thiết bị phải tồn tại và được định dạng giống như trình điều khiển thiết bị Xilinx. Điều này bao gồm đoạn mã CDF và các tệp xtag trong thư mục con / build / vxworks5_4 . Trình điều khiển thiết bị của ngư ời dùng không được tích hợp tự động vào bất kỳ giao diện hệ điều hành nào (ví dụ: SI0), nhưng chúng có sẵn để ứng dụng truy cập trực tiếp.

Sai lệch

Danh sách sau đây tóm tắt sự khác biệt giữa BSP do SDK tạo và BSP truyền thống.

- Một cấu trúc thư mục bổ sung được thêm vào thư mục BSP gốc để chứa nguồn trình điều khiển thiết bị các tệp mã.
- Để giữ cho BSP có thể xây dựng trong khi duy trì khả năng tư ơng thích với cơ sở Dự án Tornado, một tập hợp các tệp có tên <procname>_drv_<driver>_<version>.c điền vào thư mục BSP chỉ cần # bao gồm mã nguồn từ thư mục con trình điều khiển của BSP.
- BSP Makefile đã được sửa đổi để trình biên dịch có thể tìm thấy mã nguồn của trình điều khiển. Makefile chứa nhiều thông tin hơn về độ lệch này và ý nghĩa của nó.
- Việc sử dụng SystemACE có thể yêu cầu thay đổi đối với các tệp mã nguồn VxWorks được tìm thấy trong thư mục phân phối Tornado. Xem [Bootrom với SystemACE làm Thiết bị khởi động](#).

Hạn chế

BSP được tạo tự động nên được coi là một điểm đầu tốt cho ngư ời dùng, nhưng không nên được mong đợi để đáp ứng tất cả các nhu cầu của ngư ời dùng ngay khi xuất xưởng. Do sự phức tạp tiềm ẩn của BSP, nhiều tính năng có thể được bao gồm trong BSP và sự hỗ trợ cần thiết cho các thiết bị bo mạch bên ngoài FPGA, BSP được tạo tự động có thể sẽ yêu cầu ngư ời dùng cài tiền. Tuy nhiên, BSP được tạo sẽ có thể biên dịch được và sẽ chứa các trình điều khiển thiết bị cần thiết được trình bày trong hệ thống nhúng dựa trên FPGA. Một số thiết bị thường được sử dụng cũng được tích hợp hệ điều hành. Các hạn chế cụ thể được liệt kê dưới đây.

- Bộ điều khiển ngắt được kết nối với chân ngắt quan trọng PowerPC™ 405 không được tích hợp tự động vào sơ đồ ngắt của VxWorks. Hiện chỉ hỗ trợ ngắt bên ngoài.
- Không hỗ trợ phát hiện lỗi xe buýt từ cầu xe buýt hoặc trọng tài.
- Tornado BSP dòng lệnh mặc định sử dụng trình biên dịch GNU. Ngư ời dùng phải thủ công thay đổi Makefile để sử dụng trình biên dịch DIAB hoặc chỉ định trình biên dịch DIAB khi tạo dự án Tornado dựa trên BSP.
- Các dải địa chỉ bộ nhớ có thể cần được điều chỉnh trong config.h để phù hợp với các thiết bị bộ nhớ cụ thể và dải địa chỉ của chúng.
- Bộ nhớ đệm PowerPC™ 405 bị tắt theo mặc định. Ngư ời dùng phải kích hoạt bộ nhớ đệm theo cách thủ công thông qua tệp config.h hoặc menu dự án Tornado.
- Khi cài đặt System ACE bị tắt theo mặc định.
- Khi SystemACE được thiết lập để tải hình ảnh VxWorks vào RAM thông qua JTAG, tắt cả các khởi động đều lạnh (tức là không khởi động ấm). Điều này là do bộ điều khiển System ACE đặt lại bộ xử lý bất cứ khi nào nó thực hiện tải xuống tệp ace. Một tác động của điều này có thể khiến các thông báo ngoại lệ do VxWorks tạo ra không được in trên bảng điều khiển khi hệ thống được khởi động lại do một ngoại lệ trong ISR hoặc sự cố hạt nhân.
- Không thể sử dụng hình ảnh nén nào với SystemACE. Điều này áp dụng cho các hình ảnh nén tiêu chuẩn được tạo bằng Tornado, chẳng hạn như "bootrom". Hình ảnh nén không thể được đặt trên SystemACE dưới dạng tệp ace. Hệ thống ACE không thể giải nén dữ liệu khi nó ghi vào RAM. Bắt đầu một hình ảnh như vậy sẽ dẫn đến sự cố hệ thống.

- Một bản dựng dòng lệnh không thể khởi tạo mạng khi SystemACE là thiết bị khởi động. Đây yêu cầu ứng dụng cung cấp mã để khởi tạo mạng khi SystemACE là thiết bị khởi động. Để giải quyết vấn đề này, hãy xem thảo luận về \$ WIND_BASE / target /src/config/usinetwork.c trong phần Bootrom với SystemACE làm Thiết bị khởi động.
- Trên bộ xử lý PowerPC™ 405, vectơ đặt lại ở địa chỉ vật lý 0xFFFFFFF. Có một cửa sổ thời gian ngắn trong đó bộ xử lý sẽ cố gắng tìm nạp và thực thi lệnh tại địa chỉ này trong khi SystemACE xử lý tệp ace. Bộ xử lý cần được cung cấp một cái gì đó để làm trong thời gian này ngay cả khi đó là một vòng quay:

FFFFFFFFFFC b.

Nếu BRAM chiếm phạm vi địa chỉ này, thì người thiết kế tạo ra dòng bit phải đặt hưng dẫn tại đây bằng tiện ích elf to BRAM được tìm thấy trong các công cụ Xilinx ISE.

Khởi động VxWorks Trình tự khởi động VxWorks

Có nhiều biến thể của hình ảnh VxWorks với một số dựa trên RAM, một số dựa trên ROM. Tùy thuộc vào thiết kế bảng, không phải tất cả những hình ảnh này đều được hỗ trợ. Danh sách sau đây thảo luận về các loại hình ảnh khác nhau:

- Ảnh nén ROM** - Những ảnh này bắt đầu thực thi trong ROM và giải nén ảnh BSP vào RAM, sau đó chuyển quyền điều khiển sang ảnh đã giải nén trong RAM. Loại hình ảnh này không tương thích với SystemACE vì SystemACE không biết hình ảnh đã được nén và sẽ đặt nó vào RAM tại một địa chỉ sẽ bị thuật toán giải nén ghi đè khi nó bắt đầu. Có thể làm cho loại hình ảnh này hoạt động nếu các sửa đổi được thực hiện đối với các cấu hình tiêu chuẩn của Tornado để xử lý tình huống này.
- Hình ảnh dựa trên RAM** - Những hình ảnh này được tải vào RAM bằng bộ nạp khởi động, SystemACE hoặc trình giả lập. Những hình ảnh này được hỗ trợ đầy đủ.
- Hình ảnh dựa trên ROM** - Những hình ảnh này bắt đầu thực thi trong ROM, tự sao chép vào RAM sau đó chuyển việc thực thi vào RAM. Trong các thiết kế với SystemACE làm bộ nạp khởi động, hình ảnh được tự động sao chép vào RAM. Ví dụ BSP được mã hóa thủ công làm ngắn mạch hoạt động sao chép VxWorks để quá trình sao chép không xảy ra nữa sau khi quyền điều khiển được SystemACE chuyển sang RAM (xem romInit.s).
- Hình ảnh thường trú trong ROM** - Những hình ảnh này bắt đầu thực thi trong ROM, sao chép phần dữ liệu vào RAM và quá trình thực thi vẫn còn trong ROM. Trong các hệ thống chỉ có SystemACE, hình ảnh này không được hỗ trợ. Về mặt lý thuyết, BRAM có thể được sử dụng như một ROM, tuy nhiên, các bộ phận Virtex-II Pro hiện tại đang được sử dụng trong bảng đánh giá không có khả năng lưu trữ hình ảnh VxWorks có kích thước từ 200KB đến hơn 700KB.

Trình tự khởi động "vxWorks"

Hình ảnh chuẩn này được thiết kế để một số thiết bị tải xuống dung lượng RAM mục tiêu. Sau khi tải xuống, bộ xử lý được thiết lập để bắt đầu thực thi tại function _sysInit tại địa chỉ RAM_LOW_ADRS. (hàng số này được định nghĩa trong config.h và Makefile). Hầu hết thời gian, thiết bị thực hiện tải xuống sẽ tự động thực hiện việc này vì nó có thể trích xuất điểm nhập từ hình ảnh.

- _sysInit:** Hàm hợp ngữ này chạy hết RAM sẽ thực hiện khởi tạo mức thấp.
Khi hoàn thành, hàm này sẽ thiết lập ngăn xếp ban đầu và gọi hàm "C" đầu tiên usrInit(). _sysInit nằm trong tệp mã nguồn <bspname>/sysALib.s.
- usrInit ():** Hàm "C" này chạy hết RAM sẽ thiết lập môi trường thời gian chạy "C" và thực hiện khởi tạo tiền nhân. Nó gọi sysHwInit() (được triển khai trong sysLib.c) để đặt HW ở trạng thái tĩnh. Khi hoàn thành, hàm này sẽ gọi kernelInit() để hiển thị nhân VxWorks.
Hàm này sẽ lần lượt gọi usrRoot() làm tác vụ đầu tiên.
- usrRoot ():** Thực hiện khởi tạo hậu nhân. Kết nối đồng hồ hệ thống, khởi tạo ngắn xếp TCP / IP, v.v. Nó gọi sysHwInit2() (được triển khai trong sysLib.c) để đính kèm và kích hoạt ngắt HW.
Khi hoàn tất, usrRoot() gọi mã khởi động ứng dụng người dùng usrAppInit() nếu được định cấu hình trong BSP.

Cả `usrInit()` và `usrRoot()` đều được thực hiện bởi Wind River. Các tệp mã nguồn mà chúng tồn tại khác nhau tùy thuộc vào dòng lệnh hoặc cơ sở Dự án Tornado đang được sử dụng để biên dịch hệ thống. Trong giao diện dòng lệnh, chúng được triển khai tại `$ WIND_BASE / target / config / all / usrConfig.c`. Dưới cơ sở dự án, chúng được duy trì trong thư mục dự án của người dùng.

Trình tự khởi động "bootrom_ncmp"

Hình ảnh tiêu chuẩn này dựa trên ROM nhưng trong thực tế, nó được liên kết để thực thi các địa chỉ RAM. Trong khi thực thi từ ROM, hình ảnh này sử dụng thủ thuật định địa chỉ tương đối để gọi các hàm xử lý các tác vụ trước khi chuyển đến RAM.

1. Bật nguồn. Vector bộ xử lý thành `0xFFFFFFFFC` nơi đặt lệnh nhảy chuyển quyền điều khiển tới `bootrom` tại địa chỉ `_romInit`.
 2. `_romInit`: Chức năng hợp ngữ này chạy hết ROM lưu ý rằng đây là khởi động ngoại rời nhảy để bắt đầu. Cả `_romInit` và `start` đều nằm trong tệp mã nguồn `<bspname> /romInit.s`.
 3. `start`: Chức năng hợp ngữ này chạy hết ROM sẽ thiết lập bộ xử lý, làm mát hiệu lực của bộ nhớ đệm và chuẩn bị cho hệ thống hoạt động khi hết RAM. Thao tác cuối cùng là gọi hàm "C" `romStart()` được thực hiện bởi Wind River và nằm trong tệp mã nguồn `$ WIND_BASE / target / config / all / bootInit.c`.
 4. `romStart()`: Hàm "C" này khi hết ROM sẽ sao chép VxWorks vào địa chỉ bắt đầu RAM của nó nằm tại `RAM_HIGH_ADRS` (hàng số này được định nghĩa trong `config.h` và `Makefile`). Sau khi sao chép VxWorks, quyền điều khiển được chuyển đến hàm `usrInit()` trong RAM.
 5. Làm theo bước 2 & 3 của trình tự khởi động "vxWorks".
- "bootrom_ncmp" Trình tự khởi động với SystemACE
- Hình ảnh không chuẩn này tương tự như hình ảnh được thảo luận trong phần trước ngoại trừ việc SystemACE được sử dụng để tải nó. Một số thay đổi phải được thực hiện đối với quá trình khởi động. Có thể tìm thêm thông tin trong phần `<RD Red> <BT BoldType> Bootrom` với SystemACE làm Thiết bị khởi động `<RD Red>`, trang 13
1. Bật nguồn. SystemACE tải hình ảnh vào RAM tại `RAM_HIGH_ADRS` (hàng số này được định nghĩa trong `config.h` và `Makefile`) và đặt bộ xử lý bắt đầu tìm nạp các hướng dẫn tại địa chỉ `_romInit`. 2. `_romInit`: Hàm hợp ngữ này chạy hết RAM lưu ý rằng đây là khởi động ngoại rời nhảy để bắt đầu. Cả `_romInit` và `start` đều nằm trong tệp mã nguồn `<bspname> /romInit.s`.
 3. `start`: Hàm hợp ngữ này chạy hết RAM chỉ cần chuyển đến hàm `_sysInit`. Lệnh gọi tới `romStart()` bị bỏ qua vì SystemACE đã tải bootrom vào địa chỉ RAM đích của nó.
 4. Làm theo các bước 1, 2 và 3 của trình tự khởi động "vxWorks".

Bootroms

Bootrom là một hình ảnh VxWorks được thu nhỏ lại, hoạt động theo cách giống như BIOS của PC. Công việc chính của nó là tìm và khởi động một hình ảnh VxWorks đầy đủ. Hình ảnh VxWorks đầy đủ có thể nằm trên đĩa, trong bộ nhớ flash hoặc trên một số máy chủ thông qua Ethernet. Bootrom phải được biên dịch theo cách mà nó có khả năng truy xuất hình ảnh. Nếu hình ảnh được truy xuất từ mạng Ethernet, thì bootrom phải có ngăn xếp TCP / IP được biên dịch, nếu hình ảnh nằm trên đĩa, thì bootrom phải có hỗ trợ truy cập đĩa được biên dịch trong, v.v. Các bootrom không làm gì khác hơn là truy xuất và bắt đầu hình ảnh đầy đủ và duy trì một dòng khởi động. Dòng khởi động là một chuỗi văn bản đặt các đặc điểm người dùng nhất định như địa chỉ IP của mục tiêu nếu sử dụng Ethernet và đường dẫn tệp đến hình ảnh VxWorks để khởi động.

Bootrom không phải là một yêu cầu bắt buộc. Chúng thường được sử dụng trong môi trường phát triển sau đó được thay thế bằng hình ảnh VxWorks sẵn xuất.

Tạo Bootroms

Tại trình bao lệnh trong thư mục BSP, hãy phát hành lệnh sau để tạo hình ảnh bootrom không nén (bắt buộc đối với SystemACE):

```
tạo bootrom_ncmp
hoặc

làm bootrom
```

để tạo ra một hình ảnh nén phù hợp để đặt trong một mảng bộ nhớ flash.

Hiển thị Bootrom

Khi khởi động nguồn, nếu các bootrom hoạt động bình thường, đầu ra tự động tự như sau sẽ được nhìn thấy trên cổng nối tiếp của bảng điều khiển:

Khởi động hệ thống VxWorks

Bản quyền 1984-1998 Wind River Systems, Inc.

CPU: PPC405 Rev D
Phiên bản: 5.4.2
Phiên bản BSP: 1.2 / 0
Ngày tạo: 26 tháng 7 năm 2002, 12:51:32

Nhấn phím bất kỳ để dừng tự động khởi động ...
3

[VxWorks Boot]:

Gõ "trợ giúp" tại dấu nhắc này sẽ liệt kê các lệnh có sẵn.

Bootline

Dòng khởi động là một chuỗi văn bản xác định các đặc điểm có thể phục vụ của người dùng như địa chỉ IP của bảng đích và cách tìm hình ảnh vxWorks để khởi động. Dòng khởi động được bootrom duy trì trong thời gian chạy và thường được giữ trong một số vùng lưu trữ không bay hơi (NVRAM) của hệ thống như EEPROM hoặc bộ nhớ flash. Nếu không có NVRAM hoặc xảy ra lỗi khi đọc nó, thì dòng khởi động được mã hóa cứng bằng DEFAULT_BOOT_LINE được xác định trong tệp mã nguồn config.h của BSP. Trong các hệ thống hoàn toàn mới mà NVRAM chưa được khởi tạo, thì dòng khởi động có thể vô nghĩa.

Dòng khởi động có thể được thay đổi nếu chuỗi đếm ngược tự động khởi động bị gián đoạn bằng cách nhập một ký tự trên cổng nối tiếp bảng điều khiển. Sau đó, lệnh "c" có thể được sử dụng để chỉnh sửa tự động dòng khởi động. Nhập "p" để xem đường khởi động. Trên ảnh không phải bootrom, bạn vẫn có thể thay đổi dòng khởi động bằng cách nhập lệnh bootChange tại dấu nhắc máy chủ hoặc trình bao đích.

Danh sách sau đây trình bày ý nghĩa của các trường dòng khởi động:

boot device: Thiết bị khởi động từ. Đây có thể là Ethernet hoặc một đĩa cục bộ. Lưu ý rằng khi thay đổi dòng khởi động, số đơn vị có thể được hiển thị thêm vào trường này ("xemac0" hoặc "sysace = 10) khi nhắc thiết bị khởi động mới. Số này có thể được bỏ qua.

số bộ xử lý: Luôn luôn là 0 với các hệ thống bộ xử lý đơn lẻ.

Tên máy chủ: Đặt tên khi cần thiết.

tên tệp: Hình ảnh VxWorks để khởi động. inet

trên ethernet (e): Địa chỉ Internet IP của mục tiêu. Nếu không có giao diện mạng, thì trường này có thể được để trống.

host inet (h): Địa chỉ Internet IP của máy chủ. Nếu không có giao diện mạng, thì trự ờng này có thể để trống.

user (u): Tên người dùng để truy cập hệ thống tệp máy chủ. Chọn bất kỳ tên nào phù hợp với bạn. Máy chủ ftp của bạn phải được thiết lập để cho phép người dùng này truy cập vào hệ thống tệp máy chủ.

Mật khẩu ftp (pw): Mật khẩu để truy cập hệ thống tệp máy chủ. Chọn bất kỳ tên nào phù hợp với bạn. Của bạn. Máy chủ ftp phải được thiết lập để cho phép người dùng này truy cập vào hệ thống tệp máy chủ.

flags (f): Để biết danh sách các tùy chọn, hãy nhập lệnh "help" tại dấu nhắc [VxWorks Boot]:.

target name (tn): Đặt tên khi cần thiết. Đặt theo yêu cầu mạng.

other (o): Trự ờng này hữu ích khi bạn có thiết bị không phải Ethernet làm thiết bị khởi động. Trong trự ờng hợp này, VxWorks sẽ không khởi động mạng khi khởi động. Chỉ định thiết bị Ethernet ở đây sẽ kích hoạt thiết bị đó tại thời điểm khởi động với các tham số mạng được chỉ định trong các trự ờng dòng khởi động khác.

inet trên bảng nồi da năng (b): Thư ờng để trống nếu hệ thống đích không nằm trên VME hoặc PCI bảng nồi da năng.

Công inet (g): Nhập địa chỉ IP vào đây nếu bạn phải đi qua cổng để đến máy tính chủ. Nếu không thì để trống.

(các) tập lệnh khởi động: Đường dẫn đến một tệp trên máy tính chủ chứa các lệnh shell để thực thi sau khi khởi động xong. Để trống nếu không sử dụng tập lệnh. Ví dụ:

Tập lệnh thư ờng trú của SystemACE: /cf0/vxworks/scripts/myscript.txt

Tập lệnh thư ờng trú trên máy chủ: c: /temp/myscript.txt

Bootrom với SystemACE làm thiết bị khởi động

Các bootrom hỗ trợ SystemACE có khả năng khởi động hình ảnh VxWorks trực tiếp từ thiết bị Compact Flash dưới dạng tệp elf thông thư ờng hoặc tệp ace.

Các chế độ bắt buộc đối với Nguồn VxWorks

1. Mặc dù SDK có khả năng tạo BSP sử dụng SystemACE trong hình ảnh "vxWorks" có thể mở và đóng tệp trong hệ thống tệp DOS, nó không thể tạo BSP sử dụng SystemACE làm thiết bị khởi động "bootrom". Để sử dụng SystemACE theo cách này, cần phải sửa đổi nhiều đối với mã bootrom do Wind River cung cấp. Wind River cho phép các nhà phát triển BSP thay đổi các tệp mã nguồn Tornado miễn là họ giữ các thay đổi cục bộ đối với BSP và giữ nguyên mã gốc. Hai tệp phải được sửa đổi từ phiên bản gốc của chúng là:

1. \$ WIND_BASE / target / config / all / bootConfig.c: Tệp này bị ghi đè bằng tệp được tìm thấy trong thư mục <bspname>. Các thay đổi cần thiết là thêm mã để phân tích cú pháp dòng khởi động đúng cách và khởi tạo và sử dụng SystemACE làm thiết bị khởi động JTAG và DOS. Để ghi đè bootConfig.c mặc định, dòng sau phải được thêm vào Makefile của BSP: BOOTCONFIG = ./bootConfig.c.

2. \$ WIND_BASE / target / config / comps / src / net / usrNetBoot.c: Tệp này được ghi đè bằng tệp được tìm thấy trong thư mục <bspname> /net/usrNetBoot.c. Những thay đổi cần thiết là thêm mã để làm cho VxWorks biết rằng SystemACE là một hệ thống dựa trên đĩa như IDE, SCSI hoặc ổ đĩa mềm. Thay đổi này cho phép một BSP được xây dựng từ Dự án Tornado được tải xuống với bootrom hỗ trợ SystemACE để xử lý đúng trự ờng "khác" của đư ờng khởi động. Sự tồn tại của tệp đã sửa đổi trong thư mục BSP sẽ tự động ghi đè tệp gốc.

Cả hai tệp này đều không được SDK cung cấp vì chúng được Wind River duy trì ở dạng ban đầu.

Không thể ghi đè tệp thứ ba, \$ WIND_BASE / target / src / config / usrnNetwork.c do kiến trúc của quá trình xây dựng BSP dòng lệnh. Điều này ánh hường đến các BSP có khả năng mạng được xây dựng từ dòng lệnh được tải xuống với bootrom hỗ trợ SystemACE. Nếu không sửa đổi usrnNetwork.c, các BSP bị ánh hường không thể khởi chạy thiết bị mạng của họ và phải dựa vào mã ứng dụng để thực hiện chức năng này. Nếu người dùng muốn, họ có thể thực hiện thay đổi đối với tệp này trong cài đặt Tornado của họ. Có

như gợc điểm đổi với cách tiếp cận này vì bất kỳ chỉnh sửa nào đư ợc thực hiện đổi với tệp này đều ảnh hưởng đến tất cả người dùng cài đặt đó và có thể bị mất nếu người dùng nâng cấp hoặc cài đặt lại Tornado. Đổi với những người thích mạo hiểm, sự thay đổi đối với usrNetwork.c xảy ra trong hàm usrNetInit () .

từ:

```
if ((strncpy (params.bootDev, "scsi", 4) == 0) ||
    (strncpy (params.bootDev, "ide", 3) == 0) ||
    (strncpy (params.bootDev, "ata", 3) == 0) ||
    (strncpy (params.bootDev, "fd", 2) == 0) (strncpy    ||
    (params.bootDev, "tffs", 4) == 0))

đến

if ((strncpy (params.bootDev, "scsi", 4) == 0) ||
    (strncpy (params.bootDev, "ide", 3) == 0) ||
    (strncpy (params.bootDev, "ata", 3) == 0) ||
    (strncpy (params.bootDev, "fd", 2) == 0) (strncpy    ||
    (params.bootDev, "sysace", 6) == 0) ||
    (strncpy (params.bootDev, "tffs", 4) == 0))
```

Chỉnh sửa mã này có nguy cơ của riêng bạn.

Cấu hình đặc biệt

Chuẩn bị hình ảnh bootrom_ncmp có thẻ tải xuống bởi SystemACE dưới dạng tệp ace cần có cấu hình đặc biệt. Những thay đổi này là bắt buộc vì bootrom đư ợc liên kết để bắt đầu chạy hết thiết bị nhớ không bay hơi, sao chép chính nó vào RAM, sau đó chuyển quyền điều khiển sang bản sao RAM. Các thay đổi sẽ ngăn hoạt động sao chép vì SystemACE đã đặt bootrom vào thiết bị RAM khi đặt lại.

một. Thay đổi định nghĩa của ROM_TEXT_ADRS và ROM_WARM_ADRS thành giá trị tương ứng với RAM_HIGH_ADRS trong cả config.h và Makefile.

b. Thay đổi mã hợp ngữ tại nhãn "start" trong romInit.s để chỉ cần chuyển đến chức năng _sysInit:
bắt đầu:

```
LOADPTR (r1, _sysInit)
mtlr      r1
blrl
```

Định dạng dòng khởi động

Trường "thiết bị khởi động" của dòng khởi động đư ợc chỉ định bằng cú pháp sau:

sysace = <số phân vùng>

trong đó <số phân vùng> là phân vùng để khởi động từ đó. Thông thường, giá trị này đư ợc đặt thành 1, nhưng một số thiết bị CF không có bảng phân vùng và đư ợc định dạng như thế chúng là một đĩa mềm lớn. Trong trường hợp này, chỉ định 0 làm số phân vùng. Việc không lấy đúng số phân vùng sẽ dẫn đến lỗi đư ợc các thư viện dosFS của VxWork thông báo khi ổ đĩa đư ợc gắn kết.

Trường "tên tệp" của dòng khởi động đư ợc đặt tùy thuộc vào cách Hệ thống ACE khởi động hệ thống. Có hai phư ơng pháp khởi động:

- Khởi động từ một tệp thông thường. Điều này tư ơng tự như khởi động mạng trong đó hình ảnh vxWorks nằm trong thiết bị lưu trữ flash nhỏ gọn SystemACE thay vì hệ thống tệp máy chủ. Thiết bị flash nhỏ gọn là một phân vùng hệ thống tệp DOS FAT. Chỉ cần xây dựng vxWorks bằng các công cụ Tornado, sau đó sao chép tệp hình ảnh kết quả vào thiết bị flash nhỏ gọn bằng đầu đọc thẻ USB hoặc công cụ tư ơng tự. Sau đó chỉ định tệp đó trong trường "tên tệp" của rom khởi động.

"Tên tệp" phải có cú pháp sau:

/ cf0 / <path / to / vxWorks / image>

trong đó `cf0` là điểm gắn kết. <path / to / vxWorks / image> phải cung cấp đường dẫn đầy đủ đến hình ảnh VxWorks để khởi động. Khi được chỉ định theo cách này, bootrom sẽ gắn ổ đĩa dưới dạng đĩa được định dạng FAT, tải tệp vào bộ nhớ và bắt đầu thực thi.

2. Khởi động từ một tệp ace. Tệp ace chỉ có thể chứa HW, SW only hoặc HW + SW. Khi khởi động từ một tệp ace với HW, FPGA được lập trình lại. Nếu tệp ace chứa SW, thì nó được tải vào bộ nhớ, PC của bộ xử lý được đặt thành điểm nhập và được giải phóng để bắt đầu tìm nạp các hướng dẫn. Phương pháp khởi động này linh hoạt ở chỗ một cấu hình HW hoàn toàn khác có thể được "khởi động" từ bootrom VxWorks.

"Tên tệp" phải có cú pháp sau:

```
cfgaddr [x]
```

trong đó [X] là một số từ 0 đến 7 tương ứng với một trong các thư mục cấu hình được chỉ định trong cấu trúc tệp XILINX.SYS trong thư mục gốc của thiết bị flash nhỏ gọn. Nếu [X] bị bỏ qua, thì cấu hình mặc định sẽ được sử dụng. Cấu hình mặc định thường được chọn bằng một công tắc xoay được gắn ở đâu đó trên bảng đánh giá. Bootrom sẽ kích hoạt tải xuống JTAG của tệp ace được trả đến bởi địa chỉ cấu hình được chỉ định. Chỉ nên có một tệp duy nhất có phần mở rộng .ace trong thư mục cấu hình đã chọn.

Trong cả hai trường hợp khởi động, nếu bạn muốn thiết bị Ethernet khởi động khi VxWorks đã tải xuống khởi động, thì hãy sửa đổi trường "khác" của dòng khởi động để chứa tên của thiết bị mạng.

Bootrom với 10/100 Ethernet (EMAC) làm Thiết bị khởi động

SDK sẽ tạo ra một BSP có khả năng được xây dựng như một bootrom bằng cách sử dụng EMAC làm thiết bị khởi động. Cấu hình người dùng rất ít được yêu cầu. Địa chỉ MAC được mã hóa cứng trong tệp nguồn sysNet.c. BSP có thể được sử dụng với MAC mặc định miễn là mục tiêu nằm trên một mạng riêng và không có nhiều hơn một mục tiêu trên mạng đó có cùng địa chỉ MAC mặc định. Nếu không, nhà thiết kế nên thay thế MAC này bằng một chức năng để lấy một MAC từ thiết bị nhớ không bay hơi trên bảng đích của họ.

Để chỉ định EMAC làm thiết bị khởi động trong bootrom, hãy thay đổi trường "thiết bị khởi động" trong dòng khởi động thành "xemac". Nếu có một EMAC duy nhất, thì hãy đặt "số đơn vị" thành 0.

Bootrom với 1 Gigabit Ethernet (GEMAC) làm Thiết bị khởi động

SDK sẽ tạo một BSP có khả năng được xây dựng như một bootrom bằng cách sử dụng GEMAC làm thiết bị khởi động. Cấu hình người dùng rất ít được yêu cầu. Địa chỉ MAC được mã hóa cứng trong tệp nguồn sysNet.c. BSP có thể được sử dụng với MAC mặc định miễn là mục tiêu nằm trên một mạng riêng và không có nhiều hơn một mục tiêu trên mạng đó có cùng địa chỉ MAC mặc định. Nếu không, nhà thiết kế nên thay thế MAC này bằng một chức năng để lấy một MAC từ thiết bị nhớ không bay hơi trên bảng đích của họ.

Để chỉ định GEMAC làm thiết bị khởi động trong bootrom, hãy thay đổi trường "thiết bị khởi động" trong dòng khởi động thành "xgemac". Nếu có một GEMAC duy nhất, thì hãy đặt "số đơn vị" thành 0.

Ví dụ về dòng khởi động

Ví dụ sau khởi động từ ethernet bằng cách sử dụng Xilinx "xemac" làm thiết bị khởi động. Hình ảnh được khởi động nằm trên hệ thống tệp máy chủ trên ổ C.

thiết bị khởi động	:	xemac
số đơn vị	:	0
tên máy chủ số bộ xử lý	:	0
	:	chủ nhà
tên tập tin	:	c: / tornado / target / config / ml507 / vxWorks
inet trên ethernet (e):	192.168.0.2	
máy chủ inet (h)	:	192.168.0.1

```

người dùng (u) : xemhost
mật khẩu ftp (pw) cờ (f) : sao cung đư ợc
tên đích (tn) khác (o) : 0x0
: vxtarget
:

```

Ví dụ sau khởi động từ một tập tin cư trú trên phân vùng đầu tiên của thiết bị flash nhỏ gọn SystemACE. Nếu tệp đư ợc khởi động từ / cf0 / vxworks / images / vxWorks sử dụng mạng, thì thiết bị "xemac" sẽ đư ợc khởi chạy.

```

thiết bị khởi động : sysace = 1
số đơn vị : 0
tên máy chủ số bộ xử lý : 0
: chủ nhà
tên tập tin : / cf0 / vxworks / images / vxWorks
inet trên ethernet (e): 192.168.0.2
máy chủ inet (h) : 192.168.0.1
người dùng (u) : xemhost
mật khẩu ftp (pw) cờ (f) : sao cung đư ợc
tên đích (tn) khác (o) : 0x0
: vxtarget
: xemac

```

Ví dụ sau khởi động từ một cư dân tệp ace trên phân vùng đầu tiên của thiết bị flash nhỏ gọn SystemACE. Vì trí của tệp ace đư ợc đặt bởi XILINX.SYS nằm trong thư mục gốc của thiết bị flash nhỏ gọn. Nếu tệp ace chứa hình ảnh VxWorks SW sử dụng mạng, thì thiết bị "xemac" đư ợc khởi tạo cho hình ảnh đó.

```

thiết bị khởi động : sysace = 1
số đơn vị : 0
tên máy chủ số bộ xử lý : 0
: chủ nhà
tên tập tin : cfgaddr2
inet trên ethernet (e): 192.168.0.2
máy chủ inet (h) : 192.168.0.1
người dùng (u) : xemhost
mật khẩu ftp (pw) cờ (f) : sao cung đư ợc
tên đích (tn) khác (o) : 0x0
: vxtarget
: xemac

```

Bộ nhớ đệm

Hướng dẫn và bộ nhớ đệm dữ liệu đư ợc quản lý bởi các thư viện bộ đệm VxWorks. Chúng đư ợc kích hoạt bằng cách sửa đổi các hằng số sau trong config.h hoặc bằng cách sử dụng cơ sở Dự án Tornado để thay đổi các hằng số cùng tên:

- INCLUDE_CACHE_SUPPORT - Nếu đư ợc xác định, các thư viện bộ đệm VxWorks đư ợc liên kết vào hình ảnh. Nếu bộ nhớ đệm không đư ợc mong muón, thì hãy # hoàn thiện hằng số này.
- USER_I_CACHE_ENABLE - Nếu đư ợc định nghĩa, VxWorks sẽ kích hoạt bộ đệm chỉ lệnh lúc khởi động. Yêu cầu INCLUDE_CACHE_SUPPORT đư ợc xác định để có bất kỳ tác dụng nào.
- USER_D_CACHE_ENABLE - Nếu đư ợc định nghĩa, VxWorks sẽ bật bộ đệm dữ liệu vào thời gian khởi động. Yêu cầu INCLUDE_CACHE_SUPPORT đư ỢC xác định để có bất kỳ tác dụng nào.

MMU

Nếu MMU đư ỢC bật, thì điều khiển bộ đệm đư ỢC thảo luận trong phần trư ớc có thẻ không có bất kỳ tác dụng nào. MMU đư ỢC quản lý bởi các thư viện độc quyền của VxWorks nhưng thiết lập ban đầu đư ỢC xác định trong BSP. Đèn

kích hoạt MMU, hằng số INCLUDE_MMU_BASIC phải được xác định trong config.h hoặc bằng cách sử dụng Project Facility. Hằng số USER_D_MMU_ENABLE và USER_I_MMU_ENABLE kiểm soát liệu MMU hứa hẹn dẫn và / hoặc dữ liệu có được sử dụng hay không.

VxWorks khởi tạo MMU dựa trên dữ liệu trong cấu trúc sysPhysMemDesc được định nghĩa trong sysCache.c. Trong số những thứ khác, bảng này định cấu hình các vùng bộ nhớ với các thuộc tính sau:

- Cho phép thực hiện lệnh hay không.
- Cho phép ghi dữ liệu
- Các thuộc tính về khả năng lưu vào bộ nhớ cache của lệnh & dữ liệu.
- Phần bù dịch được sử dụng để tạo địa chỉ ảo.

PPC405 có khả năng có các thuộc tính khác bao gồm bảo vệ vùng, tuy nhiên, tài liệu về Wind River khá thiếu về lĩnh vực này và không rõ liệu gói MMU cơ bản có hỗ trợ chúng hay không. Một tiện ích bổ sung có sẵn từ Wind River cho các hoạt động MMU nâng cao.

Khi VxWorks khởi tạo MMU, nó sẽ lấy các định nghĩa từ sysPhysMemDesc và tạo các mục nhập bảng trang (PTE) trong RAM. Mỗi PTE mô tả 4KB vùng bộ nhớ (mặc dù bộ xử lý có khả năng đại diện lên đến 16MB cho mỗi PTE). Hãy lưu ý rằng việc chỉ định các vùng bộ nhớ lớn sử dụng lượng RAM đáng kể để lưu trữ các PTE. Để ánh xạ 4MB không gian bộ nhớ liền kề, cần 8KB RAM để lưu trữ các PTE.

Để tăng hiệu suất với gói MMU cơ bản của VxWorks cho bộ xử lý PPC405, có thể có lợi nếu không bật MMU hứa hẹn dẫn và dựa vào cài đặt điều khiển bộ đệm trong thanh ghi ICCR.

Chiến lược này có thể giảm đáng kể số lượng lỗi trang trong khi vẫn giữ các hứa hẹn dẫn trong bộ nhớ cache. Cài đặt ban đầu của ICCR được xác định trong tệp tiêu đề <bspname>.h.

Nếu không bật MMU, các quy tắc sau áp dụng cho việc định cấu hình các thuộc tính truy cập bộ nhớ và bộ nhớ đệm:

- Không có bản dịch địa chỉ, tất cả các địa chỉ hiệu quả là vật lý.
- Mức độ chi tiết của kiểm soát bộ nhớ cache là 128MB.
- Thuộc tính được bảo vệ chỉ áp dụng cho các lần tìm nạp lệnh suy đoán trên PPC405.



UG703

Tự động tạo các gói hỗ trợ bo mạch của Wind River VxWorks 6.3

Bản tóm tắt

Tài liệu này mô tả cách tạo tự động của Gói hỗ trợ bảng làm việc (BSP) bằng cách sử dụng Bộ phát triển phần mềm Xilinx® (SDK) (1). Tài liệu này chứa những điều sau đây phần:

- “[Tổng quan](#)”
- “[Tạo VxWorks 6.3 BSP](#)”
- “[VxWorks 6.3 BSP](#)”
- “[Khởi động VxWorks](#)”

Tổng quan

Một trong những hoạt động phát triển hệ thống nhúng chính là phát triển BSP. Việc tạo ra một BSP có thể là một quá trình kéo dài và tẻ nhạt phải phát sinh khi có sự thay đổi trong tổ hợp bộ vi xử lý bao gồm bộ xử lý và các thiết bị ngoại vi liên quan. Mặc dù việc quản lý những thay đổi này áp dụng cho bất kỳ dự án nào dựa trên bộ vi xử lý, nhưng giờ đây những thay đổi có thể được thực hiện nhanh hơn với sự ra đời của phần cứng Hệ thống trên chip (SoC) có thể lập trình được.

Tài liệu này mô tả việc tạo tự động VxWorks 6.3 BSP tùy chỉnh cho bộ xử lý IBM PowerPC® 405/440 và thiết bị ngoại vi như được định nghĩa trong Xilinx FPGA. BSP được tạo tự động cho phép các nhà thiết kế hệ thống nhúng:

- Giảm chu kỳ phát triển, do đó giảm thời gian đưa ra thị trường
- Tạo BSP tùy chỉnh để phù hợp với phần cứng và ứng dụng
- Loại bỏ lỗi thiết kế BSP (được tạo tự động dựa trên các thành phần được chứng nhận)
- Cho phép phát triển phần mềm ứng dụng bằng cách loại bỏ thời gian chờ phát triển BSP

VxWorks 6.3 BSP được tạo từ SDK, một IDE được phân phối như một phần của Bộ phát triển nhúng Xilinx (EDK) hoặc có sẵn riêng từ Xilinx. SDK được sử dụng để tạo các ứng dụng phần mềm cho các hệ thống nhúng trong Xilinx FPGA. VxWorks BSP chứa tất cả phần mềm hỗ trợ cần thiết cho hệ thống, bao gồm mã khởi động, trình điều khiển thiết bị và khởi tạo RTOS. BSP được tùy chỉnh dựa trên các thiết bị ngoại vi được người dùng chọn và cấu hình cho hệ thống nhúng dựa trên FPGA.

Các nhà thiết kế BSP có kinh nghiệm nên dễ dàng tích hợp BSP đã tạo vào hệ thống mục tiêu của họ. Người lại, những người dùng ít kinh nghiệm hơn có thể gặp khó khăn vì mặc dù SDK có thể tạo BSP hoạt động cho một tập hợp phần cứng IP nhất định, nhưng sẽ luôn có một số cấu hình và điều chỉnh bổ sung cần thiết để tạo ra hiệu suất tốt nhất từ hệ thống mục tiêu. Người dùng nên có sẵn Hướng dẫn dành cho nhà phát triển Wind River VxWorks BSP và Hướng dẫn dành cho lập trình viên ứng dụng VxWorks hoặc xem xét các lớp Wind River trên thiết kế BSP, có sẵn với một khoản chi phí bổ sung.

1. Xilinx SDK được sử dụng làm môi trường phát triển phần mềm chính cho người dùng Xilinx Embedded Development Kit (EDK) kể từ EDK 11.1i. Khả năng phát triển phần mềm của Xilinx Platform Studio (XPS) hiện không được dùng nữa. Các luồng được mô tả trong tài liệu này liên quan đến SDK, mặc dù chúng vẫn có thể áp dụng chung cho XPS trong khi các tính năng đó được cung cấp trong công cụ.

Yêu cầu

Bộ phát triển Wind River Workbench 2.5 phải được cài đặt trên máy tính chủ.

Vì SDK tạo ra các BSP có thể di chuyển lại được biên dịch và định cấu hình bên ngoài môi trường SDK, nên máy tính chủ không cần phải cài đặt cả Xilinx SDK và Workbench.

Định nghĩa thư viện bộ vi xử lý

SDK hỗ trợ giao diện trình cắm thêm cho các thư viện và hệ điều hành của bên thứ 3 thông qua giao diện Định nghĩa Thư viện Bộ vi xử lý (MLD), do đó cho phép các nhà cung cấp bên thứ 3 cung cấp phần mềm của họ cho người dùng SDK. Ngoài ra, nó cung cấp cho các nhà cung cấp một phương tiện để điều chỉnh thư viện hoặc BSP của họ cho phù hợp với hệ thống nhúng dựa trên FPGA. Bởi vì hệ thống có thể thay đổi dễ dàng, khả năng này rất quan trọng trong việc hỗ trợ đúng cách các hệ thống nhúng trong FPGA.

Xilinx phát triển và duy trì VxWorks 6.3 MLD trong các bản phát hành SDK của mình. MLD được sử dụng để tự động tạo VxWorks 6.3 BSP.

Phương pháp Tiếp cận Dựa trên Mẫu

Một tập hợp các tệp mẫu VxWorks 6.3 BSP được phát hành cùng với SDK. Các tệp mẫu này được sử dụng trong quá trình tạo BSP tự động và các sửa đổi thích hợp được thực hiện dựa trên cấu trúc của hệ thống nhúng dựa trên FPGA.

Các tệp mẫu này có thể được sử dụng làm tài liệu tham khảo để xây dựng BSP từ đầu nếu người dùng chọn không tự động tạo BSP.

Trình điều khiển thiết bị

Một tập hợp các tệp nguồn trình điều khiển thiết bị được phát hành cùng với SDK và nằm trong thư mục cài đặt. Trong quá trình tạo BSP tùy chỉnh, mã nguồn trình điều khiển thiết bị được sao chép từ thư mục cài đặt này sang thư mục BSP. Chỉ mã nguồn liên quan đến các thiết bị được tích hợp trong hệ thống nhúng dựa trên FPGA mới được sao chép. Bản sao này cung cấp cho người dùng một thư mục BSP độc lập, độc lập có thể được sửa đổi hoặc di dời. Nếu người dùng thực hiện các thay đổi đối với mã nguồn trình điều khiển thiết bị cho BSP này, sau đó muốn hoàn tác các thay đổi, các công cụ SDK có thể được sử dụng để tạo lại BSP. Tại thời điểm đó, các tệp nguồn của trình điều khiển thiết bị được mở lại từ thư mục cài đặt vào BSP.

Tạo VxWorks 6.3 BSP

Sử dụng SDK

SDK có sẵn dưới dạng tệp thực thi độc lập hoặc trong EDK và là môi trường phát triển phần mềm cho phần mềm nhúng xung quanh bộ xử lý Xilinx PowerPC 405/440 hoặc hệ thống nhúng dựa trên MicroBlaze™. Phần này mô tả các bước cần thiết để tạo BSP VxWorks 6.3 bằng SDK. Các bước này có thể áp dụng khi sử dụng công cụ The Xilinx 11.1i trở lên.

Giả định rằng một thiết kế phần cứng hợp lệ đã được tạo và xuất sang SDK, và SDK đã được mở và trỏ đến thiết kế phần cứng.

1. Sử dụng Tệp> Mới, tạo một dự án Gói Hỗ trợ Ban mới. Trong hộp thoại, nhập tên dự án và chọn vxworks6_3 làm Loại gói hỗ trợ hội đồng quản trị. Lưu ý rằng SDK có thể quản lý nhiều dự án thuộc các loại BSP khác nhau.

Hộp thoại Cài đặt Gói Hỗ trợ Ban ... sẽ hiển thị.

2. Định cấu hình thiết bị bảng điều khiển VxWorks:

Nếu một thiết bị nối tiếp, chẳng hạn như Uart được sử dụng làm bảng điều khiển VxWorks, hãy chọn hoặc nhập tên phiên bản của thiết bị nối tiếp làm thiết bị ngoại vi STDIN / STDOUT trong hộp thoại Cài đặt Gói Hỗ trợ Bảng. Điều quan trọng là phải nhập cùng một thiết bị cho cả STDIN và STDOUT. Hiện tại, chỉ có các thiết bị Uart 16550/16450 và UartLite được hỗ trợ làm thiết bị VxWorks console.

3. Tích hợp trình điều khiển thiết bị:

a. Kết nối với VxWorks:

Hộp thoại kết nối_periph nằm trong hộp thoại Thiết đặt gói hỗ trợ hội đồng quản trị

Các thiết bị ngoại vi đã được di chuyển sẵn để thuận tiện cho người dùng. Sử dụng hộp thoại này để sửa đổi các thiết bị ngoại vi đó để được tích hợp chặt chẽ với Hệ điều hành, bao gồm thiết bị đã được chọn làm thiết bị ngoại vi STDIN / STDOUT. Xem "[Tích hợp thiết bị](#)" trang 7 để biết thêm chi tiết về tích hợp chặt chẽ các thiết bị.

b. Kích thước bộ nhớ:

Trường này được sử dụng để định cấu hình BSP để phù hợp với kích thước bộ nhớ phần cứng thực tế trên bo mạch của bạn.

c. Uart16550_baud_rate:

Trường này được sử dụng để nhập tốc độ truyền cho các dự án có lõi UART 16550/16450. Không cần thiết phải nhập giá trị ở đây cho các dự án có lõi UART Lite vì tốc độ truyền được đặt cho UART Lite tại thời điểm xây dựng phần cứng.

4. Tạo VxWorks 6.3 BSP:

Bấm OK trên hộp thoại Cài đặt Gói Hỗ trợ Ban ... để tạo BSP. Đầu ra của lệnh gọi này được hiển thị trong cửa sổ bảng điều khiển SDK. Sau khi hoàn tất, VxWorks 6.3 BSP sẽ tồn tại trong không gian làm việc SDK của bạn, dưới tên thư mục dự án mà bạn đã tạo ở bước 1 trong thư mục con phiên bản PowerPC 405/440. Ví dụ: nếu trong thiết kế phần cứng là phiên bản bộ xử lý PowerPC 405, ppc405_0, thì BSP sẽ nằm tại <SDK workspace> / <SDK project name> / ppc405_0 / bsp_ppc405_0.

Sao lưu

Để tránh mất các thay đổi đối với tệp nguồn BSP, các tệp hiện có trong vị trí thư mục của BSP được sao chép vào thư mục sao lưu trước khi bị ghi đè. Thư mục sao lưu được đặt tên là backuptimestamp trong đó timestamp đại diện cho ngày và giờ hiện tại và nằm trong thư mục BSP. Vì BSP được tạo bởi SDK có thể định vị lại, bạn nên di chuyển BSP từ thư mục dự án SDK sang thư mục phát triển BSP thích hợp ngay khi nền tảng phần cứng ổn định.

VxWorks 6.3 BSP

Phần này mô tả cách tạo BSP cho VxWorks 6.3 BSP của SDK. Giả sử bạn đã quen thuộc với Wind River Workbench 2.5 IDE và một môi trường Workbench đã thiết lập. Bạn có thể sử dụng wrenv dòng lệnh tiện ích môi trường Wind River trên nền tảng Windows. Xem Hướng dẫn người dùng dòng lệnh Wind River Workbench: "Tạo lớp vỏ phát triển với wrenv" để biết thêm thông tin về cách sử dụng các tiện ích dòng lệnh.

BSP được tạo tự động được tích hợp vào Workbench IDE. BSP có thể được biên dịch từ dòng lệnh bằng cách sử dụng các công cụ tạo Workbench hoặc từ cơ sở Workbench Project (còn được gọi là Workbench IDE). Khi BSP đã được tạo, hãy nhập make vxWorks từ dòng lệnh để biên dịch hình ảnh RAM có thể khởi động.

Nếu sử dụng cơ sở Dự án Workbench, bạn có thể tạo một dự án dựa trên BSP mới được tạo, sau đó sử dụng môi trường xây dựng được cung cấp qua IDE để biên dịch BSP.

Trong Workbench 2.5, trình biên dịch diab được hỗ trợ ngoài trình biên dịch GNU. Bạn có thể sửa đổi VxWorks 6.3 BSP Makefile được tạo bởi SDK để sử dụng trình biên dịch diab thay vì trình biên dịch gnu. Tùy biến make có tên TOOLS và đặt giá trị thành sfdiab thay vì sfgnu. Đối với bộ xử lý PowerPC 440 có hệ thống Khởi tạo điểm cứng (FPU), hãy chọn diab hoặc gnu. Nếu sử dụng cơ sở Dự án Workbench, bạn có thể chọn công cụ mong muốn khi dự án được tạo lần đầu tiên.

Tổ chức tài xế

Phần này thảo luận ngắn gọn về cách trình điều khiển Xilinx được biên dịch và liên kết và cuối cùng được sử dụng bởi các cấu hình Workbench để đưa vào hình ảnh VxWorks.

Các trình điều khiển Xilinx được triển khai bằng ngôn ngữ lập trình C và có thể được phân phối giữa một số tệp nguồn không giống như các trình điều khiển VxWorks truyền thống, bao gồm các tệp triển khai và tiêu đề C duy nhất.

Có ba thành phần cho trình điều khiển Xilinx:

- Bao gồm nguồn trình điều khiển
- Triển khai độc lập hệ điều hành
- Triển khai phụ thuộc vào hệ điều hành (tùy chọn)

Bao gồm nguồn trình điều khiển để cập nhật cách trình điều khiển Xilinx được biên dịch. Đối với mọi trình điều khiển, có một tệp có tên procname_drv_dev_version.c. Sử dụng lệnh #include sẽ bao gồm (các) tệp nguồn (*.c) cho mỗi trình điều khiển cho mỗi thiết bị nhất định.

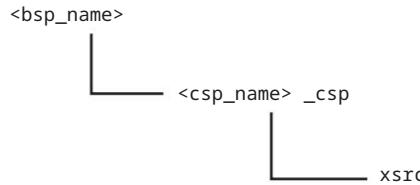
Quá trình này tương tự như cách nguồn của VxWorks sysLib.c #include cho các trình điều khiển được cung cấp bởi Wind River. Các tệp Xilinx không được bao gồm trong sysLib.c, do xung đột không gian tên và các vấn đề về khả năng bảo trì. Nếu tất cả các tệp Xilinx là một phần của một biên dịch thì đơn vị, hàm tĩnh và dữ liệu không còn là riêng tư nữa. Điều này đặt ra những hạn chế đối với trình điều khiển thiết bị và sẽ phụ thuộc tính độc lập của hệ điều hành của chúng.

Phần độc lập với hệ điều hành của trình điều khiển được thiết kế để sử dụng với bất kỳ hệ điều hành hoặc bất kỳ bộ xử lý nào. Nó cung cấp một API sử dụng chức năng của phần cứng bên dưới. Phần phụ thuộc vào hệ điều hành của trình điều khiển sẽ điều chỉnh trình điều khiển để sử dụng với VxWorks. Các ví dụ như vậy là trình điều khiển SIO cho các cổng nối tiếp hoặc trình điều khiển END cho bộ điều hợp ethernet. Không phải tất cả các trình điều khiển đều yêu cầu trình điều khiển phụ thuộc vào hệ điều hành, cũng như không bắt buộc phải bao gồm phần phụ thuộc vào hệ điều hành của trình điều khiển trong bản dựng VxWorks.

Vị trí trình điều khiển thiết bị

BSP được tạo tự động giống với hầu hết các BSP Workbench khác ngoại trừ vị trí của mã trình điều khiển thiết bị. Mã trình điều khiển thiết bị có sẵn được phân phối với IDE Workbench thư ờng nằm trong thư mục vxworks-6.3 / target / src / drv trong thư mục cài đặt Workbench. Mã trình điều khiển thiết bị cho BSP được tạo tự động nằm trong thư mục BSP. Sự sai lệch nhỏ này là do bản chất động của hệ thống nhúng dựa trên FPGA. Trở thành hệ thống nhúng dựa trên FPGA có thể được lập trình lại với IP mới hoặc thay đổi, cấu hình trình điều khiển thiết bị có thể thay đổi, yêu cầu vị trí năng động hơn của các tệp nguồn trình điều khiển thiết bị.

Cây thư mục cho BSP được tạo tự động được hiển thị trong hình sau.



Hình 1: Vị trí thư mục trình điều khiển

Thư mục cấp cao nhất được đặt tên theo tên của phiên bản bộ xử lý trong dự án thiết kế phần cứng. Các tệp nguồn BSP tùy chỉnh nằm trong thư mục này. Có một thư mục con trong thư mục BSP được đặt tên theo phiên bản bộ xử lý với _drv_csp làm hậu tố. Thư mục trình điều khiển chứa hai thư mục con. Thư mục con xsr chứa tất cả các tệp nguồn liên quan đến trình điều khiển thiết bị. Nếu xây dựng từ cơ sở Dự án Workbench, các tệp được tạo trong quá trình xây dựng nằm ở \$ PRJ_DIR / \$ BUILD_SPEC.

Cấu hình

BSP do SDK tạo ra được định cấu hình giống như bất kỳ BSP nào khác của VxWorks 6.3. Có rất ít khả năng cấu hình cho trình điều khiển Xilinx vì phần cứng IP đã được cấu hình trước. Cấu hình duy nhất có sẵn nói chung là liệu trình điều khiển có được bao gồm trong bản dựng VxWorks hay không. Quá trình bao gồm / loại trừ trình điều khiển phụ thuộc vào việc cơ sở Dự án hoặc phuơng pháp dòng lệnh đang được sử dụng để thực hiện các hoạt động cấu hình.

Lưu ý: bao gồm trình điều khiển thiết bị Xilinx không có nghĩa là trình điều khiển được sử dụng tự động. Hầu hết các trình điều khiển với bộ điều hợp VxWorks đều có mã khởi tạo. Trong một số trường hợp, người dùng được yêu cầu thêm các lệnh gọi hàm khởi tạo trình điều khiển thích hợp vào BSP.

Khi sử dụng SDK để tạo BSP, các tệp BSP kết quả có thể chứa các nhận xét "VIỆC CẦN LÀM". Những nhận xét này, nhiều trong số đó bắt nguồn từ mẫu BSP của bộ xử lý PowerPC 405/440 do Wind River cung cấp, đưa ra gợi ý về những gì người dùng phải cung cấp để định cấu hình BSP cho bo mạch đích. Hướng dẫn dành cho nhà phát triển VxWorks BSP và Hướng dẫn dành cho người lập trình ứng dụng VxWorks là các tài nguyên dành cho cấu hình BSP.

Bao gồm / Loại trừ Trình điều khiển Dòng lệnh

Trong BSP, một tập hợp các hằng số (một cho mỗi trình điều khiển) được xác định trong procname_drv_config.h và tuân theo định dạng:

```
#define INCLUDE_ <XDRIVER>
```

Tệp này được bao gồm gần đầu config.h. Theo mặc định, tất cả các trình điều khiển được bao gồm trong bản dựng. Để loại trừ trình điều khiển, hãy thêm dòng sau vào config.h sau khi bao gồm tệp tiêu đề procname_drv_config.h .

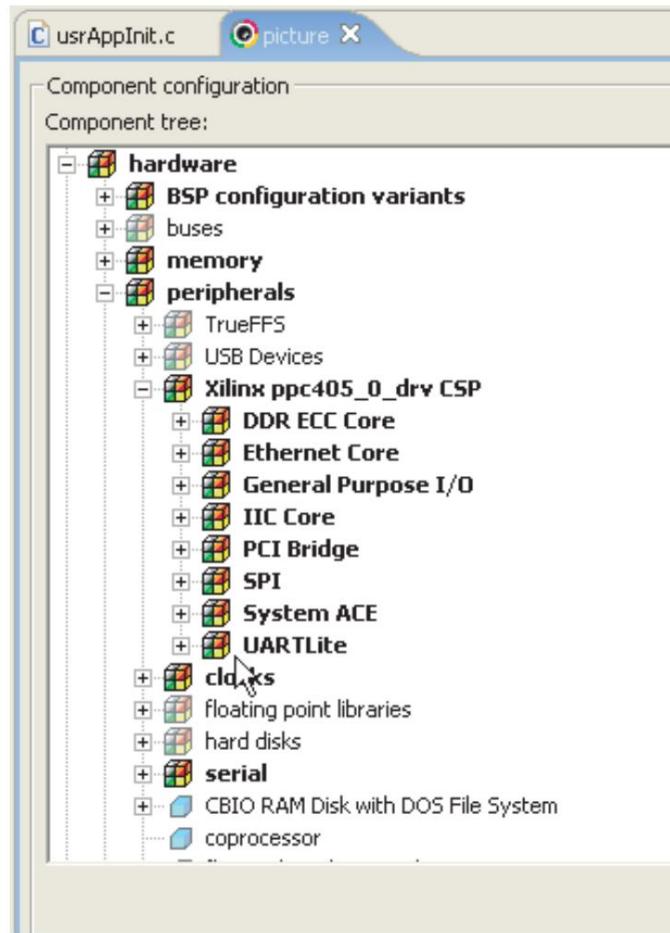
```
#undef INCLUDE_ <XDRIVER>
```

Loại trừ này ngăn trình điều khiển được biên dịch và liên kết với bản dựng. Để cài đặt lại trình điều khiển, hãy xóa dòng #undef khỏi config.h. Một số cần thận là cần thiết cho một số trình điều khiển.

Ví dụ, Ethernet có thể yêu cầu phải có trình điều khiển DMA. Việc hủy xác định trình điều khiển DMA sẽ khiến bản dựng không thành công.

Bao gồm / Loại trừ Trình điều khiển Cơ sở Dự án

Tệp 50 <csp_name>.cdf nằm trong thư mục BSP và được điều chỉnh trong quá trình tạo BSP. Tệp này tích hợp trình điều khiển thiết bị Xilinx vào IDE Workbench. Trình điều khiển thiết bị Xilinx được nối vào IDE tại thư mục con phần cứng / thiết bị ngoại vi của tab thành phần. Dưới đây là các thư mục trình điều khiển thiết bị riêng lẻ. Ví dụ về GUI với trình điều khiển Xilinx được hiển thị trong hình sau. Để thêm hoặc xóa trình điều khiển Xilinx, hãy bao gồm hoặc loại trừ các thành phần trình điều khiển như với bất kỳ thành phần VxWorks nào khác.



Hình 2: Workbench 2.5 Project IDE - VxWorks

Lưu ý: Bất kỳ cấu hình nào được chỉ định trong procname_drv_config.h và config.h đều bị cơ sở dự án ghi đè.

Xây dựng VxWorks

Các BSP được tạo tự động tuân theo các quy ước Workbench tiêu chuẩn khi tạo hình ảnh VxWorks. Tham khảo tài liệu Workbench về cách tạo hình ảnh VxWorks.

Tiện ích mở rộng bản dựng BSP dòng lệnh

Các trình điều khiển Xilinx được biên dịch / liên kết với cùng một chuỗi công cụ mà VxWorks được xây dựng. Các bổ sung nhỏ cho Makefile được yêu cầu để giúp Workbench tìm thấy vị trí của các tệp mã nguồn trình điều khiển.

Project BSP Build Extensions

Có thể thấy số lư ợng tệp mới đư ợc sử dụng để tích hợp trình điều khiển thiết bị Xilinx vào quá trình xây dựng Workbench trong thư mục bsp_name. Như đã nêu trư ớc đó, các tệp này đư ợc tạo tự động bởi SDK. Ngư ời dùng chỉ cần biết rằng các tệp tồn tại. Các tệp này có tiền tố là tên phiên bản của bộ xử lý.

Tích hợp thiết bị

Các thiết bị trong hệ thống nhúng dựa trên FPGA có các mức độ tích hợp khác nhau với hệ điều hành VxWorks. Ngư ời dùng SDK có thể chọn mức độ tích hợp trong hộp thoại Thiết bị ngoại vi đư ợc kết nối của tab Tham số Thư viện / Hệ điều hành. Dưới đây là danh sách các thiết bị hiện đư ợc hỗ trợ và mức độ tích hợp của chúng.

- Một hoặc nhiều thiết bị UART 16450/16550 / Lite có thể đư ợc tích hợp vào giao diện VxWorks Serial I / O (SIO). Điều này làm cho một UART khả dụng cho I / O tệp và printf / stdio. Chỉ một thiết bị UART có thể đư ợc chọn làm bảng điều khiển, nơi I / O tiêu chuẩn (stdin, stdout và stderr) đư ợc huy ứng đến. Một thiết bị UART, khi đư ợc tích hợp vào giao diện SIO, phải có khả năng tạo ra một ngắt. Tham khảo tệp sysSerial.c của BSP để xem chi tiết về tích hợp này.
- Các thiết bị Ethernet MAC 10/100, Ethernet Lite 10/100, Gigabit Ethernet MAC và 10/100/1000 Tri speed MAC có thể đư ợc tích hợp vào giao diện VxWorks Enhanced Network Driver (END). Điều này làm cho thiết bị khả dụng với ngăn xếp mạng VxWorks và do đó là các ứng dụng cáp ỗ cắm. Một thiết bị Ethernet, khi đư ợc tích hợp vào giao diện END, phải có khả năng tạo ra các ngắt. Tham khảo tệp configNet.h và sysNet.c của BSP để xem chi tiết về tích hợp này. Ngoài ra, ngư ời dùng có thể cần sửa đổi các giá trị dòng khởi động mặc định trong config.h để thiết bị Ethernet đư ợc sử dụng làm thiết bị khởi động.
- Bộ điều khiển ngắt có thể đư ợc kết nối với xử lý ngoại lệ VxWorks intLib và chân ngắt không quan trọng bên ngoài PowerPC 405/440. BSP đư ợc tạo hiện không xử lý tích hợp bộ điều khiển ngắt cho chân ngắt quan trọng của PowerPC 405/440, cũng như không hỗ trợ kết nối trực tiếp của một thiết bị ngắt đơn (không phải intc) với bộ xử lý. Tuy nhiên, ngư ời dùng luôn có thể thêm tích hợp này theo cách thủ công trong tệp sysInterrupt.c của BSP.
- Bộ điều khiển System ACE™ có thể đư ợc kết nối với VxWorks như một thiết bị khồi, cho phép ngư ời dùng gắn hệ thống tập tin vào thiết bị CompactFlash đư ợc kết nối với bộ điều khiển System ACE. Ngư ời dùng phải gọi thủ công các hàm BSP để khởi tạo Hệ thống ACE / CompactFlash như một thiết bị khồi và gắn nó vào hệ điều hành DOS. Các chức năng hiện có sẵn cho ngư ời dùng là: sysSystemAceInitFS () và sysSystemAceMount (). Bộ điều khiển ACE hệ thống, khi đư ợc tích hợp vào giao diện thiết bị khồi, phải có khả năng tạo ra ngắt. Tham khảo tệp sysSystemAce.c trong BSP để biết thêm chi tiết. BSP sẽ gắn flash compact dưới dạng phân vùng đĩa DOS FAT bằng tiện ích bổ sung Wind River DosFs2.0. Để các thư viện VxWorks đư ợc đưa vào hình ảnh, các gói sau phải đư ợc định nghĩa trong config.h hoặc bởi Project Facility:

- INCLUDE_DOSFS_MAIN
- INCLUDE_DOSFS_FAT
- INCLUDE_DISK_CACHE
- INCLUDE_DISK_PART
- INCLUDE_DOSFS_DIR_FIXED
- INCLUDE_DOSFS_DIR_VFAT
- INCLUDE_XBD_BLK_DEV
- INCLUDE_XBD_PART_LIB

Theo chương trình, một ứng dụng có thể gán kết hệ thống tệp DOS bằng cách sử dụng các lệnh gọi API sau:

```
TẬP TIN * fp;

sysSystemAceInitFS ();
if (sysSystemAceMount ("/ cf0", 1) != OK)
{
    /* xử lý lỗi */
}

fp = fopen ("/ cf0 / myfile.dat", "r");
```

- Cầu PCI có thể được khởi tạo và có sẵn cho trình điều khiển VxWorks PCI tiêu chuẩn và các chức năng cấu hình. Người dùng được yêu cầu chỉnh sửa config.h và sysBusPci.c Các tệp BSP để điều chỉnh địa chỉ và cấu hình bộ nhớ PCI cho hệ thống đích của chúng. Lưu ý rằng ngắt PCI không được tích hợp tự động vào BSP.
- Bộ điều khiển thiết bị USB có thể được tích hợp vào giao diện bộ điều khiển ngoại vi USB của các thành phần VxWorks BSP. Để kiểm tra bộ điều khiển ngoại vi USB bằng cách sử dụng thành phần giả lập Bộ nhớ chung hiện có của VxWorks, các thay đổi sau phải được thực hiện trong tệp nguồn VxWorks usbTargMsLib.c và trong tệp BSP config.h . Những thay đổi này phải được thực hiện trước khi tạo dự án VxWorks.
 - Sửa đổi giá trị không đổi MS_BULK_OUT_ENDPOINT_NUM thành "2" trong usbTargMsLib.c tập tin. Tệp này nằm trong thư mục WindRiver-Installed-Directory / Vx Works6.3 / target / src / drv / usb / target / .
 - Sau khi sửa đổi, nguồn VxWorks sẽ được biên dịch tại thư mục này. Các lệnh biên dịch cho hệ thống dựa trên bộ xử lý PowerPC 440 là làm cho CPU = PPC32 và đổi với hệ thống dựa trên bộ xử lý PowerPC 405, nó đặt CPU = PPC405.
 - Trình giả lập USB MassStorage sử dụng bộ nhớ cục bộ cho vùng lưu trữ. Người dùng cần cung cấp dung lượng tối thiểu 4MB (sửa đổi giá trị hằng số LOCAL_MEM_SIZE trong tệp config.h thành 0x400000) trong RAM. Mã giả lập MassStorage mô phỏng vùng lưu trữ mặc định là 32k.
- Tất cả các thiết bị khác và trình điều khiển thiết bị liên quan không được tích hợp chặt chẽ vào giao diện VxWorks. Thay vào đó, chúng được tích hợp lỏng lẻo và khả năng truy cập vào các thiết bị này bằng cách truy cập trực tiếp vào trình điều khiển thiết bị được liên kết từ ứng dụng của người dùng.
- Lỗi người dùng và trình điều khiển thiết bị liên quan, nếu được bao gồm trong dự án SDK, được hỗ trợ thông qua luồng tạo BSP. Trình điều khiển thiết bị lỗi của người dùng sẽ được sao chép vào BSP giống như cách sao chép trình điều khiển thiết bị Xilinx. Điều này giả định cấu trúc thư mục của trình điều khiển thiết bị lỗi người dùng khớp với cấu trúc của trình điều khiển thiết bị Xilinx. / Dữ liệu và / xây dựng các thư mục con của trình điều khiển thiết bị phải tồn tại và được định dạng giống như trình điều khiển thiết bị Xilinx. Điều này bao gồm đoạn mã CDF và các tệp xtag trong thư mục con / build / vxworks5_4 . Trình điều khiển thiết bị của người dùng không được tích hợp tự động vào bất kỳ giao diện hệ điều hành nào (ví dụ: SIO), nhưng chúng có sẵn để ứng dụng truy cập trực tiếp.

Sai lệch

Danh sách sau đây tóm tắt sự khác biệt giữa BSP do SDK tạo và BSP truyền thống.

- Một cấu trúc thư mục bổ sung được thêm vào thư mục BSP gốc để chứa trình điều khiển thiết bị các tệp mã nguồn.
 - Để giữ cho BSP có thể xây dựng trong khi duy trì khả năng tương thích với cơ sở Dự án Workbench, một tập hợp các tệp có tên procname_drv_driver_version.c điền vào thư mục BSP chỉ cần # bao gồm mã nguồn từ thư mục con trình điều khiển của BSP. • BSP Makefile đã được sửa đổi để trình biên dịch có thể tìm thấy mã nguồn của trình điều khiển.
- Makefile chứa nhiều thông tin hơn về độ lệch này và ý nghĩa của nó.

- Việc sử dụng SystemACE có thể yêu cầu thay đổi đối với các tệp mã nguồn VxWorks được tìm thấy trong Thư mục phân phối Workbench. Xem phần "[Bootrom với SystemACE làm Thiết bị Khởi động](#)", trang 13.

Hạn chế

BSP được tạo tự động là một điểm khởi đầu tốt, nhưng không nên mong đợi đáp ứng tất cả các yêu cầu mà không cần cấu hình. Do sự phức tạp tiềm ẩn của BSP, nhiều tính năng có thể được bao gồm trong BSP và sự hỗ trợ cần thiết cho các thiết bị bo mạch bên ngoài FPGA, BSP được tạo tự động có thể sẽ yêu cầu cải tiến. Tuy nhiên, BSP được tạo sẽ có thể biên dịch được và sẽ chứa các trình điều khiển thiết bị cần thiết được tích hợp trình bày trong hệ thống nhúng dựa trên FPGA. Một số thiết bị thường được sử dụng cũng được tích hợp hệ điều hành. Các hạn chế cụ thể được liệt kê dưới đây.

- Bộ điều khiển ngắt kết nối với chân ngắt quan trọng của bộ xử lý PowerPC 405/440 không được tích hợp tự động vào sơ đồ ngắt của VxWorks. Hiện chỉ hỗ trợ ngắt bên ngoài.
- Không hỗ trợ phát hiện lỗi xe buýt từ cầu xe buýt hoặc trọng tài.
- Đóng lệnh VxWorks 6.3 BSP mặc định sử dụng trình biên dịch GNU. Người dùng phải thay đổi thủ công Makefile để sử dụng trình biên dịch DIAB hoặc chỉ định trình biên dịch DIAB khi tạo dự án Workbench dựa trên BSP.
- Các dải địa chỉ bộ nhớ có thể cần được điều chỉnh trong config.h để phù hợp với bộ nhớ cụ thể thiết bị và phạm vi địa chỉ của chúng.
- Bộ nhớ đệm của bộ xử lý PowerPC 405/440 bị tắt theo mặc định. Bạn phải kích hoạt thủ công vào bộ nhớ đệm thông qua tệp config.h hoặc menu dự án Workbench.
- Khi SystemACE được thiết lập để tải hình ảnh VxWorks vào RAM bằng JTAG, tắt cả các khởi động đều lạnh (không khởi động ấm). Điều này là do bộ điều khiển System ACE đặt lại bộ xử lý bất cứ khi nào nó thực hiện tải xuống tệp ace. Một tác động của điều này có thể khiến các thông báo ngoại lệ do VxWorks tạo ra không được in trên bảng điều khiển khi hệ thống được khởi động lại do một ngoại lệ trong ISR hoặc sự cố hạt nhân.

Lưu ý: Không thể sử dụng hình ảnh nén với SystemACE. Điều này áp dụng cho các hình ảnh nén tiêu chuẩn được tạo bằng Workbench chẳng hạn như bootrom. Hình ảnh nén không thể được đặt trên SystemACE dưới dạng tệp ace. SystemACE không thể giải nén dữ liệu khi nó ghi vào RAM. Đầu tiên một hình ảnh như vậy sẽ dẫn đến sự cố hệ thống.

- Một bản dựng dòng lệnh không thể khởi tạo mạng khi SystemACE là thiết bị khởi động. Điều này yêu cầu ứng dụng cung cấp mã để khởi tạo mạng khi SystemACE là thiết bị khởi động. Để khắc phục sự cố này, hãy xem thảo luận về \$ WIND_BASE / target / src / config / usrNetwork.c trong "[Bootrom với SystemACE làm Thiết bị khởi động](#)", trang 13.

- Trên bộ xử lý PowerPC 405/440, vector đặt lại ở địa chỉ vật lý 0xFFFFFFFF. Có một cửa sổ thời gian ngắn trong đó bộ xử lý sẽ cố gắng tìm nạp và thực thi lệnh tại địa chỉ này trong khi SystemACE xử lý tệp ace. Bộ xử lý cần được cung cấp một cái gì đó để làm trong thời gian này ngay cả khi đó là một vòng quay:

FFFFFFFC b.

Nếu RAM khởi chiếm phạm vi địa chỉ này, thì người thiết kế tạo dòng bit nên đặt các hướnng dẫn tại đây với tiện ích ELF để chặn RAM được tìm thấy trong Xilinx ISE® công cụ.

- VxBus không được hỗ trợ.

Khởi động VxWorks

Trình tự khởi động VxWorks

Có nhiều biến thể của hình ảnh VxWorks với một số dựa trên RAM, một số dựa trên ROM.

Tùy thuộc vào thiết kế bảng, không phải tất cả những hình ảnh này đều được hỗ trợ. Danh sách sau đây thảo luận về các loại hình ảnh khác nhau:

- **Ảnh nén ROM** - Những ảnh này bắt đầu thực thi trong ROM và giải nén ảnh BSP vào RAM, sau đó chuyển quyền điều khiển sang ảnh đã giải nén trong RAM. Loại hình ảnh này không tương thích với SystemACE vì SystemACE không biết hình ảnh đã được nén và sẽ đặt nó vào RAM tại một địa chỉ sẽ bị thuật toán giải nén đè khi nó bắt đầu. Có thể làm cho loại hình ảnh này hoạt động nếu các sửa đổi được thực hiện đối với các cấu hình Workbench tiêu chuẩn để xử lý tình huống này.
- **Hình ảnh dựa trên RAM** - Những hình ảnh này được tải vào RAM bởi bộ nạp khởi động, SystemACE hoặc một trình giả lập. Những hình ảnh này được hỗ trợ đầy đủ.
- **Hình ảnh dựa trên ROM** - Những hình ảnh này bắt đầu thực thi trong ROM, tự sao chép vào RAM sau đó chuyển việc thực thi vào RAM. Trong các thiết kế với SystemACE làm bộ nạp khởi động, hình ảnh được tự động sao chép vào RAM. Ví dụ BSP được mã hóa thủ công làm ngắn mạch hoạt động sao chép VxWorks để quá trình sao chép không xảy ra nữa sau khi quyền điều khiển được SystemACE chuyển sang RAM (xem romInit.s).
- **Hình ảnh thường trú trong ROM** - Những hình ảnh này bắt đầu thực thi trong ROM, sao chép phần dữ liệu vào RAM và quá trình thực thi vẫn còn trong ROM. Trong các hệ thống chỉ có SystemACE, hình ảnh này không được hỗ trợ. Về mặt lý thuyết, khói RAM có thể được sử dụng làm ROM, tuy nhiên, các bộ phận Virtex-II Pro hiện tại đang được sử dụng trong bảng đánh giá không có khả năng lưu trữ hình ảnh VxWorks có kích thước từ 200KB đến hơn 700KB.

Trình tự khởi động VxWorks

Hình ảnh chuẩn này được thiết kế để một số thiết bị tải xuống dung lưỡng RAM mục tiêu.

Sau khi tải xuống, bộ xử lý được thiết lập để bắt đầu thực thi tại function _sysInit tại địa chỉ RAM_LOW_ADRS. (hàng số này được định nghĩa trong config.h và Makefile). Hầu hết thời gian, thiết bị thực hiện tải xuống sẽ tự động thực hiện việc này vì nó có thể trích xuất điểm nhập từ hình ảnh.

1. **_sysInit:** Hàm hợp ngữ này chạy hết RAM sẽ thực hiện ở mức thấp khởi tạo. Khi hoàn thành, hàm này sẽ thiết lập ngăn xếp ban đầu và gọi hàm C đầu tiên usrInit (). _SysInit nằm trong tệp mã nguồn <bspname> /sysALib.s.
2. **usrInit ():** Hàm C này chạy hết RAM sẽ thiết lập môi trường thời gian chạy C và thực hiện khởi tạo tiền nhân. Nó gọi sysHwInit () (được triển khai trong sysLib.c) để đặt HW ở trạng thái tĩnh. Khi hoàn thành, hàm này sẽ gọi kernelInit () để hiển thị nhân VxWorks. Hàm này sẽ lần lượt gọi usrRoot () làm tác vụ đầu tiên.
3. **usrRoot ():** Thực hiện khởi tạo hậu nhân. Kết nối đồng hồ hệ thống, khởi tạo ngăn xếp TCP / IP, v.v. Nó gọi sysHwInit2 () (được triển khai trong sysLib.c) để đính kèm và kích hoạt ngắt HW. Khi hoàn tất, usrRoot () gọi mã khởi động ứng dụng người dùng usrAppInit () nếu được định cấu hình trong BSP.

Cả usrInit () và usrRoot () đều được thực hiện bởi Wind River. Các tệp mã nguồn mà chúng tồn tại khác nhau tùy thuộc vào dòng lệnh hoặc cơ sở Dự án Workbench đang được sử dụng để biên dịch hệ thống. Trong giao diện dòng lệnh, chúng được triển khai tại \$ WIND_BASE / target / config / all / usrConfig.c. Dưới cơ sở dự án, chúng được duy trì trong thư mục dự án.

Trình tự khởi động "bootrom_ncmp"

Hình ảnh tiêu chuẩn này dựa trên ROM như ng trong thực tế, nó được liên kết để thực thi các địa chỉ RAM. Trong khi thực thi từ ROM, hình ảnh này sử dụng thủ thuật định địa chỉ tương đối để gọi các hàm xử lý các tác vụ trước khi chuyển đến RAM.

1. Bật nguồn. Vector bộ xử lý thành 0xFFFFFFF0 nơi đặt lệnh nhảy chuyển quyền kiểm soát tới bootrom tại địa chỉ _romInit.
2. _romInit : Chức năng hợp ngữ này chạy hết ROM lưu ý rằng đây là khởi động ngoại rời nhảy để bắt đầu. Cả _romInit và start đều nằm trong tệp mã nguồn bspname / romInit.s.
3. start : Chức năng hợp ngữ này chạy hết ROM sẽ thiết lập bộ xử lý, làm mất hiệu lực của bộ nhớ đệm và chuẩn bị cho hệ thống hoạt động trên RAM. Thao tác cuối cùng là gọi hàm C romStart () được thực hiện bởi Wind River và nằm trong tệp mã nguồn \$ WIND_BASE / target / config / all / bootInit.c.
4. romStart () : Hàm C này chạy hết ROM sẽ sao chép VxWorks vào địa chỉ bắt đầu RAM của nó nằm tại RAM_HIGH_ADRS (hàng số này được định nghĩa trong config.h và Makefile). Sau khi sao chép VxWorks, quyền điều khiển được chuyển đến hàm usrInit () trong RAM.
5. Thực hiện theo các bước 2 và 3 của "[Trình tự khởi động VxWorks](#)".

"bootrom_ncmp" Trình tự khởi động với SystemACE

Hình ảnh không chuẩn này tương tự như hình ảnh được thảo luận trong phần trước ngoại trừ việc SystemACE được sử dụng để tải nó. Một số thay đổi phải được thực hiện đối với quá trình khởi động. Có thể tìm thêm thông tin trong phần "[Bootrom với SystemACE làm Thiết bị khởi động](#)," trang 13.

1. Bật nguồn. SystemACE tải hình ảnh vào RAM tại RAM_HIGH_ADRS (hàng số này là được định nghĩa trong config.h và Makefile) và đặt bộ xử lý bắt đầu tìm nạp các hướng dẫn tại địa chỉ _romInit.
2. _romInit : Hàm hợp ngữ này chạy hết RAM lưu ý rằng đây là khởi động ngoại rời nhảy để bắt đầu. Cả _romInit và start đều nằm trong tệp mã nguồn <bspname> / romInit.s.
3. start : Chức năng hợp ngữ này khi hết RAM chỉ cần chuyển sang chức năng _sysInit. Lệnh gọi tới romStart () bị bỏ qua vì SystemACE đã tải bootrom vào địa chỉ RAM đích của nó.

Thực hiện theo các bước 1, 2 và 3 của "[Trình tự khởi động VxWorks](#)", trang 10.

Bootroms

Bootrom là một hình ảnh VxWorks thu nhỏ hoạt động theo cách giống như BIOS của PC.

Chức năng chính của nó là tìm và khởi động một hình ảnh VxWorks đầy đủ. Hình ảnh VxWorks đầy đủ có thể nằm trên đĩa, trong bộ nhớ flash hoặc trên một số máy chủ sử dụng Ethernet. Bootrom phải được biên dịch theo cách mà nó có khả năng truy xuất hình ảnh. Nếu hình ảnh được truy xuất từ mạng Ethernet, thì bootrom phải có ngăn xếp TCP / IP được biên dịch trong, nếu hình ảnh nằm trên đĩa, thì bootrom phải có hỗ trợ truy cập đĩa được biên dịch trong, v.v. Các bootrom không làm gì khác ngoài việc truy xuất và bắt đầu hình ảnh đầy đủ và duy trì một dòng khởi động. Dòng khởi động là một chuỗi văn bản đặt các đặc điểm người dùng nhất định như đích địa chỉ IP khi sử dụng Ethernet và đường dẫn tệp đến hình ảnh VxWorks để khởi động.

Bootrom không phải là một yêu cầu. Chúng thường được sử dụng trong môi trường phát triển sau đó được thay thế bằng hình ảnh VxWorks sẵn xuất.

Tạo Bootroms

Tại trình bao lệnh trong thư mục BSP, hãy phát hành lệnh sau để tạo hình ảnh bootrom không nén (bắt buộc đối với SystemACE):

```
tạo bootrom_uncmp
```

hoặc

```
làm bootrom
```

để tạo ra một hình ảnh nén phù hợp để đặt trong một mảng bộ nhớ flash.

Hiển thị Bootrom

Khi khởi động nguồn, nếu các bootrom hoạt động bình thường, đầu ra tư ờng tự như sau sẽ được nhìn thấy trên cổng nối tiếp của bảng điều khiển:

Khởi động hệ thống VxWorks

Bản quyền 1984-2006 Wind River Systems, Inc.

CPU: ppc405_0 VirtexII Pro PPC405 Phiên bản:
VxWorks 6.3

Phiên bản BSP: 1.2 / 0

Ngày tạo: 11 tháng 8, 2006, 16:40:32

Nhấn phím bất kỳ để dừng tự động khởi động ...
3

[VxWorks Boot]:

Trợ giúp nhập tại dấu nhắc này liệt kê các lệnh có sẵn.

Bootline

Dòng khởi động là một chuỗi văn bản xác định các đặc điểm có thể phục vụ của người dùng như địa chỉ IP của bảng đích và cách tìm hình ảnh VxWorks để khởi động. Dòng khởi động được bootrom duy trì trong thời gian chạy và thường được giữ trong một số vùng lưu trữ không bay hơi (NVRAM) của hệ thống như EEPROM hoặc bộ nhớ flash. Nếu không có NVRAM hoặc xảy ra lỗi khi đọc nó, thì dòng khởi động được mã hóa cứng bằng DEFAULT_BOOT_LINE được xác định trong tệp mã nguồn config.h của BSP. Trong các hệ thống mới mà NVRAM chưa được khởi tạo, dòng khởi động có thể là dữ liệu không xác định.

Dòng khởi động có thể được thay đổi nếu chuỗi đếm ngược tự động khởi động bị gián đoạn bằng cách nhập một ký tự trên cổng nối tiếp bảng điều khiển. Sau đó, lệnh c có thể được sử dụng để chỉnh sửa tư ờng tác dòng khởi động. Nhập p để xem dòng khởi động. Trên ảnh không phải bootrom, dòng khởi động có thể được thay đổi bằng cách nhập lệnh bootChange tại dấu nhắc máy chủ hoặc trình bao dích.

Các trư ờng dòng khởi động được định nghĩa bên dưới:

- boot device: Thiết bị khởi động từ đó. Đây có thể là ethernet hoặc một đĩa cục bộ.

Lưu ý: Khi thay đổi dòng khởi động, số đơn vị có thể được hiển thị thêm vào trư ờng này (xemac0 hoặc sysace = 10) khi nhắc thiết bị khởi động mới. Con số này có thể được bỏ qua.

- số bộ xử lý: Luôn là 0 với các hệ thống bộ xử lý đơn lẻ.

- tên máy chủ: Đặt tên khi cần thiết.

- tên tệp: Hình ảnh VxWorks để khởi động. • inet trên

ethernet (e): Địa chỉ Internet IP của mục tiêu. Nếu không có giao diện mạng, thì trư ờng này có thể được để trống.

- host inet (h): Địa chỉ Internet IP của máy chủ. Nếu không có giao diện mạng, thì điều này trừ ờng có thể đư ợc đẻ trống.
- user (u): Tên ngư ời dùng đẻ truy cập hệ thống tệp máy chủ. Chọn bất kỳ tên nào phù hợp với bạn. FTP của bạn máy chủ phái đư ợc thiết lập đẻ cho phép ngư ời dùng này truy cập vào hệ thống tệp máy chủ.
- Mật khẩu ftp (pw): Mật khẩu bạn chọn đẻ truy cập hệ thống tệp máy chủ. Máy chủ FTP của bạn phái đư ợc thiết lập đẻ cho phép ngư ời dùng này truy cập vào hệ thống tệp máy chủ.
- flags (f): Đẻ biết danh sách các tùy chọn, hãy nhập lệnh trợ giúp tại dấu nhắc [VxWorks Boot]:.
- target name (tn): Đặt tên khi cần thiết. Đặt theo yêu cầu mạng.
- other (o): Tru ờng này hữu ích khi bạn có thiết bị không phái ethernet làm thiết bị khởi động. Trong tru ờng hợp này, VxWorks sē không khởi động mạng khi khởi động. Việc chỉ định thiết bị ethernet ở đây sē kích hoạt thiết bị đó tại thời điểm khởi động với các tham số mạng đư ợc chỉ định trong các tru ờng dòng khởi động khác.
- inet trên bảng nối đa năng (b): Thư ờng đẻ trống nếu hệ thống đích không nằm trên VME hoặc PCI bảng nối đa năng.
- gateway inet (g): Nhập địa chỉ IP vào đây nếu bạn phái đi qua cổng đẻ đến máy tính chủ. Nếu không thì đẻ trống.
- (các) tập lệnh khởi động: Đư ờng dẫn đến tệp trên máy tính chủ chứa các lệnh shell đẻ thực thi sau khi khởi động xong. Đẻ trống nếu không sử dụng tập lệnh. Ví dụ như :
 - Tập lệnh thư ờng trú của SystemACE: /cf0/vxworks/scripts/myscript.txt
 - Tập lệnh thư ờng trú trên máy chủ: c: /temp/myscript.txt

Bootrom với SystemACE làm thiết bị khởi động

Các bootrom hỗ trợ SystemACE có khả năng khởi động hình ảnh VxWorks trực tiếp từ thiết bị Compact Flash dưới dạng tệp elf thông thư ờng hoặc tệp ace.

Các sửa đổi bắt buộc đối với Nguồn VxWorks

Mặc dù SDK có khả năng tạo BSP sử dụng SystemACE trong hình ảnh VxWorks có thể mở và đóng tệp trong hệ thống tệp DOS, nó không thể tạo BSP sử dụng SystemACE làm thiết bị khởi động bootrom. Để sử dụng SystemACE theo cách này, cần phải sửa đổi nhiều đối với mã bootrom do Wind River cung cấp. Wind River cho phép các nhà phát triển BSP thay đổi các tệp mã nguồn Workbench miễn là họ giữ các thay đổi cục bộ đối với BSP và giữ nguyên mã gốc. Hai tệp phái đư ợc sửa đổi từ phiên bản gốc của chúng là:

1. \$ WIND_BASE / target / config / all / bootConfig.c: Tệp này bị ghi đè bằng một tệp tìm thấy trong thư mục bspname. Những thay đổi cần thiết là thêm mã để phân tích cú pháp dòng khởi động đúng cách và khởi tạo và sử dụng SystemACE làm thiết bị khởi động JTAG và DOS. Để ghi đè bootConfig.c mặc định, dòng sau phái đư ợc thêm vào Makefile cho BSP:


```
BOOTCONFIG = ./bootConfig.c.
```
2. \$ WIND_BASE / target / config / comps / src / net / usrlNetBoot.c: Tệp này đư ợc ghi đè bằng tệp đư ợc tìm thấy trong thư mục bspname / net / usrlNetBoot.c. Những thay đổi cần thiết là thêm mã để làm cho VxWorks biết rằng SystemACE là một hệ thống dựa trên đĩa như IDE, SCSI hoặc ổ đĩa mềm. Thay đổi này cho phép một BSP đư ợc xây dựng từ Dự án Workbench đư ợc tải xuống với bootrom hỗ trợ SystemACE đẻ xử lý đúng tru ờng khac của dòng khởi động. Sự tồn tại của tệp đã sửa đổi trong thư mục BSP sē tự động ghi đè tệp gốc.

Cả hai tệp này đều không đư ợc SDK cung cấp vì chúng đư ợc Wind River duy trì ở dạng ban đầu.

Không thể ghi đè tệp thứ ba, \$ WIND_BASE / target / src / config / usrlNetwork.c do kiến trúc của quá trình xây dựng BSP dòng lệnh. Điều này ảnh hứ ờng đến các BSP có khả năng mạng đư ợc xây dựng từ dòng lệnh đư ợc tải xuống với bootrom hỗ trợ SystemACE. Nếu không sửa đổi usrlNetwork.c, các BSP bị ảnh hứ ờng không thể khởi chạy thiết bị mạng của họ và phái dựa vào mã ứng dụng đẻ thực hiện chức năng này.

Nếu người dùng muốn, họ có thể thực hiện thay đổi đối với tệp này trong cài đặt Workbench của họ. Phương pháp này có những bất lợi vì bất kỳ chỉnh sửa nào được thực hiện đối với tệp này đều ảnh hưởng đến tất cả người dùng cài đặt đó và có thể bị mất nếu người dùng nâng cấp hoặc cài đặt lại Workbench. Thay đổi đối với `usrNetwork.c` xảy ra trong hàm `usrNetDevStart()`.

từ:

```
if ((strcmp (params.bootDev, "scsi", 4) == 0) ||
    (strcmp (params.bootDev, "ide", 3) == 0) ||
    (strcmp (params.bootDev, "ata", 3) == 0) ||
    (strcmp (params.bootDev, "fd", 2) == 0) (strcmp      ||
    (params.bootDev, "tffs", 4) == 0))

đến

if ((strcmp (params.bootDev, "scsi", 4) == 0) ||
    (strcmp (params.bootDev, "ide", 3) == 0) ||
    (strcmp (params.bootDev, "ata", 3) == 0) ||
    (strcmp (params.bootDev, "fd", 2) == 0) (strcmp      ||
    (params.bootDev, "sysace", 6) == 0) ||
    (strcmp (params.bootDev, "tffs", 4) == 0))
```

Lưu ý: Bạn có thể tự chịu rủi ro khi chỉnh sửa mã này.

Cấu hình đặc biệt

Chuẩn bị hình ảnh bootrom_ncmp có thể tải xuống bởi SystemACE dưới dạng tệp ace cần có cấu hình đặc biệt. Những thay đổi này là bắt buộc vì bootrom được liên kết để bắt đầu chạy hết thiết bị bộ nhớ không bay hơi, sao chép chính nó vào RAM, sau đó chuyển quyền điều khiển sang bản sao RAM. Các thay đổi sẽ ngăn hoạt động sao chép vì SystemACE đã đặt bootrom vào thiết bị RAM khi đặt lại.

một. Thay đổi định nghĩa của ROM_TEXT_ADRS và ROM_WARM_ADRS thành giá trị tương ứng với RAM_HIGH_ADRS trong cả config.h và Makefile.

b. Thay đổi mã hợp ngữ ở nhãn bắt đầu trong `romInit.s` để chuyển đến chức năng `_sysInit:`

bắt đầu:

```
LOADPTR (r1, _sysInit)
mtlr r1
blrl
```

Định dạng dòng khởi động

Trường thiết bị khởi động 6_3 của dòng khởi động được chỉ định bằng cú pháp sau:

`sysace = số phân vùng`

trong đó số phân vùng là phân vùng để khởi động. Thông thường, giá trị này được đặt thành 1, nhưng một số thiết bị flash nhỏ gọn không có bảng phân vùng và được định dạng như thế chúng là một đĩa mềm lớn. Trong trường hợp này, chỉ định 0 làm số phân vùng. Việc không lấy đúng số phân vùng sẽ dẫn đến lỗi được các thư viện dosFS của VxWork thông báo khi ổ đĩa được gắn kết.

Trường tên tệp của dòng khởi động được đặt tùy thuộc vào cách Hệ thống ACE khởi động hệ thống. Có hai phương pháp khởi động:

- Khởi động từ một tệp thông thường. Điều này tương tự như khởi động mạng trong đó hình ảnh vxWorks nằm trong thiết bị lưu trữ flash nhỏ gọn SystemACE thay vì hệ thống tệp máy chủ. Thiết bị flash nhỏ gọn là một phân vùng hệ thống tệp DOS FAT. Xây dựng vxWorks bằng công cụ Workbench, sao chép tệp hình ảnh kết quả vào thiết bị flash nhỏ gọn bằng đầu đọc thẻ USB hoặc công cụ tương tự, sau đó chỉ định tệp đó trong trường tên tệp của rom khởi động. Tên tệp phải có cú pháp sau:

`cf0 / đường dẫn đến hình ảnh vxWorks`

ở đâu:

- cf0 là điểm gắn kết -

Để ờng dẫn đến hình ảnh VxWorks cung cấp đư ờng dẫn đầy đủ đến hình ảnh VxWorks để khởi động.

Khi đư ợc chỉ định theo cách này, bootrom sẽ gắn ở đĩa dữ ời dạng đĩa đư ợc định dạng FAT, tải tệp vào bộ nhớ và bắt đầu thực thi.

2. Khởi động từ một tệp ace. Tệp ace chỉ có thể chứa HW, SW only hoặc HW + SW. Khi nào khởi động từ một tệp ace với HW, FPGA đư ợc lập trình lại. Nếu tệp ace chứa SW, thì nó đư ợc tải vào bộ nhớ, PC của bộ xử lý đư ợc đặt thành điểm nhập và đư ợc giải phóng để bắt đầu tìm nạp các lệnh. Phương pháp khởi động này linh hoạt ở chố một cấu hình HW hoàn toàn khác có thể đư ợc khởi động từ bootrom VxWorks. Tên tệp phải có cú pháp sau:

`cfgaddr [x]`

trong đó [X] là một số từ 0 đến 7 tương ứng với một trong các thư mục cấu hình đư ợc chỉ định trong cù trú tệp XILINX.SYS trong thư mục gốc của thiết bị flash nhỏ gọn. Nếu [X] bị bỏ qua, thì cấu hình mặc định sẽ đư ợc sử dụng. Cấu hình mặc định thư ờng đư ợc chọn bằng một công tắc xoay đư ợc gắn ở đâu đó trên bảng đánh giá. Bootrom sẽ kích hoạt tài xuống JTAG của tệp ace đư ợc trả đến bởi địa chỉ cấu hình đư ợc chỉ định. Chỉ nên có một tệp duy nhất có phần mở rộng .ace trong thư mục cấu hình đã chọn.

Trong cả hai tru ờng hợp khởi động, nếu thiết bị ethernet đư ợc khởi động khi khởi động VxWorks đã tải xuống, thì tru ờng "khác" của dòng khởi động phải đư ợc sửa đổi để chứa tên của thiết bị mạng.

Bootrom với 10/100 Ethernet (EMAC) làm Thiết bị khởi động

SDK sẽ tạo ra một BSP có khả năng đư ợc xây dựng như một bootrom bằng cách sử dụng EMAC làm thiết bị khởi động. Cấu hình người dùng rất ít đư ợc yêu cầu. Địa chỉ MAC đư ợc mã hóa cứng trong tệp nguồn sysNet.c. BSP có thể đư ợc sử dụng với MAC mặc định miễn là mục tiêu nằm trên một mạng riêng và không có nhiều hơn một mục tiêu trên mạng đó có cùng địa chỉ MAC mặc định. Nếu không, nhà thiết kế nên thay thế MAC này bằng một chức năng để lấy một MAC từ thiết bị nhớ không bay hơi i trên bảng đích của họ.

Để chỉ định EMAC làm thiết bị khởi động trong bootrom, hãy thay đổi tru ờng thiết bị khởi động trong dòng khởi động thành xemac. Nếu chỉ có một EMAC, hãy đặt số đơn vị thành 0.

Bootrom với 1 Gigabit Ethernet (GEMAC) làm Thiết bị khởi động

SDK sẽ tạo một BSP có khả năng đư ợc xây dựng như một bootrom bằng cách sử dụng GEMAC làm thiết bị khởi động. Cấu hình người dùng rất ít đư ợc yêu cầu. Địa chỉ MAC đư ỢC mã hóa cứng trong tệp nguồn sysNet.c. BSP có thể đư ỢC sử dụng với MAC mặc định miễn là mục tiêu nằm trên một mạng riêng và không có nhiều hơn một mục tiêu trên mạng đó có cùng địa chỉ MAC mặc định. Nếu không, nhà thiết kế nên thay thế MAC này bằng một chức năng để lấy một MAC từ thiết bị nhớ không bay hơi i trên bảng đích của họ.

Để chỉ định GEMAC làm thiết bị khởi động trong bootrom, hãy thay đổi tru ờng thiết bị khởi động trong dòng khởi động thành xgemac. Nếu có một GEMAC, hãy đặt số đơn vị thành 0.

Ví dụ về dòng khởi động

Ví dụ sau khởi động từ ethernet bằng cách sử dụng Xilinx xemac làm thiết bị khởi động. Hình ảnh đư ợc khởi động nằm trên hệ thống tệp máy chủ trên ổ C.

```

thiết bị khởi động          : xemac
số đơn vị                  : 0
tên máy chủ số bộ xử lý   : 0
                               : chủ nhà
tên tập tin                 : c: /WindRiver/vxworks-6.3/target/config/ml507/vxWorks
inet trên ethernet (e): 192.168.0.2
máy chủ inet (h)            : 192.168.0.1
người dùng (u)              : xemhost
mật khẩu ftp (pw) cờ (f)   : sao cung đư ợc
tên đích (tn) khác (o)     : 0x0
                               : vxtarget
                               :

```

Ví dụ sau khởi động từ một tập tin cư trú trên phân vùng đầu tiên của thiết bị flash nhỏ gọn SystemACE. Nếu tệp đư ợc khởi động từ / cf0 / vxworks / images / vxWorks sử dụng mạng, thì thiết bị xemac đư ợc khởi tạo.

```

thiết bị khởi động          : sysace = 1
số đơn vị                  : 0
tên máy chủ số bộ xử lý   : 0
                               : chủ nhà
tên tập tin                 : / cf0 / vxworks / images / vxWorks
inet trên ethernet (e): 192.168.0.2
máy chủ inet (h)            : 192.168.0.1
người dùng (u)              : xemhost
mật khẩu ftp (pw) cờ (f)   : sao cung đư ợc
tên đích (tn) khác (o)     : 0x0
                               : vxtarget
                               : xemac

```

Ví dụ sau khởi động từ một cư dân tệp ace trên phân vùng đầu tiên của thiết bị flash compact SystemACE. Vị trí của tệp ace đư ợc đặt bởi XILINX.SYS nằm trong thư mục gốc của thiết bị flash nhỏ gọn. Nếu tệp ace có chứa hình ảnh SW của VxWorks sử dụng mạng, thì thiết bị xemac đư ợc khởi tạo cho hình ảnh đó.

```

thiết bị khởi động          : sysace = 1
số đơn vị                  : 0
tên máy chủ số bộ xử lý   : 0
                               : chủ nhà
tên tập tin                 : cfgaddr2
inet trên ethernet (e): 192.168.0.2
máy chủ inet (h)            : 192.168.0.1
người dùng (u)              : xemhost
mật khẩu ftp (pw) cờ (f)   : sao cung đư ợc
tên đích (tn) khác (o)     : 0x0
                               : vxtarget
                               : xemac

```

Bộ nhớ đệm

Hướng dẫn và bộ nhớ đệm dữ liệu được quản lý bởi các thư viện độc quyền của VxWorks. Chúng được kích hoạt bằng cách sửa đổi các hằng số sau trong config.h hoặc bằng cách sử dụng cơ sở Dự án Workbench để thay đổi các hằng số có cùng tên:

- INCLUDE_CACHE_SUPPORT: Nếu được xác định, các thư viện bộ nhớ VxWorks được liên kết với hình ảnh. Nếu bộ nhớ đệm không được mong muốn, thì hãy # hoàn thiện hằng số này.
- USER_I_CACHE_ENABLE: Nếu được định nghĩa, VxWorks sẽ kích hoạt bộ đệm chỉ lệnh lúc khởi động. Yêu cầu INCLUDE_CACHE_SUPPORT được xác định để có bất kỳ tác dụng nào.
- USER_D_CACHE_ENABLE: Nếu được định nghĩa, VxWorks sẽ bật bộ đệm dữ liệu vào thời gian khởi động. Yêu cầu INCLUDE_CACHE_SUPPORT được xác định để có bất kỳ tác dụng nào.

MMU

Nếu MMU được bật, thì điều khiển bộ đệm được thảo luận trong phần trưc có thể không có bất kỳ tác dụng nào. MMU được quản lý bởi các thư viện độc quyền của VxWorks nhưng thiết lập ban đầu được xác định trong BSP. Để bật MMU, hằng số INCLUDE_MMU_BASIC phải được xác định trong config.h hoặc bằng cách sử dụng Project Facility. Hằng số USER_D_MMU_ENABLE và USER_I_MMU_ENABLE kiểm soát liệu MMU hướng dẫn và / hoặc dữ liệu có được sử dụng hay không.

VxWorks khởi tạo MMU dựa trên dữ liệu trong cấu trúc sysPhysMemDesc được định nghĩa trong sysCache.c. Bộ nhớ dành riêng cho người dùng và ED&R (khi INCLUDE_EDR_PM được bật) bộ nhớ dành riêng được bao gồm trong bảng này. Trong số những thứ khác, bảng này định cấu hình các vùng bộ nhớ với các thuộc tính sau:

- Cho phép thực hiện lệnh hay không
- Cho phép ghi dữ liệu
- Thuộc tính hướng dẫn và dữ liệu có khả năng lưu vào bộ nhớ cache
- Phần bù dịch được sử dụng để tạo địa chỉ ảo.

Khi VxWorks khởi tạo MMU, nó sẽ lấy các định nghĩa từ sysPhysMemDesc và tạo các mục nhập bảng trang (PTE) trong RAM. Mỗi PTE mô tả 4KB vùng bộ nhớ (mặc dù bộ xử lý có khả năng đại diện lên đến 16MB cho mỗi PTE). Hãy lưu ý rằng việc chỉ định các vùng bộ nhớ lớn sử dụng lượng RAM đáng kể để lưu trữ các PTE. Để ánh xạ 4MB không gian bộ nhớ liền kề, cần 8KB RAM để lưu trữ các PTE.

Để tăng hiệu suất với gói MMU cơ bản của VxWorks cho bộ xử lý PowerPC 405/440, bạn có thể không bật MMU hướng dẫn và dựa vào cài đặt điều khiển bộ đệm trong thanh ghi ICCR. Chiến lược này có thể giảm đáng kể số lượng lỗi trang trong khi vẫn giữ các hướng dẫn trong bộ nhớ cache. Cài đặt ban đầu của ICCR được xác định trong tệp tiêu đề bspname.h.

Nếu không bật MMU, các quy tắc sau áp dụng cho việc định cấu hình các thuộc tính truy cập bộ nhớ và bộ nhớ đệm:

- Không có bản dịch địa chỉ, tất cả các địa chỉ hiệu quả là vật lý.
- Mức độ chi tiết của kiểm soát bộ nhớ cache là 128MB.
- Thuộc tính được bảo vệ chỉ áp dụng cho các lần tìm nạp lệnh suy đoán trên PowerPC 405 bộ xử lý.

FPU

Đơn vị dấu phẩy động cứng (FPU) được hỗ trợ cho hệ thống bộ xử lý PowerPC 440. Để bật đơn vị dấu phẩy động cứng, hãy chọn diab hoặc gnu trong CÔNG CỤ Tạo tệp BSP đã tạo. Để tắt đơn vị dấu phẩy động cứng, hãy chọn sfdiab hoặc sfgnu trong CÔNG CỤ tạo biến số của Makefile.



Tự động tạo gói hỗ trợ bo mạch Wind River VxWorks 6.5

Bản tóm tắt

Tài liệu này mô tả việc tạo tự động Gói hỗ trợ bảng làm việc (BSP) bằng cách sử dụng Bộ phát triển phần mềm Xilinx (SDK) (1). Tài liệu bao gồm những điều sau đây các phần.

- “[Tổng quan](#)”
- “[Tạo VxWorks 6.5 BSP](#)”
- “[VxWorks 6.5 BSP](#)”
- “[Khởi động VxWorks](#)”

Tổng quan

Một trong những hoạt động phát triển hệ thống nhúng chính là phát triển BSP. Việc tạo ra một BSP có thể là một quá trình kéo dài và tẻ nhạt phải phát sinh khi có sự thay đổi trong tổ hợp bộ vi xử lý bao gồm bộ xử lý và các thiết bị ngoại vi liên quan. Mặc dù việc quản lý những thay đổi này áp dụng cho bất kỳ dự án nào dựa trên bộ vi xử lý, nhưng giờ đây những thay đổi có thể được thực hiện nhanh hơn với sự ra đời của phần cứng Hệ thống trên chip (SoC) có thể lập trình được.

Tài liệu này mô tả việc tạo tự động VxWorks 6.5 BSP tùy chỉnh cho bộ vi xử lý IBM PowerPC™ 405/440 và các thiết bị ngoại vi của nó như được định nghĩa trong Xilinx FPGA. BSP được tạo tự động cho phép các nhà thiết kế hệ thống nhúng:

- Giảm đáng kể các chu kỳ phát triển, do đó làm giảm thời gian đưa ra thị trường
- Tạo BSP tùy chỉnh để phù hợp với phần cứng và ứng dụng
- Loại bỏ lỗi thiết kế BSP (được tạo tự động dựa trên các thành phần được chứng nhận)
- Cho phép phát triển phần mềm ứng dụng bằng cách loại bỏ thời gian chờ phát triển BSP

VxWorks 6.5 BSP được tạo từ SDK, một IDE được phân phối như một phần của Bộ phát triển nhúng Xilinx (EDK) hoặc có sẵn riêng từ Xilinx. SDK được sử dụng để tạo các ứng dụng phần mềm cho các hệ thống nhúng trong Xilinx FPGA. VxWorks BSP chứa tất cả phần mềm hỗ trợ cần thiết cho hệ thống, bao gồm mã khởi động, trình điều khiển thiết bị và RTOS

1. Xilinx SDK được sử dụng làm môi trường phát triển phần mềm chính cho người dùng Xilinx Embedded Development Kit (EDK) kể từ EDK 11.1i. Khả năng phát triển phần mềm của Xilinx Platform Studio (XPS) hiện không được dùng nữa và sẽ bị xóa khỏi XPS trong các bản phát hành sau. Các luồng được mô tả trong tài liệu này liên quan đến SDK, mặc dù chúng vẫn có thể áp dụng chung cho XPS trong khi các tính năng đó vẫn còn trong công cụ.

© 2006-2009 Xilinx, Inc. XILINX, biểu trưng Xilinx, Virtex, Spartan, ISE và các nhãn hiệu được chỉ định khác có trong tài liệu này là nhãn hiệu của Xilinx tại Hoa Kỳ và các quốc gia khác. Tất cả các nhãn hiệu khác là tài sản của chủ sở hữu tương ứng của họ.

Xilinx sẽ tiết lộ hướng dẫn sử dụng, hướng dẫn sử dụng, ghi chú phát hành và / hoặc thông số kỹ thuật này ("Tài liệu") cho bạn chỉ để sử dụng trong việc phát triển các thiết kế để vận hành với các thiết bị phần cứng của Xilinx. Bạn không được phép sao chép, phân phối, tái xuất bản, tải xuống, hiển thị, đăng hoặc truyền Tài liệu dưới bất kỳ hình thức nào hoặc bằng bất kỳ phương tiện nào bao gồm nhưng không giới hạn ở Xilinx. Xilinx từ chối rõ ràng bất kỳ trách nhiệm pháp lý nào phát sinh từ việc bạn sử dụng Tài liệu. Xilinx bảo lưu quyền, theo quyết định riêng của mình, thay đổi Tài liệu mà không cần thông báo bất kỳ lúc nào. Xilinx không có nghĩa vụ phải sửa bất kỳ lỗi nào có trong Tài liệu hoặc thông báo cho bạn về bất kỳ chỉnh sửa hoặc cập nhật nào. Xilinx từ chối rõ ràng bất kỳ trách nhiệm pháp lý nào liên quan đến hỗ trợ kỹ thuật hoặc hỗ trợ có thể được cung cấp cho bạn liên quan đến Thông tin.

TÀI LIỆU ĐƯỢC CÔNG BỐ CHO BẠN "NGUYÊN TRANG" KHÔNG ĐƯỢC BẢO HÀNH BẤT KỲ HÌNH THỨC NÀO. XILINX KHÔNG CÓ BẢO ĐÀM KHÁC, DÙ THỂ HIỆN RỘ RẰNG, NGƯỜI HOẶC TRUYỀN NGHĨA, LIÊN QUAN ĐẾN TÀI LIỆU, BAO GỒM BẤT KỲ BẢO ĐÀM NÀO VỀ TÍNH KHẢ NĂNG, PHÙ HỢP VỚI MỤC ĐÍCH CỤ THỂ HOẶC KHÔNG BẢO ĐÀM BẢN QUYỀN CỦA BÊN THỨ BA. TRONG MỌI TRƯỜNG HỢP XILINX SẼ CHỊU TRÁCH NHIỆM PHÁP LÝ ĐỐI VỚI BẤT KỲ THIỆT HẠI HẬU QUẢ, ĐÚNG, BẤT CỨ, ĐẶC BIỆT HOẶC SỰ CỐ NÀO, BAO GỒM BẤT CỨ MẤT DỮ LIỆU HOẶC MẤT LỢI NHUẬN NÀO PHÁT SINH TỪ VIỆC BẠN SỬ DỤNG TÀI LIỆU.

khởi tạo. BSP đư ợc tùy chỉnh dựa trên các thiết bị ngoại vi đư ợc ngư ời dùng chọn và cấu hình cho hệ thống nhúng dựa trên FPGA.

Các nhà thiết kế BSP có kinh nghiệm nên dễ dàng tích hợp BSP đã tạo vào hệ thống mục tiêu của họ. Ngư ợc lại, những ngư ời dùng ít kinh nghiệm hơn có thể gặp khó khăn vì mặc dù SDK có thể tạo BSP hoạt động cho một tập hợp phần cứng IP nhất định, như ng sẽ luôn có một số cấu hình và điều chỉnh bổ sung cần thiết để tạo ra hiệu suất tốt nhất từ hệ thống mục tiêu. Ngư ời dùng nên có sẵn Hướng dẫn dành cho nhà phát triển Wind River VxWorks BSP và Hướng dẫn dành cho lập trình viên ứng dụng VxWorks hoặc xem xét các lớp Wind River trên thiết kế BSP, có sẵn với một khoản chi phí bổ sung.

Yêu cầu

Bộ phát triển Wind River Workbench 2.6.1 phải đư ợc cài đặt trên máy tính chủ.

Vì SDK tạo ra các BSP có thể di chuyển lại đư ợc biên dịch và định cấu hình bên ngoài môi trường SDK, nên máy tính chủ không cần phải cài đặt cả Xilinx SDK và Workbench.

Định nghĩa thư viện bộ vi xử lý

SDK hỗ trợ giao diện trinh cắm thêm cho các thư viện và hệ điều hành của bên thứ 3 thông qua giao diện Định nghĩa Thư viện Bộ vi xử lý (MLD), do đó cho phép các nhà cung cấp bên thứ 3 cung cấp phần mềm của họ cho ngư ời dùng SDK. Ngoài ra, nó cung cấp cho các nhà cung cấp một phư ơng tiện để điều chỉnh thư viện hoặc BSP của họ cho phù hợp với hệ thống nhúng dựa trên FPGA đư ợc tạo trong các công cụ Xilinx. Bởi vì hệ thống có thể thay đổi dễ dàng, khả năng này rất quan trọng trong việc hỗ trợ đúng cách các hệ thống nhúng trong FPGA.

Xilinx phát triển và duy trì VxWorks 6.5 MLD trong các bản phát hành SDK của mình. MLD đư ợc sử dụng để tự động tạo VxWorks 6.5 BSP.

Phư ơng pháp Tiếp cận Dựa trên Mẫu

Một bộ tệp mẫu VxWorks 6.5 BSP đư ợc phát hành cùng với SDK. Các tệp mẫu này đư ợc sử dụng trong quá trình tạo BSP tự động và các sửa đổi thích hợp đư ợc thực hiện dựa trên cấu trúc của hệ thống nhúng dựa trên FPGA.

Các tệp mẫu này có thể đư ợc sử dụng làm tài liệu tham khảo để xây dựng BSP từ đầu nếu ngư ời dùng chọn không tự động tạo BSP.

Trình điều khiển thiết bị

Một tập hợp các tệp nguồn trình điều khiển thiết bị đư ợc phát hành cùng với SDK và nằm trong thư mục cài đặt. Trong quá trình tạo BSP tùy chỉnh, mã nguồn trình điều khiển thiết bị đư ợc sao chép từ thư mục cài đặt này sang thư mục BSP. Chỉ mã nguồn liên quan đến các thiết bị đư ợc tích hợp trong hệ thống nhúng dựa trên FPGA mới đư ợc sao chép. Bản sao này cung cấp cho ngư ời dùng một thư mục BSP độc lập, độc lập có thể đư ợc sửa đổi hoặc di dời. Nếu ngư ời dùng thực hiện các thay đổi đối với mã nguồn trình điều khiển thiết bị cho BSP này, sau đó muốn hoàn tác các thay đổi, công cụ SDK có thể đư ợc sử dụng để tạo lại BSP. Tại thời điểm đó, các tệp nguồn của trình điều khiển thiết bị đư ợc mở lại từ thư mục cài đặt vào BSP.

Tạo VxWorks 6.5 BSP

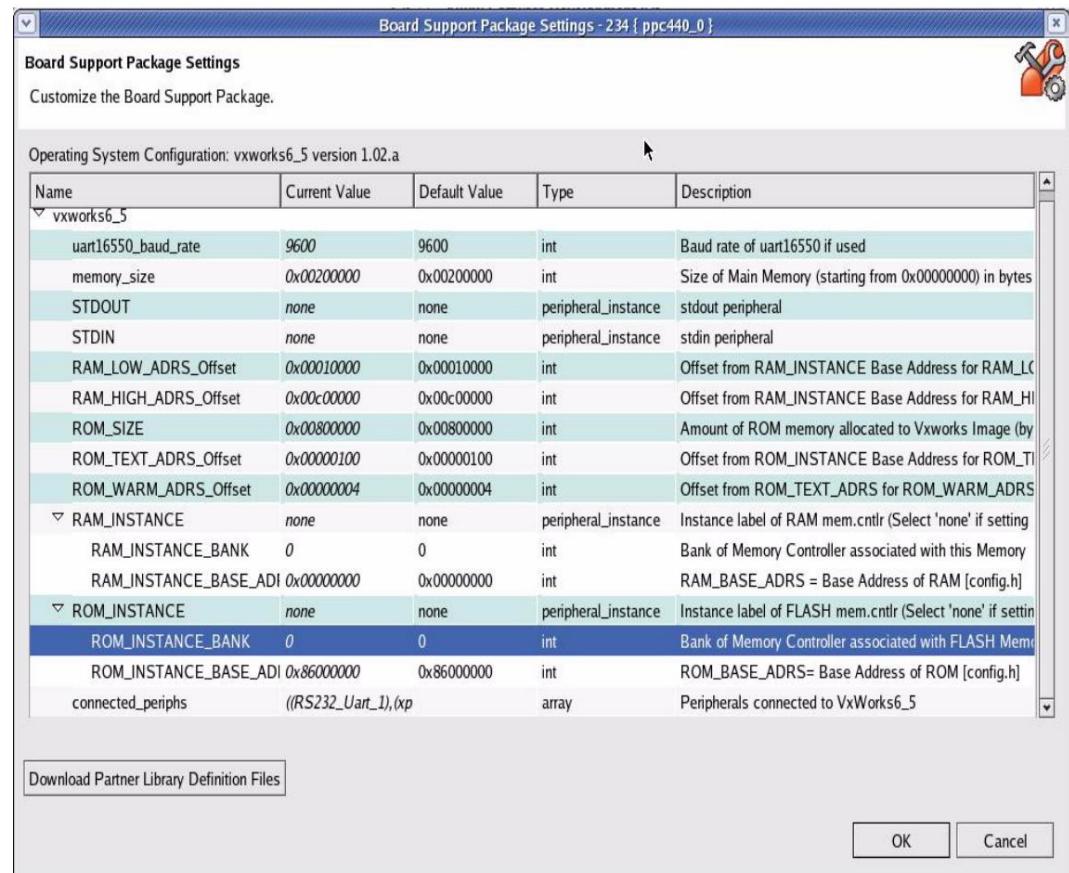
Sử dụng SDK

SDK có sẵn dưới dạng công cụ đư ợc cài đặt riêng hoặc trong EDK và là môi trường phát triển phần mềm để phát triển phần mềm nhúng xung quanh các hệ thống nhúng dựa trên Xilinx PowerPC 405/440 hoặc MicroBlaze™. Phần này mô tả các bước cần thiết để tạo BSP VxWorks 6.5 sử dụng SDK. Các bước này có thể áp dụng khi sử dụng công cụ The Xilinx 11.1i trở lên.

Giả định rằng một thiết kế phần cứng hợp lệ đã đư ợc tạo và xuất sang SDK, và SDK đã đư ợc mở và trỏ đến thiết kế phần cứng.

1. Sử dụng File-> New, tạo một dự án Gói hỗ trợ hội đồng quản trị mới. Trong hộp thoại, nhập tên dự án và chọn vxworks6_5 làm Loại gói hỗ trợ hội đồng quản trị. Lưu ý rằng SDK có thể quản lý nhiều dự án thuộc các loại BSP khác nhau.

Các bước còn lại liên quan đến hộp thoại Tools-> Board Support Package Settings ..., hộp thoại này sẽ tự động được hiển thị sau bước trên.



Hình 1: Cài đặt Gói hỗ trợ bo mạch

2. Định cấu hình thiết bị giao diện điều khiển VxWorks

Nếu một thiết bị nối tiếp như Uart được định nghĩa làm bảng điều khiển VxWorks, hãy chọn hoặc nhập tên phiên bản của thiết bị nối tiếp làm thiết bị ngoại vi STDIN / STDOUT trong hộp thoại Cài đặt Gói Hỗ trợ Bảng. Điều quan trọng là phải nhập cùng một thiết bị cho cả STDIN và STDOUT. Hiện tại, chỉ có các thiết bị Uart 16550/16450 và UartLite được hỗ trợ làm thiết bị VxWorks console.

3. Tích hợp trình điều khiển thiết bị

một. Kết nối với VxWorks

Có một hộp thoại kết nối_periph có sẵn trong hộp thoại Thiết đặt gói hỗ trợ hội đồng quản trị Các thiết bị ngoại vi đã được điều chỉnh để thuận tiện cho người dùng. Sử dụng hộp thoại này để sửa đổi các thiết bị ngoại vi đó để được tích hợp chặt chẽ với Hệ điều hành, bao gồm thiết bị đã được chọn làm thiết bị ngoại vi STDIN / STDOUT. Xem phần "[Tích hợp thiết bị](#)" để biết thêm chi tiết về tích hợp chặt chẽ các thiết bị.

b. Dung lư ợng bộ nhớ

Trường này được sử dụng để định cấu hình BSP để phù hợp với kích thước bộ nhớ phần cứng thực tế trên bo mạch của bạn.

c. Uart16550_baud_rate

Trường này được sử dụng để nhập tốc độ truyền cho các dự án có lõi UART 16550/16450. Không cần thiết phải nhập giá trị ở đây cho các dự án có lõi UART Lite vì tốc độ truyền được đặt cho UART Lite tại thời điểm xây dựng phần cứng.

d. RAM_INSTANCE

Đây là menu thả xuống để chọn phiên bản ngoại vi sẽ được sử dụng làm RAM trong BSP. Ngân hàng bộ nhớ trựòng con và địa chỉ cơ sở của RAM dưới RAM_INSTANCE phải được định cấu hình để phù hợp với cài đặt phần cứng thực tế.

e. ROM_INSTANCE

Đây là menu thả xuống để chọn phiên bản ngoại vi sẽ được sử dụng làm ROM trong BSP. FLASH là thiết bị ROM duy nhất được hỗ trợ trên bảng Đánh giá Xilinx. Nếu không có ROM trong hệ thống, người dùng có thể để cài đặt mặc định tức là không có. Nếu có FLASH trong hệ thống, ngân hàng bộ nhớ trựòng con và địa chỉ cơ sở của ROM trong ROM_INSTANCE phải được định cấu hình để phù hợp với cài đặt phần cứng thực tế.

f. RAM_LOW_ADRS_OFFSET

Trường này được sử dụng để nhập địa chỉ offset cho địa chỉ cơ sở RAM để lấy địa chỉ RAM cho vxWorks được sử dụng trong BSP và phải được cấu hình để phù hợp với cài đặt hệ thống phần cứng. g. RAM_HIGH_ADRS_OFFSET

Trường này được sử dụng để nhập địa chỉ offset cho địa chỉ cơ sở RAM để lấy địa chỉ RAM được sử dụng trong BSP cho rom khởi động và phải được định cấu hình để phù hợp với cài đặt hệ thống phần cứng.

h. ROM_LOW_ADRS_OFFSET

Trường này được sử dụng để nhập địa chỉ offset cho địa chỉ cơ sở ROM để lấy địa chỉ bắt đầu FLASH được sử dụng trong BSP và phải được cấu hình để phù hợp với cài đặt hệ thống phần cứng.

i. ROM_HIGH_ADRS_OFFSET

Trường này được sử dụng để nhập bù địa chỉ cho địa chỉ cơ sở ROM để lấy địa chỉ cuối FLASH được sử dụng trong BSP và phải được cấu hình để phù hợp với cài đặt hệ thống phần cứng.

j. ROM_SIZE

Trường này được sử dụng để định cấu hình BSP và phải khớp với cài đặt hệ thống phần cứng thực tế.

k. ROM_TEXT_ADRS_OFFSET

Trường này được sử dụng để nhập độ lệch địa chỉ cho địa chỉ cơ sở ROM để lấy địa chỉ bắt đầu phần văn bản được sử dụng trong BSP và phải được định cấu hình để phù hợp với cài đặt hệ thống phần cứng.

l. ROM_WARM_ADRS_OFFSET

Trường này được sử dụng để nhập bù địa chỉ cho địa chỉ cơ sở ROM để lấy địa chỉ nhập khởi động lại ấm được sử dụng trong BSP và phải được định cấu hình để phù hợp với cài đặt hệ thống phần cứng.

4. Tạo VxWorks 6.5 BSP

Bấm OK trên hộp thoại Cài đặt Gói Hỗ trợ Ban ... để tạo BSP. Đầu ra của lệnh gọi này được hiển thị trong cửa sổ bảng điều khiển SDK. Sau khi hoàn tất, VxWorks 6.5 BSP sẽ tồn tại trong không gian làm việc SDK của bạn, dưới tên thư mục dự án mà bạn đã tạo ở bước 1 ở trên, trong thư mục con phiên bản PowerPC 405/440. Ví dụ, nếu

trong thiết kế phần cứng, người dùng đã đặt tên cho phiên bản PowerPC 405 là myppc405, BSP sẽ nằm tại <SDK workspace> / <SDK project name> / myppc405 / bsp_ppc405.

Sao lưu

Để tránh việc người dùng vô tình làm mất các thay đổi đối với tệp nguồn BSP, các tệp hiện có trong vị trí thư mục của BSP sẽ được sao chép vào thư mục sao lưu trữ khi bị ghi đè. Thư mục sao lưu nằm trong thư mục BSP và được đặt tên là backup <timestamp>, trong đó <timestamp> đại diện cho ngày và giờ hiện tại. Vì BSP được tạo bởi SDK có thể định vị lại, bạn nên di chuyển BSP từ thư mục dự án SDK sang thư mục phát triển BSP thích hợp ngay khi nền tảng phần cứng ổn định.

VxWorks 6.5 BSP

Phần này mô tả đầu ra VxWorks 6.5 BSP của SDK. Giả định rằng người đọc đã quen thuộc với Workbench 2.6.1 IDE của Wind River.

BSP được tạo tự động được tích hợp vào Workbench IDE. BSP có thể được biên dịch từ dòng lệnh bằng cách sử dụng các công cụ tạo Workbench hoặc từ cơ sở Workbench Project (còn được gọi là Workbench IDE). Khi BSP đã được tạo, người dùng có thể nhập lệnh make vxWorks từ dòng lệnh để biên dịch hình ảnh RAM có thể khởi động. Điều này giả định rằng môi trường Workbench đã được thiết lập trước đó, có thể được thực hiện thông qua dòng lệnh bằng cách sử dụng tiện ích môi trường wren Wind River trên nền tảng Windows. Xem Hướng dẫn người dùng dòng lệnh Wind River Workbench: Tạo lớp vỏ phát triển với wren để biết thêm thông tin về cách sử dụng các tiện ích dòng lệnh. Nếu sử dụng cơ sở Dự án Workbench, người dùng có thể tạo một dự án dựa trên BSP mới được tạo, sau đó sử dụng môi trường xây dựng được cung cấp thông qua IDE để biên dịch BSP.

Trong Workbench 2.6.1, trình biên dịch diab được hỗ trợ ngoài trình biên dịch gnu . VxWorks 6.5 BSP được tạo bởi SDK có một Makefile có thể được sửa đổi bởi người dùng dòng lệnh để sử dụng trình biên dịch diab thay vì trình biên dịch gnu. Tìm biến make có tên TOOLS và đặt giá trị thành sfdiab thay vì sfgnu. Đổi với PPC440 với hệ thống đơn vị dấu phẩy động cứng (FPU), vui lòng chọn diab hoặc gnu. Nếu sử dụng cơ sở Dự án Workbench, người dùng có thể chọn công cụ mong muốn khi dự án được tạo lần đầu tiên.

Tổ chức tài xế

Phần này thảo luận ngắn gọn về cách trình điều khiển Xilinx được biên dịch và liên kết và cuối cùng được sử dụng bởi các cấu hình Workbench để đưa vào hình ảnh VxWorks.

Các trình điều khiển Xilinx được triển khai bằng ngôn ngữ lập trình C và có thể được phân phối giữa một số tệp nguồn không giống như các trình điều khiển VxWorks truyền thống, bao gồm các tệp triển khai và tiêu đề C duy nhất.

Có tám ba thành phần cho trình điều khiển Xilinx:

- Bao gồm nguồn trình điều khiển.
- Triển khai độc lập hệ điều hành
- Triển khai phụ thuộc vào hệ điều hành (tùy chọn).

Bao gồm nguồn trình điều khiển để cập đến cách trình điều khiển Xilinx được biên dịch. Đối với mọi trình điều khiển, có một tệp có tên là <procname>_drv_<dev>_<version>.c. Sử dụng lệnh #include sẽ bao gồm (các) tệp nguồn (*.c) cho mỗi trình điều khiển cho mỗi thiết bị nhất định.

Quá trình này tương tự như cách nguồn của VxWorks sysLib.c # include cho các trình điều khiển được cung cấp bởi Wind River. Lý do tại sao các tệp Xilinx không đơn giản được đưa vào sysLib.c, cũng như phần còn lại của các trình điều khiển, là do xung đột không gian tên và các vấn đề về khả năng bảo trì. Nếu tắt cả các tệp Xilinx là một phần của một đơn vị biên dịch duy nhất, thì các hàm và dữ liệu tĩnh không còn là riêng tư nữa. Điều này đặt ra những hạn chế đối với trình điều khiển thiết bị và sẽ phủ nhận tính độc lập của hệ điều hành của chúng.

Phần đặc lập hệ điều hành của trình điều khiển được thiết kế để sử dụng với bất kỳ hệ điều hành hoặc bất kỳ bộ xử lý nào. Nó cung cấp một API sử dụng chức năng của phần cứng bên dưới. Phần phụ thuộc vào hệ điều hành của trình điều khiển sẽ điều chỉnh trình điều khiển để sử dụng với VxWorks. Các ví dụ như vậy là trình điều khiển SIO cho các cổng nối tiếp hoặc trình điều khiển END cho bộ điều hợp ethernet. Không phải tất cả các trình điều khiển đều yêu cầu trình điều khiển phụ thuộc vào hệ điều hành, cũng như không bắt buộc phải bao gồm phần phụ thuộc vào hệ điều hành của trình điều khiển trong bản dựng VxWorks.

Vị trí trình điều khiển thiết bị

BSP được tạo tự động giống với hầu hết các BSP Workbench khác ngoại trừ việc đặt mã trình điều khiển thiết bị. Mã trình điều khiển thiết bị có sẵn được phân phối với IDE Workbench thư ờng nằm trong thư mục vxworks-6.5 / target / src / drv trong thư mục cài đặt Workbench. Mã trình điều khiển thiết bị cho BSP được tạo tự động nằm trong chính thư mục BSP. Sự sai lệch nhỏ này là do bản chất động của hệ thống nhúng dựa trên FPGA. Vì hệ thống nhúng dựa trên FPGA có thể được lập trình lại với IP mới hoặc thay đổi, cấu hình trình điều khiển thiết bị có thể thay đổi, yêu cầu vị trí năng động hơn của các tệp nguồn trình điều khiển thiết bị.

Cây thư mục cho BSP được tạo tự động là <bsp_name> / <csp_name> _csp / xsrsrc.

Thư mục cấp cao nhất được đặt tên theo tên của phiên bản bộ xử lý trong dự án thiết kế phần cứng. Các tệp nguồn BSP tùy chỉnh nằm trong thư mục này. Có một thư mục con trong thư mục BSP được đặt tên theo phiên bản bộ xử lý với _drv_csp như một hậu tố. Thư mục trình điều khiển chứa hai thư mục con. Thư mục con xsrsrc chứa tất cả các tệp nguồn liên quan đến trình điều khiển thiết bị. Nếu xây dựng từ cơ sở Dự án Workbench, các tệp được tạo trong quá trình xây dựng nằm ở \$ PRJ_DIR / \$ BUILD_SPEC.

Cấu hình

BSP do SDK tạo ra được định cấu hình giống như bất kỳ BSP VxWorks 6.5 nào khác. Có rất ít khả năng cấu hình cho trình điều khiển Xilinx vì phần cứng IP đã được định cấu hình trước trong hầu hết các trang hợp. Cấu hình duy nhất có sẵn nói chung là liệu trình điều khiển có được bao gồm trong bản dựng VxWorks hay không. Quá trình bao gồm / loại trừ trình điều khiển phụ thuộc vào việc cơ sở Dự án hoặc phương pháp dòng lệnh đang được sử dụng để thực hiện các hoạt động cấu hình.

Lưu ý rằng chỉ cần bao gồm trình điều khiển thiết bị Xilinx không có nghĩa là trình điều khiển đó sẽ tự động được sử dụng. Hầu hết các trình điều khiển với bộ điều hợp VxWorks đều có mã khởi tạo. Trong một số trường hợp, người dùng có thể được yêu cầu thêm các lệnh gọi hàm khởi tạo trình điều khiển thích hợp vào BSP.

Khi sử dụng SDK để tạo BSP, các tệp BSP kết quả có thể chứa các nhận xét "VIỆC CẦN LÀM". Những nhận xét này, nhiều nhận xét bắt nguồn từ mẫu PowerPC 405/440 BSP do Wind River cung cấp, đưa ra các đề xuất mà người dùng phải cung cấp để định cấu hình BSP cho bảng đích. Hướng dẫn dành cho nhà phát triển VxWorks BSP và Hướng dẫn dành cho người lập trình ứng dụng VxWorks là những tài nguyên rất hữu ích cho việc cấu hình BSP.

Bao gồm / Loại trừ Trình điều khiển Dòng lệnh

Trong BSP, một tập hợp các hằng số (một cho mỗi trình điều khiển) được xác định trong <procname> _drv_config.h và tuân theo định dạng:

```
#define INCLUDE_ <XDRIIVER>
```

Tệp này được bao gồm gần đầu config.h. Theo mặc định, tất cả các trình điều khiển được bao gồm trong bản dựng. Để loại trừ trình điều khiển, hãy thêm dòng sau vào config.h sau khi bao gồm tệp tiêu đề <procname> _drv_config.h .

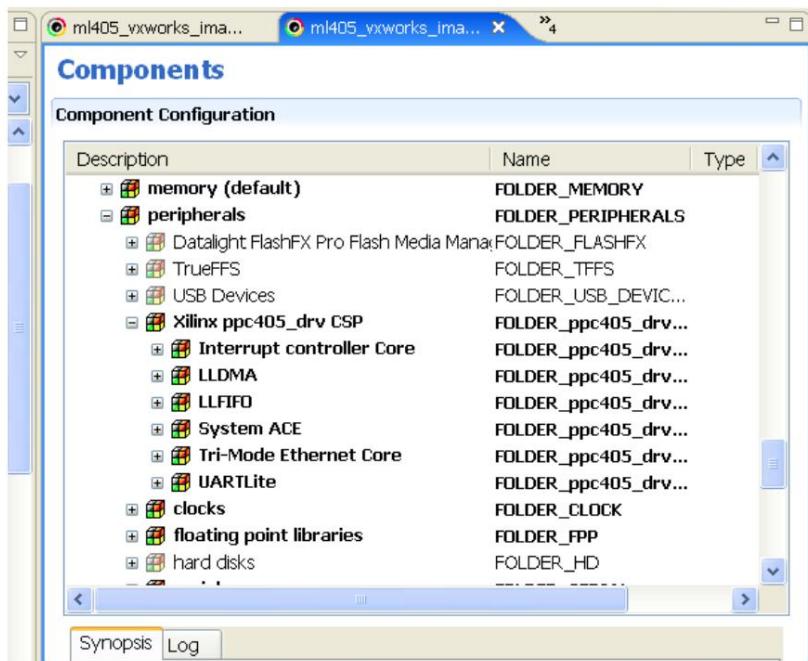
```
#undef INCLUDE_ <XDRIIVER>
```

Loại trừ này sẽ ngăn trình điều khiển được biên dịch và liên kết với bản dựng. Để cài đặt lại trình điều khiển, hãy xóa dòng #undef khỏi config.h. Một số cần thận là cần thiết cho một số trình điều khiển. Ví dụ, Ethernet có thể yêu cầu phải có trình điều khiển DMA. Việc hủy xác định trình điều khiển DMA sẽ khiến bản dựng không thành công.

Bao gồm / Loại trừ Trình điều khiển Cơ sở Dự án

Tệp 50 <csp_name>.cdf nằm trong thư mục BSP và được điều chỉnh trong quá trình tạo BSP. Tệp này tích hợp trình điều khiển thiết bị Xilinx vào IDE Workbench. Trình điều khiển thiết bị Xilinx được nối vào IDE tại thư mục con phần cứng / thiết bị ngoại vi của tab thành phần.

Đưới đây là các thư mục trình điều khiển thiết bị riêng lẻ. Ví dụ về GUI với trình điều khiển Xilinx được hiển thị trong [Hình 2](#). Để thêm hoặc xóa trình điều khiển Xilinx, hãy bao gồm hoặc loại trừ các thành phần trình điều khiển như với bất kỳ thành phần VxWorks nào khác.



Hình 2: Workbench 2.6.1 Project IDE - VxWorks

Lưu ý rằng bất kỳ cấu hình nào đã được chỉ định trong <procname>_drv_config.h và config.h sẽ bị ghi đè bởi cơ sở dự án

Xây dựng VxWorks

Các BSP được tạo tự động tuân theo các quy ước Workbench tiêu chuẩn khi tạo hình ảnh VxWorks. Tham khảo tài liệu Workbench về cách tạo hình ảnh VxWorks.

Tiện ích mở rộng bản dựng BSP dòng lệnh

Các trình điều khiển Xilinx được biên dịch / liên kết với cùng một chuỗi công cụ mà VxWorks được xây dựng. Các bổ sung nhỏ cho Makefile được yêu cầu để giúp Workbench tìm thấy vị trí của các tệp mã nguồn trình điều khiển.

Project BSP Build Extensions

Có thể thay đổi lượng tệp mới được sử dụng để tích hợp trình điều khiển thiết bị Xilinx vào quá trình xây dựng Workbench trong thư mục <bsp_name>. Như đã nêu trước đó, các tệp này được tạo tự động bởi SDK. Người dùng chỉ cần biết rằng các tệp tồn tại. Các tệp này có tiền tố là tên phiên bản của bộ xử lý.

Tích hợp thiết bị

Các thiết bị trong hệ thống nhúng dựa trên FPGA có các mức độ tích hợp khác nhau với hệ điều hành VxWorks. Người dùng SDK có thể chọn mức độ tích hợp trong hộp thoại Thiết bị ngoại vi được kết nối của tab Tham số Thư viện / Hệ điều hành. Dưới đây là danh sách các thiết bị hiện được hỗ trợ và mức độ tích hợp của chúng.

- Mô hình trình điều khiển thiết bị VxBus được hỗ trợ bắt đầu từ vxWorks6.5 BSP. Tham khảo sysLib.c và hwconf.c của BSP để xem chi tiết về quá trình di chuyển này.
- Một hoặc hai thiết bị UART 16450/16550 / Lite có thể được tích hợp vào VxWorks Serial Giao diện I / O (SIO). Điều này làm cho một UART khả dụng cho I / O tệp và printf / stdio. Chỉ một thiết bị UART có thể được chọn làm bảng điều khiển, nơi I / O tiêu chuẩn (stdin, stdout và stderr) được huy động đến. Một thiết bị UART, khi được tích hợp vào giao diện SIO, phải có khả năng tạo ra một ngắt. Nếu người dùng muốn có nhiều hơn hai thiết bị UART trong BSP của họ, thì tệp ppc405_0.h / ppc440_0.h sẽ cần được sửa đổi theo cách thủ công để thay đổi số lượng thiết bị SIO cho phù hợp.
- Thiết bị Ethernet MAC 3 tốc độ Ethernet Lite 10/100 và 10/100/1000 Local Link có thể được tích hợp vào giao diện VxWorks Enhanced Network Driver (END). Điều này làm cho thiết bị khả dụng với ngăn xếp mạng VxWorks và do đó là các ứng dụng cấp ô cắm. Một thiết bị Ethernet, khi được tích hợp vào giao diện END, phải có khả năng tạo ra các ngắt. Người dùng có thể cần sửa đổi các giá trị dòng khởi động mặc định trong config.h để thiết bị Ethernet được sử dụng làm thiết bị khởi động.
- Bộ điều khiển ngắt có thể được kết nối với xử lý ngoại lệ VxWorks intLib và chân ngắt không quan trọng bên ngoài PowerPC 405/440. BSP được tạo hiện không xử lý tích hợp bộ điều khiển ngắt cho chân ngắt quan trọng của PowerPC 405/440, cũng như không hỗ trợ kết nối trực tiếp của một thiết bị ngắt đơn (không phải intc) với bộ xử lý. Tuy nhiên, người dùng luôn có thể thêm tích hợp này theo cách thủ công trong tệp sysInterrupt.c của BSP.
- Bộ điều khiển System ACE™ có thể được kết nối với VxWorks như một thiết bị khôi, cho phép người dùng gắn hệ thống tập tin vào thiết bị CompactFlash được kết nối với bộ điều khiển System ACE. Người dùng phải gọi thủ công các hàm BSP để khởi tạo Hệ thống ACE / CompactFlash như một thiết bị khôi và gắn nó vào hệ điều hành DOS. Các chức năng hiện có sẵn cho người dùng là: sysSystemAceInitFS () và sysSystemAceMount (). Bộ điều khiển ACE hệ thống, khi được tích hợp vào giao diện thiết bị khôi, phải có khả năng tạo ra ngắt. Tham khảo tệp sysSystemAce.c trong BSP để biết thêm chi tiết. BSP sẽ gắn CF làm phân vùng đĩa DOS FAT bằng tiện ích bổ sung DosFs2.0 của Wind River. Để đưa các thư viện VxWorks cần thiết vào hình ảnh, các gói sau phải được định nghĩa trong config.h hoặc bởi Project Facility:

- INCLUDE_DOSFS_MAIN
- INCLUDE_DOSFS_FAT
- INCLUDE_DISK_CACHE
- INCLUDE_DISK_PART
- INCLUDE_DOSFS_DIR_FIXED
- INCLUDE_DOSFS_DIR_VFAT
- INCLUDE_XBD_BLK_DEV
- INCLUDE_XBD_PART_LIB

Theo chương trình, một ứng dụng có thể gắn kết hệ thống tệp DOS bằng cách sử dụng các lệnh gọi API sau:

```
TẬP TIN * fp;
sysSystemAceInitFS ();
if (sysSystemAceMount ("/ cf0", 1) != OK)
{
```

```

    /* xử lý lỗi */
}

fp = fopen ("/ cf0 / myfile.dat", "r");

```

- Cầu PCI có thể được khởi tạo và cung cấp cho trình điều khiển VxWorks PCI tiêu chuẩn và các chức năng cầu hình. Người dùng được yêu cầu chỉnh sửa các tệp config.h và hwconf.c BSP để điều chỉnh các địa chỉ và cấu hình bộ nhớ PCI cho hệ thống đích của họ. Lưu ý rằng ngắt PCI không được tích hợp tự động vào BSP.

- Bộ điều khiển thiết bị USB có thể được tích hợp vào giao diện bộ điều khiển ngoại vi USB của các thành phần VxWorks BSP. Để kiểm tra bộ điều khiển ngoại vi USB bằng cách sử dụng thành phần giả lập Bộ nhớ chung hiện có của VxWorks, các thay đổi sau phải được thực hiện trong tệp nguồn VxWorks "usbTargMsLib.c" và trong tệp BSP "config.h". Những thay đổi này phải được thực hiện trước khi tạo dự án VxWorks. Sửa đổi giá trị không đổi MS_BULK_OUT_ENDPOINT_NUM thành "2" trong

Tệp "usbTargMsLib.c". Tập tin này nằm ở thư mục <WindRiver-Installed Directory> /Vx-Works6.5/target/src/drv/usb/target/.

Sau khi sửa đổi, nguồn VxWorks sẽ được biên dịch tại thư mục này. Lệnh biên dịch cho hệ thống dựa trên PPC440 là "làm cho CPU = PPC32" và đối với hệ thống dựa trên PPC405 là "làm cho CPU = PPC405".

Trình giả lập USB MassStorage sử dụng bộ nhớ cục bộ cho vùng lưu trữ. Người dùng cần cung cấp dung lượng tối thiểu 4MB (sửa đổi giá trị hằng số LOCAL_MEM_SIZE trong tệp config.h thành 0x400000) trong RAM. Mã giả lập MassStorage mô phỏng vùng lưu trữ mặc định là 32k. • Tất cả các thiết bị khác và trình điều khiển thiết bị liên quan không được tích hợp chặt chẽ vào giao diện VxWorks. Thay vào đó, chúng được tích hợp lồng lèo và khả năng truy cập vào các thiết bị này bằng cách truy cập trực tiếp vào trình điều khiển thiết bị được liên kết từ ứng dụng của người dùng.

- Các lỗi người dùng và trình điều khiển thiết bị liên quan, nếu được bao gồm trong dự án EDK, được hỗ trợ thông qua luồng tạo BSP. Trình điều khiển thiết bị lỗi của người dùng sẽ được sao chép vào BSP giống như cách sao chép trình điều khiển thiết bị Xilinx. Điều này giả định cấu trúc thư mục của trình điều khiển thiết bị lỗi người dùng khớp với cấu trúc của trình điều khiển thiết bị Xilinx. / Xây dựng các thư mục con của trình điều khiển thiết bị phải tồn tại và được định dạng giống như trình điều khiển thiết bị Xilinx. Điều này bao gồm đoạn mã CDF và các tệp xtag trong thư mục / drivers / core_vxworks_v2_00_a / build . Trình điều khiển thiết bị của người dùng không được tích hợp tự động vào bất kỳ giao diện hệ điều hành nào (ví dụ: SIO), nhưng chúng có sẵn để ứng dụng truy cập trực tiếp.

Sai lệch

Danh sách sau đây tóm tắt sự khác biệt giữa BSP do SDK tạo và BSP truyền thống.

- Một cấu trúc thư mục bổ sung được thêm vào thư mục BSP gốc để chứa trình điều khiển thiết bị các tệp mã nguồn.
- Để giữ cho BSP có thể xây dựng trong khi duy trì khả năng tương thích với cơ sở Dự án Workbench, một tập hợp các tệp có tên <procname>_drv_<driver>_<version>.c đi kèm vào thư mục BSP chỉ cần # bao gồm mã nguồn từ thư mục con trình điều khiển của BSP.
- BSP Makefile đã được sửa đổi để trình biên dịch có thể tìm thấy mã nguồn của trình điều khiển. Makefile chứa nhiều thông tin hơn về độ lệch này và ý nghĩa của nó.
- Việc sử dụng SystemACE có thể yêu cầu thay đổi đối với các tệp mã nguồn VxWorks được tìm thấy trong Thư mục phân phối Workbench. Xem phần "[Bootrom với SystemACE làm thiết bị khởi động](#)" tiết diện.

Hạn chế

BSP được tạo tự động nên được coi là một điểm khởi đầu tốt cho người dùng, nhưng không nên được mong đợi để đáp ứng tất cả các nhu cầu của người dùng ngay khi xuất xưởng. Do sự phức tạp tiềm ẩn của BSP, nhiều tính năng có thể được bao gồm trong BSP và sự hỗ trợ cần thiết cho các thiết bị bo mạch bên ngoài FPGA, BSP được tạo tự động có thể sẽ yêu cầu người dùng cải tiến. Tuy nhiên, BSP được tạo sẽ có thể biên dịch được và sẽ chứa các trình điều khiển thiết bị cần thiết được trình bày trong hệ thống nhúng dựa trên FPGA. Một số thiết bị thường được sử dụng cũng được tích hợp hệ điều hành. Các hạn chế cụ thể được liệt kê dưới đây.

- Bộ điều khiển ngắt được kết nối với chân ngắt quan trọng PowerPC 405/440 không được tích hợp tự động vào sơ đồ ngắt của VxWorks. Hiện chỉ hỗ trợ ngắt bên ngoài.
 - Không hỗ trợ phát hiện lỗi xe buýt từ cầu xe buýt hoặc trọng tài.
 - Dòng lệnh VxWorks 6.5 BSP mặc định sử dụng trình biên dịch GNU. Người dùng phải thay đổi thủ công Makefile để sử dụng trình biên dịch DIAB hoặc chỉ định trình biên dịch DIAB khi tạo dự án Workbench dựa trên BSP.
 - Địa chỉ ROM trong config.h và Makefiles của BSP được cập nhật dựa trên phiên bản ngoại vi được chọn trong hộp menu thả xuống ROM_INSTANCE của phần cài đặt Gói hỗ trợ bo mạch của SDK. Người dùng phải chọn phiên bản ngoại vi theo cài đặt phần cứng. Trong trường hợp chọn sai, các tệp BSP sẽ được cập nhật các giá trị sai.
 - Bộ nhớ đệm PowerPC 405 bị tắt theo mặc định. Người dùng phải kích hoạt bộ nhớ đệm theo cách thủ công thông qua tệp config.h hoặc menu dự án Workbench.
 - Bộ nhớ đệm PowerPC 440 được bật theo mặc định. Người dùng phải tắt bộ nhớ đệm theo cách thủ công thông qua tệp config.h hoặc menu dự án Workbench.
 - Khi SystemACE được thiết lập để tải hình ảnh VxWorks vào RAM thông qua JTAG, tắt cả các khởi động đều lạnh (tức là không khởi động ấm). Điều này là do bộ điều khiển System ACE đặt lại bộ xử lý bất cứ khi nào nó thực hiện tải xuống tệp ace. Một tác động của điều này có thể khiến các thông báo ngoại lệ do VxWorks tạo ra không được in trên bảng điều khiển khi hệ thống được khởi động lại do một ngoại lệ trong ISR hoặc sự cố hạt nhân.
- Lưu ý: Không thể sử dụng hình ảnh nén với SystemACE. Điều này áp dụng cho các hình ảnh nén tiêu chuẩn được tạo bằng Workbench chẳng hạn như bootrom. Hình ảnh nén không thể được đặt trên SystemACE dưới dạng tệp ace. SystemACE không thể giải nén dữ liệu khi nó ghi vào RAM. Bắt đầu một hình ảnh như vậy sẽ dẫn đến sự cố hệ thống.
- Một bản dựng dòng lệnh không thể khởi tạo mạng khi SystemACE là thiết bị khởi động. Điều này yêu cầu ứng dụng cung cấp mã để khởi tạo mạng khi SystemACE là thiết bị khởi động. Để giải quyết vấn đề này, hãy xem thảo luận về \$ WIND_BASE / target / src / config / usrNetwork.c trong phần ["Bootrom với SystemACE làm thiết bị khởi động"](#).
 - Trên bộ xử lý PowerPC 405/440, vectơ đặt lại ở địa chỉ vật lý 0xFFFFFFFFC. Có một cửa sổ thời gian ngắn trong đó bộ xử lý sẽ cố gắng tìm nạp và thực thi lệnh tại địa chỉ này trong khi SystemACE xử lý tệp ace. Bộ xử lý cần được cung cấp một cái gì đó để làm trong thời gian này ngay cả khi đó là một vòng quay:
- FFFFFFFFFFC b.

Nếu BRAM chiếm phạm vi địa chỉ này, thì người thiết kế tạo ra dòng bit phải đặt hưng dẫn tại đây bằng tiện ích elf to BRAM được tìm thấy trong các công cụ Xilinx ISE.

Khởi động VxWorks

Trình tự khởi động VxWorks

Có nhiều biến thể của hình ảnh VxWorks với một số dựa trên RAM, một số dựa trên ROM.

Tùy thuộc vào thiết kế bảng, không phải tất cả những hình ảnh này đều được hỗ trợ. Danh sách sau đây thảo luận về các loại hình ảnh khác nhau:

- **Ảnh nén ROM** - Những ảnh này bắt đầu thực thi trong ROM và giải nén ảnh BSP vào RAM, sau đó chuyển quyền điều khiển sang ảnh đã giải nén trong RAM. Loại hình ảnh này không tương thích với SystemACE vì SystemACE không biết hình ảnh đã được nén và sẽ đặt nó vào RAM tại một địa chỉ sẽ bị thuật toán giải nén ghi đè khi nó bắt đầu. Có thể làm cho loại hình ảnh này hoạt động nếu các sửa đổi được thực hiện đối với các câu lệnh Workbench tiêu chuẩn để xử lý tình huống này.
- **Hình ảnh dựa trên RAM** - Những hình ảnh này được tải vào RAM bởi bộ nạp khởi động, SystemACE hoặc một trình giả lập. Những hình ảnh này được hỗ trợ đầy đủ.
- **Hình ảnh dựa trên ROM** - Những hình ảnh này bắt đầu thực thi trong ROM, tự sao chép vào RAM sau đó chuyển việc thực thi vào RAM. Trong các thiết kế với SystemACE làm bộ nạp khởi động, hình ảnh được tự động sao chép vào RAM. Ví dụ BSP được mã hóa thủ công làm ngắn mạch hoạt động sao chép VxWorks để quá trình sao chép không xảy ra nữa sau khi quyền điều khiển được SystemACE chuyển sang RAM (xem romInit.s).
- **Hình ảnh thường trú trong ROM** - Những hình ảnh này bắt đầu thực thi trong ROM, sao chép phần dữ liệu vào RAM và quá trình thực thi vẫn còn trong ROM. Trong các hệ thống chỉ có SystemACE, hình ảnh này không được hỗ trợ. Về mặt lý thuyết, BRAM có thể được sử dụng như một ROM, tuy nhiên, các bộ phận Virtex II Pro hiện tại đang được sử dụng trong các hội đồng đánh giá không có khả năng lưu trữ hình ảnh VxWorks có kích thước từ 200KB đến hơn 700KB.

Trình tự khởi động VxWorks

Hình ảnh chuẩn này được thiết kế để một số thiết bị tải xuống dung lư ợng RAM mục tiêu.

Sau khi tải xuống, bộ xử lý được thiết lập để bắt đầu thực thi tại function _sysInit tại địa chỉ RAM_LOW_ADDRS. (hàng số này được định nghĩa trong config.h và Makefile). Hầu hết thời gian, thiết bị thực hiện tải xuống sẽ tự động thực hiện việc này vì nó có thể trích xuất điểm nhập từ hình ảnh.

1. **_sysInit:** Hàm hợp ngữ này chạy hết RAM sẽ thực hiện ở mức thấp khởi tạo. Khi hoàn thành, hàm này sẽ thiết lập ngắn xếp ban đầu và gọi hàm "C" đầu tiên usrInit(). _sysInit nằm trong tệp mã nguồn <bspname>/sysALib.s.
2. **usrInit():** Hàm "C" này chạy hết RAM sẽ thiết lập môi trường thời gian chạy "C" và thực hiện khởi tạo tiền nhân. Nó gọi sysHwInit() (được triển khai trong sysLib.c) để đặt HW ở trạng thái tĩnh. Khi hoàn thành, hàm này sẽ gọi kernelInit() để hiển thị nhân VxWorks. Hàm này sẽ lần lượt gọi usrRoot() làm tác vụ đầu tiên.
3. **usrRoot():** Thực hiện khởi tạo hậu nhân. Kết nối đồng hồ hệ thống, khởi tạo ngắn xếp TCP / IP, v.v. Nó gọi sysHwInit2() (được triển khai trong sysLib.c) để định kèm và kích hoạt ngắt HW. Khi hoàn tất, usrRoot() gọi mã khởi động ứng dụng người dùng usrAppInit() nếu được định cấu hình trong BSP.

Cả usrInit() và usrRoot() đều được thực hiện bởi Wind River. Các tệp mã nguồn mà chúng tồn tại khác nhau tùy thuộc vào dòng lệnh hoặc cơ sở Dự án Workbench đang được sử dụng để biên dịch hệ thống. Trong giao diện dòng lệnh, chúng được triển khai tại \$ WIND_BASE / target / config / all / usrConfig.c. Dưới cơ sở dự án, chúng được duy trì trong thư mục dự án của người dùng.

Trình tự khởi động "bootrom_ncmp"

Hình ảnh tiêu chuẩn này dựa trên ROM như trong thực tế, nó được liên kết để thực thi các địa chỉ RAM.

Trong khi thực thi từ ROM, hình ảnh này sử dụng thủ thuật định địa chỉ tương đối để gọi các hàm xử lý các tác vụ trước khi chuyển đến RAM.

1. Bật nguồn. Vector bộ xử lý thành 0xFFFFFFF0 nơi i đặt lệnh nhảy chuyển quyền điều khiển tới bootrom tại địa chỉ _romInit.
 2. _romInit: Chức năng hợp ngữ này chạy hết ROM lưu ý rằng đây là khởi động ngoại rồi nhảy để bắt đầu. Cả _romInit và start đều nằm trong tệp mã nguồn <bspname> /romInit.s. 3. start: Chức năng hợp ngữ này chạy hết ROM sẽ thiết lập bộ xử lý,
- làm mất hiệu lực của bộ nhớ đệm và chuẩn bị cho hệ thống hoạt động trên RAM. Thao tác cuối cùng là gọi hàm "C" romStart () được thực hiện bởi Wind River và nằm trong tệp mã nguồn \$ WIND_BASE / target / config / all / bootInit.c.
4. romStart (): Hàm "C" này khi hết ROM sẽ sao chép VxWorks vào phần khởi động RAM của nó địa chỉ tại RAM_HIGH_ADRS (hàng số này được định nghĩa trong config.h và Makefile). Sau khi sao chép VxWorks, quyền điều khiển được chuyển đến hàm usrInit () trong RAM.
 5. Làm theo bước 2 & 3 của trình tự khởi động "vxWorks".

"bootrom_uncmp" Trình tự khởi động với SystemACE

Hình ảnh không chuẩn này tương tự như hình ảnh được thảo luận trong phần trước ngoại trừ việc SystemACE được sử dụng để tải nó. Một số thay đổi phải được thực hiện đối với quá trình khởi động. Có thể tìm thêm thông tin trong phần <RD Red> <BT BoldType> Bootrom với SystemACE làm Thiết bị khởi động <RD Red>, trang 14.

1. Bật nguồn. SystemACE tải hình ảnh vào RAM tại RAM_HIGH_ADRS (hàng số này được định nghĩa trong config.h và Makefile) và đặt bộ xử lý bắt đầu tìm nạp các hướng dẫn tại địa chỉ _romInit.
2. _romInit: Hàm hợp ngữ này chạy hết RAM lưu ý rằng đây là khởi động ngoại rồi nhảy để bắt đầu. Cả _romInit và start đều nằm trong tệp mã nguồn <bspname> /romInit.s.
3. start: Hàm hợp ngữ này chạy hết RAM chỉ cần chuyển đến hàm _sysInit. Lệnh gọi tới romStart () bị bỏ qua vì SystemACE đã tải bootrom vào địa chỉ RAM đích của nó.
4. Làm theo các bước 1, 2 và 3 của trình tự khởi động "vxWorks".

Bootroms

Bootrom là một hình ảnh VxWorks được thu nhỏ lại, hoạt động theo cách giống như BIOS của PC. Công việc chính của nó là tìm và khởi động một hình ảnh VxWorks đầy đủ. Hình ảnh VxWorks đầy đủ có thể nằm trên đĩa, trong bộ nhớ flash hoặc trên một số máy chủ thông qua Ethernet. Bootrom phải được biên dịch theo cách mà nó có khả năng truy xuất hình ảnh. Nếu hình ảnh được truy xuất từ mạng Ethernet, thì bootrom phải có ngăn xếp TCP / IP được biên dịch, nếu hình ảnh nằm trên đĩa, thì bootrom phải có hỗ trợ truy cập đĩa được biên dịch trong, v.v. Các bootrom không làm gì khác hơn là truy xuất và bắt đầu hình ảnh đầy đủ và duy trì một dòng khởi động. Dòng khởi động là một chuỗi văn bản đặt các đặc điểm người dùng nhất định như địa chỉ IP của mục tiêu nếu sử dụng Ethernet và đường dẫn tệp đến hình ảnh VxWorks để khởi động.

Bootrom không phải là một yêu cầu bắt buộc. Chúng thường được sử dụng trong môi trường phát triển sau đó được thay thế bằng hình ảnh VxWorks sẵn xuất.

Tạo Bootroms

Tại trình bao lệnh trong thư mục BSP, hãy phát hành lệnh sau để tạo hình ảnh bootrom không nén (bắt buộc đối với SystemACE):

```
tạo bootrom_uncmp
hoặc
làm bootrom
```

để tạo ra một hình ảnh nén phù hợp để đặt trong một mảng bộ nhớ flash.

Hiển thị Bootrom

Khi khởi động nguồn, nếu các bootrom hoạt động bình thường, đầu ra tự động tự như sau sẽ được nhìn thấy trên cổng nối tiếp của bảng điều khiển:

Khởi động hệ thống VxWorks

Bản quyền 1984-2007 Wind River Systems, Inc.

CPU: ppc405_0 VirtexII Pro PPC405 Phiên bản: VxWorks

6.5

Phiên bản BSP: 2.0 / 0.

Ngày tạo: 11 tháng 10 năm 2007, 16:40:32

Nhấn phím bất kỳ để dừng tự động khởi động ...

3

[VxWorks Boot]:

Nhập trợ giúp tại dấu nhắc này liệt kê các lệnh có sẵn.

Bootline

Dòng khởi động là một chuỗi văn bản xác định các đặc điểm có thể phục vụ của người dùng như địa chỉ IP của bảng đích và cách tìm hình ảnh vxWorks để khởi động. Dòng khởi động được bootrom duy trì trong thời gian chạy và thường được giữ trong một số vùng lưu trữ không bay hơi (NVRAM) của hệ thống như EEPROM hoặc bộ nhớ flash. Nếu không có NVRAM hoặc xảy ra lỗi khi đọc nó, thì dòng khởi động được mã hóa cứng bằng DEFAULT_BOOT_LINE được xác định trong tệp mã nguồn config.h của BSP. Trong các hệ thống mới mà NVRAM chưa được khởi tạo, dòng khởi động có thể là dữ liệu không xác định.

Dòng khởi động có thể được thay đổi nếu chuỗi đếm người tự động khởi động bị gián đoạn bằng cách nhập một ký tự trên cổng nối tiếp bảng điều khiển. Sau đó, lệnh "c" có thể được sử dụng để chỉnh sửa tự động dòng khởi động. Nhập "p" để xem dòng khởi động. Trên ảnh không phải bootrom, dòng khởi động có thể được thay đổi bằng cách nhập lệnh bootChange tại dấu nhắc máy chủ hoặc trình bao bì.

Các truy ống dòng khởi động được định nghĩa bên dưới:

boot device: Thiết bị khởi động từ. Đây có thể là Ethernet hoặc một đĩa cục bộ. Lưu ý rằng khi thay đổi dòng khởi động, số đơn vị có thể được hiển thị thêm vào truy ống này ("lltemac0" hoặc "sysace = 10) khi nhắc thiết bị khởi động mới. Số này có thể được bỏ qua.

số bộ xử lý: Luôn luôn là 0 với các hệ thống bộ xử lý đơn lẻ.

Tên máy chủ: Đặt tên khi cần thiết.

tên tệp: Hình ảnh VxWorks để khởi động. **inet** trên

ethernet (e): Địa chỉ Internet IP của mục tiêu. Nếu không có mạng giao diện, sau đó truy ống này có thể được để trống.

host inet (h): Địa chỉ Internet IP của máy chủ. Nếu không có giao diện mạng, thì truy ống này có thể được để trống.

user (u): Tên người dùng để truy cập hệ thống tệp máy chủ. Chọn bất kỳ tên nào phù hợp với bạn. Máy chủ ftp của bạn phải được thiết lập để cho phép người dùng này truy cập vào hệ thống tệp máy chủ.

Mật khẩu ftp (pw): Mật khẩu để truy cập hệ thống tệp máy chủ. Chọn bất kỳ tên nào phù hợp với bạn. Máy chủ ftp của bạn phải được thiết lập để cho phép người dùng này truy cập vào hệ thống tệp máy chủ.

flags (f): Để biết danh sách các tùy chọn, hãy nhập lệnh "trợ giúp" tại [VxWorks Boot]:
lời nhắc.

target name (tn): Đặt tên khi cần thiết. Đặt theo yêu cầu mạng.

other (o): Trưởng này hữu ích khi bạn có thiết bị không phải Ethernet làm thiết bị khởi động.

Trong trường hợp này, VxWorks sẽ không khởi động mạng khi khởi động. Chỉ định thiết bị Ethernet ở đây sẽ kích hoạt thiết bị đó tại thời điểm khởi động với các tham số mạng được chỉ định trong các trường dòng khởi động khác.

inet trên bảng nối đa năng (b): Thưòng để trống nếu hệ thống đích không nằm trên VME hoặc PCI bảng nối đa năng.

công inet (g): Nhập địa chỉ IP vào đây nếu bạn phải đi qua công vào tiếp cận máy tính chủ. Nếu không thì để trống.

(các) tập lệnh khởi động: Đường dẫn đến một tệp trên máy tính chủ chứa các lệnh shell để thực thi sau khi khởi động xong. Để trống nếu không sử dụng tập lệnh. Ví dụ:

Tập lệnh thưòng trú của SystemACE: /cf0/vxworks/scripts/myscript.txt

Tập lệnh thưòng trú trên máy chủ: c: /temp/myscript.txt

Bootrom với SystemACE làm thiết bị khởi động

Các bootrom hỗ trợ SystemACE có khả năng khởi động hình ảnh VxWorks trực tiếp từ thiết bị Compact Flash dưới dạng tệp elf thông thường hoặc tệp ace.

Các sửa đổi bắt buộc đối với Nguồn VxWorks

Mặc dù EDK có khả năng tạo BSP sử dụng SystemACE trong hình ảnh "vxWorks" có thể mở và đóng tệp trong hệ thống tệp DOS, nó không thể tạo BSP sử dụng SystemACE làm thiết bị khởi động "bootrom". Để sử dụng SystemACE theo cách này, cần phải sửa đổi nhiều đối với mã bootrom do Wind River cung cấp. Wind River cho phép các nhà phát triển BSP thay đổi các tệp mã nguồn Workbench miễn là họ giữ các thay đổi cục bộ đối với BSP và giữ nguyên mã gốc. Hai tệp phải được sửa đổi từ phiên bản gốc của chúng là:

1. \$ WIND_BASE / target / config / all / bootConfig.c: Tệp này được ghi đè bằng một tệp được tìm thấy trong thư mục <bspname>. Những thay đổi cần thiết là thêm mã để phân tích cú pháp dòng khởi động đúng cách và khởi tạo và sử dụng SystemACE làm thiết bị khởi động JTAG và DOS. Để ghi đè bootConfig.c mặc định, dòng sau phải được thêm vào Makefile của BSP:

BOOTCONFIG = ./bootConfig.c.

2. \$ WIND_BASE / target / config / comps / src / net / usrNetBoot.c: Tệp này được ghi đè bằng tệp được tìm thấy trong thư mục <bspname> /net/usrNetBoot.c. Những thay đổi cần thiết là thêm mã để làm cho VxWorks biết rằng SystemACE là một hệ thống dựa trên đĩa như IDE, SCSI hoặc ổ đĩa mềm. Thay đổi này cho phép một BSP được xây dựng từ Dự án Workbench được tải xuống với bootrom hỗ trợ SystemACE để xử lý đúng trường khác của dòng khởi động. Sự tồn tại của tệp đã sửa đổi trong thư mục BSP sẽ tự động ghi đè tệp gốc.

Cả hai tệp này đều không được EDK cung cấp vì chúng được Wind River duy trì ở dạng ban đầu.

Không thể ghi đè tệp thứ ba, \$ WIND_BASE / target / src / config / usrNetwork.c do kiến trúc của quá trình xây dựng BSP dòng lệnh. Điều này ảnh hưởng đến các BSP có khả năng mạng được xây dựng từ dòng lệnh được tải xuống với bootrom hỗ trợ SystemACE.

Nếu không sửa đổi usrNetwork.c, các BSP bị ảnh hưởng không thể khởi chạy thiết bị mạng của họ và phải dựa vào mã ứng dụng để thực hiện chức năng này. Nếu người dùng muốn, họ có thể thực hiện thay đổi đối với tệp này trong cài đặt Workbench của họ. Phương pháp này có những bất lợi vì bất kỳ chỉnh sửa nào được thực hiện đối với tệp này đều ảnh hưởng đến tất cả người dùng cài đặt đó và có thể bị mất nếu người dùng nâng cấp hoặc cài đặt lại Workbench. Thay đổi đối với usrNetwork.c xảy ra trong hàm usrNetDevStart () .

từ:

```
if ((strncpy (params.bootDev, "scsi", 4) == 0) ||
    (strncpy (params.bootDev, "ide", 3) == 0) ||
    (strncpy (params.bootDev, "ata", 3) == 0) ||
```

```

        (strncmp (params.bootDev, "fd", 2) == 0) (strncmp      ||
        (params.bootDev, "tffs", 4) == 0))
dến

if ((strncmp (params.bootDev, "scsi", 4) == 0) ||
    (strncmp (params.bootDev, "ide", 3) == 0) ||
    (strncmp (params.bootDev, "ata", 3) == 0) ||
    (strncmp (params.bootDev, "fd", 2) == 0) (strncmp      ||
    (params.bootDev, "sysace", 6) == 0) ||
    (strncmp (params.bootDev, "tffs", 4) == 0))

```

Chỉnh sửa mã này có nguy cơ của riêng bạn.

Cấu hình đặc biệt

Chuẩn bị hình ảnh bootrom_ncmp có thể tải xuống bởi SystemACE dưới dạng tệp ace cần có cấu hình đặc biệt. Những thay đổi này là bắt buộc vì bootrom được liên kết để bắt đầu chạy hết thiết bị bộ nhớ không bay hơi, sao chép chính nó vào RAM, sau đó chuyển quyền điều khiển sang bản sao RAM. Các thay đổi sẽ ngăn hoạt động sao chép vì SystemACE đã đặt bootrom vào thiết bị RAM khi đặt lại.

một. Thay đổi định nghĩa của ROM_TEXT_ADRS và ROM_WARM_ADRS thành giá trị tương ứng với RAM_HIGH_ADRS trong cả config.h và Makefile.

b. Thay đổi mã hợp ngữ ở nhãn bắt đầu trong romInit.s để chuyển đến chức năng _sysInit:

bắt đầu:

```

LOADPTR (r1, _sysInit)
mtlr r1
blr1

```

Định dạng dòng khởi động

Trường thiết bị khởi động của dòng khởi động được chỉ định bằng cú pháp sau:

`sysace = <số phân vùng>`

trong đó <số phân vùng> là phân vùng để khởi động từ đó. Thông thường, giá trị này được đặt thành 1, nhưng một số thiết bị CF không có bảng phân vùng và được định dạng như thế chúng là một đĩa mềm lớn. Trong trường hợp này, chỉ định 0 làm số phân vùng. Việc không lấy đúng số phân vùng sẽ dẫn đến lỗi được các thư viện dosFS của VxWork thông báo khi ổ đĩa được gắn kết.

Trường tên tệp của dòng khởi động được đặt tùy thuộc vào cách Hệ thống ACE khởi động hệ thống. Có hai phương pháp khởi động:

1. Khởi động từ một tệp thông thường. Điều này tương tự như khởi động mạng trong hình ảnh vxWorks nằm trong thiết bị lưu trữ flash nhỏ gọn SystemACE thay vì hệ thống tệp máy chủ. Thiết bị flash nhỏ gọn là một phân vùng hệ thống tệp DOS FAT. Xây dựng vxWorks bằng công cụ Workbench, sao chép tệp hình ảnh kết quả vào thiết bị flash nhỏ gọn bằng đầu đọc thẻ USB hoặc công cụ tương tự, sau đó chỉ định tệp đó trong trường tên tệp của bootrom. Tên tệp phải có cú pháp sau:

```
/ cf0 / <path / to / vxWorks / image>
```

trong đó cf0 là điểm gắn kết. <path / to / vxWorks / image> phải cung cấp đường dẫn đầy đủ đến hình ảnh VxWorks để khởi động. Khi được chỉ định theo cách này, bootrom sẽ gắn ổ đĩa dưới dạng đĩa định dạng FAT, tải tệp vào bộ nhớ và bắt đầu thực thi.

2. Khởi động từ một tệp ace. Tệp ace chỉ có thể chứa HW, SW only hoặc HW + SW. Khi nào khởi động từ một tệp ace với HW, FPGA được lập trình lại. Nếu tệp ace chứa SW, thì nó được tải vào bộ nhớ, PC của bộ xử lý được đặt thành điểm nhập và được giải phóng để bắt đầu tìm nạp các hướng dẫn. Phương pháp khởi động này linh hoạt ở chỗ một cấu hình HW hoàn toàn khác có thể được khởi động từ bootrom VxWorks. Tên tệp phải có những điều sau đây

cú pháp:

```
cfgaddr [x]
```

trong đó [X] là một số từ 0 đến 7 tương ứng với một trong các thư mục cấu hình được chỉ định trong cù trú tệp XILINX.SYS trong thư mục gốc của thiết bị flash nhỏ gọn. Nếu [X] bị bỏ qua, thì cấu hình mặc định sẽ được sử dụng. Cấu hình mặc định thường được chọn bằng một công tắc xoay được gắn ở đâu đó trên bảng đánh giá. Bootrom sẽ kích hoạt tải xuống JTAG của tệp ace được trả về bởi địa chỉ cấu hình được chỉ định. Chỉ nên có một tệp duy nhất có phần mở rộng .ace trong thư mục cấu hình đã chọn.

Trong cả hai trường hợp khởi động, nếu thiết bị Ethernet được khởi động khi khởi động VxWorks đã tải xuống, truờng "khác" của dòng khởi động phải được sửa đổi để chứa tên của thiết bị mạng.

Bootrom với Ethernet ba chế độ liên kết cục bộ (LLTEMAC) làm thiết bị khởi động

SDK sẽ tạo một BSP có khả năng được xây dựng như một bootrom bằng cách sử dụng LLTEMAC làm thiết bị khởi động. Cấu hình ngẫu nhiên dùng rất ít được yêu cầu. Địa chỉ MAC được mã hóa cứng trong tệp nguồn hwconf.c. BSP có thể được sử dụng với MAC mặc định miễn là mục tiêu nằm trên một mạng riêng và không có nhiều hơn một mục tiêu trên mạng đó có cùng địa chỉ MAC mặc định. Nếu không, nhà thiết kế nên thay thế MAC này bằng một chức năng để lấy một MAC từ thiết bị nhớ không bay hơi trên bảng đích của họ.

Để chỉ định LLTEMAC làm thiết bị khởi động trong bootrom, hãy thay đổi truờng thiết bị khởi động trong dòng khởi động thành lltemac. Nếu chỉ có một LLTEMAC, hãy đặt số đơn vị thành 0.

Ví dụ về dòng khởi động

Ví dụ sau khởi động từ ethernet bằng Xilinx lltemacas thiết bị khởi động. Hình ảnh được khởi động nằm trên hệ thống tệp máy chủ trên ổ C.

```
thiết bị khởi động : lltemac
số đơn vị : 0
tên máy chủ số bộ xử lý : 0
: chủ nhà
tên tập tin : c:/WindRiver/vxworks-6.5/target/config/ml507/vxWorks
inet trên ethernet (e): 192.168.0.2
máy chủ inet (h) : 192.168.0.1
ngẫu nhiên dùng (u) : xemhost
mật khẩu ftp (pw) cờ (f) : sao cung đư ợc
tên đích (tn) khác (o) : 0x0
: vxtarget
: :
```

Ví dụ sau khởi động từ một tập tin cù trú trên phân vùng đầu tiên của thiết bị flash nhỏ gọn SystemACE. Nếu tệp được khởi động từ / cf0 / vxworks / images / vxWorks sử dụng mạng, thì thiết bị xemac được khởi tạo.

```
thiết bị khởi động : sysace = 1
số đơn vị : 0
tên máy chủ số bộ xử lý : 0
: chủ nhà
tên tập tin : / cf0 / vxworks / images / vxWorks
inet trên ethernet (e): 192.168.0.2
máy chủ inet (h) : 192.168.0.1
ngẫu nhiên dùng : xemhost
(u) mật khẩu ftp (pw) cờ : sao cung đư ợc
(f) tên mục tiêu (tn) khác : 0x0
(o) : vxtarget
: xemac
```

Ví dụ sau khởi động từ một cùn dân tệp ace trên phân vùng đầu tiên của thiết bị flash nhỏ gọn SystemACE. Vị trí của tệp ace được đặt bởi XILINX.SYS nằm trong thư mục gốc của thiết bị flash nhỏ gọn. Nếu tệp ace chứa hình ảnh VxWorks SW sử dụng mạng, thì thiết bị xemac được khởi tạo cho hình ảnh đó.

```

thiết bị khởi động          : sysace = 1
số đơ n vị                 : 0
tên máy chủ số bộ xử lý   : 0
                               : chủ nhà
tên tập tin                 : cfgaddr2
inet trên ethernet (e): 192.168.0.2
máy chủ inet (h)           : 192.168.0.1
người dùng (u)              : xemhost
mật khẩu ftp (pw) cờ (f)   : sao cung đư ợc
tên đích (tn) khác (o)      : 0x0
                               : vxtarget
                               : xemac

```

Bộ nhớ đệm

Hướng dẫn và bộ nhớ đệm dữ liệu được quản lý bởi các thư viện bộ nhớ VxWorks. Chúng được kích hoạt bằng cách sửa đổi các hằng số sau trong config.h hoặc bằng cách sử dụng cơ sở Dự án Workbench để thay đổi các hằng số có cùng tên:

- INCLUDE_CACHE_SUPPORT: Nếu được xác định, các thư viện bộ nhớ VxWorks được liên kết với hình ảnh. Nếu bộ nhớ đệm không được mong muốn, thi hãy # hoàn thiện hằng số này.
- USER_I_CACHE_ENABLE: Nếu được định nghĩa, VxWorks sẽ kích hoạt bộ nhớ chỉ lệnh lúc khởi động. Yêu cầu INCLUDE_CACHE_SUPPORT được xác định để có bất kỳ tác dụng nào.
- USER_D_CACHE_ENABLE: Nếu được định nghĩa, VxWorks sẽ kích hoạt bộ nhớ dữ liệu tại thời điểm khởi động. Yêu cầu INCLUDE_CACHE_SUPPORT được xác định để có bất kỳ tác dụng nào.

MMU

Nếu MMU được bật, thì điều khiển bộ nhớ được thảo luận trong phần trự ớc có thể không có bất kỳ tác dụng nào. MMU được quản lý bởi các thư viện bộ nhớ VxWorks như ng thiết lập ban đầu được xác định trong BSP. Để bật MMU, hằng số INCLUDE_MMU_BASIC phải được xác định trong config.h hoặc bằng cách sử dụng Project Facility. Hằng số USER_D_MMU_ENABLE và USER_I_MMU_ENABLE kiểm soát liệu MMU hướng dẫn và / hoặc dữ liệu có được sử dụng hay không.

VxWorks khởi tạo MMU dựa trên dữ liệu trong cấu trúc sysPhysMemDesc được định nghĩa trong sysCache.c. Bộ nhớ dành riêng cho người dùng và ED&R (khi INCLUDE_EDR_PM được bật) bộ nhớ dành riêng được bao gồm trong bảng này. Trong số những thứ khác, bảng này định cấu hình các vùng bộ nhớ với các thuộc tính sau:

- Cho phép thực hiện lệnh hay không.
- Cho phép ghi dữ liệu
- Các thuộc tính về khả năng lưu trữ dữ liệu và hướng dẫn.
- Phần bù dịch được sử dụng để tạo địa chỉ ảo.

Khi VxWorks khởi tạo MMU, nó sẽ lấy các định nghĩa từ sysPhysMemDesc và tạo các mục nhập bảng trang (PTE) trong RAM. Mỗi PTE mô tả 4KB vùng bộ nhớ (mặc dù bộ xử lý có khả năng đại diện lên đến 16MB cho mỗi PTE). Hãy lưu ý rằng việc chỉ định các vùng bộ nhớ lớn sử dụng lưu lượng RAM đáng kể để lưu trữ các PTE. Để ánh xạ 4MB không gian bộ nhớ liền kề, cần 8KB RAM để lưu trữ các PTE.

Để tăng hiệu suất với gói MMU cơ bản của VxWorks cho bộ xử lý PPC405 / 440, có thể có lợi nếu không bật MMU hướng dẫn và dựa vào cài đặt điều khiển bộ nhớ trong thanh ghi ICCR. Chiến lược này có thể giảm đáng kể số lưu lượng lỗi trang trong khi vẫn

giữ hứa hẹn dẫn trong bộ nhớ cache. Cài đặt ban đầu của ICCR được xác định trong tệp tiêu đề <bspname>.h.

Đối với PPC440, bộ nhớ đệm và MMU được bật theo mặc định.

Nếu không bật MMU, các quy tắc sau áp dụng cho việc định cấu hình các thuộc tính truy cập bộ nhớ và bộ nhớ đệm:

- Không có bản dịch địa chỉ, tất cả các địa chỉ hiệu quả là vật lý.
- Mức độ chi tiết của kiểm soát bộ nhớ cache là 128MB.
- Thuộc tính được bảo vệ chỉ áp dụng cho các lần tìm nạp lệnh suy đoán trên PPC405.

FPU

Đơn vị dấu phẩy động cứng (FPU) được hỗ trợ cho các hệ thống PPC440. Để bật đơn vị dấu phẩy động cứng, vui lòng chọn diab hoặc gnu trong CÔNG CỤ Makefile BSP đã tạo. Để tắt đơn vị dấu phẩy động cứng, hãy chọn sfdiab hoặc sfgnu trong CÔNG CỤ tạo biến số của Makefile.



UG705

Tự động tạo gói hỗ trợ bo mạch WindRiver VxWorks 6.7

Bản tóm tắt

Tài liệu này mô tả cách tạo tự động của Gói hỗ trợ bằng làm việc (BSP) bằng cách sử dụng Bộ phát triển phần mềm Xilinx® (SDK) (1). Tài liệu bao gồm những điều sau đây các phần.

- [“Tổng quan”](#)
- [“Tạo VxWorks 6.7 BSP”](#)
- [“VxWorks BSP”](#)
- [“Khởi động VxWorks”](#)

Tổng quan

Một trong những hoạt động phát triển hệ thống nhúng chính là phát triển BSP. Việc tạo ra một BSP có thể là một quá trình kéo dài và tẻ nhạt phải phát sinh khi có sự thay đổi trong tổ hợp bộ vi xử lý bao gồm bộ xử lý và các thiết bị ngoại vi liên quan. Mặc dù việc quản lý những thay đổi này áp dụng cho bất kỳ dự án nào dựa trên bộ vi xử lý, nhưng giờ đây những thay đổi có thể được thực hiện nhanh hơn với sự ra đời của phần cứng Hệ thống trên chip (SoC) có thể lập trình được.

Tài liệu này mô tả việc tạo tự động VxWorks 6.7 BSP tùy chỉnh cho bộ vi xử lý IBM PowerPC® 440 và các thiết bị ngoại vi của nó như được định nghĩa trong Xilinx FPGA. BSP được tạo tự động cho phép các nhà thiết kế hệ thống nhúng:

- Giảm chu kỳ phát triển, do đó giảm thời gian đưa ra thị trường
- Tạo BSP tùy chỉnh để phù hợp với phần cứng và ứng dụng
- Loại bỏ lỗi thiết kế BSP (được tạo tự động dựa trên các thành phần được chứng nhận)
- Cho phép phát triển phần mềm ứng dụng bằng cách loại bỏ thời gian chờ phát triển BSP

VxWorks 6.7 BSP được tạo từ SDK, một IDE được phân phối như một phần của Bộ phát triển nhúng Xilinx (EDK) hoặc có sẵn riêng từ Xilinx. SDK được sử dụng để tạo các ứng dụng phần mềm cho các hệ thống nhúng trong Xilinx FPGA. VxWorks BSP chứa tất cả phần mềm hỗ trợ cần thiết cho hệ thống, bao gồm mã khởi động, trình điều khiển thiết bị và khởi tạo RTOS. BSP được tùy chỉnh dựa trên các thiết bị ngoại vi được người dùng chọn và cấu hình cho hệ thống nhúng dựa trên FPGA.

Các nhà thiết kế BSP có kinh nghiệm nên dễ dàng tích hợp BSP đã tạo vào hệ thống mục tiêu của họ. Ngoài lại, những người dùng ít kinh nghiệm hơn có thể gặp khó khăn vì mặc dù SDK có thể tạo BSP hoạt động cho một tập hợp phần cứng IP nhất định, nhưng sẽ luôn có một số câu hỏi và điều chỉnh bổ sung cần thiết để tạo ra hiệu suất tốt nhất từ hệ thống mục tiêu. Người dùng nên có sẵn Hướng dẫn dành cho nhà phát triển Wind River VxWorks BSP và Hướng dẫn dành cho lập trình viên ứng dụng VxWorks hoặc xem xét các lớp Wind River trên thiết kế BSP, có sẵn với một khoản chi phí bổ sung.

1. Xilinx SDK được sử dụng làm môi trường phát triển phần mềm chính cho người dùng Xilinx Embedded Development Kit (EDK) kể từ EDK 11.1i. Khả năng phát triển phần mềm của Xilinx Platform Studio (XPS) hiện không được dùng nữa và sẽ bị xóa khỏi XPS trong các bản phát hành sau. Các luồng được mô tả trong tài liệu này liên quan đến SDK, mặc dù chúng vẫn có thể áp dụng chung cho XPS trong khi các tính năng đó vẫn còn trong công cụ.

Yêu cầu

Bộ phát triển Wind River Workbench 3.1 phải được cài đặt trên máy tính chủ.

Vì SDK tạo ra các BSP có thể di chuyển lại được biên dịch và định cấu hình bên ngoài môi trường SDK, nên máy tính chủ không cần cài đặt cả Xilinx SDK và Workbench.

Định nghĩa thư viện bộ vi xử lý

SDK hỗ trợ giao diện trình cắm thêm cho các thư viện và hệ điều hành của bên thứ 3 thông qua giao diện Định nghĩa Thư viện Bộ vi xử lý (MLD), do đó cho phép các nhà cung cấp bên thứ 3 cung cấp phần mềm của họ cho người dùng SDK. Ngoài ra, nó cung cấp cho các nhà cung cấp một phương tiện để điều chỉnh thư viện hoặc BSP của họ cho phù hợp với hệ thống nhúng dựa trên FPGA được tạo trong các công cụ Xilinx. Bởi vì hệ thống có thể thay đổi dễ dàng, khả năng này rất quan trọng trong việc hỗ trợ đúng cách các hệ thống nhúng trong FPGA.

Xilinx phát triển và duy trì VxWorks 6.7 MLD trong các bản phát hành SDK của mình. MLD được sử dụng để tự động tạo VxWorks 6.7 BSP.

Phương pháp Tiếp cận Dựa trên Mẫu

Một tập hợp các tệp mẫu VxWorks 6.7 BSP được phát hành cùng với SDK. Các tệp mẫu này được sử dụng trong quá trình tạo BSP tự động và các sửa đổi thích hợp được thực hiện dựa trên cấu trúc của hệ thống nhúng dựa trên FPGA.

Các tệp mẫu này có thể được sử dụng làm tài liệu tham khảo để xây dựng BSP từ đầu nếu người dùng chọn không tự động tạo BSP.

Trình điều khiển thiết bị

Một tập hợp các tệp nguồn trình điều khiển thiết bị được phát hành cùng với SDK và nằm trong thư mục cài đặt. Trong quá trình tạo BSP tùy chỉnh, mã nguồn trình điều khiển thiết bị được sao chép từ thư mục cài đặt này sang thư mục BSP. Chỉ mã nguồn liên quan đến các thiết bị được tích hợp trong hệ thống nhúng dựa trên FPGA mới được sao chép. Bản sao này cung cấp cho người dùng một thư mục BSP độc lập, độc lập có thể được sửa đổi hoặc di dời. Nếu người dùng thực hiện các thay đổi đối với mã nguồn trình điều khiển thiết bị cho BSP này, sau đó muốn hoàn tác các thay đổi, công cụ SDK có thể được sử dụng để tạo lại BSP. Tại thời điểm đó, các tệp nguồn của trình điều khiển thiết bị được mở lại từ thư mục cài đặt vào BSP.

Tạo VxWorks 6.7 BSP

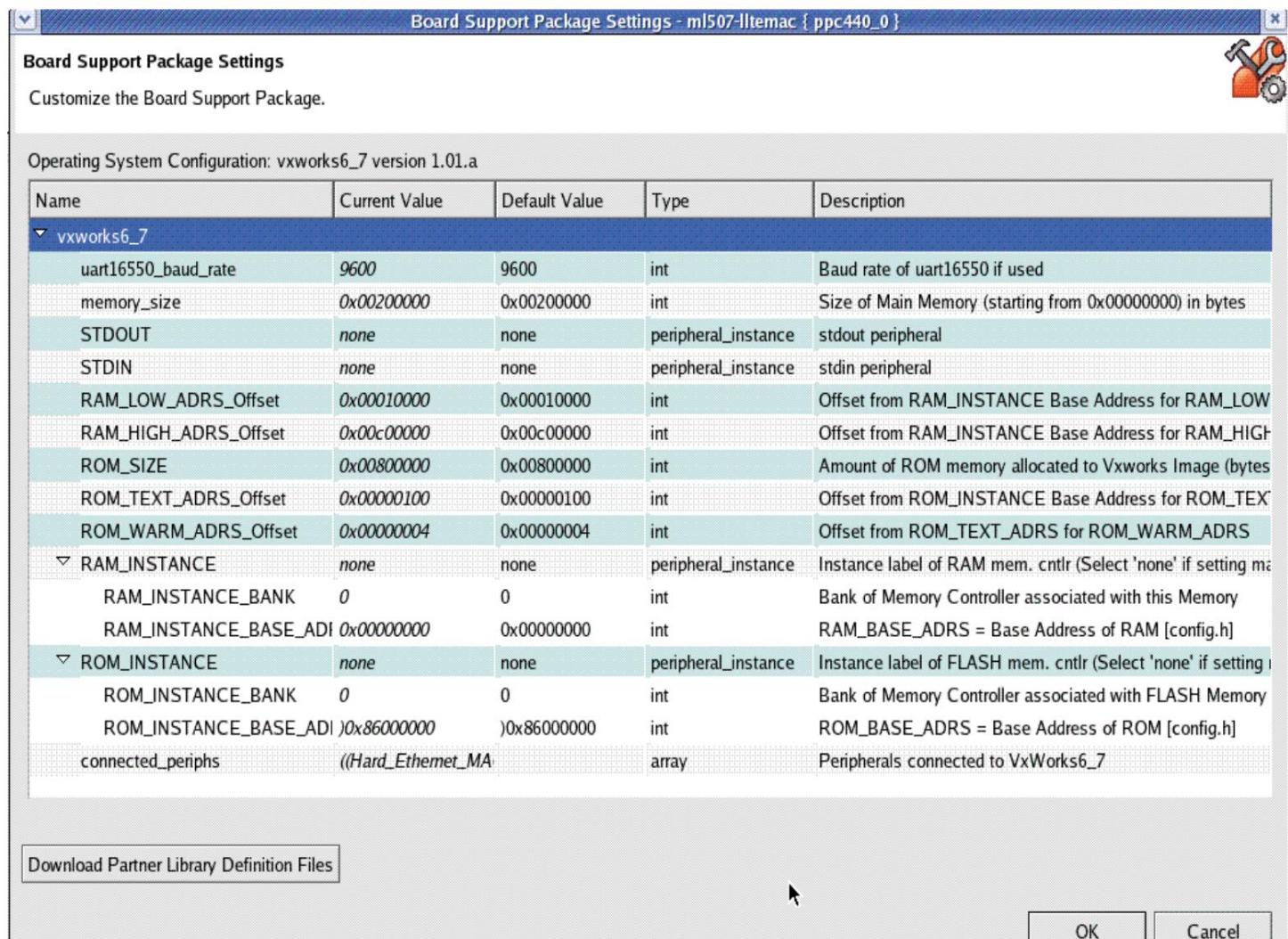
Sử dụng SDK

SDK có sẵn dưới dạng công cụ được cài đặt riêng hoặc trong EDK và là môi trường phát triển phần mềm để phát triển phần mềm nhúng xung quanh các hệ thống nhúng dựa trên Xilinx PowerPC 405/440 hoặc MicroBlaze™. Phần này mô tả các bước cần thiết để tạo VxWorks 6.7 BSP bằng SDK. Các bước này có thể áp dụng khi sử dụng công cụ The Xilinx 11.1i trở lên.

Giả định rằng một thiết kế phần cứng hợp lệ đã được tạo và xuất sang SDK, và SDK đã được mở và trỏ đến thiết kế phần cứng.

1. Sử dụng Tệp> Mới, tạo một dự án Gói Hỗ trợ Ban mới. Trong hộp thoại, nhập tên dự án và chọn vxworks6_7 làm Loại gói hỗ trợ hội đồng quản trị. Lưu ý rằng SDK có thể quản lý nhiều dự án thuộc các loại BSP khác nhau.

Các bước còn lại liên quan đến hộp thoại Tools> Board Support Package Settings ... , hộp thoại này sẽ tự động được hiển thị sau bước trên.



Hình 1: Cài đặt Gói hỗ trợ bo mạch

2. Định cấu hình thiết bị giao diện điều khiển VxWorks.

Nếu một thiết bị nối tiếp như Uart được dự định sử dụng làm bảng điều khiển VxWorks, hãy chọn hoặc nhập tên phiên bản của thiết bị nối tiếp làm thiết bị ngoại vi STDIN / STDOUT trong hộp thoại Cài đặt Gói Hỗ trợ Bảng. Điều quan trọng là phải nhập cùng một thiết bị cho cả STDIN và STDOUT. Hiện tại, chỉ có các thiết bị Uart 16550/16450 và UartLite được hỗ trợ làm thiết bị VxWorks console.

3. Tích hợp các trình điều khiển thiết bị.

một. Kết nối với VxWorks.

Hộp thoại connect_periph có sẵn trong hộp thoại Cài đặt gói hỗ trợ hội đồng quản trị Các thiết bị ngoại vi đã được diền sẵn để thuận tiện cho người dùng. Sử dụng hộp thoại này để sửa đổi các thiết bị ngoại vi đó để được tích hợp chặt chẽ với Hệ điều hành, bao gồm thiết bị đã được chọn làm thiết bị ngoại vi STDIN / STDOUT. Xem phần "[Tích hợp thiết bị](#)" để biết thêm chi tiết về tích hợp chặt chẽ các thiết bị.

b. Dung lư ợng bộ nhớ

Trường này được sử dụng để định cấu hình BSP để phù hợp với kích thước bộ nhớ phần cứng thực tế trên bo mạch của bạn.

c. Uart16550_baud_rate

Trường này được sử dụng để nhập tốc độ truyền cho các dự án có lõi UART 16550/16450. Không cần thiết phải nhập giá trị ở đây cho các dự án có lõi UART Lite vì tốc độ truyền được đặt cho UART Lite tại thời điểm xây dựng phần cứng.

d. RAM_INSTANCE

Đây là menu thả xuống để chọn phiên bản ngoại vi sẽ được sử dụng làm RAM trong BSP. Ngân hàng bộ nhớ trựòng con và địa chỉ cơ sở của RAM dưới RAM_INSTANCE phải được định cấu hình để phù hợp với cài đặt phần cứng thực tế.

e. ROM_INSTANCE

Đây là menu thả xuống để chọn phiên bản ngoại vi sẽ được sử dụng làm ROM trong BSP. FLASH là thiết bị ROM duy nhất được hỗ trợ trên bảng Đánh giá Xilinx. Nếu không có ROM trong hệ thống, người dùng có thể để cài đặt mặc định tức là không có. Nếu có FLASH trong hệ thống, ngân hàng bộ nhớ trựòng con và địa chỉ cơ sở của ROM trong ROM_INSTANCE phải được định cấu hình để phù hợp với cài đặt phần cứng thực tế.

f. RAM_LOW_ADRS_OFFSET

Trường này được sử dụng để nhập địa chỉ offset cho địa chỉ cơ sở RAM để lấy địa chỉ RAM cho vxWorks được sử dụng trong BSP và phải được định cấu hình để phù hợp với cài đặt hệ thống phần cứng. g. RAM_HIGH_ADRS_OFFSET

Trường này được sử dụng để nhập địa chỉ offset cho địa chỉ cơ sở RAM để lấy địa chỉ RAM được sử dụng trong BSP cho ROM khởi động và phải được định cấu hình để phù hợp với cài đặt hệ thống phần cứng.

h. ROM_LOW_ADRS_OFFSET

Trường này được sử dụng để nhập địa chỉ offset cho địa chỉ cơ sở ROM để lấy địa chỉ bắt đầu FLASH được sử dụng trong BSP và phải được định cấu hình để phù hợp với cài đặt hệ thống phần cứng.

i. ROM_HIGH_ADRS_OFFSET

Trường này được sử dụng để nhập bù địa chỉ cho địa chỉ cơ sở ROM để lấy địa chỉ cuối FLASH được sử dụng trong BSP và phải được định cấu hình để phù hợp với cài đặt hệ thống phần cứng.

j. ROM_SIZE

Trường này được sử dụng để định cấu hình BSP và phải khớp với cài đặt hệ thống phần cứng thực tế.

k. ROM_TEXT_ADRS_OFFSET

Trường này được sử dụng để nhập độ lệch địa chỉ cho địa chỉ cơ sở ROM để lấy địa chỉ bắt đầu phần văn bản được sử dụng trong BSP và phải được định cấu hình để phù hợp với cài đặt hệ thống phần cứng.

l. ROM_WARM_ADRS_OFFSET

Trường này được sử dụng để nhập bù địa chỉ cho địa chỉ cơ sở ROM để lấy địa chỉ nhập khởi động lại ấm được sử dụng trong BSP và phải được định cấu hình để phù hợp với cài đặt hệ thống phần cứng.

4. Tạo VxWorks 6.7 BSP

Bấm OK trên hộp thoại Cài đặt Gói Hỗ trợ Ban ... để tạo BSP. Đầu ra của lệnh gọi này được hiển thị trong cửa sổ bảng điều khiển SDK. Sau khi hoàn tất, VxWorks 6.7 BSP kết quả sẽ tồn tại trong không gian làm việc SDK của bạn, dưới tên thư mục dự án mà bạn đã tạo ở bước 1 ở trên, trong thư mục con phiên bản PowerPC 440. Ví dụ: nếu trong thiết kế phần cứng, người dùng đã đặt tên phiên bản PowerPC 440 là myppc440, thì BSP sẽ nằm ở không gian làm việc SDK / tên dự án SDK / myppc440 / bsp_ppc440.

Sao lưu u

Để tránh việc ngư ời dùng vô tình làm mất các thay đổi đối với tệp nguồn BSP, các tệp hiện có trong vị trí thư mục của BSP sẽ được sao chép vào thư mục sao lưu trước khi bị ghi đè. Thư mục sao lưu nằm trong thư mục BSP và được đặt tên là `backuptimestamp`, trong đó timestamp đại diện cho ngày và giờ hiện tại. Vì BSP được tạo bởi SDK có thể định vị lại, bạn nên di chuyển BSP từ thư mục dự án SDK sang thư mục phát triển BSP thích hợp ngay khi nền tảng phần cứng ổn định.

VxWorks BSP

Phần này mô tả cách tạo ra VxWorks 6.7 BSP của SDK. Giả định rằng ngư ời đọc đã quen thuộc với Workbench 3.1 IDE của Wind River.

BSP được tạo tự động và tích hợp vào Workbench IDE. BSP có thể được biên dịch từ dòng lệnh bằng cách sử dụng các công cụ tạo Workbench hoặc từ cơ sở Workbench Project (còn được gọi là Workbench IDE). Khi BSP đã được tạo, ngư ời dùng có thể nhập lệnh `make vxWorks` từ dòng lệnh để biên dịch hình ảnh RAM có thể khởi động. Điều này giả định rằng môi trường Workbench đã được thiết lập trước đó, có thể được thực hiện thông qua dòng lệnh bằng cách sử dụng tiện ích môi trường `wrenv` Wind River trên nền tảng Windows. Xem Hướng dẫn ngư ời dùng dòng lệnh Wind River Workbench: Tạo lớp vỏ phát triển với `wrenv` để biết thêm thông tin về cách sử dụng các tiện ích dòng lệnh. Nếu sử dụng cơ sở Dự án Workbench, ngư ời dùng có thể tạo một dự án dựa trên BSP mới được tạo, sau đó sử dụng môi trường xây dựng để cung cấp thông qua IDE để biên dịch BSP.

Trong Workbench 3.1, trình biên dịch `diab` được hỗ trợ ngoài trình biên dịch `gnu`. VxWorks 6.7 BSP do SDK tạo ra có `Makefile` có thể được sửa đổi bởi ngư ời dùng dòng lệnh để sử dụng trình biên dịch `diab` thay vì trình biên dịch `gnu`. Tìm biến `make` có tên `TOOLS` và đặt giá trị thành `sfdiab` thay vì `sfgnu`. Đối với PPC440 với hệ thống đơn vị đầu phẩy động cứng (FPU), vui lòng chọn `diab` hoặc `gnu`. Nếu sử dụng cơ sở Dự án Workbench, ngư ời dùng có thể chọn công cụ mong muốn khi dự án được tạo lần đầu tiên.

Tổ chức tài xế

Phần này thảo luận ngắn gọn về cách trình điều khiển Xilinx được biên dịch và liên kết và cuối cùng được sử dụng bởi các cấu hình Workbench để đưa vào hình ảnh VxWorks.

Các trình điều khiển Xilinx được triển khai bằng ngôn ngữ lập trình C và có thể được phân phối giữa một số tệp nguồn không giống như các trình điều khiển VxWorks truyền thống, bao gồm các tệp triển khai và tiêu đề C duy nhất.

Có ba thành phần cho trình điều khiển Xilinx:

- Bao gồm nguồn trình điều khiển
- Triển khai độc lập hệ điều hành
- Triển khai phụ thuộc vào hệ điều hành (tùy chọn)

Bao gồm nguồn trình điều khiển để cập đến cách trình điều khiển Xilinx được biên dịch. Đối với mọi trình điều khiển, có một tệp có tên `procname_drv_dev_version.c`. Sử dụng lệnh `#include` bao gồm (các) tệp nguồn (*.c) cho mỗi trình điều khiển cho mỗi thiết bị nhất định.

Quá trình này tương tự như cách nguồn của VxWorks `sysLib.c` # `include` cho các trình điều khiển được cung cấp bởi Wind River. Lý do tại sao các tệp Xilinx không đơn giản được đưa vào `sysLib.c`, cũng như phần còn lại của các trình điều khiển, là do xung đột không gian tên và các vấn đề về khả năng bảo trì. Nếu tắt cả các tệp Xilinx là một phần của một đơn vị biên dịch duy nhất, thì các hàm và dữ liệu tĩnh không còn là riêng tư nữa. Điều này đặt ra những hạn chế đối với trình điều khiển thiết bị và sẽ phủ nhận tính độc lập của hệ điều hành của chúng.

Phần độc lập hệ điều hành của trình điều khiển được thiết kế để sử dụng với bất kỳ hệ điều hành hoặc bất kỳ bộ xử lý nào. Nó cung cấp một API sử dụng chức năng của phần cứng bên dưới.

Phần phụ thuộc vào hệ điều hành của trình điều khiển sẽ điều chỉnh trình điều khiển để sử dụng với VxWorks. Các ví dụ như vậy là trình điều khiển SIO cho các cổng nối tiếp hoặc trình điều khiển IPNET cho bộ điều hợp ethernet. Không phải tất cả các trình điều khiển đều yêu cầu trình điều khiển phụ thuộc vào hệ điều hành, cũng như không bắt buộc phải bao gồm phần phụ thuộc vào hệ điều hành của trình điều khiển trong bản dựng VxWorks.

Vị trí trình điều khiển thiết bị

BSP được tạo tự động giống với hầu hết các BSP Workbench khác ngoại trừ việc đặt mã trình điều khiển thiết bị. Mã trình điều khiển thiết bị có sẵn được phân phối với IDE Workbench thư ờng nằm trong thư mục vxworks-6.7 / target / src / drv trong thư mục cài đặt Workbench. Mã trình điều khiển thiết bị cho BSP được tạo tự động nằm trong chính thư mục BSP. Sai lệch nhỏ này là do bản chất động của hệ thống nhúng dựa trên FPGA. Vì hệ thống nhúng dựa trên FPGA có thể được lập trình lại với IP mới hoặc thay đổi, cấu hình trình điều khiển thiết bị có thể thay đổi, yêu cầu vị trí năng động hơn của các tệp nguồn trình điều khiển thiết bị.

Cây thư mục cho BSP được tạo tự động là bsp_name / csp_name_csp / xsrc.

Thư mục cấp cao nhất được đặt tên theo tên của phiên bản bộ xử lý trong dự án thiết kế phần cứng. Các tệp nguồn BSP tùy chỉnh nằm trong thư mục này. Có một thư mục con trong thư mục BSP được đặt tên theo phiên bản bộ xử lý với hậu tố là _drv_csp. Thư mục trình điều khiển chứa hai thư mục con. Thư mục xsr chứa tất cả các tệp nguồn liên quan đến trình điều khiển thiết bị. Nếu xây dựng từ cơ sở Dự án Workbench, các tệp được tạo trong quá trình xây dựng nằm ở \$ PRJ_DIR / \$ BUILD_SPEC.

Cấu hình

BSP do SDK tạo ra được định cấu hình giống như bất kỳ BSP VxWorks 6.7 nào khác. Có rất ít khả năng cấu hình cho trình điều khiển Xilinx vì phần cứng IP đã được định cấu hình trước trong hầu hết các trư ờng hợp. Cấu hình duy nhất có sẵn nói chung là liệu trình điều khiển có được bao gồm trong bản dựng VxWorks hay không. Quá trình bao gồm / loại trừ trình điều khiển phụ thuộc vào việc cơ sở Dự án hoặc phương pháp dòng lệnh đang được sử dụng để thực hiện các hoạt động cấu hình.

Lưu ý rằng chỉ cần bao gồm trình điều khiển thiết bị Xilinx không có nghĩa là trình điều khiển đó sẽ tự động được sử dụng. Hầu hết các trình điều khiển với bộ điều hợp VxWorks đều có mã khởi tạo. Trong một số trư ờng hợp, người dùng có thể được yêu cầu thêm các lệnh gọi hàm khởi tạo trình điều khiển thích hợp vào BSP.

Khi sử dụng SDK để tạo BSP, các tệp BSP kết quả có thể chứa các nhận xét "VIỆC CẦN LÀM". Những nhận xét này, nhiều trong số đó bắt nguồn từ mẫu PowerPC 440 BSP do Wind River cung cấp, đưa ra gợi ý về những gì người dùng phải cung cấp để định cấu hình BSP cho bảng đích. Hướng dẫn dành cho nhà phát triển VxWorks BSP và Hướng dẫn dành cho người lập trình ứng dụng VxWorks là những tài nguyên rất hữu ích cho việc cấu hình BSP.

Bao gồm / Loại trừ Trình điều khiển Dòng lệnh

Trong BSP, một tập hợp các hằng số (một cho mỗi trình điều khiển) được xác định trong procname_drv_config.h và tuân theo định dạng:

```
#define INCLUDE_XDRIVER
```

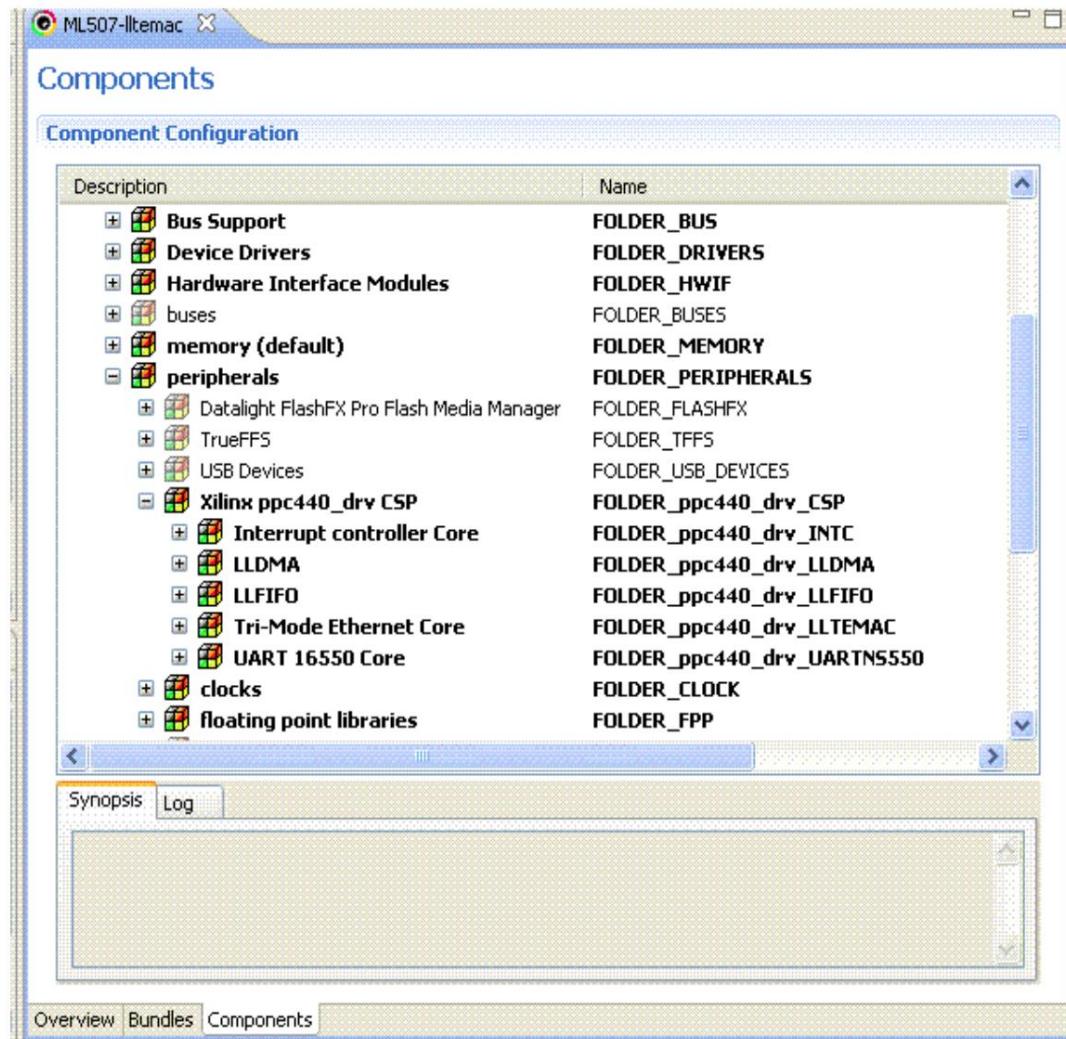
Tệp này được bao gồm gần đầu config.h. Theo mặc định, tất cả các trình điều khiển được bao gồm trong bản dựng. Để loại trừ trình điều khiển, hãy thêm dòng sau vào config.h sau khi bao gồm tệp tiêu đề procname_drv_config.h .

```
#undef INCLUDE_XDRIVER
```

Loại trừ này sẽ ngăn trình điều khiển được biên dịch và liên kết với bản dựng. Để cài đặt lại trình điều khiển, hãy xóa dòng #undef khỏi config.h. Một số cần thận là cần thiết cho một số trình điều khiển. Ví dụ, Ethernet có thể yêu cầu phải có trình điều khiển DMA. Việc hủy xác định trình điều khiển DMA sẽ khiến bản dựng không thành công.

Bao gồm / Loại trừ Trình điều khiển Cơ sở Dự án

Tệp 50csp_name.cdf nằm trong thư mục BSP và được điều chỉnh trong quá trình tạo BSP. Tệp này tích hợp trình điều khiển thiết bị Xilinx vào IDE Workbench. Trình điều khiển thiết bị Xilinx được nối vào IDE tại thư mục con phần cứng / thiết bị ngoại vi của tab thành phần. Dưới đây là các thư mục trình điều khiển thiết bị riêng lẻ. Ví dụ về GUI với trình điều khiển Xilinx được hiển thị trong [Hình 2](#). Để thêm hoặc xóa trình điều khiển Xilinx, hãy bao gồm hoặc loại trừ các thành phần trình điều khiển như với bất kỳ thành phần VxWorks nào khác.



Hình 2: Workbench 3.1 Project IDE - VxWorks

Lưu ý: Cấu hình được chỉ định trong procname_drv_config.h và config.h bị ghi đè bởi cơ sở dự án.

Xây dựng VxWorks

Các BSP được tạo tự động tuân theo các quy ước Workbench tiêu chuẩn khi tạo hình ảnh VxWorks. Tham khảo tài liệu Workbench về cách tạo hình ảnh VxWorks.

Tiện ích mở rộng bản dựng BSP dòng lệnh

Các trình điều khiển Xilinx được biên dịch / liên kết với cùng một chuỗi công cụ mà VxWorks được xây dựng. Các bổ sung nhỏ cho Makefile được yêu cầu để giúp Workbench tìm thấy vị trí của các tệp mã nguồn trình điều khiển.

Project BSP Build Extensions

Có thể thấy số lượng tệp mới được sử dụng để tích hợp trình điều khiển thiết bị Xilinx vào quá trình xây dựng Workbench trong thư mục <bsp_name>. Các tệp này được tạo tự động bởi SDK. Người dùng chỉ cần biết rằng các tệp tồn tại. Các tệp này có tiền tố là tên phiên bản của bộ xử lý.

Tích hợp thiết bị

Các thiết bị trong hệ thống nhúng dựa trên FPGA có các mức độ tích hợp khác nhau với hệ điều hành VxWorks. Người dùng SDK có thể chọn mức độ tích hợp trong hộp thoại Thiết bị ngoại vi được kết nối của tab Tham số Thư viện / Hệ điều hành. Dưới đây là danh sách các thiết bị hiện được hỗ trợ và mức độ tích hợp của chúng.

- Mô hình trình điều khiển thiết bị VxBus được hỗ trợ bắt đầu từ VxWorks6.5 BSP. Tham khảo sysLib.c và hwconf.c của BSP để xem chi tiết về quá trình di chuyển này.
- Một hoặc hai thiết bị UART 16450/16550 / Lite có thể được tích hợp vào giao diện VxWorks Serial I / O (SIO). Điều này làm cho một UART khả dụng cho I / O tệp và printf / stdio. Chỉ một thiết bị UART có thể được chọn làm bảng điều khiển, nơi I / O tiêu chuẩn (stdin, stdout và stderr) được hưỡng đến. Một thiết bị UART, khi được tích hợp vào giao diện SIO, phải có khả năng tạo ra một ngắt. Nếu người dùng muốn có nhiều hơn hai thiết bị UART trong BSP của họ, thì tệp ppc440_0.h phải được sửa đổi theo cách thủ công để thay đổi số lượng thiết bị SIO cho phù hợp.
- Ethernet Lite 10/100 và 10/100/1000 Local Link Tri-speed MAC thiết bị Ethernet MAC có thể được tích hợp vào giao diện VxWorks IPNET. Điều này làm cho thiết bị khả dụng với ngăn xếp mạng VxWorks và do đó là các ứng dụng cấp ổ cắm. Một thiết bị Ethernet, khi được tích hợp vào giao diện IPNET, phải có khả năng tạo ra các ngắt. Người dùng có thể cần sửa đổi các giá trị dòng khởi động mặc định trong config.h để thiết bị Ethernet được sử dụng làm thiết bị khởi động.
- Bộ điều khiển ngắt có thể được kết nối với xử lý ngoại lệ VxWorks intLib và chân ngắt không quan trọng bên ngoài PowerPC 440. BSP được tạo hiện không xử lý tích hợp bộ điều khiển ngắt cho chân ngắt quan trọng của PowerPC 440, cũng như không hỗ trợ kết nối trực tiếp của một thiết bị ngắt duy nhất (ngoài intc) với bộ xử lý. Tuy nhiên, người dùng luôn có thể thêm tích hợp này theo cách thủ công trong tệp sysInterrupt.c của BSP.
- Bộ điều khiển System ACE™ có thể được kết nối với VxWorks như một thiết bị khôi, cho phép người dùng gắn hệ thống tập tin vào thiết bị CompactFlash được kết nối với bộ điều khiển System ACE. Người dùng phải gọi thủ công các hàm BSP để khởi tạo Hệ thống ACE / CompactFlash như một thiết bị khôi và gắn nó vào hệ điều hành DOS. Các chức năng hiện có sẵn cho người dùng là: sysSystemAceInitFS () và sysSystemAceMount (). Bộ điều khiển ACE hệ thống, khi được tích hợp vào giao diện thiết bị khôi, phải có khả năng tạo ra ngắt. Tham khảo tệp sysSystemAce.c trong BSP để biết thêm chi tiết. BSP sẽ gắn CF làm phân vùng đĩa DOS FAT bằng tiện ích bổ sung Wind River DosFs2.0. Để đưa các thư viện VxWorks cần thiết vào hình ảnh, các gói sau phải được định nghĩa trong config.h hoặc bởi Project Facility:

- INCLUDE_DOSFS_MAIN
- INCLUDE_DOSFS_FAT
- INCLUDE_DISK_CACHE
- INCLUDE_DISK_PART
- INCLUDE_DOSFS_DIR_FIXED
- INCLUDE_DOSFS_DIR_VFAT
- INCLUDE_XBD_BLK_DEV
- INCLUDE_XBD_PART_LIB

Theo chương trình, một ứng dụng có thể gắn kết hệ thống tệp DOS bằng cách sử dụng các lệnh gọi API sau:

```
TẬP TIN * fp;

sysSystemAceInitFS ();
if (sysSystemAceMount ("/ cf0", 1) != OK)
{
    /* xử lý lỗi */
}
fp = fopen ("/ cf0 / myfile.dat", "r");
```

- Cầu PCI có thể được khởi tạo và có sẵn cho trình điều khiển VxWorks PCI tiêu chuẩn và các chức năng cấu hình. Người dùng được yêu cầu chỉnh sửa các tệp config.h và hwconf.c BSP để điều chỉnh các địa chỉ và cấu hình bộ nhớ PCI cho hệ thống đích của họ. Lưu ý rằng ngắt PCI không được tích hợp tự động vào BSP.
- Bộ điều khiển thiết bị USB có thể được tích hợp vào giao diện bộ điều khiển ngoại vi USB của các thành phần VxWorks BSP. Để kiểm tra bộ điều khiển ngoại vi USB bằng cách sử dụng thành phần giả lập Bộ nhớ chung hiện có của VxWorks, các thay đổi sau phải được thực hiện trong tệp nguồn usbTargMsLib.c và trong tệp BSP config.h. Những thay đổi này phải được thực hiện trước khi tạo dự án VxWorks.
 - Sửa đổi giá trị không đổi MS_BULK_OUT_ENDPOINT_NUM thành 2 trong usbTargMsLib.c tập tin. Tập tin này nằm tại thư mục WindRiver-Installed-Directory / Vx Works6.7 / target / src / drv / usb / target /.
 - Sau khi sửa đổi, nguồn VxWorks sẽ được biên dịch tại thư mục này. Lệnh biên dịch cho hệ thống dựa trên PPC440 là làm cho CPU = PPC32.
 - Trình giả lập USB MassStorage sử dụng bộ nhớ cục bộ cho vùng lưu trữ. Người dùng cần cung cấp dung lượng tối thiểu 4MB (sửa đổi giá trị hằng số LOCAL_MEM_SIZE trong tệp config.h thành 0x400000) trong RAM. Mã giả lập MassStorage mô phỏng vùng lưu trữ mặc định là 32k.
- Tất cả các thiết bị khác và trình điều khiển thiết bị liên quan không được tích hợp chặt chẽ vào giao diện VxWorks. Thay vào đó, chúng được tích hợp lỏng lẻo và khả năng truy cập vào các thiết bị này bằng cách truy cập trực tiếp vào trình điều khiển thiết bị được liên kết từ ứng dụng của người dùng.
- Các lỗi người dùng và trình điều khiển thiết bị liên quan, nếu được bao gồm trong dự án EDK, được hỗ trợ thông qua luồng tạo BSP. Trình điều khiển thiết bị lỗi của người dùng sẽ được sao chép vào BSP giống như cách sao chép trình điều khiển thiết bị Xilinx. Điều này giả định cấu trúc thư mục của trình điều khiển thiết bị lỗi người dùng khớp với cấu trúc của trình điều khiển thiết bị Xilinx. Tùy thuộc vào các thư mục con của trình điều khiển thiết bị phải tồn tại và được định dạng giống như trình điều khiển thiết bị Xilinx. Điều này bao gồm đoạn mã CDF và các tệp xtag trong thư mục con / drivers / core_vxworks_v2_00_a / build . Trình điều khiển thiết bị của người dùng không được tích hợp tự động vào bất kỳ giao diện hệ điều hành nào (ví dụ: SIO), nhưng chúng có sẵn để ứng dụng truy cập trực tiếp.

Sai lệch

Danh sách sau đây tóm tắt sự khác biệt giữa BSP do SDK tạo và BSP truyền thống.

- Một cấu trúc thư mục bổ sung được thêm vào thư mục BSP gốc để chứa trình điều khiển thiết bị các tệp mã nguồn.
 - Để giữ cho BSP có thể xây dựng trong khi duy trì khả năng tương thích với cơ sở Dự án Workbench, một tập hợp các tệp có tên procname_drv_driver_version.c điền vào thư mục BSP # bao gồm mã nguồn từ thư mục con trình điều khiển của BSP. • BSP Makefile đã được sửa đổi để trình biên dịch có thể tìm thấy mã nguồn của trình điều khiển.
- Makefile chứa nhiều thông tin hơn về độ lệch này và ý nghĩa của nó.

- Việc sử dụng SystemACE làm thiết bị khởi động có thể yêu cầu thay đổi đối với các tệp mã nguồn VxWorks được tìm thấy trong thư mục phân phối Workbench. Những thay đổi này nằm ngoài phạm vi của tài liệu này.

Hạn chế

BSP được tạo tự động nên được coi là một điểm khởi đầu tốt cho người dùng, nhưng không nên được mong đợi để đáp ứng tất cả các nhu cầu của người dùng ngay khi xuất xưởng. Do sự phức tạp tiềm ẩn của BSP, nhiều tính năng có thể được bao gồm trong BSP và sự hỗ trợ cần thiết cho các thiết bị bo mạch bên ngoài FPGA, BSP được tạo tự động có thể sẽ yêu cầu người dùng cài tiến. Tuy nhiên, BSP được tạo sẽ có thể biên dịch được và sẽ chứa các trình điều khiển thiết bị cần thiết được trình bày trong hệ thống nhúng dựa trên FPGA.

Một số thiết bị thường được sử dụng cũng được tích hợp hệ điều hành. Các hạn chế cụ thể được liệt kê dưới đây.

- Bộ điều khiển ngắt được kết nối với chân ngắt quan trọng PowerPC 440 không tự động tích hợp vào sơ đồ ngắt VxWorks. Hiện chỉ hỗ trợ ngắt bên ngoài.
- Không hỗ trợ phát hiện lỗi xe buýt từ cầu xe buýt hoặc trọng tải.
- Dòng lệnh VxWorks 6.7 BSP mặc định sử dụng trình biên dịch GNU. Người dùng phải thay đổi thủ công Makefile để sử dụng trình biên dịch DIAB hoặc chỉ định trình biên dịch DIAB khi tạo dự án Workbench dựa trên BSP.
- Địa chỉ ROM trong config.h và Makefiles của BSP được cập nhật dựa trên phiên bản ngoại vi được chọn trong hộp menu thả xuống ROM_INSTANCE của cài đặt Gói hỗ trợ bảng của SDK. Người dùng phải chọn phiên bản ngoại vi theo cài đặt phần cứng. Trong trường hợp chọn sai, các tệp BSP sẽ được cập nhật các giá trị sai.
- Bộ nhớ đệm PowerPC 440 được bật theo mặc định. Người dùng phải tắt bộ nhớ đệm theo cách thủ công thông qua tệp config.h hoặc menu dự án Workbench.
- Khi SystemACE được thiết lập để tải hình ảnh VxWorks vào RAM thông qua JTAG, tắt cả các khởi động đều lạnh (tức là không khởi động ấm). Điều này là do bộ điều khiển System ACE đặt lại bộ xử lý bất cứ khi nào nó thực hiện tải xuống tệp ace. Một tác động của điều này có thể khiến các thông báo ngoại lệ do VxWorks tạo ra không được in trên bảng điều khiển khi hệ thống được khởi động lại do một ngoại lệ trong ISR hoặc sự cố hạt nhân.

Lưu ý: Không thể sử dụng hình ảnh nén với SystemACE. Điều này áp dụng cho các hình ảnh nén tiêu chuẩn được tạo bằng Workbench chẳng hạn như bootrom. Hình ảnh nén không thể được đặt trên SystemACE dưới dạng tệp ace. SystemACE không thể giải nén dữ liệu khi nó ghi vào RAM. Bắt đầu một hình ảnh như vậy sẽ dẫn đến sự cố hệ thống.

- Trên bộ xử lý PowerPC 440, vector đặt lại ở địa chỉ vật lý 0xFFFFFFFFC. Có một cửa sổ thời gian ngắn trong đó bộ xử lý sẽ cố gắng tìm nạp và thực thi lệnh tại địa chỉ này trong khi SystemACE xử lý tệp ace. Bộ xử lý cần được cung cấp một cái gì đó để làm trong thời gian này ngay cả khi đó là một vòng quay:

FFFFFFFC b.

Nếu RAM khởi chiếm phạm vi địa chỉ này, thì người thiết kế tạo dòng bit nên đặt các hú ứng dẫn tại đây với tiện ích ELF để chặn RAM được tìm thấy trong Xilinx ISE® công cụ.

Khởi động VxWorks

Trình tự khởi động VxWorks

Có nhiều biến thể của hình ảnh VxWorks với một số dựa trên RAM, một số dựa trên ROM.

Tùy thuộc vào thiết kế bảng, không phải tất cả những hình ảnh này đều được hỗ trợ. Danh sách sau đây thảo luận về các loại hình ảnh khác nhau:

- **Ảnh nén ROM** - Những ảnh này bắt đầu thực thi trong ROM và giải nén ảnh BSP vào RAM, sau đó chuyển quyền điều khiển sang ảnh đã giải nén trong RAM. Loại hình ảnh này không tương thích với SystemACE vì SystemACE không biết hình ảnh đã được nén và sẽ đặt nó vào RAM tại một địa chỉ sẽ bị thuật toán giải nén đè khi nó bắt đầu. Có thể làm cho loại hình ảnh này hoạt động nếu các sửa đổi được thực hiện đối với các câu lệnh Workbench tiêu chuẩn để xử lý tình huống này.
- **Hình ảnh dựa trên RAM** - Những hình ảnh này được tải vào RAM bởi bộ nạp khởi động, SystemACE hoặc một trình giả lập. Những hình ảnh này được hỗ trợ đầy đủ.
- **Hình ảnh dựa trên ROM** - Những hình ảnh này bắt đầu thực thi trong ROM, tự sao chép vào RAM sau đó chuyển việc thực thi vào RAM. Trong các thiết kế với SystemACE làm bộ nạp khởi động, hình ảnh được tự động sao chép vào RAM. Ví dụ BSP được mã hóa thủ công làm ngắn mạch hoạt động sao chép VxWorks để quá trình sao chép không xảy ra nữa sau khi quyền điều khiển được SystemACE chuyển sang RAM (xem romInit.s).
- **Hình ảnh thường trú trong ROM** - Những hình ảnh này bắt đầu thực thi trong ROM, sao chép phần dữ liệu vào RAM và quá trình thực thi vẫn còn trong ROM. Trong các hệ thống chỉ có SystemACE, hình ảnh này không được hỗ trợ. Về mặt lý thuyết, BRAM có thể được sử dụng như một ROM, tuy nhiên, các bộ phận Virtex II Pro hiện tại đang được sử dụng trong các hội đồng đánh giá không có khả năng lưu trữ hình ảnh VxWorks có kích thước từ 200KB đến hơn 700KB.

Trình tự khởi động VxWorks

Hình ảnh chuẩn này được thiết kế để một số thiết bị tải xuống dung lượng RAM mục tiêu.

Sau khi tải xuống, bộ xử lý được thiết lập để bắt đầu thực thi tại function _sysInit tại địa chỉ RAM_LOW_ADRS. (hàng số này được định nghĩa trong config.h và Makefile). Hầu hết thời gian, thiết bị thực hiện tải xuống sẽ tự động thực hiện việc này vì nó có thể trích xuất điểm nhập từ hình ảnh.

1. **_sysInit**: Hàm hợp ngữ này chạy hết RAM sẽ thực hiện ở mức thấp khởi tạo. Khi hoàn thành, hàm này sẽ thiết lập ngăn xếp ban đầu và gọi hàm C đầu tiên usrInit(). _sysInit nằm trong tệp mã nguồn bspname / sysALib.s.
2. **usrInit()**: Hàm C này chạy hết RAM sẽ thiết lập môi trường thời gian chạy C và thực hiện khởi tạo tiền nhân. Nó gọi sysHwInit() (được triển khai trong sysLib.c) để đặt phần cứng ở trạng thái tĩnh. Khi hoàn thành, hàm này sẽ gọi kernelInit() để hiển thị nhân VxWorks. Hàm này sẽ lần lượt gọi usrRoot() làm tác vụ đầu tiên.
3. **usrRoot()**: Thực hiện khởi tạo hậu nhân. Kết nối dòng hồ hệ thống, khởi tạo ngăn xếp TCP / IP, v.v. Nó gọi sysHwInit2() (được triển khai trong sysLib.c) để đính kèm và kích hoạt ngắt HW. Khi hoàn tất, usrRoot() gọi mã khởi động ứng dụng người dùng usrAppInit() nếu được định cấu hình trong BSP.

Cả usrInit() và usrRoot() đều được thực hiện bởi Wind River. Các tệp mã nguồn mà chúng tồn tại khác nhau tùy thuộc vào dòng lệnh hoặc cơ sở Dự án Workbench đang được sử dụng để biên dịch hệ thống. Trong giao diện dòng lệnh, chúng được triển khai tại \$ WIND_BASE / target / config / all / usrConfig.c. Dưới cơ sở dự án, chúng được duy trì trong thư mục dự án của người dùng.

Trình tự khởi động "bootrom_uncmp"

Hình ảnh tiêu chuẩn này dựa trên ROM như ng trong thực tế, nó được liên kết để thực thi các địa chỉ RAM. Trong khi thực thi từ ROM, hình ảnh này sử dụng thủ thuật định địa chỉ tương đối để gọi các hàm xử lý các tác vụ trước khi chuyển đến RAM.

1. Bật nguồn. Vector bộ xử lý thành 0xFFFFFFFFC nơi i đặt lệnh nhảy chuyển quyền kiểm soát tới bootrom tại địa chỉ _romInit.
2. _romInit: Chức năng hợp ngữ này chạy hết ROM lưu ý rằng đây là khởi động ngoài rồi nhảy để bắt đầu. Cả _romInit và start đều nằm trong tệp mã nguồn bspname / romInit.s.
3. start: Chức năng hợp ngữ này chạy hết ROM sẽ thiết lập bộ xử lý, làm mất hiệu lực của bộ nhớ đệm và chuẩn bị cho hệ thống hoạt động trên RAM. Thao tác cuối cùng là gọi hàm C romStart () được thực hiện bởi Wind River và nằm trong tệp mã nguồn \$ WIND_BASE / target / config / all / bootInit.c.
4. romStart (): Hàm C này chạy hết ROM sẽ sao chép VxWorks vào địa chỉ bắt đầu RAM của nó nằm tại RAM_HIGH_ADRS (hằng số này được định nghĩa trong config.h và Makefile). Sau khi sao chép VxWorks, quyền điều khiển được chuyển đến hàm usrInit () trong RAM.
5. Thực hiện theo các bước 2 và 3 của "[Trình tự khởi động VxWorks](#)", trang 11.

Bootroms

Bootrom là một hình ảnh VxWorks được thu nhỏ lại, hoạt động theo cách giống như BIOS của PC. Công việc chính của nó là tìm và khởi động một hình ảnh VxWorks đầy đủ. Hình ảnh VxWorks đầy đủ có thể nằm trên đĩa, trong bộ nhớ flash hoặc trên một số máy chủ thông qua Ethernet. Bootrom phải được biên dịch theo cách mà nó có khả năng truy xuất hình ảnh. Nếu hình ảnh được truy xuất từ mạng Ethernet, thì bootrom phải có ngắn xếp TCP / IP được biên dịch trong, nếu hình ảnh nằm trên đĩa, thì bootrom phải có hỗ trợ truy cập đĩa được biên dịch trong, v.v. Các bootrom truy xuất và bắt đầu hình ảnh đầy đủ và duy trì một dòng khởi động. Dòng khởi động là một chuỗi văn bản đặt các đặc điểm người dùng nhất định như địa chỉ IP của mục tiêu nếu sử dụng Ethernet và đường dẫn tệp đến hình ảnh VxWorks để khởi động.

Bootrom không phải là một yêu cầu bắt buộc. Chúng thường được sử dụng trong môi trường phát triển sau đó được thay thế bằng hình ảnh VxWorks sẵn xuất.

Tạo Bootroms

Tại trình bao lệnh trong thư mục BSP, hãy phát hành lệnh sau để tạo hình ảnh bootrom không nén (bắt buộc đối với SystemACE):

tạo bootrom_uncmp

hoặc

làm bootrom

để tạo ra một hình ảnh nén phù hợp để đặt trong một mảng bộ nhớ flash.

Hiển thị Bootrom

Khi khởi động nguồn, nếu các bootrom hoạt động bình thường, điều ra tương tự như sau sẽ được nhìn thấy trên cổng nối tiếp của bảng điều khiển:

Khởi động hệ thống VxWorks

Bản quyền 1984-2008 Wind River Systems, Inc.

CPU: Xilinx Virtex5 ppc440x5

Phiên bản: VxWorks 6.7

Phiên bản BSP: 2.0 / 0.

Ngày tạo: 11/07/2009, 16:40:32

Nhấn phím bất kỳ để dừng tự động khởi động ...

3

[VxWorks Boot]:

Trợ giúp nhập tại dấu nháy này liệt kê các lệnh có sẵn.

Bootline

Dòng khởi động là một chuỗi văn bản xác định các đặc điểm có thể phục vụ của người dùng như địa chỉ IP của bảng đích và cách tìm hình ảnh vxWorks để khởi động. Dòng khởi động được bootrom duy trì trong thời gian chạy và thường được giữ trong một số vùng lưu trữ không bay hơi (NVRAM) của hệ thống như EEPROM hoặc bộ nhớ flash. Nếu không có NVRAM hoặc xảy ra lỗi khi đọc nó, thì dòng khởi động được mã hóa cứng bằng DEFAULT_BOOT_LINE được xác định trong tệp mã nguồn config.h của BSP. Trong các hệ thống mới mà NVRAM chưa được khởi tạo, dòng khởi động có thể là dữ liệu không xác định.

Dòng khởi động có thể được thay đổi nếu chuỗi đếm người tự động khởi động bị gián đoạn bằng cách nhập một ký tự trên cổng nối tiếp bảng điều khiển. Sau đó, lệnh có thể được sử dụng để chỉnh sửa tương tác dòng khởi động. Nhập p để xem dòng khởi động. Trên ảnh không phải bootrom, dòng khởi động có thể được thay đổi bằng cách nhập lệnh bootChange tại dấu nháy máy chủ hoặc trình bao đích.

Các truy ống dòng khởi động được định nghĩa bên dưới:

- boot device: Thiết bị khởi động từ. Đây có thể là Ethernet hoặc một đĩa cục bộ. Lưu ý rằng khi thay đổi dòng khởi động, số đơn vị có thể được hiển thị thêm vào truy ống này (l1temac0) khi nháy thiết bị khởi động mới. Con số này có thể được bỏ qua.
- số bộ xử lý: Luôn luôn là 0 với các hệ thống bộ xử lý đơn lẻ.
- tên máy chủ: Đặt tên khi cần thiết.
- tên tệp: Hình ảnh VxWorks để khởi động. - inet on
- ethernet (e): Địa chỉ Internet IP của mục tiêu. Nếu không có mạng giao diện, sau đó truy ống này có thể được để trống.
- host inet (h): Địa chỉ Internet IP của máy chủ. Nếu không có giao diện mạng, thì truy ống này có thể được để trống.
- user (u): Tên người dùng bạn chọn để truy cập hệ thống tệp máy chủ. Máy chủ FTP của bạn phải được thiết lập để cho phép người dùng này truy cập vào hệ thống tệp máy chủ.
- Mật khẩu ftp (pw): Mật khẩu bạn chọn để truy cập hệ thống tệp máy chủ. Máy chủ FTP của bạn phải được thiết lập để cho phép người dùng này truy cập vào hệ thống tệp máy chủ.
- flags (f): Để biết danh sách các tùy chọn, hãy nhập lệnh trợ giúp tại [VxWorks Boot]: lời nhắc.
- target name (tn): Đặt tên khi cần thiết. Đặt theo yêu cầu mạng.

- other (o): Trong trường hợp này hữu ích khi bạn có một thiết bị không phải ethernet làm thiết bị khởi động. Trong trường hợp này, VxWorks sẽ không khởi động mạng khi khởi động. Chỉ định thiết bị Ethernet ở đây sẽ kích hoạt thiết bị đó tại thời điểm khởi động với các tham số mạng được chỉ định trong các trường dòng khởi động khác.
- inet trên bảng nối đa năng (b): Thư ờng để trống nếu hệ thống đích không nằm trên VME hoặc PCI bảng nối đa năng.
- gateway inet (g): Nhập địa chỉ IP vào đây nếu bạn phải đi qua cổng vào tiếp cận máy tính chủ. Nếu không thì để trống.
- (các) tập lệnh khởi động: Đường dẫn đến tập trên máy tính chủ chứa các lệnh shell để thực thi sau khi khởi động xong. Để trống nếu không sử dụng tập lệnh. Ví dụ:

Tập lệnh thư ờng trú trên máy chủ: c: /temp/myscript.txt

Bootrom với Ethernet ba chế độ liên kết cục bộ (LLTEMAC) làm thiết bị khởi động

SDK sẽ tạo một BSP có khả năng được xây dựng như một bootrom bằng cách sử dụng LLTEMAC làm thiết bị khởi động. Cấu hình ngẫu nhiên dùng rất ít được yêu cầu. Địa chỉ MAC được mã hóa cứng trong tệp nguồn hwconf.c. BSP có thể được sử dụng với MAC mặc định miễn là mục tiêu nằm trên một mạng riêng và không có nhiều hơn một mục tiêu trên mạng đó có cùng địa chỉ MAC mặc định. Nếu không, nhà thiết kế nên thay thế MAC này bằng một chức năng để lấy một MAC từ thiết bị nhớ không bay hơi trên bảng đích của họ.

Để chỉ định LLTEMAC làm thiết bị khởi động trong bootrom, hãy thay đổi trường thiết bị khởi động trong dòng khởi động thành lltemac. Nếu chỉ có một LLTEMAC, hãy đặt số đơn vị thành 0.

Ví dụ sau khởi động từ ethernet bằng Xilinx lltemac làm thiết bị khởi động.

Hình ảnh được khởi động nằm trên hệ thống tệp máy chủ trên ổ C.

```

thiết bị khởi động          : lltemac
số đơn vị                 : 0
tên máy chủ số bộ xử lý   : 0
                           : chủ nhà
tên tập tin                : c: /WindRiver/vxworks-6.7/target/config/ml507/vxWorks
inet trên ethernet (e): 192.168.0.2
máy chủ inet (h)           : 192.168.0.1
nguồn dùng (u)             : xemhost
mật khẩu ftp (pw) cờ (f)  : sao cung được
tên đích (tn) khác (o)    : 0x0
                           : vxtarget
                           :

```

Bộ nhớ đệm

Hướng dẫn và bộ nhớ đệm dữ liệu được quản lý bởi các thư viện bộ nhớ VxWorks. Chúng được kích hoạt bằng cách sửa đổi các hàng số sau trong config.h hoặc bằng cách sử dụng cơ sở Dự án Workbench để thay đổi các hàng số có cùng tên:

- INCLUDE_CACHE_SUPPORT: Nếu được xác định, các thư viện bộ nhớ VxWorks được liên kết với hình ảnh. Nếu bộ nhớ đệm không được mong muốn, thì hãy # hoàn thiện hàng số này.
- USER_I_CACHE_ENABLE: Nếu được định nghĩa, VxWorks sẽ kích hoạt bộ đệm chỉ lệnh lúc khởi động. Yêu cầu INCLUDE_CACHE_SUPPORT được xác định để có bất kỳ tác dụng nào.
- USER_D_CACHE_ENABLE: Nếu được định nghĩa, VxWorks sẽ kích hoạt bộ đệm dữ liệu tại thời điểm khởi động. Yêu cầu INCLUDE_CACHE_SUPPORT được xác định để có bất kỳ tác dụng nào.

MMU

Nếu MMU được bật, thì điều khiển bộ đệm được thảo luận trong phần trư ớc có thể không có bất kỳ tác dụng nào. MMU được quản lý bởi các thư viện độc quyền của VxWorks nhưng thiết lập ban đầu được xác định trong BSP. Để bật MMU, hằng số INCLUDE_MMU_BASIC phải được xác định trong config.h hoặc bằng cách sử dụng Project Facility. Hằng số USER_D_MMU_ENABLE và USER_I_MMU_ENABLE kiểm soát liệu MMU hoạt động dẫn và / hoặc dữ liệu có được sử dụng hay không.

VxWorks khởi tạo MMU dựa trên dữ liệu trong cấu trúc sysPhysMemDesc được định nghĩa trong sysCache.c. Bộ nhớ dành riêng cho người dùng và ED&R (khi INCLUDE_EDR_PM được bật) bộ nhớ dành riêng được bao gồm trong bảng này. Trong số những thứ khác, bảng này định cấu hình các vùng bộ nhớ với các thuộc tính sau:

- Cho phép thực hiện lệnh hay không
- Cho phép ghi dữ liệu
- Thuộc tính lệnh và bộ nhớ cache dữ liệu
- Phần bù dịch được sử dụng để tạo địa chỉ

Khi VxWorks khởi tạo MMU, nó sẽ lấy các định nghĩa từ sysPhysMemDesc và tạo các mục nhập bảng trang (PTE) trong RAM. Mỗi PTE mô tả 4KB vùng bộ nhớ (mặc dù bộ xử lý có khả năng đại diện lên đến 16MB cho mỗi PTE.) Hãy lưu ý rằng việc chỉ định các vùng bộ nhớ lớn sử dụng lượng RAM đáng kể để lưu trữ các PTE. Để ánh xạ 4MB không gian bộ nhớ liền kề, cần 8KB RAM để lưu trữ các PTE.

Để tăng hiệu suất với gói MMU cơ bản của VxWorks cho bộ xử lý PowerPC 440, có thể có lợi nếu không bật MMU hoạt động dẫn và dựa vào cài đặt điều khiển bộ đệm trong thanh ghi ICCR. Chiến lược này có thể giảm đáng kể số lượng lỗi trang trong khi vẫn giữ các hướng dẫn trong bộ nhớ cache. Cài đặt ban đầu của ICCR được xác định trong tệp tiêu đề bspname.h .

Đối với PPC440, bộ nhớ đệm và MMU được bật theo mặc định.

Nếu không bật MMU, các quy tắc sau áp dụng cho việc định cấu hình các thuộc tính truy cập bộ nhớ và bộ nhớ đệm:

- Không có bản dịch địa chỉ, tất cả các địa chỉ hiệu quả là vật lý.
- Mức độ chi tiết của kiểm soát bộ nhớ cache là 128MB.

FPU

Đơn vị dấu chấm động cứng (FPU) được hỗ trợ cho các hệ thống PPC440. Để bật đơn vị dấu phẩy động cứng, vui lòng chọn diab hoặc gnu trong CÔNG CỤ Makefile BSP đã tạo. Để tắt đơn vị dấu phẩy động cứng, hãy chọn sfdiab hoặc sfgnu trong biến TOOLS make của Makefile.



UG708

Tự động tạo gói hỗ trợ bo mạch Linux 2.6

Bản tóm tắt

Tài liệu này mô tả việc tạo tự động Gói hỗ trợ bo mạch Linux 2.6 (BSP) thông qua Bộ phát triển phần mềm Xilinx® (SDK) (1). Tài liệu bao gồm các phần sau:

- “[Tổng quan](#)”
- “[Bắt đầu với Linux 2.6](#)”
- “[Tạo BSP từ SDK](#)”
- “[Cấu trúc thư mục](#)”
- “[Định cấu hình nhân Linux](#)”
- “[Tham khảo thiết bị Linux](#)”
- “[Thông tin liên quan](#)”

Tổng quan

Trong môi trường phát triển nhúng điển hình, một trong những nhiệm vụ là tạo phần mềm để hỗ trợ phần cứng tùy chỉnh trên hệ thống nhúng cho O / S đích. Phần mềm hỗ trợ phần cứng tùy chỉnh nhúng này thường được gọi là Gói hỗ trợ bảng (BSP). Trong môi trường mà phần cứng được xác định trong Hệ thống trên chip (SoC) có thể lập trình được, các thay đổi phần cứng có thể xảy ra nhanh hơn nhiều, khiến BSP khó duy trì tính cập nhật với các bản sửa đổi trong phần cứng.

Để giảm bớt tình trạng này, Xilinx cung cấp một quy trình được gọi là Tự động tạo BSP để điều chỉnh BSP theo cấu hình phần cứng hiện tại của FPGA.

Tự động tạo BSP được thực hiện bằng cách sử dụng Xilinx SDK, có sẵn trong Bộ phát triển nhúng Xilinx (EDK) hoặc dưới dạng công cụ được cài đặt riêng. SDK tạo BSP dựa trên cấu hình phần cứng đã xác định. Đối với Linux, SDK tạo cây nguồn nhân Linux thưa thừa chỉ chứa các tệp phần cứng cụ thể cho BSP. Đối với Linux 2.6, SDK hỗ trợ cả bản phân phối MontaVista và Wind River Linux.

Nói chung, luồng công việc để sử dụng Linux trên hệ thống nhúng sử dụng FPGA như sau:

1. Xác định các thành phần phần cứng trong Xilinx Platform Studio (XPS) và xuất dự án sang SDK Xilinx.
2. Chọn bản phân phối Linux 2.6 làm hệ điều hành mục tiêu trong SDK.
3. Chỉ định các thông số hệ điều hành.
4. Tạo BSP trong SDK.
5. Cấu hình hạt nhân.
6. Xác định hệ thống tệp gốc.
7. Xây dựng nhân.

-
1. Xilinx SDK được sử dụng làm môi trường phát triển phần mềm chính cho người dùng Xilinx Embedded Development Kit (EDK) kể từ EDK 11.1i. Khả năng phát triển phần mềm của Xilinx Platform Studio (XPS) hiện không được dùng nữa và sẽ bị xóa khỏi XPS trong các bản phát hành sau. Các luồng được mô tả trong tài liệu này liên quan đến SDK, mặc dù chúng vẫn có thể áp dụng chung cho XPS trong khi các tính năng đó vẫn còn trong công cụ.

8. Cài đặt hạt nhân và hệ thống tệp gốc
9. Phát triển và chạy mã ứng dụng cụ thể

Hướng dẫn này mô tả các bước từ 2 đến 5 và 7. Các bước còn lại nằm ngoài phạm vi của hướng dẫn này.

Bắt đầu với Linux 2.6

Các bản phân phối Linux 2.6 hiện được hỗ trợ trong SDK là của MontaVista và Wind River.

Các bản phân phối này có thể được mua trực tiếp từ các nhà cung cấp đó. Các sản phẩm MontaVista được đặt tên là Linux Professional Edition 4.0.1 và 5.0, mỗi sản phẩm bao gồm một cây nguồn nhân, các công cụ phát triển và hỗ trợ kỹ thuật. Các sản phẩm của Wind River có tên là General Purpose Platform, Linux Edition 1.3 và Linux Edition 2.0. Các sản phẩm Linux này cung cấp môi trường phát triển chéo PowerPC 4xx chạy trên các hệ điều hành máy chủ khác nhau.

Xem trang web của nhà cung cấp để biết danh sách các hệ điều hành máy chủ được hỗ trợ.

Để bắt đầu, trước tiên hãy cài đặt đĩa CD phân phối MontaVista hoặc Wind River Linux cho bộ xử lý PowerPC® 4xx. Nếu sử dụng FPU trên Virtex-5 FXT, hãy cài đặt các công cụ biên dịch cho bộ xử lý PowerPC 440. Sau khi bản phân phối chính được cài đặt, mỗi nhà cung cấp cung cấp một đĩa CD Xilinx BSP hoặc hình ảnh tải xuống có thể được cài đặt trên bản phân phối chính. Vui lòng làm theo hướng dẫn cài đặt dành riêng cho nhà cung cấp.

MontaVista sử dụng thuật ngữ LSP thay vì BSP. LSP là viết tắt của Gói hỗ trợ Linux, nhưng nên được coi là tương tự với Gói hỗ trợ bảng. Các BSP Linux 2.6 được cung cấp dành cho các thiết kế tham chiếu cụ thể cho các bảng phát triển Xilinx ML507 và ML403. Các thiết kế tham khảo có thể được tìm thấy trên trang web của Xilinx tại www.xilinx.com/ml507 và www.xilinx.com/ml403 tương ứng. Khi phát triển thiết kế phần cứng tùy chỉnh cho các bo mạch này hoặc cho các bo mạch khác, người dùng nên sử dụng BSP được tạo tự động từ Platform Studio kết hợp với một trong các BSP nói trên.

Lưu ý: Xilinx đã thử nghiệm các BSP từ MontaVista và Wind River cho các bảng phát triển (ml403 và ml507) mà không áp dụng bất kỳ bản vá hoặc bản cập nhật nào của nhà cung cấp Linux. Các bản vá và cập nhật từ các nhà cung cấp Linux này được khuyến nghị. Tuy nhiên, nếu bạn đang gặp sự cố, bạn có thể muốn thử trước mà không có các bản vá và cập nhật.

Khi xây dựng thiết kế phần cứng trên bo mạch tùy chỉnh, FPGA mục tiêu cần phải có bộ xử lý PowerPC. Cổng nối tiếp hoặc một số thiết bị có thể được sử dụng như một thiết bị bảng điều khiển sẽ rất hữu ích. Ngoài ra, trừ khi đĩa ram sẽ được sử dụng, một số thiết bị sẽ được sử dụng để truy cập hệ thống tệp gốc cần được xem xét (ví dụ: Ethernet cho hệ thống tệp gốc NFS hoặc System ACE cho hệ thống tệp gốc CompactFlash).

Tạo một cây nhân làm việc

Bạn nên tạo một bản sao đang hoạt động của cây nhân Linux được cài đặt với bản phân phối MontaVista hoặc Wind River của bạn. Điều này đảm bảo rằng bản sao đã cài đặt được giữ nguyên bản.

MontaVista Linux

Khi được cài đặt trên hệ thống Linux, thư mục cài đặt mặc định cho nguồn hạt nhân MontaVista Linux, cho MontaVista Linux Professional Edition, ở đây:

```
/ opt / montavista / pro / devkit / lsp / <target board> /linux-2.6.10_mvl401
```

Cần phải cẩn thận một số để sao chép chính xác cây nguồn hạt nhân để bảo vệ các liên kết và các thuộc tính tệp khác. Một phư ơng pháp là sử dụng tar để tạo một tarball của cây nguồn và sau đó giải nén nó đến vị trí đích. Phư ơng pháp tar này thậm chí có thể được thực hiện bằng cách sử dụng một tệp tar tạm thời bằng cách đưa ra của quá trình tạo tar vào quy trình trích xuất tar như sau:

```
tar cf - source_dir | tar xvf - -C target_dir
```

Đây là một ví dụ cụ thể:

```
tar cf - / opt / montavista / pro / devkit / lsp / xilinx-ml40x-ppc_405 / linux
2.6.10_mvl401 | tar xvf - -C my_linux-2.6.10
```

Tren Windows, thực hiện sao chép trong một trình bao bash cygwin để các thuộc tính tệp Linux có thể được bảo toàn, vì bản dựng nhân Linux phụ thuộc vào một số liên kết mềm nhất định.

Wind River Linux

Các bước để tạo cây nhân Linux hoạt động cho Wind River Linux khác với MontaVista Linux. Tham khảo hướng dẫn Bắt đầu trong bản phân phối Wind River Linux của bạn để biết chi tiết về cách tạo hạt nhân và hệ thống tệp bằng phương pháp RPM để tạo sẵn hoặc phương pháp tạo nguồn hoặc sử dụng Workbench IDE. Các bước sau đây mô tả việc tạo một hạt nhân làm việc từ dòng lệnh bằng cách sử dụng phương pháp xây dựng nguồn. Lưu ý rằng các bước này tránh xây dựng hệ thống tệp.

- Tạo một thư mục làm việc nơi bạn muốn cây nhân cư trú
- Từ thư mục làm việc, chạy tập lệnh cấu hình để sao chép và định cấu hình cây nhân cho BSP cụ thể mà bạn đang nhắm mục tiêu. Ví dụ, đối với ML403 BSP:


```
WINDRIVER_INSTALL_DIR / wrlinux-1.3 / wrlinux / config --enable-kernel = cgl -enable-board = xilinx_ml403
```
- Gõ "make -C dist linux.rebuild" để xây dựng cây nhân đang hoạt động, cây này sẽ nằm trong dist / linux-2.6.14-cgl trong thư mục làm việc (dành cho Wind River GPP LE 1.3).

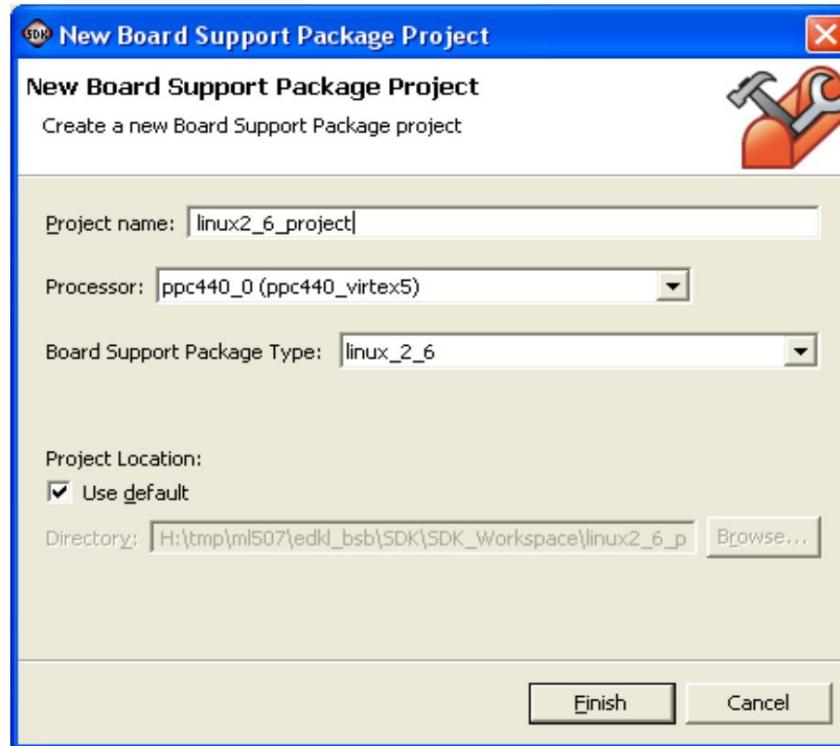
Tạo BSP từ SDK

SDK có sẵn dưới dạng công cụ được cài đặt riêng hoặc trong EDK và là môi trường phát triển phần mềm để phát triển phần mềm nhúng xung quanh bộ xử lý PowerPC 405/440 hoặc hệ thống nhúng dựa trên MicroBlaze™. Phần này mô tả các bước cần thiết để tạo BSP Linux 2.6 bằng SDK. Các bước này có thể áp dụng khi sử dụng công cụ The Xilinx 11.1i trở lên.

Giả định rằng một thiết kế phần cứng hợp lệ đã được tạo và xuất sang SDK, và SDK đã được mở và trỏ đến thiết kế phần cứng.

Tạo dự án gói hỗ trợ hội đồng quản trị

Sau khi các thành phần phần cứng đã được xác định và định cấu hình trong XPS và được xuất sang SDK, thì dự án Gói hỗ trợ bảng phải được tạo với SDK để chọn Hệ điều hành mục tiêu. Chọn mục Gói Hỗ trợ Bảng từ menu Tệp > Mới để mở hộp thoại Gói Hỗ trợ Bảng Mới. Xem [Hình 1, trang 4](#).

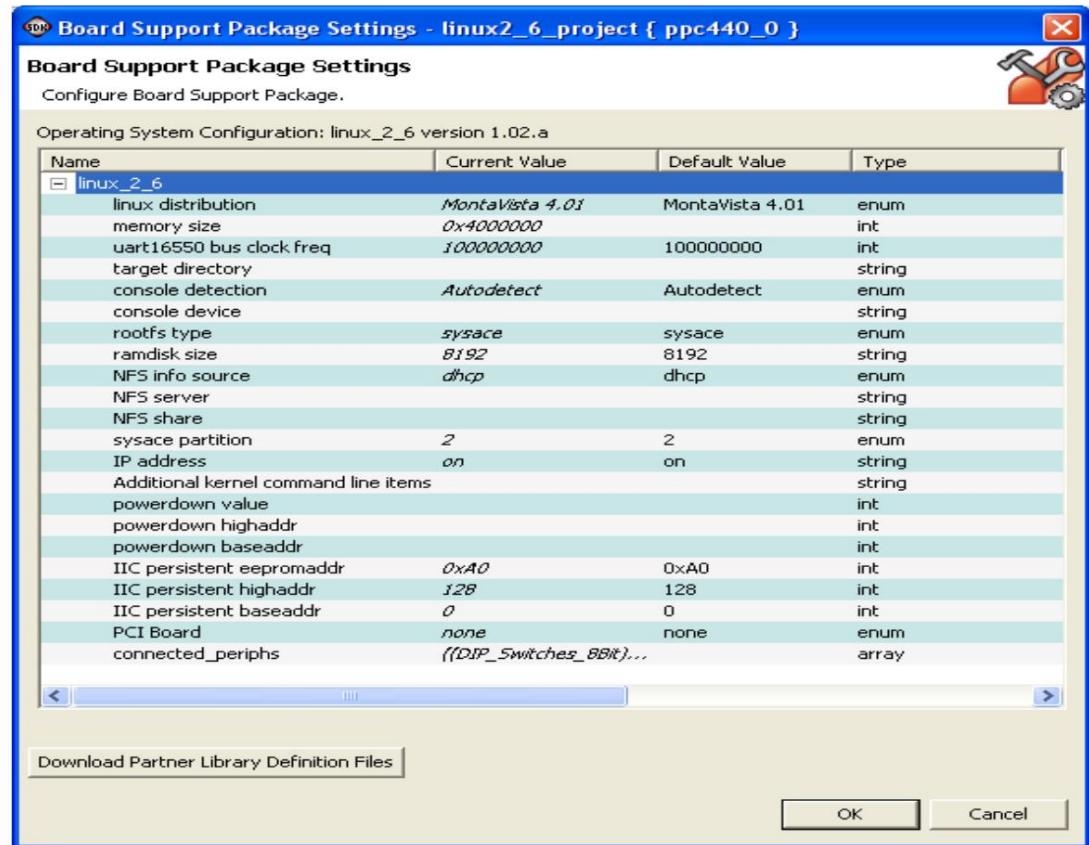


Hình 1: Tạo dự án gói hỗ trợ hội đồng quản trị

Nhập tên dự án và chọn linux_2_6 từ danh sách thả xuống Loại Gói Hỗ trợ Bảng và nhấp vào Kết thúc để khởi chạy hộp thoại Cài đặt Gói Hỗ trợ Bảng.

Cấu hình gói hỗ trợ hội đồng quản trị

Có một số tùy chọn cấu hình có sẵn từ bên trong hộp thoại Cài đặt Gói Hỗ trợ Bảng như thể hiện trong [Hình 2, trang 5](#).



Hình 2: Thiết lập các thông số thư viện / hệ điều hành

Các tùy chọn, một số trong số đó là bắt buộc và một số là tùy chọn, như sau:

phân phối linux (bắt buộc)

Tham số này chỉ định bản phân phối bạn đang sử dụng, Wind River Linux hoặc MontaVista Linux. Mặc định là MontaVista Linux. Hạt nhân Linux của bạn có thể không được xây dựng đúng cách nếu thông số này được đặt không chính xác. Tham số này xác định nội dung của tệp Kconfig.

kích thước bộ nhớ (bắt buộc)

Cài đặt này chỉ cho phép HĐH biết dung lượng RAM dành cho mục đích chung mà nó có thể sử dụng trong hệ thống. Rõ ràng giá trị này phải nhỏ hơn hoặc bằng dung lượng RAM vật lý dành cho mục đích chung có sẵn trong hệ thống. Mặc dù vậy, nó có thể được đặt thành một giá trị nhỏ hơn dung lượng RAM vật lý dành cho mục đích chung nếu mở rộng một lượng RAM nhỏ hơn hoặc nếu một số vùng RAM được dành cho các mục đích khác.

Lưu ý rằng cấu hình phân cứng nên được đặt để bộ nhớ bắt đầu ở địa chỉ 0x0.

Tần suất đồng hồ xe buýt UART16550 (tùy chọn)

Tham số này chỉ định tần số của bus (tính bằng Hz) mà thiết bị nối tiếp bằng điều khiển được gắn vào. Nhân Linux sử dụng giá trị này để lập trình tốc độ truyền 16550/16450 UART. Lưu ý rằng cài đặt này chỉ được yêu cầu nếu UART 16550 được bao gồm trong thiết kế phân cứng.

thư mục đích (tùy chọn)

Thư mục đích nơi BSP được tạo có thể được chỉ định. Thông thường, giá trị trả đến một bản sao của cây nguồn nhân Linux để BSP được tạo sẽ phủ trực tiếp cây nhân đang hoạt động với các trình điều khiển mới. Để lớp phủ như vậy hoạt động chính xác, target_dir phải trả đến thư mục trên cùng trong cây nhân đang hoạt động.

Nếu thư mục đích này bị bỏ trống, nó sẽ mặc định là:

thư mục dự án / tên bộ xử lý / libsrc / linux_2_6_v1_02_a / linux

Thư mục này sẽ chứa một cây nhân thư a thớt với các trình điều khiển thiết bị được cập nhật được cấu hình để phù hợp với thiết kế phần cứng. Thư mục này sau đó sẽ được sao chép qua cây nhân Linux đang hoạt động của người dùng. Vui lòng sử dụng dấu gạch chéo để phân cách tên thư mục.

loại rootfs (tùy chọn)

Tham số kiểu rootfs chỉ định kiểu hệ thống tệp gốc nào sẽ được sử dụng cho cây nhân này. Lưu ý rằng đây là kiểu mặc định ban đầu và nó luôn có thể được thay đổi trong .config hạt nhân của bạn.

Danh sách thả xuống trong cột Giá trị Hiện tại cung cấp cho người dùng lựa chọn nfs, ramdisk hoặc sysace.

Lựa chọn hiển thị trong dòng lệnh kernel mặc định. Giá trị mặc định là sysace.

Lưu ý rằng tính đến thời điểm viết bài này, Wind River Linux không chính thức hỗ trợ rootfs ramdisk.

kích thước đĩa ram (tùy chọn)

Tham số này chỉ định kích thước của đĩa ram nếu đĩa ram được chọn cho loại rootfs. Giá trị mặc định là 8192 khối byte 1K (8 MB). Lưu ý rằng đây là giá trị mặc định ban đầu và nó có thể được thay đổi trong .config hạt nhân của bạn.

Nguồn thông tin NFS (tùy chọn)

Tham số này chỉ định cách hệ thống tệp gốc NFS sẽ được truy xuất trong quá trình khởi động nếu nfs được chọn cho loại rootfs. Giá trị mặc định là dhcp, có nghĩa là thông tin NFS sẽ được lấy từ máy chủ DHCP. Tùy chọn khác là chọn dòng lệnh kernel để lấy thông tin NFS trực tiếp từ chuỗi lệnh kernel (ví dụ: nfsroot =).

Máy chủ NFS (tùy chọn)

Tham số này chỉ định tên của máy chủ NFS mà từ đó hệ thống tệp gốc sẽ được gắn trong quá trình khởi động nếu nfs được chọn cho loại rootfs và dòng lệnh hạt nhân được chọn cho nguồn thông tin NFS.

Chia sẻ NFS (tùy chọn)

Tham số này chỉ định tên của chia sẻ NFS trên máy chủ NFS. Tham số này chỉ được sử dụng nếu tên máy chủ NFS được cung cấp và nfs được chọn cho loại rootfs và dòng lệnh hạt nhân được chọn cho nguồn thông tin NFS.

phân vùng sysace (tùy chọn)

Tham số này chỉ định phân vùng thẻ CompactFlash chứa hệ thống tệp gốc.

Tham số này chỉ được sử dụng nếu sysace được chọn cho loại rootfs. Giá trị mặc định là 2.

Địa chỉ IP (tùy chọn)

Tham số này có thể được đặt thành bật, tắt hoặc một địa chỉ IP tĩnh. Nếu bật được chọn, DHCP sẽ được sử dụng để truy xuất địa chỉ IP của mục tiêu trong quá trình khởi động. Nếu tắt được chọn, mạng sẽ bị vô hiệu hóa trong quá trình khởi động. Nếu địa chỉ IP tĩnh được chỉ định, địa chỉ IP này sẽ được gán cho giao diện Ethernet chính trong quá trình khởi động. Giá trị mặc định được bật.

Các mục dòng lệnh kernel bổ sung (tùy chọn)

Tham số này có thể được sử dụng để chỉ định các tùy chọn dòng lệnh bổ sung nếu không được quyết bằng các tùy chọn nêu trên. Ví dụ: các tùy chọn cho cấu hình kgdb hoặc thay đổi thiết bị bằng điều khiển có thể được chỉ định tại đây.

Thông số tắt nguồn (tùy chọn)

Các tham số tắt nguồn là trình giữ chỗ để hỗ trợ tạo tính năng ngắt nguồn mềm trong bảng của bạn. Một số bo mạch Xilinx hỗ trợ tính năng tắt nguồn mềm thông qua địa chỉ GPIO.

Lưu ý: Tham số này chỉ áp dụng nếu bạn có tính năng tắt nguồn trên bo mạch và nó có thể truy cập được thông qua địa chỉ ánh xạ bộ nhớ (ví dụ: GPIO). Vui lòng xem hướng dẫn sử dụng bảng của bạn như một tài liệu tham khảo.

Các giá trị này nhằm hỗ trợ một phương pháp tắt nguồn trong đó giá trị tắt nguồn được ghi vào powerdown baseaddr để bắt đầu trình tự tắt nguồn phần cứng. Phần mềm powerdown highaddr được sử dụng để chỉ ra một phạm vi bộ nhớ được sử dụng để ánh xạ các trang với một tập hợp các trang vật lý.

Thông số IIC (tùy chọn)

Các thông số IIC dựa trên phần cứng trên bo mạch Xilinx ML403 và ML507.

Bo mạch có gắn EEPROM vào bus I2C. Ngoài ra, Linux được thiết lập để đọc địa chỉ MAC cho trình điều khiển Ethernet từ một địa chỉ trên EEPROM này. Các tham số IIC chỉ định địa chỉ nào trong EEPROM sẽ được sử dụng để đọc địa chỉ MAC này cũng như chỉ định ID thiết bị của EEPROM trên bus I2C.

Nếu bo mạch của bạn không có EEPROM trên bus I2C, các tham số này có thể được bỏ qua một cách an toàn.

Giá trị baseaddr liên tục của IIC chỉ định địa chỉ cơ sở trong EEPROM nơi lưu trữ địa chỉ MAC.

Giá trị highaddr liên tục của IIC không được sử dụng để khởi động. Giá trị này được sử dụng bởi các tiện ích khác có sẵn với bảng Xilinx ghi vào EEPROM.

Giá trị eepromaddr liên tục IIC chỉ định ID thiết bị bus I2C của EEPROM.

Bảng PCI (tùy chọn)

Tham số này chỉ định bảng đích cho hệ thống PCI. Hiện tại, chỉ có bo mạch Xilinx ML410 và ML510 được hỗ trợ. Ví dụ: khi tham số này được đặt thành "ml410", các thiết bị ngoại vi PCI trên bo mạch phổ biến nhất được bật trong cấu hình nhân Linux thông qua cấu hình nhân tự động trong Platform Studio (xem phần Cấu hình nhân Linux).

kết nối_periph (bắt buộc)

Tham số này chỉ định thiết bị phần cứng nào sẽ được hỗ trợ trong Linux thông qua BSP được tạo. Xem [Bảng 1, trang 24](#). Nhập vào cột Giá trị Hiện tại sẽ xuất hiện hộp thoại trong đó bạn có thể chỉ định thiết bị ngoại vi nào sẽ được kết nối với Hệ điều hành. Theo mặc định, tham số này có danh sách tất cả các thiết bị ngoại vi trong thiết kế phần cứng. Trong hầu hết các trường hợp, thông số này có thể được giữ nguyên.

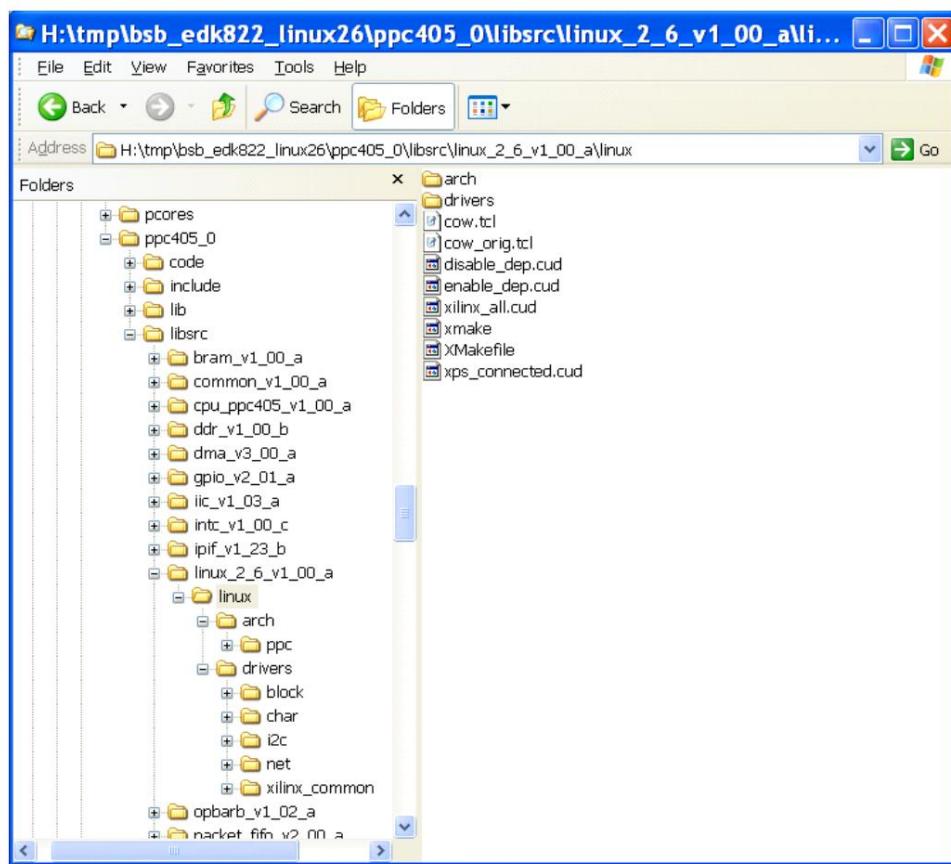
Tạo BSP

Chọn OK trên hộp thoại Cài đặt Gói Hỗ trợ Hội đồng quản trị để tạo BSP.

Danh mục cấu trúc

Nếu thư mục đích bị bỏ trống, thư mục đích sẽ được mặc định là thư mục dự án \ tên bộ xử lý \ libsrc \ linux_2_6_v1_02_c \ linux.

Thư mục Linux được hiển thị trong [Hình 3, trang 8](#), chứa một cây thư mục gồm các trình điều khiển Linux khác nhau cho các thiết bị phần cứng hiện được cấu hình. Nếu thư mục đích được chỉ định không tham chiếu đến cùng một thư mục với cây nguồn nhân Linux đang hoạt động, hãy sao chép thư mục này vào cây nguồn nhân Linux đang hoạt động.



Hình 3: Cấu trúc thư mục mẫu cho BSP Linux đã tạo

Sao chép BSP sang Cây nguồn nhân Linux

Nếu tùy chọn thư mục đích trong cài đặt Gói Hỗ trợ Hội đồng quản trị để cập đến cây nguồn hạt nhân Linux đang hoạt động của bạn, thì BSP đã tạo không cần phải sao chép. Nếu không, BSP đã tạo cần được sao chép trên đầu cây nguồn hạt nhân đang hoạt động. Các thư mục cao nhất trong BSP được tạo khớp với một số thư mục trong cấu trúc thư mục nhân Linux đang hoạt động.

Sao chép các tệp BSP đã tạo để các tệp trong thư mục / Arch và / drivers lần lượt đi vào thư mục / Arch và / drivers trong cây nguồn hạt nhân đang hoạt động.

Cần cẩn thận khi sao chép BSP bằng giao thức SMB (Mạng Windows).

Khi sử dụng giao thức chia sẻ tệp này, các liên kết tự động trỏ đến các thư mục trên đích có thể bị ghi đè thành các thư mục riêng biệt. Việc xây dựng nhân Linux thường như yêu cầu các liên kết tự động trỏ phải có mặt hơn là có các thư mục riêng biệt. Đặc biệt, hãy chú ý đến / asm các thư mục.

Định cấu hình Nền tảng Linux

Quy trình tạo BSP của Linux 2.6 thông qua SDK khác với quy trình của Linux 2.4 ở chỗ, quá trình tùy biến hạt nhân tốt hơn được thực hiện thông qua quy trình. Điều này dẫn đến việc người dùng cần thực hiện rất ít, nếu có, tùy chỉnh hạt nhân ngoài tùy chỉnh dành riêng cho ứng dụng. Đối với MontaVista Linux 4.0.1 và WindRiver Linux 1.3 / 2.0, Cấu hình hạt nhân có thể được cập nhật để kích hoạt trình điều khiển cho thiết bị trong thiết kế phần cứng. Đối với MontaVista Linux 5.0, tất cả các trình điều khiển liên quan đến Xilinx đều được bật theo mặc định và các thiết bị được bật tại thời điểm chạy thông qua việc sử dụng cấu trúc cây thiết bị.

Có ba phương pháp để định cấu hình hạt nhân của bạn để phù hợp với thiết kế phần cứng FPGA:

1. Thư mục đích trỏ đến cây nhân Linux 2.6 hợp lệ.

Phương pháp này được khuyến nghị nếu bắt đầu với nguồn nhân Linux chưa sửa đổi vì nó tự động cập nhật tệp .config của nhân để phù hợp với thiết kế phần cứng. Phương pháp này không được khuyến nghị nếu các sửa đổi hạt nhân đã được thực hiện và cần được bảo tồn.

Người dùng không cần phải cấu hình kernel theo cách thủ công (ví dụ bằng cách chạy make menuconfig) để thay đổi các tùy chọn menu / cài đặt ẩn riêng lẻ trong các menu và menu phụ khác nhau. Thay vào đó, chỉ cần tạo BSP từ SDK, sau đó là biên dịch hạt nhân sẽ tạo ra một hạt nhân hoạt động trên thiết kế phần cứng cụ thể. (Lưu ý rằng điều này không ngăn chặn người dùng chạy menuconfig truyền thống để thay đổi cài đặt hạt nhân riêng lẻ). Một bản sao của .config cũ được lưu trữ khi cập nhật nó với cấu hình dành riêng cho thiết kế. .Config đã lưu có tên .config.bak.month_day_year_hour_min_sec. (Lưu ý rằng nếu tạo BSP cho MontaVista Linux 5.0, .config không được cập nhật và cũng không được sao chép vào tệp sao lưu.)

Để biên dịch cây nhân trong MontaVista Linux, hãy thay đổi thư mục thành cây nguồn nhân Linux và nhập:

`make ARCH=oldconfig bzImage`

Để biên dịch cây hạt nhân trong Wind River Linux, hãy thay đổi thư mục thành thư mục làm việc Linux (thư mục chứa thư mục con / dist) và nhập:

`make -C dist linux.rebuild`

hoặc, từ gốc của cây nguồn nhân Linux, hãy nhập:

`make ARCH=ppc CROSS_COMPILE=powerpc-wrs-linux-gnu- oldconfig bzImage`

Khi lần đầu tiên chạy, hãy tạo oldconfig hoặc khi lần đầu tiên sử dụng xmake (được mô tả trong phương pháp sau), đặc biệt khi có sự thay đổi trong kiến trúc bảng từ BSP cơ sở, bạn có thể được nhắc về các tùy chọn cấu hình trên dòng lệnh. Nếu điều này xảy ra, chỉ cần nhấn enter để chấp nhận giá trị mặc định cho tất cả các câu hỏi.

Để thực hiện tùy chỉnh này, nhiều tùy chọn hơn đã có sẵn trong hộp thoại Cài đặt gói hỗ trợ bảng trong SDK, như được mô tả trước đó trong tài liệu này.

Lưu ý: Nếu bạn muốn SDK để riêng tệp .config trong hạt nhân của mình, thì hãy để trống tham số target_dir và sao chép cây thư a thớt kết quả từ dự án SDK vào cây hạt nhân đang hoạt động của bạn.

2. Thư mục đích được để trống và công cụ cập nhật cấu hình do Xilinx cung cấp được sử dụng.

Đối với MontaVista Linux 4.0.1 và WindRiver Linux 1.3 / 2.0:

Nếu thư mục đích bị bỏ trống hoặc không trỏ đến cây nguồn nhân Linux hợp lệ, người dùng phải sao chép cây Linux thư a thớt từ vùng dự án sang cây nhân đang hoạt động.

Xilinx cung cấp tập lệnh Tcl có thể chạy sau bản sao cây thư a thớt để cập nhật .config cho phù hợp với thiết kế phần cứng XPS.

Để chạy lệnh, hãy đảm bảo rằng một trình thông dịch Tcl / Tk thích hợp được cài đặt trên hệ thống máy chủ lưu trữ cây nhân Linux đang hoạt động. Nếu hệ thống máy chủ này đã cài đặt các công cụ Xilinx EDK, thì trình thông dịch thích hợp đã được cài đặt và bạn có thể sử dụng trình bao EDK / cygwin để thực hiện bước tiếp theo. Nếu không, hãy đảm bảo máy chủ đã cài đặt trình thông dịch Tcl / Tk 8.0 hoặc mới hơn.

Để cập nhật tệp .config, hãy thay đổi thư mục thành cây nguồn nhân Linux, sau đó nhập:

```
$ tclsh cow.tcl
```

Khi tập lệnh Tcl được chạy một lần, không cần phải chạy lại trừ khi BSP mới được tạo từ Platform Studio. Ngoài ra, người dùng phải biên dịch hạt nhân sau khi tập lệnh được chạy.

Lưu ý rằng bạn có thể sử dụng xmake thay cho lệnh tạo chuẩn để cập nhật tệp .config. Ví dụ: bạn sẽ nhập:

```
./xmake bzImage
```

trong thư mục gốc của cây nguồn nhân Linux. Công cụ xmake trước tiên sẽ gọi tập lệnh Tcl ở trên và sau đó gọi lệnh tạo Linux tiêu chuẩn.

Đối với MontaVista Linux 5.0:

Nếu thư mục đích bị bỏ trống hoặc không trả đến cây nguồn nhân Linux hợp lệ, người dùng phải sao chép cây Linux thưa thoát từ vùng dự án sang cây nhân đang hoạt động.

Cây nguồn nhân Linux mặc định cho MontaVista Linux 5.0 đã bật tất cả các trình điều khiển liên quan đến xilinx. Các trình điều khiển cụ thể sau đó được kích hoạt khi hệ thống thăm dò cây thiết bị, mô tả phần cứng trong hệ thống. Do đó, người dùng không cần các công cụ Xilinx cũng như kích hoạt các trình điều khiển cụ thể trong nhân Linux.

Để xây dựng hạt nhân, đầu tiên hãy nhập:

```
make ml403_defconfig (dành cho bo mạch virtex4
```

hoặc

```
make ml507_defconfig (dành cho bo mạch virtex5
```

Thao tác này sẽ thiết lập hạt nhân cho bộ xử lý trên bo mạch (ppc405 hoặc ppc440).

Sau đó, nếu sử dụng đĩa ram, hãy nhập:

tạo zImage.initrd

nếu không thì

làm cho zImage

3. Thư mục đích được để trống và hạt nhân được cấu hình theo cách thủ công.

Nếu thư mục đích bị bỏ trống, hãy sao chép cây Linux thưa thoát từ vùng dự án sang cây nhân đang hoạt động. Sau đó, người dùng có thể cấu hình hạt nhân theo cách thủ công bằng cách sử dụng "make menuconfig" hoặc tương đương. Xem phần bên dưới về Cấu hình hạt nhân thủ công.

Định cấu hình nhân theo cách thủ công

Phần này cung cấp thông tin chi tiết về cách định cấu hình hạt nhân theo cách thủ công cho IP liên quan đến Xilinx. Nếu bạn đang sử dụng MontaVista Linux 5.0 hoặc nếu thư mục đích được chỉ định trả đến cây nhân Linux hợp lệ, bạn có thể không cần sử dụng các bước này, có nghĩa là .config nhân đã được cập nhật trong quá trình tạo BSP để phù hợp với thiết kế phần cứng.

Tệp cấu hình hạt nhân mặc định đi kèm với bản phân phối Linux 2.6 chứa một tập hợp các tùy chọn hạt nhân chung. Nguồn nhân Linux MontaVista đi kèm với các tệp cấu hình nhân được xác định trước cho các bảng phát triển khác nhau. Một trong những tệp cấu hình khác này có thể là điểm khởi đầu tốt hơn cho nhu cầu của bạn. Các tệp cấu hình khác này có thể được tìm thấy tại:

```
linux / Arch / ppc / configs
```

trong cây nguồn nhân Linux. Để sử dụng một trong các tệp cấu hình này, hãy sao chép tệp cấu hình mong muốn vào:

```
linux / .config
```

Trước tiên, bạn nên lưu một bản sao của tệp cấu hình gốc, .config, trong trường hợp cấu hình gốc cần được khôi phục trong tương lai.

Một trong những phư ơng pháp phổ biến để cấu hình hạt nhân Linux là sử dụng lệnh make menuconfig . Có một số phư ơng pháp khác để định cấu hình hạt nhân Linux, như ng đối với hứ ơng dẫn này, các tùy chọn cấu hình đư ợc mô tả bằng cách sử dụng phư ơng thức make menuconfig .

Vie ệc bao gồm mọi tùy chọn cấu hình hạt nhân cho các phiên bản khác nhau của hạt nhân Linux nằm ngoài phạm vi của hứ ơng dẫn này; tuy nhiên, thông tin về việc sử dụng MontaVista Linux 4.0.1 đư ợc đưa vào làm ví dụ về cách hoàn thành một số nhiệm vụ phát triển ban đầu có thể sẽ gặp phải trong dự án của bạn. Các tùy chọn cấu hình cho các bản phân phối Linux khác và các phiên bản hạt nhân khác có thể khác với các ví dụ.

Khởi động từ thẻ nhớ flash nhỏ gọn (sử dụng System ACE™)

Có nhiều bộ tải khởi động khác nhau có thể đư ợc chọn để khởi động Linux từ thẻ Compact Flash (CF). Tuy nhiên, Xilinx và các bo mạch liên quan thường cung cấp một phư ơng pháp khởi động thay thế đư ợc gọi là Khởi động từ SystemACE. Khởi động từ SystemACE khác với các bộ tải khởi động khác ở chỗ nó cũng tải dòng bit phần cứng vào FPGA.

Khởi động từ System ACE bao gồm các bư ớc sau:

1. Tạo dòng bit phần cứng.
2. Xây dựng nhân Linux.
3. Tạo tệp System ACE.
4. Phân vùng thẻ CF.
5. Sao chép tệp System ACE vào thẻ CF.

Các bo mạch cung cấp khả năng khởi động từ System ACE có một chip, đư ợc gọi là chip System ACE, chip này sẽ đọc thẻ CF đư ợc lắp vào và tìm kiếm tệp có phần mở rộng .ace . Tệp ace này chứa dòng bit phần cứng cùng với có thể là một chương trình thực thi. Sau đó, chip System ACE sẽ tải FPGA bằng dòng bit phần cứng và nếu có một chương trình thực thi, nó sẽ tải chương trình đó vào bộ nhớ và bắt đầu thực thi chương trình.

Hãy nhớ rằng dòng bit phần cứng cũng có thể có một ứng dụng chạy trong bộ nhớ RAM khói trong FPGA. Khi khởi động từ System ACE, bạn nên có một ứng dụng như vậy trong dòng bit phần cứng, chẳng hạn như bộ nạp khởi động hoặc ứng dụng bootloop. Điều này sẽ đảm bảo rằng bộ xử lý không thực hiện các lệnh ngẫu nhiên trong cửa sổ thời gian giữa khi FPGA đư ợc lập trình và khi ứng dụng trong tệp ace đư ợc tải và chạy. Khi nghỉ ngơi, chỉ cần sử dụng chương trình bootloop của bộ xử lý EDK.

Để khởi động từ System ACE, tệp ace cần nằm trong phân vùng đầu tiên trên thẻ CF. Phân vùng này cần đư ợc định dạng để có hệ thống tệp DOS. Phân vùng DOS này có thể cần đư ợc tạo trên thẻ CF và phải đủ lớn để chứa tệp ace. Cũng có thể cần thêm không gian cho các tệp ace bổ sung. 10 megabyte là đủ cho hầu hết các trường hợp. Nếu nhiều tệp ace đang đư ợc quản lý trên thẻ CF, hãy tham khảo bảng dữ liệu SystemACE Compact Flash Solution (DS080), có thể tìm thấy trên trang web Xilinx (<http://www.xilinx.com>).

Lưu ý: Bạn có thể tìm thấy hình ảnh thẻ CF cho bảng Xilinx ML403 và ML507 tại <http://www.xilinx.com/ml403> và <http://www.xilinx.com/ml507> dưới liên kết Bản trình diễn và Thiết kế Tham chiếu.

Cuối cùng, tệp tin ace cần đư ợc sao chép vào phân vùng DOS trên thẻ CF. Tham khảo biểu dữ liệu Xilinx System ACE để biết thông tin về vị trí trong phân vùng DOS mà tệp ace cần cư trú. Để khởi động hệ thống, hãy đảm bảo rằng thẻ CF nằm trong khe đọc thẻ thích hợp trước khi cấp nguồn cho bo mạch. Nếu mọi thứ theo đúng thứ tự, chip System ACE sẽ đảm nhận phần còn lại.

Thiết lập Ethernet

Phần này sẽ mô tả cụ thể các bư ớc cần thiết để ethernet hoạt động trên bo mạch Xilinx. Các bảng phát triển khác có thể đư ợc thiết lập tự động tự, vì vậy phần này có thể vẫn hữu ích cho các bảng khác.

Trong bo mạch Xilinx ML403 và ML507, địa chỉ MAC ethernet duy nhất cho bo mạch được lưu trữ trong EEPROM. EEPROM được truy cập từ FPGA bằng bus I2C và do đó là bộ điều khiển I2C trong FPGA. Các BSP Linux cho các bo mạch Xilinx này có gắng đọc địa chỉ MAC qua I2C trong quá trình khởi tạo. Trong Linux, nếu không thể tìm thấy địa chỉ MAC vì lý do này hay lý do khác, địa chỉ MAC mặc định sẽ được sử dụng. Địa chỉ MAC mặc định này có thể sử dụng được.

Mặc dù vậy, sẽ không thuận tiện lắm nếu mạng của bạn có nhiều bo mạch phát triển này được gắn vào, vì mỗi bo mạch sẽ cần phần mềm hạt nhân khác nhau, mỗi phần mềm có một địa chỉ MAC mặc định khác nhau. Địa chỉ MAC mặc định được xác định trong Arch / ppc / boot / simple / nhung_config.c.

Trong nhiều trường hợp cần sử dụng Ethernet mà không có bus I2C, do đó trình điều khiển I2C sẽ không có mặt. Trong các trường hợp khác, các phương pháp truy xuất địa chỉ MAC khác được mong muốn. Nếu, vì bất kỳ lý do gì, có nhu cầu sử dụng Ethernet mà không cần trình điều khiển I2C, trong file / ppc / boot / simple / nhung_config.c, dòng

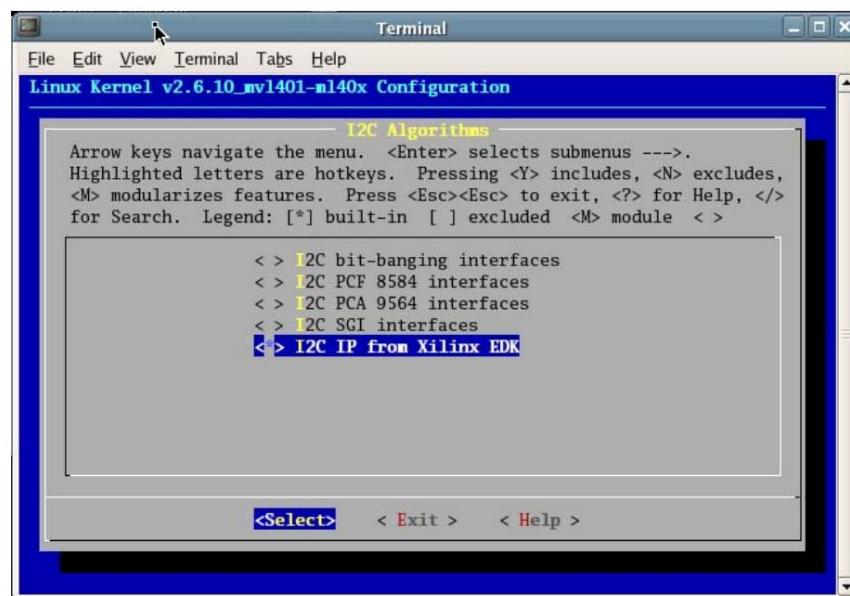
```
#error I2C cần thiết để lấy địa chỉ MAC Ethernet
```

có thể cần được xóa hoặc thay đổi thành cảnh báo hoặc dòng nhận xét để tránh biên dịch lỗi.

Để lấy địa chỉ MAC này, bus I2C được sử dụng để đọc EEPROM chứa địa chỉ MAC. Để sử dụng Ethernet với địa chỉ MAC được lưu trữ, cần thực hiện các bước sau để định cấu hình hạt nhân Linux:

1. Kích hoạt trình điều khiển I2C trong hạt nhân
2. Kích hoạt trình điều khiển Ethernet trong hạt nhân

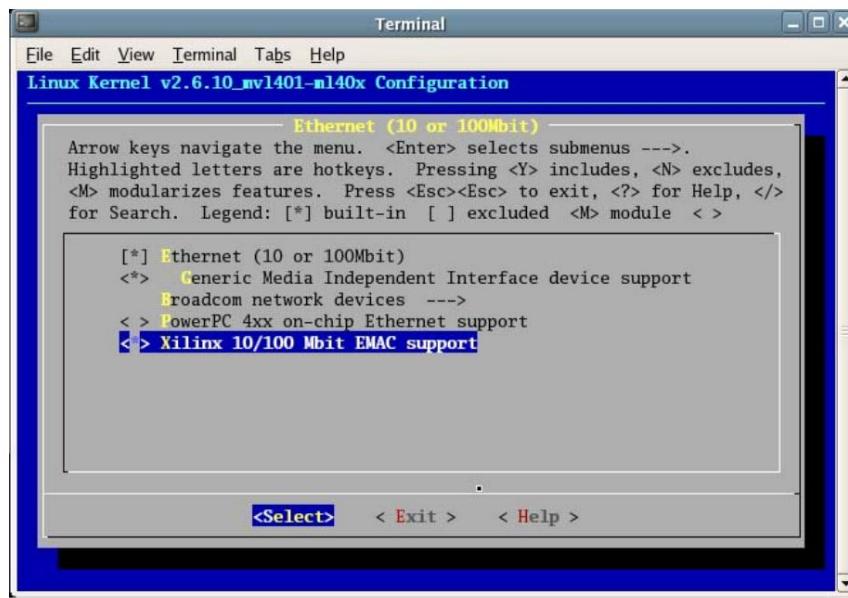
Lưu ý: Một lỗi thường gặp là truy cập phần này của tài liệu và cố gắng bật trình điều khiển I2C trong khi quên rằng lỗi IP I2C không được bao gồm trong thiết kế phần cứng của FPGA. Hãy nhớ đảm bảo rằng lỗi IP I2C được bao gồm trong cấu hình phần cứng.



Hình 4: Kích hoạt I2C trong Kernel

Trình điều khiển I2C có thể được kích hoạt trong hạt nhân bằng cách chọn Trình điều khiển thiết bị> Hỗ trợ I2C> Thuật toán I2C, sau đó chọn IP I2C từ Xilinx EDK trong menu tạo menuconfig .

Nếu hệ thống tệp gốc sẽ nằm trên NFS, tốt nhất là nên có trình điều khiển I2C trong nhân. Nếu không, trình điều khiển I2C có thể được xây dựng dưới dạng mô-đun.



Hình 5: Kích hoạt 10/100 Ethernet trong Kernel

Trình điều khiển ethernet có thể được kích hoạt trong hạt nhân từ make menuconfig bằng cách chọn Trình điều khiển thiết bị> Hỗ trợ mạng> Ethernet (10 hoặc 100Mbit) sau đó hỗ trợ Xilinx 10/100 Mbit EMAC hoặc Ethernet (1000 Mbit) sau đó hỗ trợ Xilinx 10/100/1000 Mbit TEMAC. Nếu hệ thống tệp gốc sẽ nằm trên NFS, thì trình điều khiển này phải được tích hợp sẵn trong hạt nhân. Nếu không, nó có thể được xây dựng như một mô-đun. Theo mặc định, BSP cho bo mạch Xilinx trong Linux có 10/100 Ethernet được kích hoạt trong nhân (không phải dưới dạng mô-đun).

Thiết lập hệ thống tệp tin gốc Linux

Vị trí của hệ thống tệp gốc có thể nằm ở một số nơi khác nhau bắt kể hệ thống khởi động như thế nào. Hệ thống tệp gốc có thể nằm trên chia sẻ mạng NFS, trên thẻ CF hoặc thậm chí được tải vào RAM, trong số các lựa chọn khác.

Có nhiều phương pháp khác nhau để tạo nội dung hệ thống tệp gốc bao gồm sử dụng các công cụ của nhà cung cấp. Mô tả cách tạo hệ thống tệp gốc hoặc thậm chí mô tả những tệp thực thi nào cần thiết trên hệ thống tệp gốc nằm ngoài phạm vi của hướng dẫn này. Một số tài nguyên tốt để nhận trợ giúp trong lĩnh vực này là:

Xây dựng Hệ thống Linux nhúng (<http://www.oreilly.com/catalog/belinuxsys/index.html>)

Linux Từ Dự án Scratch (<http://www.linuxfromscratch.org>)

Tiêu chuẩn phân cấp hệ thống tệp (<http://www.pathname.com/fhs>)

Lưu ý: Nếu Xilinx ML403 hoặc ML507 đang được sử dụng và bạn đang sử dụng MontaVista Linux, bạn có thể bắt đầu với hệ thống tệp gốc có trên thẻ CompactFlash đi kèm với bảng. Lưu ý rằng hệ thống tệp gốc trên thẻ CF được xây dựng cho MontaVista Linux và không được đảm bảo hoạt động với Wind River Linux.

Trong một hệ thống nhúng, thường có yêu cầu tách hệ thống tệp gốc tĩnh (được sử dụng cho các quy trình khởi động) khỏi một khu vực đang lưu trữ dữ liệu tạm thời, dữ liệu sử dụng trờ ơng hoặc dữ liệu ngắn dùng cuối. Sự tách biệt này có thể ngăn dữ liệu động vô tình ghi đè lên các tệp hệ thống hoặc chỉ đơn giản là làm đầy hệ thống tệp gốc ngăn hệ thống khởi động.

Linux hỗ trợ một loạt các hệ thống tệp như Ext2, Ext3, ReiserFS, JFS, XFS và các hệ thống khác. Hệ thống tệp gốc của nhiều hệ thống nhúng sẽ hầu như vẫn ở trạng thái tĩnh. Trong trường hợp này, một hệ thống tệp tốt để sử dụng là Ext2, được sử dụng rộng rãi và đã được thử nghiệm tốt.

Nếu hệ thống nhúng sẽ ghi nhiều tệp, đặc biệt là các tệp lớn, thì hệ thống tệp XFS là một lựa chọn tốt. Ext3 cũng thường được sử dụng. Lưu ý rằng việc sử dụng Ext3 hoặc XFS trên thẻ CF không đặc biệt hữu ích, vì thẻ CF tương đối chậm và có dung lượng tương đối thấp. Nếu tất cả những gì đang được viết hoặc sửa đổi là dữ liệu cấu hình được ghi thông qua các phương thức được xác định rõ ràng, thì chỉ cần sử dụng một phân vùng gốc là đủ. Việc sử dụng Ext3, XFS hoặc một số hệ thống tệp khác có lợi hơn khi có một phương tiện lưu trữ khá nhanh hoặc dung lượng lớn trong hệ thống nhúng, chẳng hạn như đĩa cứng.

Sử dụng hệ thống tệp gốc trên thẻ nhớ flash nhỏ gọn

Phần này giải thích cách sử dụng hệ thống tệp gốc trên thẻ CF, được truy cập thông qua System ACE. Nếu hệ thống tệp gốc nằm trên thẻ CF, thì cần thực hiện các bước sau:

1. Phân vùng thẻ CF.
2. Tạo hệ thống tệp trên thẻ CF.
3. Sao chép các tệp và thư mục của hệ thống tệp gốc.
4. Định cấu hình hạt nhân để biên dịch trong trình điều khiển thẻ CF.
5. Định cấu hình các thông số khởi động hạt nhân để sử dụng hệ thống tệp gốc trên thẻ CF.

Nếu thẻ CF cần được phân vùng lại để giữ hệ thống tệp gốc, thì một cách dễ dàng để phân vùng nó là gắn một đầu đọc thẻ CF vào máy trạm Linux và sử dụng các công cụ Linux để phân vùng ổ đĩa.

Linux fdisk thường như hoạt động tốt. Trên một hệ thống ở đây, thẻ CF có thể được truy cập thông qua /dev / sda, mặc dù điều này có thể khác trên hệ thống của bạn.

Lưu ý: Nếu thẻ CF cũng sẽ được sử dụng để khởi động từ System ACE, hãy nhớ để phân vùng đầu tiên làm phân vùng DOS.

Về mặt kỹ thuật, phân vùng hoán đổi là không cần thiết. Tuy nhiên, có một phân vùng hoán đổi sẽ làm tăng dung lượng bộ nhớảo có sẵn. Kích thước phân vùng hoán đổi được khuyến nghị thường gấp đôi kích thước của RAM. Tuy nhiên, không gian hoán đổi nhiều hơn hoặc ít hơn có thể được chỉ định tùy thuộc vào nhu cầu của hệ thống. Khi tạo phân vùng hoán đổi, hãy nhớ đặt kiểu của phân vùng thành Linux Swap (kiểu 82).

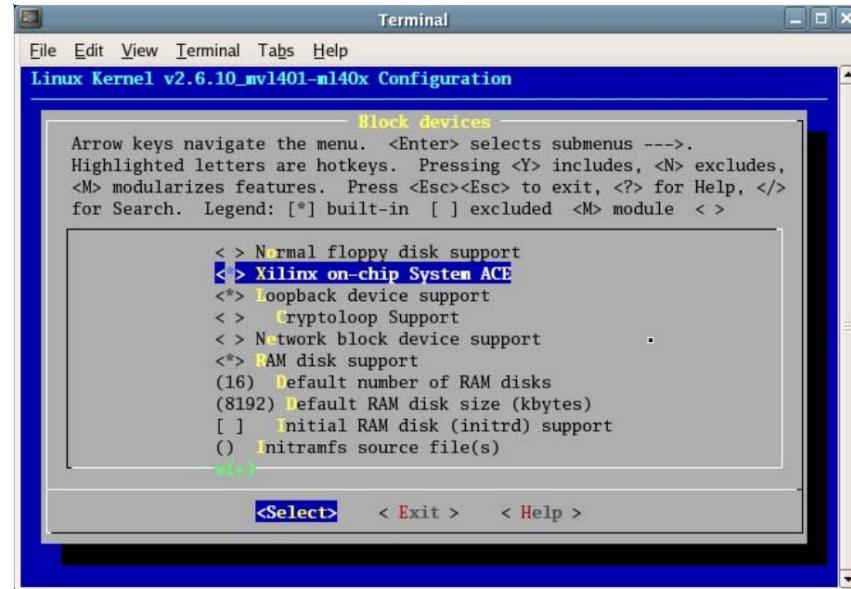
Các phân vùng cho hệ thống tệp gốc phải đủ lớn để chứa các tệp đang được đặt trong hệ thống tệp gốc. Một tiện ích hữu ích để xác định kích thước hệ thống tệp gốc, nếu nó nằm trong khu vực dàn dựng, là du. Khi tạo phân vùng gốc, hãy nhớ đặt loại phân vùng là Linux (loại 83).

Khi các phân vùng đã được tạo, hệ thống tệp trên các phân vùng Linux sẽ cần được tạo. Một số công cụ Linux như parted có khả năng tạo hệ thống tệp trống tại thời điểm mỗi phân vùng được tạo. Nếu không, sẽ cần một công cụ như mkfs.

Thường thì nội dung của hệ thống tệp gốc được đặt trong một vùng dàn cho phép sao chép toàn bộ thư mục cùng một lúc. Khi sao chép một cây thư mục như vậy, bạn nên đặt các thuộc tính tệp và thư mục một cách chính xác trước khi thực hiện sao chép. Đảm bảo khi thực hiện sao chép sử dụng lệnh sẽ bao toàn các thuộc tính như cp -a hoặc tar.

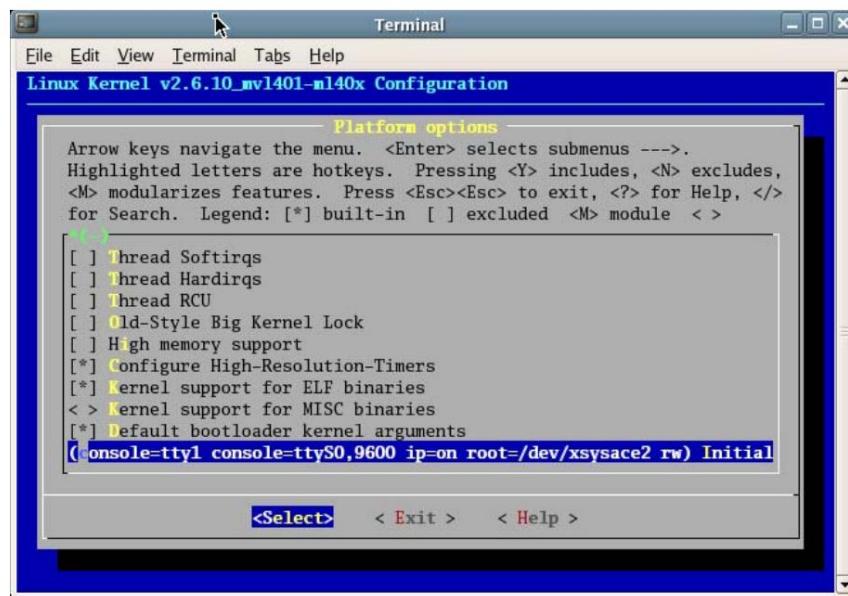
Sau khi các tệp của hệ thống tệp gốc đã được sao chép qua thẻ CF, hạt nhân sẽ cần được định cấu hình để sử dụng hệ thống tệp gốc đó. Để hạt nhân có thể đọc thẻ CF, trình điều khiển hạt nhân System ACE phải được bật.

Lưu ý: Một lỗi phổ biến tại thời điểm này có thể là đã quên bao gồm lỗi IP System ACE trong dự án phần cứng của bạn.



Hình 6: Lựa chọn hỗ trợ hạt nhân SystemACE

Trình điều khiển System ACE có thể được bật bằng cách chọn Trình điều khiển thiết bị> Chặn thiết bị, sau đó chọn SystemACE trên chip Xilinx trong menu từ make menuconfig. Đàm bảo hỗ trợ cho thiết bị này được bao gồm trong hạt nhân, không phải dưới dạng mô-đun. Xem [Hình 4, trang 12](#). Theo mặc định trong BSP cho bo mạch Xilinx trong Linux, tùy chọn này được bật.



Hình 7: Tùy chọn chuỗi lệnh nhân ban đầu

Tiếp theo, chỉnh sửa tùy chọn chuỗi lệnh kernel ban đầu. Có thể tìm thấy tùy chọn này bằng cách chọn Tùy chọn nền tảng trong menu chính của make menuconfig, để cho hạt nhân biết vị trí của hệ thống tệp gốc. Các đối số hạt nhân của bộ nạp khởi động mặc định cũng phải được chọn để tùy chọn này xuất hiện. Mục root = của chuỗi lệnh nhân ban đầu này, trong số các tùy chọn khác, phải chứa

```
root = / dev / xsysace2 rw
```

trong đó N là số phân vùng của hệ thống tệp gốc trên thẻ CF.

Theo mặc định trong BSP cho bo mạch Xilinx, tùy chọn này được thiết lập để sử dụng hệ thống tệp gốc trên thẻ CF.

Sử dụng Hệ thống tệp gốc trong đĩa RAM

Khi sử dụng đĩa RAM cho hệ thống tệp gốc, ảnh đĩa RAM được liên kết với ảnh hạt nhân. Quá trình sử dụng hệ thống tệp gốc gần giống với quá trình sử dụng đĩa RAM ban đầu (initrd). Sự khác biệt duy nhất là trong trình tự khởi động thay vì thực hiện root pivot tới hệ thống tệp gốc trên một phư ơng tiện khác, hạt nhân thực hiện root pivot trở lại hệ thống tệp đĩa RAM. Lưu ý rằng tính đến thời điểm viết bài này, Wind River Linux không chính thức hỗ trợ rootfs ramdisk.

Các bước cần thiết để sử dụng hệ thống tệp gốc đĩa RAM là:

1. Tạo tệp đĩa RAM.
2. Định cấu hình hạt nhân để có trình điều khiển đĩa RAM.
3. Cấu hình chuỗi lệnh ban đầu của hạt nhân để sử dụng gốc từ đĩa RAM.
4. Xây dựng hạt nhân để nó bao gồm đĩa ram.

MontaVista Linux Professional Edition 4.0 bao gồm một ảnh đĩa RAM dựng sẵn. Hình ảnh tạo sẵn này có dung lượng khoảng 6MB không nén và rất có thể cung cấp nội dung đủ cho quá trình phát triển ban đầu của bạn. Nếu có thể sử dụng ảnh đĩa RAM dựng sẵn này, thì có thể bỏ qua bước đầu tiên của việc xây dựng tệp đĩa RAM. Hướng dẫn về cách sử dụng hình ảnh dựng sẵn này được mô tả bên dưới. Nếu vì lý do nào đó, ảnh đĩa RAM này không hoạt động cho dự án của bạn, thì ảnh đĩa RAM khác sẽ phải được tạo. Bản phân phối Wind River Linux không chứa ảnh đĩa RAM dựng sẵn.

Đĩa RAM khởi động dữ trữ dạng tệp hình ảnh chứa hệ thống tệp sẽ được tải vào bộ nhớ tại thời điểm khởi động. Hình ảnh đĩa RAM này phải chứa một hệ thống tệp ext2 tiêu chuẩn.

Cách dễ nhất để tạo tệp hình ảnh là sử dụng một máy trạm Linux có sẵn và bắt đầu bằng các lệnh sau:

```
dd if = / dev / zero of = initrd.img bs = 1k count = kbytes size
mke2fs -F -v -m0 initrd.img
```

Các lệnh này tạo ra một hình ảnh đĩa RAM trống. Bước tiếp theo là gắn tệp hình ảnh đó và sao chép tệp hệ thống tệp gốc vào tệp hình ảnh .. Để gắn tệp hình ảnh vào / mnt / tmp, có thể sử dụng lệnh sau:

```
mount -o vòng lặp initrd.img / mnt / tmp
```

Bây giờ, sao chép các tệp và thư mục, giữ nguyên các thuộc tính của chúng, vào hệ thống tệp gốc của đĩa RAM, sau đó ngắt kết nối nó. Hãy nhớ sử dụng cp -a hoặc tar khi thực hiện sao chép để các thuộc tính tệp và thư mục có thể được giữ nguyên. Chạy trình umount được sử dụng để ngắt kết nối hệ thống tệp. Nếu hình ảnh của bạn được gắn trên / mnt / tmp như trong ví dụ trên, lệnh

```
umount / mnt / tmp
```

sẽ ngắt kết nối hệ thống tệp trong tệp hình ảnh.

Cuối cùng nén tệp hình ảnh bằng lệnh sau:

```
gzip -9 initrd.img ramdisk.image.gz
```

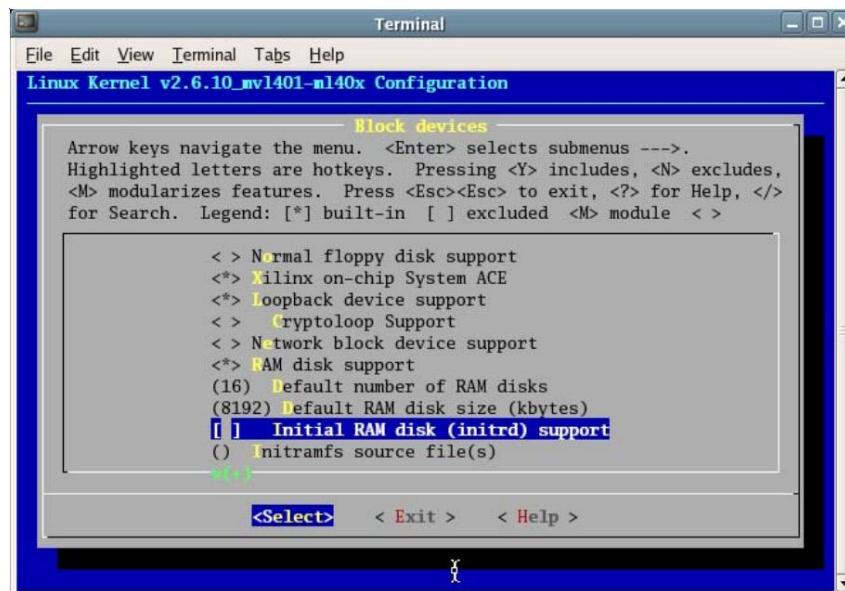
Và sau đó sao chép ramdisk.image.gz vào thư mục sau trong cây nguồn nhân Linux đang hoạt động của bạn:

```
Arch / ppc / boot / hình ảnh
```

Nếu bạn đang sử dụng ảnh đĩa RAM dựng sẵn từ MontaVista, chỉ cần sao chép ảnh tạo sẵn vào cây nguồn nhân Linux. Đây là một ví dụ, giả sử bạn đã cài đặt các công cụ ppc405 từ MontaVista ở vị trí mặc định. Từ trên cùng của cây nhân làm việc linux, hãy sử dụng lệnh sau:

```
cp /opt/montavista/pro/devkit/ppc/405/images/ramdisk.gz Arch / ppc / boot /
images / ramdisk.image.gz
```

Bây giờ tệp hình ảnh đĩa RAM đã được tạo và ở đúng vị trí, hạt nhân phải được cấu hình để sử dụng tệp đó. Trình điều khiển đĩa RAM phải được kích hoạt và chuỗi lệnh ban đầu của hạt nhân phải được đặt để nó sử dụng đĩa RAM làm hệ thống tệp / gốc .



Hình 8: Tùy chọn đĩa RAM

Các tùy chọn để bật hỗ trợ đĩa RAM trong nhân có thể được tìm thấy trong make menuconfig bằng cách chọn Trình điều khiển thiết bị-> Chặn thiết bị trong menu cấp cao nhất như trong [Hình 6, trang 15](#).

Để thiết lập hạt nhân cho gốc đĩa RAM, hỗ trợ đĩa RAM cũng như hỗ trợ đĩa RAM ban đầu (initrd) đều phải được bật trong hạt nhân (không phải dưới dạng mô-đun). Kích thư ớc đĩa RAM mặc định nên được đặt thành giá trị lớn hơn một chút so với kích thư ớc không nén của ảnh đĩa RAM thực tế để có chỗ cho các tệp tạm thời được sử dụng trong quá trình khởi động. Việc làm cho kích thư ớc đĩa RAM trong kernel lớn hơn 8K so với kích thư ớc hình ảnh không nén đã được quan sát là hoạt động tốt.

Để cấu hình hạt nhân để sử dụng đĩa RAM làm hệ thống tệp gốc, chuỗi lệnh hạt nhân ban đầu phải được sửa đổi. Tùy chọn này có thể được tìm thấy trong make menuconfig bằng cách chọn Tùy chọn nền tảng trong menu chính như thể hiện trong [Hình 5, trang 13](#). Lưu ý rằng các đối số hạt nhân của bộ nạp khởi động mặc định cũng phải được chọn để tùy chọn này xuất hiện.

Để tiếp tục sử dụng đĩa RAM ban đầu làm hệ thống tệp gốc của bạn, bạn sẽ phải đặt mục root = của chuỗi lệnh hạt nhân ban đầu để có

```
root = / dev / ram rw
```

Lưu ý không có tùy chọn root = nào khác trên chuỗi lệnh này. Wind River Linux có thể sử dụng

```
root = / dev / ram0 rw
```

thay vì.

Theo mặc định trong bo mạch BSPs Xilinx trong Linux, tùy chọn này không được thiết lập để sử dụng hệ thống tệp gốc đĩa RAM. Phần của dòng đọc

```
root = / dev / xsysace2
```

cần được thay thế bằng văn bản chính xác, như được mô tả ở trên.

Cáu hình hệ thống tệp tin gốc NFS

Hầu hết các hệ thống nhúng sẽ không sử dụng NFS để lưu trữ hệ thống tệp gốc trong sản phẩm cuối cùng.

Tuy nhiên, việc sử dụng NFS cho hệ thống tệp gốc trong quá trình phát triển có thể hữu ích. Với hệ thống tệp gốc NFS, bạn không cần phải lo lắng về các yêu cầu về kích thư ớc. Điều này đặc biệt hữu ích khi làm việc với các tệp gỡ lỗi tạm thời. Việc cập nhật hệ thống tệp gốc NFS cũng dễ dàng hơn nhiều, trái ngược với thẻ CF hoặc hình ảnh đĩa RAM, khi trong quá trình phát triển, các chương trình mới được phát hiện là cần thiết.

Để sử dụng chia sẻ NFS cho hệ thống tệp gốc, có ba bước:

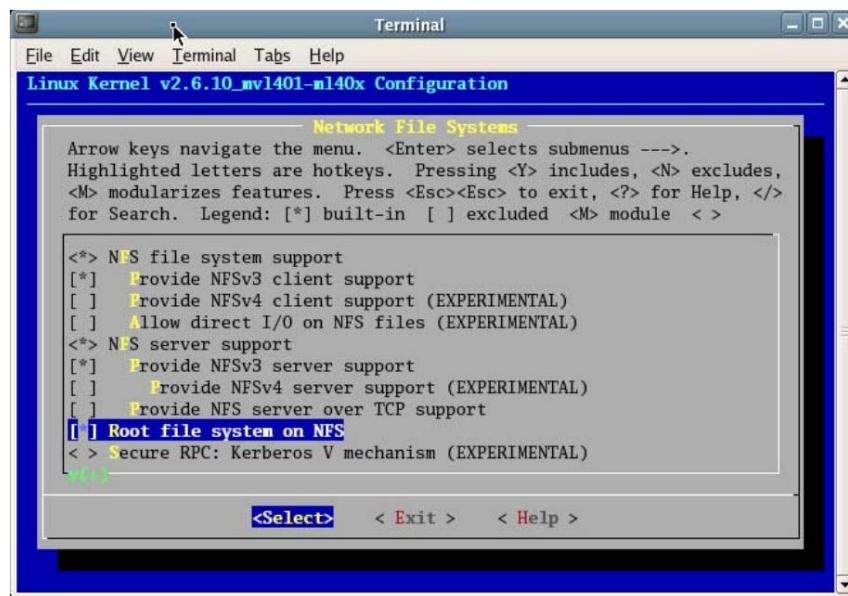
1. Tạo hệ thống tệp gốc trên chia sẻ NFS
2. Thiết lập Ethernet (xem ở trên)
3. Định cấu hình các thông số khởi động hạt nhân để sử dụng gốc NFS

Để tạo hệ thống tệp gốc trên chia sẻ NFS, hãy đặt tệp hệ thống tệp gốc đích trong cây thư mục sẽ được xuất qua NFS. Hãy nhớ rằng các tệp cho phần chia sẻ này không nhất thiết phải giống với các tệp chạy trên hệ thống máy chủ. Trên thực tế, hầu hết thời gian, hệ thống lưu trữ NFS sẽ không có cùng kiến trúc bộ xử lý với hệ thống đích. Có nhiều tài nguyên có sẵn mô tả cách chia sẻ một thư mục thông qua NFS. NFS sẽ không được đề cập đầy đủ ở đây. Chia sẻ NFS cơ bản có thể được tạo trên hệ thống máy chủ Linux bằng cách thêm

```
đường dẫn thư mục * (rw)
```

sang tệp / etc / export và sau đó khởi động lại daemon NFS. Hãy nhớ rằng bạn phải đăng nhập bằng quyền root để chỉnh sửa tệp đó và khởi động lại daemon NFS.

Để thiết lập ethernet trong hạt nhân, hãy xem phần "[Thiết lập Ethernet](#)", trang 11.



Hình 9: Hệ thống tệp gốc trên NFS

Cuối cùng, hạt nhân phải được yêu cầu sử dụng NFS cho hệ thống tệp gốc cùng với vị trí trên mạng của NFS để sử dụng. Trong Hệ thống tệp> Hệ thống tệp mạng, bật Hệ thống tệp gốc trên NFS. Xem [Hình 7](#), [Trang 16](#). Chuỗi lệnh hạt nhân ban đầu một lần nữa được sửa đổi để hoàn tất cài đặt để sử dụng hệ thống tệp gốc qua NFS. Tùy chọn này có thể được tìm thấy trong make menuconfig bằng cách chọn Tùy chọn nền tảng trong menu chính như thể hiện trong [Hình 5](#), [trang 13](#).

Nếu sử dụng máy chủ NFS và chia sẻ, mục root = của chuỗi lệnh hạt nhân ban đầu sẽ được thay thế bằng:

```
ip = on nfsroot = nfs share rw
Đây là một ví dụ
```

```
ip = on nfsroot = 192.168.1.10: / export / virtex5_root rw
```

Nếu sử dụng DHCP để truy xuất máy chủ NFS và tên chia sẻ, mục root = phải chứa:

```
ip = trên root = / dev / nfs
```

Theo mặc định trong BSP cho bo mạch Xilinx trong Linux, tùy chọn này không được thiết lập để sử dụng hệ thống tệp gốc qua NFS. Phần của dòng đọc

```
root = / dev / xsysace2
```

cần được thay thế bằng văn bản chính xác như mô tả ở trên.

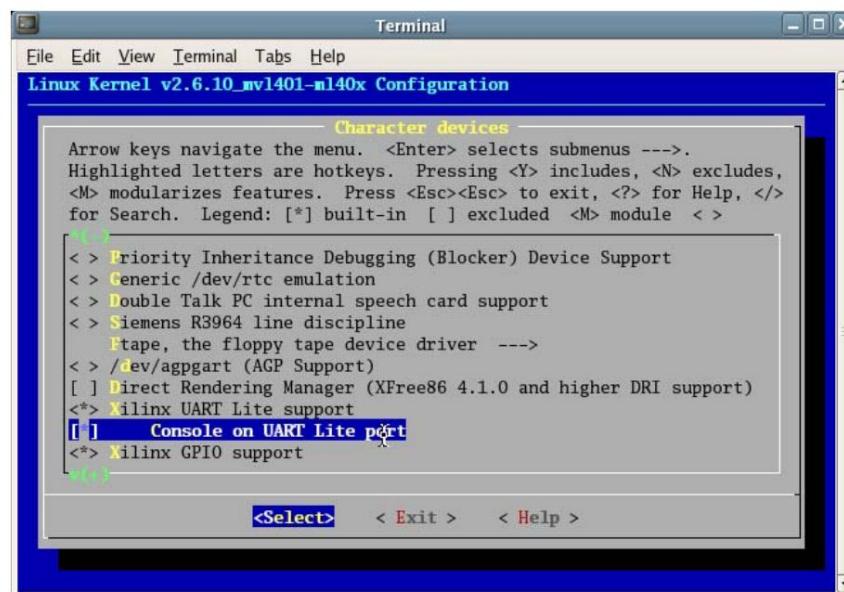
Định cấu hình Bảng điều khiển chính nối tiếp / UART

Linux cung cấp cho nhiều bảng điều khiển khác nhau được kết nối. Một số có thể qua cổng nối tiếp trong khi số khác thông qua bàn phím và màn hình. Bàn điều khiển chính là bàn điều khiển mà trên đó hạt nhân hiển thị các thông báo. Các bảng điều khiển khác chỉ cung cấp các phiên đăng nhập bổ sung.

Để có bảng điều khiển chính sử dụng cổng nối tiếp, có ba bước:

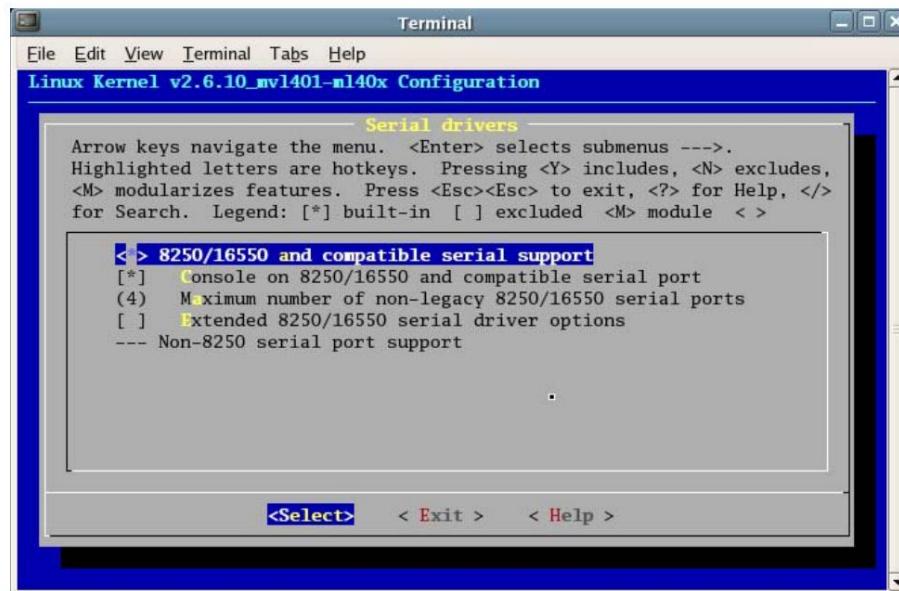
1. Định cấu hình hạt nhân để bao gồm trình điều khiển UART.
2. Định cấu hình trình điều khiển UART để bao gồm hỗ trợ cho bảng điều khiển chính.
3. Định cấu hình các tham số khởi động hạt nhân để sử dụng UART làm bảng điều khiển chính.

Để thiết lập hạt nhân để nó sử dụng cổng nối tiếp cho bảng điều khiển chính, trình điều khiển UART chính xác cần được kích hoạt theo cấu hình phần cứng. Trình điều khiển UART Lite dành cho lõi IP UART Lite có thể có trong cấu hình phần cứng của bạn. Nếu không, trình điều khiển Linux UART tiêu chuẩn sẽ được sử dụng.



Hình 10: Cấu hình Kernel cho UART Lite

Bạn có thể tìm thấy tùy chọn bật trình điều khiển Xilinx UART Lite trong trình đơn Trình điều khiển thiết bị> Thiết bị ký tự trên menu chính trong tạo menuconfig. Nếu sử dụng trình điều khiển UART Lite cho bảng điều khiển chính qua cổng nối tiếp, trình điều khiển này phải được kích hoạt trong nhân, thay vì là một mô-đun. Khi Xilinx UART Lite đã được bật đúng cách, Bảng điều khiển trên cổng UART Lite cũng phải được chọn.



Hình 11: Cấu hình hạt nhân cho UART 16550

Nếu thay vì lõi IP UART Lite, lõi IP UART16550 được sử dụng, thì nên sử dụng trình điều khiển nối tiếp Linux 16550 tiêu chuẩn. Bạn có thể tìm thấy tùy chọn này trong menu Trình điều khiển thiết bị> Thiết bị ký tự> Trình điều khiển nối tiếp. Cũng giống như với UART Lite, trình điều khiển này nên được bao gồm trong hạt nhân thay vì là một mô-đun và Bảng điều khiển trên 8250/16550 và tùy chọn cổng nối tiếp tương thích cũng nên được bật.

Khi trình điều khiển UART đã được kích hoạt và nó đã được cấu hình để cho phép một bảng điều khiển nối tiếp, tùy chọn chuỗi lệnh hạt nhân ban đầu cần được sửa đổi. Xem [Hình 5, trang 13](#).

Đảm bảo rằng tùy chọn chuỗi lệnh hạt nhân ban đầu chứa văn bản sau:

cho UART Lite (WR GPP LE 1.3 và MVL 4.0.1):

```
console = số ttlport
```

cho UART Lite (WR GPP LE 2.0):

```
console = ttyULport number
```

đối với UART 16550:

```
console = số cổng ttyS, tốc độ truyền
```

ở đâu:

- số cổng đè cập đến cổng nối tiếp nào được sử dụng.
- tốc độ truyền là tốc độ của cổng nối tiếp.

Ví dụ: đối với bảng điều khiển nối tiếp 16550 trên cổng nối tiếp đầu tiên có tốc độ truyền là 9600, chuỗi trông giống như

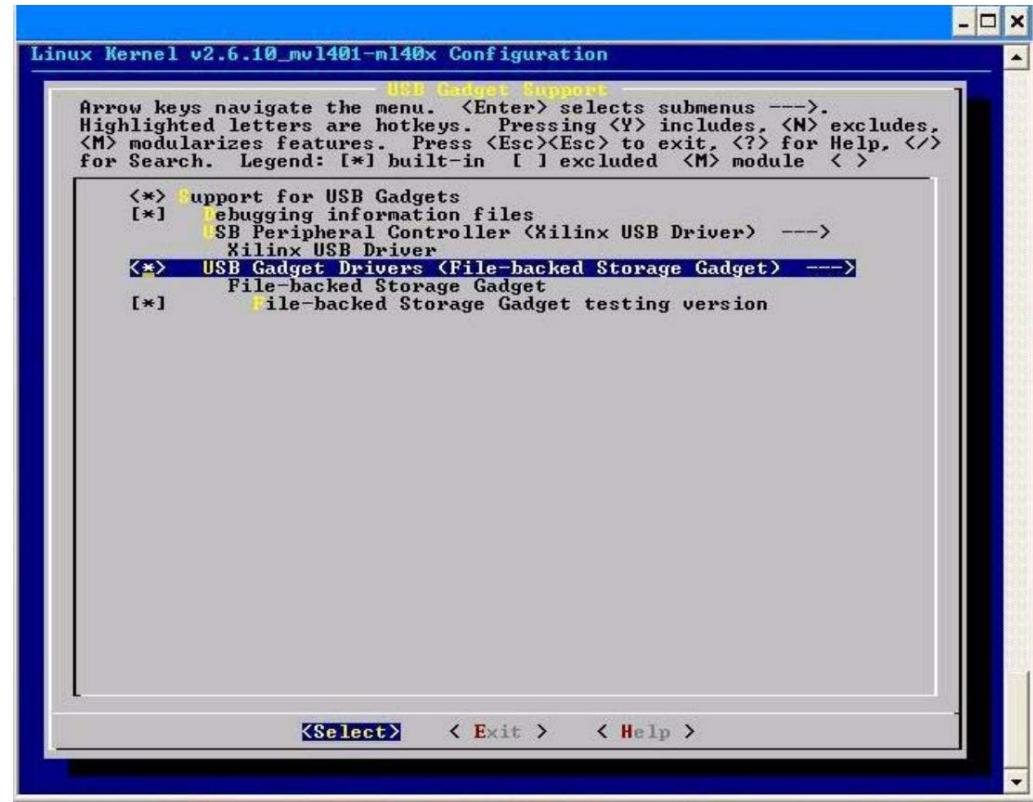
```
console = ttyS0,9600
```

Khi sử dụng một chuỗi, như trong ví dụ trên, ứng dụng đầu cuối máy khách phải kết nối bằng 9600 bps, 8 bit dữ liệu, không có chẵn lẻ và 1 bit dừng. Lưu ý rằng tốc độ truyền của UART Lite được cố định tại thời gian xây dựng phần cứng, vì vậy không cần chỉ định tốc độ này trong phần mềm.

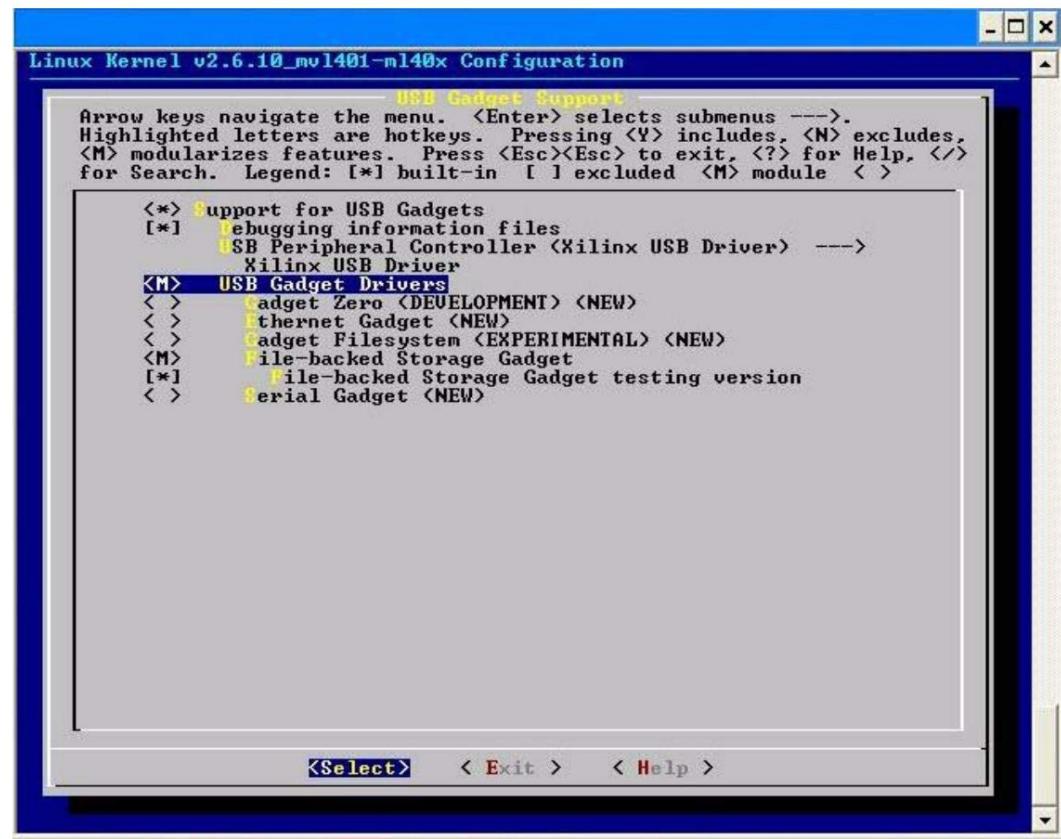
Đôi khi có thể có sự không chắc chắn về cách thiết lập bảng điều khiển chính và cách thiết lập bảng điều khiển tiếp theo. Với bảng điều khiển chính trên cổng nối tiếp, tắt cả các thông báo khởi động sẽ được gửi đến cổng nối tiếp. Đây thường là kết quả mong muốn cho phát triển nhúng. Bằng cách này, bất kỳ thông báo lỗi nào được trình bày ở đó có thể được nhìn thấy trong trình tự khởi động. Không nên nhằm lẩn phuơng pháp được mô tả ở đây để thiết lập bảng điều khiển nối tiếp chính với các phuơng pháp để định kèm bảng điều khiển bổ sung thông qua tệp / etc / inittab.

Định cấu hình Trình điều khiển Bộ điều khiển Ngoại vi USB và Trình điều khiển Tiện ích

Trình điều khiển Bộ điều khiển Ngoại vi USB Xilinx chỉ hỗ trợ Trình điều khiển Tiện ích Lưu trữ được hỗ trợ bởi Tệp. Trong BSP được tạo, Tiện ích lưu trữ được sao lưu tệp được đặt theo mặc định như thể hiện trong [Hình 12, trang 22](#). Tiện ích Lưu trữ được sao lưu tệp nên được chọn làm mô-đun trong Trình điều khiển thiết bị> Hỗ trợ USB> Hỗ trợ tiện ích USB trong menu từ make menuconfig như [hình 13, trang 23](#).



Hình 12: Hỗ trợ trình điều khiển tiện ích USB mặc định



Hình 13: Định cấu hình Hỗ trợ Trình điều khiển Tiện ích USB làm Mô-đun

Cấu hình PCI và các thiết bị ngoại vi liên quan

Phần này mô tả các tùy chọn cấu hình cho bảng ML410. Nếu bạn đang sử dụng tính năng cập nhật cấu hình tự động trong Platform Studio (cow.tcl hoặc xmake), bạn có thể bỏ qua phần này.

Sau đây là danh sách nhanh các tùy chọn "tạo menuconfig" cho từng thiết bị PCI trên ML410 sử dụng MVL 4.0.1.

- Bật Tùy chọn xe buýt> Hỗ trợ PCI
 - Kích hoạt cơ sở dữ liệu tên thiết bị PCI
- Bật Trình điều khiển thiết bị> Hỗ trợ ATA / ATAPI / MFM / RLL> ATA / ATAPI / MFM / RLL
 - Bật đĩa IDE / MFM / RLL nâng cao / cdrom / băng / đĩa mềm / hỗ trợ
 - Bật hỗ trợ Bao gồm IDE / ATA-2 DISK
 - Kích hoạt hỗ trợ Bao gồm IDE / ATAPI CDROM
 - Bật hỗ trợ Bao gồm IDE / ATAPI TAPE
 - Kích hoạt hỗ trợ chipset PCI IDE
 - Kích hoạt Chia sẻ PCI IDE ngắt nguồn
 - Kích hoạt hỗ trợ đầu tiên cho các chipset ngoài bo mạch khởi động
 - Bật hỗ trợ chipset IDE chung PCI
 - Bật hỗ trợ DMA bus-master PCI chung
 - Kích hoạt hỗ trợ chipset ALI M15x3

- Bật PROMIS PDC202 {46 | 62 | 65 |}
- Bật Trình điều khiển thiết bị> Hỗ trợ thiết bị SCSI> Hỗ trợ thiết bị SCSI
 - Bật hỗ trợ chung SCSI
- Nếu sử dụng thẻ Ethernet 3COM:
 - Kích hoạt Trình điều khiển thiết bị> Hỗ trợ mạng> Ethernet (10 hoặc 100Mbit)> 3COM thẻ
 - Kích hoạt hỗ trợ 3c590 / 3c900 series (592/595/597) Vortex / Bommerang
 - Kích hoạt Trình điều khiển thiết bị> Hỗ trợ USB> Hỗ trợ USB phía máy chủ
 - Kích hoạt hệ thống tập tin thiết bị USB

Lưu ý rằng các tùy chọn trên có thể thay đổi tùy theo nhu cầu của bạn.

Thiết bị Linux

Tài liệu tham khảo

Để giảm lựợng thời gian dành cho việc tìm kiếm tệp và cài đặt, [Bảng 1](#) cung cấp mối quan hệ giữa các môđun trình điều khiển, lõi IP được cung cấp bởi Xilinx, vị trí của tệp nguồn trình điều khiển và các mục cấu hình hạt nhân.

Bảng 1: Trình điều khiển và lõi IP được hỗ trợ trong Linux

Trình điều khiển Xilinx	Lõi IP Xilinx	Vị trí tài xế trong LSP	Mục cấu hình hạt nhân Linux
n / a	opb_uart16550 plb_uart16550 xps_uart16550	linux / drivers / serial / 8250.c	Trình điều khiển thiết bị / Thiết bị ký tự / Trình điều khiển nối tiếp / 8250/16550 và hỗ trợ nối tiếp tương thích
uartlite	opb_uartlite xps_uartlite	linux / drivers / char / xilinx_uartlite	Trình điều khiển thiết bị / Thiết bị nhân vật / Hỗ trợ Xilinx UART Lite
emac	opb_ethernet plb_ethernet	linux / drivers / net / xilinx_emac	Trình điều khiển thiết bị / Hỗ trợ mạng / Hỗ trợ thiết bị mạng / Ethernet (10 hoặc 100Mbit) / Xilinx Hỗ trợ EMAC 10/100 Mbit
temac	plb_temac	linux / drivers / net / xilinx_temac	Trình điều khiển thiết bị / Hỗ trợ mạng / Hỗ trợ thiết bị mạng / Ethernet (1000 Mbit) / Xilinx Hỗ trợ 10/100/1000 Mbit TEMAC
lltemac	xps_ll_temac	linux / drivers / net / xilinx_temac	Trình điều khiển thiết bị / Hỗ trợ mạng / Hỗ trợ thiết bị mạng / Ethernet (1000 Mbit) / Xilinx Hỗ trợ 10/100/1000 Mbit TEMAC
llfifo	xps_ll_fifo	linux / drivers / xilinx_common	n / a
lldma	mpmc_w / sdma ppc440_dma	linux / drivers / xilinx_common	n / a
n / a	opb_intc dcr_intc xps_intc	linux / Arch / ppc / syslib / xilinx_pic.c	n / a
iic	opb_iic xps_iic	linux / drivers / i2c / algos / xilinx_iic	Trình điều khiển thiết bị / hỗ trợ I2C / I2C Thuật toán / I2C IP từ Xilinx EDK
n / a	plb_tft_cntlr_ref	linux / driver / video / xilinxfb.c	n / a
touchscreen_ref opb_tsrd_ref		linux / drivers / char / xilinx_ts	n / a
ps2_ref	opb_ps2_dual_ref opb_ps2_ref	linux / driver / input / serio / xilinx_ps2	Trình điều khiển thiết bị / Hỗ trợ thiết bị đầu vào / Hỗ trợ bộ điều khiển Xilinx PS / 2

Bảng 1: Trình điều khiển và lõi IP được hỗ trợ trong Linux (Tiếp theo)

Trình điều khiển Xilinx	Lõi IP Xilinx	Vị trí tài xé trong LSP	Mục cấu hình hạt nhân Linux
mii_nhan	opb_spi xps_spi	linux / drivers / char / xilinx_spi	n / a
sysace	opb_sysace xps_sysace	linux / drivers / block / xilinx_sysace	Trình điều khiển thiết bị / Thiết bị chặn / Xilinx trên chip Hệ thống ACE
gpio	opb_gpio plb_gpio xps_gpio	linux / drivers / char / xilinx_gpio	Trình điều khiển thiết bị / Thiết bị ký tự / Xilinx Hỗ trợ GPIO
USB	opb_usb2_device xps_usb2_device	linux / driver / usb / gadget / xilinx_usb	Trình điều khiển thiết bị / Hỗ trợ USB / USB Hỗ trợ Tiện ích / Hỗ trợ cho USB Tiện ích / Bộ điều khiển ngoại vi USB (Trình điều khiển USB Xilinx) Trình điều khiển thiết bị / Hỗ trợ USB / USB Hỗ trợ Tiện ích / Hỗ trợ cho USB Trình điều khiển Tiện ích / USB Tiện ích
phổ thông	n / a	linux / drivers / xilinx_common	n / a

Cấu hình trình điều khiển và Bus nền tảng

Các bản phân phối Linux 2.6 và trình điều khiển thiết bị Xilinx đã bắt đầu chuyển sang mô hình bus nền tảng để khởi tạo trình điều khiển. Điều này có nghĩa là (gần như) tất cả các trình điều khiển Xilinx không còn phụ thuộc vào xparameters.h, chúng cũng không sử dụng tệp _g.c để cấu hình trình điều khiển. Thay vào đó, các tệp virtex.c và virtex.h trong Arch / ppc / platform / 4xx và xilinx_devices.h trong include / linux chỉ định cấu hình trình điều khiển Xilinx. Trình điều khiển thiết bị lấy cấu hình của chúng từ cấu trúc bus nền tảng được điền trong tệp virtex.c. Các tệp virtex.* Hiện phụ thuộc vào xparameters.h, nhưng trong tương lai có thể được thực hiện để phụ thuộc thay vào dữ liệu cấu hình không tĩnh, chẳng hạn như dữ liệu được chuyển tới hạt nhân từ bộ nạp khởi động. Mục đích cuối cùng là cây nhân không cần phải được biên dịch lại để cấu hình lại trình điều khiển thiết bị Xilinx.

Có liên quan Thông tin

Nếu bạn có câu hỏi hoặc sự có liên quan đến IP Xilinx hoặc các trình điều khiển được liên kết với IP đó, hãy liên hệ với Bộ phận hỗ trợ của Xilinx. Trang web hỗ trợ Xilinx ở đây:

<http://support.xilinx.com>

Mặt khác, nếu bạn đã mua MontaVista hoặc Wind River Linux, bạn có quyền được hỗ trợ từ các nhà cung cấp đó.

Internet cũng chứa nhiều thông tin về Linux. Một số tài nguyên Internet có thể được tìm thấy ở đây:

Linux Từ Dự án Scratch (<http://www.linuxfromscratch.org>)

Tiêu chuẩn phân cấp hệ thống tệp (<http://www.pathname.com/fhs>)

Ngoài việc chỉ sử dụng các tìm kiếm trên web, còn có nhiều danh sách email khác nhau mà bạn có thể tìm kiếm hoặc tham gia. Tại thời điểm viết bài này, có thể tìm thấy một danh sách như vậy được gọi là linuxppc-nhúng, ở đây:

<https://ozlabs.org/mailman/listinfo/linuxppc-embedded>

Các kho lưu trữ cho danh sách này có thể được tìm thấy ở đây:

<http://ozlabs.org/pipermail/linuxppc-embedded>

O'Reilly cũng xuất bản một số cuốn sách hay về cách sử dụng và phát triển phần mềm cho Linux. Dưới đây là một số cuốn sách như vậy mà bạn có thể thấy hữu ích:

Chạy Linux, Phiên bản thứ 4 (<http://www.oreilly.com/catalog/runux4/>)

Xây dựng Hệ thống Linux nhúng (<http://www.oreilly.com/catalog/belinuxsys/index.html>)

Tìm hiểu về nhân Linux, Phiên bản thứ 3 (<http://www.oreilly.com/catalog/undosystemlk/index.html>)

Trình điều khiển thiết bị Linux, Phiên bản thứ 3 (<http://www.oreilly.com/catalog/linuxdrive3/index.html>)

Hướng dẫn của Quản trị viên Mạng Linux (<http://www.oreilly.com/catalog/linag3/index.html>)