

libparanut

Generated by Doxygen 1.8.13

Contents

1	libparanut Documentation	1
1.1	Copyright	1
1.2	Modules of the libparanut	1
1.3	Overview	2
1.4	HOWTO	3
1.5	Expectations to the application	4
1.6	Future Ideas for libparanut	5
2	Todo List	7
3	Module Index	11
3.1	Modules	11
4	Class Index	13
4.1	Class List	13
5	File Index	15
5.1	File List	15

6	Module Documentation	17
6.1	Commonly Used Defines	17
6.1.1	Detailed Description	17
6.1.2	Macro Definition Documentation	17
6.1.2.1	ALL_HALTED	17
6.1.2.2	BEGIN_THREADED_LINKED_SEC_CHECK	18
6.1.2.3	BEGIN_THREADED_LINKED_SEC_CHECK_M	18
6.1.2.4	CONVERT_NUMC_TO_MASK	18
6.1.2.5	COPU_CHECK	18
6.1.2.6	TERMNL	18
6.2	Global Variables	19
6.2.1	Detailed Description	19
6.2.2	Variable Documentation	19
6.2.2.1	sp_loc	19
6.2.2.2	tp_loc	19
6.3	ParaNut Control and Status Registers	20
6.3.1	Detailed Description	20
6.3.2	Macro Definition Documentation	20
6.3.2.1	mtval	21
6.3.2.2	pncache	21
6.3.2.3	pncacheinfo	21
6.3.2.4	pncachesets	21
6.3.2.5	pncause	21
6.3.2.6	pnce	21
6.3.2.7	pnclockinfo	22
6.3.2.8	pncpus	22
6.3.2.9	pnepc	22
6.3.2.10	pngrp sel	22
6.3.2.11	pnlm	22
6.3.2.12	pnm2cap	22

6.3.2.13	pnmemsize	23
6.3.2.14	pnx	23
6.3.2.15	pnxsel	23
6.4	ParaNut Custom Instructions	24
6.4.1	Detailed Description	24
6.4.2	Macro Definition Documentation	24
6.4.2.1	CINV	24
6.4.2.2	CWB	25
6.4.2.3	HALT	25
6.5	Internal Assembly Calls	26
6.5.1	Detailed Description	29
6.5.2	Function Documentation	29
6.5.2.1	cache_banks_as()	29
6.5.2.2	cache_sets_as()	30
6.5.2.3	coreid_as()	30
6.5.2.4	destroy_as()	30
6.5.2.5	disable_cache_as()	30
6.5.2.6	ecall_as()	30
6.5.2.7	enable_cache_as()	31
6.5.2.8	enable_CPU_as()	31
6.5.2.9	enter_threaded_mode_as()	31
6.5.2.10	exception_entry_as()	31
6.5.2.11	exception_init_as()	31
6.5.2.12	flush_1024_as()	32
6.5.2.13	flush_128_as()	32
6.5.2.14	flush_2048_as()	32
6.5.2.15	flush_256_as()	32
6.5.2.16	flush_32_as()	32
6.5.2.17	flush_512_as()	33
6.5.2.18	flush_64_as()	33

6.5.2.19	flush_bulk_1024_as()	33
6.5.2.20	flush_bulk_128_as()	33
6.5.2.21	flush_bulk_2048_as()	33
6.5.2.22	flush_bulk_256_as()	34
6.5.2.23	flush_bulk_32_as()	34
6.5.2.24	flush_bulk_512_as()	34
6.5.2.25	flush_bulk_64_as()	34
6.5.2.26	freq_as()	34
6.5.2.27	halt_as()	35
6.5.2.28	init_as()	35
6.5.2.29	invalidate_1024_as()	35
6.5.2.30	invalidate_128_as()	35
6.5.2.31	invalidate_2048_as()	35
6.5.2.32	invalidate_256_as()	36
6.5.2.33	invalidate_32_as()	36
6.5.2.34	invalidate_512_as()	36
6.5.2.35	invalidate_64_as()	36
6.5.2.36	invalidate_bulk_1024_as()	36
6.5.2.37	invalidate_bulk_128_as()	37
6.5.2.38	invalidate_bulk_2048_as()	37
6.5.2.39	invalidate_bulk_256_as()	37
6.5.2.40	invalidate_bulk_32_as()	37
6.5.2.41	invalidate_bulk_512_as()	37
6.5.2.42	invalidate_bulk_64_as()	38
6.5.2.43	m2cap_as()	38
6.5.2.44	mem_size_as()	38
6.5.2.45	numcores_as()	38
6.5.2.46	progress_mepc_as()	39
6.5.2.47	read_MTVAL_as()	39
6.5.2.48	read_PNCAUSE_as()	39

6.5.2.49	read_PNCE_as()	39
6.5.2.50	read_PNEPC_as()	40
6.5.2.51	read_PNLM_as()	40
6.5.2.52	read_PNX_as()	40
6.5.2.53	set_linked_as()	40
6.5.2.54	set_threaded_as()	41
6.5.2.55	simulation_as()	41
6.5.2.56	ticks_as()	41
6.5.2.57	trylock_as()	41
6.5.2.58	unlock_as()	42
6.5.2.59	write_MSTATUS_as()	42
6.5.2.60	write_PNLM_as()	43
6.5.2.61	write_PNXSEL_as()	43
6.5.2.62	writeback_1024_as()	43
6.5.2.63	writeback_128_as()	43
6.5.2.64	writeback_2048_as()	44
6.5.2.65	writeback_256_as()	44
6.5.2.66	writeback_32_as()	44
6.5.2.67	writeback_512_as()	44
6.5.2.68	writeback_64_as()	44
6.5.2.69	writeback_bulk_1024_as()	45
6.5.2.70	writeback_bulk_128_as()	45
6.5.2.71	writeback_bulk_2048_as()	45
6.5.2.72	writeback_bulk_256_as()	45
6.5.2.73	writeback_bulk_32_as()	45
6.5.2.74	writeback_bulk_512_as()	46
6.5.2.75	writeback_bulk_64_as()	46
6.6	libparanut Helpers	47
6.6.1	Detailed Description	47
6.7	Typedefs	48

6.7.1	Detailed Description	48
6.7.2	Typedef Documentation	48
6.7.2.1	_pn_spinlock	48
6.7.2.2	PN_CID	49
6.7.2.3	PN_CMSK	49
6.7.2.4	PN_NUMC	49
6.7.2.5	PN_NUMG	49
6.8	Error Codes	50
6.8.1	Detailed Description	50
6.8.2	Macro Definition Documentation	50
6.8.2.1	PN_ERR_CACHE_LINESIZE	51
6.8.2.2	PN_ERR_COPU	51
6.8.2.3	PN_ERR_EXC	51
6.8.2.4	PN_ERR_LOCKOCC	52
6.8.2.5	PN_ERR_MATCH	52
6.8.2.6	PN_ERR_NOIMP	52
6.8.2.7	PN_ERR_PARAM	52
6.8.2.8	PN_SUCCESS	52
6.9	Modes	53
6.9.1	Detailed Description	53
6.10	libparanut Modules	54
6.10.1	Detailed Description	54
6.11	Base Module	55
6.11.1	Detailed Description	55
6.11.2	Function Documentation	56
6.11.2.1	pn_coreid()	56
6.11.2.2	pn_coreid_g()	56
6.11.2.3	pn_halt()	57
6.11.2.4	pn_halt_CoPU()	57
6.11.2.5	pn_halt_CoPU_gm()	58

6.11.2.6	<code>pn_halt_CoPU_m()</code>	58
6.11.2.7	<code>pn_m2cap()</code>	59
6.11.2.8	<code>pn_m2cap_g()</code>	59
6.11.2.9	<code>pn_m3cap()</code>	60
6.11.2.10	<code>pn_m3cap_g()</code>	60
6.11.2.11	<code>pn_numcores()</code>	61
6.11.2.12	<code>pn_simulation()</code>	61
6.11.2.13	<code>pn_time_ns()</code>	62
6.12	Link Module	63
6.12.1	Detailed Description	63
6.12.2	Function Documentation	63
6.12.2.1	<code>pn_begin_linked()</code>	64
6.12.2.2	<code>pn_begin_linked_gm()</code>	64
6.12.2.3	<code>pn_begin_linked_m()</code>	65
6.12.2.4	<code>pn_end_linked()</code>	66
6.13	Thread Module	67
6.13.1	Detailed Description	67
6.13.2	Function Documentation	67
6.13.2.1	<code>pn_begin_threaded()</code>	67
6.13.2.2	<code>pn_begin_threaded_gm()</code>	68
6.13.2.3	<code>pn_begin_threaded_m()</code>	69
6.13.2.4	<code>pn_end_threaded()</code>	70
6.13.2.5	<code>pn_thread_entry()</code>	70
6.14	Cache Module	71
6.14.1	Detailed Description	71
6.14.2	Function Documentation	72
6.14.2.1	<code>pn_cache_disable()</code>	72
6.14.2.2	<code>pn_cache_enable()</code>	72
6.14.2.3	<code>pn_cache_flush()</code>	72
6.14.2.4	<code>pn_cache_flush_all()</code>	73

6.14.2.5	<code>pn_cache_init()</code>	73
6.14.2.6	<code>pn_cache_invalidate()</code>	73
6.14.2.7	<code>pn_cache_invalidate_all()</code>	74
6.14.2.8	<code>pn_cache_linesize()</code>	74
6.14.2.9	<code>pn_cache_size()</code>	74
6.14.2.10	<code>pn_cache_writeback()</code>	74
6.14.2.11	<code>pn_cache_writeback_all()</code>	75
6.15	Exception Module	76
6.15.1	Detailed Description	76
6.15.2	Function Documentation	76
6.15.2.1	<code>pn_ecall()</code>	77
6.15.2.2	<code>pn_exception_set_handler()</code>	77
6.15.2.3	<code>pn_interrupt_disable()</code>	77
6.15.2.4	<code>pn_interrupt_enable()</code>	78
6.15.2.5	<code>pn_progress_mepc()</code>	78
6.16	Spinlock Module	79
6.16.1	Detailed Description	79
6.16.2	Function Documentation	80
6.16.2.1	<code>pn_spinlock_destroy()</code>	80
6.16.2.2	<code>pn_spinlock_init()</code>	80
6.16.2.3	<code>pn_spinlock_lock()</code>	81
6.16.2.4	<code>pn_spinlock_trylock()</code>	81
6.16.2.5	<code>pn_spinlock_unlock()</code>	82
6.17	libparanut Compile Time Parameters	83
6.17.1	Detailed Description	83
6.17.2	Macro Definition Documentation	83
6.17.2.1	<code>PN_CACHE_LINESIZE</code>	83
6.17.2.2	<code>PN_COMPILE_RAW</code>	84
6.17.2.3	<code>PN_RWIDTH</code>	84

7	Class Documentation	85
7.1	__pn_spinlock Struct Reference	85
7.1.1	Detailed Description	85
7.1.2	Member Data Documentation	86
7.1.2.1	owner_ID	86
8	File Documentation	87
8.1	common/common.h File Reference	87
8.1.1	Detailed Description	89
8.2	common/common_RV32I.S File Reference	90
8.2.1	Detailed Description	90
8.3	common/custom_RV32I.S File Reference	92
8.3.1	Detailed Description	93
8.4	libparanut.h File Reference	94
8.4.1	Detailed Description	98
8.5	Makefile File Reference	98
8.5.1	Detailed Description	98
8.6	pn_base/pn_base.c File Reference	101
8.6.1	Detailed Description	103
8.7	pn_base/pn_base_RV32I.S File Reference	103
8.7.1	Detailed Description	103
8.8	pn_cache/pn_cache.c File Reference	104
8.8.1	Detailed Description	108
8.8.2	Macro Definition Documentation	108
8.8.2.1	BULK_LINES	108
8.8.2.2	sec_check	109
8.9	pn_cache/pn_cache_RV32I_1024.S File Reference	109
8.9.1	Detailed Description	110
8.10	pn_cache/pn_cache_RV32I_128.S File Reference	113
8.10.1	Detailed Description	114
8.11	pn_cache/pn_cache_RV32I_2048.S File Reference	117

8.11.1 Detailed Description	118
8.12 pn_cache/pn_cache_RV32I_256.S File Reference	122
8.12.1 Detailed Description	123
8.13 pn_cache/pn_cache_RV32I_32.S File Reference	126
8.13.1 Detailed Description	127
8.14 pn_cache/pn_cache_RV32I_512.S File Reference	130
8.14.1 Detailed Description	131
8.15 pn_cache/pn_cache_RV32I_64.S File Reference	135
8.15.1 Detailed Description	136
8.16 pn_cache/pn_cache_RV32I_auto.S File Reference	139
8.16.1 Detailed Description	142
8.17 pn_cache/pn_cache_RV32I_buildscript.py File Reference	158
8.17.1 Detailed Description	158
8.18 pn_config.h File Reference	166
8.18.1 Detailed Description	167
8.19 pn_exception/pn_exception.c File Reference	167
8.19.1 Detailed Description	168
8.20 pn_exception/pn_exception_RV32I.S File Reference	169
8.20.1 Detailed Description	169
8.21 pn_link/pn_link.c File Reference	173
8.21.1 Detailed Description	174
8.22 pn_link/pn_link_RV32I.S File Reference	174
8.22.1 Detailed Description	174
8.23 pn_spinlock/pn_spinlock.c File Reference	177
8.23.1 Detailed Description	178
8.24 pn_spinlock/pn_spinlock_RV32I.S File Reference	178
8.24.1 Detailed Description	179
8.25 pn_thread/pn_thread.c File Reference	180
8.25.1 Detailed Description	181
8.26 pn_thread/pn_thread_RV32I.S File Reference	182
8.26.1 Detailed Description	182

Chapter 1

libparanut Documentation

1.1 Copyright

Copyright 2019-2020 Anna Pfuetzner (annakerstin.pfuetzner@gmail.com)

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.2 Modules of the libparanut

Welcome to the libparanut documentation! This is a hardware abstraction library you can use to program your ParaNut without having to stare at Assembly Code.

To do that, the libparanut serves several functions to you on a silver platter. These are grouped into [libparanut Modules](#) for your convenience.

1. The [Base Module](#) contains functions for getting general information about the ParaNut, like the number of cores and such. There's also some functions for halting cores.

2. The [Link Module](#) contains functions for stopping and starting Linked Mode execution. Don't know what that is? Check [Modes](#) or the fine ParaNut Manual.
3. The [Thread Module](#) contains functions for stopping and starting Threaded Mode execution. Same advice goes.
4. The [Cache Module](#) contains functions for controlling the ParaNut cache and getting information about it.
5. The [Exception Module](#) contains functions for controlling Interrupts and Exceptions.
6. The [Spinlock Module](#) is an implementation of a spinlock so you can properly protect and synchronize your stuff in Threaded Mode.

1.3 Overview

The following picture shows a system overview. It might be a little confusing at first. Don't worry about that, just read some more documentation and it will probably all fall into place.

Todo Add English Version and fix format

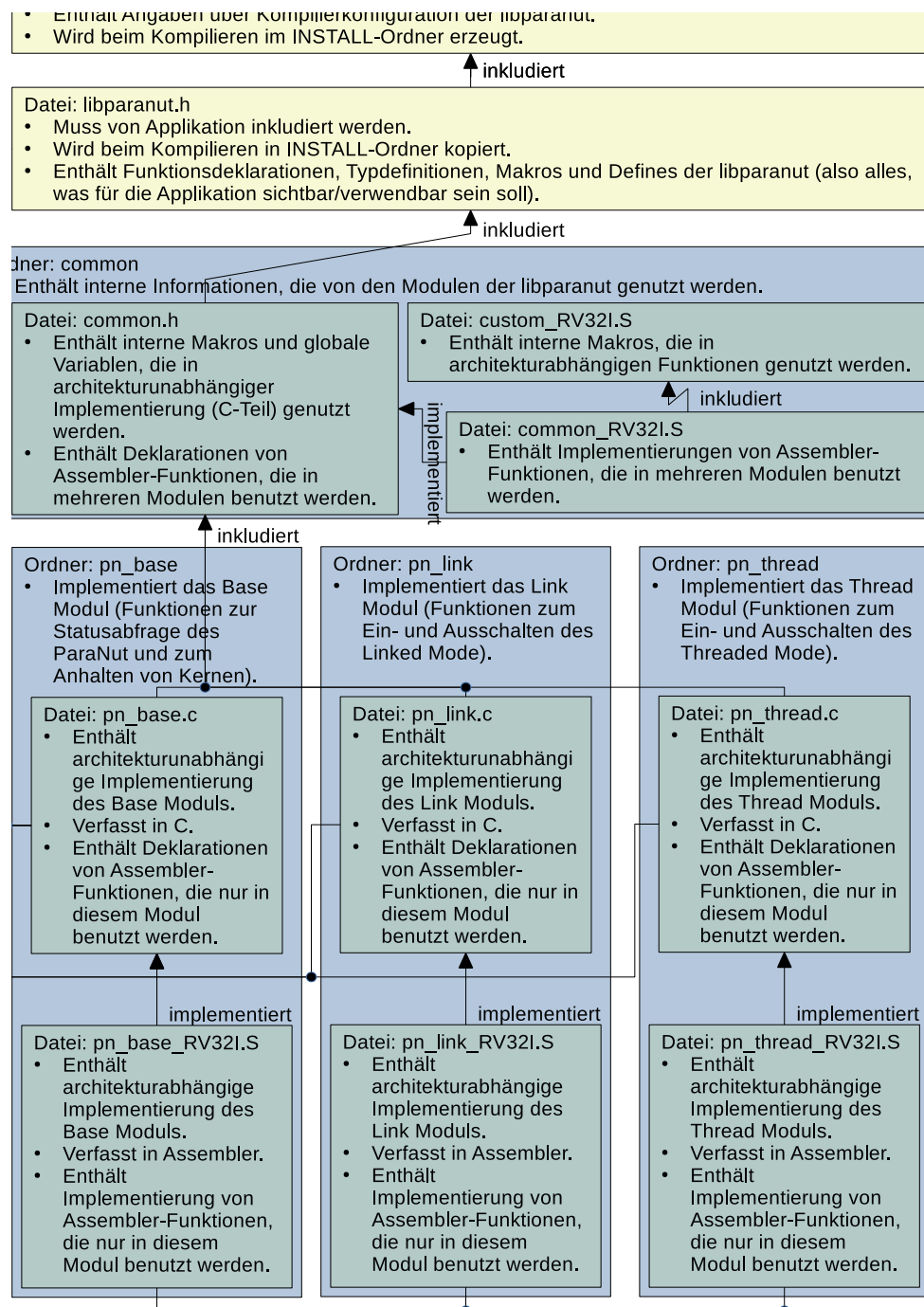


Figure 1.1 Overview of the libparanut

1.4 HOWTO

Todo Add documentation here in case other compilers/ISAs are used someday.

Todo The [Makefile](#) might not be compliant with any other tool than GNU make.

How to make libparanut run with our in-house ParaNut implementation: You will need GNU make and the RISC-V GNU Toolchain for that (see Appendix A in ParaNut Manual). Also, you may need to have Python on your system (so far, no restrictions on the version are known). This assumes you already got the ParaNut up and running!

1. Open up the wonderful [Makefile](#) in the libparanut directory. This is the place where all your wishes come true. Right at the start, there's the section "System Configuration" where you can set a few things to tweak your libparanut. Check out the [libparanut Compile Time Parameters](#) to see what those values mean. The default should be fine at first.
2. Onto the section "Compiler Configuration". See the "PN_ISA" in the [Makefile](#)? This is the Instruction Set Architecture. The very nice [Makefile](#) tree will chose all the right assembly source code files for the libparanut if this variable is set correctly. Currently, our ParaNut implements RISC-V ISA, which is why this value says RV32I. This means you don't have to touch it.
3. The [Makefile](#) also lets you chose the compiler. If you want another compiler than GCC, you will also need to add some if-else logic in the section "Compiler and Assembler Flags".
4. To reduce the code size of the libparanut, you could set "PN_DEBUG" to 0. This means that the libparanut will not be compiled with debug symbols. It should be fine to compile it with debug symbols at first, though.
5. Want everything compiled now? Say

```
gmake all
```

and the magic is done. All the modules are compiled and statically linked into "libparanut.a". You can find it in the newly generated directory named "INSTALL". Do not forget to call clean beforehand if you changed something in the Makefile.

6. Include [libparanut.h](#) in your application. Set your include-path to the INSTALL directory while compiling:

```
-I/path/to/libparanut/INSTALL
```

7. Link libparanut.a into your application. To do that in GCC, you can just put

```
-L/path/to/libparanut/INSTALL -lparanut
```

at the end of your linker/compilation command.

I hope the following example can explain what I mean:

Todo Prettier example, explain Link Module and Spinlock Module.

```
cd /path/to/libparanut
gmake all
cd /path/to/my_application.c
my_gcc -c my_compiler_options -I/path/to/libparanut/INSTALL my_application.c
my_gcc my_link_options my_object.o -L/path/to/libparanut/INSTALL -lparanut
```

Do not forget to also compile in the startup code. To see a full example of what you need to do, check the Makefiles in the other sw/ subdirectories in the GIT.

1.5 Expectations to the application

Here's a list of things that the libparanut expects you to have been done before usage:

1. The [Cache Module](#) expects you to have called [pn_cache_init\(\)](#) before using any of its other functions. Behaviour of the other functions is undefined if not.

2. The [Exception Module](#) expects you to have called `pn_exception_init()` before calling `pn_exception_set_handler()`. I mean, you can set your own exception handler beforehand, but it will not be called if the exception occurs. The `pn_interrupt_enable()`, `pn_interrupt_disable()`, and `pn_progress_mepc()` functions will work with no problem, though.
3. The [Thread Module](#) is the complicated one. It's a little tricky because of its workflow. The CePU internally sets a few needed variables which indicate a status to the CoPU. After a CoPU is woken up, they start executing code at the reset address. This means that the end of the startup code for the CoPUs also needs to be their entrance point to the [Thread Module](#), where they can read the internal variables and figure out what they are supposed to do with them. This entrance point is `pn_thread_entry()`, and it needs to be called in the startup code on all CoPUs. Our in-house startup code does this for you, but I'm leaving that here just in case you want to know.
4. The [Thread Module](#) has one more expectation. The state of the CePU and its memory need to be shared with all cores. That means that there has to be a shared memory section that has to be as big as the memory for the individual cores. The start address of this area has to be put into a globally known location called `shared_mem_start`, and the size has to be put into `shared_mem_size`. This also happens in our build-in startup code, so you do not need to worry about it.

Also, here's some general advice: If you want your startup code to be compatible with many versions of the libparanut, you can check if a certain module is available. The file `pn_config.h`, which is created during compilation time, has some defines set for every module that is enabled. Check out the `pn_config.h` documentation to find out more!

1.6 Future Ideas for libparanut

1. In the [Cache Module](#), split `pn_cache_enable()` and `pn_cache_disable()` into separate functions for data and instruction cache.
2. In the [Thread Module](#), build in POSIX style thread functions.
3. Build a state machine for every core that tracks whether a core is used in linked mode, threaded mode, or doing POSIX threads. Enable the user of the libparanut to use threaded and linked mode in a mixed way.
4. In [Exception Module](#), provide a function to hang in a "raw" exception handler (basically change mtvec).
5. Implement a possibility to check whether or not a module has been initialized already. Also implement a `pn_init()` function which checks for all modules if the module is already initialized, and if it is not, initializes them (provided the module was compiled in).
6. Concerning the `init()` functions: If a function in a module is used despite of the module not being initialized, throw an exception (use `pn_ecall()`).
7. Implement a `pn_strerror()` function which takes an error value and prints what the error means.
8. Implement a global checkable error variable.
9. For threaded and linked mode, implement the possibility to pass a stack size that shall be copied. It should also be able to tell these functions that only the stack from the top function shall be copied.
10. Implement a `pn_atomic_increment()` and `pn_atomic_decrement()` function.
11. Implement `pn_printf()` which can be used in linked mode.
12. In the Makefile, copy the libparanut to the place where the RISC-V toolchain is installed. This would make it easier for applications to compile.

Chapter 2

Todo List

Member [ALL_HALTED](#)

May need a change if there's ever a ParaNut implementation with more than 1 Mode 3 capable CPU.

Module [as](#)

If there ever is a new ParaNut with a different ISA, documentation for the new .S files has to be added in the respective directories.

Module [ba](#)

Currently (Oct. 2019), there is no ParaNut implementation that actually has to use more than one group, which is why the group functions are only implemented as stubs right now.

Member [BEGIN_THREADED_LINKED_SEC_CHECK](#) Take out PN_ERR_NOIMP when group selection is implemented.

Module [ca](#) May need some changes in the future when ParaNut cores get some cache on their own or a proper cache controller is implemented.

Virtual addressing is not implemented in the current ParaNut implementation, but may be featured later on. When the day comes, put in a physical address calculation in the implementation of these functions.

Member [COPU_CHECK](#) May need a change if there's ever a ParaNut implementation with more than 1 Mode 3 capable CPU.

Member [enter_threaded_mode_as](#) (void) Needs changes in case it will ever be allowed to run some cores in linked mode and some in threaded mode.

Module [ex](#) All of these functions may be too RISC-V specific to ever use them on another architecture. I have not done more research on this topic yet. If that's the case, we might even need to add architecture specific .c files for this module. I would name them pn_cache_RV32I.c, pn_cache_RV64I.c, and so on. The Makefile and Documentation will probably need some changes then.

page [libparanut Documentation](#) Add English Version and fix format

Add documentation here in case other compilers/ISAs are used someday.

The [Makefile](#) might not be compliant with any other tool than GNU make.

Prettier example, explain Link Module and Spinlock Module.

File [Makefile](#) The Makefile might not be compliant with any other tool than GNU make. Compatibility with other tools should be tested and listed (maybe in the mainpage). The [HOWTO](#) section needs to be changed, then.

Module [ms](#) The information above may change some day.

Member [pn_begin_linked](#) (PN_NUMC numcores) This functions specification depends a lot on ParaNut hardware design. In later implementations, using linked and threaded mode in a mixed way might be possible. The documentation and implementation will have to be changed, then.

Not yet implemented for more cores than what is available in one group. Will return [PN_ERR_NOIMP](#) when called with more cores.

Not yet implemented for more cores than what is available in one group. Will return [PN_ERR_NOIMP](#) when called with more cores.

Member [pn_begin_linked_gm](#) (PN_CMSK *coremask_array, PN_NUMG array_size) This functions specification depends a lot on ParaNut hardware design. In later implementations, using linked and threaded mode in a mixed way might be possible. The documentation and implementation will have to be changed, then.

Currently only a stub. Will therefore always return [PN_ERR_NOIMP](#).

Currently only a stub. Will therefore always return [PN_ERR_NOIMP](#).

Member [pn_begin_linked_m](#) (PN_CMSK coremask) This functions specification depends a lot on ParaNut hardware design. In later implementations, using linked and threaded mode in a mixed way might be possible. The documentation and implementation will have to be changed, then.

Member [pn_begin_threaded](#) (PN_NUMC numcores) This functions specification depends a lot on ParaNut hardware design. In later implementations, using linked and threaded mode in a mixed way might be possible. The documentation and implementation will have to be changed, then.

Not yet implemented for more cores than what is available in one group. Will return [PN_ERR_NOIMP](#) when called with more cores.

Not yet implemented for more cores than what is available in one group. Will return [PN_ERR_NOIMP](#) when called with more cores.

Member [pn_begin_threaded_gm](#) (PN_CMSK *coremask_array, PN_NUMG array_size) This functions specification depends a lot on ParaNut hardware design. In later implementations, using linked and threaded mode in a mixed way might be possible. The documentation and implementation will have to be changed, then.

Currently only a stub. Will therefore always return [PN_ERR_NOIMP](#).

Currently only a stub. Will therefore always return [PN_ERR_NOIMP](#).

Member [pn_begin_threaded_m](#) (PN_CMSK coremask) This functions specification depends a lot on ParaNut hardware design. In later implementations, using linked and threaded mode in a mixed way might be possible. The

documentation and implementation will have to be changed, then.

Member `pn_coreid_g` (PN_NUMG *groupnum) Currently only a stub. Will therefore always return `PN_ERR_NOIMP`.

Member `PN_ERR_CACHE_LINESIZE` This description should change if other ParaNut architectures or cache line sizes are introduced.

File `pn_exception.c` Layer this better in later versions of libparanut.

Member `pn_halt_CoPU` (PN_CID coreid) Not yet implemented for given core IDs outside of group 0. Will return `PN_ERR_NOIMP` in this case.

Member `pn_halt_CoPU_gm` (PN_CMSK *coremask_array, PN_NUMG array_size) Currently only a stub. Will therefore always return either `PN_ERR_COPU` or `PN_ERR_NOIMP` if executed on CePU.

Member `pn_m2cap_g` (PN_NUMG groupnum) Currently only a stub. Will therefore always return either `PN_ERR_COPU` or `PN_ERR_NOIMP` if executed on CePU.

Member `pn_m3cap` (void) If other cores are ever capable of Mode 3 (and if there ever is a register to get the information from), implement this properly.

Member `pn_m3cap_g` (PN_NUMG groupnum) Currently only a stub. Will therefore always return either `PN_ERR_COPU` or `PN_ERR_NOIMP` if executed on CePU.

Member `pn_time_ns` (void) On RISC-V C calling convention, long long type fits for both RV32 and RV64. Does this hold true on other architectures?

Member `sec_check` Can these functions be used on CoPU?

Member `set_threaded_as` (PN_CMSK coremask) Needs changes in case it will ever be allowed to run some cores in linked mode and some in threaded mode.

Correct this after rewrite of thread module

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Commonly Used Defines	17
Global Variables	19
ParaNut Control and Status Registers	20
ParaNut Custom Instructions	24
Internal Assembly Calls	26
libparanut Helpers	47
Typedefs	48
Error Codes	50
Modes	53
libparanut Modules	54
Base Module	55
Link Module	63
Thread Module	67
Cache Module	71
Exception Module	76
Spinlock Module	79
libparanut Compile Time Parameters	83

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

__pn_spinlock	A synchronization primitive. Use _pn_spinlock instead of this	85
-------------------------------	---	----

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

libparanut.h	API of the libparanut	94
Makefile	Makefile of the libparanut	98
pn_config.h	See Documentaion of libparanut Compile Time Parameters	166
common/ common.h	Contains architecture independent internal prototypes and defines needed in libparanut Modules	87
common/ common_RV32I.S	Contains RV32I assembly implementations of assembly functions called in all modules	90
common/ custom_RV32I.S	Contains defines used in assembly functions of all modules	92
pn_base/ pn_base.c	Contains architecture independent implementations of the Base Module functions	101
pn_base/ pn_base_RV32I.S	Contains RV32I assembly implementations of assembly functions called in pn_base.c	103
pn_cache/ pn_cache.c	Contains architecture independent implementations of the Cache Module functions	104
pn_cache/ pn_cache_RV32I_1024.S	Contains RV32I assembly implementations of assembly functions called in pn_cache.c for cache line size 1024	109
pn_cache/ pn_cache_RV32I_128.S	Contains RV32I assembly implementations of assembly functions called in pn_cache.c for cache line size 128	113
pn_cache/ pn_cache_RV32I_2048.S	Contains RV32I assembly implementations of assembly functions called in pn_cache.c for cache line size 2048	117
pn_cache/ pn_cache_RV32I_256.S	Contains RV32I assembly implementations of assembly functions called in pn_cache.c for cache line size 256	122
pn_cache/ pn_cache_RV32I_32.S	Contains RV32I assembly implementations of assembly functions called in pn_cache.c for cache line size 32	126
pn_cache/ pn_cache_RV32I_512.S	Contains RV32I assembly implementations of assembly functions called in pn_cache.c for cache line size 512	130

pn_cache/ pn_cache_RV32I_64.S	
Contains RV32I assembly implementations of assembly functions called in pn_cache.c for cache line size 64	135
pn_cache/ pn_cache_RV32I_auto.S	
Contains RV32I assembly implementations of assembly functions called in pn_cache.c for all cache line sizes	139
pn_cache/ pn_cache_RV32I_buildscript.py	
Builds the architecture dependent assembly files of the Cache Module since they are performance critical, very dependent on cache line size, and should not take up to much space . . .	158
pn_exception/ pn_exception.c	
Contains (somewhat) architecture independent implementations of the Exception Module functions	167
pn_exception/ pn_exception_RV32I.S	
Contains RV32I assembly implementations of assembly functions called in pn_exception.c . .	169
pn_link/ pn_link.c	
Contains architecture independent implementations of the Link Module functions	173
pn_link/ pn_link_RV32I.S	
Contains RV32I assembly implementations of assembly functions called in pn_link.c	174
pn_spinlock/ pn_spinlock.c	
Contains architecture independent implementations of the Spinlock Module functions	177
pn_spinlock/ pn_spinlock_RV32I.S	
Contains RV32I assembly implementations of assembly functions called in pn_spinlock.c . . .	178
pn_thread/ pn_thread.c	
Contains architecture independent implementations of the Thread Module functions	180
pn_thread/ pn_thread_RV32I.S	
Contains RV32I assembly implementations of assembly functions called in pn_thread.c	182

Chapter 6

Module Documentation

6.1 Commonly Used Defines

Architecture independent defines for internal usage in [libparanut Modules](#).

- `#define COPU_CHECK` if (`coreid_as()` != 0) return `PN_ERR_COPU`;
Checks if we are running on CoPU.
- `#define ALL_HALTED` if (`read_PNCE_as()` > 1) return `PN_ERR_MATCH`;
Checks if all CoPUs are halted.
- `#define BEGIN_THREADED_LINKED_SEC_CHECK`
Performs security checks in `pn_begin_linked()` and `pn_begin_threaded()`.
- `#define BEGIN_THREADED_LINKED_SEC_CHECK_M`
Performs security checks in `pn_begin_linked_m()` and `pn_begin_threaded_m()`.
- `#define CONVERT_NUMC_TO_MASK`
Converts a given number of cores to a minimal bitmask.
- `#define TERMNL "\n\r"`
Terminal newline.

6.1.1 Detailed Description

Architecture independent defines for internal usage in [libparanut Modules](#).

6.1.2 Macro Definition Documentation

6.1.2.1 ALL_HALTED

```
#define ALL_HALTED if (read_PNCE_as() > 1) return PN_ERR_MATCH;
```

Checks if all CoPUs are halted.

Todo May need a change if there's ever a ParaNut implementation with more than 1 Mode 3 capable CPU.

6.1.2.2 BEGIN_THREADED_LINKED_SEC_CHECK

```
#define BEGIN_THREADED_LINKED_SEC_CHECK
```

Value:

```
COPU_CHECK
    ALL_HALTED
    if ((numcores > numcores_as()) || (numcores <= 1))
        return PN_ERR_PARAM;
    if (numcores >= PN_RWIDTH)
        return PN_ERR_NOIMP;
```

Performs security checks in [pn_begin_linked\(\)](#) and [pn_begin_threaded\(\)](#).

Todo Take out PN_ERR_NOIMP when group selection is implemented.

6.1.2.3 BEGIN_THREADED_LINKED_SEC_CHECK_M

```
#define BEGIN_THREADED_LINKED_SEC_CHECK_M
```

Value:

```
COPU_CHECK
    ALL_HALTED
    if (!(coremask & 0b1))
        return PN_ERR_PARAM;
```

Performs security checks in [pn_begin_linked_m\(\)](#) and [pn_begin_threaded_m\(\)](#).

6.1.2.4 CONVERT_NUMC_TO_MASK

```
#define CONVERT_NUMC_TO_MASK
```

Value:

```
for (i = 2; i < numcores; i++)
    coremask |= (1 << i);
```

Converts a given number of cores to a minimal bitmask.

6.1.2.5 COPU_CHECK

```
#define COPU_CHECK if (coreid_as() != 0) return PN_ERR_COPU;
```

Checks if we are running on CoPU.

Todo May need a change if there's ever a ParaNut implementation with more than 1 Mode 3 capable CPU.

6.1.2.6 TERMNL

```
#define TERMNL "\n\r"
```

Terminal newline.

6.2 Global Variables

Global Variables for internal usage in [libparanut Modules](#).

- int [sp_loc](#)
Stack Pointer location, used by [set_linked_as\(\)](#) and [set_threaded_as\(\)](#).
- int [tp_loc](#)
Thread Pointer location, used in [Thread Module](#).

6.2.1 Detailed Description

Global Variables for internal usage in [libparanut Modules](#).

6.2.2 Variable Documentation

6.2.2.1 [sp_loc](#)

`sp_loc`

Stack Pointer location, used by [set_linked_as\(\)](#) and [set_threaded_as\(\)](#).

6.2.2.2 [tp_loc](#)

`tp_loc`

Thread Pointer location, used in [Thread Module](#).

6.3 ParaNut Control and Status Registers

Architecture dependent defines for using the addresses of ParaNut Control and Status Registers.

- `#define pngrpsel 0x7C0`
ParaNut CPU group select register.
- `#define pnce 0x7C1`
ParaNut CPU enable register.
- `#define pnlnm 0x7C2`
ParaNut CPU linked mode register.
- `#define pnxscl 0x7C3`
ParaNut CoPU exception select register.
- `#define pncache 0x7C4`
ParaNut Cache control register.
- `#define pncpus 0xFC0`
ParaNut number of CPUs register.
- `#define pnm2cap 0xFC1`
ParaNut CPU capabilities register.
- `#define pnxc 0xFC2`
ParaNut CoPU exception pending register.
- `#define pncause 0xFC3`
ParaNut CoPU trap cause ID register.
- `#define pnepc 0xFC4`
ParaNut CoPU exception program counter register.
- `#define pncacheinfo 0xFC5`
ParaNut cache information register.
- `#define pncachesets 0xFC6`
ParaNut number of cache sets register.
- `#define pnclockinfo 0xFC7`
ParaNut clock speed information register.
- `#define pnmemsize 0xFC8`
ParaNut memory size register.
- `#define mtval 0x343`
Machine Trap Value Register.

6.3.1 Detailed Description

Architecture dependent defines for using the addresses of ParaNut Control and Status Registers.

Even though it's a convention to write defines in all capital letters, I'm writing all of these in small letters to use them more like I would use a normal RISC-V register name.

Check out the ParaNut Manual for more information.

6.3.2 Macro Definition Documentation

6.3.2.1 mtval

```
#define mtval 0x343
```

Machine Trap Value Register.

Compiler does not recognize this one for some reason, even though it's RISC-V and not ParaNut specific.

6.3.2.2 pncache

```
#define pncache 0x7C4
```

ParaNut Cache control register.

6.3.2.3 pncacheinfo

```
#define pncacheinfo 0xFC5
```

ParaNut cache information register.

6.3.2.4 pncachesets

```
#define pncachesets 0xFC6
```

ParaNut number of cache sets register.

6.3.2.5 pncause

```
#define pncause 0xFC3
```

ParaNut CoPU trap cause ID register.

6.3.2.6 pnice

```
#define pnice 0x7C1
```

ParaNut CPU enable register.

6.3.2.7 pnclockinfo

```
#define pnclockinfo 0xFC7
```

ParaNut clock speed information register.

6.3.2.8 pncpus

```
#define pncpus 0xFC0
```

ParaNut number of CPUs register.

6.3.2.9 pnepc

```
#define pnepc 0xFC4
```

ParaNut CoPU exception program counter register.

6.3.2.10 pngrp sel

```
#define pngrp sel 0x7C0
```

ParaNut CPU group select register.

6.3.2.11 pnlnm

```
#define pnlnm 0x7C2
```

ParaNut CPU linked mode register.

6.3.2.12 pnm2cap

```
#define pnm2cap 0xFC1
```

ParaNut CPU capabilities register.

6.3.2.13 pnmemsize

```
#define pnmemsize 0xFC8
```

ParaNut memory size register.

6.3.2.14 pnx

```
#define pnx 0xFC2
```

ParaNut CoPU exception pending register.

6.3.2.15 pnxsel

```
#define pnxsel 0x7C3
```

ParaNut CoPU exception select register.

6.4 ParaNut Custom Instructions

Architecture dependent defines for ParaNut Custom Instructions which have to be added as binary machine code.

- `#define HALT .word 0x0000000B`
Halts the core which this instruction is executed on.
- `#define CINV(regnum, offset) .word (0x100B | ((regnum) << 15) | (((offset) & 0xfff) << 20))`
Cache invalidate.
- `#define CWB(regnum, offset) .word (0x200B | ((regnum) << 15) | (((offset) & 0xfff) << 20))`
Cache writeback.

6.4.1 Detailed Description

Architecture dependent defines for ParaNut Custom Instructions which have to be added as binary machine code.

Check out the ParaNut Manual for more information. The translation to bytecode can be taken from RISC-V specification.

6.4.2 Macro Definition Documentation

6.4.2.1 CINV

```
#define CINV(  
    regnum,  
    offset ) .word (0x100B | ((regnum) << 15) | (((offset) & 0xfff) << 20))
```

Cache invalidate.

Cache flush.

Parameters

in	<i>regnum</i>	is the number of the hardware register that contains the base address.
in	<i>offset</i>	is the offset that is added to the base address. The resulting address is the memory which is cached and you want to invalidate.

Parameters

in	<i>regnum</i>	is the number of the hardware register that contains the base address.
in	<i>offset</i>	is the offset that is added to the base address. The resulting address is the memory which is cached and you want to flush.

6.4.2.2 CWB

```
#define CWB(  
    regnum,  
    offset ) .word (0x200B | ((regnum) << 15) | (((offset) & 0xffff) << 20))
```

Cache writeback.

Parameters

in	<i>regnum</i>	is the number of the hardware register that contains the base adress.
in	<i>offset</i>	is the offset that is added to the base adress. The resulting adress is the memory which is cached and you want to write back.

6.4.2.3 HALT

```
#define HALT .word 0x0000000B
```

Halts the core which this instruction is executed on.

6.5 Internal Assembly Calls

Calls from the architecture independent implementation to the architecture specific assembly code.

- void `halt_as` (void)
Halts the core this is executed on.
- `PN_NUMC numcores_as` (void)
Get the number of cores in your system.
- `PN_CID coreid_as` (void)
Get the core ID.
- `PN_CMSK read_PNCE_as` (void)
Reads ParaNut CPU enable register.
- void `enable_CPU_as` (`PN_CMSK` coremask)
Enables the cores marked in the bitmask, disables the other ones.
- `PN_CMSK m2cap_as` (void)
Get the register where Mode 2 capability cores are marked.
- int `simulation_as` (void)
Checks if we run in simulation instead of real hardware.

- unsigned long `ticks_as` (void)
Reads machine cycle counter.
- int `freq_as` (void)
Reads pnclockinfo register which contains clock speed.

- void `enable_cache_as` (void)
Enables cache.
- void `disable_cache_as` (void)
Disables cache.
- unsigned int `cache_banks_as` (void)
Returns number of cache banks.
- unsigned int `cache_sets_as` (void)
Returns number of cache sets.
- unsigned int `mem_size_as` (void)
Reads memory size.
- void `invalidate_1024_as` (unsigned int addr, unsigned int endaddr)
Invalidates single cache lines until end address is reached.
- void `invalidate_bulk_1024_as` (unsigned int addr, unsigned int endaddr)
Invalidates 32 cache lines at once until end address is reached.
- void `writeback_1024_as` (unsigned int addr, unsigned int endaddr)
Writes back single cache lines until end address is reached.
- void `writeback_bulk_1024_as` (unsigned int addr, unsigned int endaddr)
Writes back 32 cache lines at once until end address is reached.
- void `flush_1024_as` (unsigned int addr, unsigned int endaddr)
Flushes single cache lines until end address is reached.
- void `flush_bulk_1024_as` (unsigned int addr, unsigned int endaddr)
Flushes 32 cache lines at once until end address is reached.

- void `invalidate_128_as` (unsigned int addr, unsigned int endaddr)

- Invalidates single cache lines until end address is reached.*

 - void `invalidate_bulk_128_as` (unsigned int addr, unsigned int endaddr)

Invalidates 32 cache lines at once until end address is reached.

 - void `writeback_128_as` (unsigned int addr, unsigned int endaddr)

Writes back single cache lines until end address is reached.

 - void `writeback_bulk_128_as` (unsigned int addr, unsigned int endaddr)

Writes back 32 cache lines at once until end address is reached.

 - void `flush_128_as` (unsigned int addr, unsigned int endaddr)

Flushes single cache lines until end address is reached.

 - void `flush_bulk_128_as` (unsigned int addr, unsigned int endaddr)

Flushes 32 cache lines at once until end address is reached.

- void `invalidate_2048_as` (unsigned int addr, unsigned int endaddr)

Invalidates single cache lines until end address is reached.

 - void `invalidate_bulk_2048_as` (unsigned int addr, unsigned int endaddr)

Invalidates 32 cache lines at once until end address is reached.

 - void `writeback_2048_as` (unsigned int addr, unsigned int endaddr)

Writes back single cache lines until end address is reached.

 - void `writeback_bulk_2048_as` (unsigned int addr, unsigned int endaddr)

Writes back 32 cache lines at once until end address is reached.

 - void `flush_2048_as` (unsigned int addr, unsigned int endaddr)

Flushes single cache lines until end address is reached.

 - void `flush_bulk_2048_as` (unsigned int addr, unsigned int endaddr)

Flushes 32 cache lines at once until end address is reached.

- void `invalidate_256_as` (unsigned int addr, unsigned int endaddr)

Invalidates single cache lines until end address is reached.

 - void `invalidate_bulk_256_as` (unsigned int addr, unsigned int endaddr)

Invalidates 32 cache lines at once until end address is reached.

 - void `writeback_256_as` (unsigned int addr, unsigned int endaddr)

Writes back single cache lines until end address is reached.

 - void `writeback_bulk_256_as` (unsigned int addr, unsigned int endaddr)

Writes back 32 cache lines at once until end address is reached.

 - void `flush_256_as` (unsigned int addr, unsigned int endaddr)

Flushes single cache lines until end address is reached.

 - void `flush_bulk_256_as` (unsigned int addr, unsigned int endaddr)

Flushes 32 cache lines at once until end address is reached.

- void `invalidate_32_as` (unsigned int addr, unsigned int endaddr)

Invalidates single cache lines until end address is reached.

 - void `invalidate_bulk_32_as` (unsigned int addr, unsigned int endaddr)

Invalidates 32 cache lines at once until end address is reached.

 - void `writeback_32_as` (unsigned int addr, unsigned int endaddr)

Writes back single cache lines until end address is reached.

 - void `writeback_bulk_32_as` (unsigned int addr, unsigned int endaddr)

Writes back 32 cache lines at once until end address is reached.

 - void `flush_32_as` (unsigned int addr, unsigned int endaddr)

Flushes single cache lines until end address is reached.

 - void `flush_bulk_32_as` (unsigned int addr, unsigned int endaddr)

Flushes 32 cache lines at once until end address is reached.

- void [invalidate_512_as](#) (unsigned int addr, unsigned int endaddr)
Invalidate single cache lines until end address is reached.
 - void [invalidate_bulk_512_as](#) (unsigned int addr, unsigned int endaddr)
Invalidate 32 cache lines at once until end address is reached.
 - void [writeback_512_as](#) (unsigned int addr, unsigned int endaddr)
Writes back single cache lines until end address is reached.
 - void [writeback_bulk_512_as](#) (unsigned int addr, unsigned int endaddr)
Writes back 32 cache lines at once until end address is reached.
 - void [flush_512_as](#) (unsigned int addr, unsigned int endaddr)
Flushes single cache lines until end address is reached.
 - void [flush_bulk_512_as](#) (unsigned int addr, unsigned int endaddr)
Flushes 32 cache lines at once until end address is reached.
-
- void [invalidate_64_as](#) (unsigned int addr, unsigned int endaddr)
Invalidate single cache lines until end address is reached.
 - void [invalidate_bulk_64_as](#) (unsigned int addr, unsigned int endaddr)
Invalidate 32 cache lines at once until end address is reached.
 - void [writeback_64_as](#) (unsigned int addr, unsigned int endaddr)
Writes back single cache lines until end address is reached.
 - void [writeback_bulk_64_as](#) (unsigned int addr, unsigned int endaddr)
Writes back 32 cache lines at once until end address is reached.
 - void [flush_64_as](#) (unsigned int addr, unsigned int endaddr)
Flushes single cache lines until end address is reached.
 - void [flush_bulk_64_as](#) (unsigned int addr, unsigned int endaddr)
Flushes 32 cache lines at once until end address is reached.
-
- void [exception_init_as](#) (void)
Sets Machine Trap-Vector Base-Address Register correctly and disables interrupts.
 - void [exception_entry_as](#) (void)
Entry point for exceptions (includes interrupts).
 - [PN_CMSK read_PNX_as](#) (void)
Reads ParaNut CoPU exception pending register.
 - [PN_CMSK read_PNCAUSE_as](#) (void)
Reads ParaNut CoPU trap cause ID register.
 - [PN_CMSK read_PNEPC_as](#) (void)
Reads ParaNut CoPU exception program counter register.
 - unsigned int [read_MTVAl_as](#) (void)
Reads Machine Trap Value register.
 - void [write_PNXSEL_as](#) ([PN_CMSK](#) coremask)
Writes ParaNut CoPU exception select register.
 - void [write_MSTATUS_as](#) (unsigned int register)
Writes Machine Status register.
 - void [ecall_as](#) (void)
Causes environment call exception.
 - void [progress_mepc_as](#) (void)
Sets MEPC to next instruction.

- `PN_CID set_linked_as (PN_CMSK coremask)`
Puts the cores marked in the bitmask to linked mode and enables them.
- `PN_CMSK read_PNLM_as (void)`
Reads PNLM register.
- `void write_PNLM_as (PN_CMSK coremask)`
Writes PNLM register.

- `int init_as (_pn_spinlock *spinlock)`
Sets lock to free, no conditions checked.
- `int trylock_as (_pn_spinlock *spinlock, PN_CID coreid)`
Tries locking the lock. Fails if not free.
- `int unlock_as (_pn_spinlock *spinlock, PN_CID coreid)`
Tries unlocking. Fails if not owned by current hart.
- `int destroy_as (_pn_spinlock *spinlock, PN_CID coreid)`
Sets lock to dead if it's free or it's owned by current hart.

- `void set_threaded_as (PN_CMSK coremask)`
Takes the linked flag away from all cores and enables the cores in the coremask.
- `void enter_threaded_mode_as (void)`
Prepares CoPU for threaded mode by taking the CePUs stack.

6.5.1 Detailed Description

Calls from the architecture independent implementation to the architecture specific assembly code.

Todo If there ever is a new ParaNut with a different ISA, documentation for the new .S files has to be added in the respective directories.

6.5.2 Function Documentation

6.5.2.1 `cache_banks_as()`

```
unsigned int cache_banks_as (
    void )
```

Returns number of cache banks.

Returns

Number of cache banks.

6.5.2.2 cache_sets_as()

```
unsigned int cache_sets_as (  
    void )
```

Returns number of cache sets.

Returns

Number of cache sets.

6.5.2.3 coreid_as()

```
PN_CID coreid_as (  
    void )
```

Get the core ID.

Returns

The core ID.

6.5.2.4 destroy_as()

```
int destroy_as (  
    _pn_spinlock * spinlock,  
    PN_CID coreid )
```

Sets lock to dead if it's free or it's owned by current hart.

6.5.2.5 disable_cache_as()

```
void disable_cache_as (  
    void )
```

Disables cache.

6.5.2.6 ecall_as()

```
void ecall_as (  
    void )
```

Causes environment call exception.

6.5.2.7 enable_cache_as()

```
void enable_cache_as (
    void )
```

Enables cache.

6.5.2.8 enable_CPU_as()

```
void enable_CPU_as (
    PN_CMSK coremask )
```

Enables the cores marked in the bitmask, disables the other ones.

Has to check stability, too.

6.5.2.9 enter_threaded_mode_as()

```
void enter_threaded_mode_as (
    void )
```

Prepares CoPU for threaded mode by taking the CePUs stack.

Todo Needs changes in case it will ever be allowed to run some cores in linked mode and some in threaded mode.

6.5.2.10 exception_entry_as()

```
void exception_entry_as (
    void )
```

Entry point for exceptions (includes interrupts).

6.5.2.11 exception_init_as()

```
void exception_init_as (
    void )
```

Sets Machine Trap-Vector Base-Address Register correctly and disables interrupts.

6.5.2.12 flush_1024_as()

```
void flush_1024_as (
    unsigned int addr,
    unsigned int endaddr )
```

Flushes single cache lines until end address is reached.

Will at least flush one line.

6.5.2.13 flush_128_as()

```
void flush_128_as (
    unsigned int addr,
    unsigned int endaddr )
```

Flushes single cache lines until end address is reached.

Will at least flush one line.

6.5.2.14 flush_2048_as()

```
void flush_2048_as (
    unsigned int addr,
    unsigned int endaddr )
```

Flushes single cache lines until end address is reached.

Will at least flush one line.

6.5.2.15 flush_256_as()

```
void flush_256_as (
    unsigned int addr,
    unsigned int endaddr )
```

Flushes single cache lines until end address is reached.

Will at least flush one line.

6.5.2.16 flush_32_as()

```
void flush_32_as (
    unsigned int addr,
    unsigned int endaddr )
```

Flushes single cache lines until end address is reached.

Will at least flush one line.

6.5.2.17 flush_512_as()

```
void flush_512_as (
    unsigned int addr,
    unsigned int endaddr )
```

Flushes single cache lines until end address is reached.

Will at least flush one line.

6.5.2.18 flush_64_as()

```
void flush_64_as (
    unsigned int addr,
    unsigned int endaddr )
```

Flushes single cache lines until end address is reached.

Will at least flush one line.

6.5.2.19 flush_bulk_1024_as()

```
void flush_bulk_1024_as (
    unsigned int addr,
    unsigned int endaddr )
```

Flushes 32 cache lines at once until end address is reached.

Will at least flush 32 lines.

6.5.2.20 flush_bulk_128_as()

```
void flush_bulk_128_as (
    unsigned int addr,
    unsigned int endaddr )
```

Flushes 32 cache lines at once until end address is reached.

Will at least flush 32 lines.

6.5.2.21 flush_bulk_2048_as()

```
void flush_bulk_2048_as (
    unsigned int addr,
    unsigned int endaddr )
```

Flushes 32 cache lines at once until end address is reached.

Will at least flush 32 lines.

6.5.2.22 flush_bulk_256_as()

```
void flush_bulk_256_as (
    unsigned int addr,
    unsigned int endaddr )
```

Flushes 32 cache lines at once until end address is reached.

Will at least flush 32 lines.

6.5.2.23 flush_bulk_32_as()

```
void flush_bulk_32_as (
    unsigned int addr,
    unsigned int endaddr )
```

Flushes 32 cache lines at once until end address is reached.

Will at least flush 32 lines.

6.5.2.24 flush_bulk_512_as()

```
void flush_bulk_512_as (
    unsigned int addr,
    unsigned int endaddr )
```

Flushes 32 cache lines at once until end address is reached.

Will at least flush 32 lines.

6.5.2.25 flush_bulk_64_as()

```
void flush_bulk_64_as (
    unsigned int addr,
    unsigned int endaddr )
```

Flushes 32 cache lines at once until end address is reached.

Will at least flush 32 lines.

6.5.2.26 freq_as()

```
int freq_as (
    void )
```

Reads pnclockinfo register which contains clock speed.

Returns

pnclockinfo register.

6.5.2.27 halt_as()

```
void halt_as (
    void )
```

Halts the core this is executed on.

6.5.2.28 init_as()

```
int init_as (
    _pn_spinlock * spinlock )
```

Sets lock to free, no conditions checked.

6.5.2.29 invalidate_1024_as()

```
void invalidate_1024_as (
    unsigned int addr,
    unsigned int endaddr )
```

Invalidates single cache lines until end address is reached.

Will at least invalidate one line.

6.5.2.30 invalidate_128_as()

```
void invalidate_128_as (
    unsigned int addr,
    unsigned int endaddr )
```

Invalidates single cache lines until end address is reached.

Will at least invalidate one line.

6.5.2.31 invalidate_2048_as()

```
void invalidate_2048_as (
    unsigned int addr,
    unsigned int endaddr )
```

Invalidates single cache lines until end address is reached.

Will at least invalidate one line.

6.5.2.32 invalidate_256_as()

```
void invalidate_256_as (
    unsigned int addr,
    unsigned int endaddr )
```

Invalidates single cache lines until end address is reached.

Will at least invalidate one line.

6.5.2.33 invalidate_32_as()

```
void invalidate_32_as (
    unsigned int addr,
    unsigned int endaddr )
```

Invalidates single cache lines until end address is reached.

Will at least invalidate one line.

6.5.2.34 invalidate_512_as()

```
void invalidate_512_as (
    unsigned int addr,
    unsigned int endaddr )
```

Invalidates single cache lines until end address is reached.

Will at least invalidate one line.

6.5.2.35 invalidate_64_as()

```
void invalidate_64_as (
    unsigned int addr,
    unsigned int endaddr )
```

Invalidates single cache lines until end address is reached.

Will at least invalidate one line.

6.5.2.36 invalidate_bulk_1024_as()

```
void invalidate_bulk_1024_as (
    unsigned int addr,
    unsigned int endaddr )
```

Invalidates 32 cache lines at once until end address is reached.

Will at least invalidate 32 lines.

6.5.2.37 invalidate_bulk_128_as()

```
void invalidate_bulk_128_as (
    unsigned int addr,
    unsigned int endaddr )
```

Invalidates 32 cache lines at once until end address is reached.

Will at least invalidate 32 lines.

6.5.2.38 invalidate_bulk_2048_as()

```
void invalidate_bulk_2048_as (
    unsigned int addr,
    unsigned int endaddr )
```

Invalidates 32 cache lines at once until end address is reached.

Will at least invalidate 32 lines.

6.5.2.39 invalidate_bulk_256_as()

```
void invalidate_bulk_256_as (
    unsigned int addr,
    unsigned int endaddr )
```

Invalidates 32 cache lines at once until end address is reached.

Will at least invalidate 32 lines.

6.5.2.40 invalidate_bulk_32_as()

```
void invalidate_bulk_32_as (
    unsigned int addr,
    unsigned int endaddr )
```

Invalidates 32 cache lines at once until end address is reached.

Will at least invalidate 32 lines.

6.5.2.41 invalidate_bulk_512_as()

```
void invalidate_bulk_512_as (
    unsigned int addr,
    unsigned int endaddr )
```

Invalidates 32 cache lines at once until end address is reached.

Will at least invalidate 32 lines.

6.5.2.42 invalidate_bulk_64_as()

```
void invalidate_bulk_64_as (
    unsigned int addr,
    unsigned int endaddr )
```

Invalidates 32 cache lines at once until end address is reached.

Will at least invalidate 32 lines.

6.5.2.43 m2cap_as()

```
PN_CMSK m2cap_as (
    void )
```

Get the register where Mode 2 capability cores are marked.

Returns

The core mask.

6.5.2.44 mem_size_as()

```
unsigned int mem_size_as (
    void )
```

Reads memory size.

Returns

Content of pnmemsize register.

6.5.2.45 numcores_as()

```
PN_NUMC numcores_as (
    void )
```

Get the number of cores in your system.

Returns

The number of cores in your ParaNut implementation.

6.5.2.46 progress_mepc_as()

```
void progress_mepc_as (  
    void )
```

Sets MEPC to next instruction.

6.5.2.47 read_MTVAL_as()

```
unsigned int read_MTVAL_as (  
    void )
```

Reads Machine Trap Value register.

Returns

Machine Trap Value register.

6.5.2.48 read_PNCAUSE_as()

```
PN_CMSK read_PNCAUSE_as (  
    void )
```

Reads ParaNut CoPU trap cause ID register.

Returns

Mask that represents CPUs.

6.5.2.49 read_PNCE_as()

```
PN_CMSK read_PNCE_as (  
    void )
```

Reads ParaNut CPU enable register.

Returns

Mask that represents enabled CPUs.

6.5.2.50 read_PNEPC_as()

```
PN_CMSK read_PNEPC_as (
    void )
```

Reads ParaNut CoPU exception program counter register.

Returns

Mask that represents CPUs.

6.5.2.51 read_PNLM_as()

```
PN_CMSK read_PNLM_as (
    void )
```

Reads PNLM register.

Returns

Bitmask representing cores in linked mode.

6.5.2.52 read_PNX_as()

```
PN_CMSK read_PNX_as (
    void )
```

Reads ParaNut CoPU exception pending register.

Returns

Mask that represents CPUs.

6.5.2.53 set_linked_as()

```
PN_CID set_linked_as (
    PN_CMSK coremask )
```

Puts the cores marked in the bitmask to linked mode and enables them.

Saves the CePUs registers which are preserved across call, sets CPUs to linked, enables the CoPUs, writes the saved values into everyone's registers, and returns the core ID.

Returns

Core ID.

6.5.2.54 set_threaded_as()

```
void set_threaded_as (
    PN_CMSK coremask )
```

Takes the linked flag away from all cores and enables the cores in the coremask.

Todo Needs changes in case it will ever be allowed to run some cores in linked mode and some in threaded mode.
Correct this after rewrite of thread module

6.5.2.55 simulation_as()

```
int simulation_as (
    void )
```

Checks if we run in simulation instead of real hardware.

Returns

Zero if we run on hardware, non-zero if we run in simulation.

6.5.2.56 ticks_as()

```
unsigned long ticks_as (
    void )
```

Reads machine cycle counter.

Returns

Machine cycle counter.

6.5.2.57 trylock_as()

```
int trylock_as (
    _pn_spinlock * spinlock,
    PN_CID coreid )
```

Tries locking the lock. Fails if not free.

6.5.2.58 unlock_as()

```
int unlock_as (
    _pn_spinlock * spinlock,
    PN_CID coreid )
```

Tries unlocking. Fails if not owned by current hart.

6.5.2.59 write_MSTATUS_as()

```
void write_MSTATUS_as (
    unsigned int register )
```

Writes Machine Status register.

Parameters

in	<i>register</i>	is the new value of the register.
----	-----------------	-----------------------------------

6.5.2.60 write_PNLM_as()

```
void write_PNLM_as (
    PN_CMSK coremask )
```

Writes PNLM register.

Parameters

in	<i>Bitmask</i>	representing cores in linked mode.
----	----------------	------------------------------------

6.5.2.61 write_PNXSEL_as()

```
void write_PNXSEL_as (
    PN_CMSK coremask )
```

Writes ParaNut CoPU exception select register.

Parameters

in	<i>coremask</i>	is a mask that represents CPUs. Only 1 bit shall be set.
----	-----------------	--

6.5.2.62 writeback_1024_as()

```
void writeback_1024_as (
    unsigned int addr,
    unsigned int endaddr )
```

Writes back single cache lines until end address is reached.

Will at least writeback one line.

6.5.2.63 writeback_128_as()

```
void writeback_128_as (
    unsigned int addr,
    unsigned int endaddr )
```

Writes back single cache lines until end address is reached.

Will at least writeback one line.

6.5.2.64 writeback_2048_as()

```
void writeback_2048_as (
    unsigned int addr,
    unsigned int endaddr )
```

Writes back single cache lines until end address is reached.

Will at least writeback one line.

6.5.2.65 writeback_256_as()

```
void writeback_256_as (
    unsigned int addr,
    unsigned int endaddr )
```

Writes back single cache lines until end address is reached.

Will at least writeback one line.

6.5.2.66 writeback_32_as()

```
void writeback_32_as (
    unsigned int addr,
    unsigned int endaddr )
```

Writes back single cache lines until end address is reached.

Will at least writeback one line.

6.5.2.67 writeback_512_as()

```
void writeback_512_as (
    unsigned int addr,
    unsigned int endaddr )
```

Writes back single cache lines until end address is reached.

Will at least writeback one line.

6.5.2.68 writeback_64_as()

```
void writeback_64_as (
    unsigned int addr,
    unsigned int endaddr )
```

Writes back single cache lines until end address is reached.

Will at least writeback one line.

6.5.2.69 writeback_bulk_1024_as()

```
void writeback_bulk_1024_as (
    unsigned int addr,
    unsigned int endaddr )
```

Writes back 32 cache lines at once until end address is reached.

Will at least writeback 32 lines.

6.5.2.70 writeback_bulk_128_as()

```
void writeback_bulk_128_as (
    unsigned int addr,
    unsigned int endaddr )
```

Writes back 32 cache lines at once until end address is reached.

Will at least writeback 32 lines.

6.5.2.71 writeback_bulk_2048_as()

```
void writeback_bulk_2048_as (
    unsigned int addr,
    unsigned int endaddr )
```

Writes back 32 cache lines at once until end address is reached.

Will at least writeback 32 lines.

6.5.2.72 writeback_bulk_256_as()

```
void writeback_bulk_256_as (
    unsigned int addr,
    unsigned int endaddr )
```

Writes back 32 cache lines at once until end address is reached.

Will at least writeback 32 lines.

6.5.2.73 writeback_bulk_32_as()

```
void writeback_bulk_32_as (
    unsigned int addr,
    unsigned int endaddr )
```

Writes back 32 cache lines at once until end address is reached.

Will at least writeback 32 lines.

6.5.2.74 writeback_bulk_512_as()

```
void writeback_bulk_512_as (
    unsigned int addr,
    unsigned int endaddr )
```

Writes back 32 cache lines at once until end address is reached.

Will at least writeback 32 lines.

6.5.2.75 writeback_bulk_64_as()

```
void writeback_bulk_64_as (
    unsigned int addr,
    unsigned int endaddr )
```

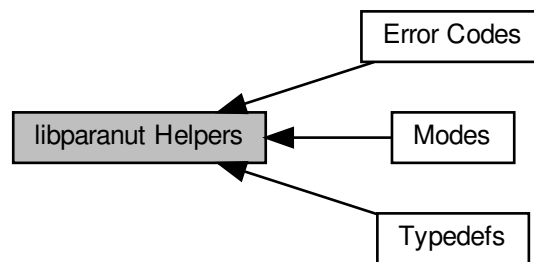
Writes back 32 cache lines at once until end address is reached.

Will at least writeback 32 lines.

6.6 libparanut Helpers

Typedefs and defines of libparanut which can also be used in your application.

Collaboration diagram for libparanut Helpers:



Modules

- [Typedefs](#)
Needed typedefs.
- [Error Codes](#)
Error codes returned by the functions in this library.
- [Modes](#)
Modes of the ParaNut Cores.

6.6.1 Detailed Description

Typedefs and defines of libparanut which can also be used in your application.

6.7 Typedefs

Needed typedefs.

Collaboration diagram for Typedefs:



Typedefs

- typedef struct [__pn_spinlock](#) [__pn_spinlock](#)
Renaming of struct [__pn_spinlock](#) for your convenience.
- typedef int32_t [PN_CID](#)
Signed type that can be used to address any core in this architecture.
- typedef int32_t [PN_NUMC](#)
Signed type that can hold the maximum number of cores in this architecture.
- typedef uint32_t [PN_CMSK](#)
Unsigned type that can act as a core mask.
- typedef int32_t [PN_NUMG](#)
Signed type that can be used to address any group in this architecture.

6.7.1 Detailed Description

Needed typedefs.

Please note that there are several typedefs for differing register widths of the ParaNut. Which ones are used depend on how you set [PN_RWIDTH](#) while compiling ([pn_config.h](#)). Doxygen can only document one of them, so here you are, stuck with the 32-bit version. Check source code of [libparanut.h](#), section Typedefs, if you want to see the others.

6.7.2 Typedef Documentation

6.7.2.1 [__pn_spinlock](#)

[__pn_spinlock](#)

Renaming of struct [__pn_spinlock](#) for your convenience.

Check documentation of [__pn_spinlock](#) to get more information.

6.7.2.2 PN_CID

[PN_CID](#)

Signed type that can be used to address any core in this architecture.

See documentation of [PN_NUMC](#) to understand why we use only the actual register width and not more.

6.7.2.3 PN_CMSK

[PN_CMSK](#)

Unsigned type that can act as a core mask.

This is, of course, the same as the register width of the ParaNut. Unsigned because it directly represents a register.

6.7.2.4 PN_NUMC

[PN_NUMC](#)

Signed type that can hold the maximum number of cores in this architecture.

Let's say your ParaNut has a group register width of 32 bits. This means that there are 4.294.967.296 potential groups. Every group has 32 bits to represent different cores. That means there are 137.438.953.472 cores that can be addressed.

This does, in theory, mean that we need 64 bit to represent the possible number of cores. However, it is deemed to be pretty unrealistic that there will be a ParaNut version with more than 4.294.967.296 cores anytime soon. So, for optimization purposes, we just use 32 bit here. Even half of that is probably not possible in my lifetime, which is why we are not even using unsigned (also because some compilers could throw errors when mixing signed and unsigned types, e.g. in a for loop). One more plus is that we can use these values to signal errors when they are returned by a function.

Same explanation goes in all other register widths. If you really need more, feel free to double the bits.

6.7.2.5 PN_NUMG

[PN_NUMG](#)

Signed type that can be used to address any group in this architecture.

This is, of course, the same as the register width of the ParaNut.

6.8 Error Codes

Error codes returned by the functions in this library.

Collaboration diagram for Error Codes:



- `#define PN_SUCCESS 0`
Successful execution.
- `#define PN_ERR_PARAM (-1)`
Parameter error.
- `#define PN_ERR_NOIMP (-2)`
Function not implemented.
- `#define PN_ERR_COPU (-3)`
CoPU error.
- `#define PN_ERR_MATCH (-4)`
Mode begin and end matching error.
- `#define PN_ERR_LOCKOCC (-5)`
Lock occupied error.
- `#define PN_ERR_CACHE_LINESIZE (-6)`
Weird cache line size error.
- `#define PN_ERR_EXC (-8)`
Exception code not implemented error.

6.8.1 Detailed Description

Error codes returned by the functions in this library.

6.8.2 Macro Definition Documentation

6.8.2.1 PN_ERR_CACHE_LINESIZE

```
#define PN_ERR_CACHE_LINESIZE (-6)
```

Weird cache line size error.

Can occur if libparanut is supposed to run on an architecture that has a cache line size which is not either 32, 64, 128, 256, 512, 1024 or 2048 bit. Should be more of a development error instead of a normal usage error.

In other words, it should not occur if you are just developing middle end stuff while using the libparanut on a deployed ParaNut. Contact the maintainers about this if it still does. The following steps would be necessary to introduce a new cache line size on RV32I libparanut:

- Go to the top of [pn_cache_RV32I_buildscript.py](#) and add the new cache line size in bits to "cache_line_size_list".
- Go to the [Makefile](#) and take a look at the target "cache". Add a new call to the buildscript for your size.
- Go into [pn_cache.c](#) and look at "Assembly Functions". Add your references to the others.
- Also look at the function [pn_cache_init\(\)](#). You will see that there is a switch that decides what functions to use. Add a new case there (before the default, of course!) and put in your newly added functions.
- After a

```
make clean all
```

your new cache line size should be ready to use.

- Change the places in the documentation where the linesizes are listed (should only be here and at the top of [pn_cache_RV32I_buildscript.py](#)). Add your new assembly file to the GIT repo and push that stuff.

Todo This description should change if other ParaNut architectures or cache line sizes are introduced.

6.8.2.2 PN_ERR_COPU

```
#define PN_ERR_COPU (-3)
```

CoPU error.

Function that isn't allowed on CoPU was being executed on CoPU.

6.8.2.3 PN_ERR_EXC

```
#define PN_ERR_EXC (-8)
```

Exception code not implemented error.

You tried callin [pn_exception_set_handler\(\)](#) with an invalid exception code.

6.8.2.4 PN_ERR_LOCKOCC

```
#define PN_ERR_LOCKOCC (-5)
```

Lock occupied error.

Can occur if you tried destroying an occupied lock or used the trylock function on an occupied lock.

6.8.2.5 PN_ERR_MATCH

```
#define PN_ERR_MATCH (-4)
```

Mode begin and end matching error.

Functions for beginning and ending linked and threaded mode have to be matched. Linked and threaded mode shall not be mixed.

6.8.2.6 PN_ERR_NOIMP

```
#define PN_ERR_NOIMP (-2)
```

Function not implemented.

The libparanut function is not yet implemented.

6.8.2.7 PN_ERR_PARAM

```
#define PN_ERR_PARAM (-1)
```

Parameter error.

The parameters given to a function were wrong (i.e. out of range).

6.8.2.8 PN_SUCCESS

```
#define PN_SUCCESS 0
```

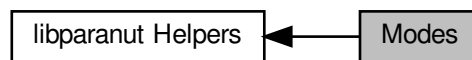
Successful execution.

Implies that a function finished successfully.

6.9 Modes

Modes of the ParaNut Cores.

Collaboration diagram for Modes:



- #define `PN_M0` 0x0U
Mode 0 (halted Mode).
- #define `PN_M1` 0x1U
Mode 1 (linked Mode).
- #define `PN_M2` 0x2U
Mode 2 (unlinked or threaded Mode).
- #define `PN_M3` 0x3U
Mode 3 (autonomous Mode).

6.9.1 Detailed Description

Modes of the ParaNut Cores.

The CePU can only ever operate in Mode 3 (autonomous). It is still shown as capable of Mode 2 (threaded Mode) because Mode 3 is an extension in functionality in comparison to Mode 2. Mode 2 cores do not handle their own interrupts/exceptions, which a Mode 3 core does.

It can also be set into Mode 0, which does not break hardware debugging support.

The CePU is the only core capable of changing other cores Modes.

The CoPUs are never capable of Mode 3. They may be capable of Mode 2, which means they are able to fetch their own instructions and are therefore able to do different work in parallel to the CePU. They are, at minimum, capable of Mode 1 (linked Mode), which means it will execute the same instructions as the CePU on different data. This does not start until the CePU is also told to now start executing in linked mode, though.

Todo The information above may change some day.

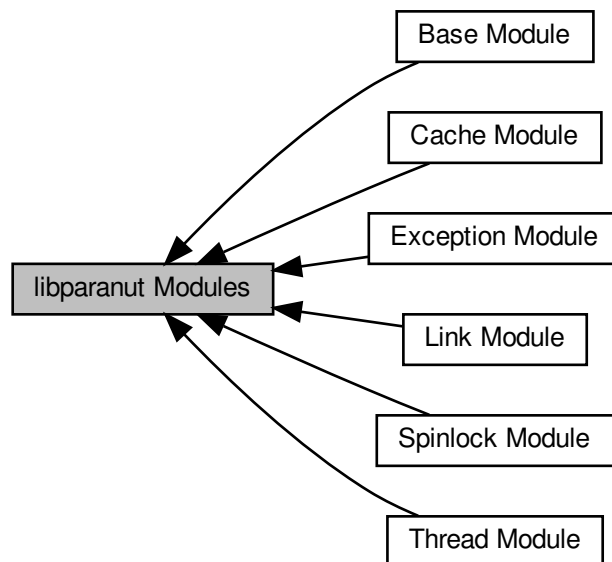
Which Mode the CoPUs are in after system reset is an implementation detail of the ParaNut itself and the startup code.

For further information, check ParaNut Manual.

6.10 libparanut Modules

Modules of libparanut.

Collaboration diagram for libparanut Modules:



Modules

- [Base Module](#)
Functions for getting the status of your ParaNut and halting cores.
- [Link Module](#)
Functions for using the linked mode.
- [Thread Module](#)
Functions for using the threaded mode.
- [Cache Module](#)
Special functions for controlling the shared ParaNut cache.
- [Exception Module](#)
Functions for controlling the handling of interrupts/exceptions.
- [Spinlock Module](#)
Functions and structure used for synchronizing memory access.

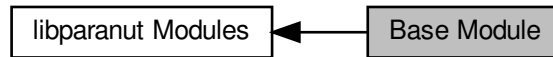
6.10.1 Detailed Description

Modules of libparanut.

6.11 Base Module

Functions for getting the status of your ParaNut and halting cores.

Collaboration diagram for Base Module:



- [PN_NUMC pn_numcores](#) (void)
Get the number of cores in your system.
- [PN_CMSK pn_m2cap](#) (void)
Check which cores are capable of Mode 2 operation.
- [PN_CMSK pn_m2cap_g](#) (PN_NUMG groupnum)
Check which cores are capable of Mode 2 operation.
- [PN_CMSK pn_m3cap](#) (void)
Check which cores are capable of Mode 3 operation.
- [PN_CMSK pn_m3cap_g](#) (PN_NUMG groupnum)
Check which cores are capable of Mode 3 operation.
- [PN_CID pn_coreid](#) (void)
Get the ID of the core that this function is executed on.
- [PN_CID pn_coreid_g](#) (PN_NUMG *groupnum)
Get the ID and group number of the core that this code is running on.
- void [pn_halt](#) (void)
Halt whatever core the function is executed on.
- int [pn_halt_CoPU](#) (PN_CID coreid)
Halts a CoPU.
- int [pn_halt_CoPU_m](#) (PN_CMSK coremask)
Halts one or more CoPUs.
- int [pn_halt_CoPU_gm](#) (PN_CMSK *coremask_array, PN_NUMG array_size)
Halts the CoPUs specified in the coremask_array.
- long long int [pn_time_ns](#) (void)
Returns system time in ns. Does not care for overflow.
- int [pn_simulation](#) (void)
Checks if we run in simulation instead of real hardware.

6.11.1 Detailed Description

Functions for getting the status of your ParaNut and halting cores.

Concerning the `_g` functions: If your ParaNut implementation has more cores than what is the standard register size on your system (i.e. register size is 32, but you got more than 32 cores including the CePU), you have to chose the group of cores that you want to talk to. For that, most functions in this section have a group version (suffix `_g`) which has an additional group parameter.

This works the following way: If your register size is 32, than the CePU is in group 0 (0x00000000) and is always represented by the first bit in the core masks (0x00000001). The first CoPU is also in group 0 and represented by the second bit (0x00000002). After that comes the second CoPU, represented by 0x00000004. And so on until CoPU 30. The 31st CoPU is then represented by group 1 (0x00000001) and the first bit (0x00000001). The 63rd CoPU is represented by group 2 (0x00000002) and the first bit (0x00000001). The 95th CoPU is represented by group 3 (0x00000003) and the first bit (0x00000001). This information may become untrue if a big-endian version of the ParaNut is ever released.

For further information on this topic, you should check the ParaNut Manual itself. Also, the documentation on the ParaNut [Modes](#) that is included in here could clear some things up. The base module also contains functions that are used in other modules as well.

Todo Currently (Oct. 2019), there is no ParaNut implementation that actually has to use more than one group, which is why the group functions are only implemented as stubs right now.

6.11.2 Function Documentation

6.11.2.1 `pn_coreid()`

```
PN_CID pn_coreid (
    void )
```

Get the ID of the core that this function is executed on.

Returns

The core ID. Starts with 0 for CePU. Can not return an error.

6.11.2.2 `pn_coreid_g()`

```
PN_CID pn_coreid_g (
    PN_NUMG * groupnum )
```

Get the ID and group number of the core that this code is running on.

Todo Currently only a stub. Will therefore always return `PN_ERR_NOIMP`.

Parameters

out	<i>groupnum</i>	is a reference to be filled with the group number of the core.
-----	-----------------	--

Returns

The core ID. Starts with 0 for CePU. Does not start again when in another group than group 0. Can not return an error.

Todo Group function implementation.

6.11.2.3 pn_halt()

```
void pn_halt (
    void )
```

Halt whatever core the function is executed on.

If executed on a core, it will be set to Mode 0 and stop operation. Causes reset of program counter to reset address on a Mode 2 capable CPU.

If executed on CePU, also halts all other CoPUs on system.

See documentation of [Modes](#) for more information.

This function returns nothing because it should not be possible for any core to leave this state on its own.

6.11.2.4 pn_halt_CoPU()

```
int pn_halt_CoPU (
    PN_CID coreid )
```

Halts a CoPU.

Sets the CoPU with the given ID into a halted state (Mode 0).

See documentation of [Modes](#) for more information.

Cannot be executed on CoPU.

Halting an already halted core results in failure ([PN_ERR_PARAM](#)) to aid debugging.

Todo Not yet implemented for given core IDs outside of group 0. Will return [PN_ERR_NOIMP](#) in this case.

Parameters

in	<i>coreid</i>	is the ID of the CoPUs you want to halt. Since ID 0 represents the CePU, this function will throw an error when given 0 to aid debugging. If you want the CePU to halt, use function pn_halt() .
----	---------------	--

Returns

Either [PN_SUCCESS](#), [PN_ERR_PARAM](#) or [PN_ERR_COPU](#).

Todo Group selection.

6.11.2.5 pn_halt_CoPU_gm()

```
int pn_halt_CoPU_gm (
    PN_CMSK * coremask_array,
    PN_NUMG array_size )
```

Halts the CoPUs specified in the coremask_array.

Todo Currently only a stub. Will therefore always return either [PN_ERR_COPU](#) or [PN_ERR_NOIMP](#) if executed on CePU.

Sets the CoPUs represented by bitmask and their position in the array (= their group number) into a halted state (Mode 0).

See documentation of [Modes](#) for more information.

Cannot be executed on CoPU.

Halting an already halted core results in failure ([PN_ERR_PARAM](#)) to aid debugging.

Parameters

in	<i>coremask_array</i>	is a pointer to the start of a coremask array. The position of the mask in the array represents the group number of the cores. When all bits are set to 0, this function will return an error (PN_ERR_PARAM) to aid debugging. Also, setting the first bit to 1 will return an error (PN_ERR_PARAM) since it represents the CePU, which should be halted with pn_halt() .
in	<i>array_size</i>	is the number of entries in the coremask_array.

Returns

Either [PN_SUCCESS](#), [PN_ERR_PARAM](#) or [PN_ERR_COPU](#).

Todo Group function implementation.

6.11.2.6 pn_halt_CoPU_m()

```
int pn_halt_CoPU_m (
    PN_CMSK coremask )
```

Halts one or more CoPUs.

Sets the CoPUs represented in the bitmask into a halted state (Mode 0).

See documentation of [Modes](#) for more information.

Cannot be executed on CoPU.

Halting an already halted core results in failure ([PN_ERR_PARAM](#)) to aid debugging.

Parameters

in	<i>coremask</i>	is the bitmask representing the CoPUs you want to halt. When all bits are set to 0, this function will return an error (PN_ERR_PARAM) to aid debugging. Also, setting the first bit to 1 will return an error (PN_ERR_PARAM) since it represents the CePU, which should be halted with pn_halt() .
----	-----------------	--

Returns

Either [PN_SUCCESS](#), [PN_ERR_PARAM](#) or [PN_ERR_COPU](#).

6.11.2.7 `pn_m2cap()`

```
PN_CMSK pn_m2cap (
    void )
```

Check which cores are capable of Mode 2 operation.

See documentation of [Modes](#) for more information.

Cannot be called from CoPU.

Returns

A bitmask representing your ParaNut cores or [PN_ERR_COPU](#). If a bit is set to 1, it means that the core is capable of operating in Mode 2.

6.11.2.8 `pn_m2cap_g()`

```
PN_CMSK pn_m2cap_g (
    PN_NUMG groupnum )
```

Check which cores are capable of Mode 2 operation.

Todo Currently only a stub. Will therefore always return either [PN_ERR_COPU](#) or [PN_ERR_NOIMP](#) if executed on CePU.

See documentation of [Modes](#) for more information.

Cannot be called from CoPU.

Parameters

in	<i>groupnum</i>	is the group of cores you want to know about.
----	-----------------	---

Returns

A bitmask representing your ParaNut cores or [PN_ERR_COPU](#). If a bit is set to 1, it means that the core is capable of operating in Mode 2.

Todo Group function implementation.

6.11.2.9 `pn_m3cap()`

```
PN_CMSK pn_m3cap (
    void )
```

Check which cores are capable of Mode 3 operation.

Attention

This function will, in the current ParaNut implementation, return a hard coded 1 or [PN_ERR_COPU](#) if executed on CoPU. The reason for this is that only the CePU is capable of Mode 3.

See documentation of [Modes](#) for more information.

Cannot be called from CoPU.

Returns

A bitmask representing your ParaNut cores or [PN_ERR_COPU](#). If a bit is set to 1, it means that the core is capable of operating in Mode 3.

Todo If other cores are ever capable of Mode 3 (and if there ever is a register to get the information from), implement this properly.

6.11.2.10 `pn_m3cap_g()`

```
PN_CMSK pn_m3cap_g (
    PN_NUMG groupnum )
```

Check which cores are capable of Mode 3 operation.

Todo Currently only a stub. Will therefore always return either [PN_ERR_COPU](#) or [PN_ERR_NOIMP](#) if executed on CePU.

See documentation of [Modes](#) for more information.

Cannot be called from CoPU.

Parameters

in	<i>groupnum</i>	is the group of cores you want to know about.
----	-----------------	---

Returns

A bitmask representing your ParaNut cores or [PN_ERR_COPU](#). If a bit is set to 1, it means that the core is capable of operating in Mode 3.

Todo Group function implementation.

6.11.2.11 pn_numcores()

```
PN_NUMC pn_numcores (  
    void )
```

Get the number of cores in your system.

Cannot be called from CoPU.

Returns

The number of cores in your ParaNut implementation or [PN_ERR_COPU](#).

6.11.2.12 pn_simulation()

```
int pn_simulation (  
    void )
```

Checks if we run in simulation instead of real hardware.

Warning

This function is a thing that only works with our specific ParaNut simulation. If the simulation changes, this needs to be changed, too.

Returns

Zero if we run on hardware, non-zero if we run in simulation.

6.11.2.13 pn_time_ns()

```
long long int pn_time_ns (
    void )
```

Returns system time in ns. Does not care for overflow.

Cannot be executed on CoPU.

The first time executing this function takes the longest time since it has to initialize the frequency and an internal conversion factor. So if you want to use it for time measurement, you can call the function once before actual measurement to make the values more comparable.

When testing on my ParaNut, this made a difference of around 2000 ticks.

Returns

System time in ns or [PN_ERR_COPU](#).

Todo On RISC-V C calling convention, long long type fits for both RV32 and RV64. Does this hold true on other architectures?

$\text{time in nanoseconds} = \text{time in seconds} * 1000000000$
 $\text{frequency} = \text{ticks} / \text{second} \Rightarrow \text{time in seconds} = \text{ticks} / \text{frequency}$

From those two, it follows that $\text{time in nanoseconds} = (\text{ticks} / \text{frequency}) * 1000000000$

Which is equivalent to $\text{time in nanoseconds} = \text{ticks} * (1000000000 / \text{frequency})$

With the frequency values that the ParaNut currently has, this is accurate enough.

Todo This might change in the future.

6.12 Link Module

Functions for using the linked mode.

Collaboration diagram for Link Module:



- [PN_CID pn_begin_linked](#) ([PN_NUMC](#) numcores)
Links a given number of CoPUs to the CePU.
- [PN_CID pn_begin_linked_m](#) ([PN_CMSK](#) coremask)
Links the CPUs specified in the coremask.
- [PN_CID pn_begin_linked_gm](#) ([PN_CMSK](#) *coremask_array, [PN_NUMG](#) array_size)
Links the CPUs specified in the coremask_array.
- `int pn_end_linked` (void)
Ends linked execution.

6.12.1 Detailed Description

Functions for using the linked mode.

Also see [Modes](#).

Warning

If you want to use the Linked Module on RISC-V ParaNut, your ParaNut has to support the M Extension and you have to compile your application with the flag `mabi=rv32im`. The libparanut [Makefile](#) sets this flag automatically when you chose the [Link Module](#) or one of the Modules that has [Link Module](#) as a dependency.

6.12.2 Function Documentation

6.12.2.1 `pn_begin_linked()`

```
PN_CID pn_begin_linked (
    PN_NUMC numcores )
```

Links a given number of CoPUs to the CePU.

Todo This functions specification depends a lot on ParaNut hardware design. In later implementations, using linked and threaded mode in a mixed way might be possible. The documentation and implementation will have to be changed, then.

Todo Not yet implemented for more cores than what is available in one group. Will return `PN_ERR_NOIMP` when called with more cores.

Sets `numcores-1` CoPUs to Mode 1 (linked Mode) so they start executing the instruction stream fetched by the CePU. This function will return an error (`PN_ERR_MATCH`) if the CoPUs are not all halted.

See documentation of [Modes](#) for more information.

Cannot be executed on CoPU. Cannot be mixed with threaded mode or other linked mode functions, until `pn_end_linked()` is called.

Attention

All data that you want to preserve after `pn_end_linked()` was called can not be stored on stack. You can make it static or global.

Warning

There is no guarantee that the execution of code actually happens at the same time on CePUs and CoPUs.

Parameters

in	<code>numcores</code>	is the number of cores that shall be linked together. A value of 0 or 1 will return an error (<code>PN_ERR_PARAM</code>) to aid debugging.
----	-----------------------	--

Returns

The ID of the core, or `PN_ERR_MATCH`, `PN_ERR_PARAM`, or `PN_ERR_COPU`.

6.12.2.2 `pn_begin_linked_gm()`

```
PN_CID pn_begin_linked_gm (
    PN_CMSK * coremask_array,
    PN_NUMG array_size )
```

Links the CPUs specified in the `coremask_array`.

Todo This functions specification depends a lot on ParaNut hardware design. In later implementations, using linked and threaded mode in a mixed way might be possible. The documentation and implementation will have to be changed, then.

Todo Currently only a stub. Will therefore always return `PN_ERR_NOIMP`.

Sets the CoPUs represented by bitmask and their position in the array (= their group number) to Mode 1 (linked Mode) so they start executing the instruction stream fetched by the CePU. This function will return an error (`PN_ERR_MATCH`) if the CoPUs are not all halted.

See documentation of [Modes](#) for more information.

Cannot be executed on CoPU. Cannot be mixed with threaded mode or other linked mode functions, until `pn_end_linked()` is called.

Attention

All data that you want to preserve after `pn_end_linked()` was called can not be stored on stack. You can make it static or global.

Warning

There is no guarantee that the execution of code actually happens at the same time on CePUs and CoPUs.

Parameters

in	<code>coremask_array</code>	is a pointer to the start of a coremask array. The position of the mask in the array represents the group number of the cores. When all bits are set to 0, this function will return an error (<code>PN_ERR_PARAM</code>) to aid debugging. Also, not setting the first bit to 1 will return an error (<code>PN_ERR_PARAM</code>) since it represents the CePU (which needs to be linked, too).
in	<code>array_size</code>	is the number of entries in the coremask_array.

Returns

The ID of the core, or `PN_ERR_MATCH`, `PN_ERR_PARAM`, or `PN_ERR_COPU`.

Todo Group function implementation.

6.12.2.3 `pn_begin_linked_m()`

```
PN_CID pn_begin_linked_m (
    PN_CMSK coremask )
```

Links the CPUs specified in the coremask.

Todo This functions specification depends a lot on ParaNut hardware design. In later implementations, using linked and threaded mode in a mixed way might be possible. The documentation and implementation will have to be changed, then.

Sets the CoPUs represented by the bitmask to Mode 1 (linked Mode) so they start executing the instruction stream fetched by the CePU. This function will return an error ([PN_ERR_MATCH](#)) if the CoPUs are not all halted.

See documentation of [Modes](#) for more information.

Cannot be executed on CoPU. Cannot be mixed with threaded mode or other linked mode functions, until [pn_end_linked\(\)](#) is called.

Attention

All data that you want to preserve after [pn_end_linked\(\)](#) was called can not be stored on stack. You can make it static or global.

Warning

There is no guarantee that the execution of code actually happens at the same time on CePUs and CoPUs.

Parameters

in	<i>coremask</i>	is the bitmask representing the CoPUs you want to link. When all bits are set to 0, this function will return an error (PN_ERR_PARAM) to aid debugging. Also, not setting the first bit to 1 will return an error (PN_ERR_PARAM) since it represents the CePU (which needs to be linked, too).
----	-----------------	--

Returns

The ID of the core, or [PN_ERR_MATCH](#), [PN_ERR_PARAM](#), or [PN_ERR_COPU](#).

6.12.2.4 [pn_end_linked\(\)](#)

```
int pn_end_linked (
    void )
```

Ends linked execution.

Halts all CoPUs that are currently linked together, effectively ending the linked execution. Will fail if there are no cores linked together.

See documentation of [Modes](#) for more information.

Can be executed on CoPU, but will do nothing then.

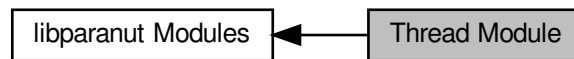
Returns

Either [PN_SUCCESS](#) or [PN_ERR_MATCH](#).

6.13 Thread Module

Functions for using the threaded mode.

Collaboration diagram for Thread Module:



- void `pn_thread_entry` (void)
Function that has to be called for CoPUs at the end of the startup code.
- `PN_CID pn_begin_threaded` (`PN_NUMC` numcores)
Puts numcores CPUs in threaded mode.
- `PN_CID pn_begin_threaded_m` (`PN_CMSK` coremask)
Puts the CPUs specified in the coremask in threaded mode.
- `PN_CID pn_begin_threaded_gm` (`PN_CMSK` *coremask_array, `PN_NUMG` array_size)
Puts the CPUs specified in the coremask_array in threaded mode.
- int `pn_end_threaded` (void)
Ends threaded execution.

6.13.1 Detailed Description

Functions for using the threaded mode.

Also see [Modes](#).

6.13.2 Function Documentation

6.13.2.1 `pn_begin_threaded()`

```
PN_CID pn_begin_threaded (
    PN_NUMC numcores )
```

Puts numcores CPUs in threaded mode.

Todo This functions specification depends a lot on ParaNut hardware design. In later implementations, using linked and threaded mode in a mixed way might be possible. The documentation and implementation will have to be changed, then.

Todo Not yet implemented for more cores than what is available in one group. Will return `PN_ERR_NOIMP` when called with more cores.

Sets `numcores-1` CoPUs to Mode 2 (unlinked Mode) so they start executing the following code in parallel. This function will return an error (`PN_ERR_MATCH`) if the CoPUs are not all halted.

See documentation of [Modes](#) for more information.

Cannot be executed on CoPU. Cannot be mixed with linked mode or other begin threaded functions, until `pn_end_threaded()` is called.

Attention

All data that you want to preserve after `pn_end_threaded()` was called can not be stored on stack. You can make it static or global.

Warning

There is no guarantee that the execution of code actually happens at the same time on CePUs and CoPUs.

Parameters

in	<i>numcores</i>	is the number of cores that shall run threaded. A value of 0 or 1 will return an error (<code>PN_ERR_PARAM</code>) to aid debugging.
----	-----------------	--

Returns

The ID of the core, or `PN_ERR_MATCH`, `PN_ERR_PARAM`, or `PN_ERR_COPU`.

6.13.2.2 `pn_begin_threaded_gm()`

```
PN_CID pn_begin_threaded_gm (
    PN_CMSK * coremask_array,
    PN_NUMG array_size )
```

Puts the CPUs specified in the `coremask_array` in threaded mode.

Todo This functions specification depends a lot on ParaNut hardware design. In later implementations, using linked and threaded mode in a mixed way might be possible. The documentation and implementation will have to be changed, then.

Todo Currently only a stub. Will therefore always return `PN_ERR_NOIMP`.

Sets the CoPUs represented by bitmask and their position in the array (= their group number) to Mode 2 (unlinked Mode) so they start executing the following code in parallel. This function will return an error (`PN_ERR_MATCH`) if the CoPUs are not all halted.

See documentation of [Modes](#) for more information.

Cannot be executed on CoPU. Cannot be mixed with linked mode or other begin threaded functions, until `pn_end_threaded()` is called.

Attention

All data that you want to preserve after `pn_end_threaded()` was called can not be stored on stack. You can make it static or global.

Warning

There is no guarantee that the execution of code actually happens at the same time on CePUs and CoPUs.

Parameters

in	<code>coremask_array</code>	is a pointer to the start of a coremask array. The position of the mask in the array represents the group number of the cores. When all bits are set to 0, this function will return an error (<code>PN_ERR_PARAM</code>) to aid debugging. Also, not setting the first bit to 1 will return an error (<code>PN_ERR_PARAM</code>) since it represents the CePU (which needs to run threaded, too).
in	<code>array_size</code>	is the number of entries in the <code>coremask_array</code> .

Returns

The ID of the core, or `PN_ERR_MATCH`, `PN_ERR_PARAM`, or `PN_ERR_COPU`.

Todo Group function implementation.

6.13.2.3 pn_begin_threaded_m()

```
PN_CID pn_begin_threaded_m (
    PN_CMSK coremask )
```

Puts the CPUs specified in the coremask in threaded mode.

Todo This functions specification depends a lot on ParaNut hardware design. In later implementations, using linked and threaded mode in a mixed way might be possible. The documentation and implementation will have to be changed, then.

Sets the CoPUs represented by the bitmask to Mode 2 (unlinked Mode) so they start executing the following code in parallel. This function will return an error (`PN_ERR_MATCH`) if the CoPUs are not all halted.

See documentation of [Modes](#) for more information.

Cannot be executed on CoPU. Cannot be mixed with linked mode or other begin threaded functions, until `pn_end_threaded()` is called.

Attention

All data that you want to preserve after `pn_end_threaded()` was called can not be stored on stack. You can make it static or global.

Warning

There is no guarantee that the execution of code actually happens at the same time on CePUs and CoPUs.

Parameters

in	<i>coremask</i>	is the bitmask representing the CoPUs you want to run threaded. When all bits are set to 0, this function will return an error (PN_ERR_PARAM) to aid debugging. Also, not setting the first bit to 1 will return an error (PN_ERR_PARAM) since it represents the CePU (which needs to run threaded, too).
----	-----------------	---

Returns

The ID of the core, or [PN_ERR_MATCH](#), [PN_ERR_PARAM](#), or [PN_ERR_COPU](#).

6.13.2.4 pn_end_threaded()

```
int pn_end_threaded (
    void )
```

Ends threaded execution.

On CoPU, halts the core. On CePU, waits until all other cores are halted. This ends the threaded mode.

See documentation of [Modes](#) for more information.

Returns

Either [PN_SUCCESS](#) or [PN_ERR_MATCH](#).

6.13.2.5 pn_thread_entry()

```
void pn_thread_entry (
    void )
```

Function that has to be called for CoPUs at the end of the startup code.

Marks the entry point of CoPUs into the [Thread Module](#). Necessary for threaded Mode.

The CoPUs are set up to work correctly in threaded Mode.

Execution is only effective on CoPU. Can be executed on CePU, will do nothing then.

Should not be called in a normal application at all. Left in here for startup code writers convenience.

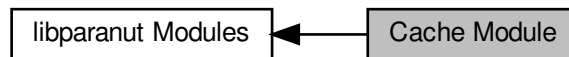
Attention

What comes after this part in the startup code is irrelevant, since the CoPUs that landed there are either put into threaded mode or halted. The function can therefore not return any errors.

6.14 Cache Module

Special functions for controlling the shared ParaNut cache.

Collaboration diagram for Cache Module:



- int [pn_cache_init](#) (void)
Function that has to be called in the main function before any of the functions in the [Cache Module](#) can be called. Initializes some internal data and enables the cache.
- int [pn_cache_enable](#) (void)
Enables instruction and data cache. When changing this, make sure that the [pn_cache_init\(\)](#) function is still correct!
- int [pn_cache_disable](#) (void)
Disables instruction and data cache.
- unsigned long [pn_cache_linesize](#) (void)
Returns the cache line size in bit.
- unsigned long [pn_cache_size](#) (void)
Returns the cache size in Byte.
- int [pn_cache_invalidate](#) (void *addr, unsigned long size)
Invalidates the cache entries containing the given address range.
- int [pn_cache_invalidate_all](#) (void)
Invalidates the whole cache. When changing this, make sure that the [pn_cache_init\(\)](#) function is still correct!
- int [pn_cache_writeback](#) (void *addr, unsigned long size)
Writes back the cache lines that cached the given address range.
- int [pn_cache_writeback_all](#) (void)
Writes whole cache back.
- int [pn_cache_flush](#) (void *addr, unsigned long size)
Combination of [pn_cache_invalidate\(\)](#) and [pn_cache_writeback\(\)](#).
- int [pn_cache_flush_all](#) (void)
Flushes the whole cache.

6.14.1 Detailed Description

Special functions for controlling the shared ParaNut cache.

If you want to see how to add a new cache line size, check [PN_ERR_CACHE_LINESIZE](#).

Todo May need some changes in the future when ParaNut cores get some cache on their own or a proper cache controller is implemented.

Virtual addressing is not implemented in the current ParaNut implementation, but may be featured later on. When the day comes, put in a physical address calculation in the implementation of these functions.

6.14.2 Function Documentation

6.14.2.1 `pn_cache_disable()`

```
int pn_cache_disable (
    void )
```

Disables instruction and data cache.

Attention

Careful here: ParaNut Cache is flushed completely before disabling.

Warning

Atomic memory operations ([Spinlock Module](#)) are not possible on a disabled cache.

Can only be used on CePU.

Returns

Either [PN_SUCCESS](#) or [PN_ERR_COPU](#).

6.14.2.2 `pn_cache_enable()`

```
int pn_cache_enable (
    void )
```

Enables instruction and data cache. When changing this, make sure that the [pn_cache_init\(\)](#) function is still correct!

Attention

Careful here: ParaNut Cache is flushed completely before disabling.

Can only be used on CePU.

Returns

Either [PN_SUCCESS](#) or [PN_ERR_COPU](#).

6.14.2.3 `pn_cache_flush()`

```
int pn_cache_flush (
    void * addr,
    unsigned long size )
```

Combination of [pn_cache_invalidate\(\)](#) and [pn_cache_writeback\(\)](#).

Parameters

in	<i>addr</i>	is the (virtual)start address of the memory you want flushed.
in	<i>size</i>	is the size of the address range you want flushed in byte. The size will always be aligned to the cache line size.

Returns

Either [PN_SUCCESS](#) or [PN_ERR_PARAM](#) if given size is bigger than memory size.

6.14.2.4 pn_cache_flush_all()

```
int pn_cache_flush_all (  
    void )
```

Flushes the whole cache.

Returns

Can only return [PN_SUCCESS](#), is not made void for the sake of making the internal implementation more similar for cache functions.

6.14.2.5 pn_cache_init()

```
int pn_cache_init (  
    void )
```

Function that has to be called in the main function before any of the functions in the [Cache Module](#) can be called. Initializes some internal data and enables the cache.

Can only be used on CePU.

Returns

Either [PN_SUCCESS](#), [PN_ERR_COPU](#), or [PN_CACHE_LINESIZE](#).

6.14.2.6 pn_cache_invalidate()

```
int pn_cache_invalidate (  
    void * addr,  
    unsigned long size )
```

Invalidates the cache entries containing the given address range.

Parameters

in	<i>addr</i>	is the (virtual) start address of the memory you want to invalidate.
in	<i>size</i>	is the size of the address range you want to invalidate in byte. The size will always be aligned to the cache line size.

Returns

Either [PN_SUCCESS](#) or [PN_ERR_PARAM](#) if given size is bigger than memory size.

6.14.2.7 pn_cache_invalidate_all()

```
int pn_cache_invalidate_all (  
    void )
```

Invalidates the whole cache. When changing this, make sure that the [pn_cache_init\(\)](#) function is still correct!

Returns

Can only return [PN_SUCCESS](#), is not made void for the sake of making the internal implementation more similar for cache functions.

6.14.2.8 pn_cache_linesize()

```
unsigned long pn_cache_linesize (  
    void )
```

Returns the cache line size in bit.

Returns

The cache line size in bit. Can not return an error.

6.14.2.9 pn_cache_size()

```
unsigned long pn_cache_size (  
    void )
```

Returns the cache size in Byte.

Returns

The cache size in byte. Can not return an error.

6.14.2.10 pn_cache_writeback()

```
int pn_cache_writeback (  
    void * addr,  
    unsigned long size )
```

Writes back the cache lines that cached the given address range.

Parameters

in	<i>addr</i>	is the (virtual) start address of the memory you want written back.
in	<i>size</i>	is the size of the address range you want written back in byte. The size will always be aligned to the cache line size.

Returns

Either [PN_SUCCESS](#) or [PN_ERR_PARAM](#) if given size is bigger than memory size.

6.14.2.11 pn_cache_writeback_all()

```
int pn_cache_writeback_all (  
    void )
```

Writes whole cache back.

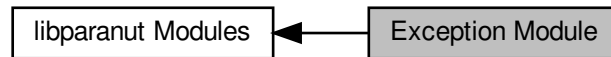
Returns

Can only return [PN_SUCCESS](#), is not made void for the sake of making the internal implementation more similar for cache functions.

6.15 Exception Module

Functions for controlling the handling of interrupts/exceptions.

Collaboration diagram for Exception Module:



- void [pn_exception_init](#) (void)
Initializes libparanut internal exception handling. Interrupts (not exceptions in general!) are disabled after. Should be called before using [pn_exception_set_handler](#)).
- int [pn_exception_set_handler](#) (void(*handler)(unsigned int cause, unsigned int program_counter, unsigned int mtval), unsigned int exception_code)
Set your own exception handler.
- void [pn_ecall](#) (void)
Raises an environment call exception.
- void [pn_interrupt_enable](#) (void)
Enables interrupts only.
- void [pn_interrupt_disable](#) (void)
Disables interrupts only.
- void [pn_progress_mepc](#) (void)
Sets program counter of the register which keeps the exception return adress to next instruction.

6.15.1 Detailed Description

Functions for controlling the handling of interrupts/exceptions.

Why are we calling this exceptions, even though most people would call this an interrupt? Historic reasons. The libparanut was first written for the RISC-V implementation of the ParaNUT, and the RISC-V specification refers to both interrupts and exceptions as exceptions.

Todo All of these functions may be too RISC-V specific to ever use them on another architecture. I have not done more research on this topic yet. If that's the case, we might even need to add architecture specific .c files for this module. I would name them `pn_cache_RV32I.c`, `pn_cache_RV64I.c`, and so on. The Makefile and Documentation will probably need some changes then.

Also, when the ParaNUT has a more advanced mstatus register, the API implementation of the [pn_interrupt_enable\(\)](#) and [pn_interrupt_disable\(\)](#) functions has to change.

6.15.2 Function Documentation

6.15.2.1 `pn_ecall()`

```
void pn_ecall (
    void )
```

Raises an environment call exception.

Can be called without using `pn_exception_init()` first.

6.15.2.2 `pn_exception_set_handler()`

```
int pn_exception_set_handler (
    void(*) (unsigned int cause, unsigned int program_counter, unsigned int mtval)
    handler,
    unsigned int exception_code )
```

Set your own exception handler.

Can be called without using `pn_exception_init()` first, will not work though.

Already does the work of saving away registers, setting program counter etc. for you. You can just hang in what you want to do.

Attention

For exceptions, the register that contains the adress where execution resumes is set to the faulty instruction that threw the exception. For interrupts, it already points to the instruction where execution should resume. Consider this in your handler. If you need to, use `pn_progress_mepc`.

Parameters

in	<i>handler</i>	is a function pointer to your exception handler. Will return an error (PN_ERR_PARAM) if NULL is given.
in	<i>exception_code</i>	is the number that your exception has. You can look up the exception codes in the ParaNut Manual. For interrupts, the value of the most significant bit of the exception code has to be 1. For synchronous exceptions, it has to be 0. This function will return an error (PN_ERR_EXC) if a non implemented value for cause is given.

Returns

Either [PN_SUCCESS](#), [PN_ERR_EXC](#), or [PN_ERR_PARAM](#).

6.15.2.3 `pn_interrupt_disable()`

```
void pn_interrupt_disable (
    void )
```

Disables interrupts only.

Can be called without using `pn_exception_init()` first.

6.15.2.4 `pn_interrupt_enable()`

```
void pn_interrupt_enable (  
    void )
```

Enables interrupts only.

Can be called without using [pn_exception_init\(\)](#) first.

6.15.2.5 `pn_progress_mepc()`

```
void pn_progress_mepc (  
    void )
```

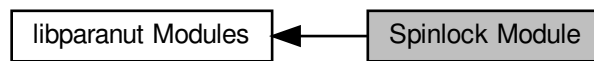
Sets program counter of the register which keeps the exception return adress to next instruction.

Can be called without using [pn_exception_init\(\)](#) first.

6.16 Spinlock Module

Functions and structure used for synchronizing memory access.

Collaboration diagram for Spinlock Module:



Classes

- struct [__pn_spinlock](#)
A synchronization primitive. Use [__pn_spinlock](#) instead of this.
- int [pn_spinlock_init](#) ([__pn_spinlock](#) *spinlock)
Creates a lock.
- int [pn_spinlock_lock](#) ([__pn_spinlock](#) *spinlock)
Waits for a lock. Forever, if it must. Use with caution.
- int [pn_spinlock_trylock](#) ([__pn_spinlock](#) *spinlock)
Tries to acquire a lock. Nonblocking.
- int [pn_spinlock_unlock](#) ([__pn_spinlock](#) *spinlock)
Unlocks a lock.
- int [pn_spinlock_destroy](#) ([__pn_spinlock](#) *spinlock)
Destroys a lock.

6.16.1 Detailed Description

Functions and structure used for synchronizing memory access.

Warning

The functions in here are really kinda performance critical, since it is always important to do as little as possible when you have reserved a memory area. This means that the functions will do extremely little security checks, which means you have to use them the way they are described. Read the detailed descriptions of the functions carefully. Or don't. I'm not the coding police.

Using the spinlock functions in linked Mode (see [Link Module](#) and [Modes](#)) results in undefined behaviour.

If you want to use the Spinlock Module on RISC-V ParaNut, your ParaNut has to support the A Extension and you have to compile your application with the flag mabi=rv32ia. The libparanut [Makefile](#) sets this flag automatically when you chose the [Spinlock Module](#) or one of the Modules that has [Spinlock Module](#) as a dependency.

Return value of functions is always error code, except when stated otherwise in the description of the function.

6.16.2 Function Documentation

6.16.2.1 `pn_spinlock_destroy()`

```
int pn_spinlock_destroy (
    __pn_spinlock * spinlock )
```

Destroys a lock.

Behaviour of this function is undefined if the lock wasn't initialized by `pn_spinlock_init()`.

A destroyed lock can be re-initialized by using `pn_spinlock_init()`.

The lock can either be owned by current hart or unlocked, else the function will fail.

Parameters

<i>spinlock</i>	is a pointer to a lock that we want to destroy. The function will return <code>PN_ERR_PARAM</code> if NULL is passed.
-----------------	---

Returns

Either `PN_SUCCESS`, `PN_ERR_LOCKOCC` or `PN_ERR_PARAM`. If something internally went very wrong, the function is also theoretically able to return `PN_ERR_NOIMP`.

6.16.2.2 `pn_spinlock_init()`

```
int pn_spinlock_init (
    __pn_spinlock * spinlock )
```

Creates a lock.

You allocate the space for the spinlock. Really don't care where you get it from (check `__pn_spinlock` for a recommendation). You pass a reference to this function, and the function initializes it for you. And you shall never touch what's in it.

Afterwards, you can use the other functions in this module on the same lock. Behaviour will always be undefined if you don't call this function first.

The function does not care if your lock was already initialized. It will fail if the CPU did not get the memory reservation (`PN_ERR_LOCKOCC`), which should never happen. This is a very good indicator that something is very wrong with your program.

After initialization, the lock is free. It will not be automatically owned by the hart that initialized it.

Parameters

<i>spinlock</i>	is a pointer to the lock. The function will return <code>PN_ERR_PARAM</code> if NULL is passed.
-----------------	---

Returns

Either [PN_SUCCESS](#), [PN_ERR_PARAM](#), or [PN_ERR_LOCKOCC](#). If something internally went very wrong, the function is also theoretically able to return [PN_ERR_NOIMP](#).

6.16.2.3 pn_spinlock_lock()

```
int pn_spinlock_lock (
    _pn_spinlock * spinlock )
```

Waits for a lock. Forever, if it must. Use with caution.

Behaviour of this function is undefined if the lock wasn't initialized by [pn_spinlock_init\(\)](#).

Warning

The function will be stuck in eternity if the lock is already in the current harts possession, or if someone else owns the lock and forgot to unlock it, or if the lock was destroyed.

Parameters

<i>spinlock</i>	is a pointer to a lock that we want to aquire. The function will return PN_ERR_PARAM if NULL is passed.
-----------------	---

Returns

Either [PN_SUCCESS](#) or [PN_ERR_PARAM](#).

6.16.2.4 pn_spinlock_trylock()

```
int pn_spinlock_trylock (
    _pn_spinlock * spinlock )
```

Tries to acquire a lock. Nonblocking.

Behaviour of this function is undefined if the lock wasn't initialized by [pn_spinlock_init\(\)](#).

Will fail if lock is already owned (no matter by whom), another CPU got the memory reservation, or if lock was destroyed.

Parameters

<i>spinlock</i>	is a pointer to a lock that we want to aquire. The function will return PN_ERR_PARAM if NULL is passed.
-----------------	---

Returns

Either [PN_SUCCESS](#), [PN_ERR_LOCKOCC](#) or [PN_ERR_PARAM](#). If something internally went very wrong, the function is also theoretically able to return [PN_ERR_NOIMP](#).

6.16.2.5 `pn_spinlock_unlock()`

```
int pn_spinlock_unlock (
    _pn_spinlock * spinlock )
```

Unlocks a lock.

Behaviour of this function is undefined if the lock wasn't initialized by [pn_spinlock_init\(\)](#).

Will fail is lock is not owned by current hart ([PN_ERR_PARAM](#)).

Parameters

<i>spinlock</i>	is a pointer to a lock that we want to unlock. The function will return PN_ERR_PARAM if NULL is passed.
-----------------	---

Returns

Either [PN_SUCCESS](#) or [PN_ERR_PARAM](#). If something internally went very wrong, the function is also theoretically able to return [PN_ERR_NOIMP](#).

6.17 libparanut Compile Time Parameters

Group contains defines that inform the application writer how the libparanut was compiled.

Macros

- `#define PN_CACHE_LINESIZE`
Size of a cache line in bit.
- `#define PN_RWIDTH 32`
Register width in bit.
- `#define PN_COMPILE_RAW`
All security checks in libparanut are dropped if this is set to 1.
- `#define PN_WITH_BASE`
libparanut was compiled with [Base Module](#). Also check [Modules of the libparanut](#) for more information.
- `#define PN_WITH_CACHE`
libparanut was compiled with [Cache Module](#). Also check [Modules of the libparanut](#) for more information.
- `#define PN_WITH_LINK`
libparanut was compiled with [Link Module](#). Also check [Modules of the libparanut](#) for more information.
- `#define PN_WITH_THREAD`
libparanut was compiled with [Thread Module](#). Also check [Modules of the libparanut](#) for more information.
- `#define PN_WITH_EXCEPTION`
libparanut was compiled with [Exception Module](#). Also check [Modules of the libparanut](#) for more information.
- `#define PN_WITH_SPINLOCK`
libparanut was compiled with [Spinlock Module](#). Also check [Modules of the libparanut](#) for more information.

6.17.1 Detailed Description

Group contains defines that inform the application writer how the libparanut was compiled.

The libparanut is a very flexible piece of software. Some modules may have been compiled in, others may not. The cache line size could have a fixed value to improve speed, or it could be set on auto which is more compatible. Find out by including [pn_config.h](#) in your application and checking the defines listed in here!

6.17.2 Macro Definition Documentation

6.17.2.1 PN_CACHE_LINESIZE

```
#define PN_CACHE_LINESIZE
```

Size of a cache line in bit.

This decides which assembly file was included during compilation of the [Cache Module](#). If "auto" was chosen, the file that contains functions for all possible cache line sizes is included. This means great binary compatibility, but terribly big code size. When your application is deployed, you should definitely compile and link a version of libparanut with this parameter set to the cache line size you want to use eventually.

Also check documentation of [pn_cache_RV32I_buildscript.py](#).

6.17.2.2 PN_COMPILE_RAW

```
#define PN_COMPILE_RAW
```

All security checks in libparanut are dropped if this is set to 1.

Since functions in this library may be timing critical, you can compile the libparanut with this parameter and disable all the security checks.

While you are developing, it is recommended you don't do this, as the security checks will tell you when you are giving input that does not make sense. You can enable it when you properly tested your system to get optimal performance. Or don't. I mean, it's not like I'm the code police.

6.17.2.3 PN_RWIDTH

```
#define PN_RWIDTH 32
```

Register width in bit.

Was set to 32 bit in this documentation to enable Doxygen to properly write down the typedefs in the Typedefs section of [libparanut.h](#). This should not be of interest at the moment since there is only a 32 bit version of the ParaNut, but it may become relevant in the future.

Chapter 7

Class Documentation

7.1 `__pn_spinlock` Struct Reference

A synchronization primitive. Use `__pn_spinlock` instead of this.

```
#include <libparanut.h>
```

Public Attributes

- `PN_CID owner_ID`

7.1.1 Detailed Description

A synchronization primitive. Use `__pn_spinlock` instead of this.

A simple implementation of a synchronization primitive. Might be used for implementing POSIX Threads later on.

Warning

You are not supposed to touch anything in this struct, which is why you can't see anything about the members in this documentation.

Attention

I highly recommend to not put locks on stack. The reason is that the stack (or at least a part of it) will be copied when putting the ParaNut into threaded Mode (see `pn_begin_threaded()`). In other words, if you put a lock on stack, it will be copied, and the new instances of the lock will be available per core. You should make it static, global or get the memory by allocation.

Warning

Atomic memory operations are not possible on a disabled cache.

Note for other architectures: Check your ABI on how structs are aligned and what data size integer type has. On RV32I GCC, I have no problems with this, since int size is 4 bytes, PN_CID is just int32 (also 4 bytes), and ABI convention says that int is aligned at 4 bytes. I assume that GCC uses the convention, but switching to an obscure compiler could destroy that. So the stuff in here for GCC can just be treated as lying right behind each other in 4 bytes of memory, which is convenient for me. It should work on all other compilers using the ABI convention, but I just didn't want this to go unsaid.

7.1.2 Member Data Documentation

7.1.2.1 owner_ID

[PN_CID](#) `__pn_spinlock::owner_ID`

ID of the CPU that owns me. Can also be negative to represent statuses free (not locked by anyone) or dead (not initialized).

The documentation for this struct was generated from the following file:

- [libparanut.h](#)

Chapter 8

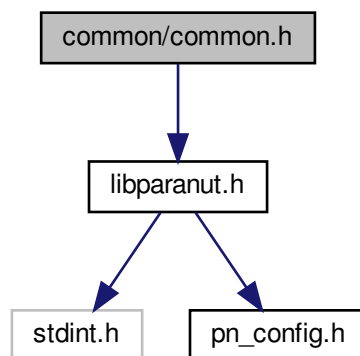
File Documentation

8.1 common/common.h File Reference

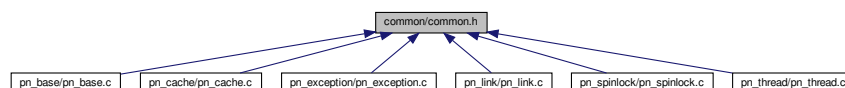
Contains architecture independent internal prototypes and defines needed in [libparanut Modules](#).

```
#include "libparanut.h"
```

Include dependency graph for common.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define COPU_CHECK` if (`coreid_as()` != 0) return `PN_ERR_COPU`;
Checks if we are running on CoPU.
- `#define ALL_HALTED` if (`read_PNCE_as()` > 1) return `PN_ERR_MATCH`;
Checks if all CoPUs are halted.
- `#define BEGIN_THREADED_LINKED_SEC_CHECK`
Performs security checks in `pn_begin_linked()` and `pn_begin_threaded()`.
- `#define BEGIN_THREADED_LINKED_SEC_CHECK_M`
Performs security checks in `pn_begin_linked_m()` and `pn_begin_threaded_m()`.
- `#define CONVERT_NUMC_TO_MASK`
Converts a given number of cores to a minimal bitmask.
- `#define TERMNL "\n\r"`
Terminal newline.

Functions

- void `halt_as` (void)
Halts the core this is executed on.
- `PN_NUMC numcores_as` (void)
Get the number of cores in your system.
- `PN_CID coreid_as` (void)
Get the core ID.
- `PN_CMSK read_PNCE_as` (void)
Reads ParaNut CPU enable register.
- void `enable_CPU_as` (`PN_CMSK` coremask)
Enables the cores marked in the bitmask, disables the other ones.
- `PN_CMSK m2cap_as` (void)
Get the register where Mode 2 capability cores are marked.
- int `simulation_as` (void)
Checks if we run in simulation instead of real hardware.

- `PN_CMSK read_PNLM_as` (void)
Reads PNLM register.

Variables

- int `sp_loc`
Stack Pointer location, used by `set_linked_as()` and `set_threaded_as()`.
- int `tp_loc`
Thread Pointer location, used in `Thread Module`.

8.1.1 Detailed Description

Contains architecture independent internal prototypes and defines needed in [libparanut Modules](#).

Is included by all modules and includes the [libparanut.h](#) itself, thereby is a "common layer" for all modules.

```

1  /*
2  * Copyright 2019-2020 Anna Pfuetzner (<annakerstin.pfuetzner@gmail.com>)
3  *
4  * Redistribution and use in source and binary forms, with or without
5  * modification, are permitted provided that the following conditions are met:
6  *
7  * 1. Redistributions of source code must retain the above copyright notice,
8  * this list of conditions and the following disclaimer.
9  *
10 * 2. Redistributions in binary form must reproduce the above copyright notice,
11 * this list of conditions and the following disclaimer in the documentation
12 * and/or other materials provided with the distribution.
13 *
14 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
15 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
16 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
17 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
18 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
19 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
20 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
21 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
22 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
23 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
24 * POSSIBILITY OF SUCH DAMAGE.
25 */
26
40 /*Includes*****
41
42 #include "libparanut.h"
43
44 /*Commonly Used Defines*****
45
69 #define COPU_CHECK    if (coreid_as() != 0) return PN_ERR_COPU;
70
79 #define ALL_HALTED    if (read_PNCE_as() > 1) return PN_ERR_MATCH;
80
89 #define BEGIN_THREADED_LINKED_SEC_CHECK
90     COPU_CHECK
91     ALL_HALTED
92     if ((numcores > numcores_as()) || (numcores <= 1))
93         return PN_ERR_PARAM;
94     if (numcores >= PN_RWIDTH)
95         return PN_ERR_NOIMP;
96
103 #define BEGIN_THREADED_LINKED_SEC_CHECK_M
104     COPU_CHECK
105     ALL_HALTED
106     if (!(coremask & 0b1))
107         return PN_ERR_PARAM;
108
109
114 #define CONVERT_NUMC_TO_MASK
115     for (i = 2; i < numcores; i++)
116         coremask |= (1 << i);
117
123 #define TERMNL        "\n\r"
124
133 /*Global Variables*****
134
156 int                sp_loc;
157
163 int                tp_loc;
164
173 /*Commonly Used Assembly Functions*****
174
175 extern void        halt_as(void);
176 extern PN_NUMC     numcores_as(void);
177 extern PN_CID      coreid_as(void);
178 extern PN_CMSK     read_PNCE_as(void);
179 extern PN_CMSK     read_PNLM_as(void);
180 extern void        enable_CPU_as(PN_CMSK coremask);
181 extern PN_CMSK     m2cap_as(void);
182 extern int         simulation_as(void);
183
184 /*EOF*****

```

8.2 common/common_RV32I.S File Reference

Contains RV32I assembly implementations of assembly functions called in all modules.

Functions

- void `halt_as` (void)
Halts the core this is executed on.
- `PN_NUMC numcores_as` (void)
Get the number of cores in your system.
- `PN_CID coreid_as` (void)
Get the core ID.
- `PN_CMSK read_PNCE_as` (void)
Reads ParaNut CPU enable register.
- void `enable_CPU_as` (`PN_CMSK` coremask)
Enables the cores marked in the bitmask, disables the other ones.
- `PN_CMSK m2cap_as` (void)
Get the register where Mode 2 capability cores are marked.
- int `simulation_as` (void)
Checks if we run in simulation instead of real hardware.

8.2.1 Detailed Description

Contains RV32I assembly implementations of assembly functions called in all modules.

```

1 /*
2  * Copyright 2019-2020 Anna Pfuetzner (<annakerstin.pfuetzner@gmail.com>)
3  *
4  * Redistribution and use in source and binary forms, with or without
5  * modification, are permitted provided that the following conditions are met:
6  *
7  * 1. Redistributions of source code must retain the above copyright notice,
8  * this list of conditions and the following disclaimer.
9  *
10 * 2. Redistributions in binary form must reproduce the above copyright notice,
11 * this list of conditions and the following disclaimer in the documentation
12 * and/or other materials provided with the distribution.
13 *
14 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
15 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
16 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
17 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
18 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
19 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
20 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
21 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
22 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
23 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
24 * POSSIBILITY OF SUCH DAMAGE.
25 */
26
27
28 /*
29  * Put in here so Doxygen will know that it is implemented in this file.
30  * Sadly, Doxygen has no built in assembly interpreter, so we are stuck with
31  * this.
32  */
33
34 #ifdef DOXYGEN
35
36 void halt_as(void) {}

```

```

61
62  PN_NUMC numcores_as(void) {}
63
64  PN_CID coreid_as(void) {}
65
66  PN_CMSK read_PNCE_as(void) {}
67
68  void enable_CPU_as(PN_CMSK coremask) {}
69
70  PN_CMSK m2cap_as(void) {}
71
72  int simulation_as(void) {}
73
74 #endif /* DOXYGEN */
75
76 /*Header*****
77
78 #ifndef DOXYGEN
79
80      /* enter text section
81      /* align Code to 2^2 Bytes
82
83  /* declare labels in here to be global */
84  .globl  halt_as
85  .globl  numcores_as
86  .globl  coreid_as
87  .globl  read_PNCE_as
88  .globl  enable_CPU_as
89  .globl  m2cap_as
90  .globl  simulation_as
91
92  /* ParaNut Custom Registers and Instructions */
93  #include "custom_RV32I.S"
94
95  /*Functions*****
96
97  halt_as:
98
99      fence
100      /* make sure memory is synchronized
101      HALT
102      /* halt executing core
103      ret
104      /* return
105
106  /*-----*/
107
108  numcores_as:
109
110      csrr a0,  pncpus
111      /* put ncpus in return value
112      ret
113      /* return
114
115  /*-----*/
116
117  coreid_as:
118
119      csrr a0,  mhartid
120      /* put mhartid in return value
121      ret
122      /* return
123
124  /*-----*/
125
126  read_PNCE_as:
127
128      csrr a0,  pnce
129      /* read pnce, store in return value
130      ret
131      /* return
132
133  /*-----*/
134
135  enable_CPU_as:
136
137      fence
138      /* synchronize memory before enabling
139      csrw pnce, a0
140      /* sets pnce to passed bitmask
141      ret
142      /* return
143
144  /*-----*/
145
146  m2cap_as:
147
148      csrr a0,  pnm2cap
149      /* put pnm2cap in return value
150      ret
151      /* return
152
153  /*-----*/
154
155  simulation_as:
156
157      la  t0,  _thread_data
158      /* get address of thread data
159      li  a0,  0
160      /* clear a0
161      lb  a0,  0(t0)
162      /* load first byte from thread data
163      xori a0,  a0,  'S'
164      /* check if it is an 'S'
165      beqz a0,  is_sim
166      /* if a0 is 0, it is a simulation

```

```

198
199 is_not_sim:                                /* we got here if we run on hardware */
200     li    a0,    0                          /* put zero as return value */
201     ret                                /* return */
202
203 is_sim:                                    /* we got here if we run in simulation */
204     li    a0,    1                          /* put non-zero as return value */
205     ret                                /* return */
206
207 #endif /* !DOXYGEN */
208
209 /*EOF*****

```

8.3 common/custom_RV32I.S File Reference

Contains defines used in assembly functions of all modules.

Macros

- #define [pngrpsel](#) 0x7C0
ParaNut CPU group select register.
- #define [pnce](#) 0x7C1
ParaNut CPU enable register.
- #define [pnlm](#) 0x7C2
ParaNut CPU linked mode register.
- #define [pnxsel](#) 0x7C3
ParaNut CoPU exception select register.
- #define [pncache](#) 0x7C4
ParaNut Cache control register.
- #define [pncpus](#) 0xFC0
ParaNut number of CPUs register.
- #define [pnm2cap](#) 0xFC1
ParaNut CPU capabilities register.
- #define [pnx](#) 0xFC2
ParaNut CoPU exception pending register.
- #define [pncause](#) 0xFC3
ParaNut CoPU trap cause ID register.
- #define [pnepc](#) 0xFC4
ParaNut CoPU exception program counter register.
- #define [pncacheinfo](#) 0xFC5
ParaNut cache information register.
- #define [pncachesets](#) 0xFC6
ParaNut number of cache sets register.
- #define [pnclockinfo](#) 0xFC7
ParaNut clock speed information register.
- #define [pnmemsize](#) 0xFC8
ParaNut memory size register.
- #define [mtval](#) 0x343
Machine Trap Value Register.

- `#define HALT .word 0x0000000B`
Halts the core which this instruction is executed on.
- `#define CINV(regnum, offset) .word (0x100B | ((regnum) << 15) | (((offset) & 0xfff) << 20))`
Cache invalidate.
- `#define CWB(regnum, offset) .word (0x200B | ((regnum) << 15) | (((offset) & 0xfff) << 20))`
Cache writeback.

8.3.1 Detailed Description

Contains defines used in assembly functions of all modules.

```

1 /*
2  * Copyright 2019-2020 Anna Pfuetzner (<annakerstin.pfuetzner@gmail.com>)
3  *
4  * Redistribution and use in source and binary forms, with or without
5  * modification, are permitted provided that the following conditions are met:
6  *
7  * 1. Redistributions of source code must retain the above copyright notice,
8  * this list of conditions and the following disclaimer.
9  *
10 * 2. Redistributions in binary form must reproduce the above copyright notice,
11 * this list of conditions and the following disclaimer in the documentation
12 * and/or other materials provided with the distribution.
13 *
14 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
15 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
16 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
17 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
18 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
19 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
20 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
21 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
22 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
23 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
24 * POSSIBILITY OF SUCH DAMAGE.
25 */
26
27 /*ParaNut Control and Status Registers*****//
28
29 #define pngrpsel      0x7C0
30
31 #define pnice         0x7C1
32
33 #define pnlim         0x7C2
34
35 #define pnxsel        0x7C3
36
37 #define pncache       0x7C4
38
39 #define pncpus        0xFC0
40
41 #define pnm2cap       0xFC1
42
43 #define pnx           0xFC2
44
45 #define pncause       0xFC3
46
47 #define pnepc         0xFC4
48
49 #define pncacheinfo   0xFC5
50
51 #define pncachesets   0xFC6
52
53 #define pnclockinfo   0xFC7
54
55 #define pnmemsize     0xFC8
56
57 #define mtval         0x343
58
59 /*ParaNut Custom Instructions*****//
60
61 #define HALT          .word 0x0000000B
62

```

```

216 #define CINV(regnum, offset) \
217     .word (0x100B | ((regnum) << 15) | (((offset) & 0xfff) << 20)) \
218
230 #define CWB(regnum, offset) \
231     .word (0x200B | ((regnum) << 15) | (((offset) & 0xfff) << 20)) \
232
244 #define CFLUSH(regnum, offset) \
245     .word (0x300B | ((regnum) << 15) | (((offset) & 0xfff) << 20)) \
246
255 /*EOF*****

```

8.4 libparanut.h File Reference

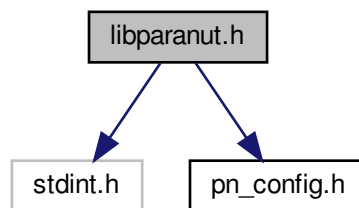
API of the libparanut.

```

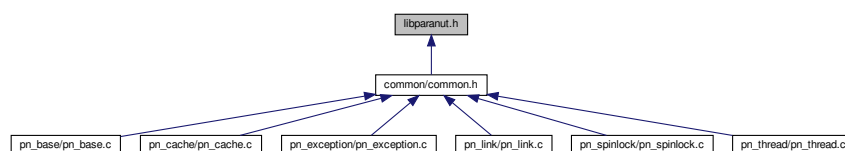
#include <stdint.h>
#include "pn_config.h"

```

Include dependency graph for libparanut.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct [__pn_spinlock](#)

A synchronization primitive. Use [_pn_spinlock](#) instead of this.

Macros

- `#define PN_SUCCESS 0`
Successful execution.
 - `#define PN_ERR_PARAM (-1)`
Parameter error.
 - `#define PN_ERR_NOIMP (-2)`
Function not implemented.
 - `#define PN_ERR_COPU (-3)`
CoPU error.
 - `#define PN_ERR_MATCH (-4)`
Mode begin and end matching error.
 - `#define PN_ERR_LOCKOCC (-5)`
Lock occupied error.
 - `#define PN_ERR_CACHE_LINESIZE (-6)`
Weird cache line size error.
 - `#define PN_ERR_EXC (-8)`
Exception code not implemented error.
-
- `#define PN_M0 0x0U`
Mode 0 (halted Mode).
 - `#define PN_M1 0x1U`
Mode 1 (linked Mode).
 - `#define PN_M2 0x2U`
Mode 2 (unlinked or threaded Mode).
 - `#define PN_M3 0x3U`
Mode 3 (autonomous Mode).

Typedefs

- `typedef struct __pn_spinlock _pn_spinlock`
Renaming of struct `__pn_spinlock` for your convenience.
-
- `typedef int32_t PN_CID`
Signed type that can be used to address any core in this architecture.
 - `typedef int32_t PN_NUMC`
Signed type that can hold the maximum number of cores in this architecture.
 - `typedef uint32_t PN_CMSK`
Unsigned type that can act as a core mask.
 - `typedef int32_t PN_NUMG`
Signed type that can be used to address any group in this architecture.

Functions

- [PN_NUMC pn_numcores](#) (void)
Get the number of cores in your system.
 - [PN_CMSK pn_m2cap](#) (void)
Check which cores are capable of Mode 2 operation.
 - [PN_CMSK pn_m2cap_g](#) (PN_NUMG groupnum)
Check which cores are capable of Mode 2 operation.
 - [PN_CMSK pn_m3cap](#) (void)
Check which cores are capable of Mode 3 operation.
 - [PN_CMSK pn_m3cap_g](#) (PN_NUMG groupnum)
Check which cores are capable of Mode 3 operation.
 - [PN_CID pn_coreid](#) (void)
Get the ID of the core that this function is executed on.
 - [PN_CID pn_coreid_g](#) (PN_NUMG *groupnum)
Get the ID and group number of the core that this code is running on.
 - void [pn_halt](#) (void)
Halt whatever core the function is executed on.
 - int [pn_halt_CoPU](#) (PN_CID coreid)
Halts a CoPU.
 - int [pn_halt_CoPU_m](#) (PN_CMSK coremask)
Halts one or more CoPUs.
 - int [pn_halt_CoPU_gm](#) (PN_CMSK *coremask_array, PN_NUMG array_size)
Halts the CoPUs specified in the coremask_array.
 - long long int [pn_time_ns](#) (void)
Returns system time in ns. Does not care for overflow.
 - int [pn_simulation](#) (void)
Checks if we run in simulation instead of real hardware.
-
- [PN_CID pn_begin_linked](#) (PN_NUMC numcores)
Links a given number of CoPUs to the CePU.
 - [PN_CID pn_begin_linked_m](#) (PN_CMSK coremask)
Links the CPUs specified in the coremask.
 - [PN_CID pn_begin_linked_gm](#) (PN_CMSK *coremask_array, PN_NUMG array_size)
Links the CPUs specified in the coremask_array.
 - int [pn_end_linked](#) (void)
Ends linked execution.
-
- void [pn_thread_entry](#) (void)
Function that has to be called for CoPUs at the end of the startup code.
 - [PN_CID pn_begin_threaded](#) (PN_NUMC numcores)
Puts numcores CPUs in threaded mode.

- [PN_CID pn_begin_threaded_m](#) ([PN_CMSK](#) coremask)
Puts the CPUs specified in the coremask in threaded mode.
- [PN_CID pn_begin_threaded_gm](#) ([PN_CMSK](#) *coremask_array, [PN_NUMG](#) array_size)
Puts the CPUs specified in the coremask_array in threaded mode.
- int [pn_end_threaded](#) (void)
Ends threaded execution.

- int [pn_cache_init](#) (void)
Function that has to be called in the main function before any of the functions in the [Cache Module](#) can be called. Initializes some internal data and enables the cache.
- int [pn_cache_enable](#) (void)
Enables instruction and data cache. When changing this, make sure that the [pn_cache_init\(\)](#) function is still correct!
- int [pn_cache_disable](#) (void)
Disables instruction and data cache.
- unsigned long [pn_cache_linesize](#) (void)
Returns the cache line size in bit.
- unsigned long [pn_cache_size](#) (void)
Returns the cache size in Byte.
- int [pn_cache_invalidate](#) (void *addr, unsigned long size)
Invalidates the cache entries containing the given address range.
- int [pn_cache_invalidate_all](#) (void)
Invalidates the whole cache. When changing this, make sure that the [pn_cache_init\(\)](#) function is still correct!
- int [pn_cache_writeback](#) (void *addr, unsigned long size)
Writes back the cache lines that cached the given address range.
- int [pn_cache_writeback_all](#) (void)
Writes whole cache back.
- int [pn_cache_flush](#) (void *addr, unsigned long size)
Combination of [pn_cache_invalidate\(\)](#) and [pn_cache_writeback\(\)](#).
- int [pn_cache_flush_all](#) (void)
Flushes the whole cache.

- void [pn_exception_init](#) (void)
Initializes libparanut internal exception handling. Interrupts (not exceptions in general!) are disabled after. Should be called before using [pn_exception_set_handler\(\)](#).
- int [pn_exception_set_handler](#) (void(*handler)(unsigned int cause, unsigned int program_counter, unsigned int mtval), unsigned int exception_code)
Set your own exception handler.
- void [pn_ecall](#) (void)
Raises an environment call exception.
- void [pn_interrupt_enable](#) (void)
Enables interrupts only.
- void [pn_interrupt_disable](#) (void)
Disables interrupts only.
- void [pn_progress_mepc](#) (void)
Sets program counter of the register which keeps the exception return adress to next instruction.

- int [pn_spinlock_init](#) ([_pn_spinlock](#) *spinlock)
Creates a lock.
- int [pn_spinlock_lock](#) ([_pn_spinlock](#) *spinlock)
Waits for a lock. Forever, if it must. Use with caution.
- int [pn_spinlock_trylock](#) ([_pn_spinlock](#) *spinlock)
Tries to acquire a lock. Nonblocking.
- int [pn_spinlock_unlock](#) ([_pn_spinlock](#) *spinlock)
Unlocks a lock.
- int [pn_spinlock_destroy](#) ([_pn_spinlock](#) *spinlock)
Destroys a lock.

8.4.1 Detailed Description

API of the libparanut.

8.5 Makefile File Reference

Makefile of the libparanut.

8.5.1 Detailed Description

Makefile of the libparanut.

This Makefile is supposed to make (ha!) the compilation of libparanut more handy. To check out exactly how this works, see the section [HOWTO](#) in the mainpage!

Todo The Makefile might not be compliant with any other tool than GNU make. Compatibility with other tools should be tested and listed (maybe in the mainpage). The [HOWTO](#) section needs to be changed, then.

Attention

Why am I calling a Python Skript when I'm building `pn_cache_RV32I_nnn.S` (example: [pn_cache_RV32I_↔auto.S](#)) instead of just writing the file myself? And why does that cache target exist? Click [pn_cache_RV32I_↔I_buildscript.py](#) to find out!

```

1 #####
2 # Copyright 2019-2020 Anna Pfuetzner (<annakerstin.pfuetzner@gmail.com>)
3 #
4 # Redistribution and use in source and binary forms, with or without modification, are permitted
5 # provided that the following conditions are met:
6 #
7 # 1. Redistributions of source code must retain the above copyright notice, this list of conditions
8 # and the following disclaimer.
9 #
10 # 2. Redistributions in binary form must reproduce the above copyright notice, this list of
11 # conditions and the following disclaimer in the documentation and/or other materials provided with
12 # the distribution.
13 #
14 # THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR
15 # IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
16 # FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR

```

```

17 # CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
18 # DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
19 # DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
20 # IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
21 # OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
22 #####
23
24 #System Configuration#####
25
26 # Call clean when changing these!
27
28 # System Parameters
29 PN_CACHE_LINESIZE      = auto      # Cache Linesize in Bits, Default: auto
30                          # auto means all power of two linesizes from 32 to 2048
31 PN_RWIDTH              = 32        # Register Width in Bits, Default: 32
32
33 # Switch for raw compilation (optimized for performance, will drop security checks)
34 # Default: 0
35 PN_COMPILE_RAW          = 0
36
37 # Chose modules - Set to 0 to switch off
38 # Default: 1
39 PN_WITH_BASE            = 1
40 PN_WITH_CACHE           = 1
41 PN_WITH_LINK            = 1
42 PN_WITH_THREAD          = 1
43 PN_WITH_EXCEPTION       = 1
44 PN_WITH_SPINLOCK        = 1
45
46 #Compiler Configuration#####
47
48 # Instruction Set Architecture
49 # Currently available:
50 #   1. RISCv 32 bit - Set to RV32I
51 # Don't forget to change compiler when touching this!
52 PN_ISA                  = RV32I
53
54 # Compiler
55 # Currently available:
56 #   1. GCC - Available GCC for chosen ISA
57 PN_COMPILER              = GCC
58
59 # Compile with Debug Symbols
60 # Default: 1
61 PN_DEBUG                = 1
62
63 #Compiler and Assembler Flags#####
64
65 ifeq ($(PN_ISA),RV32I)
66     ifeq ($(PN_COMPILER),GCC)
67
68         # Set the list of system parameters together
69         PN_CONFIG_DEFINES = -D PN_CACHE_LINESIZE=$(PN_CACHE_LINESIZE) -D PN_RWIDTH=$(PN_RWIDTH)
70         PN_CONFIG_DEFINES += -D PN_JOBQUEUE_SIZE=$(PN_JOBQUEUE_SIZE)
71
72         # Put in define for raw compilation if it was set before
73         ifeq ($(PN_COMPILE_RAW),1)
74             PN_CONFIG_DEFINES += -D PN_COMPILE_RAW
75         endif
76
77         # Actual Compiler and Assembler
78         CC = riscv32-unknown-elf-gcc
79         ASM = riscv32-unknown-elf-as
80
81         # Compiler Flags
82         CFLAGS = -c -ansi -O3 -Wall -Werror -I./ -I./common $(PN_CONFIG_DEFINES)
83         CFLAGS += -mabi=ilp32
84
85         ifeq ($(PN_WITH_SPINLOCK),0)
86             ifeq ($(PN_WITH_LINK),0)
87                 CFLAGS += -march=rv32i
88             else
89                 CFLAGS += -march=rv32im
90             endif
91         else
92             ifeq ($(PN_WITH_LINK),0)
93                 CFLAGS += -march=rv32ia
94             else
95                 CFLAGS += -march=rv32ima
96             endif
97         endif
98     endif
99
100     ifeq ($(PN_DEBUG),1)
101         CFLAGS += -g
102     endif
103

```

```

104     CFLAGS                += -o
105
106     # Assembler Flags
107     AFLAGS                 = --fatal-warnings -mabi=ilp32 -I./common
108
109     ifeq ($(PN_WITH_SPINLOCK),0)
110         ifeq ($(PN_WITH_LINK),0)
111             AFLAGS          += -march=rv32i
112         else
113             AFLAGS          += -march=rv32im
114         endif
115     else
116         ifeq ($(PN_WITH_LINK),0)
117             AFLAGS          += -march=rv32ia
118         else
119             AFLAGS          += -march=rv32ima
120         endif
121     endif
122
123     ifeq ($(PN_DEBUG),1)
124         AFLAGS              += -g
125     endif
126
127     AFLAGS                  += -o
128
129     else
130
131         $(error No valid compiler set for the ISA that you chose.)
132
133     endif
134
135 else
136
137     $(error No valid ISA set.)
138
139 endif
140
141 #Lists#####
142
143 # Assemble object list so we know what objects to build
144 OBJECT_LIST =
145
146 ifeq ($(PN_WITH_BASE),1)
147     PN_CONFIG_DEFINES    += -D PN_WITH_BASE
148     BPATH                = ./pn_base/pn_base
149     OBJECT_LIST          += $(BPATH).o $(BPATH)_$(PN_ISA).o
150 endif
151
152 ifeq ($(PN_WITH_LINK),1)
153     PN_CONFIG_DEFINES    += -D PN_WITH_LINK
154     LPATH                = ./pn_link/pn_link
155     OBJECT_LIST          += $(LPATH).o $(LPATH)_$(PN_ISA).o
156 endif
157
158 ifeq ($(PN_WITH_THREAD),1)
159     PN_CONFIG_DEFINES    += -D PN_WITH_THREAD
160     TPATH                = ./pn_thread/pn_thread
161     OBJECT_LIST          += $(TPATH).o $(TPATH)_$(PN_ISA).o
162 endif
163
164 ifeq ($(PN_WITH_CACHE),1)
165     PN_CONFIG_DEFINES    += -D PN_WITH_CACHE
166     CPATH                = ./pn_cache/pn_cache
167     OBJECT_LIST          += $(CPATH).o $(CPATH)_$(PN_ISA)_$(strip $(PN_CACHE_LINESIZE)).o
168 endif
169
170 ifeq ($(PN_WITH_EXCEPTION),1)
171     PN_CONFIG_DEFINES    += -D PN_WITH_EXCEPTION
172     EPATH                = ./pn_exception/pn_exception
173     OBJECT_LIST          += $(EPATH).o $(EPATH)_$(PN_ISA).o
174 endif
175
176 ifeq ($(PN_WITH_SPINLOCK),1)
177     PN_CONFIG_DEFINES    += -D PN_WITH_SPINLOCK
178     SPATH                = ./pn_spinlock/pn_spinlock
179     OBJECT_LIST          += $(SPATH).o $(SPATH)_$(PN_ISA).o
180 endif
181
182 # Check if object list is empty because it would not make any sense to build the library then
183 ifeq ($(strip $(OBJECT_LIST)),)
184     $(error No modules enabled.)
185 endif
186
187 # Add the objects for the common part
188 OBJECT_LIST              += ./common/common_$(PN_ISA).o
189
190

```



```

191 # Everything in this list has to be put into INSTALL directory
192 INSTALL_LIST      = ./INSTALL/libparanut.a ./INSTALL/libparanut.h
193 INSTALL_LIST      += ./INSTALL/pn_config.h
194
195
196 #Target Magic#####
197
198 # Creates everything for usage of libparanut
199 all: $(INSTALL_LIST)
200
201 # Creates all cache files for RV32I anew in case of changes in buildscript.
202 # Important for development only.
203 cache: ./pn_cache/pn_cache_RV32I_buildscript.py
204     python $< 32
205     python $< 64
206     python $< 128
207     python $< 256
208     python $< 512
209     python $< 1024
210     python $< 2048
211     python $< auto
212
213 # Generates library from object files
214 ./INSTALL/libparanut.a: $(OBJECT_LIST)
215     mkdir -p INSTALL
216     riscv32-unknown-elf-ar cr $@ $(OBJECT_LIST)
217
218 # Builds objects from C source code
219 %.o: %.c $(CC) libparanut.h pn_config.h
220     $(CC) $(CFLAGS) $@ $<;
221
222 # Cache Module Assembly Code for RV32I is generated at compile time
223 ./pn_cache/pn_cache_RV32I_*.S: ./pn_cache/pn_cache_RV32I_buildscript.py
224     python $< $(strip $(PN_CACHE_LINESIZE))
225
226 # Put libparanut.h (API) into INSTALL directory
227 ./INSTALL/libparanut.h: libparanut.h pn_config.h
228     mkdir -p INSTALL
229     cp libparanut.h ./INSTALL/
230
231 # Generate pn_config header from system parameters - Only dependent on Makefile itself
232 SHELL = /bin/bash
233 ./INSTALL/pn_config.h:
234     mkdir -p INSTALL
235     touch ./INSTALL/pn_config.h
236     echo "/* Automatically generated file. See Makefile. No edits here! */" > ./INSTALL/pn_config.h
237     echo "#define PN_CACHE_LINESIZE $(PN_CACHE_LINESIZE)" >> ./INSTALL/pn_config.h
238     echo "#define PN_RWIDTH $(PN_RWIDTH)" >> ./INSTALL/pn_config.h
239     if ((${PN_COMPILE_RAW} == 1)); \
240     then echo "#define PN_COMPILE_RAW" >> ./INSTALL/pn_config.h; fi
241     if ((${PN_WITH_BASE} == 1)); \
242     then echo "#define PN_WITH_BASE" >> ./INSTALL/pn_config.h; fi
243     if ((${PN_WITH_CACHE} == 1)); \
244     then echo "#define PN_WITH_CACHE" >> ./INSTALL/pn_config.h; fi
245     if ((${PN_WITH_EXCEPTION} == 1)); \
246     then echo "#define PN_WITH_EXCEPTION" >> ./INSTALL/pn_config.h; fi
247     if ((${PN_WITH_LINK} == 1)); \
248     then echo "#define PN_WITH_LINK" >> ./INSTALL/pn_config.h; fi
249     if ((${PN_WITH_THREAD} == 1)); \
250     then echo "#define PN_WITH_THREAD" >> ./INSTALL/pn_config.h; fi
251     if ((${PN_WITH_SPINLOCK} == 1)); \
252     then echo "#define PN_WITH_SPINLOCK" >> ./INSTALL/pn_config.h; fi
253
254 # The other INSTALL targets just have to be copied from common directory
255 ./INSTALL/%: ./common/%
256     mkdir -p INSTALL
257     cp $< $@ ./INSTALL/
258
259 # Removes library, object files, and auto generated files
260 clean:
261     find . -name '*.o' -delete
262     if [ -d INSTALL ]; then rm -r INSTALL; fi
263
264 #EOF#####

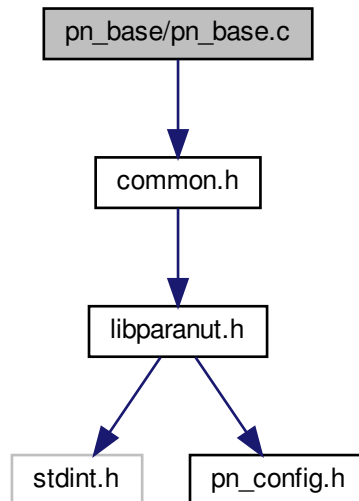
```

8.6 pn_base/pn_base.c File Reference

Contains architecture independent implementations of the [Base Module](#) functions.

```
#include "common.h"
```

Include dependency graph for pn_base.c:



Functions

- [PN_NUMC pn_numcores](#) (void)
Get the number of cores in your system.
- [PN_CMSK pn_m2cap](#) (void)
Check which cores are capable of Mode 2 operation.
- [PN_CMSK pn_m2cap_g](#) (PN_NUMG groupnum)
Check which cores are capable of Mode 2 operation.
- [PN_CMSK pn_m3cap](#) (void)
Check which cores are capable of Mode 3 operation.
- [PN_CMSK pn_m3cap_g](#) (PN_NUMG groupnum)
Check which cores are capable of Mode 3 operation.
- [PN_CID pn_coreid](#) (void)
Get the ID of the core that this function is executed on.
- [PN_CID pn_coreid_g](#) (PN_NUMG *groupnum)
Get the ID and group number of the core that this code is running on.
- void [pn_halt](#) (void)
Halt whatever core the function is executed on.
- int [pn_halt_CoPU](#) (PN_CID coreid)
Halts a CoPU.
- int [pn_halt_CoPU_m](#) (PN_CMSK coremask)
Halts one or more CoPUs.
- int [pn_halt_CoPU_gm](#) (PN_CMSK *coremask_array, PN_NUMG array_size)
Halts the CoPUs specified in the coremask_array.
- long long int [pn_time_ns](#) (void)
Returns system time in ns. Does not care for overflow.

- int [pn_simulation](#) (void)
Checks if we run in simulation instead of real hardware.
- unsigned long [ticks_as](#) (void)
Reads machine cycle counter.
- int [freq_as](#) (void)
Reads pnclockinfo register which contains clock speed.

8.6.1 Detailed Description

Contains architecture independent implementations of the [Base Module](#) functions.

Functions with suffix `_as` are architecture specific and therefore implemented in the `pn_base_$(PN_ISA).S` file in the same directory.

8.7 pn_base/pn_base_RV32I.S File Reference

Contains RV32I assembly implementations of assembly functions called in [pn_base.c](#).

Functions

- unsigned long [ticks_as](#) (void)
Reads machine cycle counter.
- int [freq_as](#) (void)
Reads pnclockinfo register which contains clock speed.

8.7.1 Detailed Description

Contains RV32I assembly implementations of assembly functions called in [pn_base.c](#).

```

1 /*
2  * Copyright 2019-2020 Anna Pfuetzner (<annakerstin.pfuetzner@gmail.com>)
3  *
4  * Redistribution and use in source and binary forms, with or without
5  * modification, are permitted provided that the following conditions are met:
6  *
7  * 1. Redistributions of source code must retain the above copyright notice,
8  * this list of conditions and the following disclaimer.
9  *
10 * 2. Redistributions in binary form must reproduce the above copyright notice,
11 * this list of conditions and the following disclaimer in the documentation
12 * and/or other materials provided with the distribution.
13 *
14 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
15 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
16 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
17 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
18 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR

```

```

19  * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
20  * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
21  * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
22  * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
23  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
24  * POSSIBILITY OF SUCH DAMAGE.
25  */
26
38  /*
39  * Put in here so Doxygen will know that it is implemented in this file.
40  * Sadly, Doxygen has no built in assembly interpreter, so we are stuck with
41  * this.
42  */
43
44  #ifndef DOXYGEN
45
46      unsigned long ticks_as(void) {}
47
48      int freq_as(void) {}
49  #endif /* DOXYGEN */
50
51  /*Header*****
52
53  #ifndef DOXYGEN
54
55      .text                /* enter text section                */
56      .align 2             /* align Code to 2^2 Bytes            */
57
58      /* declare labels in here to be global */
59      .globl  ticks_as
60      .globl  freq_as
61
62      /* ParaNut Custom Registers and Instructions */
63      #include "custom_RV32I.S"
64
65      /*Functions*****
66
67      ticks_as:
68
69          csrr a0, mcycle                /* put mcycle in return value        */
70          csrr a1, mcycleh               /* put mcycleh in second return value */
71          ret                            /* return                            */
72
73      /*-----*/
74
75      freq_as:
76
77          csrr a0, pnclockinfo           /* put pnclockinfo in return value    */
78          ret                            /* return                            */
79
80  #endif /* !DOXYGEN */
81
82  /*EOF*****

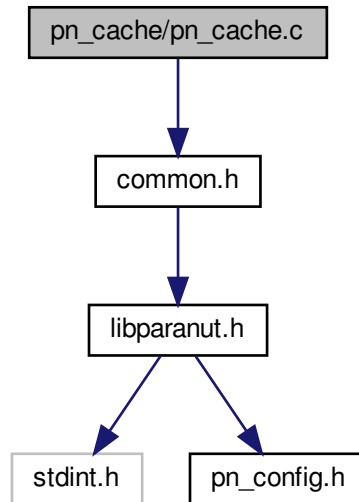
```

8.8 pn_cache/pn_cache.c File Reference

Contains architecture independent implementations of the [Cache Module](#) functions.

```
#include "common.h"
```

Include dependency graph for pn_cache.c:



Macros

- `#define BULK_LINES 32`
Number of lines that will at least be affected when calling the bulk functions.
- `#define sec_check`
Check if size is larger than actual memory.

Functions

- `int pn_cache_init (void)`
Function that has to be called in the main function before any of the functions in the [Cache Module](#) can be called. Initializes some internal data and enables the cache.
- `int pn_cache_enable (void)`
Enables instruction and data cache. When changing this, make sure that the `pn_cache_init()` function is still correct!
- `int pn_cache_disable (void)`
Disables instruction and data cache.
- `unsigned long pn_cache_linesize (void)`
Returns the cache line size in bit.
- `unsigned long pn_cache_size (void)`
Returns the cache size in Byte.
- `int pn_cache_invalidate (void *addr, unsigned long size)`
Invalidates the cache entries containing the given address range.
- `int pn_cache_invalidate_all (void)`
Invalidates the whole cache. When changing this, make sure that the `pn_cache_init()` function is still correct!
- `int pn_cache_writeback (void *addr, unsigned long size)`

Writes back the cache lines that cached the given address range.

- int [pn_cache_writeback_all](#) (void)

Writes whole cache back.

- int [pn_cache_flush](#) (void *addr, unsigned long size)

Combination of [pn_cache_invalidate\(\)](#) and [pn_cache_writeback\(\)](#).

- int [pn_cache_flush_all](#) (void)

Flushes the whole cache.

- void [enable_cache_as](#) (void)

Enables cache.

- void [disable_cache_as](#) (void)

Disables cache.

- unsigned int [cache_banks_as](#) (void)

Returns number of cache banks.

- unsigned int [cache_sets_as](#) (void)

Returns number of cache sets.

- unsigned int [mem_size_as](#) (void)

Reads memory size.

- void [invalidate_1024_as](#) (unsigned int addr, unsigned int endaddr)

Invalidates single cache lines until end address is reached.

- void [invalidate_bulk_1024_as](#) (unsigned int addr, unsigned int endaddr)

Invalidates 32 cache lines at once until end address is reached.

- void [writeback_1024_as](#) (unsigned int addr, unsigned int endaddr)

Writes back single cache lines until end address is reached.

- void [writeback_bulk_1024_as](#) (unsigned int addr, unsigned int endaddr)

Writes back 32 cache lines at once until end address is reached.

- void [flush_1024_as](#) (unsigned int addr, unsigned int endaddr)

Flushes single cache lines until end address is reached.

- void [flush_bulk_1024_as](#) (unsigned int addr, unsigned int endaddr)

Flushes 32 cache lines at once until end address is reached.

- void [invalidate_32_as](#) (unsigned int addr, unsigned int endaddr)

Invalidates single cache lines until end address is reached.

- void [invalidate_bulk_32_as](#) (unsigned int addr, unsigned int endaddr)

Invalidates 32 cache lines at once until end address is reached.

- void [writeback_32_as](#) (unsigned int addr, unsigned int endaddr)

Writes back single cache lines until end address is reached.

- void [writeback_bulk_32_as](#) (unsigned int addr, unsigned int endaddr)

Writes back 32 cache lines at once until end address is reached.

- void [flush_32_as](#) (unsigned int addr, unsigned int endaddr)

Flushes single cache lines until end address is reached.

- void [flush_bulk_32_as](#) (unsigned int addr, unsigned int endaddr)

Flushes 32 cache lines at once until end address is reached.

- void [invalidate_64_as](#) (unsigned int addr, unsigned int endaddr)
Invalidate single cache lines until end address is reached.
 - void [invalidate_bulk_64_as](#) (unsigned int addr, unsigned int endaddr)
Invalidate 32 cache lines at once until end address is reached.
 - void [writeback_64_as](#) (unsigned int addr, unsigned int endaddr)
Write back single cache lines until end address is reached.
 - void [writeback_bulk_64_as](#) (unsigned int addr, unsigned int endaddr)
Write back 32 cache lines at once until end address is reached.
 - void [flush_64_as](#) (unsigned int addr, unsigned int endaddr)
Flush single cache lines until end address is reached.
 - void [flush_bulk_64_as](#) (unsigned int addr, unsigned int endaddr)
Flush 32 cache lines at once until end address is reached.
-
- void [invalidate_128_as](#) (unsigned int addr, unsigned int endaddr)
Invalidate single cache lines until end address is reached.
 - void [invalidate_bulk_128_as](#) (unsigned int addr, unsigned int endaddr)
Invalidate 32 cache lines at once until end address is reached.
 - void [writeback_128_as](#) (unsigned int addr, unsigned int endaddr)
Write back single cache lines until end address is reached.
 - void [writeback_bulk_128_as](#) (unsigned int addr, unsigned int endaddr)
Write back 32 cache lines at once until end address is reached.
 - void [flush_128_as](#) (unsigned int addr, unsigned int endaddr)
Flush single cache lines until end address is reached.
 - void [flush_bulk_128_as](#) (unsigned int addr, unsigned int endaddr)
Flush 32 cache lines at once until end address is reached.
-
- void [invalidate_256_as](#) (unsigned int addr, unsigned int endaddr)
Invalidate single cache lines until end address is reached.
 - void [invalidate_bulk_256_as](#) (unsigned int addr, unsigned int endaddr)
Invalidate 32 cache lines at once until end address is reached.
 - void [writeback_256_as](#) (unsigned int addr, unsigned int endaddr)
Write back single cache lines until end address is reached.
 - void [writeback_bulk_256_as](#) (unsigned int addr, unsigned int endaddr)
Write back 32 cache lines at once until end address is reached.
 - void [flush_256_as](#) (unsigned int addr, unsigned int endaddr)
Flush single cache lines until end address is reached.
 - void [flush_bulk_256_as](#) (unsigned int addr, unsigned int endaddr)
Flush 32 cache lines at once until end address is reached.
-
- void [invalidate_512_as](#) (unsigned int addr, unsigned int endaddr)
Invalidate single cache lines until end address is reached.
 - void [invalidate_bulk_512_as](#) (unsigned int addr, unsigned int endaddr)

- Invalidates 32 cache lines at once until end address is reached.*

 - void [writeback_512_as](#) (unsigned int addr, unsigned int endaddr)

Writes back single cache lines until end address is reached.

- void [writeback_bulk_512_as](#) (unsigned int addr, unsigned int endaddr)

Writes back 32 cache lines at once until end address is reached.

- void [flush_512_as](#) (unsigned int addr, unsigned int endaddr)

Flushes single cache lines until end address is reached.

- void [flush_bulk_512_as](#) (unsigned int addr, unsigned int endaddr)

Flushes 32 cache lines at once until end address is reached.

- void [invalidate_2048_as](#) (unsigned int addr, unsigned int endaddr)

Invalidates single cache lines until end address is reached.

- void [invalidate_bulk_2048_as](#) (unsigned int addr, unsigned int endaddr)

Invalidates 32 cache lines at once until end address is reached.

- void [writeback_2048_as](#) (unsigned int addr, unsigned int endaddr)

Writes back single cache lines until end address is reached.

- void [writeback_bulk_2048_as](#) (unsigned int addr, unsigned int endaddr)

Writes back 32 cache lines at once until end address is reached.

- void [flush_2048_as](#) (unsigned int addr, unsigned int endaddr)

Flushes single cache lines until end address is reached.

- void [flush_bulk_2048_as](#) (unsigned int addr, unsigned int endaddr)

Flushes 32 cache lines at once until end address is reached.

8.8.1 Detailed Description

Contains architecture independent implementations of the [Cache Module](#) functions.

Functions with suffix `_as` are architecture specific and therefore implemented in the `pn_cache_$(PN_ISA).S` file in the same directory.

For RV32I architecture, it was not practical to implement the assembly code by hand. A buildscript called [pn_cache_RV32I_buildscript.py](#) was created instead which takes the `PN_CACHE_LINESIZE` parameter and creates a correct version of the `pn_cache_RV32I_nnn.S` (example: [pn_cache_RV32I_auto.S](#)) file. For reasons on why this was done, see the documentation of [pn_cache_RV32I_buildscript.py](#) itself.

8.8.2 Macro Definition Documentation

8.8.2.1 BULK_LINES

```
#define BULK_LINES 32
```

Number of lines that will at least be affected when calling the bulk functions.

8.8.2.2 sec_check

```
#define sec_check
```

Value:

```
if (size > s_mem_size) \
    return PN_ERR_PARAM;
```

Check if size is larger than actual memory.

Todo Can these functions be used on CoPU?

8.9 pn_cache/pn_cache_RV32I_1024.S File Reference

Contains RV32I assembly implementations of assembly functions called in [pn_cache.c](#) for cache line size 1024.

Functions

- void [enable_cache_as](#) (void)
Enables cache.
- void [disable_cache_as](#) (void)
Disables cache.
- unsigned int [cache_banks_as](#) (void)
Returns number of cache banks.
- unsigned int [cache_sets_as](#) (void)
Returns number of cache sets.
- unsigned int [mem_size_as](#) (void)
Reads memory size.
- void [invalidate_1024_as](#) (unsigned int addr, unsigned int endaddr)
Invalidates single cache lines until end address is reached.
- void [invalidate_bulk_1024_as](#) (unsigned int addr, unsigned int endaddr)
Invalidates 32 cache lines at once until end address is reached.
- void [writeback_1024_as](#) (unsigned int addr, unsigned int endaddr)
Writes back single cache lines until end address is reached.
- void [writeback_bulk_1024_as](#) (unsigned int addr, unsigned int endaddr)
Writes back 32 cache lines at once until end address is reached.
- void [flush_1024_as](#) (unsigned int addr, unsigned int endaddr)
Flushes single cache lines until end address is reached.
- void [flush_bulk_1024_as](#) (unsigned int addr, unsigned int endaddr)
Flushes 32 cache lines at once until end address is reached.

8.9.1 Detailed Description

Contains RV32I assembly implementations of assembly functions called in [pn_cache.c](#) for cache line size 1024.

This file (for this specific architecture, others may do it differently) was created by [pn_cache_RV32I_buildscript.py](#) while compiling (automatically called by [Makefile](#)), as are the other files with the pattern `pn_cache_RV32I_*.S` for name. The reasons for this are described in the [pn_cache_RV32I_buildscript.py](#) documentation itself.

The code shown below was created with a cache line size of 1024 Bit.

So far, the following cache line sizes are available (in bits): 32 64 128 256 512 1024 2048

Contains implementations of:

- [enable_cache_as\(\)](#)
- [disable_cache_as\(\)](#)
- [cache_banks_as\(\)](#)
- [cache_sets_as\(\)](#)
- [mem_size_as\(\)](#)
- [invalidate_1024_as\(\)](#)
- [invalidate_bulk_1024_as\(\)](#)
- [writeback_1024_as\(\)](#)
- [writeback_bulk_1024_as\(\)](#)
- [flush_1024_as\(\)](#)
- [flush_bulk_1024_as\(\)](#)

```

1  /*
2  * I was automatically generated by pn_cache_RV32I_buildscript.py,
3  * invoked by libparanut Makefile -> No edits!
4  */
5
43 /*
44 * Put in here so Doxygen will know that it is implemented in this
45 * file. Sadly, Doxygen has no built in assembly interpreter, so we
46 * are stuck with this.
47 */
48
49 #ifdef DOXYGEN
50     void enable_cache_as(void) {}
51
52     void disable_cache_as(void) {}
53
54     unsigned int cache_banks_as(void) {}
55
56     unsigned int cache_sets_as(void) {}
57
58     unsigned int mem_size_as(void) {}
59
60     void invalidate_1024_as(unsigned int addr, unsigned int endaddr) {}
61
62     void invalidate_bulk_1024_as(unsigned int addr, unsigned int endaddr) {}
63
64     void writeback_1024_as(unsigned int addr, unsigned int endaddr) {}
65
66     void writeback_bulk_1024_as(unsigned int addr, unsigned int endaddr) {}
67
68     void flush_1024_as(unsigned int addr, unsigned int endaddr) {}
69
70     void flush_bulk_1024_as(unsigned int addr, unsigned int endaddr) {}
71
72 #endif /* DOXYGEN */
73

```

```

171 /*Header*****
172
173 #ifndef DOXYGEN
174
175 .text                /* enter text section                */
176 .align 2            /* align Code to 2^2 Bytes                */
177
178 /* declare labels in here to be global */
179 .globl enable_cache_as
180 .globl disable_cache_as
181 .globl cache_banks_as
182 .globl cache_sets_as
183 .globl mem_size_as
184 .globl invalidate_1024_as
185 .globl invalidate_bulk_1024_as
186 .globl writeback_1024_as
187 .globl writeback_bulk_1024_as
188 .globl flush_1024_as
189 .globl flush_bulk_1024_as
190
191 /* ParaNUT Custom Registers and Instructions */
192 #include "custom_RV32I.S"
193
194 /*Functions*****
195
196 enable_cache_as:
197     fence
198     li    t0,      3
199     csrs  pncache, t0
200     ret
201
202 /*-----*/
203
204 disable_cache_as:
205     fence
206     li    t0,      0
207     csrw  pncache, t0
208     ret
209
210 /*-----*/
211
212 cache_banks_as:
213     csrr  a0,      pncacheinfo
214     srli  a0,      a0,      8
215     ret
216
217 /*-----*/
218
219 cache_sets_as:
220     csrr  a0,      pncachesets
221     ret
222
223 /*-----*/
224
225 mem_size_as:
226     csrr  a0,      pnmemsize
227     ret
228
229 /*-----*/
230
231 invalidate_1024_as:
232     fence
233 invalidate_1024_as_loop:
234     CINV(10, 0x000)
235     addi  a0, a0, 128
236     blt  a0, a1, invalidate_1024_as_loop
237     ret
238
239 /*-----*/
240
241 invalidate_bulk_1024_as:
242     fence
243     li    t0, 2048
244 invalidate_bulk_1024_as_loop:
245     CINV(10, 0)
246     CINV(10, 128)
247     CINV(10, 256)
248     CINV(10, 384)
249     CINV(10, 512)
250     CINV(10, 640)
251     CINV(10, 768)
252     CINV(10, 896)
253     CINV(10, 1024)
254     CINV(10, 1152)
255     CINV(10, 1280)
256     CINV(10, 1408)
257     CINV(10, 1536)

```

```

258     CINV(10, 1664)
259     CINV(10, 1792)
260     CINV(10, 1920)
261     add a0, a0, t0
262     CINV(10, 0)
263     CINV(10, 128)
264     CINV(10, 256)
265     CINV(10, 384)
266     CINV(10, 512)
267     CINV(10, 640)
268     CINV(10, 768)
269     CINV(10, 896)
270     CINV(10, 1024)
271     CINV(10, 1152)
272     CINV(10, 1280)
273     CINV(10, 1408)
274     CINV(10, 1536)
275     CINV(10, 1664)
276     CINV(10, 1792)
277     CINV(10, 1920)
278     add a0, a0, t0
279     blt a0, a1, invalidate_bulk_1024_as_loop
280     ret
281
282 /*-----*/
283
284 writeback_1024_as:
285     fence
286 writeback_1024_as_loop:
287     CWB(10, 0x000)
288     addi a0, a0, 128
289     blt a0, a1, writeback_1024_as_loop
290     ret
291
292 /*-----*/
293
294 writeback_bulk_1024_as:
295     fence
296 li t0, 2048
297 writeback_bulk_1024_as_loop:
298     CWB(10, 0)
299     CWB(10, 128)
300     CWB(10, 256)
301     CWB(10, 384)
302     CWB(10, 512)
303     CWB(10, 640)
304     CWB(10, 768)
305     CWB(10, 896)
306     CWB(10, 1024)
307     CWB(10, 1152)
308     CWB(10, 1280)
309     CWB(10, 1408)
310     CWB(10, 1536)
311     CWB(10, 1664)
312     CWB(10, 1792)
313     CWB(10, 1920)
314     add a0, a0, t0
315     CWB(10, 0)
316     CWB(10, 128)
317     CWB(10, 256)
318     CWB(10, 384)
319     CWB(10, 512)
320     CWB(10, 640)
321     CWB(10, 768)
322     CWB(10, 896)
323     CWB(10, 1024)
324     CWB(10, 1152)
325     CWB(10, 1280)
326     CWB(10, 1408)
327     CWB(10, 1536)
328     CWB(10, 1664)
329     CWB(10, 1792)
330     CWB(10, 1920)
331     add a0, a0, t0
332     blt a0, a1, writeback_bulk_1024_as_loop
333     ret
334
335 /*-----*/
336
337 flush_1024_as:
338     fence
339 flush_1024_as_loop:
340     CFLUSH(10, 0x000)
341     addi a0, a0, 128
342     blt a0, a1, flush_1024_as_loop
343     ret
344

```

```

345 /*-----*/
346
347 flush_bulk_1024_as:
348     fence
349     li t0, 2048
350 flush_bulk_1024_as_loop:
351     CFLUSH(10, 0)
352     CFLUSH(10, 128)
353     CFLUSH(10, 256)
354     CFLUSH(10, 384)
355     CFLUSH(10, 512)
356     CFLUSH(10, 640)
357     CFLUSH(10, 768)
358     CFLUSH(10, 896)
359     CFLUSH(10, 1024)
360     CFLUSH(10, 1152)
361     CFLUSH(10, 1280)
362     CFLUSH(10, 1408)
363     CFLUSH(10, 1536)
364     CFLUSH(10, 1664)
365     CFLUSH(10, 1792)
366     CFLUSH(10, 1920)
367     add a0, a0, t0
368     CFLUSH(10, 0)
369     CFLUSH(10, 128)
370     CFLUSH(10, 256)
371     CFLUSH(10, 384)
372     CFLUSH(10, 512)
373     CFLUSH(10, 640)
374     CFLUSH(10, 768)
375     CFLUSH(10, 896)
376     CFLUSH(10, 1024)
377     CFLUSH(10, 1152)
378     CFLUSH(10, 1280)
379     CFLUSH(10, 1408)
380     CFLUSH(10, 1536)
381     CFLUSH(10, 1664)
382     CFLUSH(10, 1792)
383     CFLUSH(10, 1920)
384     add a0, a0, t0
385     blt a0, a1, flush_bulk_1024_as_loop
386     ret
387
388 #endif /* !DOXYGEN */
389
390 /*EOF******/

```

8.10 pn_cache/pn_cache_RV32I_128.S File Reference

Contains RV32I assembly implementations of assembly functions called in [pn_cache.c](#) for cache line size 128.

Functions

- void [enable_cache_as](#) (void)
Enables cache.
- void [disable_cache_as](#) (void)
Disables cache.
- unsigned int [cache_banks_as](#) (void)
Returns number of cache banks.
- unsigned int [cache_sets_as](#) (void)
Returns number of cache sets.
- unsigned int [mem_size_as](#) (void)
Reads memory size.

- void [invalidate_128_as](#) (unsigned int addr, unsigned int endaddr)
Invalidates single cache lines until end address is reached.
- void [invalidate_bulk_128_as](#) (unsigned int addr, unsigned int endaddr)
Invalidates 32 cache lines at once until end address is reached.
- void [writeback_128_as](#) (unsigned int addr, unsigned int endaddr)
Writes back single cache lines until end address is reached.
- void [writeback_bulk_128_as](#) (unsigned int addr, unsigned int endaddr)
Writes back 32 cache lines at once until end address is reached.
- void [flush_128_as](#) (unsigned int addr, unsigned int endaddr)
Flushes single cache lines until end address is reached.
- void [flush_bulk_128_as](#) (unsigned int addr, unsigned int endaddr)
Flushes 32 cache lines at once until end address is reached.

8.10.1 Detailed Description

Contains RV32I assembly implementations of assembly functions called in [pn_cache.c](#) for cache line size 128.

This file (for this specific architecture, others may do it differently) was created by [pn_cache_RV32I_buildscript.py](#) while compiling (automatically called by [Makefile](#)), as are the other files with the pattern `pn_cache_RV32I_*.S` for name. The reasons for this are described in the [pn_cache_RV32I_buildscript.py](#) documentation itself.

The code shown below was created with a cache line size of 128 Bit.

So far, the following cache line sizes are available (in bits): 32 64 128 256 512 1024 2048

Contains implementations of:

- [enable_cache_as\(\)](#)
- [disable_cache_as\(\)](#)
- [cache_banks_as\(\)](#)
- [cache_sets_as\(\)](#)
- [mem_size_as\(\)](#)
- [invalidate_128_as\(\)](#)
- [invalidate_bulk_128_as\(\)](#)
- [writeback_128_as\(\)](#)
- [writeback_bulk_128_as\(\)](#)
- [flush_128_as\(\)](#)
- [flush_bulk_128_as\(\)](#)

```

1  /*
2  * I was automatically generated by pn_cache_RV32I_buildscript.py,
3  * invoked by libparanut Makefile -> No edits!
4  */
5
43 /*
44 * Put in here so Doxygen will know that it is implemented in this
45 * file. Sadly, Doxygen has no built in assembly interpreter, so we
46 * are stuck with this.
47 */
48
49 #ifndef DOXYGEN
50
65     void enable_cache_as(void) {}
66
72     void disable_cache_as(void) {}
73
81     unsigned int cache_banks_as(void) {}
82
90     unsigned int cache_sets_as(void) {}
91
99     unsigned int mem_size_as(void) {}
100
109     void invalidate_128_as(unsigned int addr, unsigned int endaddr) {}
110
119     void invalidate_bulk_128_as(unsigned int addr, unsigned int endaddr) {}
120
129     void writeback_128_as(unsigned int addr, unsigned int endaddr) {}
130
139     void writeback_bulk_128_as(unsigned int addr, unsigned int endaddr) {}
140
149     void flush_128_as(unsigned int addr, unsigned int endaddr) {}
150
159     void flush_bulk_128_as(unsigned int addr, unsigned int endaddr) {}
160
169 #endif /* DOXYGEN */
170
171 /*Header*****
172
173 #ifndef DOXYGEN
174
175 .text                                /* enter text section          */
176 .align 2                            /* align Code to 2^2 Bytes      */
177
178 /* declare labels in here to be global */
179 .globl enable_cache_as
180 .globl disable_cache_as
181 .globl cache_banks_as
182 .globl cache_sets_as
183 .globl mem_size_as
184 .globl invalidate_128_as
185 .globl invalidate_bulk_128_as
186 .globl writeback_128_as
187 .globl writeback_bulk_128_as
188 .globl flush_128_as
189 .globl flush_bulk_128_as
190
191 /* ParaNut Custom Registers and Instructions */
192 #include "custom_RV32I.S"
193
194 /*Functions*****
195
196 enable_cache_as:
197     fence
198     li    t0,    3
199     csrs  pncache, t0
200     ret
201
202 /*-----*/
203
204 disable_cache_as:
205     fence
206     li    t0,    0
207     csrw  pncache, t0
208     ret
209
210 /*-----*/
211
212 cache_banks_as:
213     csrr a0,    pncacheinfo
214     srli a0,    a0,    8
215     ret
216
217 /*-----*/
218
219 cache_sets_as:
220     csrr a0,    pncachesets

```

```

221     ret
222
223 /*-----*/
224
225 mem_size_as:
226     csrr a0,          pnmemsize
227     ret
228
229 /*-----*/
230
231 invalidate_128_as:
232     fence
233 invalidate_128_as_loop:
234     CINV(10, 0x000)
235     addi a0, a0, 16
236     blt a0, a1, invalidate_128_as_loop
237     ret
238
239 /*-----*/
240
241 invalidate_bulk_128_as:
242     fence
243 invalidate_bulk_128_as_loop:
244     CINV(10, 0)
245     CINV(10, 16)
246     CINV(10, 32)
247     CINV(10, 48)
248     CINV(10, 64)
249     CINV(10, 80)
250     CINV(10, 96)
251     CINV(10, 112)
252     CINV(10, 128)
253     CINV(10, 144)
254     CINV(10, 160)
255     CINV(10, 176)
256     CINV(10, 192)
257     CINV(10, 208)
258     CINV(10, 224)
259     CINV(10, 240)
260     CINV(10, 256)
261     CINV(10, 272)
262     CINV(10, 288)
263     CINV(10, 304)
264     CINV(10, 320)
265     CINV(10, 336)
266     CINV(10, 352)
267     CINV(10, 368)
268     CINV(10, 384)
269     CINV(10, 400)
270     CINV(10, 416)
271     CINV(10, 432)
272     CINV(10, 448)
273     CINV(10, 464)
274     CINV(10, 480)
275     CINV(10, 496)
276     addi a0, a0, 512
277     blt a0, a1, invalidate_bulk_128_as_loop
278     ret
279
280 /*-----*/
281
282 writeback_128_as:
283     fence
284 writeback_128_as_loop:
285     CWB(10, 0x000)
286     addi a0, a0, 16
287     blt a0, a1, writeback_128_as_loop
288     ret
289
290 /*-----*/
291
292 writeback_bulk_128_as:
293     fence
294 writeback_bulk_128_as_loop:
295     CWB(10, 0)
296     CWB(10, 16)
297     CWB(10, 32)
298     CWB(10, 48)
299     CWB(10, 64)
300     CWB(10, 80)
301     CWB(10, 96)
302     CWB(10, 112)
303     CWB(10, 128)
304     CWB(10, 144)
305     CWB(10, 160)
306     CWB(10, 176)
307     CWB(10, 192)

```



```

308    CWB(10, 208)
309    CWB(10, 224)
310    CWB(10, 240)
311    CWB(10, 256)
312    CWB(10, 272)
313    CWB(10, 288)
314    CWB(10, 304)
315    CWB(10, 320)
316    CWB(10, 336)
317    CWB(10, 352)
318    CWB(10, 368)
319    CWB(10, 384)
320    CWB(10, 400)
321    CWB(10, 416)
322    CWB(10, 432)
323    CWB(10, 448)
324    CWB(10, 464)
325    CWB(10, 480)
326    CWB(10, 496)
327    addi a0, a0, 512
328    blt a0, a1, writeback_bulk_128_as_loop
329    ret
330
331    /*-----*/
332
333    flush_128_as:
334        fence
335    flush_128_as_loop:
336        CFLUSH(10, 0x000)
337        addi a0, a0, 16
338        blt a0, a1, flush_128_as_loop
339        ret
340
341    /*-----*/
342
343    flush_bulk_128_as:
344        fence
345    flush_bulk_128_as_loop:
346        CFLUSH(10, 0)
347        CFLUSH(10, 16)
348        CFLUSH(10, 32)
349        CFLUSH(10, 48)
350        CFLUSH(10, 64)
351        CFLUSH(10, 80)
352        CFLUSH(10, 96)
353        CFLUSH(10, 112)
354        CFLUSH(10, 128)
355        CFLUSH(10, 144)
356        CFLUSH(10, 160)
357        CFLUSH(10, 176)
358        CFLUSH(10, 192)
359        CFLUSH(10, 208)
360        CFLUSH(10, 224)
361        CFLUSH(10, 240)
362        CFLUSH(10, 256)
363        CFLUSH(10, 272)
364        CFLUSH(10, 288)
365        CFLUSH(10, 304)
366        CFLUSH(10, 320)
367        CFLUSH(10, 336)
368        CFLUSH(10, 352)
369        CFLUSH(10, 368)
370        CFLUSH(10, 384)
371        CFLUSH(10, 400)
372        CFLUSH(10, 416)
373        CFLUSH(10, 432)
374        CFLUSH(10, 448)
375        CFLUSH(10, 464)
376        CFLUSH(10, 480)
377        CFLUSH(10, 496)
378        addi a0, a0, 512
379        blt a0, a1, flush_bulk_128_as_loop
380        ret
381
382    #endif /* !DOXYGEN */
383
384    /*EOF******/

```

8.11 pn_cache/pn_cache_RV32I_2048.S File Reference

Contains RV32I assembly implementations of assembly functions called in [pn_cache.c](#) for cache line size 2048.

Functions

- void [enable_cache_as](#) (void)
Enables cache.
 - void [disable_cache_as](#) (void)
Disables cache.
 - unsigned int [cache_banks_as](#) (void)
Returns number of cache banks.
 - unsigned int [cache_sets_as](#) (void)
Returns number of cache sets.
 - unsigned int [mem_size_as](#) (void)
Reads memory size.
-
- void [invalidate_2048_as](#) (unsigned int addr, unsigned int endaddr)
Invalidates single cache lines until end address is reached.
 - void [invalidate_bulk_2048_as](#) (unsigned int addr, unsigned int endaddr)
Invalidates 32 cache lines at once until end address is reached.
 - void [writeback_2048_as](#) (unsigned int addr, unsigned int endaddr)
Writes back single cache lines until end address is reached.
 - void [writeback_bulk_2048_as](#) (unsigned int addr, unsigned int endaddr)
Writes back 32 cache lines at once until end address is reached.
 - void [flush_2048_as](#) (unsigned int addr, unsigned int endaddr)
Flushes single cache lines until end address is reached.
 - void [flush_bulk_2048_as](#) (unsigned int addr, unsigned int endaddr)
Flushes 32 cache lines at once until end address is reached.

8.11.1 Detailed Description

Contains RV32I assembly implementations of assembly functions called in [pn_cache.c](#) for cache line size 2048.

This file (for this specific architecture, others may do it differently) was created by [pn_cache_RV32I_buildscript.py](#) while compiling (automatically called by [Makefile](#)), as are the other files with the pattern `pn_cache_RV32I_*.S` for name. The reasons for this are described in the [pn_cache_RV32I_buildscript.py](#) documentation itself.

The code shown below was created with a cache line size of 2048 Bit.

So far, the following cache line sizes are available (in bits): 32 64 128 256 512 1024 2048

Contains implementations of:

- [enable_cache_as\(\)](#)
- [disable_cache_as\(\)](#)
- [cache_banks_as\(\)](#)
- [cache_sets_as\(\)](#)

- [mem_size_as\(\)](#)
- [invalidate_2048_as\(\)](#)
- [invalidate_bulk_2048_as\(\)](#)
- [writeback_2048_as\(\)](#)
- [writeback_bulk_2048_as\(\)](#)
- [flush_2048_as\(\)](#)
- [flush_bulk_2048_as\(\)](#)

```

1  /*
2  * I was automatically generated by pn_cache_RV32I_buildscript.py,
3  * invoked by libparanut Makefile -> No edits!
4  */
5
63  /*
64  * Put in here so Doxygen will know that it is implemented in this
65  * file. Sadly, Doxygen has no built in assembly interpreter, so we
66  * are stuck with this.
67  */
68
69  #ifndef DOXYGEN
70
71      void enable_cache_as(void) {}
72
73      void disable_cache_as(void) {}
74
75      unsigned int cache_banks_as(void) {}
76
77      unsigned int cache_sets_as(void) {}
78
79      unsigned int mem_size_as(void) {}
80
81      void invalidate_2048_as(unsigned int addr, unsigned int endaddr) {}
82
83      void invalidate_bulk_2048_as(unsigned int addr, unsigned int endaddr) {}
84
85      void writeback_2048_as(unsigned int addr, unsigned int endaddr) {}
86
87      void writeback_bulk_2048_as(unsigned int addr, unsigned int endaddr) {}
88
89      void flush_2048_as(unsigned int addr, unsigned int endaddr) {}
90
91      void flush_bulk_2048_as(unsigned int addr, unsigned int endaddr) {}
92
93  #endif /* DOXYGEN */
94
95  /*Header*****
96  #ifndef DOXYGEN
97
98      .text                                /* enter text section          */
99      .align 2                            /* align Code to 2^2 Bytes        */
100
101      /* declare labels in here to be global */
102      .globl enable_cache_as
103      .globl disable_cache_as
104      .globl cache_banks_as
105      .globl cache_sets_as
106      .globl mem_size_as
107      .globl invalidate_2048_as
108      .globl invalidate_bulk_2048_as
109      .globl writeback_2048_as
110      .globl writeback_bulk_2048_as
111      .globl flush_2048_as
112      .globl flush_bulk_2048_as
113
114      /* ParaNut Custom Registers and Instructions */
115      #include "custom_RV32I.S"
116
117      /*Functions*****
118
119      enable_cache_as:
120          fence
121          li    t0,    3
122          csrs  pncache, t0
123          ret
124
125  202  /*-----*/

```

```

203
204 disable_cache_as:
205     fence
206     li    t0,    0
207     csrwr pncache, t0
208     ret
209
210 /*-----*/
211
212 cache_banks_as:
213     csrr a0,    pncacheinfo
214     srli a0,    a0,    8
215     ret
216
217 /*-----*/
218
219 cache_sets_as:
220     csrr a0,    pncachesets
221     ret
222
223 /*-----*/
224
225 mem_size_as:
226     csrr a0,    pnmemsize
227     ret
228
229 /*-----*/
230
231 invalidate_2048_as:
232     fence
233 invalidate_2048_as_loop:
234     CINV(10, 0x000)
235     addi a0, a0, 256
236     blt a0, a1, invalidate_2048_as_loop
237     ret
238
239 /*-----*/
240
241 invalidate_bulk_2048_as:
242     fence
243     li t0, 2048
244 invalidate_bulk_2048_as_loop:
245     CINV(10, 0)
246     CINV(10, 256)
247     CINV(10, 512)
248     CINV(10, 768)
249     CINV(10, 1024)
250     CINV(10, 1280)
251     CINV(10, 1536)
252     CINV(10, 1792)
253     add a0, a0, t0
254     CINV(10, 0)
255     CINV(10, 256)
256     CINV(10, 512)
257     CINV(10, 768)
258     CINV(10, 1024)
259     CINV(10, 1280)
260     CINV(10, 1536)
261     CINV(10, 1792)
262     add a0, a0, t0
263     CINV(10, 0)
264     CINV(10, 256)
265     CINV(10, 512)
266     CINV(10, 768)
267     CINV(10, 1024)
268     CINV(10, 1280)
269     CINV(10, 1536)
270     CINV(10, 1792)
271     add a0, a0, t0
272     CINV(10, 0)
273     CINV(10, 256)
274     CINV(10, 512)
275     CINV(10, 768)
276     CINV(10, 1024)
277     CINV(10, 1280)
278     CINV(10, 1536)
279     CINV(10, 1792)
280     add a0, a0, t0
281     blt a0, a1, invalidate_bulk_2048_as_loop
282     ret
283
284 /*-----*/
285
286 writeback_2048_as:
287     fence
288 writeback_2048_as_loop:
289     CWB(10, 0x000)

```

```
290     addi a0, a0, 256
291     blt a0, a1, writeback_2048_as_loop
292     ret
293
294 /*-----*/
295
296 writeback_bulk_2048_as:
297     fence
298     li t0, 2048
299 writeback_bulk_2048_as_loop:
300     CWB(10, 0)
301     CWB(10, 256)
302     CWB(10, 512)
303     CWB(10, 768)
304     CWB(10, 1024)
305     CWB(10, 1280)
306     CWB(10, 1536)
307     CWB(10, 1792)
308     add a0, a0, t0
309     CWB(10, 0)
310     CWB(10, 256)
311     CWB(10, 512)
312     CWB(10, 768)
313     CWB(10, 1024)
314     CWB(10, 1280)
315     CWB(10, 1536)
316     CWB(10, 1792)
317     add a0, a0, t0
318     CWB(10, 0)
319     CWB(10, 256)
320     CWB(10, 512)
321     CWB(10, 768)
322     CWB(10, 1024)
323     CWB(10, 1280)
324     CWB(10, 1536)
325     CWB(10, 1792)
326     add a0, a0, t0
327     CWB(10, 0)
328     CWB(10, 256)
329     CWB(10, 512)
330     CWB(10, 768)
331     CWB(10, 1024)
332     CWB(10, 1280)
333     CWB(10, 1536)
334     CWB(10, 1792)
335     add a0, a0, t0
336     blt a0, a1, writeback_bulk_2048_as_loop
337     ret
338
339 /*-----*/
340
341 flush_2048_as:
342     fence
343 flush_2048_as_loop:
344     CFLUSH(10, 0x000)
345     addi a0, a0, 256
346     blt a0, a1, flush_2048_as_loop
347     ret
348
349 /*-----*/
350
351 flush_bulk_2048_as:
352     fence
353     li t0, 2048
354 flush_bulk_2048_as_loop:
355     CFLUSH(10, 0)
356     CFLUSH(10, 256)
357     CFLUSH(10, 512)
358     CFLUSH(10, 768)
359     CFLUSH(10, 1024)
360     CFLUSH(10, 1280)
361     CFLUSH(10, 1536)
362     CFLUSH(10, 1792)
363     add a0, a0, t0
364     CFLUSH(10, 0)
365     CFLUSH(10, 256)
366     CFLUSH(10, 512)
367     CFLUSH(10, 768)
368     CFLUSH(10, 1024)
369     CFLUSH(10, 1280)
370     CFLUSH(10, 1536)
371     CFLUSH(10, 1792)
372     add a0, a0, t0
373     CFLUSH(10, 0)
374     CFLUSH(10, 256)
375     CFLUSH(10, 512)
376     CFLUSH(10, 768)
```

```

377     CFLUSH(10, 1024)
378     CFLUSH(10, 1280)
379     CFLUSH(10, 1536)
380     CFLUSH(10, 1792)
381     add a0, a0, t0
382     CFLUSH(10, 0)
383     CFLUSH(10, 256)
384     CFLUSH(10, 512)
385     CFLUSH(10, 768)
386     CFLUSH(10, 1024)
387     CFLUSH(10, 1280)
388     CFLUSH(10, 1536)
389     CFLUSH(10, 1792)
390     add a0, a0, t0
391     blt  a0, a1, flush_bulk_2048_as_loop
392     ret
393
394 #endif /* !DOXYGEN */
395
396 /*EOF*****

```

8.12 pn_cache/pn_cache_RV32I_256.S File Reference

Contains RV32I assembly implementations of assembly functions called in [pn_cache.c](#) for cache line size 256.

Functions

- void [enable_cache_as](#) (void)
Enables cache.
 - void [disable_cache_as](#) (void)
Disables cache.
 - unsigned int [cache_banks_as](#) (void)
Returns number of cache banks.
 - unsigned int [cache_sets_as](#) (void)
Returns number of cache sets.
 - unsigned int [mem_size_as](#) (void)
Reads memory size.
-
- void [invalidate_256_as](#) (unsigned int addr, unsigned int endaddr)
Invalidates single cache lines until end address is reached.
 - void [invalidate_bulk_256_as](#) (unsigned int addr, unsigned int endaddr)
Invalidates 32 cache lines at once until end address is reached.
 - void [writeback_256_as](#) (unsigned int addr, unsigned int endaddr)
Writes back single cache lines until end address is reached.
 - void [writeback_bulk_256_as](#) (unsigned int addr, unsigned int endaddr)
Writes back 32 cache lines at once until end address is reached.
 - void [flush_256_as](#) (unsigned int addr, unsigned int endaddr)
Flushes single cache lines until end address is reached.
 - void [flush_bulk_256_as](#) (unsigned int addr, unsigned int endaddr)
Flushes 32 cache lines at once until end address is reached.

8.12.1 Detailed Description

Contains RV32I assembly implementations of assembly functions called in [pn_cache.c](#) for cache line size 256.

This file (for this specific architecture, others may do it differently) was created by [pn_cache_RV32I_buildscript.py](#) while compiling (automatically called by [Makefile](#)), as are the other files with the pattern `pn_cache_RV32I_*.S` for name. The reasons for this are described in the [pn_cache_RV32I_buildscript.py](#) documentation itself.

The code shown below was created with a cache line size of 256 Bit.

So far, the following cache line sizes are available (in bits): 32 64 128 256 512 1024 2048

Contains implementations of:

- [enable_cache_as\(\)](#)
- [disable_cache_as\(\)](#)
- [cache_banks_as\(\)](#)
- [cache_sets_as\(\)](#)
- [mem_size_as\(\)](#)
- [invalidate_256_as\(\)](#)
- [invalidate_bulk_256_as\(\)](#)
- [writeback_256_as\(\)](#)
- [writeback_bulk_256_as\(\)](#)
- [flush_256_as\(\)](#)
- [flush_bulk_256_as\(\)](#)

```

1  /*
2  * I was automatically generated by pn_cache_RV32I_buildscript.py,
3  * invoked by libparanut Makefile -> No edits!
4  */
5
43 /*
44 * Put in here so Doxygen will know that it is implemented in this
45 * file. Sadly, Doxygen has no built in assembly interpreter, so we
46 * are stuck with this.
47 */
48
49 #ifdef DOXYGEN
50     void enable_cache_as(void) {}
51
52     void disable_cache_as(void) {}
53
54     unsigned int cache_banks_as(void) {}
55
56     unsigned int cache_sets_as(void) {}
57
58     unsigned int mem_size_as(void) {}
59
60     void invalidate_256_as(unsigned int addr, unsigned int endaddr) {}
61
62     void invalidate_bulk_256_as(unsigned int addr, unsigned int endaddr) {}
63
64     void writeback_256_as(unsigned int addr, unsigned int endaddr) {}
65
66     void writeback_bulk_256_as(unsigned int addr, unsigned int endaddr) {}
67
68     void flush_256_as(unsigned int addr, unsigned int endaddr) {}
69
70     void flush_bulk_256_as(unsigned int addr, unsigned int endaddr) {}
71
72 #endif /* DOXYGEN */
73

```

```

171 /*Header******/
172
173 #ifndef DOXYGEN
174
175 .text                /* enter text section                */
176 .align 2             /* align Code to 2^2 Bytes                */
177
178 /* declare labels in here to be global */
179 .globl enable_cache_as
180 .globl disable_cache_as
181 .globl cache_banks_as
182 .globl cache_sets_as
183 .globl mem_size_as
184 .globl invalidate_256_as
185 .globl invalidate_bulk_256_as
186 .globl writeback_256_as
187 .globl writeback_bulk_256_as
188 .globl flush_256_as
189 .globl flush_bulk_256_as
190
191 /* ParaNut Custom Registers and Instructions */
192 #include "custom_RV32I.S"
193
194 /*Functions******/
195
196 enable_cache_as:
197     fence
198     li    t0,      3
199     csrs  pncache, t0
200     ret
201
202 /*-----*/
203
204 disable_cache_as:
205     fence
206     li    t0,      0
207     csrw  pncache, t0
208     ret
209
210 /*-----*/
211
212 cache_banks_as:
213     csrr  a0,      pncacheinfo
214     srli  a0,      a0,      8
215     ret
216
217 /*-----*/
218
219 cache_sets_as:
220     csrr  a0,      pncachesets
221     ret
222
223 /*-----*/
224
225 mem_size_as:
226     csrr  a0,      pnmemsize
227     ret
228
229 /*-----*/
230
231 invalidate_256_as:
232     fence
233 invalidate_256_as_loop:
234     CINV(10, 0x000)
235     addi  a0, a0, 32
236     blt  a0, a1, invalidate_256_as_loop
237     ret
238
239 /*-----*/
240
241 invalidate_bulk_256_as:
242     fence
243 invalidate_bulk_256_as_loop:
244     CINV(10, 0)
245     CINV(10, 32)
246     CINV(10, 64)
247     CINV(10, 96)
248     CINV(10, 128)
249     CINV(10, 160)
250     CINV(10, 192)
251     CINV(10, 224)
252     CINV(10, 256)
253     CINV(10, 288)
254     CINV(10, 320)
255     CINV(10, 352)
256     CINV(10, 384)
257     CINV(10, 416)

```



```
258     CINV(10, 448)
259     CINV(10, 480)
260     CINV(10, 512)
261     CINV(10, 544)
262     CINV(10, 576)
263     CINV(10, 608)
264     CINV(10, 640)
265     CINV(10, 672)
266     CINV(10, 704)
267     CINV(10, 736)
268     CINV(10, 768)
269     CINV(10, 800)
270     CINV(10, 832)
271     CINV(10, 864)
272     CINV(10, 896)
273     CINV(10, 928)
274     CINV(10, 960)
275     CINV(10, 992)
276     addi a0, a0, 1024
277     blt a0, a1, invalidate_bulk_256_as_loop
278     ret
279
280 /*-----*/
281
282 writeback_256_as:
283     fence
284 writeback_256_as_loop:
285     CWB(10, 0x000)
286     addi a0, a0, 32
287     blt a0, a1, writeback_256_as_loop
288     ret
289
290 /*-----*/
291
292 writeback_bulk_256_as:
293     fence
294 writeback_bulk_256_as_loop:
295     CWB(10, 0)
296     CWB(10, 32)
297     CWB(10, 64)
298     CWB(10, 96)
299     CWB(10, 128)
300     CWB(10, 160)
301     CWB(10, 192)
302     CWB(10, 224)
303     CWB(10, 256)
304     CWB(10, 288)
305     CWB(10, 320)
306     CWB(10, 352)
307     CWB(10, 384)
308     CWB(10, 416)
309     CWB(10, 448)
310     CWB(10, 480)
311     CWB(10, 512)
312     CWB(10, 544)
313     CWB(10, 576)
314     CWB(10, 608)
315     CWB(10, 640)
316     CWB(10, 672)
317     CWB(10, 704)
318     CWB(10, 736)
319     CWB(10, 768)
320     CWB(10, 800)
321     CWB(10, 832)
322     CWB(10, 864)
323     CWB(10, 896)
324     CWB(10, 928)
325     CWB(10, 960)
326     CWB(10, 992)
327     addi a0, a0, 1024
328     blt a0, a1, writeback_bulk_256_as_loop
329     ret
330
331 /*-----*/
332
333 flush_256_as:
334     fence
335 flush_256_as_loop:
336     CFLUSH(10, 0x000)
337     addi a0, a0, 32
338     blt a0, a1, flush_256_as_loop
339     ret
340
341 /*-----*/
342
343 flush_bulk_256_as:
344     fence
```

```

345 flush_bulk_256_as_loop:
346     CFLUSH(10, 0)
347     CFLUSH(10, 32)
348     CFLUSH(10, 64)
349     CFLUSH(10, 96)
350     CFLUSH(10, 128)
351     CFLUSH(10, 160)
352     CFLUSH(10, 192)
353     CFLUSH(10, 224)
354     CFLUSH(10, 256)
355     CFLUSH(10, 288)
356     CFLUSH(10, 320)
357     CFLUSH(10, 352)
358     CFLUSH(10, 384)
359     CFLUSH(10, 416)
360     CFLUSH(10, 448)
361     CFLUSH(10, 480)
362     CFLUSH(10, 512)
363     CFLUSH(10, 544)
364     CFLUSH(10, 576)
365     CFLUSH(10, 608)
366     CFLUSH(10, 640)
367     CFLUSH(10, 672)
368     CFLUSH(10, 704)
369     CFLUSH(10, 736)
370     CFLUSH(10, 768)
371     CFLUSH(10, 800)
372     CFLUSH(10, 832)
373     CFLUSH(10, 864)
374     CFLUSH(10, 896)
375     CFLUSH(10, 928)
376     CFLUSH(10, 960)
377     CFLUSH(10, 992)
378     addi a0, a0, 1024
379     blt a0, a1, flush_bulk_256_as_loop
380     ret
381
382 #endif /* !DOXYGEN */
383
384 /*EOF*****

```

8.13 pn_cache/pn_cache_RV32I_32.S File Reference

Contains RV32I assembly implementations of assembly functions called in [pn_cache.c](#) for cache line size 32.

Functions

- void [enable_cache_as](#) (void)
Enables cache.
 - void [disable_cache_as](#) (void)
Disables cache.
 - unsigned int [cache_banks_as](#) (void)
Returns number of cache banks.
 - unsigned int [cache_sets_as](#) (void)
Returns number of cache sets.
 - unsigned int [mem_size_as](#) (void)
Reads memory size.
-
- void [invalidate_32_as](#) (unsigned int addr, unsigned int endaddr)

- Invalidates single cache lines until end address is reached.*
- void [invalidate_bulk_32_as](#) (unsigned int addr, unsigned int endaddr)
- Invalidates 32 cache lines at once until end address is reached.*
- void [writeback_32_as](#) (unsigned int addr, unsigned int endaddr)
- Writes back single cache lines until end address is reached.*
- void [writeback_bulk_32_as](#) (unsigned int addr, unsigned int endaddr)
- Writes back 32 cache lines at once until end address is reached.*
- void [flush_32_as](#) (unsigned int addr, unsigned int endaddr)
- Flushes single cache lines until end address is reached.*
- void [flush_bulk_32_as](#) (unsigned int addr, unsigned int endaddr)
- Flushes 32 cache lines at once until end address is reached.*

8.13.1 Detailed Description

Contains RV32I assembly implementations of assembly functions called in [pn_cache.c](#) for cache line size 32.

This file (for this specific architecture, others may do it differently) was created by [pn_cache_RV32I_buildscript.py](#) while compiling (automatically called by [Makefile](#)), as are the other files with the pattern `pn_cache_RV32I_*.S` for name. The reasons for this are described in the [pn_cache_RV32I_buildscript.py](#) documentation itself.

The code shown below was created with a cache line size of 32 Bit.

So far, the following cache line sizes are available (in bits): 32 64 128 256 512 1024 2048

Contains implementations of:

- [enable_cache_as\(\)](#)
- [disable_cache_as\(\)](#)
- [cache_banks_as\(\)](#)
- [cache_sets_as\(\)](#)
- [mem_size_as\(\)](#)
- [invalidate_32_as\(\)](#)
- [invalidate_bulk_32_as\(\)](#)
- [writeback_32_as\(\)](#)
- [writeback_bulk_32_as\(\)](#)
- [flush_32_as\(\)](#)
- [flush_bulk_32_as\(\)](#)

```

1 /*
2  * I was automatically generated by pn_cache_RV32I_buildscript.py,
3  * invoked by libparanut Makefile -> No edits!
4  */
5
43 /*
44  * Put in here so Doxygen will know that it is implemented in this
45  * file. Sadly, Doxygen has no built in assembly interpreter, so we
46  * are stuck with this.
47  */
48
49 #ifndef DOXYGEN
50
65     void enable\_cache\_as(void) {}

```

```

66
72 void disable_cache_as(void) {}
73
81 unsigned int cache_banks_as(void) {}
82
90 unsigned int cache_sets_as(void) {}
91
99 unsigned int mem_size_as(void) {}
100
109 void invalidate_32_as(unsigned int addr, unsigned int endaddr) {}
110
119 void invalidate_bulk_32_as(unsigned int addr, unsigned int endaddr) {}
120
129 void writeback_32_as(unsigned int addr, unsigned int endaddr) {}
130
139 void writeback_bulk_32_as(unsigned int addr, unsigned int endaddr) {}
140
149 void flush_32_as(unsigned int addr, unsigned int endaddr) {}
150
159 void flush_bulk_32_as(unsigned int addr, unsigned int endaddr) {}
160
169 #endif /* DOXYGEN */
170
171 /*Header*****
172
173 #ifndef DOXYGEN
174
175 .text                /* enter text section                */
176 .align 2             /* align Code to 2^2 Bytes                */
177
178 /* declare labels in here to be global */
179 .globl enable_cache_as
180 .globl disable_cache_as
181 .globl cache_banks_as
182 .globl cache_sets_as
183 .globl mem_size_as
184 .globl invalidate_32_as
185 .globl invalidate_bulk_32_as
186 .globl writeback_32_as
187 .globl writeback_bulk_32_as
188 .globl flush_32_as
189 .globl flush_bulk_32_as
190
191 /* ParaNut Custom Registers and Instructions */
192 #include "custom_RV32I.S"
193
194 /*Functions*****
195
196 enable_cache_as:
197     fence
198     li    t0,      3
199     csrrs pncache, t0
200     ret
201
202 /*-----*/
203
204 disable_cache_as:
205     fence
206     li    t0,      0
207     csrws pncache, t0
208     ret
209
210 /*-----*/
211
212 cache_banks_as:
213     csrr a0,      pncacheinfo
214     srli a0,      a0,      8
215     ret
216
217 /*-----*/
218
219 cache_sets_as:
220     csrr a0,      pncachesets
221     ret
222
223 /*-----*/
224
225 mem_size_as:
226     csrr a0,      pnmemsize
227     ret
228
229 /*-----*/
230
231 invalidate_32_as:
232     fence
233 invalidate_32_as_loop:
234     CINV(10, 0x000)

```

```
235     addi a0, a0, 4
236     blt a0, a1, invalidate_32_as_loop
237     ret
238
239 /*-----*/
240
241 invalidate_bulk_32_as:
242     fence
243 invalidate_bulk_32_as_loop:
244     CINV(10, 0)
245     CINV(10, 4)
246     CINV(10, 8)
247     CINV(10, 12)
248     CINV(10, 16)
249     CINV(10, 20)
250     CINV(10, 24)
251     CINV(10, 28)
252     CINV(10, 32)
253     CINV(10, 36)
254     CINV(10, 40)
255     CINV(10, 44)
256     CINV(10, 48)
257     CINV(10, 52)
258     CINV(10, 56)
259     CINV(10, 60)
260     CINV(10, 64)
261     CINV(10, 68)
262     CINV(10, 72)
263     CINV(10, 76)
264     CINV(10, 80)
265     CINV(10, 84)
266     CINV(10, 88)
267     CINV(10, 92)
268     CINV(10, 96)
269     CINV(10, 100)
270     CINV(10, 104)
271     CINV(10, 108)
272     CINV(10, 112)
273     CINV(10, 116)
274     CINV(10, 120)
275     CINV(10, 124)
276     addi a0, a0, 128
277     blt a0, a1, invalidate_bulk_32_as_loop
278     ret
279
280 /*-----*/
281
282 writeback_32_as:
283     fence
284 writeback_32_as_loop:
285     CWB(10, 0x000)
286     addi a0, a0, 4
287     blt a0, a1, writeback_32_as_loop
288     ret
289
290 /*-----*/
291
292 writeback_bulk_32_as:
293     fence
294 writeback_bulk_32_as_loop:
295     CWB(10, 0)
296     CWB(10, 4)
297     CWB(10, 8)
298     CWB(10, 12)
299     CWB(10, 16)
300     CWB(10, 20)
301     CWB(10, 24)
302     CWB(10, 28)
303     CWB(10, 32)
304     CWB(10, 36)
305     CWB(10, 40)
306     CWB(10, 44)
307     CWB(10, 48)
308     CWB(10, 52)
309     CWB(10, 56)
310     CWB(10, 60)
311     CWB(10, 64)
312     CWB(10, 68)
313     CWB(10, 72)
314     CWB(10, 76)
315     CWB(10, 80)
316     CWB(10, 84)
317     CWB(10, 88)
318     CWB(10, 92)
319     CWB(10, 96)
320     CWB(10, 100)
321     CWB(10, 104)
```

```

322     CWB(10, 108)
323     CWB(10, 112)
324     CWB(10, 116)
325     CWB(10, 120)
326     CWB(10, 124)
327     addi a0, a0, 128
328     blt  a0, a1, writeback_bulk_32_as_loop
329     ret
330
331 /*-----*/
332
333 flush_32_as:
334     fence
335 flush_32_as_loop:
336     CFLUSH(10, 0x000)
337     addi a0, a0, 4
338     blt  a0, a1, flush_32_as_loop
339     ret
340
341 /*-----*/
342
343 flush_bulk_32_as:
344     fence
345 flush_bulk_32_as_loop:
346     CFLUSH(10, 0)
347     CFLUSH(10, 4)
348     CFLUSH(10, 8)
349     CFLUSH(10, 12)
350     CFLUSH(10, 16)
351     CFLUSH(10, 20)
352     CFLUSH(10, 24)
353     CFLUSH(10, 28)
354     CFLUSH(10, 32)
355     CFLUSH(10, 36)
356     CFLUSH(10, 40)
357     CFLUSH(10, 44)
358     CFLUSH(10, 48)
359     CFLUSH(10, 52)
360     CFLUSH(10, 56)
361     CFLUSH(10, 60)
362     CFLUSH(10, 64)
363     CFLUSH(10, 68)
364     CFLUSH(10, 72)
365     CFLUSH(10, 76)
366     CFLUSH(10, 80)
367     CFLUSH(10, 84)
368     CFLUSH(10, 88)
369     CFLUSH(10, 92)
370     CFLUSH(10, 96)
371     CFLUSH(10, 100)
372     CFLUSH(10, 104)
373     CFLUSH(10, 108)
374     CFLUSH(10, 112)
375     CFLUSH(10, 116)
376     CFLUSH(10, 120)
377     CFLUSH(10, 124)
378     addi a0, a0, 128
379     blt  a0, a1, flush_bulk_32_as_loop
380     ret
381
382 #endif /* !DOXYGEN */
383
384 /*EOF******/

```

8.14 pn_cache/pn_cache_RV32I_512.S File Reference

Contains RV32I assembly implementations of assembly functions called in [pn_cache.c](#) for cache line size 512.

Functions

- void [enable_cache_as](#) (void)

- Enables cache.*

 - void [disable_cache_as](#) (void)
 - Disables cache.*

 - unsigned int [cache_banks_as](#) (void)
 - Returns number of cache banks.*

 - unsigned int [cache_sets_as](#) (void)
 - Returns number of cache sets.*

 - unsigned int [mem_size_as](#) (void)
 - Reads memory size.*
-
- void [invalidate_512_as](#) (unsigned int addr, unsigned int endaddr)

Invalidates single cache lines until end address is reached.
 - void [invalidate_bulk_512_as](#) (unsigned int addr, unsigned int endaddr)

Invalidates 32 cache lines at once until end address is reached.
 - void [writeback_512_as](#) (unsigned int addr, unsigned int endaddr)

Writes back single cache lines until end address is reached.
 - void [writeback_bulk_512_as](#) (unsigned int addr, unsigned int endaddr)

Writes back 32 cache lines at once until end address is reached.
 - void [flush_512_as](#) (unsigned int addr, unsigned int endaddr)

Flushes single cache lines until end address is reached.
 - void [flush_bulk_512_as](#) (unsigned int addr, unsigned int endaddr)

Flushes 32 cache lines at once until end address is reached.

8.14.1 Detailed Description

Contains RV32I assembly implementations of assembly functions called in [pn_cache.c](#) for cache line size 512.

This file (for this specific architecture, others may do it differently) was created by [pn_cache_RV32I_buildscript.py](#) while compiling (automatically called by [Makefile](#)), as are the other files with the pattern `pn_cache_RV32I_*.S` for name. The reasons for this are described in the [pn_cache_RV32I_buildscript.py](#) documentation itself.

The code shown below was created with a cache line size of 512 Bit.

So far, the following cache line sizes are available (in bits): 32 64 128 256 512 1024 2048

Contains implementations of:

- [enable_cache_as\(\)](#)
- [disable_cache_as\(\)](#)
- [cache_banks_as\(\)](#)
- [cache_sets_as\(\)](#)
- [mem_size_as\(\)](#)
- [invalidate_512_as\(\)](#)
- [invalidate_bulk_512_as\(\)](#)
- [writeback_512_as\(\)](#)

- [writeback_bulk_512_as\(\)](#)
- [flush_512_as\(\)](#)
- [flush_bulk_512_as\(\)](#)

```

1 /*
2  * I was automatically generated by pn_cache_RV32I_buildscript.py,
3  * invoked by libparanut Makefile -> No edits!
4  */
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43 /*
44  * Put in here so Doxygen will know that it is implemented in this
45  * file. Sadly, Doxygen has no built in assembly interpreter, so we
46  * are stuck with this.
47  */
48
49 #ifdef DOXYGEN
50
51     void enable_cache_as(void) {}
52
53     void disable_cache_as(void) {}
54
55     unsigned int cache_banks_as(void) {}
56
57     unsigned int cache_sets_as(void) {}
58
59     unsigned int mem_size_as(void) {}
60
61     void invalidate_512_as(unsigned int addr, unsigned int endaddr) {}
62
63     void invalidate_bulk_512_as(unsigned int addr, unsigned int endaddr) {}
64
65     void writeback_512_as(unsigned int addr, unsigned int endaddr) {}
66
67     void writeback_bulk_512_as(unsigned int addr, unsigned int endaddr) {}
68
69     void flush_512_as(unsigned int addr, unsigned int endaddr) {}
70
71     void flush_bulk_512_as(unsigned int addr, unsigned int endaddr) {}
72
73 #endif /* DOXYGEN */
74
75 /*Header*****
76
77 #ifndef DOXYGEN
78
79     .text                                /* enter text section          */
80     .align 2                            /* align Code to 2^2 Bytes        */
81
82     /* declare labels in here to be global */
83     .globl enable_cache_as
84     .globl disable_cache_as
85     .globl cache_banks_as
86     .globl cache_sets_as
87     .globl mem_size_as
88     .globl invalidate_512_as
89     .globl invalidate_bulk_512_as
90     .globl writeback_512_as
91     .globl writeback_bulk_512_as
92     .globl flush_512_as
93     .globl flush_bulk_512_as
94
95     /* ParaNut Custom Registers and Instructions */
96     #include "custom_RV32I.S"
97
98 /*Functions*****
99
100 enable_cache_as:
101     fence
102     li    t0,    3
103     csrs  pncache, t0
104     ret
105
106 -----*/
107
108 disable_cache_as:
109     fence
110     li    t0,    0
111     csrw  pncache, t0
112     ret
113
114 -----*/
115
116 cache_banks_as:

```



```

213     csrr a0,      pncacheinfo
214     srli a0,      a0,      8
215     ret
216
217 /*-----*/
218
219 cache_sets_as:
220     csrr a0,      pncachesets
221     ret
222
223 /*-----*/
224
225 mem_size_as:
226     csrr a0,      pnmemsize
227     ret
228
229 /*-----*/
230
231 invalidate_512_as:
232     fence
233 invalidate_512_as_loop:
234     CINV(10, 0x000)
235     addi a0, a0, 64
236     blt a0, a1, invalidate_512_as_loop
237     ret
238
239 /*-----*/
240
241 invalidate_bulk_512_as:
242     fence
243 li t0, 2048
244 invalidate_bulk_512_as_loop:
245     CINV(10, 0)
246     CINV(10, 64)
247     CINV(10, 128)
248     CINV(10, 192)
249     CINV(10, 256)
250     CINV(10, 320)
251     CINV(10, 384)
252     CINV(10, 448)
253     CINV(10, 512)
254     CINV(10, 576)
255     CINV(10, 640)
256     CINV(10, 704)
257     CINV(10, 768)
258     CINV(10, 832)
259     CINV(10, 896)
260     CINV(10, 960)
261     CINV(10, 1024)
262     CINV(10, 1088)
263     CINV(10, 1152)
264     CINV(10, 1216)
265     CINV(10, 1280)
266     CINV(10, 1344)
267     CINV(10, 1408)
268     CINV(10, 1472)
269     CINV(10, 1536)
270     CINV(10, 1600)
271     CINV(10, 1664)
272     CINV(10, 1728)
273     CINV(10, 1792)
274     CINV(10, 1856)
275     CINV(10, 1920)
276     CINV(10, 1984)
277     add a0, a0, t0
278     blt a0, a1, invalidate_bulk_512_as_loop
279     ret
280
281 /*-----*/
282
283 writeback_512_as:
284     fence
285 writeback_512_as_loop:
286     CWB(10, 0x000)
287     addi a0, a0, 64
288     blt a0, a1, writeback_512_as_loop
289     ret
290
291 /*-----*/
292
293 writeback_bulk_512_as:
294     fence
295 li t0, 2048
296 writeback_bulk_512_as_loop:
297     CWB(10, 0)
298     CWB(10, 64)
299     CWB(10, 128)

```

```

300    CWB(10, 192)
301    CWB(10, 256)
302    CWB(10, 320)
303    CWB(10, 384)
304    CWB(10, 448)
305    CWB(10, 512)
306    CWB(10, 576)
307    CWB(10, 640)
308    CWB(10, 704)
309    CWB(10, 768)
310    CWB(10, 832)
311    CWB(10, 896)
312    CWB(10, 960)
313    CWB(10, 1024)
314    CWB(10, 1088)
315    CWB(10, 1152)
316    CWB(10, 1216)
317    CWB(10, 1280)
318    CWB(10, 1344)
319    CWB(10, 1408)
320    CWB(10, 1472)
321    CWB(10, 1536)
322    CWB(10, 1600)
323    CWB(10, 1664)
324    CWB(10, 1728)
325    CWB(10, 1792)
326    CWB(10, 1856)
327    CWB(10, 1920)
328    CWB(10, 1984)
329    add a0, a0, t0
330    blt  a0, a1, writeback_bulk_512_as_loop
331    ret
332
333    /*-----*/
334
335    flush_512_as:
336        fence
337    flush_512_as_loop:
338        CFLUSH(10, 0x000)
339        addi a0, a0, 64
340        blt  a0, a1, flush_512_as_loop
341        ret
342
343    /*-----*/
344
345    flush_bulk_512_as:
346        fence
347    li t0, 2048
348    flush_bulk_512_as_loop:
349        CFLUSH(10, 0)
350        CFLUSH(10, 64)
351        CFLUSH(10, 128)
352        CFLUSH(10, 192)
353        CFLUSH(10, 256)
354        CFLUSH(10, 320)
355        CFLUSH(10, 384)
356        CFLUSH(10, 448)
357        CFLUSH(10, 512)
358        CFLUSH(10, 576)
359        CFLUSH(10, 640)
360        CFLUSH(10, 704)
361        CFLUSH(10, 768)
362        CFLUSH(10, 832)
363        CFLUSH(10, 896)
364        CFLUSH(10, 960)
365        CFLUSH(10, 1024)
366        CFLUSH(10, 1088)
367        CFLUSH(10, 1152)
368        CFLUSH(10, 1216)
369        CFLUSH(10, 1280)
370        CFLUSH(10, 1344)
371        CFLUSH(10, 1408)
372        CFLUSH(10, 1472)
373        CFLUSH(10, 1536)
374        CFLUSH(10, 1600)
375        CFLUSH(10, 1664)
376        CFLUSH(10, 1728)
377        CFLUSH(10, 1792)
378        CFLUSH(10, 1856)
379        CFLUSH(10, 1920)
380        CFLUSH(10, 1984)
381        add a0, a0, t0
382        blt  a0, a1, flush_bulk_512_as_loop
383        ret
384
385    #endif /* !DOXYGEN */
386

```

387 /*EOF******/

8.15 pn_cache/pn_cache_RV32I_64.S File Reference

Contains RV32I assembly implementations of assembly functions called in [pn_cache.c](#) for cache line size 64.

Functions

- void [enable_cache_as](#) (void)
Enables cache.
 - void [disable_cache_as](#) (void)
Disables cache.
 - unsigned int [cache_banks_as](#) (void)
Returns number of cache banks.
 - unsigned int [cache_sets_as](#) (void)
Returns number of cache sets.
 - unsigned int [mem_size_as](#) (void)
Reads memory size.
-
- void [invalidate_64_as](#) (unsigned int addr, unsigned int endaddr)
Invalidates single cache lines until end address is reached.
 - void [invalidate_bulk_64_as](#) (unsigned int addr, unsigned int endaddr)
Invalidates 32 cache lines at once until end address is reached.
 - void [writeback_64_as](#) (unsigned int addr, unsigned int endaddr)
Writes back single cache lines until end address is reached.
 - void [writeback_bulk_64_as](#) (unsigned int addr, unsigned int endaddr)
Writes back 32 cache lines at once until end address is reached.
 - void [flush_64_as](#) (unsigned int addr, unsigned int endaddr)
Flushes single cache lines until end address is reached.
 - void [flush_bulk_64_as](#) (unsigned int addr, unsigned int endaddr)
Flushes 32 cache lines at once until end address is reached.

8.15.1 Detailed Description

Contains RV32I assembly implementations of assembly functions called in [pn_cache.c](#) for cache line size 64.

This file (for this specific architecture, others may do it differently) was created by [pn_cache_RV32I_buildscript.py](#) while compiling (automatically called by [Makefile](#)), as are the other files with the pattern `pn_cache_RV32I_*.S` for name. The reasons for this are described in the [pn_cache_RV32I_buildscript.py](#) documentation itself.

The code shown below was created with a cache line size of 64 Bit.

So far, the following cache line sizes are available (in bits): 32 64 128 256 512 1024 2048

Contains implementations of:

- [enable_cache_as\(\)](#)
- [disable_cache_as\(\)](#)
- [cache_banks_as\(\)](#)
- [cache_sets_as\(\)](#)
- [mem_size_as\(\)](#)
- [invalidate_64_as\(\)](#)
- [invalidate_bulk_64_as\(\)](#)
- [writeback_64_as\(\)](#)
- [writeback_bulk_64_as\(\)](#)
- [flush_64_as\(\)](#)
- [flush_bulk_64_as\(\)](#)

```

1  /*
2  * I was automatically generated by pn_cache_RV32I_buildscript.py,
3  * invoked by libparanut Makefile -> No edits!
4  */
5
43 /*
44 * Put in here so Doxygen will know that it is implemented in this
45 * file. Sadly, Doxygen has no built in assembly interpreter, so we
46 * are stuck with this.
47 */
48
49 #ifdef DOXYGEN
50     void enable_cache_as(void) {}
51
52     void disable_cache_as(void) {}
53
54     unsigned int cache_banks_as(void) {}
55
56     unsigned int cache_sets_as(void) {}
57
58     unsigned int mem_size_as(void) {}
59
60     void invalidate_64_as(unsigned int addr, unsigned int endaddr) {}
61
62     void invalidate_bulk_64_as(unsigned int addr, unsigned int endaddr) {}
63
64     void writeback_64_as(unsigned int addr, unsigned int endaddr) {}
65
66     void writeback_bulk_64_as(unsigned int addr, unsigned int endaddr) {}
67
68     void flush_64_as(unsigned int addr, unsigned int endaddr) {}
69
70     void flush_bulk_64_as(unsigned int addr, unsigned int endaddr) {}
71
72 #endif /* DOXYGEN */
73

```

```

171 /*Header******/
172
173 #ifndef DOXYGEN
174
175 .text                /* enter text section */
176 .align 2            /* align Code to 2^2 Bytes */
177
178 /* declare labels in here to be global */
179 .globl enable_cache_as
180 .globl disable_cache_as
181 .globl cache_banks_as
182 .globl cache_sets_as
183 .globl mem_size_as
184 .globl invalidate_64_as
185 .globl invalidate_bulk_64_as
186 .globl writeback_64_as
187 .globl writeback_bulk_64_as
188 .globl flush_64_as
189 .globl flush_bulk_64_as
190
191 /* ParaNUT Custom Registers and Instructions */
192 #include "custom_RV32I.S"
193
194 /*Functions******/
195
196 enable_cache_as:
197     fence
198     li    t0,      3
199     csrs  pncache, t0
200     ret
201
202 /*-----*/
203
204 disable_cache_as:
205     fence
206     li    t0,      0
207     csrw  pncache, t0
208     ret
209
210 /*-----*/
211
212 cache_banks_as:
213     csrr  a0,      pncacheinfo
214     srli  a0,      a0,      8
215     ret
216
217 /*-----*/
218
219 cache_sets_as:
220     csrr  a0,      pncachesets
221     ret
222
223 /*-----*/
224
225 mem_size_as:
226     csrr  a0,      pnmemsize
227     ret
228
229 /*-----*/
230
231 invalidate_64_as:
232     fence
233 invalidate_64_as_loop:
234     CINV(10, 0x000)
235     addi  a0, a0, 8
236     blt  a0, a1, invalidate_64_as_loop
237     ret
238
239 /*-----*/
240
241 invalidate_bulk_64_as:
242     fence
243 invalidate_bulk_64_as_loop:
244     CINV(10, 0)
245     CINV(10, 8)
246     CINV(10, 16)
247     CINV(10, 24)
248     CINV(10, 32)
249     CINV(10, 40)
250     CINV(10, 48)
251     CINV(10, 56)
252     CINV(10, 64)
253     CINV(10, 72)
254     CINV(10, 80)
255     CINV(10, 88)
256     CINV(10, 96)
257     CINV(10, 104)

```

```
258     CINV(10, 112)
259     CINV(10, 120)
260     CINV(10, 128)
261     CINV(10, 136)
262     CINV(10, 144)
263     CINV(10, 152)
264     CINV(10, 160)
265     CINV(10, 168)
266     CINV(10, 176)
267     CINV(10, 184)
268     CINV(10, 192)
269     CINV(10, 200)
270     CINV(10, 208)
271     CINV(10, 216)
272     CINV(10, 224)
273     CINV(10, 232)
274     CINV(10, 240)
275     CINV(10, 248)
276     addi a0, a0, 256
277     blt a0, a1, invalidate_bulk_64_as_loop
278     ret
279
280 /*-----*/
281
282 writeback_64_as:
283     fence
284 writeback_64_as_loop:
285     CWB(10, 0x000)
286     addi a0, a0, 8
287     blt a0, a1, writeback_64_as_loop
288     ret
289
290 /*-----*/
291
292 writeback_bulk_64_as:
293     fence
294 writeback_bulk_64_as_loop:
295     CWB(10, 0)
296     CWB(10, 8)
297     CWB(10, 16)
298     CWB(10, 24)
299     CWB(10, 32)
300     CWB(10, 40)
301     CWB(10, 48)
302     CWB(10, 56)
303     CWB(10, 64)
304     CWB(10, 72)
305     CWB(10, 80)
306     CWB(10, 88)
307     CWB(10, 96)
308     CWB(10, 104)
309     CWB(10, 112)
310     CWB(10, 120)
311     CWB(10, 128)
312     CWB(10, 136)
313     CWB(10, 144)
314     CWB(10, 152)
315     CWB(10, 160)
316     CWB(10, 168)
317     CWB(10, 176)
318     CWB(10, 184)
319     CWB(10, 192)
320     CWB(10, 200)
321     CWB(10, 208)
322     CWB(10, 216)
323     CWB(10, 224)
324     CWB(10, 232)
325     CWB(10, 240)
326     CWB(10, 248)
327     addi a0, a0, 256
328     blt a0, a1, writeback_bulk_64_as_loop
329     ret
330
331 /*-----*/
332
333 flush_64_as:
334     fence
335 flush_64_as_loop:
336     CFLUSH(10, 0x000)
337     addi a0, a0, 8
338     blt a0, a1, flush_64_as_loop
339     ret
340
341 /*-----*/
342
343 flush_bulk_64_as:
344     fence
```

```

345 flush_bulk_64_as_loop:
346     CFLUSH(10, 0)
347     CFLUSH(10, 8)
348     CFLUSH(10, 16)
349     CFLUSH(10, 24)
350     CFLUSH(10, 32)
351     CFLUSH(10, 40)
352     CFLUSH(10, 48)
353     CFLUSH(10, 56)
354     CFLUSH(10, 64)
355     CFLUSH(10, 72)
356     CFLUSH(10, 80)
357     CFLUSH(10, 88)
358     CFLUSH(10, 96)
359     CFLUSH(10, 104)
360     CFLUSH(10, 112)
361     CFLUSH(10, 120)
362     CFLUSH(10, 128)
363     CFLUSH(10, 136)
364     CFLUSH(10, 144)
365     CFLUSH(10, 152)
366     CFLUSH(10, 160)
367     CFLUSH(10, 168)
368     CFLUSH(10, 176)
369     CFLUSH(10, 184)
370     CFLUSH(10, 192)
371     CFLUSH(10, 200)
372     CFLUSH(10, 208)
373     CFLUSH(10, 216)
374     CFLUSH(10, 224)
375     CFLUSH(10, 232)
376     CFLUSH(10, 240)
377     CFLUSH(10, 248)
378     addi a0, a0, 256
379     blt  a0, a1, flush_bulk_64_as_loop
380     ret
381
382 #endif /* !DOXYGEN */
383
384 /*EOF*****

```

8.16 pn_cache/pn_cache_RV32I_auto.S File Reference

Contains RV32I assembly implementations of assembly functions called in [pn_cache.c](#) for all cache line sizes.

Functions

- void [enable_cache_as](#) (void)
Enables cache.
- void [disable_cache_as](#) (void)
Disables cache.
- unsigned int [cache_banks_as](#) (void)
Returns number of cache banks.
- unsigned int [cache_sets_as](#) (void)
Returns number of cache sets.
- unsigned int [mem_size_as](#) (void)
Reads memory size.
- void [invalidate_1024_as](#) (unsigned int addr, unsigned int endaddr)
Invalidates single cache lines until end address is reached.
- void [invalidate_bulk_1024_as](#) (unsigned int addr, unsigned int endaddr)
Invalidates 32 cache lines at once until end address is reached.
- void [writeback_1024_as](#) (unsigned int addr, unsigned int endaddr)

Writes back single cache lines until end address is reached.

- void [writeback_bulk_1024_as](#) (unsigned int addr, unsigned int endaddr)

Writes back 32 cache lines at once until end address is reached.

- void [flush_1024_as](#) (unsigned int addr, unsigned int endaddr)

Flushes single cache lines until end address is reached.

- void [flush_bulk_1024_as](#) (unsigned int addr, unsigned int endaddr)

Flushes 32 cache lines at once until end address is reached.

- void [invalidate_32_as](#) (unsigned int addr, unsigned int endaddr)

Invalidates single cache lines until end address is reached.

- void [invalidate_bulk_32_as](#) (unsigned int addr, unsigned int endaddr)

Invalidates 32 cache lines at once until end address is reached.

- void [writeback_32_as](#) (unsigned int addr, unsigned int endaddr)

Writes back single cache lines until end address is reached.

- void [writeback_bulk_32_as](#) (unsigned int addr, unsigned int endaddr)

Writes back 32 cache lines at once until end address is reached.

- void [flush_32_as](#) (unsigned int addr, unsigned int endaddr)

Flushes single cache lines until end address is reached.

- void [flush_bulk_32_as](#) (unsigned int addr, unsigned int endaddr)

Flushes 32 cache lines at once until end address is reached.

- void [invalidate_64_as](#) (unsigned int addr, unsigned int endaddr)

Invalidates single cache lines until end address is reached.

- void [invalidate_bulk_64_as](#) (unsigned int addr, unsigned int endaddr)

Invalidates 32 cache lines at once until end address is reached.

- void [writeback_64_as](#) (unsigned int addr, unsigned int endaddr)

Writes back single cache lines until end address is reached.

- void [writeback_bulk_64_as](#) (unsigned int addr, unsigned int endaddr)

Writes back 32 cache lines at once until end address is reached.

- void [flush_64_as](#) (unsigned int addr, unsigned int endaddr)

Flushes single cache lines until end address is reached.

- void [flush_bulk_64_as](#) (unsigned int addr, unsigned int endaddr)

Flushes 32 cache lines at once until end address is reached.

- void [invalidate_128_as](#) (unsigned int addr, unsigned int endaddr)

Invalidates single cache lines until end address is reached.

- void [invalidate_bulk_128_as](#) (unsigned int addr, unsigned int endaddr)

Invalidates 32 cache lines at once until end address is reached.

- void [writeback_128_as](#) (unsigned int addr, unsigned int endaddr)

Writes back single cache lines until end address is reached.

- void [writeback_bulk_128_as](#) (unsigned int addr, unsigned int endaddr)

Writes back 32 cache lines at once until end address is reached.

- void [flush_128_as](#) (unsigned int addr, unsigned int endaddr)
Flushes single cache lines until end address is reached.
- void [flush_bulk_128_as](#) (unsigned int addr, unsigned int endaddr)
Flushes 32 cache lines at once until end address is reached.

- void [invalidate_256_as](#) (unsigned int addr, unsigned int endaddr)
Invalidates single cache lines until end address is reached.
- void [invalidate_bulk_256_as](#) (unsigned int addr, unsigned int endaddr)
Invalidates 32 cache lines at once until end address is reached.
- void [writeback_256_as](#) (unsigned int addr, unsigned int endaddr)
Writes back single cache lines until end address is reached.
- void [writeback_bulk_256_as](#) (unsigned int addr, unsigned int endaddr)
Writes back 32 cache lines at once until end address is reached.
- void [flush_256_as](#) (unsigned int addr, unsigned int endaddr)
Flushes single cache lines until end address is reached.
- void [flush_bulk_256_as](#) (unsigned int addr, unsigned int endaddr)
Flushes 32 cache lines at once until end address is reached.

- void [invalidate_512_as](#) (unsigned int addr, unsigned int endaddr)
Invalidates single cache lines until end address is reached.
- void [invalidate_bulk_512_as](#) (unsigned int addr, unsigned int endaddr)
Invalidates 32 cache lines at once until end address is reached.
- void [writeback_512_as](#) (unsigned int addr, unsigned int endaddr)
Writes back single cache lines until end address is reached.
- void [writeback_bulk_512_as](#) (unsigned int addr, unsigned int endaddr)
Writes back 32 cache lines at once until end address is reached.
- void [flush_512_as](#) (unsigned int addr, unsigned int endaddr)
Flushes single cache lines until end address is reached.
- void [flush_bulk_512_as](#) (unsigned int addr, unsigned int endaddr)
Flushes 32 cache lines at once until end address is reached.

- void [invalidate_2048_as](#) (unsigned int addr, unsigned int endaddr)
Invalidates single cache lines until end address is reached.
- void [invalidate_bulk_2048_as](#) (unsigned int addr, unsigned int endaddr)
Invalidates 32 cache lines at once until end address is reached.
- void [writeback_2048_as](#) (unsigned int addr, unsigned int endaddr)
Writes back single cache lines until end address is reached.
- void [writeback_bulk_2048_as](#) (unsigned int addr, unsigned int endaddr)
Writes back 32 cache lines at once until end address is reached.
- void [flush_2048_as](#) (unsigned int addr, unsigned int endaddr)
Flushes single cache lines until end address is reached.
- void [flush_bulk_2048_as](#) (unsigned int addr, unsigned int endaddr)
Flushes 32 cache lines at once until end address is reached.

8.16.1 Detailed Description

Contains RV32I assembly implementations of assembly functions called in [pn_cache.c](#) for all cache line sizes.

This file (for this specific architecture, others may do it differently) was created by [pn_cache_RV32I_buildscript.py](#) while compiling (automatically called by [Makefile](#)), as are the other files with the pattern `pn_cache_RV32I_*.S` for name. The reasons for this are described in the [pn_cache_RV32I_buildscript.py](#) documentation itself.

The code shown below was created for all cache line sizes.

So far, the following cache line sizes are available (in bits): 32 64 128 256 512 1024 2048

Contains implementations of:

- [enable_cache_as\(\)](#)
- [disable_cache_as\(\)](#)
- [cache_banks_as\(\)](#)
- [cache_sets_as\(\)](#)
- [mem_size_as\(\)](#)
- [invalidate_32_as\(\)](#)
- [invalidate_bulk_32_as\(\)](#)
- [writeback_32_as\(\)](#)
- [writeback_bulk_32_as\(\)](#)
- [flush_32_as\(\)](#)
- [flush_bulk_32_as\(\)](#)
- [invalidate_64_as\(\)](#)
- [invalidate_bulk_64_as\(\)](#)
- [writeback_64_as\(\)](#)
- [writeback_bulk_64_as\(\)](#)
- [flush_64_as\(\)](#)
- [flush_bulk_64_as\(\)](#)
- [invalidate_128_as\(\)](#)
- [invalidate_bulk_128_as\(\)](#)
- [writeback_128_as\(\)](#)
- [writeback_bulk_128_as\(\)](#)
- [flush_128_as\(\)](#)
- [flush_bulk_128_as\(\)](#)
- [invalidate_256_as\(\)](#)
- [invalidate_bulk_256_as\(\)](#)
- [writeback_256_as\(\)](#)
- [writeback_bulk_256_as\(\)](#)

- [flush_256_as\(\)](#)
- [flush_bulk_256_as\(\)](#)
- [invalidate_512_as\(\)](#)
- [invalidate_bulk_512_as\(\)](#)
- [writeback_512_as\(\)](#)
- [writeback_bulk_512_as\(\)](#)
- [flush_512_as\(\)](#)
- [flush_bulk_512_as\(\)](#)
- [invalidate_1024_as\(\)](#)
- [invalidate_bulk_1024_as\(\)](#)
- [writeback_1024_as\(\)](#)
- [writeback_bulk_1024_as\(\)](#)
- [flush_1024_as\(\)](#)
- [flush_bulk_1024_as\(\)](#)
- [invalidate_2048_as\(\)](#)
- [invalidate_bulk_2048_as\(\)](#)
- [writeback_2048_as\(\)](#)
- [writeback_bulk_2048_as\(\)](#)
- [flush_2048_as\(\)](#)
- [flush_bulk_2048_as\(\)](#)

```

1 /*
2  * I was automatically generated by pn_cache_RV32I_buildscript.py,
3  * invoked by libparanut Makefile -> No edits!
4  */
5
79 /*
80  * Put in here so Doxygen will know that it is implemented in this
81  * file. Sadly, Doxygen has no built in assembly interpreter, so we
82  * are stuck with this.
83  */
84
85 #ifdef DOXYGEN
86
101     void enable_cache_as(void) {}
102
103     void disable_cache_as(void) {}
104
105     unsigned int cache_banks_as(void) {}
106
107     unsigned int cache_sets_as(void) {}
108
109     unsigned int mem_size_as(void) {}
110
111     void invalidate_32_as(unsigned int addr, unsigned int endaddr) {}
112
113     void invalidate_bulk_32_as(unsigned int addr, unsigned int endaddr) {}
114
115     void writeback_32_as(unsigned int addr, unsigned int endaddr) {}
116
117     void writeback_bulk_32_as(unsigned int addr, unsigned int endaddr) {}
118
119     void flush_32_as(unsigned int addr, unsigned int endaddr) {}
120
121     void flush_bulk_32_as(unsigned int addr, unsigned int endaddr) {}
122
123     void invalidate_64_as(unsigned int addr, unsigned int endaddr) {}
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206

```

```

215 void invalidate_bulk_64_as(unsigned int addr, unsigned int endaddr) {}
216
225 void writeback_64_as(unsigned int addr, unsigned int endaddr) {}
226
235 void writeback_bulk_64_as(unsigned int addr, unsigned int endaddr) {}
236
245 void flush_64_as(unsigned int addr, unsigned int endaddr) {}
246
255 void flush_bulk_64_as(unsigned int addr, unsigned int endaddr) {}
256
265 void invalidate_128_as(unsigned int addr, unsigned int endaddr) {}
266
275 void invalidate_bulk_128_as(unsigned int addr, unsigned int endaddr) {}
276
285 void writeback_128_as(unsigned int addr, unsigned int endaddr) {}
286
295 void writeback_bulk_128_as(unsigned int addr, unsigned int endaddr) {}
296
305 void flush_128_as(unsigned int addr, unsigned int endaddr) {}
306
315 void flush_bulk_128_as(unsigned int addr, unsigned int endaddr) {}
316
325 void invalidate_256_as(unsigned int addr, unsigned int endaddr) {}
326
335 void invalidate_bulk_256_as(unsigned int addr, unsigned int endaddr) {}
336
345 void writeback_256_as(unsigned int addr, unsigned int endaddr) {}
346
355 void writeback_bulk_256_as(unsigned int addr, unsigned int endaddr) {}
356
365 void flush_256_as(unsigned int addr, unsigned int endaddr) {}
366
375 void flush_bulk_256_as(unsigned int addr, unsigned int endaddr) {}
376
385 void invalidate_512_as(unsigned int addr, unsigned int endaddr) {}
386
395 void invalidate_bulk_512_as(unsigned int addr, unsigned int endaddr) {}
396
405 void writeback_512_as(unsigned int addr, unsigned int endaddr) {}
406
415 void writeback_bulk_512_as(unsigned int addr, unsigned int endaddr) {}
416
425 void flush_512_as(unsigned int addr, unsigned int endaddr) {}
426
435 void flush_bulk_512_as(unsigned int addr, unsigned int endaddr) {}
436
445 void invalidate_1024_as(unsigned int addr, unsigned int endaddr) {}
446
455 void invalidate_bulk_1024_as(unsigned int addr, unsigned int endaddr) {}
456
465 void writeback_1024_as(unsigned int addr, unsigned int endaddr) {}
466
475 void writeback_bulk_1024_as(unsigned int addr, unsigned int endaddr) {}
476
485 void flush_1024_as(unsigned int addr, unsigned int endaddr) {}
486
495 void flush_bulk_1024_as(unsigned int addr, unsigned int endaddr) {}
496
505 void invalidate_2048_as(unsigned int addr, unsigned int endaddr) {}
506
515 void invalidate_bulk_2048_as(unsigned int addr, unsigned int endaddr) {}
516
525 void writeback_2048_as(unsigned int addr, unsigned int endaddr) {}
526
535 void writeback_bulk_2048_as(unsigned int addr, unsigned int endaddr) {}
536
545 void flush_2048_as(unsigned int addr, unsigned int endaddr) {}
546
555 void flush_bulk_2048_as(unsigned int addr, unsigned int endaddr) {}
556
565 #endif /* DOXYGEN */
566
567 /**Header*****
568
569 #ifndef DOXYGEN
570
571 .text                /* enter text section                */
572 .align 2             /* align Code to 2^2 Bytes                */
573
574 /* declare labels in here to be global */
575 .globl enable_cache_as
576 .globl disable_cache_as
577 .globl cache_banks_as
578 .globl cache_sets_as
579 .globl mem_size_as
580 .globl invalidate_32_as
581 .globl invalidate_bulk_32_as

```

```

582 .globl writeback_32_as
583 .globl writeback_bulk_32_as
584 .globl flush_32_as
585 .globl flush_bulk_32_as
586 .globl invalidate_64_as
587 .globl invalidate_bulk_64_as
588 .globl writeback_64_as
589 .globl writeback_bulk_64_as
590 .globl flush_64_as
591 .globl flush_bulk_64_as
592 .globl invalidate_128_as
593 .globl invalidate_bulk_128_as
594 .globl writeback_128_as
595 .globl writeback_bulk_128_as
596 .globl flush_128_as
597 .globl flush_bulk_128_as
598 .globl invalidate_256_as
599 .globl invalidate_bulk_256_as
600 .globl writeback_256_as
601 .globl writeback_bulk_256_as
602 .globl flush_256_as
603 .globl flush_bulk_256_as
604 .globl invalidate_512_as
605 .globl invalidate_bulk_512_as
606 .globl writeback_512_as
607 .globl writeback_bulk_512_as
608 .globl flush_512_as
609 .globl flush_bulk_512_as
610 .globl invalidate_1024_as
611 .globl invalidate_bulk_1024_as
612 .globl writeback_1024_as
613 .globl writeback_bulk_1024_as
614 .globl flush_1024_as
615 .globl flush_bulk_1024_as
616 .globl invalidate_2048_as
617 .globl invalidate_bulk_2048_as
618 .globl writeback_2048_as
619 .globl writeback_bulk_2048_as
620 .globl flush_2048_as
621 .globl flush_bulk_2048_as
622
623 /* ParaNut Custom Registers and Instructions */
624 #include "custom_RV32I.S"
625
626 /*Functions*****/
627
628 enable_cache_as:
629     fence
630     li    t0,    3
631     csrs  pncache, t0
632     ret
633
634 /*-----*/
635
636 disable_cache_as:
637     fence
638     li    t0,    0
639     csrw  pncache, t0
640     ret
641
642 /*-----*/
643
644 cache_banks_as:
645     csrr  a0,    pncacheinfo
646     srli  a0,    a0,    8
647     ret
648
649 /*-----*/
650
651 cache_sets_as:
652     csrr  a0,    pncachesets
653     ret
654
655 /*-----*/
656
657 mem_size_as:
658     csrr  a0,    pnmemsize
659     ret
660
661 /*-----*/
662
663 invalidate_32_as:
664     fence
665 invalidate_32_as_loop:
666     CINV(10, 0x000)
667     addi  a0,  a0,  4
668     blt  a0,  a1,  invalidate_32_as_loop

```

```
669     ret
670
671  /*-----*/
672
673  invalidate_bulk_32_as:
674      fence
675  invalidate_bulk_32_as_loop:
676      CINV(10, 0)
677      CINV(10, 4)
678      CINV(10, 8)
679      CINV(10, 12)
680      CINV(10, 16)
681      CINV(10, 20)
682      CINV(10, 24)
683      CINV(10, 28)
684      CINV(10, 32)
685      CINV(10, 36)
686      CINV(10, 40)
687      CINV(10, 44)
688      CINV(10, 48)
689      CINV(10, 52)
690      CINV(10, 56)
691      CINV(10, 60)
692      CINV(10, 64)
693      CINV(10, 68)
694      CINV(10, 72)
695      CINV(10, 76)
696      CINV(10, 80)
697      CINV(10, 84)
698      CINV(10, 88)
699      CINV(10, 92)
700      CINV(10, 96)
701      CINV(10, 100)
702      CINV(10, 104)
703      CINV(10, 108)
704      CINV(10, 112)
705      CINV(10, 116)
706      CINV(10, 120)
707      CINV(10, 124)
708      addi a0, a0, 128
709      blt a0, a1, invalidate_bulk_32_as_loop
710      ret
711
712  /*-----*/
713
714  writeback_32_as:
715      fence
716  writeback_32_as_loop:
717      CWB(10, 0x000)
718      addi a0, a0, 4
719      blt a0, a1, writeback_32_as_loop
720      ret
721
722  /*-----*/
723
724  writeback_bulk_32_as:
725      fence
726  writeback_bulk_32_as_loop:
727      CWB(10, 0)
728      CWB(10, 4)
729      CWB(10, 8)
730      CWB(10, 12)
731      CWB(10, 16)
732      CWB(10, 20)
733      CWB(10, 24)
734      CWB(10, 28)
735      CWB(10, 32)
736      CWB(10, 36)
737      CWB(10, 40)
738      CWB(10, 44)
739      CWB(10, 48)
740      CWB(10, 52)
741      CWB(10, 56)
742      CWB(10, 60)
743      CWB(10, 64)
744      CWB(10, 68)
745      CWB(10, 72)
746      CWB(10, 76)
747      CWB(10, 80)
748      CWB(10, 84)
749      CWB(10, 88)
750      CWB(10, 92)
751      CWB(10, 96)
752      CWB(10, 100)
753      CWB(10, 104)
754      CWB(10, 108)
755      CWB(10, 112)
```

```
756     CWB(10, 116)
757     CWB(10, 120)
758     CWB(10, 124)
759     addi a0, a0, 128
760     blt a0, a1, writeback_bulk_32_as_loop
761     ret
762
763 /*-----*/
764
765 flush_32_as:
766     fence
767 flush_32_as_loop:
768     CFLUSH(10, 0x000)
769     addi a0, a0, 4
770     blt a0, a1, flush_32_as_loop
771     ret
772
773 /*-----*/
774
775 flush_bulk_32_as:
776     fence
777 flush_bulk_32_as_loop:
778     CFLUSH(10, 0)
779     CFLUSH(10, 4)
780     CFLUSH(10, 8)
781     CFLUSH(10, 12)
782     CFLUSH(10, 16)
783     CFLUSH(10, 20)
784     CFLUSH(10, 24)
785     CFLUSH(10, 28)
786     CFLUSH(10, 32)
787     CFLUSH(10, 36)
788     CFLUSH(10, 40)
789     CFLUSH(10, 44)
790     CFLUSH(10, 48)
791     CFLUSH(10, 52)
792     CFLUSH(10, 56)
793     CFLUSH(10, 60)
794     CFLUSH(10, 64)
795     CFLUSH(10, 68)
796     CFLUSH(10, 72)
797     CFLUSH(10, 76)
798     CFLUSH(10, 80)
799     CFLUSH(10, 84)
800     CFLUSH(10, 88)
801     CFLUSH(10, 92)
802     CFLUSH(10, 96)
803     CFLUSH(10, 100)
804     CFLUSH(10, 104)
805     CFLUSH(10, 108)
806     CFLUSH(10, 112)
807     CFLUSH(10, 116)
808     CFLUSH(10, 120)
809     CFLUSH(10, 124)
810     addi a0, a0, 128
811     blt a0, a1, flush_bulk_32_as_loop
812     ret
813
814 invalidate_64_as:
815     fence
816 invalidate_64_as_loop:
817     CINV(10, 0x000)
818     addi a0, a0, 8
819     blt a0, a1, invalidate_64_as_loop
820     ret
821
822 /*-----*/
823
824 invalidate_bulk_64_as:
825     fence
826 invalidate_bulk_64_as_loop:
827     CINV(10, 0)
828     CINV(10, 8)
829     CINV(10, 16)
830     CINV(10, 24)
831     CINV(10, 32)
832     CINV(10, 40)
833     CINV(10, 48)
834     CINV(10, 56)
835     CINV(10, 64)
836     CINV(10, 72)
837     CINV(10, 80)
838     CINV(10, 88)
839     CINV(10, 96)
840     CINV(10, 104)
841     CINV(10, 112)
842     CINV(10, 120)
```

```

843     CINV(10, 128)
844     CINV(10, 136)
845     CINV(10, 144)
846     CINV(10, 152)
847     CINV(10, 160)
848     CINV(10, 168)
849     CINV(10, 176)
850     CINV(10, 184)
851     CINV(10, 192)
852     CINV(10, 200)
853     CINV(10, 208)
854     CINV(10, 216)
855     CINV(10, 224)
856     CINV(10, 232)
857     CINV(10, 240)
858     CINV(10, 248)
859     addi a0, a0, 256
860     blt a0, a1, invalidate_bulk_64_as_loop
861     ret
862
863 /*-----*/
864
865 writeback_64_as:
866     fence
867 writeback_64_as_loop:
868     CWB(10, 0x000)
869     addi a0, a0, 8
870     blt a0, a1, writeback_64_as_loop
871     ret
872
873 /*-----*/
874
875 writeback_bulk_64_as:
876     fence
877 writeback_bulk_64_as_loop:
878     CWB(10, 0)
879     CWB(10, 8)
880     CWB(10, 16)
881     CWB(10, 24)
882     CWB(10, 32)
883     CWB(10, 40)
884     CWB(10, 48)
885     CWB(10, 56)
886     CWB(10, 64)
887     CWB(10, 72)
888     CWB(10, 80)
889     CWB(10, 88)
890     CWB(10, 96)
891     CWB(10, 104)
892     CWB(10, 112)
893     CWB(10, 120)
894     CWB(10, 128)
895     CWB(10, 136)
896     CWB(10, 144)
897     CWB(10, 152)
898     CWB(10, 160)
899     CWB(10, 168)
900     CWB(10, 176)
901     CWB(10, 184)
902     CWB(10, 192)
903     CWB(10, 200)
904     CWB(10, 208)
905     CWB(10, 216)
906     CWB(10, 224)
907     CWB(10, 232)
908     CWB(10, 240)
909     CWB(10, 248)
910     addi a0, a0, 256
911     blt a0, a1, writeback_bulk_64_as_loop
912     ret
913
914 /*-----*/
915
916 flush_64_as:
917     fence
918 flush_64_as_loop:
919     CFLUSH(10, 0x000)
920     addi a0, a0, 8
921     blt a0, a1, flush_64_as_loop
922     ret
923
924 /*-----*/
925
926 flush_bulk_64_as:
927     fence
928 flush_bulk_64_as_loop:
929     CFLUSH(10, 0)

```



```
930     CFLUSH(10, 8)
931     CFLUSH(10, 16)
932     CFLUSH(10, 24)
933     CFLUSH(10, 32)
934     CFLUSH(10, 40)
935     CFLUSH(10, 48)
936     CFLUSH(10, 56)
937     CFLUSH(10, 64)
938     CFLUSH(10, 72)
939     CFLUSH(10, 80)
940     CFLUSH(10, 88)
941     CFLUSH(10, 96)
942     CFLUSH(10, 104)
943     CFLUSH(10, 112)
944     CFLUSH(10, 120)
945     CFLUSH(10, 128)
946     CFLUSH(10, 136)
947     CFLUSH(10, 144)
948     CFLUSH(10, 152)
949     CFLUSH(10, 160)
950     CFLUSH(10, 168)
951     CFLUSH(10, 176)
952     CFLUSH(10, 184)
953     CFLUSH(10, 192)
954     CFLUSH(10, 200)
955     CFLUSH(10, 208)
956     CFLUSH(10, 216)
957     CFLUSH(10, 224)
958     CFLUSH(10, 232)
959     CFLUSH(10, 240)
960     CFLUSH(10, 248)
961     addi a0, a0, 256
962     blt a0, a1, flush_bulk_64_as_loop
963     ret
964
965 invalidate_128_as:
966     fence
967 invalidate_128_as_loop:
968     CINV(10, 0x000)
969     addi a0, a0, 16
970     blt a0, a1, invalidate_128_as_loop
971     ret
972
973 /*-----*/
974
975 invalidate_bulk_128_as:
976     fence
977 invalidate_bulk_128_as_loop:
978     CINV(10, 0)
979     CINV(10, 16)
980     CINV(10, 32)
981     CINV(10, 48)
982     CINV(10, 64)
983     CINV(10, 80)
984     CINV(10, 96)
985     CINV(10, 112)
986     CINV(10, 128)
987     CINV(10, 144)
988     CINV(10, 160)
989     CINV(10, 176)
990     CINV(10, 192)
991     CINV(10, 208)
992     CINV(10, 224)
993     CINV(10, 240)
994     CINV(10, 256)
995     CINV(10, 272)
996     CINV(10, 288)
997     CINV(10, 304)
998     CINV(10, 320)
999     CINV(10, 336)
1000     CINV(10, 352)
1001     CINV(10, 368)
1002     CINV(10, 384)
1003     CINV(10, 400)
1004     CINV(10, 416)
1005     CINV(10, 432)
1006     CINV(10, 448)
1007     CINV(10, 464)
1008     CINV(10, 480)
1009     CINV(10, 496)
1010     addi a0, a0, 512
1011     blt a0, a1, invalidate_bulk_128_as_loop
1012     ret
1013
1014 /*-----*/
1015
1016 writeback_128_as:
```

```
1017     fence
1018 writeback_128_as_loop:
1019     CWB(10, 0x000)
1020     addi a0, a0, 16
1021     blt a0, a1, writeback_128_as_loop
1022     ret
1023
1024 /*-----*/
1025
1026 writeback_bulk_128_as:
1027     fence
1028 writeback_bulk_128_as_loop:
1029     CWB(10, 0)
1030     CWB(10, 16)
1031     CWB(10, 32)
1032     CWB(10, 48)
1033     CWB(10, 64)
1034     CWB(10, 80)
1035     CWB(10, 96)
1036     CWB(10, 112)
1037     CWB(10, 128)
1038     CWB(10, 144)
1039     CWB(10, 160)
1040     CWB(10, 176)
1041     CWB(10, 192)
1042     CWB(10, 208)
1043     CWB(10, 224)
1044     CWB(10, 240)
1045     CWB(10, 256)
1046     CWB(10, 272)
1047     CWB(10, 288)
1048     CWB(10, 304)
1049     CWB(10, 320)
1050     CWB(10, 336)
1051     CWB(10, 352)
1052     CWB(10, 368)
1053     CWB(10, 384)
1054     CWB(10, 400)
1055     CWB(10, 416)
1056     CWB(10, 432)
1057     CWB(10, 448)
1058     CWB(10, 464)
1059     CWB(10, 480)
1060     CWB(10, 496)
1061     addi a0, a0, 512
1062     blt a0, a1, writeback_bulk_128_as_loop
1063     ret
1064
1065 /*-----*/
1066
1067 flush_128_as:
1068     fence
1069 flush_128_as_loop:
1070     CFLUSH(10, 0x000)
1071     addi a0, a0, 16
1072     blt a0, a1, flush_128_as_loop
1073     ret
1074
1075 /*-----*/
1076
1077 flush_bulk_128_as:
1078     fence
1079 flush_bulk_128_as_loop:
1080     CFLUSH(10, 0)
1081     CFLUSH(10, 16)
1082     CFLUSH(10, 32)
1083     CFLUSH(10, 48)
1084     CFLUSH(10, 64)
1085     CFLUSH(10, 80)
1086     CFLUSH(10, 96)
1087     CFLUSH(10, 112)
1088     CFLUSH(10, 128)
1089     CFLUSH(10, 144)
1090     CFLUSH(10, 160)
1091     CFLUSH(10, 176)
1092     CFLUSH(10, 192)
1093     CFLUSH(10, 208)
1094     CFLUSH(10, 224)
1095     CFLUSH(10, 240)
1096     CFLUSH(10, 256)
1097     CFLUSH(10, 272)
1098     CFLUSH(10, 288)
1099     CFLUSH(10, 304)
1100     CFLUSH(10, 320)
1101     CFLUSH(10, 336)
1102     CFLUSH(10, 352)
1103     CFLUSH(10, 368)
```

```
1104     CFLUSH(10, 384)
1105     CFLUSH(10, 400)
1106     CFLUSH(10, 416)
1107     CFLUSH(10, 432)
1108     CFLUSH(10, 448)
1109     CFLUSH(10, 464)
1110     CFLUSH(10, 480)
1111     CFLUSH(10, 496)
1112     addi a0, a0, 512
1113     blt  a0, a1, flush_bulk_128_as_loop
1114     ret
1115
1116 invalidate_256_as:
1117     fence
1118 invalidate_256_as_loop:
1119     CINV(10, 0x000)
1120     addi a0, a0, 32
1121     blt  a0, a1, invalidate_256_as_loop
1122     ret
1123
1124 /*-----*/
1125
1126 invalidate_bulk_256_as:
1127     fence
1128 invalidate_bulk_256_as_loop:
1129     CINV(10, 0)
1130     CINV(10, 32)
1131     CINV(10, 64)
1132     CINV(10, 96)
1133     CINV(10, 128)
1134     CINV(10, 160)
1135     CINV(10, 192)
1136     CINV(10, 224)
1137     CINV(10, 256)
1138     CINV(10, 288)
1139     CINV(10, 320)
1140     CINV(10, 352)
1141     CINV(10, 384)
1142     CINV(10, 416)
1143     CINV(10, 448)
1144     CINV(10, 480)
1145     CINV(10, 512)
1146     CINV(10, 544)
1147     CINV(10, 576)
1148     CINV(10, 608)
1149     CINV(10, 640)
1150     CINV(10, 672)
1151     CINV(10, 704)
1152     CINV(10, 736)
1153     CINV(10, 768)
1154     CINV(10, 800)
1155     CINV(10, 832)
1156     CINV(10, 864)
1157     CINV(10, 896)
1158     CINV(10, 928)
1159     CINV(10, 960)
1160     CINV(10, 992)
1161     addi a0, a0, 1024
1162     blt  a0, a1, invalidate_bulk_256_as_loop
1163     ret
1164
1165 /*-----*/
1166
1167 writeback_256_as:
1168     fence
1169 writeback_256_as_loop:
1170     CWB(10, 0x000)
1171     addi a0, a0, 32
1172     blt  a0, a1, writeback_256_as_loop
1173     ret
1174
1175 /*-----*/
1176
1177 writeback_bulk_256_as:
1178     fence
1179 writeback_bulk_256_as_loop:
1180     CWB(10, 0)
1181     CWB(10, 32)
1182     CWB(10, 64)
1183     CWB(10, 96)
1184     CWB(10, 128)
1185     CWB(10, 160)
1186     CWB(10, 192)
1187     CWB(10, 224)
1188     CWB(10, 256)
1189     CWB(10, 288)
1190     CWB(10, 320)
```

```
1191     CWB(10, 352)
1192     CWB(10, 384)
1193     CWB(10, 416)
1194     CWB(10, 448)
1195     CWB(10, 480)
1196     CWB(10, 512)
1197     CWB(10, 544)
1198     CWB(10, 576)
1199     CWB(10, 608)
1200     CWB(10, 640)
1201     CWB(10, 672)
1202     CWB(10, 704)
1203     CWB(10, 736)
1204     CWB(10, 768)
1205     CWB(10, 800)
1206     CWB(10, 832)
1207     CWB(10, 864)
1208     CWB(10, 896)
1209     CWB(10, 928)
1210     CWB(10, 960)
1211     CWB(10, 992)
1212     addi a0, a0, 1024
1213     blt a0, a1, writeback_bulk_256_as_loop
1214     ret
1215
1216 /*-----*/
1217
1218 flush_256_as:
1219     fence
1220 flush_256_as_loop:
1221     CFLUSH(10, 0x000)
1222     addi a0, a0, 32
1223     blt a0, a1, flush_256_as_loop
1224     ret
1225
1226 /*-----*/
1227
1228 flush_bulk_256_as:
1229     fence
1230 flush_bulk_256_as_loop:
1231     CFLUSH(10, 0)
1232     CFLUSH(10, 32)
1233     CFLUSH(10, 64)
1234     CFLUSH(10, 96)
1235     CFLUSH(10, 128)
1236     CFLUSH(10, 160)
1237     CFLUSH(10, 192)
1238     CFLUSH(10, 224)
1239     CFLUSH(10, 256)
1240     CFLUSH(10, 288)
1241     CFLUSH(10, 320)
1242     CFLUSH(10, 352)
1243     CFLUSH(10, 384)
1244     CFLUSH(10, 416)
1245     CFLUSH(10, 448)
1246     CFLUSH(10, 480)
1247     CFLUSH(10, 512)
1248     CFLUSH(10, 544)
1249     CFLUSH(10, 576)
1250     CFLUSH(10, 608)
1251     CFLUSH(10, 640)
1252     CFLUSH(10, 672)
1253     CFLUSH(10, 704)
1254     CFLUSH(10, 736)
1255     CFLUSH(10, 768)
1256     CFLUSH(10, 800)
1257     CFLUSH(10, 832)
1258     CFLUSH(10, 864)
1259     CFLUSH(10, 896)
1260     CFLUSH(10, 928)
1261     CFLUSH(10, 960)
1262     CFLUSH(10, 992)
1263     addi a0, a0, 1024
1264     blt a0, a1, flush_bulk_256_as_loop
1265     ret
1266
1267 invalidate_512_as:
1268     fence
1269 invalidate_512_as_loop:
1270     CINV(10, 0x000)
1271     addi a0, a0, 64
1272     blt a0, a1, invalidate_512_as_loop
1273     ret
1274
1275 /*-----*/
1276
1277 invalidate_bulk_512_as:
```

```
1278     fence
1279     li t0, 2048
1280     invalidate_bulk_512_as_loop:
1281         CINV(10, 0)
1282         CINV(10, 64)
1283         CINV(10, 128)
1284         CINV(10, 192)
1285         CINV(10, 256)
1286         CINV(10, 320)
1287         CINV(10, 384)
1288         CINV(10, 448)
1289         CINV(10, 512)
1290         CINV(10, 576)
1291         CINV(10, 640)
1292         CINV(10, 704)
1293         CINV(10, 768)
1294         CINV(10, 832)
1295         CINV(10, 896)
1296         CINV(10, 960)
1297         CINV(10, 1024)
1298         CINV(10, 1088)
1299         CINV(10, 1152)
1300         CINV(10, 1216)
1301         CINV(10, 1280)
1302         CINV(10, 1344)
1303         CINV(10, 1408)
1304         CINV(10, 1472)
1305         CINV(10, 1536)
1306         CINV(10, 1600)
1307         CINV(10, 1664)
1308         CINV(10, 1728)
1309         CINV(10, 1792)
1310         CINV(10, 1856)
1311         CINV(10, 1920)
1312         CINV(10, 1984)
1313         add a0, a0, t0
1314         blt a0, a1, invalidate_bulk_512_as_loop
1315         ret
1316
1317     /*-----*/
1318
1319     writeback_512_as:
1320         fence
1321     writeback_512_as_loop:
1322         CWB(10, 0x000)
1323         addi a0, a0, 64
1324         blt a0, a1, writeback_512_as_loop
1325         ret
1326
1327     /*-----*/
1328
1329     writeback_bulk_512_as:
1330         fence
1331     li t0, 2048
1332     writeback_bulk_512_as_loop:
1333         CWB(10, 0)
1334         CWB(10, 64)
1335         CWB(10, 128)
1336         CWB(10, 192)
1337         CWB(10, 256)
1338         CWB(10, 320)
1339         CWB(10, 384)
1340         CWB(10, 448)
1341         CWB(10, 512)
1342         CWB(10, 576)
1343         CWB(10, 640)
1344         CWB(10, 704)
1345         CWB(10, 768)
1346         CWB(10, 832)
1347         CWB(10, 896)
1348         CWB(10, 960)
1349         CWB(10, 1024)
1350         CWB(10, 1088)
1351         CWB(10, 1152)
1352         CWB(10, 1216)
1353         CWB(10, 1280)
1354         CWB(10, 1344)
1355         CWB(10, 1408)
1356         CWB(10, 1472)
1357         CWB(10, 1536)
1358         CWB(10, 1600)
1359         CWB(10, 1664)
1360         CWB(10, 1728)
1361         CWB(10, 1792)
1362         CWB(10, 1856)
1363         CWB(10, 1920)
1364         CWB(10, 1984)
```

```
1365     add a0, a0, t0
1366     blt  a0, a1, writeback_bulk_512_as_loop
1367     ret
1368
1369 /*-----*/
1370
1371 flush_512_as:
1372     fence
1373 flush_512_as_loop:
1374     CFLUSH(10, 0x000)
1375     addi a0, a0, 64
1376     blt  a0, a1, flush_512_as_loop
1377     ret
1378
1379 /*-----*/
1380
1381 flush_bulk_512_as:
1382     fence
1383     li t0, 2048
1384 flush_bulk_512_as_loop:
1385     CFLUSH(10, 0)
1386     CFLUSH(10, 64)
1387     CFLUSH(10, 128)
1388     CFLUSH(10, 192)
1389     CFLUSH(10, 256)
1390     CFLUSH(10, 320)
1391     CFLUSH(10, 384)
1392     CFLUSH(10, 448)
1393     CFLUSH(10, 512)
1394     CFLUSH(10, 576)
1395     CFLUSH(10, 640)
1396     CFLUSH(10, 704)
1397     CFLUSH(10, 768)
1398     CFLUSH(10, 832)
1399     CFLUSH(10, 896)
1400     CFLUSH(10, 960)
1401     CFLUSH(10, 1024)
1402     CFLUSH(10, 1088)
1403     CFLUSH(10, 1152)
1404     CFLUSH(10, 1216)
1405     CFLUSH(10, 1280)
1406     CFLUSH(10, 1344)
1407     CFLUSH(10, 1408)
1408     CFLUSH(10, 1472)
1409     CFLUSH(10, 1536)
1410     CFLUSH(10, 1600)
1411     CFLUSH(10, 1664)
1412     CFLUSH(10, 1728)
1413     CFLUSH(10, 1792)
1414     CFLUSH(10, 1856)
1415     CFLUSH(10, 1920)
1416     CFLUSH(10, 1984)
1417     add a0, a0, t0
1418     blt  a0, a1, flush_bulk_512_as_loop
1419     ret
1420
1421 invalidate_1024_as:
1422     fence
1423 invalidate_1024_as_loop:
1424     CINV(10, 0x000)
1425     addi a0, a0, 128
1426     blt  a0, a1, invalidate_1024_as_loop
1427     ret
1428
1429 /*-----*/
1430
1431 invalidate_bulk_1024_as:
1432     fence
1433     li t0, 2048
1434 invalidate_bulk_1024_as_loop:
1435     CINV(10, 0)
1436     CINV(10, 128)
1437     CINV(10, 256)
1438     CINV(10, 384)
1439     CINV(10, 512)
1440     CINV(10, 640)
1441     CINV(10, 768)
1442     CINV(10, 896)
1443     CINV(10, 1024)
1444     CINV(10, 1152)
1445     CINV(10, 1280)
1446     CINV(10, 1408)
1447     CINV(10, 1536)
1448     CINV(10, 1664)
1449     CINV(10, 1792)
1450     CINV(10, 1920)
1451     add a0, a0, t0
```

```
1452     CINV(10, 0)
1453     CINV(10, 128)
1454     CINV(10, 256)
1455     CINV(10, 384)
1456     CINV(10, 512)
1457     CINV(10, 640)
1458     CINV(10, 768)
1459     CINV(10, 896)
1460     CINV(10, 1024)
1461     CINV(10, 1152)
1462     CINV(10, 1280)
1463     CINV(10, 1408)
1464     CINV(10, 1536)
1465     CINV(10, 1664)
1466     CINV(10, 1792)
1467     CINV(10, 1920)
1468     add a0, a0, t0
1469     blt a0, a1, invalidate_bulk_1024_as_loop
1470     ret
1471
1472     /*-----*/
1473
1474     writeback_1024_as:
1475         fence
1476     writeback_1024_as_loop:
1477         CWB(10, 0x000)
1478         addi a0, a0, 128
1479         blt a0, a1, writeback_1024_as_loop
1480         ret
1481
1482     /*-----*/
1483
1484     writeback_bulk_1024_as:
1485         fence
1486         li t0, 2048
1487     writeback_bulk_1024_as_loop:
1488         CWB(10, 0)
1489         CWB(10, 128)
1490         CWB(10, 256)
1491         CWB(10, 384)
1492         CWB(10, 512)
1493         CWB(10, 640)
1494         CWB(10, 768)
1495         CWB(10, 896)
1496         CWB(10, 1024)
1497         CWB(10, 1152)
1498         CWB(10, 1280)
1499         CWB(10, 1408)
1500         CWB(10, 1536)
1501         CWB(10, 1664)
1502         CWB(10, 1792)
1503         CWB(10, 1920)
1504         add a0, a0, t0
1505         CWB(10, 0)
1506         CWB(10, 128)
1507         CWB(10, 256)
1508         CWB(10, 384)
1509         CWB(10, 512)
1510         CWB(10, 640)
1511         CWB(10, 768)
1512         CWB(10, 896)
1513         CWB(10, 1024)
1514         CWB(10, 1152)
1515         CWB(10, 1280)
1516         CWB(10, 1408)
1517         CWB(10, 1536)
1518         CWB(10, 1664)
1519         CWB(10, 1792)
1520         CWB(10, 1920)
1521         add a0, a0, t0
1522         blt a0, a1, writeback_bulk_1024_as_loop
1523         ret
1524
1525     /*-----*/
1526
1527     flush_1024_as:
1528         fence
1529     flush_1024_as_loop:
1530         CFLUSH(10, 0x000)
1531         addi a0, a0, 128
1532         blt a0, a1, flush_1024_as_loop
1533         ret
1534
1535     /*-----*/
1536
1537     flush_bulk_1024_as:
1538         fence
```

```
1539 li t0, 2048
1540 flush_bulk_1024_as_loop:
1541     CFLUSH(10, 0)
1542     CFLUSH(10, 128)
1543     CFLUSH(10, 256)
1544     CFLUSH(10, 384)
1545     CFLUSH(10, 512)
1546     CFLUSH(10, 640)
1547     CFLUSH(10, 768)
1548     CFLUSH(10, 896)
1549     CFLUSH(10, 1024)
1550     CFLUSH(10, 1152)
1551     CFLUSH(10, 1280)
1552     CFLUSH(10, 1408)
1553     CFLUSH(10, 1536)
1554     CFLUSH(10, 1664)
1555     CFLUSH(10, 1792)
1556     CFLUSH(10, 1920)
1557     add a0, a0, t0
1558     CFLUSH(10, 0)
1559     CFLUSH(10, 128)
1560     CFLUSH(10, 256)
1561     CFLUSH(10, 384)
1562     CFLUSH(10, 512)
1563     CFLUSH(10, 640)
1564     CFLUSH(10, 768)
1565     CFLUSH(10, 896)
1566     CFLUSH(10, 1024)
1567     CFLUSH(10, 1152)
1568     CFLUSH(10, 1280)
1569     CFLUSH(10, 1408)
1570     CFLUSH(10, 1536)
1571     CFLUSH(10, 1664)
1572     CFLUSH(10, 1792)
1573     CFLUSH(10, 1920)
1574     add a0, a0, t0
1575     blt a0, a1, flush_bulk_1024_as_loop
1576     ret
1577
1578 invalidate_2048_as:
1579     fence
1580 invalidate_2048_as_loop:
1581     CINV(10, 0x000)
1582     addi a0, a0, 256
1583     blt a0, a1, invalidate_2048_as_loop
1584     ret
1585
1586 /*-----*/
1587
1588 invalidate_bulk_2048_as:
1589     fence
1590 li t0, 2048
1591 invalidate_bulk_2048_as_loop:
1592     CINV(10, 0)
1593     CINV(10, 256)
1594     CINV(10, 512)
1595     CINV(10, 768)
1596     CINV(10, 1024)
1597     CINV(10, 1280)
1598     CINV(10, 1536)
1599     CINV(10, 1792)
1600     add a0, a0, t0
1601     CINV(10, 0)
1602     CINV(10, 256)
1603     CINV(10, 512)
1604     CINV(10, 768)
1605     CINV(10, 1024)
1606     CINV(10, 1280)
1607     CINV(10, 1536)
1608     CINV(10, 1792)
1609     add a0, a0, t0
1610     CINV(10, 0)
1611     CINV(10, 256)
1612     CINV(10, 512)
1613     CINV(10, 768)
1614     CINV(10, 1024)
1615     CINV(10, 1280)
1616     CINV(10, 1536)
1617     CINV(10, 1792)
1618     add a0, a0, t0
1619     CINV(10, 0)
1620     CINV(10, 256)
1621     CINV(10, 512)
1622     CINV(10, 768)
1623     CINV(10, 1024)
1624     CINV(10, 1280)
1625     CINV(10, 1536)
```



```
1626     CINV(10, 1792)
1627     add a0, a0, t0
1628     blt a0, a1, invalidate_bulk_2048_as_loop
1629     ret
1630
1631 writeback_2048_as:
1632     fence
1633 writeback_2048_as_loop:
1634     CWB(10, 0x000)
1635     addi a0, a0, 256
1636     blt a0, a1, writeback_2048_as_loop
1637     ret
1638
1639 /*-----*/
1640
1641 writeback_bulk_2048_as:
1642     fence
1643     li t0, 2048
1644 writeback_bulk_2048_as_loop:
1645     CWB(10, 0)
1646     CWB(10, 256)
1647     CWB(10, 512)
1648     CWB(10, 768)
1649     CWB(10, 1024)
1650     CWB(10, 1280)
1651     CWB(10, 1536)
1652     CWB(10, 1792)
1653     add a0, a0, t0
1654     CWB(10, 0)
1655     CWB(10, 256)
1656     CWB(10, 512)
1657     CWB(10, 768)
1658     CWB(10, 1024)
1659     CWB(10, 1280)
1660     CWB(10, 1536)
1661     CWB(10, 1792)
1662     add a0, a0, t0
1663     CWB(10, 0)
1664     CWB(10, 256)
1665     CWB(10, 512)
1666     CWB(10, 768)
1667     CWB(10, 1024)
1668     CWB(10, 1280)
1669     CWB(10, 1536)
1670     CWB(10, 1792)
1671     add a0, a0, t0
1672     CWB(10, 0)
1673     CWB(10, 256)
1674     CWB(10, 512)
1675     CWB(10, 768)
1676     CWB(10, 1024)
1677     CWB(10, 1280)
1678     CWB(10, 1536)
1679     CWB(10, 1792)
1680     add a0, a0, t0
1681     blt a0, a1, writeback_bulk_2048_as_loop
1682     ret
1683
1684 flush_2048_as:
1685     fence
1686 flush_2048_as_loop:
1687     CFLUSH(10, 0x000)
1688     addi a0, a0, 256
1689     blt a0, a1, flush_2048_as_loop
1690     ret
1691
1692 /*-----*/
1693
1694 flush_bulk_2048_as:
1695     fence
1696     li t0, 2048
1697 flush_bulk_2048_as_loop:
1698     CFLUSH(10, 0)
1699     CFLUSH(10, 256)
1700     CFLUSH(10, 512)
1701     CFLUSH(10, 768)
1702     CFLUSH(10, 1024)
1703     CFLUSH(10, 1280)
1704     CFLUSH(10, 1536)
1705     CFLUSH(10, 1792)
1706     add a0, a0, t0
1707     CFLUSH(10, 0)
1708     CFLUSH(10, 256)
1709     CFLUSH(10, 512)
1710     CFLUSH(10, 768)
1711     CFLUSH(10, 1024)
1712     CFLUSH(10, 1280)
```

```

1713     CFLUSH(10, 1536)
1714     CFLUSH(10, 1792)
1715     add a0, a0, t0
1716     CFLUSH(10, 0)
1717     CFLUSH(10, 256)
1718     CFLUSH(10, 512)
1719     CFLUSH(10, 768)
1720     CFLUSH(10, 1024)
1721     CFLUSH(10, 1280)
1722     CFLUSH(10, 1536)
1723     CFLUSH(10, 1792)
1724     add a0, a0, t0
1725     CFLUSH(10, 0)
1726     CFLUSH(10, 256)
1727     CFLUSH(10, 512)
1728     CFLUSH(10, 768)
1729     CFLUSH(10, 1024)
1730     CFLUSH(10, 1280)
1731     CFLUSH(10, 1536)
1732     CFLUSH(10, 1792)
1733     add a0, a0, t0
1734     blt a0, a1, flush_bulk_2048_as_loop
1735     ret
1736
1737 #endif /* !DOXYGEN */
1738
1739 /*EOF*****

```

8.17 pn_cache/pn_cache_RV32I_buildscript.py File Reference

Builds the architecture dependent assembly files of the [Cache Module](#) since they are performance critical, very dependent on cache line size, and should not take up to much space.

8.17.1 Detailed Description

Builds the architecture dependent assembly files of the [Cache Module](#) since they are performance critical, very dependent on cache line size, and should not take up to much space.

This means we can neither make the cache line size a variable, since that would mean a performance trade off, nor can we put an implementation for every possible cache line size, since that would take a giant amount of space in the binary. This Python Script seems the perfect solution for that, except for the fact that it reduces binary compatibility of the libparanut to ParaNuts with different cache line sizes. Having to re-compile the libparanut is, of of course, a minor inconvenience, but the only other solution, which would be code that modifies itself on startup, is extremely complex and could be a possible subject for another Bachelor Thesis.

For all those who hate having to recompile stuff, it is also possible to start this script with the option "auto", which generates a file that does contain all of the cache line size functions from 32 bit to 2048 in powers of two. Since every other linesize is deemed to be unrealistic, this version has the most binary compatibility ... but also the biggest code size of them all. Make of that what you want.

Also check documentation of [PN_CACHE_LINESIZE](#). You can take a look at an example ([pn_cache_RV32I_auto.S](#)) of the files built with this script.

Also, if you want to know how to introduce a new cache line size into the libparanut, take a look at [PN_ERR_CACHE_LINESIZE](#).

```

1 # Copyright 2019-2020 Anna Pfuetzner (<annakerstin.pfuetzner@gmail.com>)
2 # Redistribution and use in source and binary forms, with or without
3 # modification, are permitted provided that the following conditions are met:
4 #
5 # 1. Redistributions of source code must retain the above copyright notice,
6 # this list of conditions and the following disclaimer.
7 #
8 # 2. Redistributions in binary form must reproduce the above copyright notice,
9 # this list of conditions and the following disclaimer in the documentation

```

```

10 # and/or other materials provided with the distribution.
11 #
12 # THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
13 # AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
14 # IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
15 # ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
16 # LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
17 # CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
18 # SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
19 # INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
20 # CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
21 # ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
22 # POSSIBILITY OF SUCH DAMAGE.
23
24
25
26
61
62 #Imports#####
63
64 # enable command line argument parsing
65 import sys
66
67 # enable ceil function
68 import math
69
70 #Global Variables#####
71
72 cache_line_size_list = "32", "64", "128", "256", "512", "1024", "2048"
73 function_list        = "invalidate", "writeback", "flush"
74
75 #Subfunctions#####
76
77 def create_warning(filep):
78
79     filep.write(
80         "/* \n"
81         " * I was automatically generated by pn_cache_RV32I_buildscript.py,\n"
82         " * invoked by libparanut Makefile -> No edits!\n"
83         "*/\n"
84         "\n"
85     )
86
87
88
89 def create_doxy(filep, specific_size):
90
91     filep.write(
92         "/* @file */\n"
93         "\n"
94         "/*\n"
95         " * \\internal\n"
96         " * \\file\n"
97         " * \\brief      Contains RV32I assembly implementations of assembly\n"
98         )
99
100     if specific_size:
101
102         filep.write(
103             " *      functions called in \\ref pn_cache.c for cache line\n" +
104             " *      size " + sys.argv[1] + ".\n"
105             " *\n"
106         )
107
108     else:
109
110         filep.write(
111             " *      functions called in \\ref pn_cache.c for all cache \n" +
112             " *      line sizes.\n"
113             " *\n"
114         )
115
116     filep.write(
117         " * This file (for this specific architecture, others may do it\n" +
118         " * differently) was created by \\ref pn_cache_RV32I_buildscript.py\n" +
119         " * while compiling (automatically called by \\ref Makefile), as\n" +
120         " * are the other files with the pattern pn_cache_RV32I_*.S for name.\n" +
121         " * The reasons for this are described in the \\ref\n" +
122         " * pn_cache_RV32I_buildscript.py documentation itself.\n"
123         " *\n"
124     )
125
126     if specific_size:
127
128         filep.write(
129             " * The code shown below was created with a cache line size of "
130             sys.argv[1] + " Bit.\n"

```

```

131     " *\n"
132 )
133
134 else:
135
136     filep.write(
137         " * The code shown below was created for all cache line sizes.\n"      +
138         " *\n"
139     )
140
141     filep.write(
142         " * So far, the following cache line sizes are available (in bits):\n"  +
143         " * " + " ".join(cache_line_size_list) + "\n"                        +
144         " *\n"                                                                +
145         " * Contains implementations of:\n"                                    +
146         " *     - \\ref enable_cache_as()\n"                                  +
147         " *     - \\ref disable_cache_as()\n"                                  +
148         " *     - \\ref cache_banks_as()\n"                                    +
149         " *     - \\ref cache_sets_as()\n"                                    +
150         " *     - \\ref mem_size_as()\n"                                       +
151     )
152
153     if specific_size:
154
155         filep.write(
156             " *     - \\ref invalidate_" + sys.argv[1] + "_as()\n"              +
157             " *     - \\ref invalidate_bulk_" + sys.argv[1] + "_as()\n"         +
158             " *     - \\ref writeback_" + sys.argv[1] + "_as()\n"              +
159             " *     - \\ref writeback_bulk_" + sys.argv[1] + "_as()\n"         +
160             " *     - \\ref flush_" + sys.argv[1] + "_as()\n"                 +
161             " *     - \\ref flush_bulk_" + sys.argv[1] + "_as()\n"             +
162             " *\n"
163         )
164
165     else:
166
167         for cache_line_size in cache_line_size_list:
168             filep.write(
169                 " *     - \\ref invalidate_" + cache_line_size + "_as()\n"      +
170                 " *     - \\ref invalidate_bulk_" + cache_line_size + "_as()\n" +
171                 " *     - \\ref writeback_" + cache_line_size + "_as()\n"      +
172                 " *     - \\ref writeback_bulk_" + cache_line_size + "_as()\n" +
173                 " *     - \\ref flush_" + cache_line_size + "_as()\n"          +
174                 " *     - \\ref flush_bulk_" + cache_line_size + "_as()\n"    +
175             )
176
177             filep.write(" *\n")
178
179     filep.write(
180         " * \\includelinenos pn_cache/pn_cache_RV32I_" + sys.argv[1] + ".S\n"  +
181         " */\n\n"
182     )
183
184     filep.write(
185         " */\n"                                                                +
186         " * Put in here so Doxygen will know that it is implemented in this\n"  +
187         " * file. Sadly, Doxygen has no built in assembly interpreter, so we\n"  +
188         " * are stuck with this.\n"                                            +
189         " */\n"                                                                +
190         "\n"                                                                +
191         "#ifdef DOXYGEN\n"                                                  +
192         "\n"                                                                +
193         " /**\n"                                                            +
194         "  * \\addtogroup as\n"                                                +
195         "  * @{\n"                                                            +
196         "  */\n"                                                            +
197         "\n"                                                                +
198         " /**\n"                                                            +
199         "  * @{\n"                                                            +
200         "  */\n"                                                            +
201         "\n"                                                                +
202     )
203
204     filep.write(
205         " /**\n"                                                            +
206         "  * \\internal\n"                                                    +
207         "  * \\fn      void enable_cache_as(void)\n"                          +
208         "  * \\brief    Enables cache.\n"                                        +
209         "  */\n"                                                            +
210         "  void enable_cache_as(void) {}\n"                                  +
211         "\n"                                                                +
212         " /**\n"                                                            +
213         "  * \\internal\n"                                                    +
214         "  * \\fn      void disable_cache_as(void)\n"                          +
215         "  * \\brief    Disables cache.\n"                                      +
216         "  */\n"                                                            +
217         "  void disable_cache_as(void) {}\n"                                  +

```

```

218     "\n"
219     " /**\n"
220     "  * \\internal\n"
221     "  * \\fn      unsigned int cache_banks_as(void)\n"
222     "  * \\brief    Returns number of cache banks.\n"
223     "  *\n"
224     "  * \\return   Number of cache banks.\n"
225     "  */\n"
226     " unsigned int cache_banks_as(void) {}\n"
227     "\n"
228     " /**\n"
229     "  * \\internal\n"
230     "  * \\fn      unsigned int cache_sets_as(void)\n"
231     "  * \\brief    Returns number of cache sets.\n"
232     "  *\n"
233     "  * \\return   Number of cache sets.\n"
234     "  */\n"
235     " unsigned int cache_sets_as(void) {}\n"
236     "\n"
237     " /**\n"
238     "  * \\internal\n"
239     "  * \\fn      unsigned int mem_size_as(void)\n"
240     "  * \\brief    Reads memory size.\n"
241     "  *\n"
242     "  * \\return   Content of pnmemsize register.\n"
243     "  */\n"
244     " unsigned int mem_size_as(void) {}\n"
245     "\n"
246     )
247
248     if specific_size:
249         for function in function_list:
250
251             action = ""
252             if function == "invalidate":
253                 action = "Invalidates "
254             if function == "writeback":
255                 action = "Writes back "
256             if function == "flush":
257                 action = "Flushes "
258
259             filep.write(
260                 " /**\n"
261                 "  * \\internal\n"
262                 "  * \\fn      void " + function + "_" + sys.argv[1]
263                 "             _as(unsigned int addr, unsigned int endaddr)\n"
264                 "  * \\brief    " + action + "single cache lines until end address\n"
265                 "  *           is reached.\n"
266                 "  *\n"
267                 "  * Will at least " + function + " one line.\n"
268                 "  */\n"
269                 " void " + function + "_" + sys.argv[1]
270                 "             _as(unsigned int addr, unsigned int endaddr) {}\n"
271                 "\n"
272                 " /**\n"
273                 "  * \\internal\n"
274                 "  * \\fn      void " + function + "_bulk_" + sys.argv[1]
275                 "             _as(unsigned int addr, unsigned int endaddr)\n"
276                 "  * \\brief    " + action + "32 cache lines at once until end\n"
277                 "  *           address is reached.\n"
278                 "  *\n"
279                 "  * Will at least " + function + " 32 lines.\n"
280                 "  */\n"
281                 " void " + function + "_bulk_" + sys.argv[1]
282                 "             _as(unsigned int addr, unsigned int endaddr) {}\n"
283                 "\n"
284                 )
285
286             else:
287                 for cache_line_size in cache_line_size_list:
288                     for function in function_list:
289
290                         action = ""
291                         if function == "invalidate":
292                             action = "Invalidates "
293                         if function == "writeback":
294                             action = "Writes back "
295                         if function == "flush":
296                             action = "Flushes "
297
298                         filep.write(
299                             " /**\n"
300                             "  * \\internal\n"
301                             "  * \\fn      void " + function + "_" + cache_line_size
302                             "             _as(unsigned int addr, unsigned int endaddr)\n"
303                             "  * \\brief    " + action + "single cache lines until end address\n"

```

```

305         " * is reached.\n" +
306         " *\n" +
307         " * Will at least " + function + " one line.\n" +
308         " */\n" +
309         " void " + function + "_" + cache_line_size +
310         " _as(unsigned int addr, unsigned int endaddr) {}\n" +
311         "\n" +
312         " /**\n" +
313         " * \\internal\n" +
314         " * \\fn void " + function + "_bulk_" + cache_line_size +
315         " _as(unsigned int addr, unsigned int endaddr)\n" +
316         " * \\brief " + action + "32 cache lines at once until end\n" +
317         " * address is reached.\n" +
318         " *\n" +
319         " * Will at least " + function + " 32 lines.\n" +
320         " */\n" +
321         " void " + function + "_bulk_" + cache_line_size +
322         " _as(unsigned int addr, unsigned int endaddr) {}\n" +
323         "\n" +
324         )
325
326     filep.write(
327         " /**\n" +
328         " * @}\n" +
329         " */\n" +
330         "\n" +
331         " /**\n" +
332         " * @}\n" +
333         " */\n" +
334         "\n" +
335         "#endif /* DOXYGEN */\n" +
336         "\n" +
337         )
338
339
340
341     def linker_instructions(filep, specific_size):
342
343         filep.write(
344             "/*Header*****\n" +
345             "*****\n" +
346             "\n" +
347             "#ifndef DOXYGEN\n" +
348             "\n" +
349             ".text /* enter text section\n" +
350             " */\n" +
351             ".align 2 /* align Code to 2^2 Bytes\n" +
352             " */\n" +
353             "\n" +
354             "/* declare labels in here to be global */\n" +
355             ".globl enable_cache_as\n" +
356             ".globl disable_cache_as\n" +
357             ".globl cache_banks_as\n" +
358             ".globl cache_sets_as\n" +
359             ".globl mem_size_as\n" +
360             )
361
362         if specific_size:
363
364             filep.write(
365                 ".globl invalidate_" + sys.argv[1] + "_as\n" +
366                 ".globl invalidate_bulk_" + sys.argv[1] + "_as\n" +
367                 ".globl writeback_" + sys.argv[1] + "_as\n" +
368                 ".globl writeback_bulk_" + sys.argv[1] + "_as\n" +
369                 ".globl flush_" + sys.argv[1] + "_as\n" +
370                 ".globl flush_bulk_" + sys.argv[1] + "_as\n" +
371                 "\n" +
372                 )
373
374         else:
375
376             for cache_line_size in cache_line_size_list:
377                 filep.write(
378                     ".globl invalidate_" + cache_line_size + "_as\n" +
379                     ".globl invalidate_bulk_" + cache_line_size + "_as\n" +
380                     ".globl writeback_" + cache_line_size + "_as\n" +
381                     ".globl writeback_bulk_" + cache_line_size + "_as\n" +
382                     ".globl flush_" + cache_line_size + "_as\n" +
383                     ".globl flush_bulk_" + cache_line_size + "_as\n" +
384                     )
385
386             filep.write("\n")
387
388         filep.write(
389             "/* ParaNUT Custom Registers and Instructions */\n" +
390             "#include \"custom_RV32I.S\"\n" +
391             "\n" +

```

```

392     )
393
394
395 def common_functions(filep):
396
397     filep.write(
398         "enable_cache_as:\n"
399         "    fence\n"
400         "    li    t0,    3\n"
401         "    csrs pncache, t0\n"
402         "    ret\n"
403         "\n"
404         "/*-----*\n"
405         "-----*/\n"
406
407         "disable_cache_as:\n"
408         "    fence\n"
409         "    li    t0,    0\n"
410         "    csrw pncache, t0\n"
411         "    ret\n"
412         "\n"
413         "/*-----*\n"
414         "-----*/\n"
415
416         "cache_banks_as:\n"
417         "    csrr a0,    pncacheinfo\n"
418         "    srli a0,    a0,    8\n"
419         "    ret\n"
420         "\n"
421         "/*-----*\n"
422         "-----*/\n"
423
424         "cache_sets_as:\n"
425         "    csrr a0,    pncachesets\n"
426         "    ret\n"
427         "\n"
428         "/*-----*\n"
429         "-----*/\n"
430
431         "mem_size_as:\n"
432         "    csrr a0,    pnmemsize\n"
433         "    ret\n"
434         "\n"
435         "/*-----*\n"
436         "-----*/\n"
437
438     )
439
440 def macrocheck(funcname):
441
442     macroname = ""
443     if funcname == "invalidate":
444         macroname = "    CINV(10, "
445     elif funcname == "writeback":
446         macroname = "    CWB(10, "
447     elif funcname == "flush":
448         macroname = "    CFLUSH(10, "
449     else:
450         raise ValueError('This function name has no matching macro!')
451
452     return macroname
453
454 def create_function(filep, funcname, linesize):
455
456     macro = macrocheck(funcname)
457
458     #
459     # Single line function
460     #
461
462     # put the function name
463     filep.write(funcname + "_" + linesize + "_as:\n")
464
465     # put in a fence
466     filep.write("    fence\n")
467
468     # if linesize in byte is bigger than 2048, we cannot use addi later
469     if (int(linesize) / 8) >= 2048:
470         filep.write("    li t0, " + str(int(linesize) / 8) + "\n")
471
472     # put label on loop
473     looplabel = funcname + "_" + linesize + "_as_loop"
474     filep.write(looplabel + ":\n")
475
476     # actual operation
477     filep.write(macro + "0x000)\n")

```

```

478
479 # add linesize in byte
480 if (int(linesize) / 8) >= 2048:
481     filep.write("    add a0, a0, t0\n")
482 else:
483     filep.write("    addi a0, a0, " + str(int(linesize) / 8) + "\n")
484
485 # loop wrap
486 filep.write(
487     "    blt a0, a1, %s\n" % looplabel
488     "    ret\n"
489     "\n"
490     "/*-----*/\n"
491     "-----*/\n"
492     "\n"
493 )
494
495 #
496 # Bulk function
497 #
498
499 # put the function name
500 filep.write(funcname + "_bulk_" + linesize + "_as:\n")
501
502 # put in a fence
503 filep.write("    fence\n")
504
505 # get the loop label
506 looplabel = funcname + "_bulk_" + linesize + "_as_loop"
507
508 # if cache line in byte size does not fit into 2048 more than once, we can
509 # only use the single line action.
510 if (int(linesize) / 8) >= 2048:
511     filep.write("    j " + funcname + "_" + linesize + "_as\n\n")
512     return
513
514 # if the linesize and factors fit with no problem, we can roll them all out
515 if ((int(linesize) / 8) * 32) < 2048:
516
517     # put label on loop
518     filep.write(looplabel + ":\n")
519
520     # unroll loop 32 times
521     for i in range(32):
522         filep.write(macro + "%d\n" % ((int(linesize) / 8) * i))
523
524     # update start value
525     filep.write("    addi a0, a0, %d\n" % ((int(linesize) / 8) * 32))
526
527 else:
528
529     # check how many lines can be done at once
530     max_lines = int(math.ceil(2048 / float((int(linesize) / 8))))
531
532     # do not use addi later so we can possible fit one more line
533     filep.write("li t0, %i\n" % ((int(linesize) / 8) * max_lines))
534
535     # put label on loop
536     filep.write(looplabel + ":\n")
537
538     # unroll 32 times without branching
539     for i in range(0, 32, max_lines):
540
541         # unroll loop as many times as possible without addition
542         for j in range(max_lines):
543
544             filep.write(macro + "%d\n" % ((int(linesize) / 8) * j))
545
546         # update start value
547         filep.write("    add a0, a0, t0\n")
548
549     if not ((32 % max_lines) == 0):
550
551         # unroll loop for the missing lines
552         for i in range(0, 32 % max_lines, 1):
553             filep.write(macro + "%d\n" % ((int(linesize) / 8) * i))
554
555         # update start value (guaranteed to fit into 2048)
556         filep.write(
557             "    addi a0, a0, %d\n" % ((int(linesize) / 8) * (32 % max_lines))
558         )
559
560 # loop wrap
561 filep.write(
562     "    blt a0, a1, %s\n" % looplabel
563     "    ret\n"
564     "\n"

```



```

565     )
566
567 #Main#####
568 #
569 #
570 # Check if we are generating a specific cache line size or the auto file.
571 #
572
573 specific_size = sys.argv[1].isdigit()
574
575 #
576 # Check if cache line size is divisible by 32 or keyword auto.
577 #
578
579 if specific_size:
580     if (int(sys.argv[1]) % 32) != 0:
581         raise ValueError('Line size is not a multiple of word size')
582     elif sys.argv[1] != "auto":
583         raise ValueError('Given keyword was not \'auto\'!')
584
585 #
586 # Create/open the assembly source code file.
587 #
588
589 filename = "pn_cache/pn_cache_RV32I_" + sys.argv[1] + ".S"
590 filep = open(filename, "w+")
591
592 #
593 # Put in some pre-code stuff
594 #
595
596 # create warning
597 create_warning(filep)
598
599 # create Doxygen documentation
600 create_doxy(filep, specific_size)
601
602 # put in linker instructions
603 linker_instructions(filep, specific_size)
604
605 #
606 # Actual function code generation.
607 #
608 # Explanations:
609 #
610 # a0 (register 10) always contains start address.
611 # a1 (register 11) always contains end address.
612 #
613 # 0x100B is custom opcode for cache invalidate.
614 # 0x200B is custom opcode for cache writeback.
615 # 0x300B is custom opcode for cache flush.
616 #
617
618 filep.write(
619     "/*Functions*****" +
620     "*/\n"
621 )
622
623 #
624 #
625 # Put in the functions that every file needs and that are not dependent on line
626 # size.
627 #
628
629 common_functions(filep)
630
631 #
632 # all other functions for the specific cache line size files
633 #
634
635 if specific_size:
636     create_function(filep, "invalidate", sys.argv[1])
637
638     filep.write("/*-----" +
639                 "-----*/\n\n")
640
641     create_function(filep, "writeback", sys.argv[1])
642
643     filep.write("/*-----" +
644                 "-----*/\n\n")
645
646     create_function(filep, "flush", sys.argv[1])
647
648 #
649 # functions for auto file
650 #
651 #

```

```

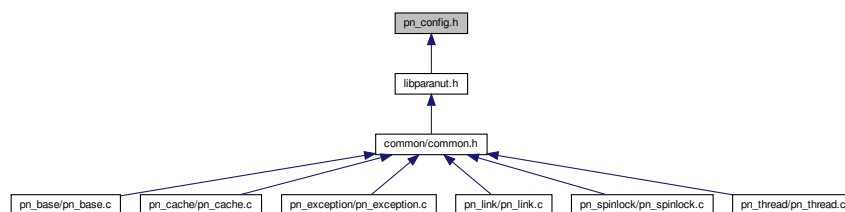
652
653 else:
654
655     for cache_line_size in cache_line_size_list:
656
657         for function in function_list:
658
659             create_function(file, function, cache_line_size)
660
661             if (cache_line_size != cache_line_size_list[-1]):
662                 if (function != "flush"):
663                     file.write("/*-----" +
664                               "-----*/\n\n")
665
666
667 filep.write(
668     "#endif /* !DOXYGEN */\n" +
669     "\n" +
670     "/*EOF*****" +
671     "*/\n")
672
673 #
674 # close file
675 #
676
677 filep.close()
678
679 #EOF#####

```

8.18 pn_config.h File Reference

See Documentaion of [libparanut Compile Time Parameters](#).

This graph shows which files directly or indirectly include this file:



Macros

- **#define PN_CACHE_LINESIZE**
Size of a cache line in bit.
- **#define PN_RWIDTH 32**
Register width in bit.
- **#define PN_COMPILE_RAW**
All security checks in libparanut are dropped if this is set to 1.
- **#define PN_WITH_BASE**
libparanut was compiled with [Base Module](#). Also check [Modules of the libparanut](#) for more information.
- **#define PN_WITH_CACHE**
libparanut was compiled with [Cache Module](#). Also check [Modules of the libparanut](#) for more information.
- **#define PN_WITH_LINK**
libparanut was compiled with [Link Module](#). Also check [Modules of the libparanut](#) for more information.
- **#define PN_WITH_THREAD**

libparanut was compiled with [Thread Module](#). Also check [Modules of the libparanut](#) for more information.

- `#define PN_WITH_EXCEPTION`

libparanut was compiled with [Exception Module](#). Also check [Modules of the libparanut](#) for more information.

- `#define PN_WITH_SPINLOCK`

libparanut was compiled with [Spinlock Module](#). Also check [Modules of the libparanut](#) for more information.

8.18.1 Detailed Description

See Documentaion of [libparanut Compile Time Parameters](#).

Please note that the file that is documented here is not the same file as the [pn_config.h](#) that is created by the [Makefile](#). This file is here for throwing errors when libparanut is compiled wrong. The [pn_config.h](#) that is created by the [Makefile](#) contains the actual defines that are then used by libparanut and by application.

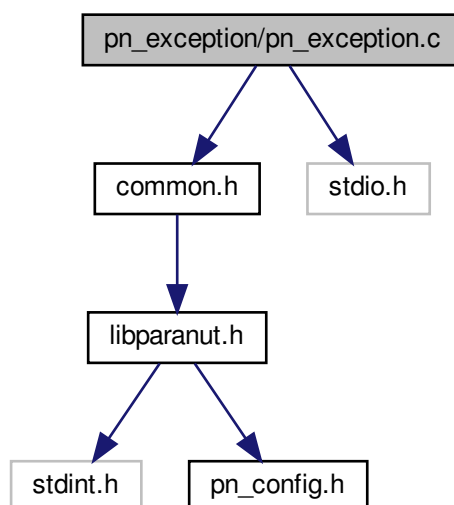
8.19 pn_exception/pn_exception.c File Reference

Contains (somewhat) architecture independent implementations of the [Exception Module](#) functions.

```
#include "common.h"
```

```
#include <stdio.h>
```

Include dependency graph for pn_exception.c:



Functions

- void [pn_exception_init](#) (void)
Initializes libparanut internal exception handling. Interrupts (not exceptions in general!) are disabled after. Should be called before using [pn_exception_set_handler](#)).
 - int [pn_exception_set_handler](#) (void(*handler)(unsigned int cause, unsigned int program_counter, unsigned int mtval), unsigned int exception_code)
Set your own exception handler.
 - void [pn_ecall](#) (void)
Raises an environment call exception.
 - void [pn_interrupt_enable](#) (void)
Enables interrupts only.
 - void [pn_interrupt_disable](#) (void)
Disables interrupts only.
 - void [pn_progress_mepc](#) (void)
Sets program counter of the register which keeps the exception return adress to next instruction.
-
- void [exception_init_as](#) (void)
Sets Machine Trap-Vector Base-Address Register correctly and disables interrupts.
 - void [exception_entry_as](#) (void)
Entry point for exceptions (includes interrupts).
 - [PN_CMSK read_PNX_as](#) (void)
Reads ParaNut CoPU exception pending register.
 - [PN_CMSK read_PNCAUSE_as](#) (void)
Reads ParaNut CoPU trap cause ID register.
 - [PN_CMSK read_PNEPC_as](#) (void)
Reads ParaNut CoPU exception program counter register.
 - unsigned int [read_MTVAL_as](#) (void)
Reads Machine Trap Value register.
 - void [write_PNXSEL_as](#) ([PN_CMSK](#) coremask)
Writes ParaNut CoPU exception select register.
 - void [write_MSTATUS_as](#) (unsigned int register)
Writes Machine Status register.
 - void [ecall_as](#) (void)
Causes environment call exception.
 - void [progress_mepc_as](#) (void)
Sets MEPC to next instruction.

8.19.1 Detailed Description

Contains (somewhat) architecture independent implementations of the [Exception Module](#) functions.

The exception module is not exactly architecture independent since it implements a RISC-V exception table.

Todo Layer this better in later versions of libparanut.

Functions with suffix `_as` are architecture specific and therefore implemented in the `pn_exception_$(PN_ISA).S` file in the same directory.

8.20 pn_exception/pn_exception_RV32I.S File Reference

Contains RV32I assembly implementations of assembly functions called in [pn_exception.c](#).

Functions

- void [exception_init_as](#) (void)
Sets Machine Trap-Vector Base-Address Register correctly and disables interrupts.
- void [exception_entry_as](#) (void)
Entry point for exceptions (includes interrupts).
- [PN_CMSK read_PNX_as](#) (void)
Reads ParaNut CoPU exception pending register.
- [PN_CMSK read_PNCAUSE_as](#) (void)
Reads ParaNut CoPU trap cause ID register.
- [PN_CMSK read_PNEPC_as](#) (void)
Reads ParaNut CoPU exception program counter register.
- unsigned int [read_MTVAl_as](#) (void)
Reads Machine Trap Value register.
- void [write_PNXSEL_as](#) ([PN_CMSK](#) coremask)
Writes ParaNut CoPU exception select register.
- void [write_MSTATUS_as](#) (unsigned int register)
Writes Machine Status register.
- void [ecall_as](#) (void)
Causes environment call exception.
- void [progress_mepc_as](#) (void)
Sets MEPC to next instruction.

8.20.1 Detailed Description

Contains RV32I assembly implementations of assembly functions called in [pn_exception.c](#).

```

1 /*
2  * Copyright 2019-2020 Anna Pfuetzner (<annakerstin.pfuetzner@gmail.com>)
3  *
4  * Redistribution and use in source and binary forms, with or without
5  * modification, are permitted provided that the following conditions are met:
6  *
7  * 1. Redistributions of source code must retain the above copyright notice,
8  * this list of conditions and the following disclaimer.
9  *
10 * 2. Redistributions in binary form must reproduce the above copyright notice,
11 * this list of conditions and the following disclaimer in the documentation
12 * and/or other materials provided with the distribution.
13 *
14 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
15 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
16 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
17 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
18 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
19 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
20 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
21 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
22 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
23 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
24 * POSSIBILITY OF SUCH DAMAGE.

```

```

25 */
26
38 /*
39  * Put in here so Doxygen will know that it is implemented in this file.
40  * Sadly, Doxygen has no built in assembly interpreter, so we are stuck with
41  * this.
42  */
43
44 #ifdef DOXYGEN
45
46     void exception_init_as(void) {}
47
48     void exception_entry_as(void) {}
49
50     PN_CMSK read_PNX_as(void) {}
51
52     PN_CMSK read_PNCAUSE_as(void) {}
53
54     PN_CMSK read_PNEPC_as(void) {}
55
56     unsigned int read_MTVAL_as(void) {}
57
58     void write_PNXSEL_as(PN_CMSK coremask) {}
59
60     void write_MSTATUS_as(unsigned int register) {}
61
62     void ecall_as(void) {}
63
64     void progress_mepc_as(void) {}
65
66 #endif /* DOXYGEN */
67
68 /*Header*****
69
70 #ifndef DOXYGEN
71
72     .text                                /* enter text section                */
73     .align 2                            /* align Code to 2^2 Bytes            */
74
75     /* declare labels in here to be global */
76     .globl exception_init_as
77     .globl exception_entry_as
78     .globl read_PNX_as
79     .globl read_PNCAUSE_as
80     .globl read_PNEPC_as
81     .globl read_MTVAL_as
82     .globl write_PNXSEL_as
83     .globl write_MSTATUS_as
84     .globl ecall_as
85     .globl progress_mepc_as
86
87     /* ParaNUT Custom Registers and Instructions */
88     #include "custom_RV32I.S"
89
90     /*Functions*****
91
92     exception_init_as:
93
94         li    t0,    0                    /* set t0 to 0                        */
95         csrw  mstatus, t0                /* disable interrupts                 */
96         la    t0,    exception_entry_as /* load address of exception_entry_as */
97         csrw  mtvec,  t0                  /* set trap-handler base address to   */
98                                         /* exception_entry_as                 */
99         ret                                /* return                             */
100
101
102     /*-----*/
103
104     exception_entry_as:
105
106     /*
107     * Save away registers on stack.
108     */
109
110     addi    sp,      sp,      -124
111     sw      x1,      0*4(sp)
112     sw      x2,      1*4(sp)
113     sw      x3,      2*4(sp)
114     sw      x4,      3*4(sp)
115     sw      x5,      4*4(sp)
116     sw      x6,      5*4(sp)
117     sw      x7,      6*4(sp)
118     sw      x8,      7*4(sp)
119     sw      x9,      8*4(sp)
120     sw      x10,     9*4(sp)
121     sw      x11,     10*4(sp)

```

```

204     sw    x12,    11*4(sp)
205     sw    x13,    12*4(sp)
206     sw    x14,    13*4(sp)
207     sw    x15,    14*4(sp)
208     sw    x16,    15*4(sp)
209     sw    x17,    16*4(sp)
210     sw    x18,    17*4(sp)
211     sw    x19,    18*4(sp)
212     sw    x20,    19*4(sp)
213     sw    x21,    20*4(sp)
214     sw    x22,    21*4(sp)
215     sw    x23,    22*4(sp)
216     sw    x24,    23*4(sp)
217     sw    x25,    24*4(sp)
218     sw    x26,    25*4(sp)
219     sw    x27,    26*4(sp)
220     sw    x28,    27*4(sp)
221     sw    x29,    28*4(sp)
222     sw    x30,    29*4(sp)
223     sw    x31,    30*4(sp)
224
225     /*
226     * Check if we got an interrupt (asynchronous exception) or a synchronous
227     * exception.
228     * If mcause has the first bit set (which makes it a negative number), we got
229     * an actual interrupt!
230     */
231
232     csrr   a0,      mcause           /* write machine trap cause to a0 */
233     csrr   a1,      mepc            /* write exception program count to a1 */
234     mv     a2,      sp              /* copy stack pointer to a2 */
235     bltz   a0,      interrupt       /* check if first bit set */
236
237
238 exception:
239
240     /*
241     * If we got here, it means we have an exception, not an interrupt.
242     * Call exception handler. Note that MEPC points to the instruction that
243     * caused the exception.
244     */
245
246     jal    handle_exc               /* call exception handler */
247     j      return                   /* return from trap */
248
249
250 interrupt:
251
252     /*
253     * If we got here, it means we have an interrupt, not an exception.
254     * First, remove the interrupt flag, then call actual interrupt handler.
255     * Afterwards, check if interrupts were previously enabled (mstatus.MPIE) and
256     * enable interrupts again if they were.
257     * For interrupts, MEPC points to the instruction where execution should
258     * resume after interrupt is handled.
259     */
260
261     slli   a0,      a0,      1       /* shift a0 to the left by one bit */
262     srli   a0,      a0,      1       /* shift a0 to the right by one bit */
263
264     jal    handle_int               /* call interrupt handler */
265
266     csrr   a0,      mstatus          /* load machine status into a0 */
267     andi   a0,      a0,      0x80    /* check on MPIE bit */
268     srli   a0,      a0,      4       /* if MPIE was set before, set MIE */
269     csrs   mstatus, a0              /* activate all that is set in a0 */
270
271 return:
272
273     /*
274     * Get back the registers from stack.
275     */
276
277     lw     x1,      0*4(sp)
278     lw     x2,      1*4(sp)
279     lw     x3,      2*4(sp)
280     lw     x4,      3*4(sp)
281     lw     x5,      4*4(sp)
282     lw     x6,      5*4(sp)
283     lw     x7,      6*4(sp)
284     lw     x8,      7*4(sp)
285     lw     x9,      8*4(sp)
286     lw     x10,     9*4(sp)
287     lw     x11,     10*4(sp)
288     lw     x12,     11*4(sp)
289     lw     x13,     12*4(sp)
290     lw     x14,     13*4(sp)

```

```

291     lw     x15,      14*4(sp)
292     lw     x16,      15*4(sp)
293     lw     x17,      16*4(sp)
294     lw     x18,      17*4(sp)
295     lw     x19,      18*4(sp)
296     lw     x20,      19*4(sp)
297     lw     x21,      20*4(sp)
298     lw     x22,      21*4(sp)
299     lw     x23,      22*4(sp)
300     lw     x24,      23*4(sp)
301     lw     x25,      24*4(sp)
302     lw     x26,      25*4(sp)
303     lw     x27,      26*4(sp)
304     lw     x28,      27*4(sp)
305     lw     x29,      28*4(sp)
306     lw     x30,      29*4(sp)
307     lw     x31,      30*4(sp)
308     addi   sp,       sp,      124
309
310     mret                                /* machine mode exception return */
311
312 /*-----*/
313
314 read_PNX_as:
315
316     csrr   a0,        pnx              /* read pnx */
317     ret                                /* return */
318
319 /*-----*/
320
321 read_PNCAUSE_as:
322
323     csrr   a0,        pncause          /* read pncause */
324     ret                                /* return */
325
326 /*-----*/
327
328 read_PNEPC_as:
329
330     csrr   a0,        pnepc           /* read pnepc */
331     ret                                /* return */
332
333 /*-----*/
334
335 read_MTVAL_as:
336
337     csrr   a0,        mtval           /* read mtval */
338     ret                                /* return */
339
340 /*-----*/
341
342 write_PNXSEL_as:
343
344     csrw   pnxsel,    a0              /* write pnxsel */
345     ret                                /* return */
346
347 /*-----*/
348
349 write_MSTATUS_as:
350
351     csrw   mstatus,   a0              /* write mstatus */
352     ret                                /* return */
353
354 /*-----*/
355
356 ecall_as:
357
358     ecall                                /* cause environment exception */
359     ret                                /* return */
360
361 /*-----*/
362
363 progress_mepc_as:
364
365     csrr   t0,        mepc            /* read MEPC into t0 */
366     addi   t0,        t0,      4      /* set to next instruction */
367     csrw   mepc,      t0              /* write back */
368     ret                                /* return */
369
370 #endif /* !DOXYGEN */
371
372 /*EOF******/

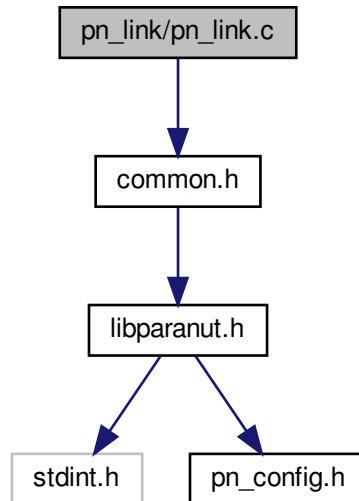
```


8.21 pn_link/pn_link.c File Reference

Contains architecture independent implementations of the [Link Module](#) functions.

```
#include "common.h"
```

Include dependency graph for pn_link.c:



Functions

- [PN_CMSK read_PNLM_as](#) (void)
Reads PNLM register.
- [PN_CID pn_begin_linked](#) (PN_NUMC numcores)
Links a given number of CoPUs to the CePU.
- [PN_CID pn_begin_linked_m](#) (PN_CMSK coremask)
Links the CPUs specified in the coremask.
- [PN_CID pn_begin_linked_gm](#) (PN_CMSK *coremask_array, PN_NUMG array_size)
Links the CPUs specified in the coremask_array.
- int [pn_end_linked](#) (void)
Ends linked execution.
- [PN_CID set_linked_as](#) (PN_CMSK coremask)
Puts the cores marked in the bitmask to linked mode and enables them.
- void [write_PNLM_as](#) (PN_CMSK coremask)
Writes PNLM register.

8.21.1 Detailed Description

Contains architecture independent implementations of the [Link Module](#) functions.

Functions with suffix `_as` are architecture specific and therefore implemented in the `pn_link_$(PN_ISA).S` file in the same directory.

8.22 pn_link/pn_link_RV32I.S File Reference

Contains RV32I assembly implementations of assembly functions called in [pn_link.c](#).

Functions

- [PN_CID set_linked_as](#) ([PN_CMSK](#) coremask)
Puts the cores marked in the bitmask to linked mode and enables them.
- [PN_CMSK read_PNLM_as](#) (void)
Reads PNLM register.
- void [write_PNLM_as](#) ([PN_CMSK](#) coremask)
Writes PNLM register.

8.22.1 Detailed Description

Contains RV32I assembly implementations of assembly functions called in [pn_link.c](#).

```

1 /*
2  * Copyright 2019-2020 Anna Pfuetzner (<annakerstin.pfuetzner@gmail.com>)
3  *
4  * Redistribution and use in source and binary forms, with or without
5  * modification, are permitted provided that the following conditions are met:
6  *
7  * 1. Redistributions of source code must retain the above copyright notice,
8  * this list of conditions and the following disclaimer.
9  *
10 * 2. Redistributions in binary form must reproduce the above copyright notice,
11 * this list of conditions and the following disclaimer in the documentation
12 * and/or other materials provided with the distribution.
13 *
14 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
15 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
16 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
17 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
18 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
19 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
20 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
21 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
22 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
23 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
24 * POSSIBILITY OF SUCH DAMAGE.
25 */
26
27 /*
28  * Put in here so Doxygen will know that it is implemented in this file.
29  * Sadly, Doxygen has no built in assembly interpreter, so we are stuck with
30  * this.
31  */
32
33 #ifdef DOXYGEN
34
35     PN_CID set_linked_as(PN_CMSK coremask) {}

```

```

68
69     PN_CMSK read_PNLM_as(void) {}
70
71     void write_PNLM_as(PN_CMSK coremask) {}
72
73 #endif /* DOXYGEN */
74
75 /*Header*****
76
77 #ifndef DOXYGEN
78
79 .text                /* enter text section                */
80 .align    2          /* align code to 2^2 bytes                */
81
82 /* declare labels in here to be global */
83 .globl    set_linked_as
84 .globl    read_PNLM_as
85 .globl    write_PNLM_as
86
87 /* ParaNut Custom Registers and Instructions */
88 #include "custom_RV32I.S"
89
90 /*Functions*****
91
92 set_linked_as:
93
94     /*
95     * Save away the CePUs state so the CoPUs can recover it later.
96     */
97
98     addi    sp,    sp,    -52                /* get space on stack                */
99
100    /* zero does not need saving                */
101    sw       ra,    0*4(sp)                  /* save return address                */
102    /* stack pointer stored elsewhere                */
103    /* global pointer is read later                */
104    /* thread pointer is not changed                */
105    /* temporary regs are caller saved                */
106    sw       s0,    1*4(sp)                  /* save callee saved register 0                */
107    sw       s1,    2*4(sp)                  /* save callee saved register 1                */
108    /* argument regs are caller saved                */
109    sw       s2,    3*4(sp)                  /* save callee saved register 2                */
110    sw       s3,    4*4(sp)                  /* save callee saved register 3                */
111    sw       s4,    5*4(sp)                  /* save callee saved register 4                */
112    sw       s5,    6*4(sp)                  /* save callee saved register 5                */
113    sw       s6,    7*4(sp)                  /* save callee saved register 6                */
114    sw       s7,    8*4(sp)                  /* save callee saved register 7                */
115    sw       s8,    9*4(sp)                  /* save callee saved register 8                */
116    sw       s9,    10*4(sp)                 /* save callee saved register 9                */
117    sw       s10,   11*4(sp)                 /* save callee saved register 10               */
118    sw       s11,   12*4(sp)                 /* save callee saved register 11               */
119    /* temporary regs are caller saved                */
120
121    /*
122    * Put the stack pointer and thread pointer somewhere safe so that the CoPUs
123    * can calculate their own stack pointer later.
124    */
125
126    la       t0,    sp_loc                  /* load adress of location                */
127    sw       sp,    0(t0)                   /* put in stack pointer                */
128    la       t0,    tp_loc                  /* load adress of location                */
129    sw       tp,    0(t0)                   /* put in thread pointer                */
130
131    /*
132    * Save a0 away since it will get overridden soon.
133    */
134
135    mv       s0,    a0                      /* copy a0 to s0                */
136
137    /*
138    * Put in a fence to make sure everything is properly stored away.
139    */
140
141    fence                                /* synchronize memory                */
142
143    /*
144    * Activate the linked mode.
145    */
146
147    csrwr    pnln, a0                      /* sets pnln to passed bitmask                */
148    csrwr    pnnc, a0                      /* sets pnnc to passed bitmask                */
149
150    /*
151    * Get back the registers so all CoPUs have the same values to work with.
152    * We are now linked!
153    * Now, to further proceed, we have to do the following:
154    * - Initialize gp of CoPUs.
155    */

```



```

264  lw   s6,    7*4(sp)           /* load callee saved register 6    */
265  lw   s7,    8*4(sp)           /* load callee saved register 7    */
266  lw   s8,    9*4(sp)           /* load callee saved register 8    */
267  lw   s9,   10*4(sp)           /* load callee saved register 9    */
268  lw   s10,  11*4(sp)           /* load callee saved register 10   */
269  lw   s11,  12*4(sp)           /* load callee saved register 11   */
270
271  addi  sp,   sp,    52           /* set back stack pointer          */
272
273  /*
274  * Get the core ID, store in return value, return
275  */
276
277  csrr  a0,   mhartid           /* put mhartid into a0             */
278  ret                                /* return                          */
279
280  /*-----*/
281
282  read_PNLM_as:
283
284  csrr  a0,   pnlm              /* put pnlm in return value        */
285  ret                                /* return                          */
286
287  /*-----*/
288
289  write_PNLM_as:
290
291  csrw  pnlm,  a0              /* put pnlm in return value        */
292  ret                                /* return                          */
293
294  #endif /* !DOXYGEN */
295
296  /*EOF******/

```

8.23 pn_spinlock/pn_spinlock.c File Reference

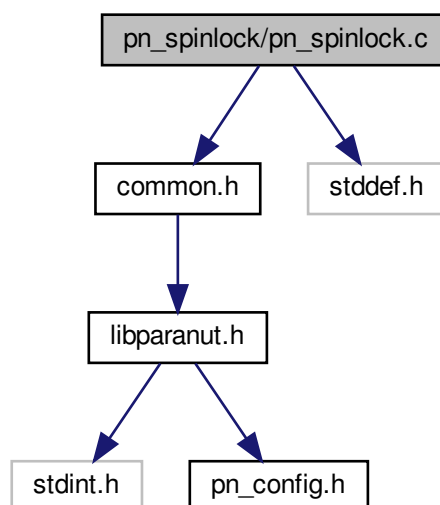
Contains architecture independent implementations of the [Spinlock Module](#) functions.

```

#include "common.h"
#include <stddef.h>

```

Include dependency graph for pn_spinlock.c:



Functions

- int [pn_spinlock_init](#) ([_pn_spinlock](#) *spinlock)
Creates a lock.
 - int [pn_spinlock_lock](#) ([_pn_spinlock](#) *spinlock)
Waits for a lock. Forever, if it must. Use with caution.
 - int [pn_spinlock_trylock](#) ([_pn_spinlock](#) *spinlock)
Tries to acquire a lock. Nonblocking.
 - int [pn_spinlock_unlock](#) ([_pn_spinlock](#) *spinlock)
Unlocks a lock.
 - int [pn_spinlock_destroy](#) ([_pn_spinlock](#) *spinlock)
Destroys a lock.
-
- int [init_as](#) ([_pn_spinlock](#) *spinlock)
Sets lock to free, no conditions checked.
 - int [trylock_as](#) ([_pn_spinlock](#) *spinlock, [PN_CID](#) coreid)
Tries locking the lock. Fails if not free.
 - int [unlock_as](#) ([_pn_spinlock](#) *spinlock, [PN_CID](#) coreid)
Tries unlocking. Fails if not owned by current hart.
 - int [destroy_as](#) ([_pn_spinlock](#) *spinlock, [PN_CID](#) coreid)
Sets lock to dead if it's free or it's owned by current hart.

8.23.1 Detailed Description

Contains architecture independent implementations of the [Spinlock Module](#) functions.

Functions with suffix `_as` are architecture specific and therefore implemented in the `pn_spinlock_$(PN_ISA).S` file in the same directory.

8.24 pn_spinlock/pn_spinlock_RV32I.S File Reference

Contains RV32I assembly implementations of assembly functions called in [pn_spinlock.c](#).

Functions

- int [init_as](#) ([_pn_spinlock](#) *spinlock)
Sets lock to free, no conditions checked.
- int [trylock_as](#) ([_pn_spinlock](#) *spinlock, [PN_CID](#) coreid)
Tries locking the lock. Fails if not free.
- int [unlock_as](#) ([_pn_spinlock](#) *spinlock, [PN_CID](#) coreid)
Tries unlocking. Fails if not owned by current hart.
- int [destroy_as](#) ([_pn_spinlock](#) *spinlock, [PN_CID](#) coreid)
Sets lock to dead if it's free or it's owned by current hart.

8.24.1 Detailed Description

Contains RV32I assembly implementations of assembly functions called in [pn_spinlock.c](#).

When changing anything in here, make sure that these changes reflect on the internal lock of the [Thread Module](#). The internal lock there needs to be created as an initialized lock!

```

1  /*
2  * Copyright 2019-2020 Anna Pfuetzner (<annakerstin.pfuetzner@gmail.com>)
3  *
4  * Redistribution and use in source and binary forms, with or without
5  * modification, are permitted provided that the following conditions are met:
6  *
7  * 1. Redistributions of source code must retain the above copyright notice,
8  * this list of conditions and the following disclaimer.
9  *
10 * 2. Redistributions in binary form must reproduce the above copyright notice,
11 * this list of conditions and the following disclaimer in the documentation
12 * and/or other materials provided with the distribution.
13 *
14 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
15 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
16 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
17 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
18 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
19 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
20 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
21 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
22 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
23 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
24 * POSSIBILITY OF SUCH DAMAGE.
25 */
26
42 /*
43 * Put in here so Doxygen will know that it is implemented in this file.
44 * Sadly, Doxygen has no built in assembly interpreter, so we are stuck with
45 * this.
46 */
47
48 #ifndef DOXYGEN
49
64     int init_as(_pn_spinlock *spinlock) {}
65
71     int trylock_as(_pn_spinlock *spinlock, PN_CID coreid) {}
72
78     int unlock_as(_pn_spinlock *spinlock, PN_CID coreid) {}
79
85     int destroy_as(_pn_spinlock *spinlock, PN_CID coreid) {}
86
95 #endif /* DOXYGEN */
96
97 /*Header*****
98
99 #ifndef DOXYGEN
100
101 .text                                /* enter text section                */
102 .align 2                            /* align Code to 2^2 Bytes            */
103
104 /* declare labels in here to be global */
105 .globl  init_as
106 .globl  trylock_as
107 .globl  unlock_as
108 .globl  destroy_as
109
110 /* local defines */
111 .equ    DEAD, -2
112 .equ    FREE, -1
113
114 /* ParaNut Custom Registers and Instructions */
115 #include "custom_RV32I.S"
116
117 /*
118 * Naming conventions in this file:
119 *
120 * a0 contains address of owner_ID or error value on return. Only the Helpers at
121 * the bottom are allowed to meddle with it.
122 *
123 * a1 contains ID of executing core.
124 *
125 * t0 is copy of owner_ID. It takes values DEAD, FREE, or ID of the CPU owning
126 * the lock.
127 */

```

```

128 * t1 indicates if original value has been touched when trying to write the copy
129 * back to memory. This will lead to occ_error.
130 *
131 * Unexpected values of t0 will lead to param_error.
132 *
133 * t2 is DEAD, t3 is FREE.
134 */
135
136 /*Functions*****
137
138 init_as:
139
140     lr.w  t0,      (a0)      /* load owner_ID into t0      */
141     li    t0,      FREE      /* bring me to live (wake me up inside) */
142     j     store     /* store back and evaluate success      */
143
144 /*-----*/
145
146 trylock_as:
147
148     li    t3,      FREE      /* load FREE      */
149     lr.w  t0,      (a0)      /* load owner_ID into t0      */
150     bne   t0,      t3,      occ_error /* if lock is not free, return occerr */
151     mv    t0,      a1        /* set owner_ID to core ID      */
152     j     store     /* store back and evaluate success      */
153
154 /*-----*/
155
156 unlock_as:
157
158     lr.w  t0,      (a0)      /* load owner_ID into t0      */
159     bne   t0,      a1,      param_error /* if lock not CPU owned, param_error */
160     li    t0,      FREE      /* set owner_ID to FREE      */
161     j     store     /* store back and evaluate success      */
162
163 /*-----*/
164
165 destroy_as:
166
167     li    t3,      FREE      /* load FREE      */
168     lr.w  t0,      (a0)      /* load owner_ID into t0      */
169     beq   t0,      t3,      free_or_mine /* if lock is free, skip next instruction */
170     bne   t0,      a1,      param_error /* if lock not CPU owned, return paramerr */
171 free_or_mine:
172     li    t0,      DEAD      /* kill lock      */
173     j     store     /* store back and evaluate success      */
174
175 /*Helpers*****
176
177 store:
178     sc.w  t1,      t0,      (a0)      /* write back owner ID      */
179     bnez  t1,      occ_error /* jump if we did not get reservation */
180     j     success_return /* successful execution      */
181
182 /*-----*/
183
184 success_return:
185     li    a0,      0          /* load return value for success      */
186     ret
187
188 /*-----*/
189
190 param_error:
191     li    a0,      1          /* load return value for parameter error */
192     ret
193
194 /*-----*/
195
196 occ_error:
197     li    a0,      2          /* load return value for occupation error */
198     ret
199
200 #endif /* !DOXYGEN */
201
202 /*EOF*****

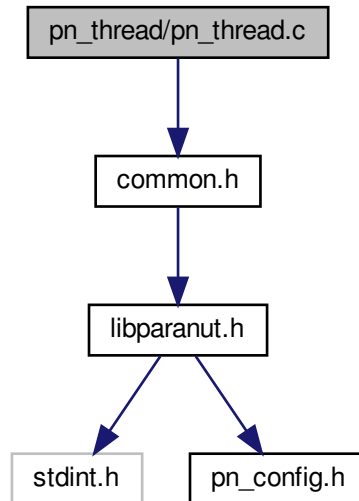
```

8.25 pn_thread/pn_thread.c File Reference

Contains architecture independent implementations of the [Thread Module](#) functions.


```
#include "common.h"
```

Include dependency graph for pn_thread.c:



Functions

- void `pn_thread_entry` ()
Function that has to be called for CoPUs at the end of the startup code.
- `PN_CID` `pn_begin_threaded` (`PN_NUMC` numcores)
Puts numcores CPUs in threaded mode.
- `PN_CID` `pn_begin_threaded_m` (`PN_CMSK` coremask)
Puts the CPUs specified in the coremask in threaded mode.
- `PN_CID` `pn_begin_threaded_gm` (`PN_CMSK` *coremask_array, `PN_NUMG` array_size)
Puts the CPUs specified in the coremask_array in threaded mode.
- int `pn_end_threaded` (void)
Ends threaded execution.

- void `set_threaded_as` (`PN_CMSK` coremask)
Takes the linked flag away from all cores and enables the cores in the coremask.
- void `enter_threaded_mode_as` (void)
Prepares CoPU for threaded mode by taking the CePUs stack.

8.25.1 Detailed Description

Contains architecture independent implementations of the [Thread Module](#) functions.

Functions with suffix `_as` are architecture specific and therefore implemented in the `pn_thread_$(PN_ISA).S` file in the same directory.

8.26 pn_thread/pn_thread_RV32I.S File Reference

Contains RV32I assembly implementations of assembly functions called in [pn_thread.c](#).

Functions

- void [set_threaded_as](#) (PN_CMSK coremask)
Takes the linked flag away from all cores and enables the cores in the coremask.
- void [enter_threaded_mode_as](#) (void)
Prepares CoPU for threaded mode by taking the CePUs stack.

8.26.1 Detailed Description

Contains RV32I assembly implementations of assembly functions called in [pn_thread.c](#).

```

1  /*
2  * Copyright 2019-2020 Anna Pfuetzner (<annakerstin.pfuetzner@gmail.com>)
3  *
4  * Redistribution and use in source and binary forms, with or without
5  * modification, are permitted provided that the following conditions are met:
6  *
7  * 1. Redistributions of source code must retain the above copyright notice,
8  * this list of conditions and the following disclaimer.
9  *
10 * 2. Redistributions in binary form must reproduce the above copyright notice,
11 * this list of conditions and the following disclaimer in the documentation
12 * and/or other materials provided with the distribution.
13 *
14 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
15 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
16 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
17 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
18 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
19 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
20 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
21 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
22 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
23 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
24 * POSSIBILITY OF SUCH DAMAGE.
25 */
26
27
28 /*
29 * Put in here so Doxygen will know that it is implemented in this file.
30 * Sadly, Doxygen has no built in assembly interpreter, so we are stuck with
31 * this.
32 */
33
34 #ifdef DOXYGEN
35     void set_threaded_as(PN_CMSK coremask) {}
36
37     void enter_threaded_mode_as(void) {}
38 #endif /* DOXYGEN */
39
40 /*Header*****
41
42 #ifndef DOXYGEN
43
44 .text                /* enter text section                */
45 .align 2             /* align Code to 2^2 Bytes                */
46
47 /* declare labels in here to be global */
48 .globl  set_threaded_as
49 .globl  enter_threaded_mode_as
50
51 /* ParaNut Custom Registers and Instructions */

```

```

99 #include "custom_RV32I.S"
100
101 /*Functions*****
102
103 set_threaded_as:
104
105 /*
106  * Save away the CePUs state so the CoPUs can recover it later.
107  */
108
109     addi    sp,    sp,    -52                /* get space on stack */
110
111     /* zero does not need saving */
112     sw      ra,    0*4(sp)                  /* save return address */
113     /* stack pointer stored elsewhere */
114     /* global pointer is read later */
115     /* thread pointer is not changed */
116     /* temporary regs are caller saved */
117     sw      s0,    1*4(sp)                  /* save callee saved register 0 */
118     sw      s1,    2*4(sp)                  /* save callee saved register 1 */
119     /* argument regs are caller saved */
120     sw      s2,    3*4(sp)                  /* save callee saved register 2 */
121     sw      s3,    4*4(sp)                  /* save callee saved register 3 */
122     sw      s4,    5*4(sp)                  /* save callee saved register 4 */
123     sw      s5,    6*4(sp)                  /* save callee saved register 5 */
124     sw      s6,    7*4(sp)                  /* save callee saved register 6 */
125     sw      s7,    8*4(sp)                  /* save callee saved register 7 */
126     sw      s8,    9*4(sp)                  /* save callee saved register 8 */
127     sw      s9,    10*4(sp)                 /* save callee saved register 9 */
128     sw      s10,   11*4(sp)                 /* save callee saved register 10 */
129     sw      s11,   12*4(sp)                 /* save callee saved register 11 */
130     /* temporary regs are caller saved */
131
132
133 /*
134  * Put the stack pointer and thread pointer somewhere safe so that the CoPUs
135  * can calculate their own stack pointer later.
136  */
137
138     la      t0,    sp_loc                  /* load adress of location */
139     sw      sp,    0(t0)                    /* put in stack pointer */
140     la      t0,    tp_loc                  /* load adress of location */
141     sw      tp,    0(t0)                    /* put in thread pointer */
142
143 /*
144  * Save a0 away since it will get overridden soon.
145  */
146
147     mv      s0,    a0                      /* copy a0 to s0 */
148
149 /*
150  * Copy the whole thread stack to shared memory location.
151  * Look at the following ASCII art to understand what I am doing.
152  * To the left, you can see the thread space of the CePU. The stack is the
153  * bottom part. It grows "downwards" which means it goes against normal address
154  * growth direction (so upwards in this picture).
155  * On the right, you can see the shared memory. This is where we want to copy
156  * the stack.
157  *
158  *      tp-----
159  *      |         |
160  *      |         |
161  *      |         |
162  *      |         |
163  *      |         |
164  *      |         |
165  *      |         |
166  *      |         |
167  *      |         |
168  *      |         |
169  *      |         |
170  *      |         |
171  *      |         |
172  *      |         |
173  *      |         |
174  *      |         |
175  *      |         |
176  *      |         |
177  *      |         |
178  *      |         |
179  *      |         |
180  *      |         |
181  *      |         |
182  *      |         |
183  *      |         |
184  *      |         |
185  *      |         |
186  *      |         |
187  *      |         |
188  *      |         |
189  *      |         |
190  *      |         |
191  *      |         |
192  *      |         |
193  *      |         |
194  *      |         |
195  *      |         |
196  *      |         |
197  *      |         |
198  *      |         |
199  *      |         |
200  *      |         |
201  *      |         |
202  *      |         |
203  *      |         |
204  *      |         |
205  *      |         |
206  *      |         |
207  *      |         |
208  *      |         |
209  *      |         |
210  *      |         |
211  *      |         |
212  *      |         |
213  *      |         |
214  *      |         |
215  *      |         |
216  *      |         |
217  *      |         |
218  *      |         |
219  *      |         |
220  *      |         |
221  *      |         |
222  *      |         |
223  *      |         |
224  *      |         |
225  *      |         |
226  *      |         |
227  *      |         |
228  *      |         |
229  *      |         |
230  *      |         |
231  *      |         |
232  *      |         |
233  *      |         |
234  *      |         |
235  *      |         |
236  *      |         |
237  *      |         |
238  *      |         |
239  *      |         |
240  *      |         |
241  *      |         |
242  *      |         |
243  *      |         |
244  *      |         |
245  *      |         |
246  *      |         |
247  *      |         |
248  *      |         |
249  *      |         |
250  *      |         |
251  *      |         |
252  *      |         |
253  *      |         |
254  *      |         |
255  *      |         |
256  *      |         |
257  *      |         |
258  *      |         |
259  *      |         |
260  *      |         |
261  *      |         |
262  *      |         |
263  *      |         |
264  *      |         |
265  *      |         |
266  *      |         |
267  *      |         |
268  *      |         |
269  *      |         |
270  *      |         |
271  *      |         |
272  *      |         |
273  *      |         |
274  *      |         |
275  *      |         |
276  *      |         |
277  *      |         |
278  *      |         |
279  *      |         |
280  *      |         |
281  *      |         |
282  *      |         |
283  *      |         |
284  *      |         |
285  *      |         |
286  *      |         |
287  *      |         |
288  *      |         |
289  *      |         |
290  *      |         |
291  *      |         |
292  *      |         |
293  *      |         |
294  *      |         |
295  *      |         |
296  *      |         |
297  *      |         |
298  *      |         |
299  *      |         |
300  *      |         |
301  *      |         |
302  *      |         |
303  *      |         |
304  *      |         |
305  *      |         |
306  *      |         |
307  *      |         |
308  *      |         |
309  *      |         |
310  *      |         |
311  *      |         |
312  *      |         |
313  *      |         |
314  *      |         |
315  *      |         |
316  *      |         |
317  *      |         |
318  *      |         |
319  *      |         |
320  *      |         |
321  *      |         |
322  *      |         |
323  *      |         |
324  *      |         |
325  *      |         |
326  *      |         |
327  *      |         |
328  *      |         |
329  *      |         |
330  *      |         |
331  *      |         |
332  *      |         |
333  *      |         |
334  *      |         |
335  *      |         |
336  *      |         |
337  *      |         |
338  *      |         |
339  *      |         |
340  *      |         |
341  *      |         |
342  *      |         |
343  *      |         |
344  *      |         |
345  *      |         |
346  *      |         |
347  *      |         |
348  *      |         |
349  *      |         |
350  *      |         |
351  *      |         |
352  *      |         |
353  *      |         |
354  *      |         |
355  *      |         |
356  *      |         |
357  *      |         |
358  *      |         |
359  *      |         |
360  *      |         |
361  *      |         |
362  *      |         |
363  *      |         |
364  *      |         |
365  *      |         |
366  *      |         |
367  *      |         |
368  *      |         |
369  *      |         |
370  *      |         |
371  *      |         |
372  *      |         |
373  *      |         |
374  *      |         |
375  *      |         |
376  *      |         |
377  *      |         |
378  *      |         |
379  *      |         |
380  *      |         |
381  *      |         |
382  *      |         |
383  *      |         |
384  *      |         |
385  *      |         |
386  *      |         |
387  *      |         |
388  *      |         |
389  *      |         |
390  *      |         |
391  *      |         |
392  *      |         |
393  *      |         |
394  *      |         |
395  *      |         |
396  *      |         |
397  *      |         |
398  *      |         |
399  *      |         |
400  *      |         |
401  *      |         |
402  *      |         |
403  *      |         |
404  *      |         |
405  *      |         |
406  *      |         |
407  *      |         |
408  *      |         |
409  *      |         |
410  *      |         |
411  *      |         |
412  *      |         |
413  *      |         |
414  *      |         |
415  *      |         |
416  *      |         |
417  *      |         |
418  *      |         |
419  *      |         |
420  *      |         |
421  *      |         |
422  *      |         |
423  *      |         |
424  *      |         |
425  *      |         |
426  *      |         |
427  *      |         |
428  *      |         |
429  *      |         |
430  *      |         |
431  *      |         |
432  *      |         |
433  *      |         |
434  *      |         |
435  *      |         |
436  *      |         |
437  *      |         |
438  *      |         |
439  *      |         |
440  *      |         |
441  *      |         |
442  *      |         |
443  *      |         |
444  *      |         |
445  *      |         |
446  *      |         |
447  *      |         |
448  *      |         |
449  *      |         |
450  *      |         |
451  *      |         |
452  *      |         |
453  *      |         |
454  *      |         |
455  *      |         |
456  *      |         |
457  *      |         |
458  *      |         |
459  *      |         |
460  *      |         |
461  *      |         |
462  *      |         |
463  *      |         |
464  *      |         |
465  *      |         |
466  *      |         |
467  *      |         |
468  *      |         |
469  *      |         |
470  *      |         |
471  *      |         |
472  *      |         |
473  *      |         |
474  *      |         |
475  *      |         |
476  *      |         |
477  *      |         |
478  *      |         |
479  *      |         |
480  *      |         |
481  *      |         |
482  *      |         |
483  *      |         |
484  *      |         |
485  *      |         |
486  *      |         |
487  *      |         |
488  *      |         |
489  *      |         |
490  *      |         |
491  *      |         |
492  *      |         |
493  *      |         |
494  *      |         |
495  *      |         |
496  *      |         |
497  *      |         |
498  *      |         |
499  *      |         |
500  *      |         |
501  *      |         |
502  *      |         |
503  *      |         |
504  *      |         |
505  *      |         |
506  *      |         |
507  *      |         |
508  *      |         |
509  *      |         |
510  *      |         |
511  *      |         |
512  *      |         |
513  *      |         |
514  *      |         |
515  *      |         |
516  *      |         |
517  *      |         |
518  *      |         |
519  *      |         |
520  *      |         |
521  *      |         |
522  *      |         |
523  *      |         |
524  *      |         |
525  *      |         |
526  *      |         |
527  *      |         |
528  *      |         |
529  *      |         |
530  *      |         |
531  *      |         |
532  *      |         |
533  *      |         |
534  *      |         |
535  *      |         |
536  *      |         |
537  *      |         |
538  *      |         |
539  *      |         |
540  *      |         |
541  *      |         |
542  *      |         |
543  *      |         |
544  *      |         |
545  *      |         |
546  *      |         |
547  *      |         |
548  *      |         |
549  *      |         |
550  *      |         |
551  *      |         |
552  *      |         |
553  *      |         |
554  *      |         |
555  *      |         |
556  *      |         |
557  *      |         |
558  *      |         |
559  *      |         |
560  *      |         |
561  *      |         |
562  *      |         |
563  *      |         |
564  *      |         |
565  *      |         |
566  *      |         |
567  *      |         |
568  *      |         |
569  *      |         |
570  *      |         |
571  *      |         |
572  *      |         |
573  *      |         |
574  *      |         |
575  *      |         |
576  *      |         |
577  *      |         |
578  *      |         |
579  *      |         |
580  *      |         |
581  *      |         |
582  *      |         |
583  *      |         |
584  *      |         |
585  *      |         |
586  *      |         |
587  *      |         |
588  *      |         |
589  *      |         |
590  *      |         |
591  *      |         |
592  *      |         |
593  *      |         |
594  *      |         |
595  *      |         |
596  *      |         |
597  *      |         |
598  *      |         |
599  *      |         |
600  *      |         |
601  *      |         |
602  *      |         |
603  *      |         |
604  *      |         |
605  *      |         |
606  *      |         |
607  *      |         |
608  *      |         |
609  *      |         |
610  *      |         |
611  *      |         |
612  *      |         |
613  *      |         |
614  *      |         |
615  *      |         |
616  *      |         |
617  *      |         |
618  *      |         |
619  *      |         |
620  *      |         |
621  *      |         |
622  *      |         |
623  *      |         |
624  *      |         |
625  *      |         |
626  *      |         |
627  *      |         |
628  *      |         |
629  *      |         |
630  *      |         |
631  *      |         |
632  *      |         |
633  *      |         |
634  *      |         |
635  *      |         |
636  *      |         |
637  *      |         |
638  *      |         |
639  *      |         |
640  *      |         |
641  *      |         |
642  *      |         |
643  *      |         |
644  *      |         |
645  *      |         |
646  *      |         |
647  *      |         |
648  *      |         |
649  *      |         |
650  *      |         |
651  *      |         |
652  *      |         |
653  *      |         |
654  *      |         |
655  *      |         |
656  *      |         |
657  *      |         |
658  *      |         |
659  *      |         |
660  *      |         |
661  *      |         |
662  *      |         |
663  *      |         |
664  *      |         |
665  *      |         |
666  *      |         |
667  *      |         |
668  *      |         |
669  *      |         |
670  *      |         |
671  *      |         |
672  *      |         |
673  *      |         |
674  *      |         |
675  *      |         |
676  *      |         |
677  *      |         |
678  *      |         |
679  *      |         |
680  *      |         |
681  *      |         |
682  *      |         |
683  *      |         |
684  *      |         |
685  *      |         |
686  *      |         |
687  *      |         |
688  *      |         |
689  *      |         |
690  *      |         |
691  *      |         |
692  *      |         |
693  *      |         |
694  *      |         |
695  *      |         |
696  *      |         |
697  *      |         |
698  *      |         |
699  *      |         |
700  *      |         |
701  *      |         |
702  *      |         |
703  *      |         |
704  *      |         |
705  *      |         |
706  *      |         |
707  *      |         |
708  *      |         |
709  *      |         |
710  *      |         |
711  *      |         |
712  *      |         |
713  *      |         |
714  *      |         |
715  *      |         |
716  *      |         |
717  *      |         |
718  *      |         |
719  *      |         |
720  *      |         |
721  *      |         |
722  *      |         |
723  *      |         |
724  *      |         |
725  *      |         |
726  *      |         |
727  *      |         |
728  *      |         |
729  *      |         |
730  *      |         |
731  *      |         |
732  *      |         |
733  *      |         |
734  *      |         |
735  *      |         |
736  *      |         |
737  *      |         |
738  *      |         |
739  *      |         |
740  *      |         |
741  *      |         |
742  *      |         |
743  *      |         |
744  *      |         |
745  *      |         |
746  *      |         |
747  *      |         |
748  *      |         |
749  *      |         |
750  *      |         |
751  *      |         |
752  *      |         |
753  *      |         |
754  *      |         |
755  *      |         |
756  *      |         |
757  *      |         |
758  *      |         |
759  *      |         |
760  *      |         |
761  *      |         |
762  *      |         |
763  *      |         |
764  *      |         |
765  *      |         |
766  *      |         |
767  *      |         |
768  *      |         |
769  *      |         |
770  *      |         |
771  *      |         |
772  *      |         |
773  *      |         |
774  *      |         |
775  *      |         |
776  *      |         |
777  *      |         |
778  *      |         |
779  *      |         |
780  *      |         |
781  *      |         |
782  *      |         |
783  *      |         |
784  *      |         |
785  *      |         |
786  *      |         |
787  *      |         |
788  *      |         |
789  *      |         |
790  *      |         |
791  *      |         |
792  *      |         |
793  *      |         |
794  *      |         |
795  *      |         |
796  *      |         |
797  *      |         |
798  *      |         |
799  *      |         |
800  *      |         |
801  *      |         |
802  *      |         |
803  *      |         |
804  *      |         |
805  *      |         |
806  *      |         |
807  *      |         |
808  *      |         |
809  *      |         |
810  *      |         |
811  *      |         |
812  *      |         |
813  *      |         |
814  *      |         |
815  *      |         |
816  *      |         |
817  *      |         |
818  *      |         |
819  *      |         |
820  *      |         |
821  *      |         |
822  *      |         |
823  *      |         |
824  *      |         |
825  *      |         |
826  *      |         |
827  *      |         |
828  *      |         |
829  *      |         |
830  *      |         |
831  *      |         |
832  *      |         |
833  *      |         |
834  *      |         |
835  *      |         |
836  *      |         |
837  *      |         |
838  *      |         |
839  *      |         |
840  *      |         |
841  *      |         |
842  *      |         |
843  *      |         |
844  *      |         |
845  *      |         |
846  *      |         |
847  *      |         |
848  *      |         |
849  *      |         |
850  *      |         |
851  *      |         |
852  *      |         |
853  *      |         |
854  *      |         |
855  *      |         |
856  *      |         |
857  *      |         |
858  *      |         |
859  *      |         |
860  *      |         |
861  *      |         |
862  *      |         |
863  *      |         |
864  *      |         |
865  *      |         |
866  *      |         |
867  *      |         |
868  *      |         |
869  *      |         |
870  *      |         |
871  *      |         |
872  *      |         |
873  *      |         |
874  *      |         |
875  *      |         |
876  *      |         |
877  *      |         |
878  *      |         |
879  *      |         |
880  *      |         |
881  *      |         |
882  *      |         |
883  *      |         |
884  *      |         |
885  *      |         |
886  *      |         |
887  *      |         |
888  *      |         |
889  *      |         |
890  *      |         |
891  *      |         |
892  *      |         |
893  *      |         |
894  *      |         |
895  *      |         |
896  *      |         |
897  *      |         |
898  *      |         |
899  *      |         |
900  *      |         |
901  *      |         |
902  *      |         |
903  *      |         |
904  *      |         |
905  *      |         |
906  *      |         |
907  *      |         |
908  *      |         |
909  *      |         |
910  *      |         |
911  *      |         |
912  *      |         |
913  *      |         |
914  *      |         |
915  *      |         |
916  *      |         |
917  *      |         |
918  *      |         |
919  *      |         |
920  *      |         |
921  *      |         |
922  *      |         |
923  *      |         |
924  *      |         |
925  *      |         |
926  *      |         |
927  *      |         |
928  *      |         |
929  *      |         |
930  *      |         |
931  *      |         |
932  *      |         |
933  *      |         |
934  *      |         |
935  *      |         |
936  *      |         |
937  *      |         |
938  *      |         |
939  *      |         |
940  *      |         |
941  *      |         |
942  *      |         |
943  *      |         |
944  *      |         |
945  *      |         |
946  *      |         |
947  *      |         |
948  *      |         |
949  *      |         |
950  *      |         |
951  *      |         |
952  *      |         |
953  *      |         |
954  *      |         |
955  *      |         |
956  *      |         |
957  *      |         |
958  *      |         |
959  *      |         |
960  *      |         |
961  *      |         |
962  *      |         |
963  *      |         |
964  *      |         |
965  *      |         |
966  *      |         |
967  *      |         |
968  *      |         |
969  *      |         |
970  *      |         |
971  *      |         |
972  *      |         |
973  *      |         |
974  *      |         |
975  *      |         |
976  *      |         |
977  *      |         |
978  *      |         |
979  *      |         |
980  *      |         |
981  *      |         |
982  *      |         |
983  *      |         |
984  *      |         |
985  *      |         |
986  *      |         |
987  *      |         |
988  *      |         |
989  *      |         |
990  *      |         |
991  *      |         |
992  *      |         |
993  *      |         |
994  *      |         |
995  *      |         |
996  *      |         |
997  *      |         |
998  *      |         |
999  *      |         |
1000  *      |         |
1001  *      |         |
1002  *      |         |
1003  *      |         |
1004  *      |         |
1005  *      |         |
1006  *      |         |
1007  *      |         |
1008  *      |         |
1009  *      |         |
1010  *      |         |
1011  *      |         |
1012  *      |         |
1013  *      |         |
1014  *      |         |
1015  *      |         |
1016  *      |         |
1017  *      |         |
1018  *      |         |
1019  *      |         |
1020  *      |         |
1021  *      |         |
1022  *      |         |
1023  *      |         |
1024  *      |         |
1025  *      |         |
1026  *      |         |
1027  *      |         |
1028  *      |         |
1029  *      |         |
1030  *      |         |
1031  *      |         |
1032  *      |         |
1033  *      |         |
1034  *      |         |
1035  *      |         |
1036  *      |         |
1037  *      |         |
1038  *      |         |
1039  *      |         |
1040  *      |         |
1041  *      |         |
1042  *      |         |
1043  *      |         |
1044  *      |         |
1045  *      |         |
1046  *      |         |
1047  *      |         |
1048  *      |         |
1049  *      |         |
1050  *      |         |
1051  *      |         |
1052  *      |         |
1053  *      |         |
1054  *      |         |
1055  *      |         |
1056  *      |         |
1057  *      |         |
1058  *      |         |
1059  *      |         |
1060  *      |         |
1061  *      |         |
1062  *      |         |
1063  *      |         |
1064  *      |         |
1065  *      |         |
1066  *      |         |
1067  *      |         |
1068  *      |         |
1069  *      |         |
1070  *      |         |
1071  *      |         |
1072  *      |         |
1073  *      |         |
1074  *      |         |
1075  *      |         |
1076  *      |         |
1077  *      |         |
1078  *      |         |
1079  *      |         |
1080  *      |         |
1081  *      |         |
1082  *      |         |
1083  *      |         |
1084  *      |         |
1085  *      |         |
1086  *      |         |
1087  *      |         |
1088  *      |         |
1089  *      |         |
1090  *      |         |
1091  *      |         |
1092  *      |         |
1093  *      |         |
1094  *      |         |
1095  *      |         |
1096  *      |         |
1097  *      |         |
1098  *      |         |
1099  *      |         |
1100  *      |         |
1101  *      |         |
1102  *      |         |
1103  *      |         |
1104  *      |         |
1105  *      |         |
1106  *      |         |
1107  *      |         |
1108  *      |         |
1109  *      |         |
1110  *      |         |
1111  *      |         |
1112  *      |         |
1113  *      |         |
1114  *      |         |
1115  *      |         |
1116  *      |         |
1117  *      |         |
1118  *      |         |
1119  *      |         |
1120  *      |         |
1121  *      |         |
1122  *      |         |
1123  *      |         |
1124  *      |         |
1125  *      |         |
1126  *      |         |
1127  *      |         |
1128  *      |         |
1129  *      |         |
1130  *      |         |
1131  *      |         |
1132  *      |         |
1133  *      |         |
1134  *      |         |
1135  *      |         |
1136  *      |         |
1137  *      |         |
1138  *      |         |
1139  *      |         |
1140  *      |         |
1141  *      |         |
1142  *      |         |
1143  *      |         |
1144  *      |         |
1145  *      |         |
1146  *      |         |
1147  *      |         |
1148  *      |         |
1149  *      |         |
1150  *      |         |
1151  *      |         |
1152  *      |         |
1153  *      |         |
1154  *      |         |
1155  *      |         |
1156  *      |         |
1157  *      |         |
1158  *      |         |
1159  *      |         |
1160  *      |         |
1161  *      |         |
1162  *      |         |
1163  *      |         |
1164  *      |         |
1165  *      |         |
1166  *      |         |
1167  *      |         |
1168  *      |         |
1169  *      |         |
1170  *      |         |
1171  *      |         |
1172  *      |         |
1173  *      |         |
1174  *      |         |
1175  *      |         |
1176  *      |         |
1177  *      |         |
1178  *      |         |
1179  *      |         |
1180  *      |         |
1181  *      |         |
1182  *      |         |
1183  *      |         |
1184  *      |         |
1185  *      |         |
1186  *      |         |
1187  *      |         |
1188  *      |         |
1189  *      |         |
1190  *      |         |
1191  *      |         |
1192  *      |         |
1193  *      |         |
1194  *      |         |
1195  *      |         |
1196  *      |         |
1197  *      |         |
1198  *      |         |
1199  *      |         |
1200  *      |         |
1201  *      |         |
1202  *      |         |
1203  *      |         |
1204  *      |         |
1205  *      |         |
1206  *      |         |
1207  *      |         |
1208  *      |         |
1209  *      |         |
1210  *      |         |
1211  *      |         |
1212  *      |         |
1213  *      |         |
1214  *      |         |
1215  *      |         |
1216  *      |         |
1217  *      |         |
1218  *      |         |
1219  *      |         |
1220  *      |         |
1221  *      |         |
1222  *      |         |
1223  *      |         |
1224  *      |         |
1225  *      |         |
1226  *      |         |
1227  *      |         |
1228  *      |         |
1229  *      |         |
1230  *      |         |
1231  *      |         |
1232  *      |         |
1233  *      |         |
1234  *      |         |
1235  *      |         |
1236  *      |         |
1237  *      |         |
1238  *      |         |
1239  *      |         |
1240  *      |         |
1241  *      |         |
1242  *      |         |
1243  *      |         |
1244  *      |         |
1245  *      |         |
1246  *      |         |
1247  *      |         |
1248  *      |         |
1249  *      |         |
1250  *      |         |
1251  *      |         |
1252  *      |         |
1253  *      |         |
1254  *      |         |
1255  *      |         |
1256  *      |         |
1257  *      |         |
1258  *      |         |
1259  *      |         |
1260  *      |         |
1261  *      |         |
1262  *      |         |
1263  *      |         |
1264  *      |         |
1265  *      |         |
1266  *      |         |
1267  *      |         |
1268  *      |         |
1269  *      |         |
1270  *      |        
```



```

273  */
274
275  fence                                /* synchronize memory          */
276
277  /*
278  * Recover the CePUs state.
279  */
280
281  lw   ra,    0*4(sp)                  /* load return address          */
282  lw   s0,    1*4(sp)                  /* load callee saved register 0 */
283  lw   s1,    2*4(sp)                  /* load callee saved register 1 */
284  lw   s2,    3*4(sp)                  /* load callee saved register 2 */
285  lw   s3,    4*4(sp)                  /* load callee saved register 3 */
286  lw   s4,    5*4(sp)                  /* load callee saved register 4 */
287  lw   s5,    6*4(sp)                  /* load callee saved register 5 */
288  lw   s6,    7*4(sp)                  /* load callee saved register 6 */
289  lw   s7,    8*4(sp)                  /* load callee saved register 7 */
290  lw   s8,    9*4(sp)                  /* load callee saved register 8 */
291  lw   s9,   10*4(sp)                  /* load callee saved register 9 */
292  lw   s10,  11*4(sp)                  /* load callee saved register 10 */
293  lw   s11,  12*4(sp)                  /* load callee saved register 11 */
294
295  addi sp,   sp,   52                  /* set back stack pointer      */
296
297  /*
298  * Returning. Note that we have changed ra to somewhere in
299  * pn_begin_threaded[_m|_gm].
300  */
301
302  ret                                /* return                      */
303
304 #endif /* !DOXYGEN */
305
306 /*EOF******/

```


Index

- [__pn_spinlock](#), [85](#)
 - [owner_ID](#), [86](#)
- [_pn_spinlock](#)
 - [Typedefs](#), [48](#)
- [ALL_HALTED](#)
 - [Commonly Used Defines](#), [17](#)
- [BEGIN_THREADED_LINKED_SEC_CHECK_M](#)
 - [Commonly Used Defines](#), [18](#)
- [BEGIN_THREADED_LINKED_SEC_CHECK](#)
 - [Commonly Used Defines](#), [17](#)
- [BULK_LINES](#)
 - [pn_cache.c](#), [108](#)
- [Base Module](#), [55](#)
 - [pn_coreid](#), [56](#)
 - [pn_coreid_g](#), [56](#)
 - [pn_halt](#), [57](#)
 - [pn_halt_CoPU_gm](#), [58](#)
 - [pn_halt_CoPU_m](#), [58](#)
 - [pn_halt_CoPU](#), [57](#)
 - [pn_m2cap](#), [59](#)
 - [pn_m2cap_g](#), [59](#)
 - [pn_m3cap](#), [60](#)
 - [pn_m3cap_g](#), [60](#)
 - [pn_numcores](#), [61](#)
 - [pn_simulation](#), [61](#)
 - [pn_time_ns](#), [61](#)
- [CINV](#)
 - [ParaNut Custom Instructions](#), [24](#)
- [CONVERT_NUMC_TO_MASK](#)
 - [Commonly Used Defines](#), [18](#)
- [COPU_CHECK](#)
 - [Commonly Used Defines](#), [18](#)
- [CWB](#)
 - [ParaNut Custom Instructions](#), [24](#)
- [Cache Module](#), [71](#)
 - [pn_cache_disable](#), [72](#)
 - [pn_cache_enable](#), [72](#)
 - [pn_cache_flush](#), [72](#)
 - [pn_cache_flush_all](#), [73](#)
 - [pn_cache_init](#), [73](#)
 - [pn_cache_invalidate](#), [73](#)
 - [pn_cache_invalidate_all](#), [74](#)
 - [pn_cache_linesize](#), [74](#)
 - [pn_cache_size](#), [74](#)
 - [pn_cache_writeback](#), [74](#)
 - [pn_cache_writeback_all](#), [75](#)
- [cache_banks_as](#)
 - [Internal Assembly Calls](#), [29](#)
- [cache_sets_as](#)
 - [Internal Assembly Calls](#), [29](#)
- [common/common.h](#), [87](#)
- [common/common_RV32I.S](#), [90](#)
- [common/custom_RV32I.S](#), [92](#)
- [Commonly Used Defines](#), [17](#)
 - [ALL_HALTED](#), [17](#)
 - [BEGIN_THREADED_LINKED_SEC_CHECK_M](#), [18](#)
 - [BEGIN_THREADED_LINKED_SEC_CHECK](#), [17](#)
 - [CONVERT_NUMC_TO_MASK](#), [18](#)
 - [COPU_CHECK](#), [18](#)
 - [TERMNL](#), [18](#)
- [coreid_as](#)
 - [Internal Assembly Calls](#), [30](#)
- [destroy_as](#)
 - [Internal Assembly Calls](#), [30](#)
- [disable_cache_as](#)
 - [Internal Assembly Calls](#), [30](#)
- [ecall_as](#)
 - [Internal Assembly Calls](#), [30](#)
- [enable_CPU_as](#)
 - [Internal Assembly Calls](#), [31](#)
- [enable_cache_as](#)
 - [Internal Assembly Calls](#), [30](#)
- [enter_threaded_mode_as](#)
 - [Internal Assembly Calls](#), [31](#)
- [Error Codes](#), [50](#)
 - [PN_ERR_CACHE_LINESIZE](#), [50](#)
 - [PN_ERR_COPU](#), [51](#)
 - [PN_ERR_EXC](#), [51](#)
 - [PN_ERR_LOCKOCC](#), [51](#)
 - [PN_ERR_MATCH](#), [52](#)
 - [PN_ERR_NOIMP](#), [52](#)
 - [PN_ERR_PARAM](#), [52](#)
 - [PN_SUCCESS](#), [52](#)
- [Exception Module](#), [76](#)
 - [pn_ecall](#), [76](#)
 - [pn_exception_set_handler](#), [77](#)
 - [pn_interrupt_disable](#), [77](#)
 - [pn_interrupt_enable](#), [77](#)
 - [pn_progress_mepc](#), [78](#)
- [exception_entry_as](#)
 - [Internal Assembly Calls](#), [31](#)
- [exception_init_as](#)
 - [Internal Assembly Calls](#), [31](#)

- flush_1024_as
 - Internal Assembly Calls, [31](#)
- flush_128_as
 - Internal Assembly Calls, [32](#)
- flush_2048_as
 - Internal Assembly Calls, [32](#)
- flush_256_as
 - Internal Assembly Calls, [32](#)
- flush_32_as
 - Internal Assembly Calls, [32](#)
- flush_512_as
 - Internal Assembly Calls, [32](#)
- flush_64_as
 - Internal Assembly Calls, [33](#)
- flush_bulk_1024_as
 - Internal Assembly Calls, [33](#)
- flush_bulk_128_as
 - Internal Assembly Calls, [33](#)
- flush_bulk_2048_as
 - Internal Assembly Calls, [33](#)
- flush_bulk_256_as
 - Internal Assembly Calls, [33](#)
- flush_bulk_32_as
 - Internal Assembly Calls, [34](#)
- flush_bulk_512_as
 - Internal Assembly Calls, [34](#)
- flush_bulk_64_as
 - Internal Assembly Calls, [34](#)
- freq_as
 - Internal Assembly Calls, [34](#)
- Global Variables, [19](#)
 - sp_loc, [19](#)
 - tp_loc, [19](#)
- HALT
 - ParaNut Custom Instructions, [25](#)
- halt_as
 - Internal Assembly Calls, [34](#)
- init_as
 - Internal Assembly Calls, [35](#)
- Internal Assembly Calls, [26](#)
 - cache_banks_as, [29](#)
 - cache_sets_as, [29](#)
 - coreid_as, [30](#)
 - destroy_as, [30](#)
 - disable_cache_as, [30](#)
 - ecall_as, [30](#)
 - enable_CPU_as, [31](#)
 - enable_cache_as, [30](#)
 - enter_threaded_mode_as, [31](#)
 - exception_entry_as, [31](#)
 - exception_init_as, [31](#)
 - flush_1024_as, [31](#)
 - flush_128_as, [32](#)
 - flush_2048_as, [32](#)
 - flush_256_as, [32](#)
 - flush_32_as, [32](#)
 - flush_512_as, [32](#)
 - flush_64_as, [33](#)
 - flush_bulk_1024_as, [33](#)
 - flush_bulk_128_as, [33](#)
 - flush_bulk_2048_as, [33](#)
 - flush_bulk_256_as, [33](#)
 - flush_bulk_32_as, [34](#)
 - flush_bulk_512_as, [34](#)
 - flush_bulk_64_as, [34](#)
 - freq_as, [34](#)
 - halt_as, [34](#)
 - init_as, [35](#)
 - invalidate_1024_as, [35](#)
 - invalidate_128_as, [35](#)
 - invalidate_2048_as, [35](#)
 - invalidate_256_as, [35](#)
 - invalidate_32_as, [36](#)
 - invalidate_512_as, [36](#)
 - invalidate_64_as, [36](#)
 - invalidate_bulk_1024_as, [36](#)
 - invalidate_bulk_128_as, [36](#)
 - invalidate_bulk_2048_as, [37](#)
 - invalidate_bulk_256_as, [37](#)
 - invalidate_bulk_32_as, [37](#)
 - invalidate_bulk_512_as, [37](#)
 - invalidate_bulk_64_as, [37](#)
 - m2cap_as, [38](#)
 - mem_size_as, [38](#)
 - numcores_as, [38](#)
 - progress_mepc_as, [38](#)
 - read_MTVAl_as, [39](#)
 - read_PNCAUSE_as, [39](#)
 - read_PNCE_as, [39](#)
 - read_PNEPC_as, [39](#)
 - read_PNLM_as, [40](#)
 - read_PNX_as, [40](#)
 - set_linked_as, [40](#)
 - set_threaded_as, [40](#)
 - simulation_as, [41](#)
 - ticks_as, [41](#)
 - trylock_as, [41](#)
 - unlock_as, [41](#)
 - write_MSTATUS_as, [42](#)
 - write_PNLM_as, [43](#)
 - write_PNXSEL_as, [43](#)
 - writeback_1024_as, [43](#)
 - writeback_128_as, [43](#)
 - writeback_2048_as, [43](#)
 - writeback_256_as, [44](#)
 - writeback_32_as, [44](#)
 - writeback_512_as, [44](#)
 - writeback_64_as, [44](#)
 - writeback_bulk_1024_as, [44](#)
 - writeback_bulk_128_as, [45](#)
 - writeback_bulk_2048_as, [45](#)
 - writeback_bulk_256_as, [45](#)
 - writeback_bulk_32_as, [45](#)
 - writeback_bulk_512_as, [45](#)

- writeback_bulk_64_as, [46](#)
- invalidate_1024_as
 - Internal Assembly Calls, [35](#)
- invalidate_128_as
 - Internal Assembly Calls, [35](#)
- invalidate_2048_as
 - Internal Assembly Calls, [35](#)
- invalidate_256_as
 - Internal Assembly Calls, [35](#)
- invalidate_32_as
 - Internal Assembly Calls, [36](#)
- invalidate_512_as
 - Internal Assembly Calls, [36](#)
- invalidate_64_as
 - Internal Assembly Calls, [36](#)
- invalidate_bulk_1024_as
 - Internal Assembly Calls, [36](#)
- invalidate_bulk_128_as
 - Internal Assembly Calls, [36](#)
- invalidate_bulk_2048_as
 - Internal Assembly Calls, [37](#)
- invalidate_bulk_256_as
 - Internal Assembly Calls, [37](#)
- invalidate_bulk_32_as
 - Internal Assembly Calls, [37](#)
- invalidate_bulk_512_as
 - Internal Assembly Calls, [37](#)
- invalidate_bulk_64_as
 - Internal Assembly Calls, [37](#)
- libparanut Compile Time Parameters, [83](#)
 - PN_CACHE_LINESIZE, [83](#)
 - PN_COMPILE_RAW, [83](#)
 - PN_RWIDTH, [84](#)
- libparanut Helpers, [47](#)
- libparanut Modules, [54](#)
- libparanut.h, [94](#)
- Link Module, [63](#)
 - pn_begin_linked, [63](#)
 - pn_begin_linked_gm, [64](#)
 - pn_begin_linked_m, [65](#)
 - pn_end_linked, [66](#)
- m2cap_as
 - Internal Assembly Calls, [38](#)
- Makefile, [98](#)
- mem_size_as
 - Internal Assembly Calls, [38](#)
- Modes, [53](#)
- mtval
 - ParaNut Control and Status Registers, [20](#)
- numcores_as
 - Internal Assembly Calls, [38](#)
- owner_ID
 - __pn_spinlock, [86](#)
- PN_CACHE_LINESIZE
 - libparanut Compile Time Parameters, [83](#)
- PN_CID
 - Typedefs, [48](#)
- PN_CMSK
 - Typedefs, [49](#)
- PN_COMPILE_RAW
 - libparanut Compile Time Parameters, [83](#)
- PN_ERR_CACHE_LINESIZE
 - Error Codes, [50](#)
- PN_ERR_COPU
 - Error Codes, [51](#)
- PN_ERR_EXC
 - Error Codes, [51](#)
- PN_ERR_LOCKOCC
 - Error Codes, [51](#)
- PN_ERR_MATCH
 - Error Codes, [52](#)
- PN_ERR_NOIMP
 - Error Codes, [52](#)
- PN_ERR_PARAM
 - Error Codes, [52](#)
- PN_NUMC
 - Typedefs, [49](#)
- PN_NUMG
 - Typedefs, [49](#)
- PN_RWIDTH
 - libparanut Compile Time Parameters, [84](#)
- PN_SUCCESS
 - Error Codes, [52](#)
- ParaNut Control and Status Registers, [20](#)
 - mtval, [20](#)
 - pncache, [21](#)
 - pncacheinfo, [21](#)
 - pncachesets, [21](#)
 - pncause, [21](#)
 - pnce, [21](#)
 - pnclockinfo, [21](#)
 - pncpus, [22](#)
 - pnepc, [22](#)
 - pngrpsel, [22](#)
 - pnlm, [22](#)
 - pnm2cap, [22](#)
 - pnmemsize, [22](#)
 - pnx, [23](#)
 - pnxsel, [23](#)
- ParaNut Custom Instructions, [24](#)
 - CINV, [24](#)
 - CWB, [24](#)
 - HALT, [25](#)
- pn_base/pn_base.c, [101](#)
- pn_base/pn_base_RV32I.S, [103](#)
- pn_begin_linked
 - Link Module, [63](#)
- pn_begin_linked_gm
 - Link Module, [64](#)
- pn_begin_linked_m
 - Link Module, [65](#)
- pn_begin_threaded

- Thread Module, [67](#)
- pn_begin_threaded_gm
 - Thread Module, [68](#)
- pn_begin_threaded_m
 - Thread Module, [69](#)
- pn_cache.c
 - BULK_LINES, [108](#)
 - sec_check, [108](#)
- pn_cache/pn_cache.c, [104](#)
- pn_cache/pn_cache_RV32I_1024.S, [109](#)
- pn_cache/pn_cache_RV32I_128.S, [113](#)
- pn_cache/pn_cache_RV32I_2048.S, [117](#)
- pn_cache/pn_cache_RV32I_256.S, [122](#)
- pn_cache/pn_cache_RV32I_32.S, [126](#)
- pn_cache/pn_cache_RV32I_512.S, [130](#)
- pn_cache/pn_cache_RV32I_64.S, [135](#)
- pn_cache/pn_cache_RV32I_auto.S, [139](#)
- pn_cache/pn_cache_RV32I_buildscript.py, [158](#)
- pn_cache_disable
 - Cache Module, [72](#)
- pn_cache_enable
 - Cache Module, [72](#)
- pn_cache_flush
 - Cache Module, [72](#)
- pn_cache_flush_all
 - Cache Module, [73](#)
- pn_cache_init
 - Cache Module, [73](#)
- pn_cache_invalidate
 - Cache Module, [73](#)
- pn_cache_invalidate_all
 - Cache Module, [74](#)
- pn_cache_linesize
 - Cache Module, [74](#)
- pn_cache_size
 - Cache Module, [74](#)
- pn_cache_writeback
 - Cache Module, [74](#)
- pn_cache_writeback_all
 - Cache Module, [75](#)
- pn_config.h, [166](#)
- pn_coreid
 - Base Module, [56](#)
- pn_coreid_g
 - Base Module, [56](#)
- pn_ecall
 - Exception Module, [76](#)
- pn_end_linked
 - Link Module, [66](#)
- pn_end_threaded
 - Thread Module, [70](#)
- pn_exception/pn_exception.c, [167](#)
- pn_exception/pn_exception_RV32I.S, [169](#)
- pn_exception_set_handler
 - Exception Module, [77](#)
- pn_halt
 - Base Module, [57](#)
- pn_halt_CoPU_gm
 - Base Module, [58](#)
- pn_halt_CoPU_m
 - Base Module, [58](#)
- pn_halt_CoPU
 - Base Module, [57](#)
- pn_interrupt_disable
 - Exception Module, [77](#)
- pn_interrupt_enable
 - Exception Module, [77](#)
- pn_link/pn_link.c, [173](#)
- pn_link/pn_link_RV32I.S, [174](#)
- pn_m2cap
 - Base Module, [59](#)
- pn_m2cap_g
 - Base Module, [59](#)
- pn_m3cap
 - Base Module, [60](#)
- pn_m3cap_g
 - Base Module, [60](#)
- pn_numcores
 - Base Module, [61](#)
- pn_progress_mepc
 - Exception Module, [78](#)
- pn_simulation
 - Base Module, [61](#)
- pn_spinlock/pn_spinlock.c, [177](#)
- pn_spinlock/pn_spinlock_RV32I.S, [178](#)
- pn_spinlock_destroy
 - Spinlock Module, [80](#)
- pn_spinlock_init
 - Spinlock Module, [80](#)
- pn_spinlock_lock
 - Spinlock Module, [81](#)
- pn_spinlock_trylock
 - Spinlock Module, [81](#)
- pn_spinlock_unlock
 - Spinlock Module, [82](#)
- pn_thread/pn_thread.c, [180](#)
- pn_thread/pn_thread_RV32I.S, [182](#)
- pn_thread_entry
 - Thread Module, [70](#)
- pn_time_ns
 - Base Module, [61](#)
- pncache
 - ParaNut Control and Status Registers, [21](#)
- pncacheinfo
 - ParaNut Control and Status Registers, [21](#)
- pncachesets
 - ParaNut Control and Status Registers, [21](#)
- pncause
 - ParaNut Control and Status Registers, [21](#)
- pnce
 - ParaNut Control and Status Registers, [21](#)
- pnclockinfo
 - ParaNut Control and Status Registers, [21](#)
- pncpus
 - ParaNut Control and Status Registers, [22](#)
- pnepc

- ParaNut Control and Status Registers, [22](#)
- pngrpssel
 - ParaNut Control and Status Registers, [22](#)
- pnlm
 - ParaNut Control and Status Registers, [22](#)
- pnm2cap
 - ParaNut Control and Status Registers, [22](#)
- pnmemsize
 - ParaNut Control and Status Registers, [22](#)
- pnx
 - ParaNut Control and Status Registers, [23](#)
- pnxsell
 - ParaNut Control and Status Registers, [23](#)
- progress_mepc_as
 - Internal Assembly Calls, [38](#)
- read_MTVAl_as
 - Internal Assembly Calls, [39](#)
- read_PNCAUSE_as
 - Internal Assembly Calls, [39](#)
- read_PNCE_as
 - Internal Assembly Calls, [39](#)
- read_PNEPC_as
 - Internal Assembly Calls, [39](#)
- read_PNLM_as
 - Internal Assembly Calls, [40](#)
- read_PNX_as
 - Internal Assembly Calls, [40](#)
- sec_check
 - pn_cache.c, [108](#)
- set_linked_as
 - Internal Assembly Calls, [40](#)
- set_threaded_as
 - Internal Assembly Calls, [40](#)
- simulation_as
 - Internal Assembly Calls, [41](#)
- sp_loc
 - Global Variables, [19](#)
- Spinlock Module, [79](#)
 - pn_spinlock_destroy, [80](#)
 - pn_spinlock_init, [80](#)
 - pn_spinlock_lock, [81](#)
 - pn_spinlock_trylock, [81](#)
 - pn_spinlock_unlock, [82](#)
- TERMNL
 - Commonly Used Defines, [18](#)
- Thread Module, [67](#)
 - pn_begin_threaded, [67](#)
 - pn_begin_threaded_gm, [68](#)
 - pn_begin_threaded_m, [69](#)
 - pn_end_threaded, [70](#)
 - pn_thread_entry, [70](#)
- ticks_as
 - Internal Assembly Calls, [41](#)
- tp_loc
 - Global Variables, [19](#)
- trylock_as
 - Internal Assembly Calls, [41](#)
- Typedefs, [48](#)
 - _pn_spinlock, [48](#)
 - PN_CID, [48](#)
 - PN_CMSK, [49](#)
 - PN_NUMC, [49](#)
 - PN_NUMG, [49](#)
- unlock_as
 - Internal Assembly Calls, [41](#)
- write_MSTATUS_as
 - Internal Assembly Calls, [42](#)
- write_PNLM_as
 - Internal Assembly Calls, [43](#)
- write_PNXSEL_as
 - Internal Assembly Calls, [43](#)
- writeback_1024_as
 - Internal Assembly Calls, [43](#)
- writeback_128_as
 - Internal Assembly Calls, [43](#)
- writeback_2048_as
 - Internal Assembly Calls, [43](#)
- writeback_256_as
 - Internal Assembly Calls, [44](#)
- writeback_32_as
 - Internal Assembly Calls, [44](#)
- writeback_512_as
 - Internal Assembly Calls, [44](#)
- writeback_64_as
 - Internal Assembly Calls, [44](#)
- writeback_bulk_1024_as
 - Internal Assembly Calls, [44](#)
- writeback_bulk_128_as
 - Internal Assembly Calls, [45](#)
- writeback_bulk_2048_as
 - Internal Assembly Calls, [45](#)
- writeback_bulk_256_as
 - Internal Assembly Calls, [45](#)
- writeback_bulk_32_as
 - Internal Assembly Calls, [45](#)
- writeback_bulk_512_as
 - Internal Assembly Calls, [45](#)
- writeback_bulk_64_as
 - Internal Assembly Calls, [46](#)