## OR1K support

Generated by Doxygen 1.8.6

Tue Nov 18 2014 11:03:05

# **Contents**

1	Mod	lule Inde	ex	1
	1.1	Module	es	1
2	Mod	ule Doc	cumentation	3
	2.1	OR1K	macros	3
		2.1.1	Detailed Description	3
		2.1.2	Macro Definition Documentation	3
			2.1.2.1 REG16	3
			2.1.2.2 REG32	3
			2.1.2.3 REG8	3
	2.2	OR1K	interrupt control	5
		2.2.1	Detailed Description	5
		2.2.2	Typedef Documentation	5
			2.2.2.1 or1k_interrupt_handler_fptr	5
		2.2.3	Function Documentation	5
			2.2.3.1 or1k_interrupt_disable	5
			2.2.3.2 or1k_interrupt_enable	5
			2.2.3.3 or1k_interrupt_handler_add	5
			2.2.3.4 or1k_interrupts_disable	6
			2.2.3.5 or1k_interrupts_enable	6
			2.2.3.6 or1k_interrupts_restore	6
	2.3	Except	tion handling	7
		2.3.1	Detailed Description	7
		2.3.2	Typedef Documentation	7
			2.3.2.1 or1k_exception_handler_fptr	7
		2.3.3	Function Documentation	7
			2.3.3.1 or1k_exception_handler_add	7
	2.4	SPR a	ccess	8
	2.5	Miscel	laneous utility functions	9
		2.5.1	Detailed Description	9
		2.5.2	Function Decumentation	

iv CONTENTS

		2.5.2.1	or1k_rand	9
		2.5.2.2	or1k_report	9
2.6	Cache	control .		10
	2.6.1	Detailed	Description	10
	2.6.2	Function	Documentation	10
		2.6.2.1	or1k_dcache_disable	10
		2.6.2.2	or1k_dcache_enable	10
		2.6.2.3	or1k_dcache_flush	10
		2.6.2.4	or1k_icache_disable	10
		2.6.2.5	or1k_icache_enable	10
		2.6.2.6	or1k_icache_flush	10
2.7	MMU d	control		11
	2.7.1	Detailed	Description	11
	2.7.2	Function	Documentation	11
		2.7.2.1	or1k_dmmu_disable	11
		2.7.2.2	or1k_dmmu_enable	11
		2.7.2.3	or1k_immu_disable	11
		2.7.2.4	or1k_immu_enable	11
2.8	Timer	control .		12
	2.8.1	Detailed	Description	12
	2.8.2	Function	Documentation	13
		2.8.2.1	or1k_timer_disable	13
		2.8.2.2	or1k_timer_enable	13
		2.8.2.3	or1k_timer_get_ticks	13
		2.8.2.4	or1k_timer_init	13
		2.8.2.5	or1k_timer_pause	14
		2.8.2.6	or1k_timer_reset	14
		2.8.2.7	or1k_timer_reset_ticks	14
		2.8.2.8	or1k_timer_restore	14
		2.8.2.9	or1k_timer_set_handler	14
		2.8.2.10	or1k_timer_set_mode	14
		2.8.2.11	or1k_timer_set_period	15
2.9	Multico	ore and Sy	nchronization Support	16
	2.9.1	Detailed	Description	16
	2.9.2	Function	Documentation	16
		2.9.2.1	or1k_coreid	16
		2.9.2.2	or1k_numcores	16
		2.9.2.3	or1k_sync_cas	16
		2.9.2.4	or1k_sync_ll	16
		2.9.2.5	or1k_sync_sc	17

CONTENTS					<u>v</u>
	2.9.2.6	or1k_sync_tsl	 	 	 . 17
Index					18

# Chapter 1

# **Module Index**

## 1.1 Modules

Here is	a list of a	all modules:	
	u		

R1K macros	3
R1K interrupt control	5
ception handling	7
R access	8
scellaneous utility functions	9
che control	10
MU control	11
ner control	12
Multicore and Synchronization Support	16

2 **Module Index** 

## **Chapter 2**

## **Module Documentation**

## 2.1 OR1K macros

## **Macros**

- #define REG8(add) \*((volatile unsigned char \*) (add))
- #define REG16(add) \*((volatile unsigned short \*) (add))
- #define REG32(add) \*((volatile unsigned long \*) (add))

## 2.1.1 Detailed Description

## 2.1.2 Macro Definition Documentation

2.1.2.1 #define REG16( add ) \*((volatile unsigned short \*) (add))

Access halfword-sized memory mapped register

Used to access a 16 byte-sized memory mapped register. It avoids usage errors when not defining register addresses volatile and handles casting correctly.

See REG8() for an example.

## **Parameters**

add	Register address
-----	------------------

2.1.2.2 #define REG32( add ) \*((volatile unsigned long \*) (add))

Access word-sized memory mapped register

Used to access a word-sized memory mapped register. It avoids usage errors when not defining register addresses volatile and handles casting correctly.

See REG8() for an example.

## **Parameters**

add	Register address

2.1.2.3 #define REG8( add ) \*((volatile unsigned char \*) (add))

Access byte-sized memory mapped register

Used to access a byte-sized memory mapped register. It avoids usage errors when not defining register addresses volatile and handles casting correctly.

Example for both read and write:

```
uint8_t status = REG8(IPBLOCK_STATUS_REG_ADDR);
REG8(IPBLOCK_ENABLE) = 1;
```

## **Parameters**

add	Register address

## 2.2 OR1K interrupt control

## **Typedefs**

typedef void(\* or1k\_interrupt\_handler\_fptr )(uint32\_t data)

## **Functions**

- void or1k\_interrupt\_handler\_add (int line, or1k\_interrupt\_handler\_fptr handler, uint32\_t data)
- void or1k\_interrupt\_enable (int line)
- void or1k interrupt disable (int line)
- uint32\_t or1k\_interrupts\_disable (void)
- void or1k\_interrupts\_enable (void)
- void or1k\_interrupts\_restore (uint32\_t status)

## 2.2.1 Detailed Description

Interrupt control function prototypes

## 2.2.2 Typedef Documentation

2.2.2.1 typedef void(\* or1k\_interrupt\_handler\_fptr)(uint32\_t data)

Function pointer to interrupt handler functions

## 2.2.3 Function Documentation

2.2.3.1 void or1k\_interrupt\_disable (int line)

Disable interrupts from a given line

Mask given interrupt line. It can be unmasked using or1k\_interrupt\_enable().

**Parameters** 

line Interrupt line to disable

## 2.2.3.2 void or1k\_interrupt\_enable ( int line )

Enable interrupts from a given line

Unmask the given interrupt line. It is also important to enable interrupts in general, e.g., using or1k\_interrupts\_enable().

**Parameters** 

line Interrupt line to enable

2.2.3.3 void or1k\_interrupt\_handler\_add (int line, or1k\_interrupt\_handler\_fptr handler, uint32\_t data)

Add interrupt handler for interrupt line

Registers a callback function for a certain interrupt line.

#### **Parameters**

	line	Interrupt line/id to register a handler for		
	handler	handler Handler to register		
data Data value passed to the handler				

#### 2.2.3.4 uint32\_t or1k\_interrupts\_disable (void)

#### Disable interrupts

This disables the interrupt exception. This is sufficient to disable all interrupts. It does not change the mask register (which is modified using or1k interrupt enable() and or1k interrupt disable()).

The interrupt exception can be enabled using or1k interrupts enable().

Finally, the status of the interrupt exception enable flag is returned by this function. That allows to call this function even if interrupts are already disabled. To restore the value of the interrupt exception enable flag, use the or1k\_interrupts restore() function. That way you avoid to accidentally enable interrupts. Example:

```
void f() {
uint32_t interrupt_status = orlk_interrupts_disable();
// do something
orlk_interrupts_restore(status);
}
```

This code will preserve the original status of the interrupt enable flag.

#### Returns

Interrupt exception enable flag before call

## 2.2.3.5 void or1k\_interrupts\_enable (void)

## Enable interrupt exception

Enable the interrupt exception. Beside the interrupt exception, it is also necessary to enable the individual interrupt lines using or1k\_interrupt\_enable().

You should avoid using this function together with or1k\_interrupts\_disable() to guard atomic blocks as it unconditionally enables the interrupt exception (see documentation of or1k\_interrupts\_disable()).

```
2.2.3.6 void or1k_interrupts_restore ( uint32_t status )
```

Restore interrupt exception enable flag

This function restores the given status to the processor. or1k\_interrupts\_restore(0) is identical to or1k\_interrupts\_disable() and or1k\_interrupts\_restore(SPR\_SR\_IEE) is identical to or1k\_interrupts\_enable().

It is for example used to guard an atomic block and restore the original status of the interrupt exception enable flag as returned by or1k interrupts disable(). See the documentation of or1k interrupts disable() for a usage example.

#### **Parameters**

status	Status of the flag to restore

## 2.3 Exception handling

## **Typedefs**

typedef void(\* or1k\_exception\_handler\_fptr )(void)

## **Functions**

• void or1k\_exception\_handler\_add (int id, or1k\_exception\_handler\_fptr handler)

## 2.3.1 Detailed Description

## 2.3.2 Typedef Documentation

2.3.2.1 typedef void(\* or1k\_exception\_handler\_fptr)(void)

Function pointer to an exception handler function

## 2.3.3 Function Documentation

2.3.3.1 void or1k\_exception\_handler\_add ( int id, or1k\_exception\_handler\_fptr handler )

Register exception handler

Register an exception handler for the given exception id. This handler is in the following then called when the exception occurs. You can thereby individually handle those exceptions.

## **Parameters**

id	Exception id
handler	Handler callback

## 2.4 SPR access

## 2.5 Miscellaneous utility functions

## **Functions**

- void or1k\_report (unsigned long int value)
- unsigned long int or1k\_rand (void)

## 2.5.1 Detailed Description

## 2.5.2 Function Documentation

2.5.2.1 unsigned long int or1k\_rand (void)

Get (pseudo) random number

This should return pseudo-random numbers, based on a Galois LFSR.

Returns

(Pseudo) Random number

2.5.2.2 void or1k\_report ( unsigned long int value )

Report value to simulator

Uses the built-in simulator functionality.

**Parameters** 

value Value to report

## 2.6 Cache control

## **Functions**

- void or1k\_icache\_enable (void)
- void or1k\_icache\_disable (void)
- void or1k\_icache\_flush (uint32\_t entry)
- void or1k\_dcache\_enable (void)
- void or1k\_dcache\_disable (void)
- void or1k\_dcache\_flush (unsigned long entry)

## 2.6.1 Detailed Description

## 2.6.2 Function Documentation

2.6.2.1 void or1k\_dcache\_disable (void)

Disable data cache

2.6.2.2 void or1k\_dcache\_enable (void)

Enable data cache

2.6.2.3 void or1k\_dcache\_flush ( unsigned long entry )

Flush data cache

Invalidate data cache entry

**Parameters** 

entry | Entry to invalidate

2.6.2.4 void or1k\_icache\_disable (void)

Disable instruction cache

2.6.2.5 void or1k\_icache\_enable (void)

Enable instruction cache

2.6.2.6 void or1k\_icache\_flush ( uint32\_t entry )

Flush instruction cache

Invalidate instruction cache entry

**Parameters** 

entry | Entry to invalidate

2.7 MMU control

## 2.7 MMU control

## **Functions**

- void or1k\_immu\_enable (void)
- void or1k\_immu\_disable (void)
- void or1k\_dmmu\_enable (void)
- void or1k\_dmmu\_disable (void)

## 2.7.1 Detailed Description

## 2.7.2 Function Documentation

2.7.2.1 void or1k\_dmmu\_disable (void)

Disable data MMU

2.7.2.2 void or1k\_dmmu\_enable (void)

Enable data MMU

2.7.2.3 void or1k\_immu\_disable (void)

Disable instruction MMU

2.7.2.4 void or1k\_immu\_enable (void)

Enable instruction MMU

## 2.8 Timer control

#### **Modules**

Multicore and Synchronization Support

#### **Functions**

- int or1k timer init (unsigned int hz)
- void or1k\_timer\_set\_period (uint32\_t hz)
- void or1k\_timer\_set\_handler (void(\*handler)(void))
- void or1k timer set mode (uint32 t mode)
- void or1k timer enable (void)
- uint32\_t or1k\_timer\_disable (void)
- void or1k\_timer\_restore (uint32\_t sr\_tee)
- void or1k timer pause (void)
- void or1k\_timer\_reset (void)
- unsigned long or1k\_timer\_get\_ticks (void)
- void or1k timer reset ticks (void)
- uint32 t or1k critical start ()
- void or1k\_critical\_end (uint32\_t restore)

## 2.8.1 Detailed Description

The tick timer can be used for time measurement, operating system scheduling etc. By default it is initialized to continuously count the ticks of a certain period after calling or1k\_timer\_init(). The period can later be changed using or1k\_timer\_set\_period().

The timer is controlled using or1k\_timer\_enable(), or1k\_timer\_disable(), or1k\_timer\_restore(), or1k\_timer\_pause(). After initialization it is required to enable the timer the first time using or1k\_timer\_enable(). or1k\_timer\_disable() only disables the tick timer interrupts, it does not disable the timer counting. If you plan to use a pair of or1k\_timer\_disable() and or1k\_timer\_enable() to protect sections of your code against interrupts you should use or1k\_timer\_disable() and or1k\_timer\_restore(), as it may be possible that the timer interrupt was not enabled before disabling it, enable would then start it unconditionally. or1k\_timer\_pause() pauses the counting.

In the default mode you can get the tick value using or1k\_timer\_get\_ticks() and reset this value using or1k\_timer\_reset\_ticks().

Example for using the default mode:

```
int main() {
  uint32_t ticks = 0;
  uint32_t timerstate;
  or1k_timer_init(100);
  or1k_timer_enable();
  while (1) {
    while (ticks == or1k_timer_get_ticks()) { }
    timerstate = or1k_timer_disable();
    // do something atomar
    or1k_timer_restore(timerstate);
    if (ticks == 100) {
        printf("A second elapsed\n");
        or1k_timer_reset_ticks();
        ticks = 0;
    }
}
```

It is possible to change the mode of the tick timer using or1k\_timer\_set\_mode(). Allowed values are the correct bit pattern (including the bit positions) for the TTMR register, it is recommended to use the macros defined in sprdefs.h. For example, implementing an operating system with scheduling decisions of varying duration favors the implementation of single run tick timer. Here, each quantum is started before leaving the operating system kernel. The counter can be restarted with or1k\_timer\_reset(). Example:

2.8 Timer control

```
void tick_handler(void) {
// Make schedule decision
// and set new thread
orlk_timer_reset();
// End of exception, new thread will run
}
int main() {
// Configure operating system and start threads..
// Configure timer
orlk_timer_init(50);
orlk_timer_set_handler(&tick_handler);
orlk_timer_set_mode(SPR_TTMR_SR);
orlk_timer_enable();
// Schedule first thread and die..
}
```

## 2.8.2 Function Documentation

```
2.8.2.1 uint32_t or1k_timer_disable ( void )
```

#### Disable timer interrupt

This disables the timer interrupt exception and returns the state of the interrupt exception enable flag before the call. This can be used with or1k\_timer\_restore() to implement sequences of code that are not allowed to be interrupted. Using or1k\_timer\_enable() will unconditionally enable the interrupt independent of the state before calling or1k\_timer\_disable(). For an example see Timer control.

## Returns

Status of timer interrupt before call

```
2.8.2.2 void or1k_timer_enable (void)
```

## Enable timer interrupt

Enable the timer interrupt exception, independent of the status before. If you want to enable the timer conditionally, for example to implement a non-interruptible sequence of code, you should use or1k\_timer\_restore(). See the description of or1k\_timer\_disable() for more details.

The enable will also restore the mode if the timer was paused previously.

```
2.8.2.3 unsigned long or1k_timer_get_ticks ( void )
```

Get timer ticks

Get the global ticks of the default configuration. This will increment the tick counter according to the preconfigured period.

Returns

Current value of ticks

```
2.8.2.4 int or1k_timer_init ( unsigned int hz )
```

Initialize tick timer

This initializes the tick timer in default mode (see Timer control for details).

#### **Parameters**

hz Initial period of the tick timer

## Returns

0 if successful, -1 if timer not present

2.8.2.5 void or1k\_timer\_pause (void)

Pause timer counter

Pauses the counter of the tick timer. The counter will hold its current value and it can be started again with or1k\_timer\_enable() which will restore the configured mode.

2.8.2.6 void or1k\_timer\_reset ( void )

Reset timer counter

2.8.2.7 void or1k\_timer\_reset\_ticks ( void )

Reset timer ticks

Resets the timer ticks in default configuration to 0.

2.8.2.8 void or1k\_timer\_restore ( uint32\_t sr\_tee )

Restore timer interrupt exception flag

Restores the timer interrupt exception flag as returned by or1k\_timer\_disable(). See the description of or1k\_timer\_disable() and Timer control for details and an example.

**Parameters** 

sr\_tee | Status of timer interrupt

2.8.2.9 void or1k\_timer\_set\_handler ( void(\*)(void) handler )

Replace the timer interrupt handler

By default the tick timer is used to handle timer ticks. The user can replace this with an own handler for example when implementing an operating system.

**Parameters** 

handler The callback function pointer to the handler

2.8.2.10 void or1k\_timer\_set\_mode ( uint32\_t mode )

Set timer mode

The timer has different modes (see architecture manual). The default is to automatically restart counting (SPR\_TT-MR\_RT), others are single run (SPR\_TTMR\_SR) and continuous run (SPR\_TTMR\_CR).

2.8 Timer control

## **Parameters**

mode	a valid mode (use definitions from spr-defs.h as it is important that those are also at the
	correct position in the bit field!)

2.8.2.11 void or1k\_timer\_set\_period ( uint32\_t hz )

## Set period of timer

Set the period of the timer to a value in Hz. The frequency from the board support package is used to determine the match value.

## 2.9 Multicore and Synchronization Support

## **Functions**

- uint32\_t or1k\_coreid (void)
- · uint32\_t or1k\_numcores (void)
- uint32\_t or1k\_sync\_ll (void \*address)
- int or1k\_sync\_sc (void \*address, uint32\_t value)
- uint32\_t or1k\_sync\_cas (void \*address, uint32\_t compare, uint32\_t swap)
- int or1k sync tsl (void \*address)
- void or1k\_uart\_set\_read\_cb (void(\*cb)(char c))

## 2.9.1 Detailed Description

## 2.9.2 Function Documentation

2.9.2.1 uint32\_t or1k\_coreid ( void )

Read core identifier

Returns

Core identifier

2.9.2.2 uint32\_t or1k\_numcores ( void )

Read number of cores

Returns

Total number of cores

2.9.2.3 uint32\_t or1k\_sync\_cas ( void \* address, uint32\_t compare, uint32\_t swap )

## Compare and Swap

Loads a data item from the memory and compares a given value to it. If the values match, a new value is written to the memory, if they mismatch, the operation is aborted. The whole operation is atomic, i.e., it is guaranteed that no other core changes the value between the read and the write.

## **Parameters**

address	Address to operate on
compare	Compare value
swap	New value to write

## Returns

The value read from memory (can be used to check for success)

2.9.2.4 uint32\_t or1k\_sync\_ll ( void \* address )

## Load linked

Load a value from the given address and link it. If the following or1k\_sync\_sc() goes to the same address and there was no conflicting access between loading and storing, the value is written back, else the write fails.

#### **Parameters**

address	Address to load value from
---------	----------------------------

## Returns

Value read from the address

2.9.2.5 int or1k\_sync\_sc ( void \* address, uint32\_t value )

## Store conditional

Conditionally store a value to the address. The address must have been read before using or1k\_sync\_ll() and there must be no other load link after that, otherwise this will always fail. In case there was no other write to the same address in between the load link and the store conditional, the store is successful, otherwise it will also fail.

#### **Parameters**

address	Address to conditionally store to
value	Value to write to address

## Returns

1 if success, 0 if fail

2.9.2.6 int or1k\_sync\_tsl ( void \* address )

## Test and Set Lock

Check for a lock on an address. This means, if there is 0 at an address it will overwrite it with one and return 0. If the lock was already set (value 1 read from address), the function returns 1. The operation is atomic.

## **Parameters**

address	Address of the lock

## Returns

0 if success, 1 if failed

# Index

Cache control, 10	or1k_dmmu_enable
or1k_dcache_disable, 10	MMU control, 11
or1k_dcache_enable, 10	or1k_exception_handler_add
or1k_dcache_flush, 10	Exception handling, 7
or1k_icache_disable, 10	or1k_exception_handler_fptr
or1k_icache_enable, 10	Exception handling, 7
or1k_icache_flush, 10	or1k_icache_disable
	Cache control, 10
Exception handling, 7	or1k_icache_enable
or1k_exception_handler_add, 7	Cache control, 10
or1k_exception_handler_fptr, 7	or1k_icache_flush
	Cache control, 10
MMU control, 11	or1k_immu_disable
or1k_dmmu_disable, 11	MMU control, 11
or1k_dmmu_enable, 11	or1k_immu_enable
or1k_immu_disable, 11	MMU control, 11
or1k_immu_enable, 11	or1k_interrupt_disable
Miscellaneous utility functions, 9	OR1K interrupt control, 5
or1k_rand, 9	or1k interrupt enable
or1k_report, 9	OR1K interrupt control, 5
Multicore and Synchronization Support, 16	or1k_interrupt_handler_add
or1k_coreid, 16	OR1K interrupt control, 5
or1k_numcores, 16	or1k_interrupt_handler_fptr
or1k_sync_cas, 16	OR1K interrupt control, 5
or1k_sync_ll, 16	or1k_interrupts_disable
or1k_sync_sc, 17	OR1K interrupt control, 6
or1k_sync_tsl, 17	or1k_interrupts_enable
	•
OR1K interrupt control, 5	OR1K interrupt control, 6
or1k_interrupt_disable, 5	or1k_interrupts_restore
or1k_interrupt_enable, 5	OR1K interrupt control, 6
or1k_interrupt_handler_add, 5	or1k_numcores
or1k_interrupt_handler_fptr, 5	Multicore and Synchronization Support, 10
or1k_interrupts_disable, 6	or1k_rand
or1k_interrupts_enable, 6	Miscellaneous utility functions, 9
or1k_interrupts_restore, 6	or1k_report
OR1K macros, 3	Miscellaneous utility functions, 9
REG16, 3	or1k_sync_cas
REG32, 3	Multicore and Synchronization Support, 1
REG8, 3	or1k_sync_ll
or1k_coreid	Multicore and Synchronization Support, 1
Multicore and Synchronization Support, 16	or1k_sync_sc
or1k_dcache_disable	Multicore and Synchronization Support, 1
Cache control, 10	or1k_sync_tsl
or1k_dcache_enable	Multicore and Synchronization Support, 1
Cache control, 10	or1k_timer_disable
or1k_dcache_flush	Timer control, 13
Cache control, 10	or1k_timer_enable
or1k_dmmu_disable	Timer control, 13
MMU control, 11	or1k_timer_get_ticks

INDEX 19

Timer control, 13
or1k_timer_init
Timer control, 13
or1k_timer_pause
Timer control, 14
or1k_timer_reset
Timer control, 14
or1k_timer_reset_ticks
Timer control, 14
or1k_timer_restore
Timer control, 14
or1k_timer_set_handler
Timer control, 14
or1k_timer_set_mode
Timer control, 14
or1k_timer_set_period
Timer control, 15
DE0.40
REG16
OR1K macros, 3
REG32
OR1K macros, 3
REG8
OR1K macros, 3
SPR access, 8
Timer control, 12
or1k_timer_disable, 13
or1k_timer_enable, 13
or1k_timer_get_ticks, 13
or1k_timer_init, 13
or1k_timer_pause, 14
or1k_timer_reset, 14
or1k_timer_reset_ticks, 14
or1k_timer_restore, 14
or1k_timer_set_handler, 1
or1k_timer_set_mode, 14
or1k timer set period, 15