Cramium SoC (RISC-V Core Complex)

Cramium, Inc.

Feb 18, 2023

CONTENTS

CHAPTER

ONE

MODULES

1.1 Interrupt Controller

This device has an `EventManager`-based interrupt system. Individual modules generate *events* which are wired into a central interrupt controller.

When an interrupt occurs, you should look the interrupt number up in the CPU- specific interrupt table and then call the relevant module.

1.1.1 Assigned Interrupts

The following interrupts are assigned on this system:

Interrupt	Module
0	*IRQARRAY0*
1	*IRQARRAY1*
10	*IRQARRAY10*
11	*IRQARRAY11*
12	*IRQARRAY12*
13	*IRQARRAY13*
14	*IRQARRAY14*
15	*IRQARRAY15*
16	*IRQARRAY16*
17	*IRQARRAY17*
18	*IRQARRAY18*
19	*IRQARRAY19*
2	*IRQARRAY2*
3	*IRQARRAY3*
4	*IRQARRAY4*
5	*IRQARRAY5*
6	*IRQARRAY6*
7	*IRQARRAY7*
8	*IRQARRAY8*
9	*IRQARRAY9*
22	*MAILBOX*
23	*MB_CLIENT*
21	*SUSRES*
20	*TICKTIMER*

1.2 CPU

This VexRiscv core provides the following bus interfaces:

- 64-bit AXI-4 instruction cache bus (read-only cached)
- Data bus crossbar
  - 0x60000000-7FFFFFFF: 32-bit AXI-4 data cache bus (r/w cached)
  - 0x40000000-4FFFFFFF: 32-bit AXI-lite peripheral bus (r/w uncached)
- All busses run at ACLK speed
- WFI signal is broken out to *wfi_active*
- *satp* signals broken out for *coreuser* interpretation

The core itself contains the following features:

- VexRiscv CPU (simple, in-order RV32-IMAC with pipelining)
- Static branch prediction
- 4k, 4-way D-cache
- 4k, 4-way I-cache
- MMU and 8-entry TLB
- AES instruction extensions
- Non-cached regions (used for I/O):
  - 0x40000000 - 5FFFFFFF
  - 0xA0000000 - FFFFFFFF
  - Any non-cached regions not routed through peripheral bus are internal to the core block

CHAPTER

TWO

REGISTER GROUPS

2.1 D11CTIME

2.1.1 Deterministic Timeout

This module creates a heartbeat that is deterministic. If used correctly, it can help reduce timing side channels on secure processes by giving them an independent, coarse source of time. The idea is that a secure process may handle a request, and then wait for a heartbeat from the D11cTime module to change polarity, which occurs at a regular interval, before returning the result.

There is a trade-off on how frequent the heartbeat is versus information leakage versus overall throughput of the secure module's responses. If the heartbeat is faster than the maximum time to complete a computation, then information leakage will occur; if it is much slower than the maximum time to complete a computation, then performance is reduced. Deterministic timeout is not the end-all solution; adding noise and computational confounders are also countermeasures to be considered, but this is one of the simpler approaches, and it is relatively hardware-efficient.

This block has been configured to default to 1.0ms period, assuming ACLK is 800.0MHz.

2.1.2 Register Listing for D11CTIME

Register	Address
D11CTIME_CONTROL	*0x58000000*
D11CTIME_HEARTBEAT	*0x58000004*

D11CTIME_CONTROL

Address: 0x58000000 + 0x0 = 0x58000000

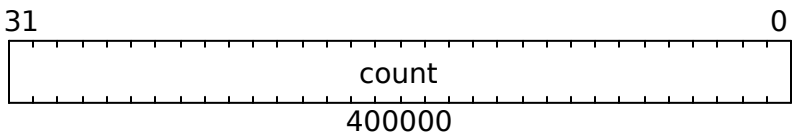

Fig. 1: D11CTIME_CONTROL

Field	Name	Description
[31:0]	COUNT	Number of ACLK ticks before creating a heart beat

D11CTIME_HEARTBEAT

Address: 0x58000000 + 0x4 = 0x58000004

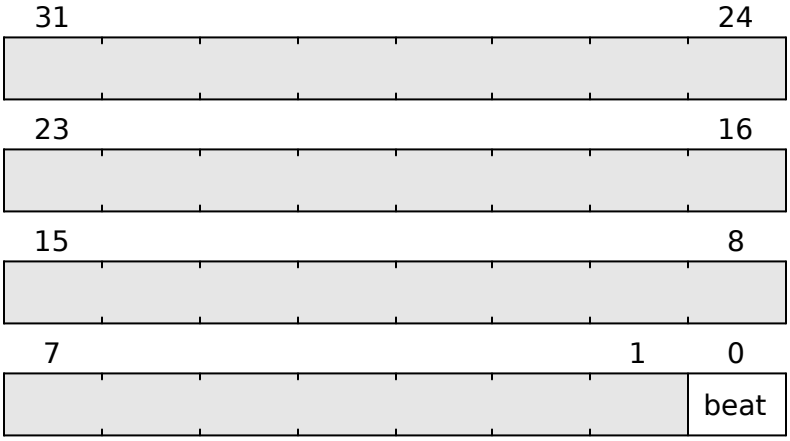

Fig. 2: D11CTIME_HEARTBEAT

Field	Name	Description
[0]	BEAT	Set to *1* at the next *count* interval rollover since *clear* was set.

2.2 SUSRES

2.2.1 Suspend/Resume Helper

This module is a utility module that assists with suspend and resume functions. It has the ability to 'reach into' the Ticktimer space to help coordinate a clean, monatomic shut down from a suspend/resume manager that exists in a different, isolated process space from the TickTimer.

It also contains a register which tracks the current resume state. The bootloader controls the kernel's behavior by setting this bit prior to resuming operation.

2.2.2 Register Listing for SUSRES

Register	Address
SUSRES_CONTROL	*0x58001000*
SUSRES_RESUME_TIME1	*0x58001004*
SUSRES_RESUME_TIME0	*0x58001008*
SUSRES_TIME1	*0x5800100c*
SUSRES_TIME0	*0x58001010*
SUSRES_STATUS	*0x58001014*
SUSRES_STATE	*0x58001018*
SUSRES_INTERRUPT	*0x5800101c*
SUSRES_EV_STATUS	*0x58001020*
SUSRES_EV_PENDING	*0x58001024*
SUSRES_EV_ENABLE	*0x58001028*

SUSRES_CONTROL

Address: 0x58001000 + 0x0 = 0x58001000

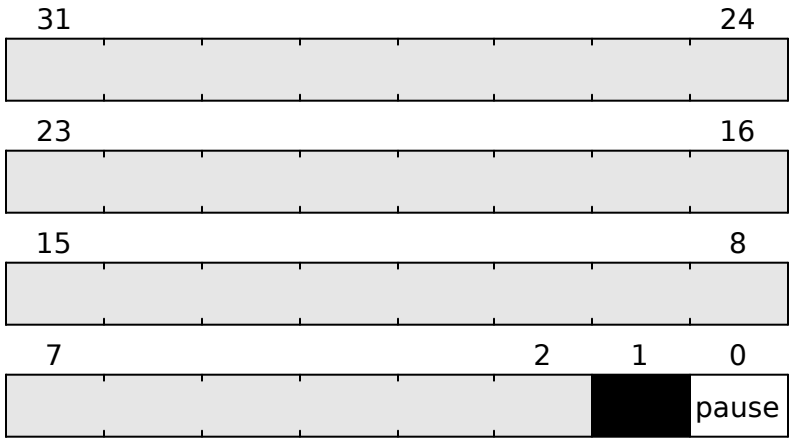

Fig. 3: SUSRES_CONTROL

Field	Name	Description
[0]	PAUSE	Write a *1* to this field to request a pause to counting, 0 for free-run. Count pauses on the next tick quanta.
[1]	LOAD	If paused, write a *1* to this bit to load a resume value to the timer. If not paused, this bit is ignored. Writing a 1 to this bit triggers the function.

SUSRES_RESUME_TIME1

Address: 0x58001000 + 0x4 = 0x58001004

Bits 32-63 of *SUSRES_RESUME_TIME*. Elapsed time to load. Loaded upon writing *1* to the load bit in the control register. This will immediately affect the msleep extension.

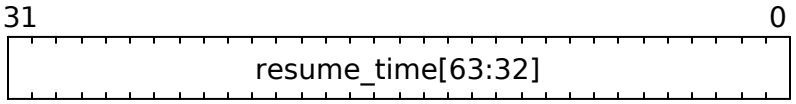

Fig. 4: SUSRES_RESUME_TIME1

SUSRES_RESUME_TIME0

Address: 0x58001000 + 0x8 = 0x58001008

Bits 0-31 of *SUSRES_RESUME_TIME*.

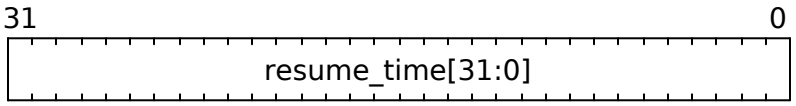

Fig. 5: SUSRES_RESUME_TIME0

SUSRES_TIME1

Address: 0x58001000 + 0xc = 0x5800100c

Bits 32-63 of *SUSRES_TIME*. Cycle-accurate mirror copy of time in systicks, from the TickTimer

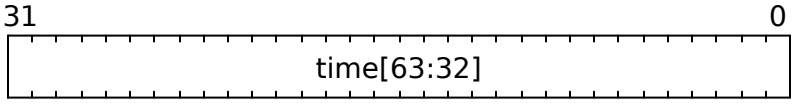

Fig. 6: SUSRES_TIME1

SUSRES_TIME0

Address: 0x58001000 + 0x10 = 0x58001010

Bits 0-31 of *SUSRES_TIME*.

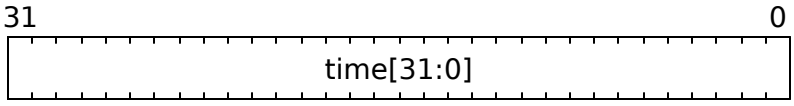

Fig. 7: SUSRES_TIME0

SUSRES_STATUS

Address: 0x58001000 + 0x14 = 0x58001014

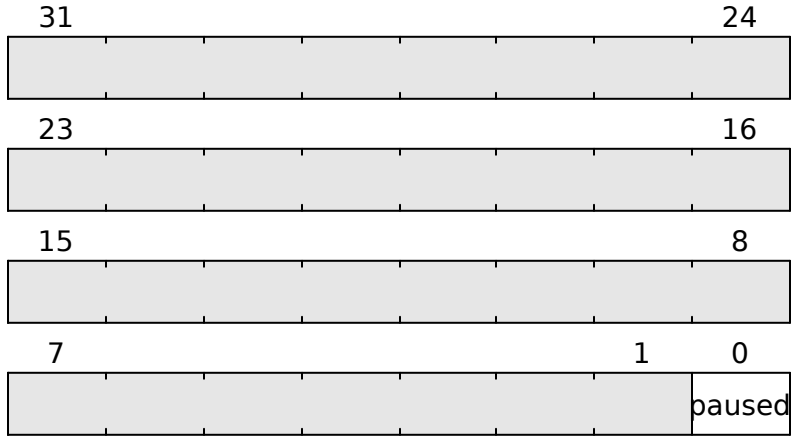

Fig. 8: SUSRES_STATUS

Field	Name	Description
[0]	PAUSED	When set, indicates that the counter has been paused

SUSRES_STATE

Address: 0x58001000 + 0x18 = 0x58001018

Field	Name	Description
[0]	RESUME	Used to transfer the resume state information from the loader to Xous. If set, indicates we are on the resume half of a suspend/resume.
[1]	WAS_FORCED	Used by the bootloader to indicate to the kernel if the current resume was from a forced suspend (e.g. a timeout happened and a server may be unclean.

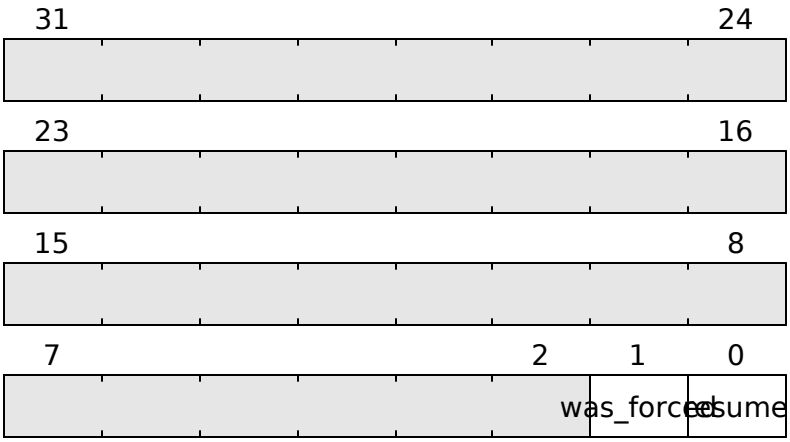

Fig. 9: SUSRES_STATE

SUSRES_INTERRUPT

Address: 0x58001000 + 0x1c = 0x5800101c

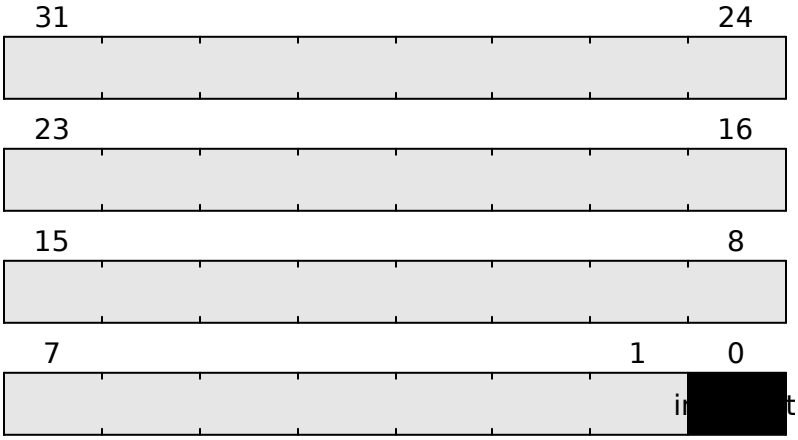

Fig. 10: SUSRES_INTERRUPT

Field	Name	Description
[0]	INTERRUPT	Writing this causes an interrupt to fire. Used by Xous to initiate suspend/resume from an interrupt context. Writing a 1 to this bit triggers the function.

SUSRES_EV_STATUS

Address: 0x58001000 + 0x20 = 0x58001020

This register contains the current raw level of the soft_int event trigger. Writes to this register have no effect.

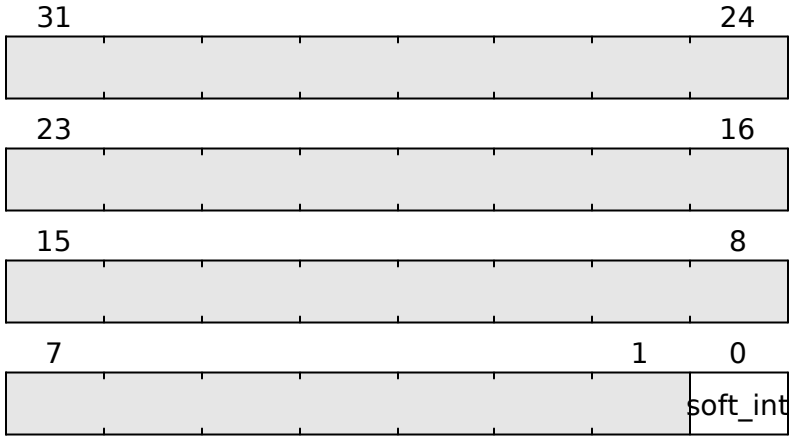

Fig. 11: SUSRES_EV_STATUS

Field	Name	Description
[0]	SOFT_INT	Level of the `soft_int` event

SUSRES_EV_PENDING

Address: 0x58001000 + 0x24 = 0x58001024

When a soft_int event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.

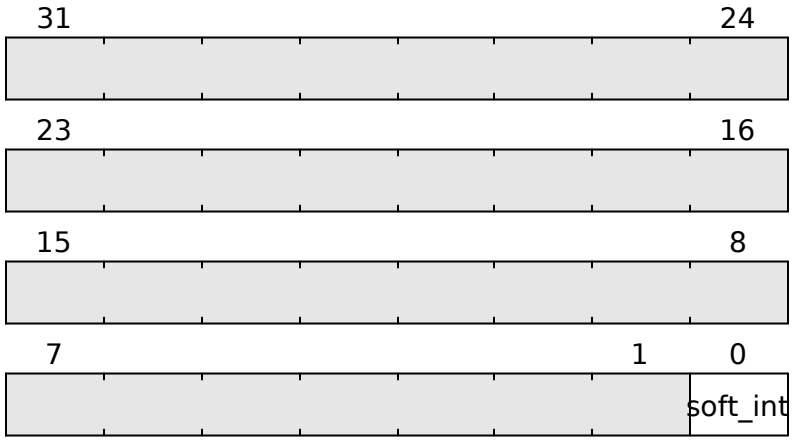

Fig. 12: SUSRES_EV_PENDING

Field	Name	Description
[0]	SOFT_INT	*1* if a *soft_int* event occurred. This Event is triggered on a **falling** edge.

SUSRES_EV_ENABLE

Address: 0x58001000 + 0x28 = 0x58001028

This register enables the corresponding soft_int events. Write a `0` to this register to disable individual events.

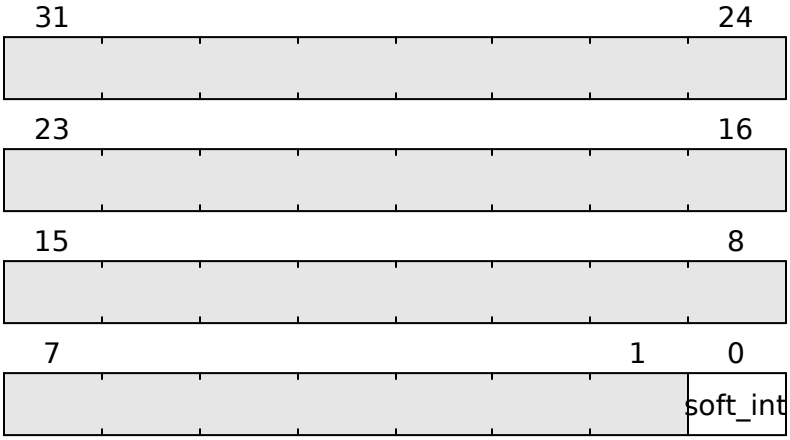

Fig. 13: SUSRES_EV_ENABLE

Field	Name	Description
[0]	SOFT_INT	Write a 1 to enable the `soft_int` Event

2.3 COREUSER

CoreUser is a hardware signal that indicates that the code executing is in a highly trusted piece of code. This is determined by examining a configurable combination of the SATP's ASID, PPN values, and/or privilege bits from *$mstatus.mpp*, allowing the OS to target certain virtual memory spaces as more trusted than others. *CoreUser* can only be computed when the RISC-V core is in Sv32 mode (that is, virtual memory has been enabled).

When specifying PPN values, two windows are provided, *a* and *b*. The windows are computed independently, and then OR'd together. The *a* and *b* windows should be non-overlapping. If they overlap, or the windows are poorly-specified, the behavior is not guaranteed. The intention of having two windows is not so that the OS can specify only two processes as *CoreUser*. Rather, the OS should design to allocate all CoreUser processes within a single range that is protected by a single window. The alternate window is provided only so that the OS can have a scratch space to re-organize or shuffle around process spaces at a higher level.

When specifying privilege, one specifies the state that must match for *coreuser* to be active. For a microkernel, one would specify a non-elevated permission level, as secure access is always delegated to a userland service. For a monokernel, one would specify an elevated permission level.

The *CoreUser* signal is not cycle-precise; it will assert roughly 2 cycles after the *satp* is updated. Furthermore, the *satp* ASID field is an advisory field that isn't used by CPU hardware to enforce page access. You can think of *coreuser* as a signal that the kernel can control to indicate if the context we are swapping into should be trusted. Fortunately, any update to *satp* in a virtual memory OS should be followed by an *sfence* instruction (to invalidate TLB mappings etc.), which gives time for the *coreuser* signal to propagate through the pipeline.

Thus in practice by the time the first instruction of user code runs, *coreuser* should be set properly. However, from a security audit perspective, it is important to keep in mind that there is a race condition between the *satp* setting and user code execution.

2.3.1 Register Listing for COREUSER

Register	Address
COREUSER_SET_ASID	*0x58002000*
COREUSER_GET_ASID_ADDR	*0x58002004*
COREUSER_GET_ASID_VALUE	*0x58002008*
COREUSER_SET_PRIVILEGE	*0x5800200c*
COREUSER_CONTROL	*0x58002010*
COREUSER_PROTECT	*0x58002014*
COREUSER_WINDOW_AL	*0x58002018*
COREUSER_WINDOW_AH	*0x5800201c*
COREUSER_WINDOW_BL	*0x58002020*
COREUSER_WINDOW_BH	*0x58002024*

COREUSER_SET_ASID

Address: 0x58002000 + 0x0 = 0x58002000

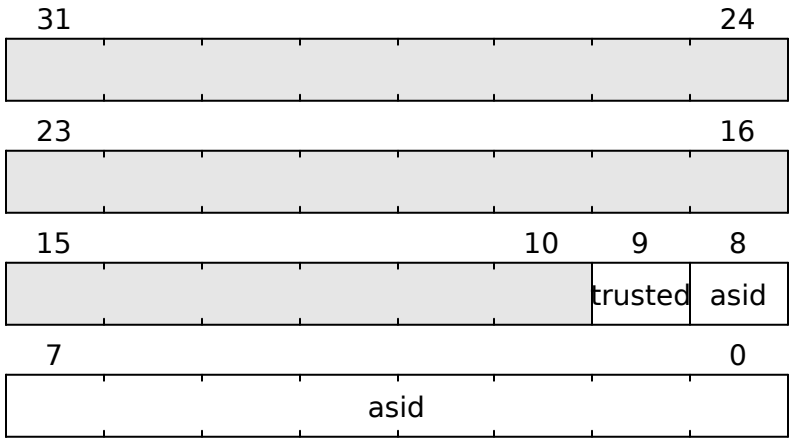

Fig. 14: COREUSER_SET_ASID

Field	Name	Description
[8:0]	ASID	ASID to set. Writing to this register commits the value in *trusted* to the specified *asid* value
[9]	TRUSTED	Set to *1* if the ASID is trusted

COREUSER_GET_ASID_ADDR

Address: 0x58002000 + 0x4 = 0x58002004

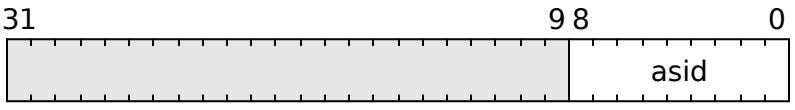

Fig. 15: COREUSER_GET_ASID_ADDR

Field	Name	Description
[8:0]	ASID	ASID to read back.

COREUSER_GET_ASID_VALUE

Address: 0x58002000 + 0x8 = 0x58002008

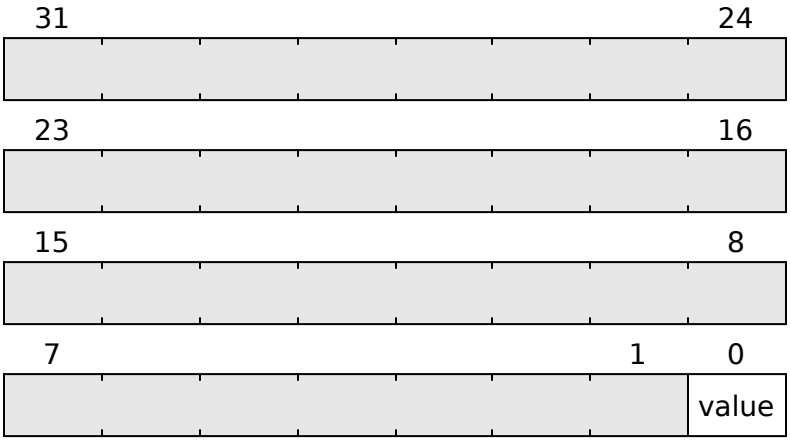

Fig. 16: COREUSER_GET_ASID_VALUE

Field	Name	Description
[0]	VALUE	Value corresponding to the ASID specified in *get_asid_addr*. *1* means trusted

COREUSER_SET_PRIVILEGE

Address: 0x58002000 + 0xc = 0x5800200c

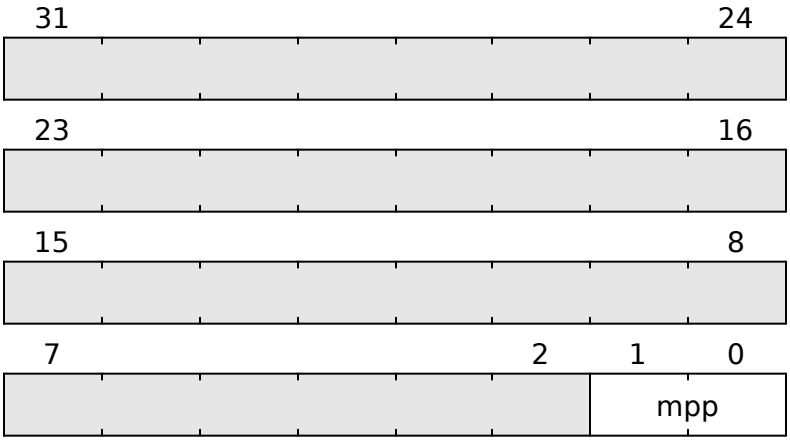

Fig. 17: COREUSER_SET_PRIVILEGE

Field	Name	Description
[1:0]	MPP	Value of *mpp* bit from *mstatus* that must match for code to be trusted

COREUSER_CONTROL

Address: 0x58002000 + 0x10 = 0x58002010

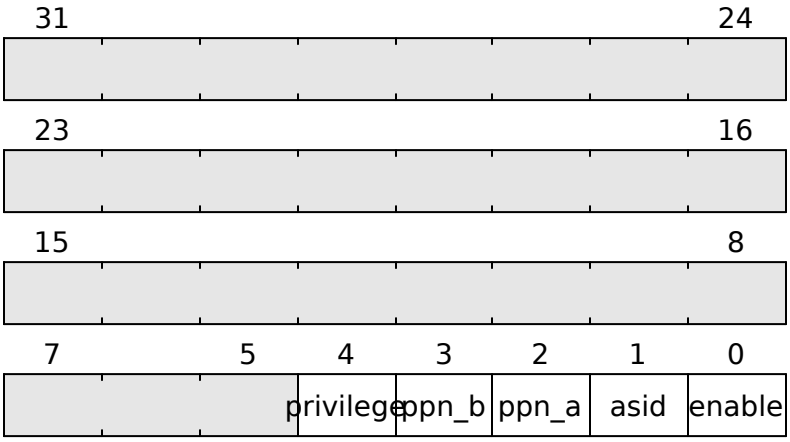

Fig. 18: COREUSER_CONTROL

Field	Name	Description
[0]	ENABLE	Enable *CoreUser* computation. When set to *1*, the settings are applied; when cleared to *0*, the *CoreUser* signal is always valid. Defaults to *0*.
[1]	ASID	When *1*, requires the ASID mapping to be trusted to assert *CoreUser*
[2]	PPN_A	When set to *1*, requires the *a ppn* window to be trusted to assert *CoreUser*
[3]	PPN_B	When set to *1*, requires the *b ppn* window to be trusted to assert *CoreUser*
[4]	PRIVILEGE	When set to *1*, requires the current privilege state to match that specified in *set_privilege.mpp*

COREUSER_PROTECT

Address: 0x58002000 + 0x14 = 0x58002014

Writing *1* to this bit prevents any further updates to CoreUser configuration status. Can only be reversed with a system reset.

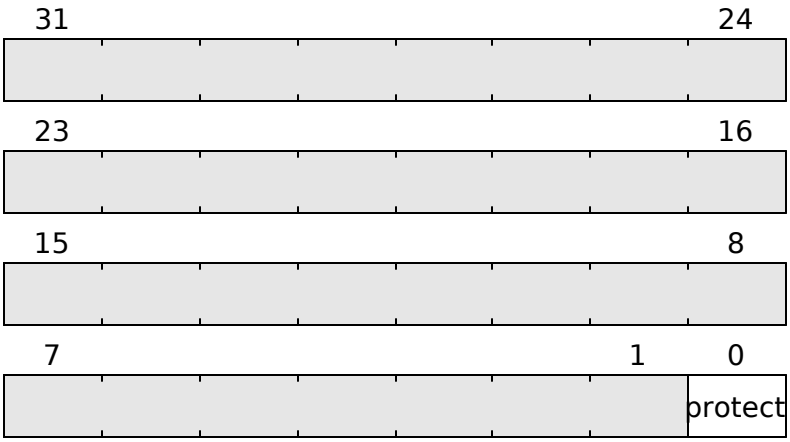

Fig. 19: COREUSER_PROTECT

COREUSER_WINDOW_AL

Address: 0x58002000 + 0x18 = 0x58002018

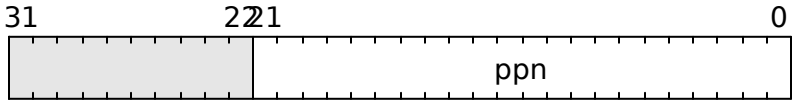

Fig. 20: COREUSER_WINDOW_AL

Field	Name	Description
[21:0]	PPN	PPN match value, *a* window lower bound. Matches if ppn is greater than or equal to this value

COREUSER_WINDOW_AH

Address: 0x58002000 + 0x1c = 0x5800201c

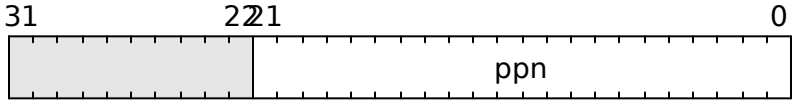

Fig. 21: COREUSER_WINDOW_AH

Field	Name	Description
[21:0]	PPN	PPN match value, *a* window upper bound. Matches if ppn is less than or equal to this value (so a value of 255 would match everything from 0 to 255; resulting in 256 total locations

COREUSER_WINDOW_BL

Address: 0x58002000 + 0x20 = 0x58002020

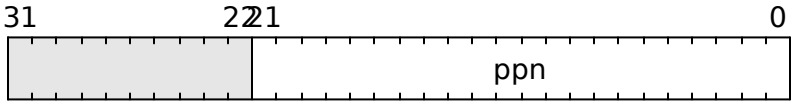

Fig. 22: COREUSER_WINDOW_BL

Field	Name	Description
[21:0]	PPN	PPN match value, *b* window lower bound. Matches if ppn is greater than or equal to this value

COREUSER_WINDOW_BH

Address: 0x58002000 + 0x24 = 0x58002024

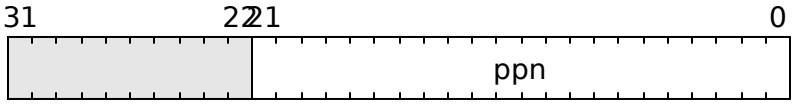

Fig. 23: COREUSER_WINDOW_BH

Field	Name	Description
[21:0]	PPN	PPN match value, *b* window upper bound. Matches if ppn is less than or equal to this value (so a value of 255 would match everything from 0 to 255; resulting in 256 total locations

2.4 CSRTEST

2.4.1 Register Listing for CSRTEST

Register	Address
CSRTEST_WTEST	*0x58003000*
CSRTEST_RTEST	*0x58003004*

CSRTEST_WTEST

Address: 0x58003000 + 0x0 = 0x58003000

Write test data here

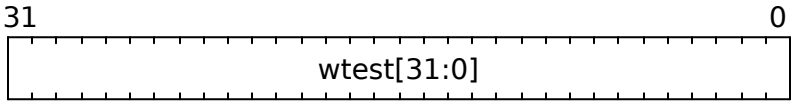

Fig. 24: CSRTEST_WTEST

CSRTEST_RTEST

Address: 0x58003000 + 0x4 = 0x58003004

Read test data here

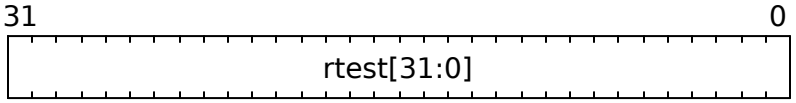

Fig. 25: CSRTEST_RTEST

2.5 IRQARRAY0

IrqArray provides a large bank of interrupts for SoC integration. It is different from e.g. the NVIC or CLINT in that the register bank is structured along page boundaries, so that the interrupt handler CSRs can be owned by a specific virtual memory process, instead of bouncing through a common handler and forcing an inter-process message to be generated to route interrupts to their final destination.

The incoming interrupt signals are assumed to be synchronized to *aclk*.

Priorities are enforced entirely through software; the handler must read the *pending* bits and decide which ones should be handled first.

The *EventSource* is an *EventSourceFlex* which can handle pulses and levels, as well as software triggers.

The interrupt pending bit is latched when the trigger goes high, and stays high until software clears the event. The trigger takes precedence over clearing, so if the interrupt source is not cleared prior to clearing the interrupt pending bit, the interrupt will trigger again.

status reflects the instantaneous value of the trigger.

A separate input line is provided so that software can induce an interrupt by writing to a soft-trigger bit.

2.5.1 Register Listing for IRQARRAY0

Register	Address
IRQARRAY0_EV_SOFT	*0x58004000*
IRQARRAY0_EV_STATUS	*0x58004004*
IRQARRAY0_EV_PENDING	*0x58004008*
IRQARRAY0_EV_ENABLE	*0x5800400c*

IRQARRAY0_EV_SOFT

Address: 0x58004000 + 0x0 = 0x58004000

Software interrupt trigger register.

Bits set to *1* will trigger an interrupt. Interrupts trigger on write, but the value will persist in the register, allowing software to determine if a software interrupt was triggered by reading back the register.

Software is responsible for clearing the register to 0.

Repeated *1* writes without clearing will still trigger an interrupt.

Fig. 26: IRQARRAY0_EV_SOFT

Field	Name	Description
[19:0]	TRIGGER	Writing a 1 to this bit triggers the function.

IRQARRAY0_EV_STATUS

Address: 0x58004000 + 0x4 = 0x58004004

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

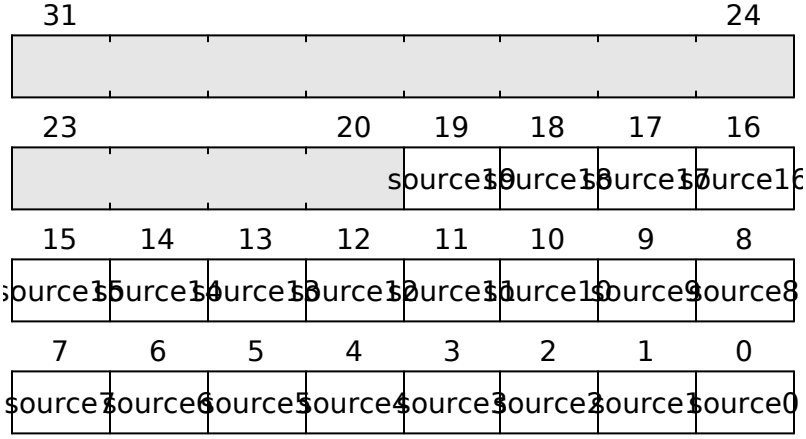

Fig. 27: IRQARRAY0_EV_STATUS

Field	Name	Description
[0]	SOURCE0	Level of the source0 event
[1]	SOURCE1	Level of the source1 event
[2]	SOURCE2	Level of the source2 event
[3]	SOURCE3	Level of the source3 event
[4]	SOURCE4	Level of the source4 event
[5]	SOURCE5	Level of the source5 event
[6]	SOURCE6	Level of the source6 event
[7]	SOURCE7	Level of the source7 event
[8]	SOURCE8	Level of the source8 event
[9]	SOURCE9	Level of the source9 event
[10]	SOURCE10	Level of the source10 event
[11]	SOURCE11	Level of the source11 event
[12]	SOURCE12	Level of the source12 event
[13]	SOURCE13	Level of the source13 event
[14]	SOURCE14	Level of the source14 event
[15]	SOURCE15	Level of the source15 event
[16]	SOURCE16	Level of the source16 event
[17]	SOURCE17	Level of the source17 event
[18]	SOURCE18	Level of the source18 event
[19]	SOURCE19	Level of the source19 event

IRQARRAY0_EV_PENDING

Address: 0x58004000 + 0x8 = 0x58004008

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

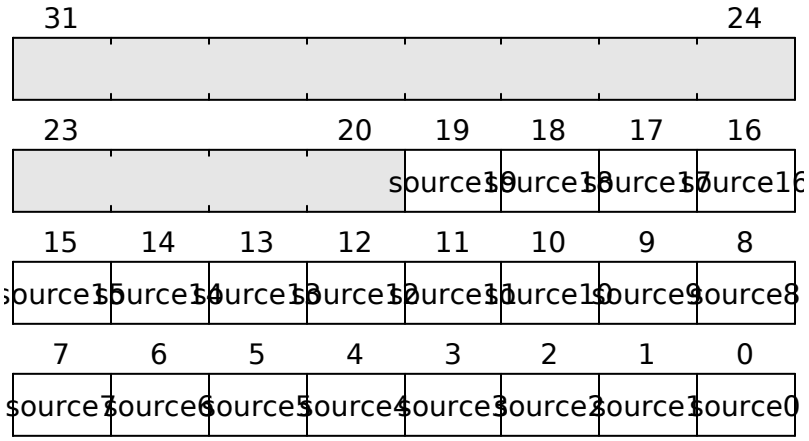

Fig. 28: IRQARRAY0_EV_PENDING

Field	Name	Description
[0]	SOURCE0	*1* when a source0 event occurs. This event uses an *EventSourceFlex* form of triggering
[1]	SOURCE1	*1* when a source1 event occurs. This event uses an *EventSourceFlex* form of triggering
[2]	SOURCE2	*1* when a source2 event occurs. This event uses an *EventSourceFlex* form of triggering
[3]	SOURCE3	*1* when a source3 event occurs. This event uses an *EventSourceFlex* form of triggering
[4]	SOURCE4	*1* when a source4 event occurs. This event uses an *EventSourceFlex* form of triggering
[5]	SOURCE5	*1* when a source5 event occurs. This event uses an *EventSourceFlex* form of triggering
[6]	SOURCE6	*1* when a source6 event occurs. This event uses an *EventSourceFlex* form of triggering
[7]	SOURCE7	*1* when a source7 event occurs. This event uses an *EventSourceFlex* form of triggering
[8]	SOURCE8	*1* when a source8 event occurs. This event uses an *EventSourceFlex* form of triggering
[9]	SOURCE9	*1* when a source9 event occurs. This event uses an *EventSourceFlex* form of triggering
[10]	SOURCE10	*1* when a source10 event occurs. This event uses an *EventSourceFlex* form of triggering
[11]	SOURCE11	*1* when a source11 event occurs. This event uses an *EventSourceFlex* form of triggering
[12]	SOURCE12	*1* when a source12 event occurs. This event uses an *EventSourceFlex* form of triggering
[13]	SOURCE13	*1* when a source13 event occurs. This event uses an *EventSourceFlex* form of triggering
[14]	SOURCE14	*1* when a source14 event occurs. This event uses an *EventSourceFlex* form of triggering
[15]	SOURCE15	*1* when a source15 event occurs. This event uses an *EventSourceFlex* form of triggering
[16]	SOURCE16	*1* when a source16 event occurs. This event uses an *EventSourceFlex* form of triggering
[17]	SOURCE17	*1* when a source17 event occurs. This event uses an *EventSourceFlex* form of triggering
[18]	SOURCE18	*1* when a source18 event occurs. This event uses an *EventSourceFlex* form of triggering
[19]	SOURCE19	*1* when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

IRQARRAY0_EV_ENABLE

Address: 0x58004000 + 0xc = 0x5800400c

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

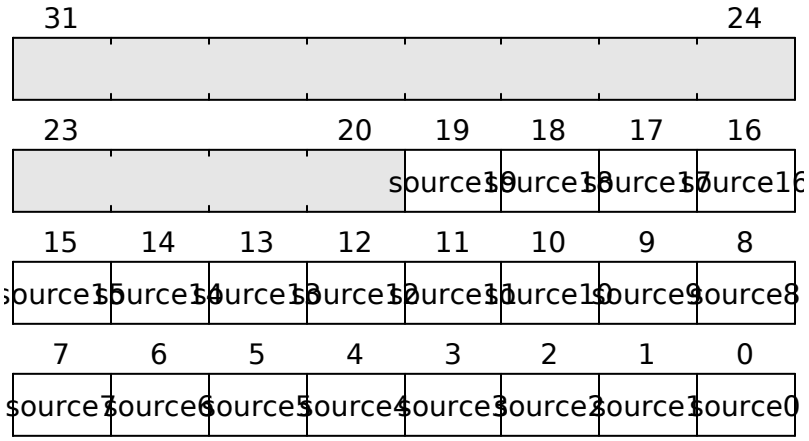

Fig. 29: IRQARRAY0_EV_ENABLE

Field	Name	Description
[0]	SOURCE0	Write a 1 to enable the `source0` Event
[1]	SOURCE1	Write a 1 to enable the `source1` Event
[2]	SOURCE2	Write a 1 to enable the `source2` Event
[3]	SOURCE3	Write a 1 to enable the `source3` Event
[4]	SOURCE4	Write a 1 to enable the `source4` Event
[5]	SOURCE5	Write a 1 to enable the `source5` Event
[6]	SOURCE6	Write a 1 to enable the `source6` Event
[7]	SOURCE7	Write a 1 to enable the `source7` Event
[8]	SOURCE8	Write a 1 to enable the `source8` Event
[9]	SOURCE9	Write a 1 to enable the `source9` Event
[10]	SOURCE10	Write a 1 to enable the `source10` Event
[11]	SOURCE11	Write a 1 to enable the `source11` Event
[12]	SOURCE12	Write a 1 to enable the `source12` Event
[13]	SOURCE13	Write a 1 to enable the `source13` Event
[14]	SOURCE14	Write a 1 to enable the `source14` Event
[15]	SOURCE15	Write a 1 to enable the `source15` Event
[16]	SOURCE16	Write a 1 to enable the `source16` Event
[17]	SOURCE17	Write a 1 to enable the `source17` Event
[18]	SOURCE18	Write a 1 to enable the `source18` Event
[19]	SOURCE19	Write a 1 to enable the `source19` Event

2.6 IRQARRAY1

IrqArray provides a large bank of interrupts for SoC integration. It is different from e.g. the NVIC or CLINT in that the register bank is structured along page boundaries, so that the interrupt handler CSRs can be owned by a specific virtual memory process, instead of bouncing through a common handler and forcing an inter-process message to be generated to route interrupts to their final destination.

The incoming interrupt signals are assumed to be synchronized to *aclk*.

Priorities are enforced entirely through software; the handler must read the *pending* bits and decide which ones should be handled first.

The *EventSource* is an *EventSourceFlex* which can handle pulses and levels, as well as software triggers.

The interrupt pending bit is latched when the trigger goes high, and stays high until software clears the event. The trigger takes precedence over clearing, so if the interrupt source is not cleared prior to clearing the interrupt pending bit, the interrupt will trigger again.

status reflects the instantaneous value of the trigger.

A separate input line is provided so that software can induce an interrupt by writing to a soft-trigger bit.

2.6.1 Register Listing for IRQARRAY1

Register	Address
IRQARRAY1_EV_SOFT	*0x58005000*
IRQARRAY1_EV_STATUS	*0x58005004*
IRQARRAY1_EV_PENDING	*0x58005008*
IRQARRAY1_EV_ENABLE	*0x5800500c*

IRQARRAY1_EV_SOFT

Address: 0x58005000 + 0x0 = 0x58005000

Software interrupt trigger register.

Bits set to *1* will trigger an interrupt. Interrupts trigger on write, but the value will persist in the register, allowing software to determine if a software interrupt was triggered by reading back the register.

Software is responsible for clearing the register to 0.

Repeated *1* writes without clearing will still trigger an interrupt.

Fig. 30: IRQARRAY1_EV_SOFT

Field	Name	Description
[19:0]	TRIGGER	Writing a 1 to this bit triggers the function.

IRQARRAY1_EV_STATUS

Address: 0x58005000 + 0x4 = 0x58005004

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

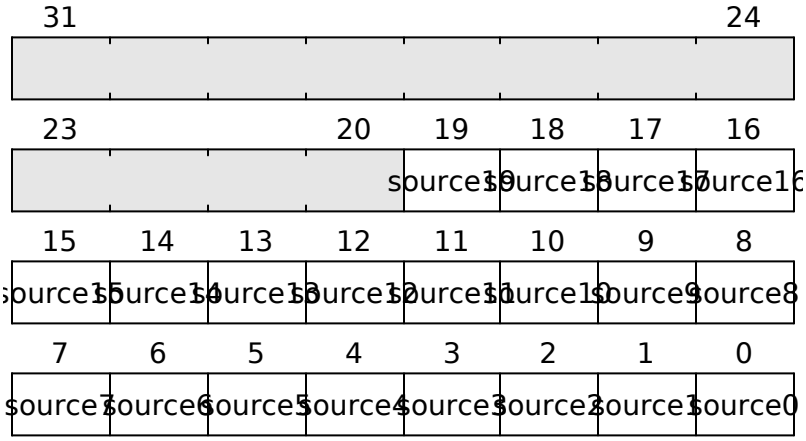

Fig. 31: IRQARRAY1_EV_STATUS

Field	Name	Description
[0]	SOURCE0	Level of the source0 event
[1]	SOURCE1	Level of the source1 event
[2]	SOURCE2	Level of the source2 event
[3]	SOURCE3	Level of the source3 event
[4]	SOURCE4	Level of the source4 event
[5]	SOURCE5	Level of the source5 event
[6]	SOURCE6	Level of the source6 event
[7]	SOURCE7	Level of the source7 event
[8]	SOURCE8	Level of the source8 event
[9]	SOURCE9	Level of the source9 event
[10]	SOURCE10	Level of the source10 event
[11]	SOURCE11	Level of the source11 event
[12]	SOURCE12	Level of the source12 event
[13]	SOURCE13	Level of the source13 event
[14]	SOURCE14	Level of the source14 event
[15]	SOURCE15	Level of the source15 event
[16]	SOURCE16	Level of the source16 event
[17]	SOURCE17	Level of the source17 event
[18]	SOURCE18	Level of the source18 event
[19]	SOURCE19	Level of the source19 event

IRQARRAY1_EV_PENDING

Address: 0x58005000 + 0x8 = 0x58005008

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

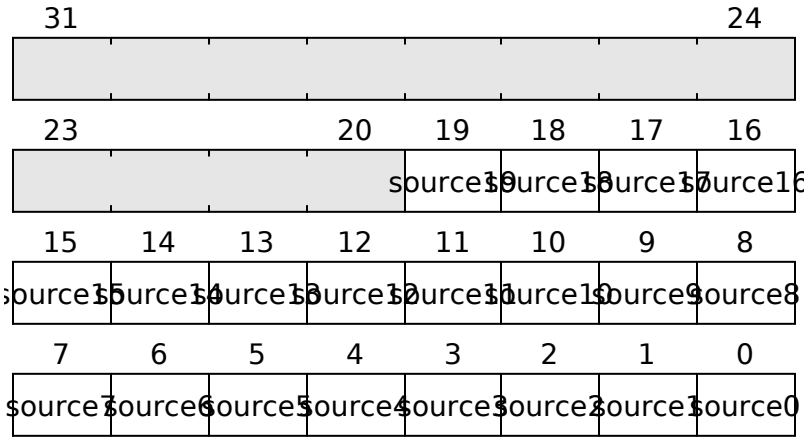

Fig. 32: IRQARRAY1_EV_PENDING

Field	Name	Description
[0]	SOURCE0	*1* when a source0 event occurs. This event uses an *EventSourceFlex* form of triggering
[1]	SOURCE1	*1* when a source1 event occurs. This event uses an *EventSourceFlex* form of triggering
[2]	SOURCE2	*1* when a source2 event occurs. This event uses an *EventSourceFlex* form of triggering
[3]	SOURCE3	*1* when a source3 event occurs. This event uses an *EventSourceFlex* form of triggering
[4]	SOURCE4	*1* when a source4 event occurs. This event uses an *EventSourceFlex* form of triggering
[5]	SOURCE5	*1* when a source5 event occurs. This event uses an *EventSourceFlex* form of triggering
[6]	SOURCE6	*1* when a source6 event occurs. This event uses an *EventSourceFlex* form of triggering
[7]	SOURCE7	*1* when a source7 event occurs. This event uses an *EventSourceFlex* form of triggering
[8]	SOURCE8	*1* when a source8 event occurs. This event uses an *EventSourceFlex* form of triggering
[9]	SOURCE9	*1* when a source9 event occurs. This event uses an *EventSourceFlex* form of triggering
[10]	SOURCE10	*1* when a source10 event occurs. This event uses an *EventSourceFlex* form of triggering
[11]	SOURCE11	*1* when a source11 event occurs. This event uses an *EventSourceFlex* form of triggering
[12]	SOURCE12	*1* when a source12 event occurs. This event uses an *EventSourceFlex* form of triggering
[13]	SOURCE13	*1* when a source13 event occurs. This event uses an *EventSourceFlex* form of triggering
[14]	SOURCE14	*1* when a source14 event occurs. This event uses an *EventSourceFlex* form of triggering
[15]	SOURCE15	*1* when a source15 event occurs. This event uses an *EventSourceFlex* form of triggering
[16]	SOURCE16	*1* when a source16 event occurs. This event uses an *EventSourceFlex* form of triggering
[17]	SOURCE17	*1* when a source17 event occurs. This event uses an *EventSourceFlex* form of triggering
[18]	SOURCE18	*1* when a source18 event occurs. This event uses an *EventSourceFlex* form of triggering
[19]	SOURCE19	*1* when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

IRQARRAY1_EV_ENABLE

Address: 0x58005000 + 0xc = 0x5800500c

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

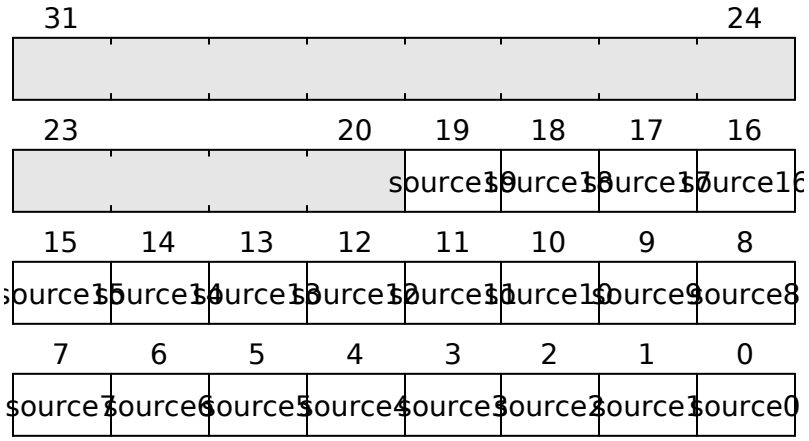

Fig. 33: IRQARRAY1_EV_ENABLE

Field	Name	Description
[0]	SOURCE0	Write a 1 to enable the `source0` Event
[1]	SOURCE1	Write a 1 to enable the `source1` Event
[2]	SOURCE2	Write a 1 to enable the `source2` Event
[3]	SOURCE3	Write a 1 to enable the `source3` Event
[4]	SOURCE4	Write a 1 to enable the `source4` Event
[5]	SOURCE5	Write a 1 to enable the `source5` Event
[6]	SOURCE6	Write a 1 to enable the `source6` Event
[7]	SOURCE7	Write a 1 to enable the `source7` Event
[8]	SOURCE8	Write a 1 to enable the `source8` Event
[9]	SOURCE9	Write a 1 to enable the `source9` Event
[10]	SOURCE10	Write a 1 to enable the `source10` Event
[11]	SOURCE11	Write a 1 to enable the `source11` Event
[12]	SOURCE12	Write a 1 to enable the `source12` Event
[13]	SOURCE13	Write a 1 to enable the `source13` Event
[14]	SOURCE14	Write a 1 to enable the `source14` Event
[15]	SOURCE15	Write a 1 to enable the `source15` Event
[16]	SOURCE16	Write a 1 to enable the `source16` Event
[17]	SOURCE17	Write a 1 to enable the `source17` Event
[18]	SOURCE18	Write a 1 to enable the `source18` Event
[19]	SOURCE19	Write a 1 to enable the `source19` Event

2.7 IRQARRAY10

IrqArray provides a large bank of interrupts for SoC integration. It is different from e.g. the NVIC or CLINT in that the register bank is structured along page boundaries, so that the interrupt handler CSRs can be owned by a specific virtual memory process, instead of bouncing through a common handler and forcing an inter-process message to be generated to route interrupts to their final destination.

The incoming interrupt signals are assumed to be synchronized to *aclk*.

Priorities are enforced entirely through software; the handler must read the *pending* bits and decide which ones should be handled first.

The *EventSource* is an *EventSourceFlex* which can handle pulses and levels, as well as software triggers.

The interrupt pending bit is latched when the trigger goes high, and stays high until software clears the event. The trigger takes precedence over clearing, so if the interrupt source is not cleared prior to clearing the interrupt pending bit, the interrupt will trigger again.

status reflects the instantaneous value of the trigger.

A separate input line is provided so that software can induce an interrupt by writing to a soft-trigger bit.

2.7.1 Register Listing for IRQARRAY10

Register	Address
IRQARRAY10_EV_SOFT	*0x58006000*
IRQARRAY10_EV_STATUS	*0x58006004*
IRQARRAY10_EV_PENDING	*0x58006008*
IRQARRAY10_EV_ENABLE	*0x5800600c*

IRQARRAY10_EV_SOFT

Address: 0x58006000 + 0x0 = 0x58006000

Software interrupt trigger register.

Bits set to *1* will trigger an interrupt. Interrupts trigger on write, but the value will persist in the register, allowing software to determine if a software interrupt was triggered by reading back the register.

Software is responsible for clearing the register to 0.

Repeated *1* writes without clearing will still trigger an interrupt.

Fig. 34: IRQARRAY10_EV_SOFT

Field	Name	Description
[19:0]	TRIGGER	Writing a 1 to this bit triggers the function.

IRQARRAY10_EV_STATUS

Address: 0x58006000 + 0x4 = 0x58006004

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

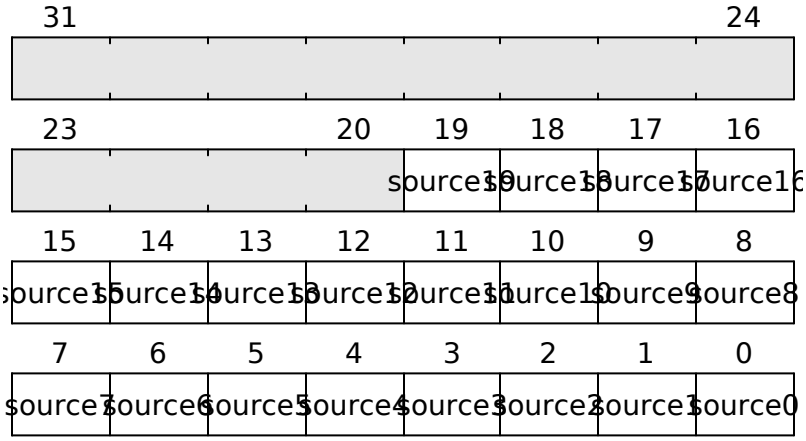

Fig. 35: IRQARRAY10_EV_STATUS

Field	Name	Description
[0]	SOURCE0	Level of the source0 event
[1]	SOURCE1	Level of the source1 event
[2]	SOURCE2	Level of the source2 event
[3]	SOURCE3	Level of the source3 event
[4]	SOURCE4	Level of the source4 event
[5]	SOURCE5	Level of the source5 event
[6]	SOURCE6	Level of the source6 event
[7]	SOURCE7	Level of the source7 event
[8]	SOURCE8	Level of the source8 event
[9]	SOURCE9	Level of the source9 event
[10]	SOURCE10	Level of the source10 event
[11]	SOURCE11	Level of the source11 event
[12]	SOURCE12	Level of the source12 event
[13]	SOURCE13	Level of the source13 event
[14]	SOURCE14	Level of the source14 event
[15]	SOURCE15	Level of the source15 event
[16]	SOURCE16	Level of the source16 event
[17]	SOURCE17	Level of the source17 event
[18]	SOURCE18	Level of the source18 event
[19]	SOURCE19	Level of the source19 event

IRQARRAY10_EV_PENDING

Address: 0x58006000 + 0x8 = 0x58006008

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

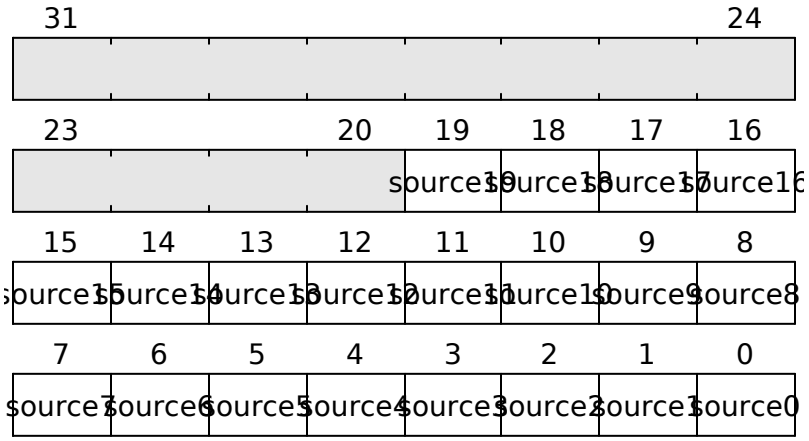

Fig. 36: IRQARRAY10_EV_PENDING

Field	Name	Description
[0]	SOURCE0	*1* when a source0 event occurs. This event uses an *EventSourceFlex* form of triggering
[1]	SOURCE1	*1* when a source1 event occurs. This event uses an *EventSourceFlex* form of triggering
[2]	SOURCE2	*1* when a source2 event occurs. This event uses an *EventSourceFlex* form of triggering
[3]	SOURCE3	*1* when a source3 event occurs. This event uses an *EventSourceFlex* form of triggering
[4]	SOURCE4	*1* when a source4 event occurs. This event uses an *EventSourceFlex* form of triggering
[5]	SOURCE5	*1* when a source5 event occurs. This event uses an *EventSourceFlex* form of triggering
[6]	SOURCE6	*1* when a source6 event occurs. This event uses an *EventSourceFlex* form of triggering
[7]	SOURCE7	*1* when a source7 event occurs. This event uses an *EventSourceFlex* form of triggering
[8]	SOURCE8	*1* when a source8 event occurs. This event uses an *EventSourceFlex* form of triggering
[9]	SOURCE9	*1* when a source9 event occurs. This event uses an *EventSourceFlex* form of triggering
[10]	SOURCE10	*1* when a source10 event occurs. This event uses an *EventSourceFlex* form of triggering
[11]	SOURCE11	*1* when a source11 event occurs. This event uses an *EventSourceFlex* form of triggering
[12]	SOURCE12	*1* when a source12 event occurs. This event uses an *EventSourceFlex* form of triggering
[13]	SOURCE13	*1* when a source13 event occurs. This event uses an *EventSourceFlex* form of triggering
[14]	SOURCE14	*1* when a source14 event occurs. This event uses an *EventSourceFlex* form of triggering
[15]	SOURCE15	*1* when a source15 event occurs. This event uses an *EventSourceFlex* form of triggering
[16]	SOURCE16	*1* when a source16 event occurs. This event uses an *EventSourceFlex* form of triggering
[17]	SOURCE17	*1* when a source17 event occurs. This event uses an *EventSourceFlex* form of triggering
[18]	SOURCE18	*1* when a source18 event occurs. This event uses an *EventSourceFlex* form of triggering
[19]	SOURCE19	*1* when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

IRQARRAY10_EV_ENABLE

Address: 0x58006000 + 0xc = 0x5800600c

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

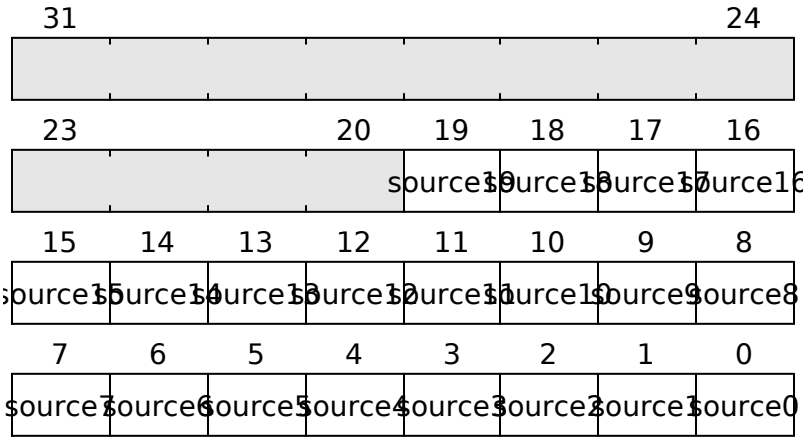

Fig. 37: IRQARRAY10_EV_ENABLE

Field	Name	Description
[0]	SOURCE0	Write a 1 to enable the `source0` Event
[1]	SOURCE1	Write a 1 to enable the `source1` Event
[2]	SOURCE2	Write a 1 to enable the `source2` Event
[3]	SOURCE3	Write a 1 to enable the `source3` Event
[4]	SOURCE4	Write a 1 to enable the `source4` Event
[5]	SOURCE5	Write a 1 to enable the `source5` Event
[6]	SOURCE6	Write a 1 to enable the `source6` Event
[7]	SOURCE7	Write a 1 to enable the `source7` Event
[8]	SOURCE8	Write a 1 to enable the `source8` Event
[9]	SOURCE9	Write a 1 to enable the `source9` Event
[10]	SOURCE10	Write a 1 to enable the `source10` Event
[11]	SOURCE11	Write a 1 to enable the `source11` Event
[12]	SOURCE12	Write a 1 to enable the `source12` Event
[13]	SOURCE13	Write a 1 to enable the `source13` Event
[14]	SOURCE14	Write a 1 to enable the `source14` Event
[15]	SOURCE15	Write a 1 to enable the `source15` Event
[16]	SOURCE16	Write a 1 to enable the `source16` Event
[17]	SOURCE17	Write a 1 to enable the `source17` Event
[18]	SOURCE18	Write a 1 to enable the `source18` Event
[19]	SOURCE19	Write a 1 to enable the `source19` Event

2.8 IRQARRAY11

IrqArray provides a large bank of interrupts for SoC integration. It is different from e.g. the NVIC or CLINT in that the register bank is structured along page boundaries, so that the interrupt handler CSRs can be owned by a specific virtual memory process, instead of bouncing through a common handler and forcing an inter-process message to be generated to route interrupts to their final destination.

The incoming interrupt signals are assumed to be synchronized to *aclk*.

Priorities are enforced entirely through software; the handler must read the *pending* bits and decide which ones should be handled first.

The *EventSource* is an *EventSourceFlex* which can handle pulses and levels, as well as software triggers.

The interrupt pending bit is latched when the trigger goes high, and stays high until software clears the event. The trigger takes precedence over clearing, so if the interrupt source is not cleared prior to clearing the interrupt pending bit, the interrupt will trigger again.

status reflects the instantaneous value of the trigger.

A separate input line is provided so that software can induce an interrupt by writing to a soft-trigger bit.

2.8.1 Register Listing for IRQARRAY11

Register	Address
IRQARRAY11_EV_SOFT	*0x58007000*
IRQARRAY11_EV_STATUS	*0x58007004*
IRQARRAY11_EV_PENDING	*0x58007008*
IRQARRAY11_EV_ENABLE	*0x5800700c*

IRQARRAY11_EV_SOFT

Address: 0x58007000 + 0x0 = 0x58007000

Software interrupt trigger register.

Bits set to *1* will trigger an interrupt. Interrupts trigger on write, but the value will persist in the register, allowing software to determine if a software interrupt was triggered by reading back the register.

Software is responsible for clearing the register to 0.

Repeated *1* writes without clearing will still trigger an interrupt.

Fig. 38: IRQARRAY11_EV_SOFT

Field	Name	Description
[19:0]	TRIGGER	Writing a 1 to this bit triggers the function.

IRQARRAY11_EV_STATUS

Address: 0x58007000 + 0x4 = 0x58007004

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

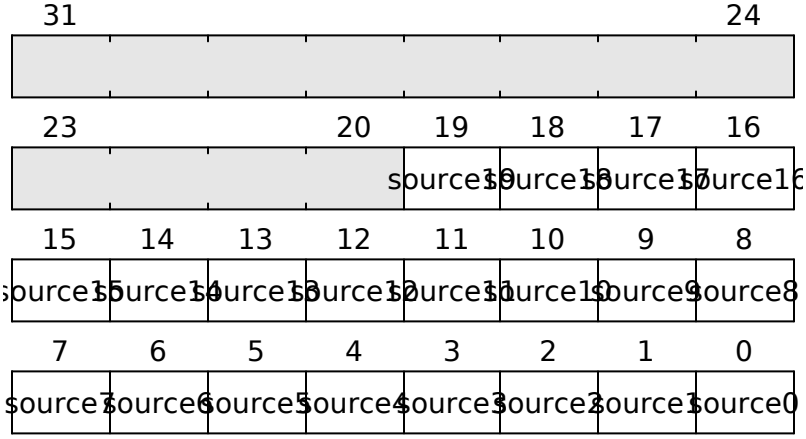

Fig. 39: IRQARRAY11_EV_STATUS

Field	Name	Description
[0]	SOURCE0	Level of the source0 event
[1]	SOURCE1	Level of the source1 event
[2]	SOURCE2	Level of the source2 event
[3]	SOURCE3	Level of the source3 event
[4]	SOURCE4	Level of the source4 event
[5]	SOURCE5	Level of the source5 event
[6]	SOURCE6	Level of the source6 event
[7]	SOURCE7	Level of the source7 event
[8]	SOURCE8	Level of the source8 event
[9]	SOURCE9	Level of the source9 event
[10]	SOURCE10	Level of the source10 event
[11]	SOURCE11	Level of the source11 event
[12]	SOURCE12	Level of the source12 event
[13]	SOURCE13	Level of the source13 event
[14]	SOURCE14	Level of the source14 event
[15]	SOURCE15	Level of the source15 event
[16]	SOURCE16	Level of the source16 event
[17]	SOURCE17	Level of the source17 event
[18]	SOURCE18	Level of the source18 event
[19]	SOURCE19	Level of the source19 event

IRQARRAY11_EV_PENDING

Address: 0x58007000 + 0x8 = 0x58007008

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

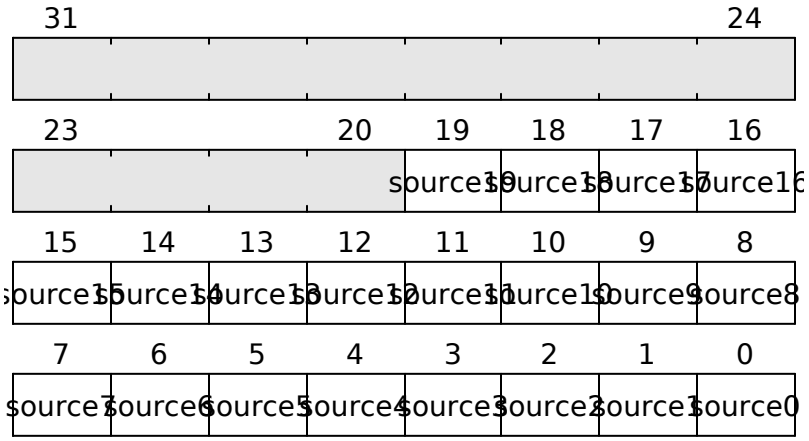

Fig. 40: IRQARRAY11_EV_PENDING

Field	Name	Description
[0]	SOURCE0	*1* when a source0 event occurs. This event uses an *EventSourceFlex* form of triggering
[1]	SOURCE1	*1* when a source1 event occurs. This event uses an *EventSourceFlex* form of triggering
[2]	SOURCE2	*1* when a source2 event occurs. This event uses an *EventSourceFlex* form of triggering
[3]	SOURCE3	*1* when a source3 event occurs. This event uses an *EventSourceFlex* form of triggering
[4]	SOURCE4	*1* when a source4 event occurs. This event uses an *EventSourceFlex* form of triggering
[5]	SOURCE5	*1* when a source5 event occurs. This event uses an *EventSourceFlex* form of triggering
[6]	SOURCE6	*1* when a source6 event occurs. This event uses an *EventSourceFlex* form of triggering
[7]	SOURCE7	*1* when a source7 event occurs. This event uses an *EventSourceFlex* form of triggering
[8]	SOURCE8	*1* when a source8 event occurs. This event uses an *EventSourceFlex* form of triggering
[9]	SOURCE9	*1* when a source9 event occurs. This event uses an *EventSourceFlex* form of triggering
[10]	SOURCE10	*1* when a source10 event occurs. This event uses an *EventSourceFlex* form of triggering
[11]	SOURCE11	*1* when a source11 event occurs. This event uses an *EventSourceFlex* form of triggering
[12]	SOURCE12	*1* when a source12 event occurs. This event uses an *EventSourceFlex* form of triggering
[13]	SOURCE13	*1* when a source13 event occurs. This event uses an *EventSourceFlex* form of triggering
[14]	SOURCE14	*1* when a source14 event occurs. This event uses an *EventSourceFlex* form of triggering
[15]	SOURCE15	*1* when a source15 event occurs. This event uses an *EventSourceFlex* form of triggering
[16]	SOURCE16	*1* when a source16 event occurs. This event uses an *EventSourceFlex* form of triggering
[17]	SOURCE17	*1* when a source17 event occurs. This event uses an *EventSourceFlex* form of triggering
[18]	SOURCE18	*1* when a source18 event occurs. This event uses an *EventSourceFlex* form of triggering
[19]	SOURCE19	*1* when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

IRQARRAY11_EV_ENABLE

Address: 0x58007000 + 0xc = 0x5800700c

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

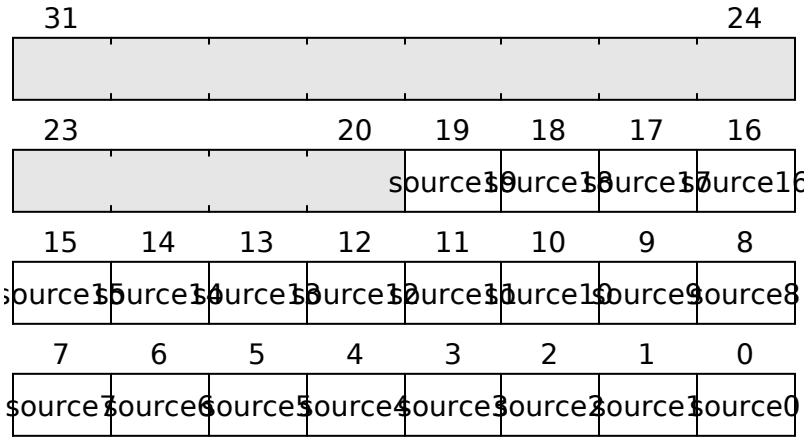

Fig. 41: IRQARRAY11_EV_ENABLE

Field	Name	Description
[0]	SOURCE0	Write a 1 to enable the `source0` Event
[1]	SOURCE1	Write a 1 to enable the `source1` Event
[2]	SOURCE2	Write a 1 to enable the `source2` Event
[3]	SOURCE3	Write a 1 to enable the `source3` Event
[4]	SOURCE4	Write a 1 to enable the `source4` Event
[5]	SOURCE5	Write a 1 to enable the `source5` Event
[6]	SOURCE6	Write a 1 to enable the `source6` Event
[7]	SOURCE7	Write a 1 to enable the `source7` Event
[8]	SOURCE8	Write a 1 to enable the `source8` Event
[9]	SOURCE9	Write a 1 to enable the `source9` Event
[10]	SOURCE10	Write a 1 to enable the `source10` Event
[11]	SOURCE11	Write a 1 to enable the `source11` Event
[12]	SOURCE12	Write a 1 to enable the `source12` Event
[13]	SOURCE13	Write a 1 to enable the `source13` Event
[14]	SOURCE14	Write a 1 to enable the `source14` Event
[15]	SOURCE15	Write a 1 to enable the `source15` Event
[16]	SOURCE16	Write a 1 to enable the `source16` Event
[17]	SOURCE17	Write a 1 to enable the `source17` Event
[18]	SOURCE18	Write a 1 to enable the `source18` Event
[19]	SOURCE19	Write a 1 to enable the `source19` Event

2.9 IRQARRAY12

IrqArray provides a large bank of interrupts for SoC integration. It is different from e.g. the NVIC or CLINT in that the register bank is structured along page boundaries, so that the interrupt handler CSRs can be owned by a specific virtual memory process, instead of bouncing through a common handler and forcing an inter-process message to be generated to route interrupts to their final destination.

The incoming interrupt signals are assumed to be synchronized to *aclk*.

Priorities are enforced entirely through software; the handler must read the *pending* bits and decide which ones should be handled first.

The *EventSource* is an *EventSourceFlex* which can handle pulses and levels, as well as software triggers.

The interrupt pending bit is latched when the trigger goes high, and stays high until software clears the event. The trigger takes precedence over clearing, so if the interrupt source is not cleared prior to clearing the interrupt pending bit, the interrupt will trigger again.

status reflects the instantaneous value of the trigger.

A separate input line is provided so that software can induce an interrupt by writing to a soft-trigger bit.

2.9.1 Register Listing for IRQARRAY12

Register	Address
IRQARRAY12_EV_SOFT	*0x58008000*
IRQARRAY12_EV_STATUS	*0x58008004*
IRQARRAY12_EV_PENDING	*0x58008008*
IRQARRAY12_EV_ENABLE	*0x5800800c*

IRQARRAY12_EV_SOFT

Address: 0x58008000 + 0x0 = 0x58008000

Software interrupt trigger register.

Bits set to *1* will trigger an interrupt. Interrupts trigger on write, but the value will persist in the register, allowing software to determine if a software interrupt was triggered by reading back the register.

Software is responsible for clearing the register to 0.

Repeated *1* writes without clearing will still trigger an interrupt.

Fig. 42: IRQARRAY12_EV_SOFT

Field	Name	Description
[19:0]	TRIGGER	Writing a 1 to this bit triggers the function.

IRQARRAY12_EV_STATUS

Address: 0x58008000 + 0x4 = 0x58008004

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

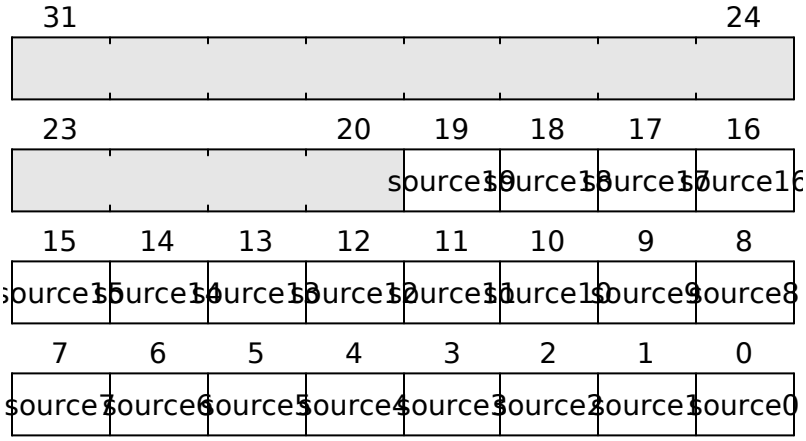

Fig. 43: IRQARRAY12_EV_STATUS

Field	Name	Description
[0]	SOURCE0	Level of the source0 event
[1]	SOURCE1	Level of the source1 event
[2]	SOURCE2	Level of the source2 event
[3]	SOURCE3	Level of the source3 event
[4]	SOURCE4	Level of the source4 event
[5]	SOURCE5	Level of the source5 event
[6]	SOURCE6	Level of the source6 event
[7]	SOURCE7	Level of the source7 event
[8]	SOURCE8	Level of the source8 event
[9]	SOURCE9	Level of the source9 event
[10]	SOURCE10	Level of the source10 event
[11]	SOURCE11	Level of the source11 event
[12]	SOURCE12	Level of the source12 event
[13]	SOURCE13	Level of the source13 event
[14]	SOURCE14	Level of the source14 event
[15]	SOURCE15	Level of the source15 event
[16]	SOURCE16	Level of the source16 event
[17]	SOURCE17	Level of the source17 event
[18]	SOURCE18	Level of the source18 event
[19]	SOURCE19	Level of the source19 event

IRQARRAY12_EV_PENDING

Address: 0x58008000 + 0x8 = 0x58008008

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

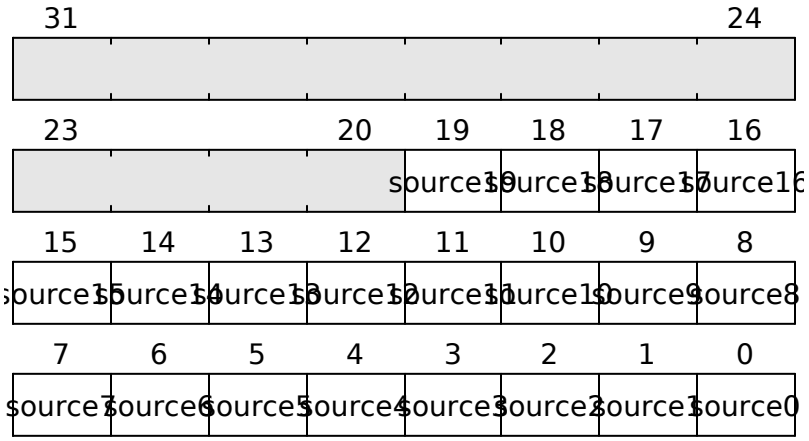

Fig. 44: IRQARRAY12_EV_PENDING

Field	Name	Description
[0]	SOURCE0	*1* when a source0 event occurs. This event uses an *EventSourceFlex* form of triggering
[1]	SOURCE1	*1* when a source1 event occurs. This event uses an *EventSourceFlex* form of triggering
[2]	SOURCE2	*1* when a source2 event occurs. This event uses an *EventSourceFlex* form of triggering
[3]	SOURCE3	*1* when a source3 event occurs. This event uses an *EventSourceFlex* form of triggering
[4]	SOURCE4	*1* when a source4 event occurs. This event uses an *EventSourceFlex* form of triggering
[5]	SOURCE5	*1* when a source5 event occurs. This event uses an *EventSourceFlex* form of triggering
[6]	SOURCE6	*1* when a source6 event occurs. This event uses an *EventSourceFlex* form of triggering
[7]	SOURCE7	*1* when a source7 event occurs. This event uses an *EventSourceFlex* form of triggering
[8]	SOURCE8	*1* when a source8 event occurs. This event uses an *EventSourceFlex* form of triggering
[9]	SOURCE9	*1* when a source9 event occurs. This event uses an *EventSourceFlex* form of triggering
[10]	SOURCE10	*1* when a source10 event occurs. This event uses an *EventSourceFlex* form of triggering
[11]	SOURCE11	*1* when a source11 event occurs. This event uses an *EventSourceFlex* form of triggering
[12]	SOURCE12	*1* when a source12 event occurs. This event uses an *EventSourceFlex* form of triggering
[13]	SOURCE13	*1* when a source13 event occurs. This event uses an *EventSourceFlex* form of triggering
[14]	SOURCE14	*1* when a source14 event occurs. This event uses an *EventSourceFlex* form of triggering
[15]	SOURCE15	*1* when a source15 event occurs. This event uses an *EventSourceFlex* form of triggering
[16]	SOURCE16	*1* when a source16 event occurs. This event uses an *EventSourceFlex* form of triggering
[17]	SOURCE17	*1* when a source17 event occurs. This event uses an *EventSourceFlex* form of triggering
[18]	SOURCE18	*1* when a source18 event occurs. This event uses an *EventSourceFlex* form of triggering
[19]	SOURCE19	*1* when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

IRQARRAY12_EV_ENABLE

Address: 0x58008000 + 0xc = 0x5800800c

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

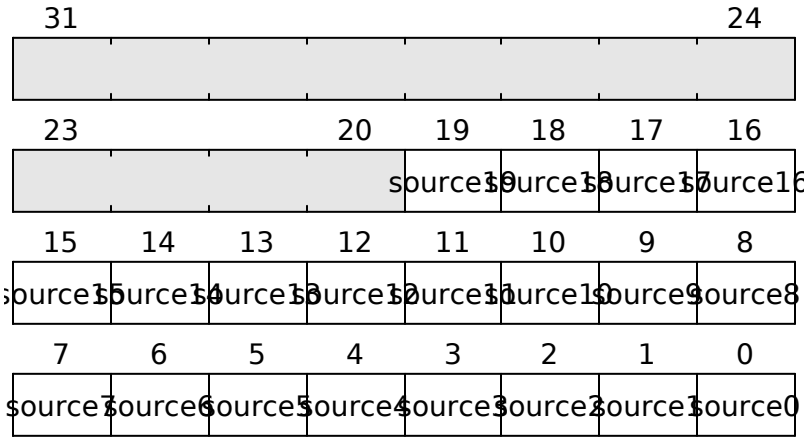

Fig. 45: IRQARRAY12_EV_ENABLE

Field	Name	Description
[0]	SOURCE0	Write a 1 to enable the `source0` Event
[1]	SOURCE1	Write a 1 to enable the `source1` Event
[2]	SOURCE2	Write a 1 to enable the `source2` Event
[3]	SOURCE3	Write a 1 to enable the `source3` Event
[4]	SOURCE4	Write a 1 to enable the `source4` Event
[5]	SOURCE5	Write a 1 to enable the `source5` Event
[6]	SOURCE6	Write a 1 to enable the `source6` Event
[7]	SOURCE7	Write a 1 to enable the `source7` Event
[8]	SOURCE8	Write a 1 to enable the `source8` Event
[9]	SOURCE9	Write a 1 to enable the `source9` Event
[10]	SOURCE10	Write a 1 to enable the `source10` Event
[11]	SOURCE11	Write a 1 to enable the `source11` Event
[12]	SOURCE12	Write a 1 to enable the `source12` Event
[13]	SOURCE13	Write a 1 to enable the `source13` Event
[14]	SOURCE14	Write a 1 to enable the `source14` Event
[15]	SOURCE15	Write a 1 to enable the `source15` Event
[16]	SOURCE16	Write a 1 to enable the `source16` Event
[17]	SOURCE17	Write a 1 to enable the `source17` Event
[18]	SOURCE18	Write a 1 to enable the `source18` Event
[19]	SOURCE19	Write a 1 to enable the `source19` Event

2.10 IRQARRAY13

IrqArray provides a large bank of interrupts for SoC integration. It is different from e.g. the NVIC or CLINT in that the register bank is structured along page boundaries, so that the interrupt handler CSRs can be owned by a specific virtual memory process, instead of bouncing through a common handler and forcing an inter-process message to be generated to route interrupts to their final destination.

The incoming interrupt signals are assumed to be synchronized to *aclk*.

Priorities are enforced entirely through software; the handler must read the *pending* bits and decide which ones should be handled first.

The *EventSource* is an *EventSourceFlex* which can handle pulses and levels, as well as software triggers.

The interrupt pending bit is latched when the trigger goes high, and stays high until software clears the event. The trigger takes precedence over clearing, so if the interrupt source is not cleared prior to clearing the interrupt pending bit, the interrupt will trigger again.

status reflects the instantaneous value of the trigger.

A separate input line is provided so that software can induce an interrupt by writing to a soft-trigger bit.

2.10.1 Register Listing for IRQARRAY13

Register	Address
IRQARRAY13_EV_SOFT	*0x58009000*
IRQARRAY13_EV_STATUS	*0x58009004*
IRQARRAY13_EV_PENDING	*0x58009008*
IRQARRAY13_EV_ENABLE	*0x5800900c*

IRQARRAY13_EV_SOFT

Address: 0x58009000 + 0x0 = 0x58009000

Software interrupt trigger register.

Bits set to *1* will trigger an interrupt. Interrupts trigger on write, but the value will persist in the register, allowing software to determine if a software interrupt was triggered by reading back the register.

Software is responsible for clearing the register to 0.

Repeated *1* writes without clearing will still trigger an interrupt.

Fig. 46: IRQARRAY13_EV_SOFT

Field	Name	Description
[19:0]	TRIGGER	Writing a 1 to this bit triggers the function.

IRQARRAY13_EV_STATUS

Address: 0x58009000 + 0x4 = 0x58009004

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

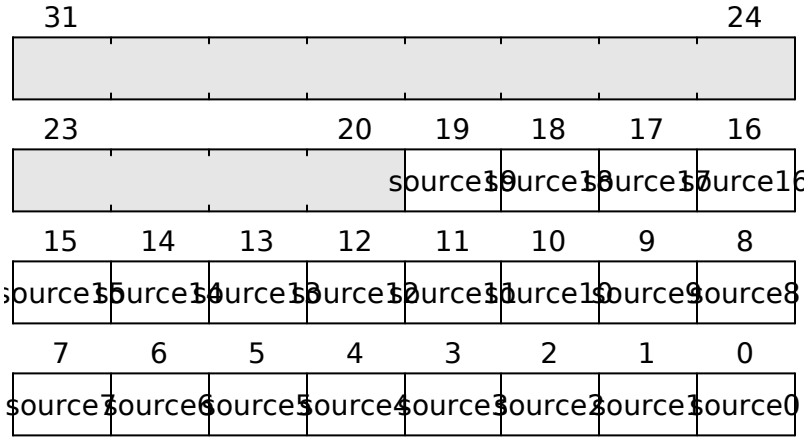

Fig. 47: IRQARRAY13_EV_STATUS

Field	Name	Description
[0]	SOURCE0	Level of the source0 event
[1]	SOURCE1	Level of the source1 event
[2]	SOURCE2	Level of the source2 event
[3]	SOURCE3	Level of the source3 event
[4]	SOURCE4	Level of the source4 event
[5]	SOURCE5	Level of the source5 event
[6]	SOURCE6	Level of the source6 event
[7]	SOURCE7	Level of the source7 event
[8]	SOURCE8	Level of the source8 event
[9]	SOURCE9	Level of the source9 event
[10]	SOURCE10	Level of the source10 event
[11]	SOURCE11	Level of the source11 event
[12]	SOURCE12	Level of the source12 event
[13]	SOURCE13	Level of the source13 event
[14]	SOURCE14	Level of the source14 event
[15]	SOURCE15	Level of the source15 event
[16]	SOURCE16	Level of the source16 event
[17]	SOURCE17	Level of the source17 event
[18]	SOURCE18	Level of the source18 event
[19]	SOURCE19	Level of the source19 event

IRQARRAY13_EV_PENDING

Address: 0x58009000 + 0x8 = 0x58009008

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

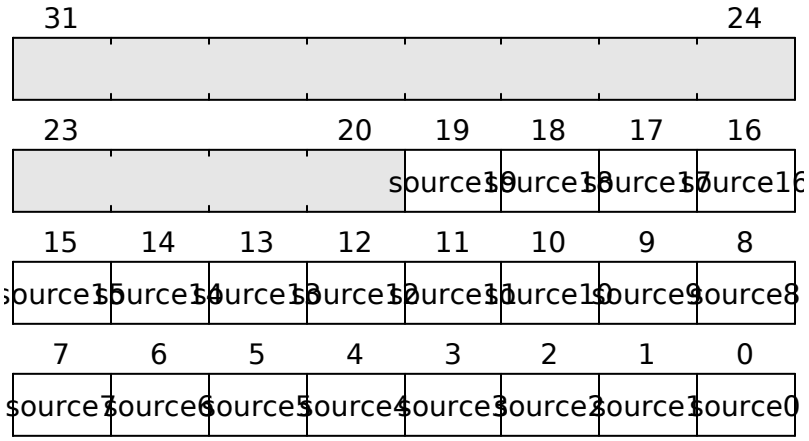

Fig. 48: IRQARRAY13_EV_PENDING

Field	Name	Description
[0]	SOURCE0	*1* when a source0 event occurs. This event uses an *EventSourceFlex* form of triggering
[1]	SOURCE1	*1* when a source1 event occurs. This event uses an *EventSourceFlex* form of triggering
[2]	SOURCE2	*1* when a source2 event occurs. This event uses an *EventSourceFlex* form of triggering
[3]	SOURCE3	*1* when a source3 event occurs. This event uses an *EventSourceFlex* form of triggering
[4]	SOURCE4	*1* when a source4 event occurs. This event uses an *EventSourceFlex* form of triggering
[5]	SOURCE5	*1* when a source5 event occurs. This event uses an *EventSourceFlex* form of triggering
[6]	SOURCE6	*1* when a source6 event occurs. This event uses an *EventSourceFlex* form of triggering
[7]	SOURCE7	*1* when a source7 event occurs. This event uses an *EventSourceFlex* form of triggering
[8]	SOURCE8	*1* when a source8 event occurs. This event uses an *EventSourceFlex* form of triggering
[9]	SOURCE9	*1* when a source9 event occurs. This event uses an *EventSourceFlex* form of triggering
[10]	SOURCE10	*1* when a source10 event occurs. This event uses an *EventSourceFlex* form of triggering
[11]	SOURCE11	*1* when a source11 event occurs. This event uses an *EventSourceFlex* form of triggering
[12]	SOURCE12	*1* when a source12 event occurs. This event uses an *EventSourceFlex* form of triggering
[13]	SOURCE13	*1* when a source13 event occurs. This event uses an *EventSourceFlex* form of triggering
[14]	SOURCE14	*1* when a source14 event occurs. This event uses an *EventSourceFlex* form of triggering
[15]	SOURCE15	*1* when a source15 event occurs. This event uses an *EventSourceFlex* form of triggering
[16]	SOURCE16	*1* when a source16 event occurs. This event uses an *EventSourceFlex* form of triggering
[17]	SOURCE17	*1* when a source17 event occurs. This event uses an *EventSourceFlex* form of triggering
[18]	SOURCE18	*1* when a source18 event occurs. This event uses an *EventSourceFlex* form of triggering
[19]	SOURCE19	*1* when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

IRQARRAY13_EV_ENABLE

Address: 0x58009000 + 0xc = 0x5800900c

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

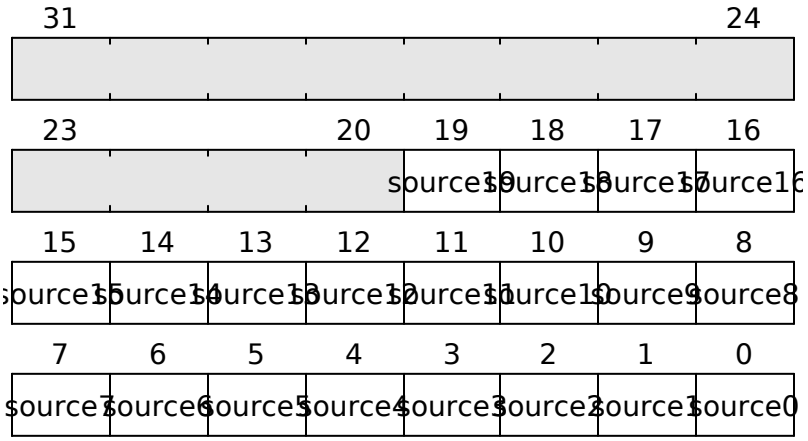

Fig. 49: IRQARRAY13_EV_ENABLE

Field	Name	Description
[0]	SOURCE0	Write a 1 to enable the `source0` Event
[1]	SOURCE1	Write a 1 to enable the `source1` Event
[2]	SOURCE2	Write a 1 to enable the `source2` Event
[3]	SOURCE3	Write a 1 to enable the `source3` Event
[4]	SOURCE4	Write a 1 to enable the `source4` Event
[5]	SOURCE5	Write a 1 to enable the `source5` Event
[6]	SOURCE6	Write a 1 to enable the `source6` Event
[7]	SOURCE7	Write a 1 to enable the `source7` Event
[8]	SOURCE8	Write a 1 to enable the `source8` Event
[9]	SOURCE9	Write a 1 to enable the `source9` Event
[10]	SOURCE10	Write a 1 to enable the `source10` Event
[11]	SOURCE11	Write a 1 to enable the `source11` Event
[12]	SOURCE12	Write a 1 to enable the `source12` Event
[13]	SOURCE13	Write a 1 to enable the `source13` Event
[14]	SOURCE14	Write a 1 to enable the `source14` Event
[15]	SOURCE15	Write a 1 to enable the `source15` Event
[16]	SOURCE16	Write a 1 to enable the `source16` Event
[17]	SOURCE17	Write a 1 to enable the `source17` Event
[18]	SOURCE18	Write a 1 to enable the `source18` Event
[19]	SOURCE19	Write a 1 to enable the `source19` Event

2.11 IRQARRAY14

IrqArray provides a large bank of interrupts for SoC integration. It is different from e.g. the NVIC or CLINT in that the register bank is structured along page boundaries, so that the interrupt handler CSRs can be owned by a specific virtual memory process, instead of bouncing through a common handler and forcing an inter-process message to be generated to route interrupts to their final destination.

The incoming interrupt signals are assumed to be synchronized to *aclk*.

Priorities are enforced entirely through software; the handler must read the *pending* bits and decide which ones should be handled first.

The *EventSource* is an *EventSourceFlex* which can handle pulses and levels, as well as software triggers.

The interrupt pending bit is latched when the trigger goes high, and stays high until software clears the event. The trigger takes precedence over clearing, so if the interrupt source is not cleared prior to clearing the interrupt pending bit, the interrupt will trigger again.

status reflects the instantaneous value of the trigger.

A separate input line is provided so that software can induce an interrupt by writing to a soft-trigger bit.

2.11.1 Register Listing for IRQARRAY14

Register	Address
IRQARRAY14_EV_SOFT	*0x5800a000*
IRQARRAY14_EV_STATUS	*0x5800a004*
IRQARRAY14_EV_PENDING	*0x5800a008*
IRQARRAY14_EV_ENABLE	*0x5800a00c*

IRQARRAY14_EV_SOFT

Address: 0x5800a000 + 0x0 = 0x5800a000

Software interrupt trigger register.

Bits set to *1* will trigger an interrupt. Interrupts trigger on write, but the value will persist in the register, allowing software to determine if a software interrupt was triggered by reading back the register.

Software is responsible for clearing the register to 0.

Repeated *1* writes without clearing will still trigger an interrupt.

Fig. 50: IRQARRAY14_EV_SOFT

Field	Name	Description
[19:0]	TRIGGER	Writing a 1 to this bit triggers the function.

IRQARRAY14_EV_STATUS

Address: 0x5800a000 + 0x4 = 0x5800a004

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

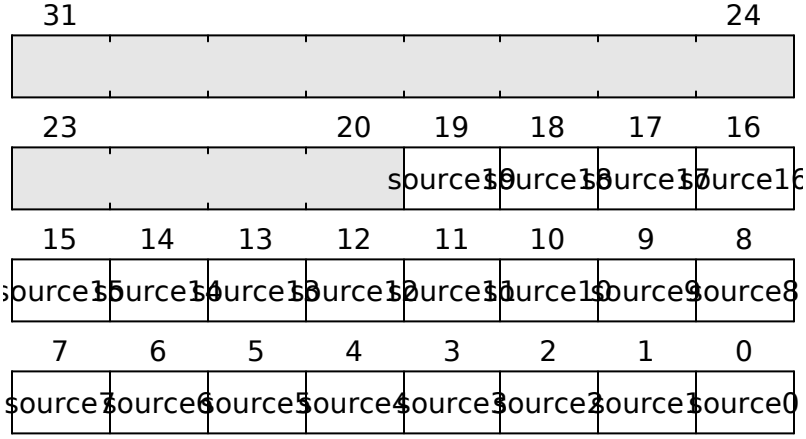

Fig. 51: IRQARRAY14_EV_STATUS

Field	Name	Description
[0]	SOURCE0	Level of the source0 event
[1]	SOURCE1	Level of the source1 event
[2]	SOURCE2	Level of the source2 event
[3]	SOURCE3	Level of the source3 event
[4]	SOURCE4	Level of the source4 event
[5]	SOURCE5	Level of the source5 event
[6]	SOURCE6	Level of the source6 event
[7]	SOURCE7	Level of the source7 event
[8]	SOURCE8	Level of the source8 event
[9]	SOURCE9	Level of the source9 event
[10]	SOURCE10	Level of the source10 event
[11]	SOURCE11	Level of the source11 event
[12]	SOURCE12	Level of the source12 event
[13]	SOURCE13	Level of the source13 event
[14]	SOURCE14	Level of the source14 event
[15]	SOURCE15	Level of the source15 event
[16]	SOURCE16	Level of the source16 event
[17]	SOURCE17	Level of the source17 event
[18]	SOURCE18	Level of the source18 event
[19]	SOURCE19	Level of the source19 event

IRQARRAY14_EV_PENDING

Address: 0x5800a000 + 0x8 = 0x5800a008

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

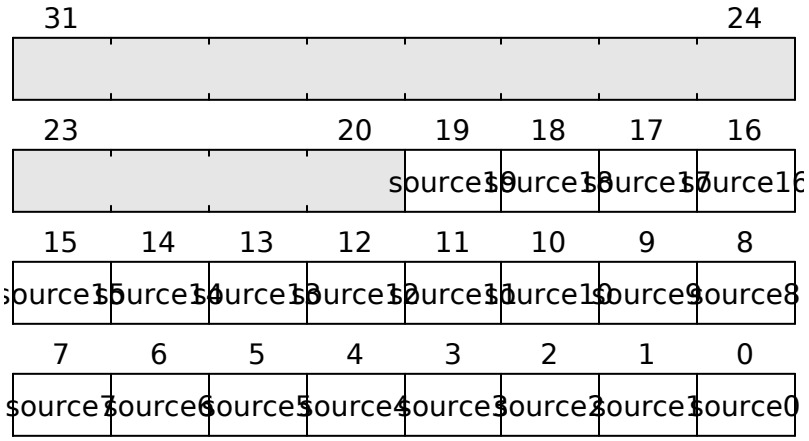

Fig. 52: IRQARRAY14_EV_PENDING

Field	Name	Description
[0]	SOURCE0	*1* when a source0 event occurs. This event uses an *EventSourceFlex* form of triggering
[1]	SOURCE1	*1* when a source1 event occurs. This event uses an *EventSourceFlex* form of triggering
[2]	SOURCE2	*1* when a source2 event occurs. This event uses an *EventSourceFlex* form of triggering
[3]	SOURCE3	*1* when a source3 event occurs. This event uses an *EventSourceFlex* form of triggering
[4]	SOURCE4	*1* when a source4 event occurs. This event uses an *EventSourceFlex* form of triggering
[5]	SOURCE5	*1* when a source5 event occurs. This event uses an *EventSourceFlex* form of triggering
[6]	SOURCE6	*1* when a source6 event occurs. This event uses an *EventSourceFlex* form of triggering
[7]	SOURCE7	*1* when a source7 event occurs. This event uses an *EventSourceFlex* form of triggering
[8]	SOURCE8	*1* when a source8 event occurs. This event uses an *EventSourceFlex* form of triggering
[9]	SOURCE9	*1* when a source9 event occurs. This event uses an *EventSourceFlex* form of triggering
[10]	SOURCE10	*1* when a source10 event occurs. This event uses an *EventSourceFlex* form of triggering
[11]	SOURCE11	*1* when a source11 event occurs. This event uses an *EventSourceFlex* form of triggering
[12]	SOURCE12	*1* when a source12 event occurs. This event uses an *EventSourceFlex* form of triggering
[13]	SOURCE13	*1* when a source13 event occurs. This event uses an *EventSourceFlex* form of triggering
[14]	SOURCE14	*1* when a source14 event occurs. This event uses an *EventSourceFlex* form of triggering
[15]	SOURCE15	*1* when a source15 event occurs. This event uses an *EventSourceFlex* form of triggering
[16]	SOURCE16	*1* when a source16 event occurs. This event uses an *EventSourceFlex* form of triggering
[17]	SOURCE17	*1* when a source17 event occurs. This event uses an *EventSourceFlex* form of triggering
[18]	SOURCE18	*1* when a source18 event occurs. This event uses an *EventSourceFlex* form of triggering
[19]	SOURCE19	*1* when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

IRQARRAY14_EV_ENABLE

Address: 0x5800a000 + 0xc = 0x5800a00c

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

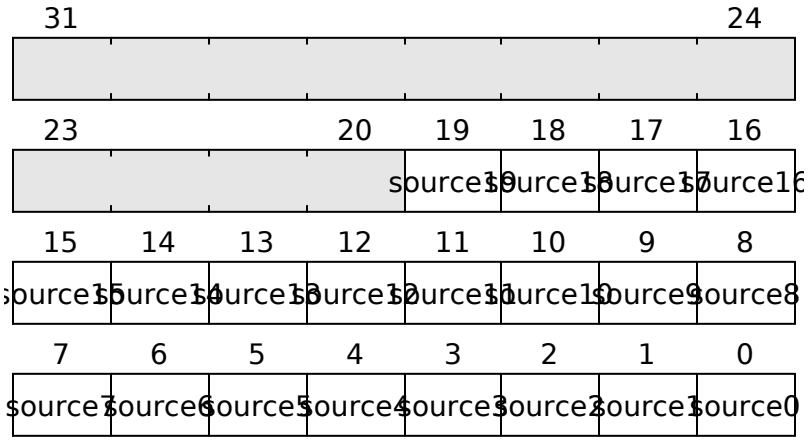

Fig. 53: IRQARRAY14_EV_ENABLE

Field	Name	Description
[0]	SOURCE0	Write a 1 to enable the `source0` Event
[1]	SOURCE1	Write a 1 to enable the `source1` Event
[2]	SOURCE2	Write a 1 to enable the `source2` Event
[3]	SOURCE3	Write a 1 to enable the `source3` Event
[4]	SOURCE4	Write a 1 to enable the `source4` Event
[5]	SOURCE5	Write a 1 to enable the `source5` Event
[6]	SOURCE6	Write a 1 to enable the `source6` Event
[7]	SOURCE7	Write a 1 to enable the `source7` Event
[8]	SOURCE8	Write a 1 to enable the `source8` Event
[9]	SOURCE9	Write a 1 to enable the `source9` Event
[10]	SOURCE10	Write a 1 to enable the `source10` Event
[11]	SOURCE11	Write a 1 to enable the `source11` Event
[12]	SOURCE12	Write a 1 to enable the `source12` Event
[13]	SOURCE13	Write a 1 to enable the `source13` Event
[14]	SOURCE14	Write a 1 to enable the `source14` Event
[15]	SOURCE15	Write a 1 to enable the `source15` Event
[16]	SOURCE16	Write a 1 to enable the `source16` Event
[17]	SOURCE17	Write a 1 to enable the `source17` Event
[18]	SOURCE18	Write a 1 to enable the `source18` Event
[19]	SOURCE19	Write a 1 to enable the `source19` Event

2.12 IRQARRAY15

IrqArray provides a large bank of interrupts for SoC integration. It is different from e.g. the NVIC or CLINT in that the register bank is structured along page boundaries, so that the interrupt handler CSRs can be owned by a specific virtual memory process, instead of bouncing through a common handler and forcing an inter-process message to be generated to route interrupts to their final destination.

The incoming interrupt signals are assumed to be synchronized to *aclk*.

Priorities are enforced entirely through software; the handler must read the *pending* bits and decide which ones should be handled first.

The *EventSource* is an *EventSourceFlex* which can handle pulses and levels, as well as software triggers.

The interrupt pending bit is latched when the trigger goes high, and stays high until software clears the event. The trigger takes precedence over clearing, so if the interrupt source is not cleared prior to clearing the interrupt pending bit, the interrupt will trigger again.

status reflects the instantaneous value of the trigger.

A separate input line is provided so that software can induce an interrupt by writing to a soft-trigger bit.

2.12.1 Register Listing for IRQARRAY15

Register	Address
IRQARRAY15_EV_SOFT	*0x5800b000*
IRQARRAY15_EV_STATUS	*0x5800b004*
IRQARRAY15_EV_PENDING	*0x5800b008*
IRQARRAY15_EV_ENABLE	*0x5800b00c*

IRQARRAY15_EV_SOFT

Address: 0x5800b000 + 0x0 = 0x5800b000

Software interrupt trigger register.

Bits set to *1* will trigger an interrupt. Interrupts trigger on write, but the value will persist in the register, allowing software to determine if a software interrupt was triggered by reading back the register.

Software is responsible for clearing the register to 0.

Repeated *1* writes without clearing will still trigger an interrupt.

Fig. 54: IRQARRAY15_EV_SOFT

Field	Name	Description
[19:0]	TRIGGER	Writing a 1 to this bit triggers the function.

IRQARRAY15_EV_STATUS

Address: 0x5800b000 + 0x4 = 0x5800b004

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

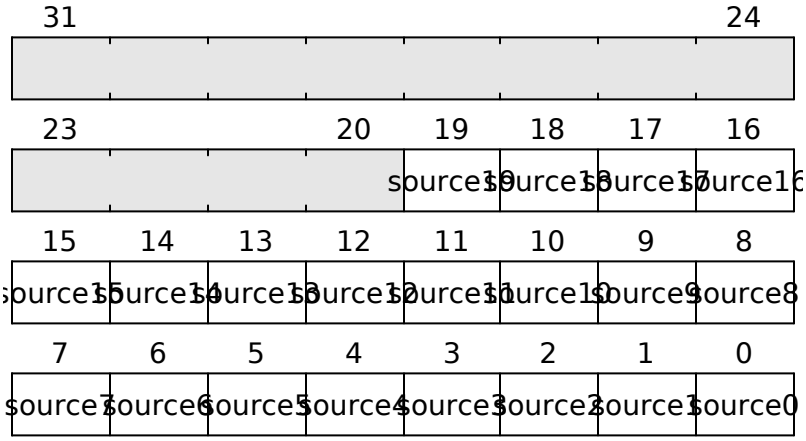

Fig. 55: IRQARRAY15_EV_STATUS

Field	Name	Description
[0]	SOURCE0	Level of the source0 event
[1]	SOURCE1	Level of the source1 event
[2]	SOURCE2	Level of the source2 event
[3]	SOURCE3	Level of the source3 event
[4]	SOURCE4	Level of the source4 event
[5]	SOURCE5	Level of the source5 event
[6]	SOURCE6	Level of the source6 event
[7]	SOURCE7	Level of the source7 event
[8]	SOURCE8	Level of the source8 event
[9]	SOURCE9	Level of the source9 event
[10]	SOURCE10	Level of the source10 event
[11]	SOURCE11	Level of the source11 event
[12]	SOURCE12	Level of the source12 event
[13]	SOURCE13	Level of the source13 event
[14]	SOURCE14	Level of the source14 event
[15]	SOURCE15	Level of the source15 event
[16]	SOURCE16	Level of the source16 event
[17]	SOURCE17	Level of the source17 event
[18]	SOURCE18	Level of the source18 event
[19]	SOURCE19	Level of the source19 event

IRQARRAY15_EV_PENDING

Address: 0x5800b000 + 0x8 = 0x5800b008

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

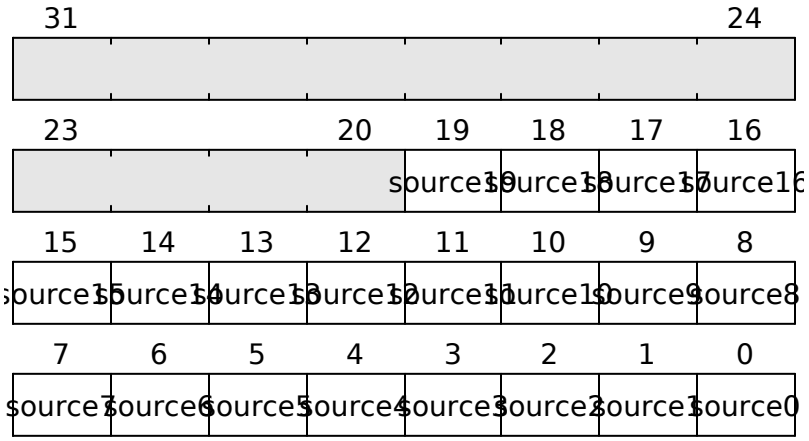

Fig. 56: IRQARRAY15_EV_PENDING

Field	Name	Description
[0]	SOURCE0	*1* when a source0 event occurs. This event uses an *EventSourceFlex* form of triggering
[1]	SOURCE1	*1* when a source1 event occurs. This event uses an *EventSourceFlex* form of triggering
[2]	SOURCE2	*1* when a source2 event occurs. This event uses an *EventSourceFlex* form of triggering
[3]	SOURCE3	*1* when a source3 event occurs. This event uses an *EventSourceFlex* form of triggering
[4]	SOURCE4	*1* when a source4 event occurs. This event uses an *EventSourceFlex* form of triggering
[5]	SOURCE5	*1* when a source5 event occurs. This event uses an *EventSourceFlex* form of triggering
[6]	SOURCE6	*1* when a source6 event occurs. This event uses an *EventSourceFlex* form of triggering
[7]	SOURCE7	*1* when a source7 event occurs. This event uses an *EventSourceFlex* form of triggering
[8]	SOURCE8	*1* when a source8 event occurs. This event uses an *EventSourceFlex* form of triggering
[9]	SOURCE9	*1* when a source9 event occurs. This event uses an *EventSourceFlex* form of triggering
[10]	SOURCE10	*1* when a source10 event occurs. This event uses an *EventSourceFlex* form of triggering
[11]	SOURCE11	*1* when a source11 event occurs. This event uses an *EventSourceFlex* form of triggering
[12]	SOURCE12	*1* when a source12 event occurs. This event uses an *EventSourceFlex* form of triggering
[13]	SOURCE13	*1* when a source13 event occurs. This event uses an *EventSourceFlex* form of triggering
[14]	SOURCE14	*1* when a source14 event occurs. This event uses an *EventSourceFlex* form of triggering
[15]	SOURCE15	*1* when a source15 event occurs. This event uses an *EventSourceFlex* form of triggering
[16]	SOURCE16	*1* when a source16 event occurs. This event uses an *EventSourceFlex* form of triggering
[17]	SOURCE17	*1* when a source17 event occurs. This event uses an *EventSourceFlex* form of triggering
[18]	SOURCE18	*1* when a source18 event occurs. This event uses an *EventSourceFlex* form of triggering
[19]	SOURCE19	*1* when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

IRQARRAY15_EV_ENABLE

Address: 0x5800b000 + 0xc = 0x5800b00c

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

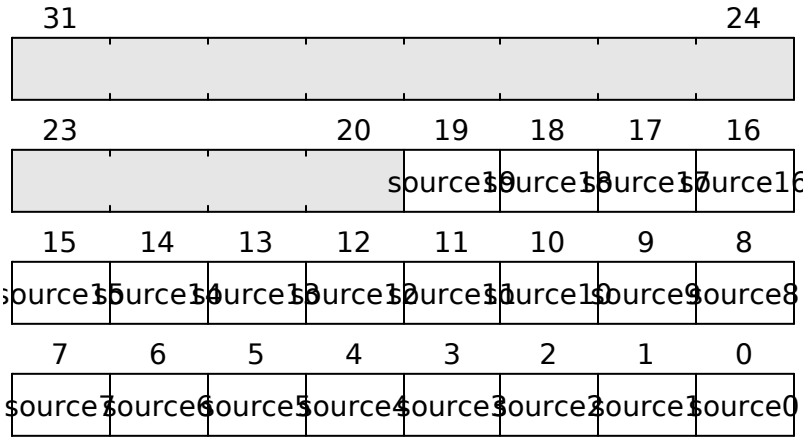

Fig. 57: IRQARRAY15_EV_ENABLE

Field	Name	Description
[0]	SOURCE0	Write a 1 to enable the `source0` Event
[1]	SOURCE1	Write a 1 to enable the `source1` Event
[2]	SOURCE2	Write a 1 to enable the `source2` Event
[3]	SOURCE3	Write a 1 to enable the `source3` Event
[4]	SOURCE4	Write a 1 to enable the `source4` Event
[5]	SOURCE5	Write a 1 to enable the `source5` Event
[6]	SOURCE6	Write a 1 to enable the `source6` Event
[7]	SOURCE7	Write a 1 to enable the `source7` Event
[8]	SOURCE8	Write a 1 to enable the `source8` Event
[9]	SOURCE9	Write a 1 to enable the `source9` Event
[10]	SOURCE10	Write a 1 to enable the `source10` Event
[11]	SOURCE11	Write a 1 to enable the `source11` Event
[12]	SOURCE12	Write a 1 to enable the `source12` Event
[13]	SOURCE13	Write a 1 to enable the `source13` Event
[14]	SOURCE14	Write a 1 to enable the `source14` Event
[15]	SOURCE15	Write a 1 to enable the `source15` Event
[16]	SOURCE16	Write a 1 to enable the `source16` Event
[17]	SOURCE17	Write a 1 to enable the `source17` Event
[18]	SOURCE18	Write a 1 to enable the `source18` Event
[19]	SOURCE19	Write a 1 to enable the `source19` Event

2.13 IRQARRAY16

IrqArray provides a large bank of interrupts for SoC integration. It is different from e.g. the NVIC or CLINT in that the register bank is structured along page boundaries, so that the interrupt handler CSRs can be owned by a specific virtual memory process, instead of bouncing through a common handler and forcing an inter-process message to be generated to route interrupts to their final destination.

The incoming interrupt signals are assumed to be synchronized to *aclk*.

Priorities are enforced entirely through software; the handler must read the *pending* bits and decide which ones should be handled first.

The *EventSource* is an *EventSourceFlex* which can handle pulses and levels, as well as software triggers.

The interrupt pending bit is latched when the trigger goes high, and stays high until software clears the event. The trigger takes precedence over clearing, so if the interrupt source is not cleared prior to clearing the interrupt pending bit, the interrupt will trigger again.

status reflects the instantaneous value of the trigger.

A separate input line is provided so that software can induce an interrupt by writing to a soft-trigger bit.

2.13.1 Register Listing for IRQARRAY16

Register	Address
IRQARRAY16_EV_SOFT	*0x5800c000*
IRQARRAY16_EV_STATUS	*0x5800c004*
IRQARRAY16_EV_PENDING	*0x5800c008*
IRQARRAY16_EV_ENABLE	*0x5800c00c*

IRQARRAY16_EV_SOFT

Address: 0x5800c000 + 0x0 = 0x5800c000

Software interrupt trigger register.

Bits set to *1* will trigger an interrupt. Interrupts trigger on write, but the value will persist in the register, allowing software to determine if a software interrupt was triggered by reading back the register.

Software is responsible for clearing the register to 0.

Repeated *1* writes without clearing will still trigger an interrupt.

Fig. 58: IRQARRAY16_EV_SOFT

Field	Name	Description
[19:0]	TRIGGER	Writing a 1 to this bit triggers the function.

IRQARRAY16_EV_STATUS

Address: 0x5800c000 + 0x4 = 0x5800c004

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

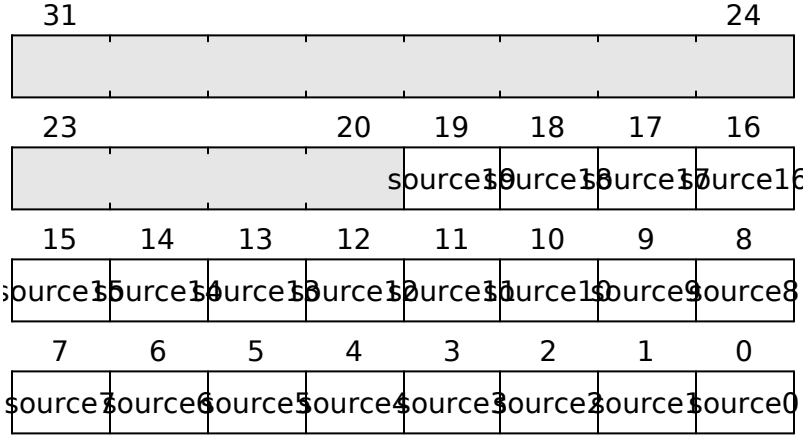

Fig. 59: IRQARRAY16_EV_STATUS

Field	Name	Description
[0]	SOURCE0	Level of the source0 event
[1]	SOURCE1	Level of the source1 event
[2]	SOURCE2	Level of the source2 event
[3]	SOURCE3	Level of the source3 event
[4]	SOURCE4	Level of the source4 event
[5]	SOURCE5	Level of the source5 event
[6]	SOURCE6	Level of the source6 event
[7]	SOURCE7	Level of the source7 event
[8]	SOURCE8	Level of the source8 event
[9]	SOURCE9	Level of the source9 event
[10]	SOURCE10	Level of the source10 event
[11]	SOURCE11	Level of the source11 event
[12]	SOURCE12	Level of the source12 event
[13]	SOURCE13	Level of the source13 event
[14]	SOURCE14	Level of the source14 event
[15]	SOURCE15	Level of the source15 event
[16]	SOURCE16	Level of the source16 event
[17]	SOURCE17	Level of the source17 event
[18]	SOURCE18	Level of the source18 event
[19]	SOURCE19	Level of the source19 event

IRQARRAY16_EV_PENDING

Address: 0x5800c000 + 0x8 = 0x5800c008

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

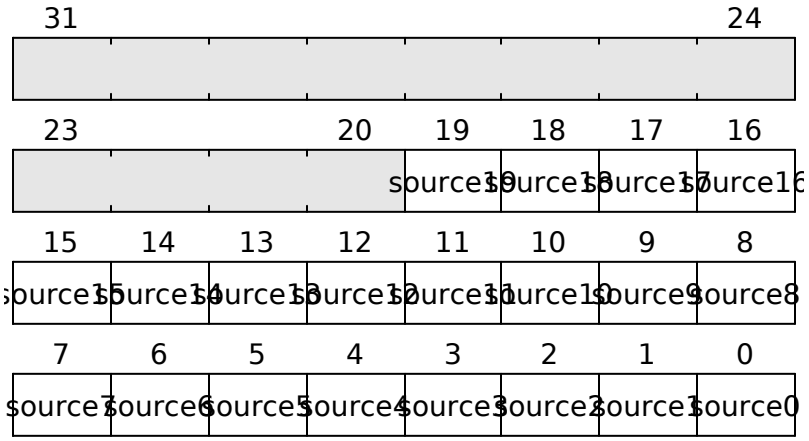

Fig. 60: IRQARRAY16_EV_PENDING

Field	Name	Description
[0]	SOURCE0	*1* when a source0 event occurs. This event uses an *EventSourceFlex* form of triggering
[1]	SOURCE1	*1* when a source1 event occurs. This event uses an *EventSourceFlex* form of triggering
[2]	SOURCE2	*1* when a source2 event occurs. This event uses an *EventSourceFlex* form of triggering
[3]	SOURCE3	*1* when a source3 event occurs. This event uses an *EventSourceFlex* form of triggering
[4]	SOURCE4	*1* when a source4 event occurs. This event uses an *EventSourceFlex* form of triggering
[5]	SOURCE5	*1* when a source5 event occurs. This event uses an *EventSourceFlex* form of triggering
[6]	SOURCE6	*1* when a source6 event occurs. This event uses an *EventSourceFlex* form of triggering
[7]	SOURCE7	*1* when a source7 event occurs. This event uses an *EventSourceFlex* form of triggering
[8]	SOURCE8	*1* when a source8 event occurs. This event uses an *EventSourceFlex* form of triggering
[9]	SOURCE9	*1* when a source9 event occurs. This event uses an *EventSourceFlex* form of triggering
[10]	SOURCE10	*1* when a source10 event occurs. This event uses an *EventSourceFlex* form of triggering
[11]	SOURCE11	*1* when a source11 event occurs. This event uses an *EventSourceFlex* form of triggering
[12]	SOURCE12	*1* when a source12 event occurs. This event uses an *EventSourceFlex* form of triggering
[13]	SOURCE13	*1* when a source13 event occurs. This event uses an *EventSourceFlex* form of triggering
[14]	SOURCE14	*1* when a source14 event occurs. This event uses an *EventSourceFlex* form of triggering
[15]	SOURCE15	*1* when a source15 event occurs. This event uses an *EventSourceFlex* form of triggering
[16]	SOURCE16	*1* when a source16 event occurs. This event uses an *EventSourceFlex* form of triggering
[17]	SOURCE17	*1* when a source17 event occurs. This event uses an *EventSourceFlex* form of triggering
[18]	SOURCE18	*1* when a source18 event occurs. This event uses an *EventSourceFlex* form of triggering
[19]	SOURCE19	*1* when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

IRQARRAY16_EV_ENABLE

Address: 0x5800c000 + 0xc = 0x5800c00c

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

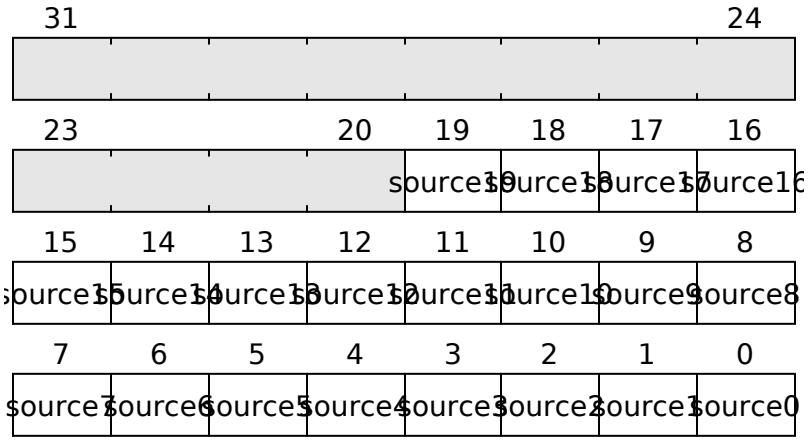

Fig. 61: IRQARRAY16_EV_ENABLE

Field	Name	Description
[0]	SOURCE0	Write a 1 to enable the `source0` Event
[1]	SOURCE1	Write a 1 to enable the `source1` Event
[2]	SOURCE2	Write a 1 to enable the `source2` Event
[3]	SOURCE3	Write a 1 to enable the `source3` Event
[4]	SOURCE4	Write a 1 to enable the `source4` Event
[5]	SOURCE5	Write a 1 to enable the `source5` Event
[6]	SOURCE6	Write a 1 to enable the `source6` Event
[7]	SOURCE7	Write a 1 to enable the `source7` Event
[8]	SOURCE8	Write a 1 to enable the `source8` Event
[9]	SOURCE9	Write a 1 to enable the `source9` Event
[10]	SOURCE10	Write a 1 to enable the `source10` Event
[11]	SOURCE11	Write a 1 to enable the `source11` Event
[12]	SOURCE12	Write a 1 to enable the `source12` Event
[13]	SOURCE13	Write a 1 to enable the `source13` Event
[14]	SOURCE14	Write a 1 to enable the `source14` Event
[15]	SOURCE15	Write a 1 to enable the `source15` Event
[16]	SOURCE16	Write a 1 to enable the `source16` Event
[17]	SOURCE17	Write a 1 to enable the `source17` Event
[18]	SOURCE18	Write a 1 to enable the `source18` Event
[19]	SOURCE19	Write a 1 to enable the `source19` Event

2.14 IRQARRAY17

IrqArray provides a large bank of interrupts for SoC integration. It is different from e.g. the NVIC or CLINT in that the register bank is structured along page boundaries, so that the interrupt handler CSRs can be owned by a specific virtual memory process, instead of bouncing through a common handler and forcing an inter-process message to be generated to route interrupts to their final destination.

The incoming interrupt signals are assumed to be synchronized to *aclk*.

Priorities are enforced entirely through software; the handler must read the *pending* bits and decide which ones should be handled first.

The *EventSource* is an *EventSourceFlex* which can handle pulses and levels, as well as software triggers.

The interrupt pending bit is latched when the trigger goes high, and stays high until software clears the event. The trigger takes precedence over clearing, so if the interrupt source is not cleared prior to clearing the interrupt pending bit, the interrupt will trigger again.

status reflects the instantaneous value of the trigger.

A separate input line is provided so that software can induce an interrupt by writing to a soft-trigger bit.

2.14.1 Register Listing for IRQARRAY17

Register	Address
IRQARRAY17_EV_SOFT	*0x5800d000*
IRQARRAY17_EV_STATUS	*0x5800d004*
IRQARRAY17_EV_PENDING	*0x5800d008*
IRQARRAY17_EV_ENABLE	*0x5800d00c*

IRQARRAY17_EV_SOFT

Address: 0x5800d000 + 0x0 = 0x5800d000

Software interrupt trigger register.

Bits set to *1* will trigger an interrupt. Interrupts trigger on write, but the value will persist in the register, allowing software to determine if a software interrupt was triggered by reading back the register.

Software is responsible for clearing the register to 0.

Repeated *1* writes without clearing will still trigger an interrupt.

Fig. 62: IRQARRAY17_EV_SOFT

Field	Name	Description
[19:0]	TRIGGER	Writing a 1 to this bit triggers the function.

IRQARRAY17_EV_STATUS

Address: 0x5800d000 + 0x4 = 0x5800d004

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

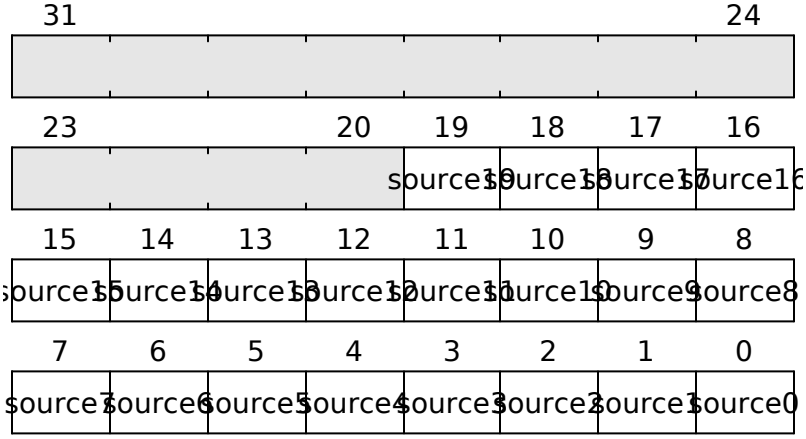

Fig. 63: IRQARRAY17_EV_STATUS

Field	Name	Description
[0]	SOURCE0	Level of the source0 event
[1]	SOURCE1	Level of the source1 event
[2]	SOURCE2	Level of the source2 event
[3]	SOURCE3	Level of the source3 event
[4]	SOURCE4	Level of the source4 event
[5]	SOURCE5	Level of the source5 event
[6]	SOURCE6	Level of the source6 event
[7]	SOURCE7	Level of the source7 event
[8]	SOURCE8	Level of the source8 event
[9]	SOURCE9	Level of the source9 event
[10]	SOURCE10	Level of the source10 event
[11]	SOURCE11	Level of the source11 event
[12]	SOURCE12	Level of the source12 event
[13]	SOURCE13	Level of the source13 event
[14]	SOURCE14	Level of the source14 event
[15]	SOURCE15	Level of the source15 event
[16]	SOURCE16	Level of the source16 event
[17]	SOURCE17	Level of the source17 event
[18]	SOURCE18	Level of the source18 event
[19]	SOURCE19	Level of the source19 event

IRQARRAY17_EV_PENDING

Address: 0x5800d000 + 0x8 = 0x5800d008

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

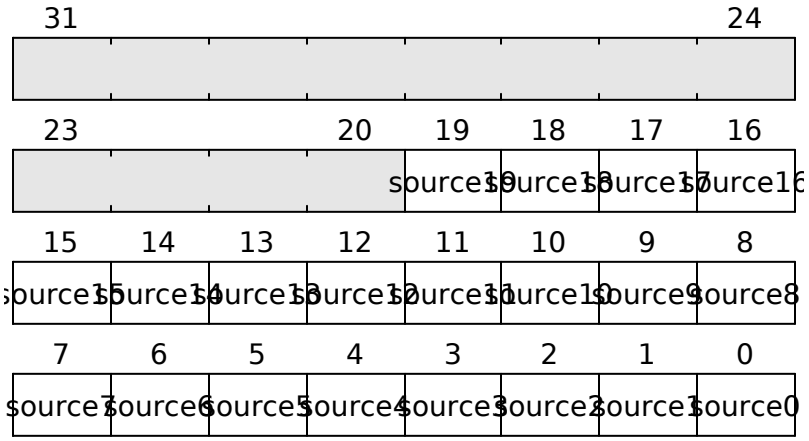

Fig. 64: IRQARRAY17_EV_PENDING

Field	Name	Description
[0]	SOURCE0	*1* when a source0 event occurs. This event uses an *EventSourceFlex* form of triggering
[1]	SOURCE1	*1* when a source1 event occurs. This event uses an *EventSourceFlex* form of triggering
[2]	SOURCE2	*1* when a source2 event occurs. This event uses an *EventSourceFlex* form of triggering
[3]	SOURCE3	*1* when a source3 event occurs. This event uses an *EventSourceFlex* form of triggering
[4]	SOURCE4	*1* when a source4 event occurs. This event uses an *EventSourceFlex* form of triggering
[5]	SOURCE5	*1* when a source5 event occurs. This event uses an *EventSourceFlex* form of triggering
[6]	SOURCE6	*1* when a source6 event occurs. This event uses an *EventSourceFlex* form of triggering
[7]	SOURCE7	*1* when a source7 event occurs. This event uses an *EventSourceFlex* form of triggering
[8]	SOURCE8	*1* when a source8 event occurs. This event uses an *EventSourceFlex* form of triggering
[9]	SOURCE9	*1* when a source9 event occurs. This event uses an *EventSourceFlex* form of triggering
[10]	SOURCE10	*1* when a source10 event occurs. This event uses an *EventSourceFlex* form of triggering
[11]	SOURCE11	*1* when a source11 event occurs. This event uses an *EventSourceFlex* form of triggering
[12]	SOURCE12	*1* when a source12 event occurs. This event uses an *EventSourceFlex* form of triggering
[13]	SOURCE13	*1* when a source13 event occurs. This event uses an *EventSourceFlex* form of triggering
[14]	SOURCE14	*1* when a source14 event occurs. This event uses an *EventSourceFlex* form of triggering
[15]	SOURCE15	*1* when a source15 event occurs. This event uses an *EventSourceFlex* form of triggering
[16]	SOURCE16	*1* when a source16 event occurs. This event uses an *EventSourceFlex* form of triggering
[17]	SOURCE17	*1* when a source17 event occurs. This event uses an *EventSourceFlex* form of triggering
[18]	SOURCE18	*1* when a source18 event occurs. This event uses an *EventSourceFlex* form of triggering
[19]	SOURCE19	*1* when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

IRQARRAY17_EV_ENABLE

Address: 0x5800d000 + 0xc = 0x5800d00c

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

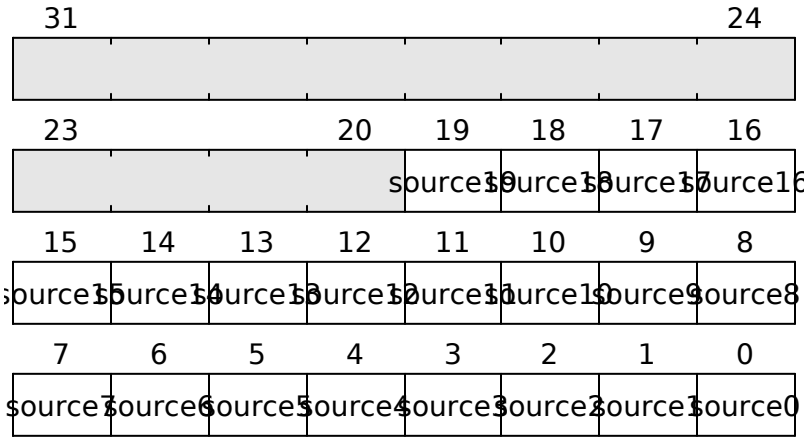

Fig. 65: IRQARRAY17_EV_ENABLE

Field	Name	Description
[0]	SOURCE0	Write a 1 to enable the `source0` Event
[1]	SOURCE1	Write a 1 to enable the `source1` Event
[2]	SOURCE2	Write a 1 to enable the `source2` Event
[3]	SOURCE3	Write a 1 to enable the `source3` Event
[4]	SOURCE4	Write a 1 to enable the `source4` Event
[5]	SOURCE5	Write a 1 to enable the `source5` Event
[6]	SOURCE6	Write a 1 to enable the `source6` Event
[7]	SOURCE7	Write a 1 to enable the `source7` Event
[8]	SOURCE8	Write a 1 to enable the `source8` Event
[9]	SOURCE9	Write a 1 to enable the `source9` Event
[10]	SOURCE10	Write a 1 to enable the `source10` Event
[11]	SOURCE11	Write a 1 to enable the `source11` Event
[12]	SOURCE12	Write a 1 to enable the `source12` Event
[13]	SOURCE13	Write a 1 to enable the `source13` Event
[14]	SOURCE14	Write a 1 to enable the `source14` Event
[15]	SOURCE15	Write a 1 to enable the `source15` Event
[16]	SOURCE16	Write a 1 to enable the `source16` Event
[17]	SOURCE17	Write a 1 to enable the `source17` Event
[18]	SOURCE18	Write a 1 to enable the `source18` Event
[19]	SOURCE19	Write a 1 to enable the `source19` Event

2.15 IRQARRAY18

IrqArray provides a large bank of interrupts for SoC integration. It is different from e.g. the NVIC or CLINT in that the register bank is structured along page boundaries, so that the interrupt handler CSRs can be owned by a specific virtual memory process, instead of bouncing through a common handler and forcing an inter-process message to be generated to route interrupts to their final destination.

The incoming interrupt signals are assumed to be synchronized to *aclk*.

Priorities are enforced entirely through software; the handler must read the *pending* bits and decide which ones should be handled first.

The *EventSource* is an *EventSourceFlex* which can handle pulses and levels, as well as software triggers.

The interrupt pending bit is latched when the trigger goes high, and stays high until software clears the event. The trigger takes precedence over clearing, so if the interrupt source is not cleared prior to clearing the interrupt pending bit, the interrupt will trigger again.

status reflects the instantaneous value of the trigger.

A separate input line is provided so that software can induce an interrupt by writing to a soft-trigger bit.

2.15.1 Register Listing for IRQARRAY18

Register	Address
IRQARRAY18_EV_SOFT	*0x5800e000*
IRQARRAY18_EV_STATUS	*0x5800e004*
IRQARRAY18_EV_PENDING	*0x5800e008*
IRQARRAY18_EV_ENABLE	*0x5800e00c*

IRQARRAY18_EV_SOFT

Address: 0x5800e000 + 0x0 = 0x5800e000

Software interrupt trigger register.

Bits set to *1* will trigger an interrupt. Interrupts trigger on write, but the value will persist in the register, allowing software to determine if a software interrupt was triggered by reading back the register.

Software is responsible for clearing the register to 0.

Repeated *1* writes without clearing will still trigger an interrupt.

Fig. 66: IRQARRAY18_EV_SOFT

Field	Name	Description
[19:0]	TRIGGER	Writing a 1 to this bit triggers the function.

IRQARRAY18_EV_STATUS

Address: 0x5800e000 + 0x4 = 0x5800e004

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

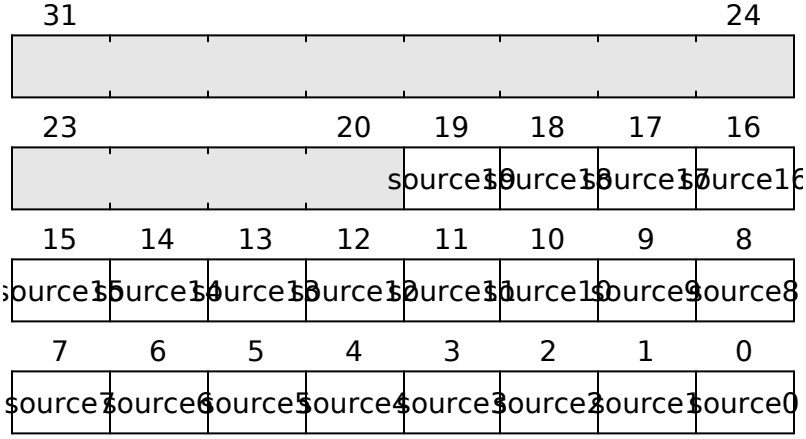

Fig. 67: IRQARRAY18_EV_STATUS

Field	Name	Description
[0]	SOURCE0	Level of the source0 event
[1]	SOURCE1	Level of the source1 event
[2]	SOURCE2	Level of the source2 event
[3]	SOURCE3	Level of the source3 event
[4]	SOURCE4	Level of the source4 event
[5]	SOURCE5	Level of the source5 event
[6]	SOURCE6	Level of the source6 event
[7]	SOURCE7	Level of the source7 event
[8]	SOURCE8	Level of the source8 event
[9]	SOURCE9	Level of the source9 event
[10]	SOURCE10	Level of the source10 event
[11]	SOURCE11	Level of the source11 event
[12]	SOURCE12	Level of the source12 event
[13]	SOURCE13	Level of the source13 event
[14]	SOURCE14	Level of the source14 event
[15]	SOURCE15	Level of the source15 event
[16]	SOURCE16	Level of the source16 event
[17]	SOURCE17	Level of the source17 event
[18]	SOURCE18	Level of the source18 event
[19]	SOURCE19	Level of the source19 event

IRQARRAY18_EV_PENDING

Address: 0x5800e000 + 0x8 = 0x5800e008

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

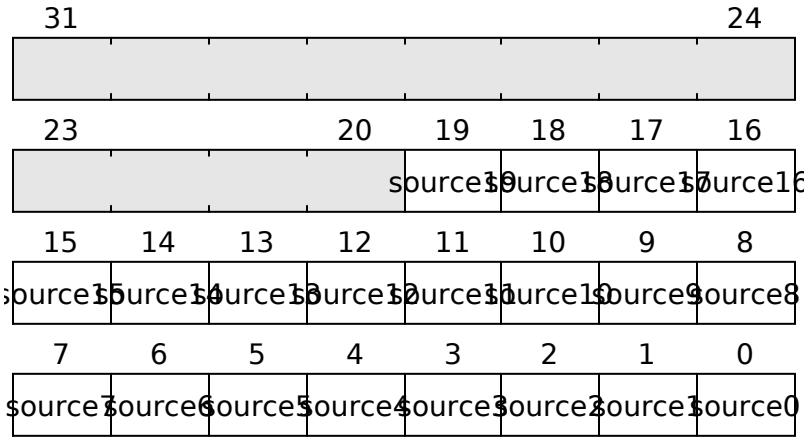

Fig. 68: IRQARRAY18_EV_PENDING

Field	Name	Description
[0]	SOURCE0	*1* when a source0 event occurs. This event uses an *EventSourceFlex* form of triggering
[1]	SOURCE1	*1* when a source1 event occurs. This event uses an *EventSourceFlex* form of triggering
[2]	SOURCE2	*1* when a source2 event occurs. This event uses an *EventSourceFlex* form of triggering
[3]	SOURCE3	*1* when a source3 event occurs. This event uses an *EventSourceFlex* form of triggering
[4]	SOURCE4	*1* when a source4 event occurs. This event uses an *EventSourceFlex* form of triggering
[5]	SOURCE5	*1* when a source5 event occurs. This event uses an *EventSourceFlex* form of triggering
[6]	SOURCE6	*1* when a source6 event occurs. This event uses an *EventSourceFlex* form of triggering
[7]	SOURCE7	*1* when a source7 event occurs. This event uses an *EventSourceFlex* form of triggering
[8]	SOURCE8	*1* when a source8 event occurs. This event uses an *EventSourceFlex* form of triggering
[9]	SOURCE9	*1* when a source9 event occurs. This event uses an *EventSourceFlex* form of triggering
[10]	SOURCE10	*1* when a source10 event occurs. This event uses an *EventSourceFlex* form of triggering
[11]	SOURCE11	*1* when a source11 event occurs. This event uses an *EventSourceFlex* form of triggering
[12]	SOURCE12	*1* when a source12 event occurs. This event uses an *EventSourceFlex* form of triggering
[13]	SOURCE13	*1* when a source13 event occurs. This event uses an *EventSourceFlex* form of triggering
[14]	SOURCE14	*1* when a source14 event occurs. This event uses an *EventSourceFlex* form of triggering
[15]	SOURCE15	*1* when a source15 event occurs. This event uses an *EventSourceFlex* form of triggering
[16]	SOURCE16	*1* when a source16 event occurs. This event uses an *EventSourceFlex* form of triggering
[17]	SOURCE17	*1* when a source17 event occurs. This event uses an *EventSourceFlex* form of triggering
[18]	SOURCE18	*1* when a source18 event occurs. This event uses an *EventSourceFlex* form of triggering
[19]	SOURCE19	*1* when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

IRQARRAY18_EV_ENABLE

Address: 0x5800e000 + 0xc = 0x5800e00c

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

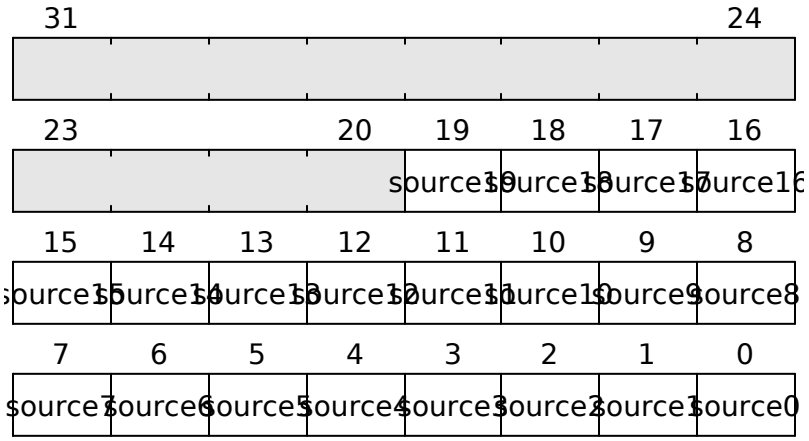

Fig. 69: IRQARRAY18_EV_ENABLE

Field	Name	Description
[0]	SOURCE0	Write a 1 to enable the `source0` Event
[1]	SOURCE1	Write a 1 to enable the `source1` Event
[2]	SOURCE2	Write a 1 to enable the `source2` Event
[3]	SOURCE3	Write a 1 to enable the `source3` Event
[4]	SOURCE4	Write a 1 to enable the `source4` Event
[5]	SOURCE5	Write a 1 to enable the `source5` Event
[6]	SOURCE6	Write a 1 to enable the `source6` Event
[7]	SOURCE7	Write a 1 to enable the `source7` Event
[8]	SOURCE8	Write a 1 to enable the `source8` Event
[9]	SOURCE9	Write a 1 to enable the `source9` Event
[10]	SOURCE10	Write a 1 to enable the `source10` Event
[11]	SOURCE11	Write a 1 to enable the `source11` Event
[12]	SOURCE12	Write a 1 to enable the `source12` Event
[13]	SOURCE13	Write a 1 to enable the `source13` Event
[14]	SOURCE14	Write a 1 to enable the `source14` Event
[15]	SOURCE15	Write a 1 to enable the `source15` Event
[16]	SOURCE16	Write a 1 to enable the `source16` Event
[17]	SOURCE17	Write a 1 to enable the `source17` Event
[18]	SOURCE18	Write a 1 to enable the `source18` Event
[19]	SOURCE19	Write a 1 to enable the `source19` Event

2.16 IRQARRAY19

IrqArray provides a large bank of interrupts for SoC integration. It is different from e.g. the NVIC or CLINT in that the register bank is structured along page boundaries, so that the interrupt handler CSRs can be owned by a specific virtual memory process, instead of bouncing through a common handler and forcing an inter-process message to be generated to route interrupts to their final destination.

The incoming interrupt signals are assumed to be synchronized to *aclk*.

Priorities are enforced entirely through software; the handler must read the *pending* bits and decide which ones should be handled first.

The *EventSource* is an *EventSourceFlex* which can handle pulses and levels, as well as software triggers.

The interrupt pending bit is latched when the trigger goes high, and stays high until software clears the event. The trigger takes precedence over clearing, so if the interrupt source is not cleared prior to clearing the interrupt pending bit, the interrupt will trigger again.

status reflects the instantaneous value of the trigger.

A separate input line is provided so that software can induce an interrupt by writing to a soft-trigger bit.

2.16.1 Register Listing for IRQARRAY19

Register	Address
IRQARRAY19_EV_SOFT	*0x5800f000*
IRQARRAY19_EV_STATUS	*0x5800f004*
IRQARRAY19_EV_PENDING	*0x5800f008*
IRQARRAY19_EV_ENABLE	*0x5800f00c*

IRQARRAY19_EV_SOFT

Address: 0x5800f000 + 0x0 = 0x5800f000

Software interrupt trigger register.

Bits set to *1* will trigger an interrupt. Interrupts trigger on write, but the value will persist in the register, allowing software to determine if a software interrupt was triggered by reading back the register.

Software is responsible for clearing the register to 0.

Repeated *1* writes without clearing will still trigger an interrupt.

Fig. 70: IRQARRAY19_EV_SOFT

Field	Name	Description
[19:0]	TRIGGER	Writing a 1 to this bit triggers the function.

IRQARRAY19_EV_STATUS

Address: 0x5800f000 + 0x4 = 0x5800f004

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

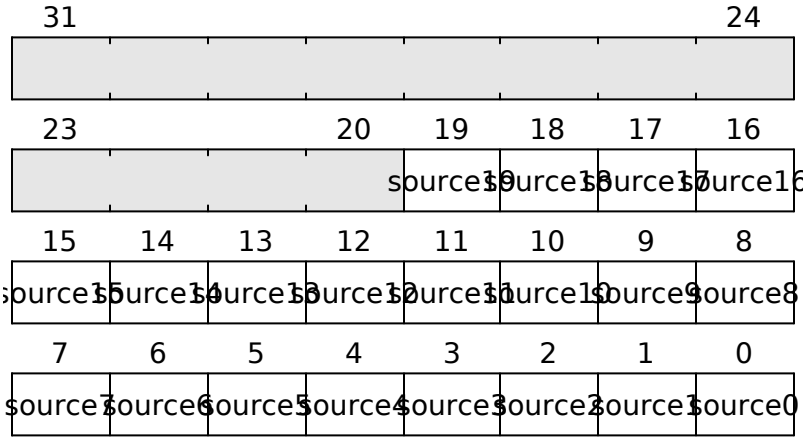

Fig. 71: IRQARRAY19_EV_STATUS

Field	Name	Description
[0]	SOURCE0	Level of the source0 event
[1]	SOURCE1	Level of the source1 event
[2]	SOURCE2	Level of the source2 event
[3]	SOURCE3	Level of the source3 event
[4]	SOURCE4	Level of the source4 event
[5]	SOURCE5	Level of the source5 event
[6]	SOURCE6	Level of the source6 event
[7]	SOURCE7	Level of the source7 event
[8]	SOURCE8	Level of the source8 event
[9]	SOURCE9	Level of the source9 event
[10]	SOURCE10	Level of the source10 event
[11]	SOURCE11	Level of the source11 event
[12]	SOURCE12	Level of the source12 event
[13]	SOURCE13	Level of the source13 event
[14]	SOURCE14	Level of the source14 event
[15]	SOURCE15	Level of the source15 event
[16]	SOURCE16	Level of the source16 event
[17]	SOURCE17	Level of the source17 event
[18]	SOURCE18	Level of the source18 event
[19]	SOURCE19	Level of the source19 event

IRQARRAY19_EV_PENDING

Address: 0x5800f000 + 0x8 = 0x5800f008

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

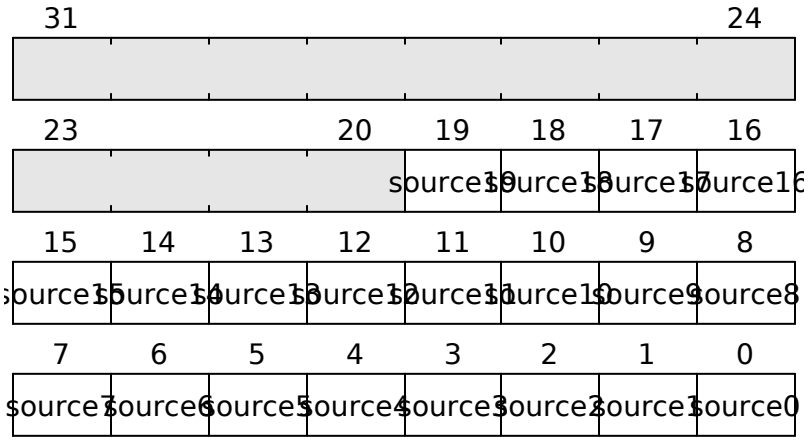

Fig. 72: IRQARRAY19_EV_PENDING

Field	Name	Description
[0]	SOURCE0	*1* when a source0 event occurs. This event uses an *EventSourceFlex* form of triggering
[1]	SOURCE1	*1* when a source1 event occurs. This event uses an *EventSourceFlex* form of triggering
[2]	SOURCE2	*1* when a source2 event occurs. This event uses an *EventSourceFlex* form of triggering
[3]	SOURCE3	*1* when a source3 event occurs. This event uses an *EventSourceFlex* form of triggering
[4]	SOURCE4	*1* when a source4 event occurs. This event uses an *EventSourceFlex* form of triggering
[5]	SOURCE5	*1* when a source5 event occurs. This event uses an *EventSourceFlex* form of triggering
[6]	SOURCE6	*1* when a source6 event occurs. This event uses an *EventSourceFlex* form of triggering
[7]	SOURCE7	*1* when a source7 event occurs. This event uses an *EventSourceFlex* form of triggering
[8]	SOURCE8	*1* when a source8 event occurs. This event uses an *EventSourceFlex* form of triggering
[9]	SOURCE9	*1* when a source9 event occurs. This event uses an *EventSourceFlex* form of triggering
[10]	SOURCE10	*1* when a source10 event occurs. This event uses an *EventSourceFlex* form of triggering
[11]	SOURCE11	*1* when a source11 event occurs. This event uses an *EventSourceFlex* form of triggering
[12]	SOURCE12	*1* when a source12 event occurs. This event uses an *EventSourceFlex* form of triggering
[13]	SOURCE13	*1* when a source13 event occurs. This event uses an *EventSourceFlex* form of triggering
[14]	SOURCE14	*1* when a source14 event occurs. This event uses an *EventSourceFlex* form of triggering
[15]	SOURCE15	*1* when a source15 event occurs. This event uses an *EventSourceFlex* form of triggering
[16]	SOURCE16	*1* when a source16 event occurs. This event uses an *EventSourceFlex* form of triggering
[17]	SOURCE17	*1* when a source17 event occurs. This event uses an *EventSourceFlex* form of triggering
[18]	SOURCE18	*1* when a source18 event occurs. This event uses an *EventSourceFlex* form of triggering
[19]	SOURCE19	*1* when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

IRQARRAY19_EV_ENABLE

Address: 0x5800f000 + 0xc = 0x5800f00c

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

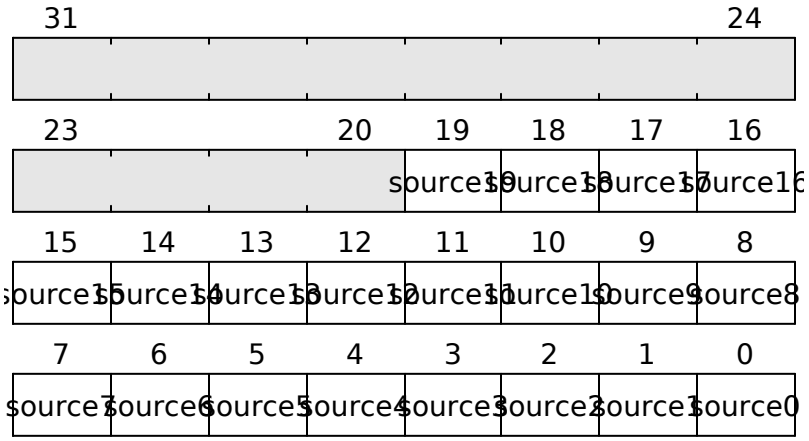

Fig. 73: IRQARRAY19_EV_ENABLE

Field	Name	Description
[0]	SOURCE0	Write a 1 to enable the `source0` Event
[1]	SOURCE1	Write a 1 to enable the `source1` Event
[2]	SOURCE2	Write a 1 to enable the `source2` Event
[3]	SOURCE3	Write a 1 to enable the `source3` Event
[4]	SOURCE4	Write a 1 to enable the `source4` Event
[5]	SOURCE5	Write a 1 to enable the `source5` Event
[6]	SOURCE6	Write a 1 to enable the `source6` Event
[7]	SOURCE7	Write a 1 to enable the `source7` Event
[8]	SOURCE8	Write a 1 to enable the `source8` Event
[9]	SOURCE9	Write a 1 to enable the `source9` Event
[10]	SOURCE10	Write a 1 to enable the `source10` Event
[11]	SOURCE11	Write a 1 to enable the `source11` Event
[12]	SOURCE12	Write a 1 to enable the `source12` Event
[13]	SOURCE13	Write a 1 to enable the `source13` Event
[14]	SOURCE14	Write a 1 to enable the `source14` Event
[15]	SOURCE15	Write a 1 to enable the `source15` Event
[16]	SOURCE16	Write a 1 to enable the `source16` Event
[17]	SOURCE17	Write a 1 to enable the `source17` Event
[18]	SOURCE18	Write a 1 to enable the `source18` Event
[19]	SOURCE19	Write a 1 to enable the `source19` Event

2.17 IRQARRAY2

IrqArray provides a large bank of interrupts for SoC integration. It is different from e.g. the NVIC or CLINT in that the register bank is structured along page boundaries, so that the interrupt handler CSRs can be owned by a specific virtual memory process, instead of bouncing through a common handler and forcing an inter-process message to be generated to route interrupts to their final destination.

The incoming interrupt signals are assumed to be synchronized to *aclk*.

Priorities are enforced entirely through software; the handler must read the *pending* bits and decide which ones should be handled first.

The *EventSource* is an *EventSourceFlex* which can handle pulses and levels, as well as software triggers.

The interrupt pending bit is latched when the trigger goes high, and stays high until software clears the event. The trigger takes precedence over clearing, so if the interrupt source is not cleared prior to clearing the interrupt pending bit, the interrupt will trigger again.

status reflects the instantaneous value of the trigger.

A separate input line is provided so that software can induce an interrupt by writing to a soft-trigger bit.

2.17.1 Register Listing for IRQARRAY2

Register	Address
IRQARRAY2_EV_SOFT	*0x58010000*
IRQARRAY2_EV_STATUS	*0x58010004*
IRQARRAY2_EV_PENDING	*0x58010008*
IRQARRAY2_EV_ENABLE	*0x5801000c*

IRQARRAY2_EV_SOFT

Address: 0x58010000 + 0x0 = 0x58010000

Software interrupt trigger register.

Bits set to *1* will trigger an interrupt. Interrupts trigger on write, but the value will persist in the register, allowing software to determine if a software interrupt was triggered by reading back the register.

Software is responsible for clearing the register to 0.

Repeated *1* writes without clearing will still trigger an interrupt.

Fig. 74: IRQARRAY2_EV_SOFT

Field	Name	Description
[19:0]	TRIGGER	Writing a 1 to this bit triggers the function.

IRQARRAY2_EV_STATUS

Address: 0x58010000 + 0x4 = 0x58010004

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

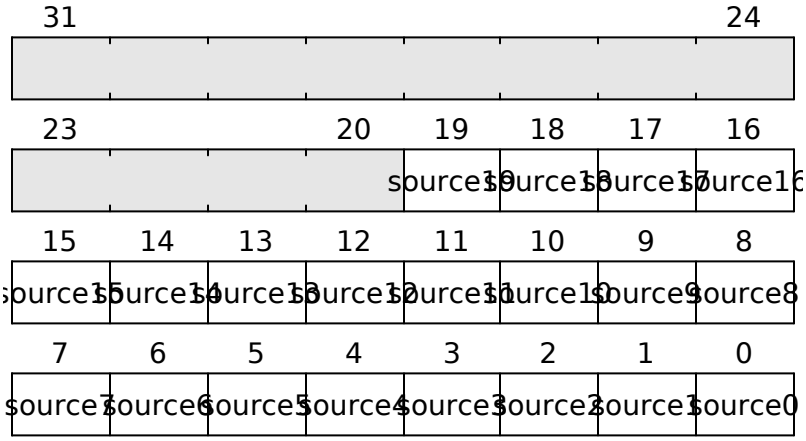

Fig. 75: IRQARRAY2_EV_STATUS

Field	Name	Description
[0]	SOURCE0	Level of the source0 event
[1]	SOURCE1	Level of the source1 event
[2]	SOURCE2	Level of the source2 event
[3]	SOURCE3	Level of the source3 event
[4]	SOURCE4	Level of the source4 event
[5]	SOURCE5	Level of the source5 event
[6]	SOURCE6	Level of the source6 event
[7]	SOURCE7	Level of the source7 event
[8]	SOURCE8	Level of the source8 event
[9]	SOURCE9	Level of the source9 event
[10]	SOURCE10	Level of the source10 event
[11]	SOURCE11	Level of the source11 event
[12]	SOURCE12	Level of the source12 event
[13]	SOURCE13	Level of the source13 event
[14]	SOURCE14	Level of the source14 event
[15]	SOURCE15	Level of the source15 event
[16]	SOURCE16	Level of the source16 event
[17]	SOURCE17	Level of the source17 event
[18]	SOURCE18	Level of the source18 event
[19]	SOURCE19	Level of the source19 event

IRQARRAY2_EV_PENDING

Address: 0x58010000 + 0x8 = 0x58010008

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

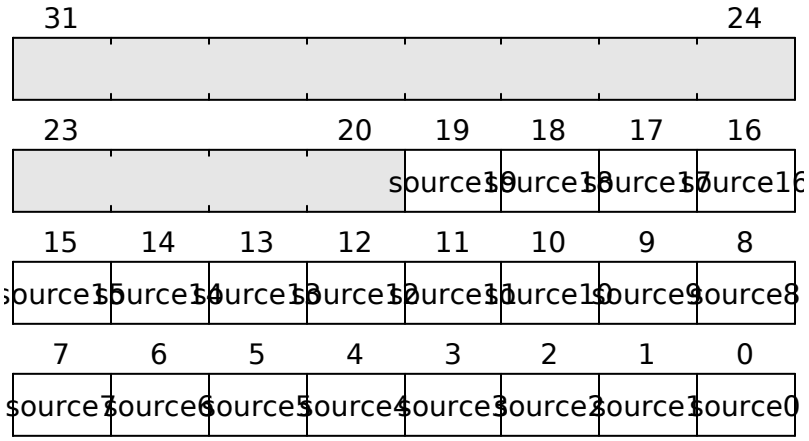

Fig. 76: IRQARRAY2_EV_PENDING

Field	Name	Description
[0]	SOURCE0	*1* when a source0 event occurs. This event uses an *EventSourceFlex* form of triggering
[1]	SOURCE1	*1* when a source1 event occurs. This event uses an *EventSourceFlex* form of triggering
[2]	SOURCE2	*1* when a source2 event occurs. This event uses an *EventSourceFlex* form of triggering
[3]	SOURCE3	*1* when a source3 event occurs. This event uses an *EventSourceFlex* form of triggering
[4]	SOURCE4	*1* when a source4 event occurs. This event uses an *EventSourceFlex* form of triggering
[5]	SOURCE5	*1* when a source5 event occurs. This event uses an *EventSourceFlex* form of triggering
[6]	SOURCE6	*1* when a source6 event occurs. This event uses an *EventSourceFlex* form of triggering
[7]	SOURCE7	*1* when a source7 event occurs. This event uses an *EventSourceFlex* form of triggering
[8]	SOURCE8	*1* when a source8 event occurs. This event uses an *EventSourceFlex* form of triggering
[9]	SOURCE9	*1* when a source9 event occurs. This event uses an *EventSourceFlex* form of triggering
[10]	SOURCE10	*1* when a source10 event occurs. This event uses an *EventSourceFlex* form of triggering
[11]	SOURCE11	*1* when a source11 event occurs. This event uses an *EventSourceFlex* form of triggering
[12]	SOURCE12	*1* when a source12 event occurs. This event uses an *EventSourceFlex* form of triggering
[13]	SOURCE13	*1* when a source13 event occurs. This event uses an *EventSourceFlex* form of triggering
[14]	SOURCE14	*1* when a source14 event occurs. This event uses an *EventSourceFlex* form of triggering
[15]	SOURCE15	*1* when a source15 event occurs. This event uses an *EventSourceFlex* form of triggering
[16]	SOURCE16	*1* when a source16 event occurs. This event uses an *EventSourceFlex* form of triggering
[17]	SOURCE17	*1* when a source17 event occurs. This event uses an *EventSourceFlex* form of triggering
[18]	SOURCE18	*1* when a source18 event occurs. This event uses an *EventSourceFlex* form of triggering
[19]	SOURCE19	*1* when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

IRQARRAY2_EV_ENABLE

Address: 0x58010000 + 0xc = 0x5801000c

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

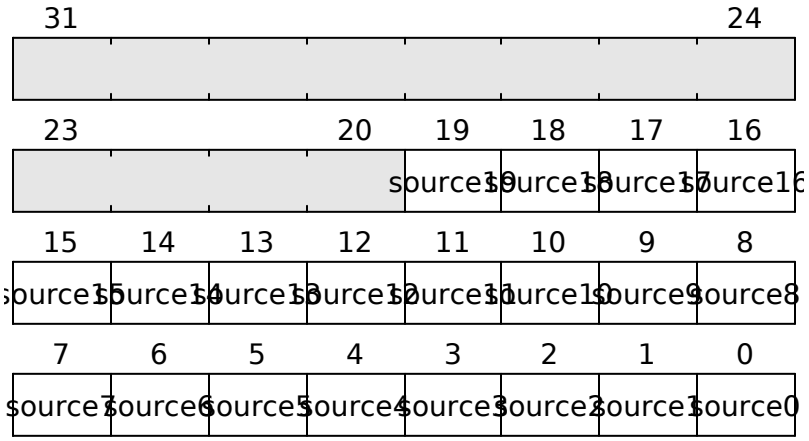

Fig. 77: IRQARRAY2_EV_ENABLE

Field	Name	Description
[0]	SOURCE0	Write a 1 to enable the `source0` Event
[1]	SOURCE1	Write a 1 to enable the `source1` Event
[2]	SOURCE2	Write a 1 to enable the `source2` Event
[3]	SOURCE3	Write a 1 to enable the `source3` Event
[4]	SOURCE4	Write a 1 to enable the `source4` Event
[5]	SOURCE5	Write a 1 to enable the `source5` Event
[6]	SOURCE6	Write a 1 to enable the `source6` Event
[7]	SOURCE7	Write a 1 to enable the `source7` Event
[8]	SOURCE8	Write a 1 to enable the `source8` Event
[9]	SOURCE9	Write a 1 to enable the `source9` Event
[10]	SOURCE10	Write a 1 to enable the `source10` Event
[11]	SOURCE11	Write a 1 to enable the `source11` Event
[12]	SOURCE12	Write a 1 to enable the `source12` Event
[13]	SOURCE13	Write a 1 to enable the `source13` Event
[14]	SOURCE14	Write a 1 to enable the `source14` Event
[15]	SOURCE15	Write a 1 to enable the `source15` Event
[16]	SOURCE16	Write a 1 to enable the `source16` Event
[17]	SOURCE17	Write a 1 to enable the `source17` Event
[18]	SOURCE18	Write a 1 to enable the `source18` Event
[19]	SOURCE19	Write a 1 to enable the `source19` Event

2.18 IRQARRAY3

IrqArray provides a large bank of interrupts for SoC integration. It is different from e.g. the NVIC or CLINT in that the register bank is structured along page boundaries, so that the interrupt handler CSRs can be owned by a specific virtual memory process, instead of bouncing through a common handler and forcing an inter-process message to be generated to route interrupts to their final destination.

The incoming interrupt signals are assumed to be synchronized to *aclk*.

Priorities are enforced entirely through software; the handler must read the *pending* bits and decide which ones should be handled first.

The *EventSource* is an *EventSourceFlex* which can handle pulses and levels, as well as software triggers.

The interrupt pending bit is latched when the trigger goes high, and stays high until software clears the event. The trigger takes precedence over clearing, so if the interrupt source is not cleared prior to clearing the interrupt pending bit, the interrupt will trigger again.

status reflects the instantaneous value of the trigger.

A separate input line is provided so that software can induce an interrupt by writing to a soft-trigger bit.

2.18.1 Register Listing for IRQARRAY3

Register	Address
IRQARRAY3_EV_SOFT	*0x58011000*
IRQARRAY3_EV_STATUS	*0x58011004*
IRQARRAY3_EV_PENDING	*0x58011008*
IRQARRAY3_EV_ENABLE	*0x5801100c*

IRQARRAY3_EV_SOFT

Address: 0x58011000 + 0x0 = 0x58011000

Software interrupt trigger register.

Bits set to *1* will trigger an interrupt. Interrupts trigger on write, but the value will persist in the register, allowing software to determine if a software interrupt was triggered by reading back the register.

Software is responsible for clearing the register to 0.

Repeated *1* writes without clearing will still trigger an interrupt.

Fig. 78: IRQARRAY3_EV_SOFT

Field	Name	Description
[19:0]	TRIGGER	Writing a 1 to this bit triggers the function.

IRQARRAY3_EV_STATUS

Address: 0x58011000 + 0x4 = 0x58011004

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

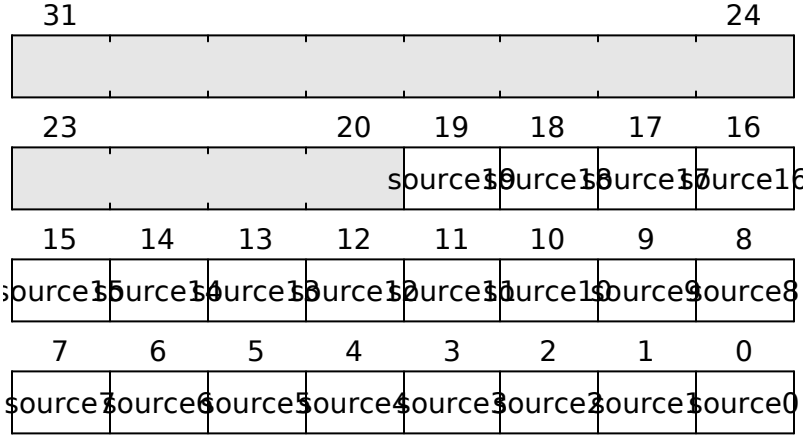

Fig. 79: IRQARRAY3_EV_STATUS

Field	Name	Description
[0]	SOURCE0	Level of the source0 event
[1]	SOURCE1	Level of the source1 event
[2]	SOURCE2	Level of the source2 event
[3]	SOURCE3	Level of the source3 event
[4]	SOURCE4	Level of the source4 event
[5]	SOURCE5	Level of the source5 event
[6]	SOURCE6	Level of the source6 event
[7]	SOURCE7	Level of the source7 event
[8]	SOURCE8	Level of the source8 event
[9]	SOURCE9	Level of the source9 event
[10]	SOURCE10	Level of the source10 event
[11]	SOURCE11	Level of the source11 event
[12]	SOURCE12	Level of the source12 event
[13]	SOURCE13	Level of the source13 event
[14]	SOURCE14	Level of the source14 event
[15]	SOURCE15	Level of the source15 event
[16]	SOURCE16	Level of the source16 event
[17]	SOURCE17	Level of the source17 event
[18]	SOURCE18	Level of the source18 event
[19]	SOURCE19	Level of the source19 event

IRQARRAY3_EV_PENDING

Address: 0x58011000 + 0x8 = 0x58011008

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

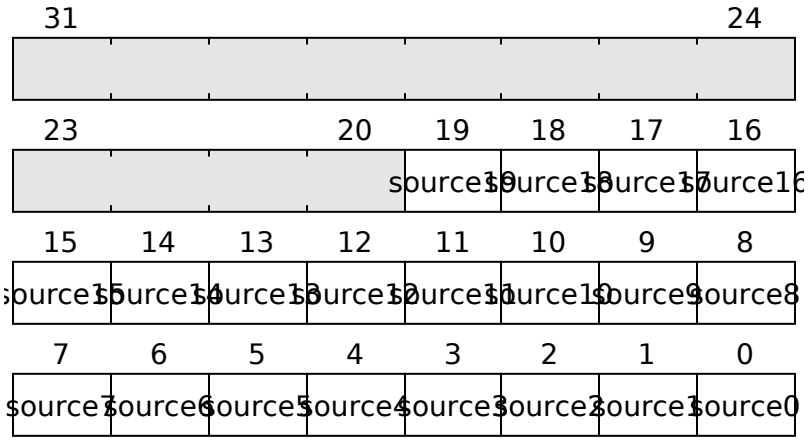

Fig. 80: IRQARRAY3_EV_PENDING

Field	Name	Description
[0]	SOURCE0	*1* when a source0 event occurs. This event uses an *EventSourceFlex* form of triggering
[1]	SOURCE1	*1* when a source1 event occurs. This event uses an *EventSourceFlex* form of triggering
[2]	SOURCE2	*1* when a source2 event occurs. This event uses an *EventSourceFlex* form of triggering
[3]	SOURCE3	*1* when a source3 event occurs. This event uses an *EventSourceFlex* form of triggering
[4]	SOURCE4	*1* when a source4 event occurs. This event uses an *EventSourceFlex* form of triggering
[5]	SOURCE5	*1* when a source5 event occurs. This event uses an *EventSourceFlex* form of triggering
[6]	SOURCE6	*1* when a source6 event occurs. This event uses an *EventSourceFlex* form of triggering
[7]	SOURCE7	*1* when a source7 event occurs. This event uses an *EventSourceFlex* form of triggering
[8]	SOURCE8	*1* when a source8 event occurs. This event uses an *EventSourceFlex* form of triggering
[9]	SOURCE9	*1* when a source9 event occurs. This event uses an *EventSourceFlex* form of triggering
[10]	SOURCE10	*1* when a source10 event occurs. This event uses an *EventSourceFlex* form of triggering
[11]	SOURCE11	*1* when a source11 event occurs. This event uses an *EventSourceFlex* form of triggering
[12]	SOURCE12	*1* when a source12 event occurs. This event uses an *EventSourceFlex* form of triggering
[13]	SOURCE13	*1* when a source13 event occurs. This event uses an *EventSourceFlex* form of triggering
[14]	SOURCE14	*1* when a source14 event occurs. This event uses an *EventSourceFlex* form of triggering
[15]	SOURCE15	*1* when a source15 event occurs. This event uses an *EventSourceFlex* form of triggering
[16]	SOURCE16	*1* when a source16 event occurs. This event uses an *EventSourceFlex* form of triggering
[17]	SOURCE17	*1* when a source17 event occurs. This event uses an *EventSourceFlex* form of triggering
[18]	SOURCE18	*1* when a source18 event occurs. This event uses an *EventSourceFlex* form of triggering
[19]	SOURCE19	*1* when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

IRQARRAY3_EV_ENABLE

Address: 0x58011000 + 0xc = 0x5801100c

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

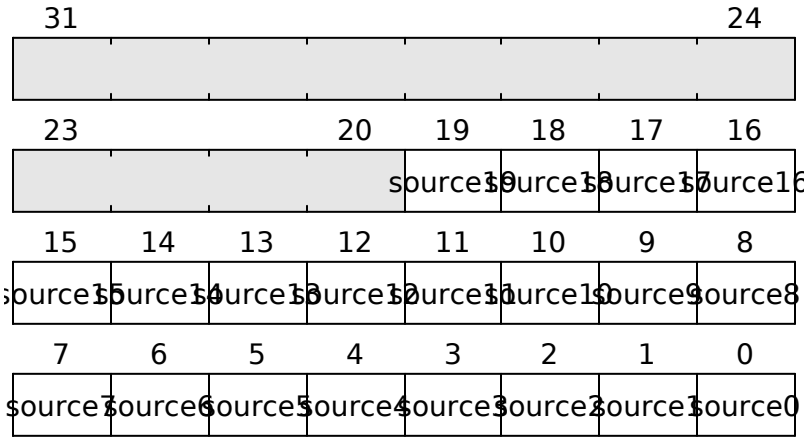

Fig. 81: IRQARRAY3_EV_ENABLE

Field	Name	Description
[0]	SOURCE0	Write a 1 to enable the `source0` Event
[1]	SOURCE1	Write a 1 to enable the `source1` Event
[2]	SOURCE2	Write a 1 to enable the `source2` Event
[3]	SOURCE3	Write a 1 to enable the `source3` Event
[4]	SOURCE4	Write a 1 to enable the `source4` Event
[5]	SOURCE5	Write a 1 to enable the `source5` Event
[6]	SOURCE6	Write a 1 to enable the `source6` Event
[7]	SOURCE7	Write a 1 to enable the `source7` Event
[8]	SOURCE8	Write a 1 to enable the `source8` Event
[9]	SOURCE9	Write a 1 to enable the `source9` Event
[10]	SOURCE10	Write a 1 to enable the `source10` Event
[11]	SOURCE11	Write a 1 to enable the `source11` Event
[12]	SOURCE12	Write a 1 to enable the `source12` Event
[13]	SOURCE13	Write a 1 to enable the `source13` Event
[14]	SOURCE14	Write a 1 to enable the `source14` Event
[15]	SOURCE15	Write a 1 to enable the `source15` Event
[16]	SOURCE16	Write a 1 to enable the `source16` Event
[17]	SOURCE17	Write a 1 to enable the `source17` Event
[18]	SOURCE18	Write a 1 to enable the `source18` Event
[19]	SOURCE19	Write a 1 to enable the `source19` Event

2.19 IRQARRAY4

IrqArray provides a large bank of interrupts for SoC integration. It is different from e.g. the NVIC or CLINT in that the register bank is structured along page boundaries, so that the interrupt handler CSRs can be owned by a specific virtual memory process, instead of bouncing through a common handler and forcing an inter-process message to be generated to route interrupts to their final destination.

The incoming interrupt signals are assumed to be synchronized to *aclk*.

Priorities are enforced entirely through software; the handler must read the *pending* bits and decide which ones should be handled first.

The *EventSource* is an *EventSourceFlex* which can handle pulses and levels, as well as software triggers.

The interrupt pending bit is latched when the trigger goes high, and stays high until software clears the event. The trigger takes precedence over clearing, so if the interrupt source is not cleared prior to clearing the interrupt pending bit, the interrupt will trigger again.

status reflects the instantaneous value of the trigger.

A separate input line is provided so that software can induce an interrupt by writing to a soft-trigger bit.

2.19.1 Register Listing for IRQARRAY4

Register	Address
IRQARRAY4_EV_SOFT	*0x58012000*
IRQARRAY4_EV_STATUS	*0x58012004*
IRQARRAY4_EV_PENDING	*0x58012008*
IRQARRAY4_EV_ENABLE	*0x5801200c*

IRQARRAY4_EV_SOFT

Address: 0x58012000 + 0x0 = 0x58012000

Software interrupt trigger register.

Bits set to *1* will trigger an interrupt. Interrupts trigger on write, but the value will persist in the register, allowing software to determine if a software interrupt was triggered by reading back the register.

Software is responsible for clearing the register to 0.

Repeated *1* writes without clearing will still trigger an interrupt.

Fig. 82: IRQARRAY4_EV_SOFT

Field	Name	Description
[19:0]	TRIGGER	Writing a 1 to this bit triggers the function.

IRQARRAY4_EV_STATUS

Address: 0x58012000 + 0x4 = 0x58012004

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

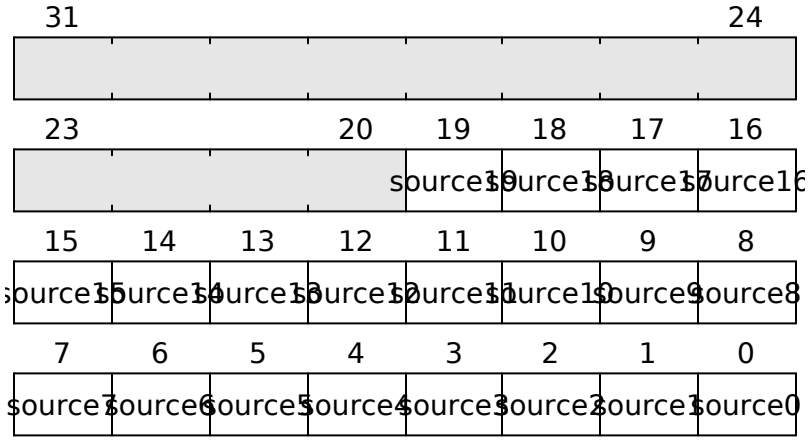

Fig. 83: IRQARRAY4_EV_STATUS

Field	Name	Description
[0]	SOURCE0	Level of the source0 event
[1]	SOURCE1	Level of the source1 event
[2]	SOURCE2	Level of the source2 event
[3]	SOURCE3	Level of the source3 event
[4]	SOURCE4	Level of the source4 event
[5]	SOURCE5	Level of the source5 event
[6]	SOURCE6	Level of the source6 event
[7]	SOURCE7	Level of the source7 event
[8]	SOURCE8	Level of the source8 event
[9]	SOURCE9	Level of the source9 event
[10]	SOURCE10	Level of the source10 event
[11]	SOURCE11	Level of the source11 event
[12]	SOURCE12	Level of the source12 event
[13]	SOURCE13	Level of the source13 event
[14]	SOURCE14	Level of the source14 event
[15]	SOURCE15	Level of the source15 event
[16]	SOURCE16	Level of the source16 event
[17]	SOURCE17	Level of the source17 event
[18]	SOURCE18	Level of the source18 event
[19]	SOURCE19	Level of the source19 event

IRQARRAY4_EV_PENDING

Address: 0x58012000 + 0x8 = 0x58012008

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

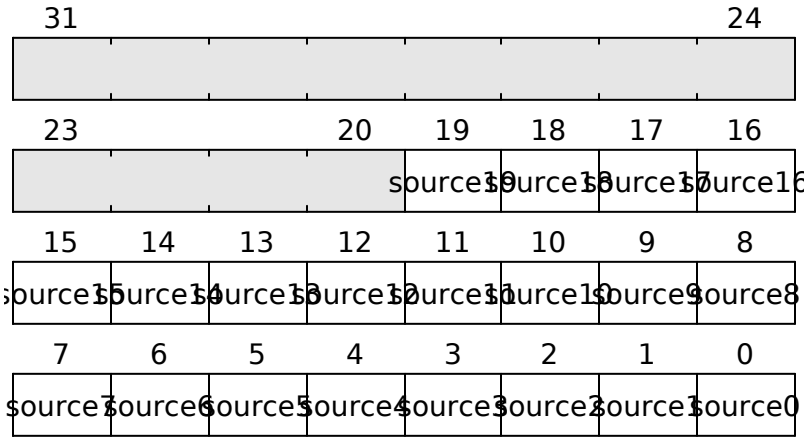

Fig. 84: IRQARRAY4_EV_PENDING

Field	Name	Description
[0]	SOURCE0	*1* when a source0 event occurs. This event uses an *EventSourceFlex* form of triggering
[1]	SOURCE1	*1* when a source1 event occurs. This event uses an *EventSourceFlex* form of triggering
[2]	SOURCE2	*1* when a source2 event occurs. This event uses an *EventSourceFlex* form of triggering
[3]	SOURCE3	*1* when a source3 event occurs. This event uses an *EventSourceFlex* form of triggering
[4]	SOURCE4	*1* when a source4 event occurs. This event uses an *EventSourceFlex* form of triggering
[5]	SOURCE5	*1* when a source5 event occurs. This event uses an *EventSourceFlex* form of triggering
[6]	SOURCE6	*1* when a source6 event occurs. This event uses an *EventSourceFlex* form of triggering
[7]	SOURCE7	*1* when a source7 event occurs. This event uses an *EventSourceFlex* form of triggering
[8]	SOURCE8	*1* when a source8 event occurs. This event uses an *EventSourceFlex* form of triggering
[9]	SOURCE9	*1* when a source9 event occurs. This event uses an *EventSourceFlex* form of triggering
[10]	SOURCE10	*1* when a source10 event occurs. This event uses an *EventSourceFlex* form of triggering
[11]	SOURCE11	*1* when a source11 event occurs. This event uses an *EventSourceFlex* form of triggering
[12]	SOURCE12	*1* when a source12 event occurs. This event uses an *EventSourceFlex* form of triggering
[13]	SOURCE13	*1* when a source13 event occurs. This event uses an *EventSourceFlex* form of triggering
[14]	SOURCE14	*1* when a source14 event occurs. This event uses an *EventSourceFlex* form of triggering
[15]	SOURCE15	*1* when a source15 event occurs. This event uses an *EventSourceFlex* form of triggering
[16]	SOURCE16	*1* when a source16 event occurs. This event uses an *EventSourceFlex* form of triggering
[17]	SOURCE17	*1* when a source17 event occurs. This event uses an *EventSourceFlex* form of triggering
[18]	SOURCE18	*1* when a source18 event occurs. This event uses an *EventSourceFlex* form of triggering
[19]	SOURCE19	*1* when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

IRQARRAY4_EV_ENABLE

Address: 0x58012000 + 0xc = 0x5801200c

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

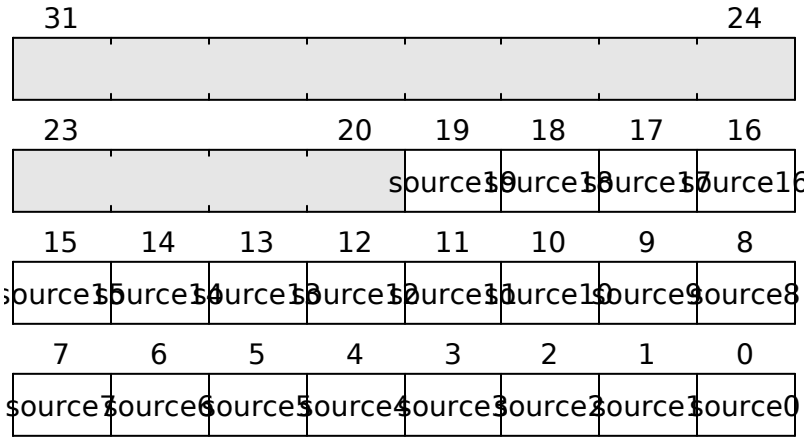

Fig. 85: IRQARRAY4_EV_ENABLE

Field	Name	Description
[0]	SOURCE0	Write a 1 to enable the `source0` Event
[1]	SOURCE1	Write a 1 to enable the `source1` Event
[2]	SOURCE2	Write a 1 to enable the `source2` Event
[3]	SOURCE3	Write a 1 to enable the `source3` Event
[4]	SOURCE4	Write a 1 to enable the `source4` Event
[5]	SOURCE5	Write a 1 to enable the `source5` Event
[6]	SOURCE6	Write a 1 to enable the `source6` Event
[7]	SOURCE7	Write a 1 to enable the `source7` Event
[8]	SOURCE8	Write a 1 to enable the `source8` Event
[9]	SOURCE9	Write a 1 to enable the `source9` Event
[10]	SOURCE10	Write a 1 to enable the `source10` Event
[11]	SOURCE11	Write a 1 to enable the `source11` Event
[12]	SOURCE12	Write a 1 to enable the `source12` Event
[13]	SOURCE13	Write a 1 to enable the `source13` Event
[14]	SOURCE14	Write a 1 to enable the `source14` Event
[15]	SOURCE15	Write a 1 to enable the `source15` Event
[16]	SOURCE16	Write a 1 to enable the `source16` Event
[17]	SOURCE17	Write a 1 to enable the `source17` Event
[18]	SOURCE18	Write a 1 to enable the `source18` Event
[19]	SOURCE19	Write a 1 to enable the `source19` Event

2.20 IRQARRAY5

IrqArray provides a large bank of interrupts for SoC integration. It is different from e.g. the NVIC or CLINT in that the register bank is structured along page boundaries, so that the interrupt handler CSRs can be owned by a specific virtual memory process, instead of bouncing through a common handler and forcing an inter-process message to be generated to route interrupts to their final destination.

The incoming interrupt signals are assumed to be synchronized to *aclk*.

Priorities are enforced entirely through software; the handler must read the *pending* bits and decide which ones should be handled first.

The *EventSource* is an *EventSourceFlex* which can handle pulses and levels, as well as software triggers.

The interrupt pending bit is latched when the trigger goes high, and stays high until software clears the event. The trigger takes precedence over clearing, so if the interrupt source is not cleared prior to clearing the interrupt pending bit, the interrupt will trigger again.

status reflects the instantaneous value of the trigger.

A separate input line is provided so that software can induce an interrupt by writing to a soft-trigger bit.

2.20.1 Register Listing for IRQARRAY5

Register	Address
IRQARRAY5_EV_SOFT	*0x58013000*
IRQARRAY5_EV_STATUS	*0x58013004*
IRQARRAY5_EV_PENDING	*0x58013008*
IRQARRAY5_EV_ENABLE	*0x5801300c*

IRQARRAY5_EV_SOFT

Address: 0x58013000 + 0x0 = 0x58013000

Software interrupt trigger register.

Bits set to *1* will trigger an interrupt. Interrupts trigger on write, but the value will persist in the register, allowing software to determine if a software interrupt was triggered by reading back the register.

Software is responsible for clearing the register to 0.

Repeated *1* writes without clearing will still trigger an interrupt.

Fig. 86: IRQARRAY5_EV_SOFT

Field	Name	Description
[19:0]	TRIGGER	Writing a 1 to this bit triggers the function.

IRQARRAY5_EV_STATUS

Address: 0x58013000 + 0x4 = 0x58013004

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

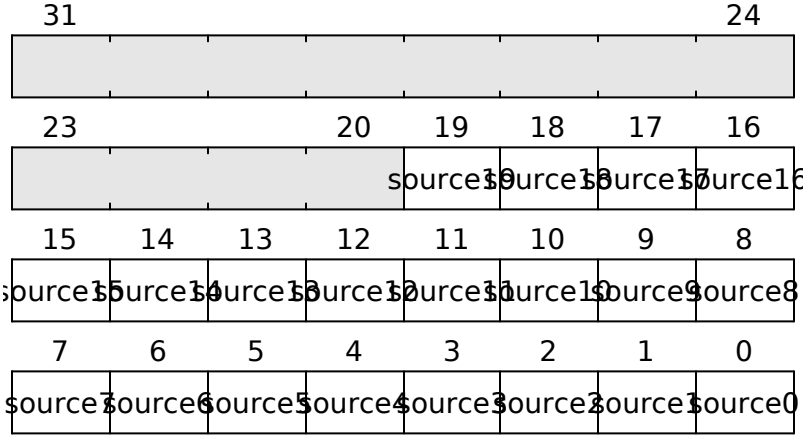

Fig. 87: IRQARRAY5_EV_STATUS

Field	Name	Description
[0]	SOURCE0	Level of the source0 event
[1]	SOURCE1	Level of the source1 event
[2]	SOURCE2	Level of the source2 event
[3]	SOURCE3	Level of the source3 event
[4]	SOURCE4	Level of the source4 event
[5]	SOURCE5	Level of the source5 event
[6]	SOURCE6	Level of the source6 event
[7]	SOURCE7	Level of the source7 event
[8]	SOURCE8	Level of the source8 event
[9]	SOURCE9	Level of the source9 event
[10]	SOURCE10	Level of the source10 event
[11]	SOURCE11	Level of the source11 event
[12]	SOURCE12	Level of the source12 event
[13]	SOURCE13	Level of the source13 event
[14]	SOURCE14	Level of the source14 event
[15]	SOURCE15	Level of the source15 event
[16]	SOURCE16	Level of the source16 event
[17]	SOURCE17	Level of the source17 event
[18]	SOURCE18	Level of the source18 event
[19]	SOURCE19	Level of the source19 event

IRQARRAY5_EV_PENDING

Address: 0x58013000 + 0x8 = 0x58013008

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

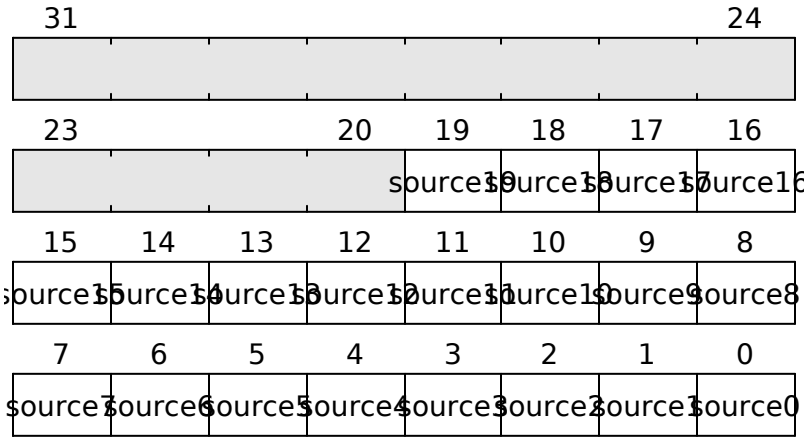

Fig. 88: IRQARRAY5_EV_PENDING

Field	Name	Description
[0]	SOURCE0	*1* when a source0 event occurs. This event uses an *EventSourceFlex* form of triggering
[1]	SOURCE1	*1* when a source1 event occurs. This event uses an *EventSourceFlex* form of triggering
[2]	SOURCE2	*1* when a source2 event occurs. This event uses an *EventSourceFlex* form of triggering
[3]	SOURCE3	*1* when a source3 event occurs. This event uses an *EventSourceFlex* form of triggering
[4]	SOURCE4	*1* when a source4 event occurs. This event uses an *EventSourceFlex* form of triggering
[5]	SOURCE5	*1* when a source5 event occurs. This event uses an *EventSourceFlex* form of triggering
[6]	SOURCE6	*1* when a source6 event occurs. This event uses an *EventSourceFlex* form of triggering
[7]	SOURCE7	*1* when a source7 event occurs. This event uses an *EventSourceFlex* form of triggering
[8]	SOURCE8	*1* when a source8 event occurs. This event uses an *EventSourceFlex* form of triggering
[9]	SOURCE9	*1* when a source9 event occurs. This event uses an *EventSourceFlex* form of triggering
[10]	SOURCE10	*1* when a source10 event occurs. This event uses an *EventSourceFlex* form of triggering
[11]	SOURCE11	*1* when a source11 event occurs. This event uses an *EventSourceFlex* form of triggering
[12]	SOURCE12	*1* when a source12 event occurs. This event uses an *EventSourceFlex* form of triggering
[13]	SOURCE13	*1* when a source13 event occurs. This event uses an *EventSourceFlex* form of triggering
[14]	SOURCE14	*1* when a source14 event occurs. This event uses an *EventSourceFlex* form of triggering
[15]	SOURCE15	*1* when a source15 event occurs. This event uses an *EventSourceFlex* form of triggering
[16]	SOURCE16	*1* when a source16 event occurs. This event uses an *EventSourceFlex* form of triggering
[17]	SOURCE17	*1* when a source17 event occurs. This event uses an *EventSourceFlex* form of triggering
[18]	SOURCE18	*1* when a source18 event occurs. This event uses an *EventSourceFlex* form of triggering
[19]	SOURCE19	*1* when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

IRQARRAY5_EV_ENABLE

Address: 0x58013000 + 0xc = 0x5801300c

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

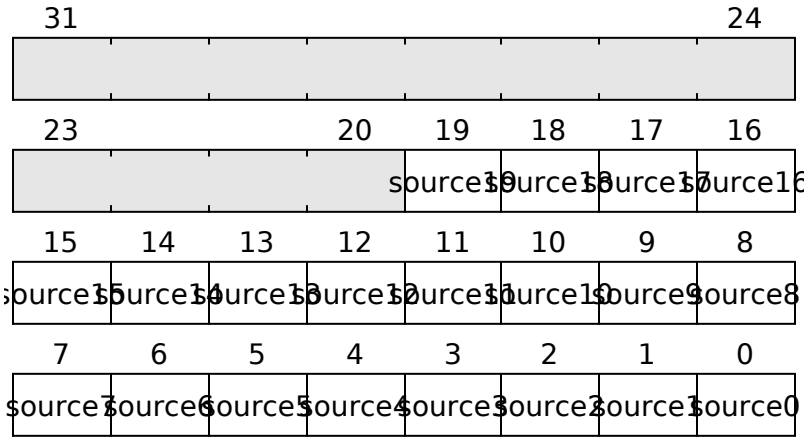

Fig. 89: IRQARRAY5_EV_ENABLE

Field	Name	Description
[0]	SOURCE0	Write a 1 to enable the `source0` Event
[1]	SOURCE1	Write a 1 to enable the `source1` Event
[2]	SOURCE2	Write a 1 to enable the `source2` Event
[3]	SOURCE3	Write a 1 to enable the `source3` Event
[4]	SOURCE4	Write a 1 to enable the `source4` Event
[5]	SOURCE5	Write a 1 to enable the `source5` Event
[6]	SOURCE6	Write a 1 to enable the `source6` Event
[7]	SOURCE7	Write a 1 to enable the `source7` Event
[8]	SOURCE8	Write a 1 to enable the `source8` Event
[9]	SOURCE9	Write a 1 to enable the `source9` Event
[10]	SOURCE10	Write a 1 to enable the `source10` Event
[11]	SOURCE11	Write a 1 to enable the `source11` Event
[12]	SOURCE12	Write a 1 to enable the `source12` Event
[13]	SOURCE13	Write a 1 to enable the `source13` Event
[14]	SOURCE14	Write a 1 to enable the `source14` Event
[15]	SOURCE15	Write a 1 to enable the `source15` Event
[16]	SOURCE16	Write a 1 to enable the `source16` Event
[17]	SOURCE17	Write a 1 to enable the `source17` Event
[18]	SOURCE18	Write a 1 to enable the `source18` Event
[19]	SOURCE19	Write a 1 to enable the `source19` Event

2.21 IRQARRAY6

IrqArray provides a large bank of interrupts for SoC integration. It is different from e.g. the NVIC or CLINT in that the register bank is structured along page boundaries, so that the interrupt handler CSRs can be owned by a specific virtual memory process, instead of bouncing through a common handler and forcing an inter-process message to be generated to route interrupts to their final destination.

The incoming interrupt signals are assumed to be synchronized to *aclk*.

Priorities are enforced entirely through software; the handler must read the *pending* bits and decide which ones should be handled first.

The *EventSource* is an *EventSourceFlex* which can handle pulses and levels, as well as software triggers.

The interrupt pending bit is latched when the trigger goes high, and stays high until software clears the event. The trigger takes precedence over clearing, so if the interrupt source is not cleared prior to clearing the interrupt pending bit, the interrupt will trigger again.

status reflects the instantaneous value of the trigger.

A separate input line is provided so that software can induce an interrupt by writing to a soft-trigger bit.

2.21.1 Register Listing for IRQARRAY6

Register	Address
IRQARRAY6_EV_SOFT	*0x58014000*
IRQARRAY6_EV_STATUS	*0x58014004*
IRQARRAY6_EV_PENDING	*0x58014008*
IRQARRAY6_EV_ENABLE	*0x5801400c*

IRQARRAY6_EV_SOFT

Address: 0x58014000 + 0x0 = 0x58014000

Software interrupt trigger register.

Bits set to *1* will trigger an interrupt. Interrupts trigger on write, but the value will persist in the register, allowing software to determine if a software interrupt was triggered by reading back the register.

Software is responsible for clearing the register to 0.

Repeated *1* writes without clearing will still trigger an interrupt.

Fig. 90: IRQARRAY6_EV_SOFT

Field	Name	Description
[19:0]	TRIGGER	Writing a 1 to this bit triggers the function.

IRQARRAY6_EV_STATUS

Address: 0x58014000 + 0x4 = 0x58014004

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

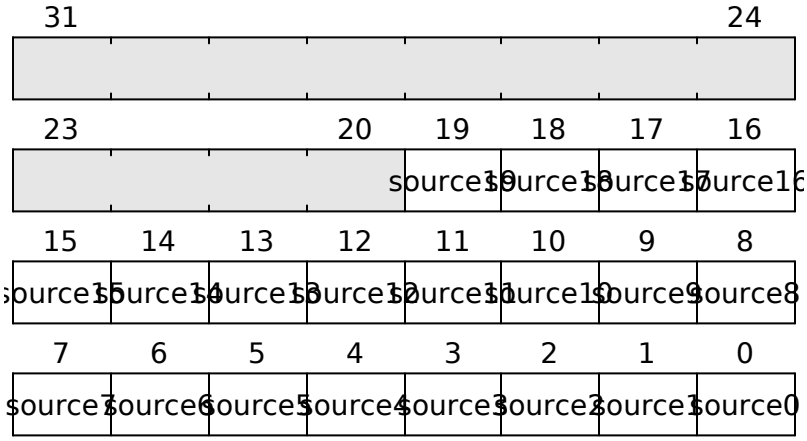

Fig. 91: IRQARRAY6_EV_STATUS

Field	Name	Description
[0]	SOURCE0	Level of the source0 event
[1]	SOURCE1	Level of the source1 event
[2]	SOURCE2	Level of the source2 event
[3]	SOURCE3	Level of the source3 event
[4]	SOURCE4	Level of the source4 event
[5]	SOURCE5	Level of the source5 event
[6]	SOURCE6	Level of the source6 event
[7]	SOURCE7	Level of the source7 event
[8]	SOURCE8	Level of the source8 event
[9]	SOURCE9	Level of the source9 event
[10]	SOURCE10	Level of the source10 event
[11]	SOURCE11	Level of the source11 event
[12]	SOURCE12	Level of the source12 event
[13]	SOURCE13	Level of the source13 event
[14]	SOURCE14	Level of the source14 event
[15]	SOURCE15	Level of the source15 event
[16]	SOURCE16	Level of the source16 event
[17]	SOURCE17	Level of the source17 event
[18]	SOURCE18	Level of the source18 event
[19]	SOURCE19	Level of the source19 event

IRQARRAY6_EV_PENDING

Address: 0x58014000 + 0x8 = 0x58014008

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

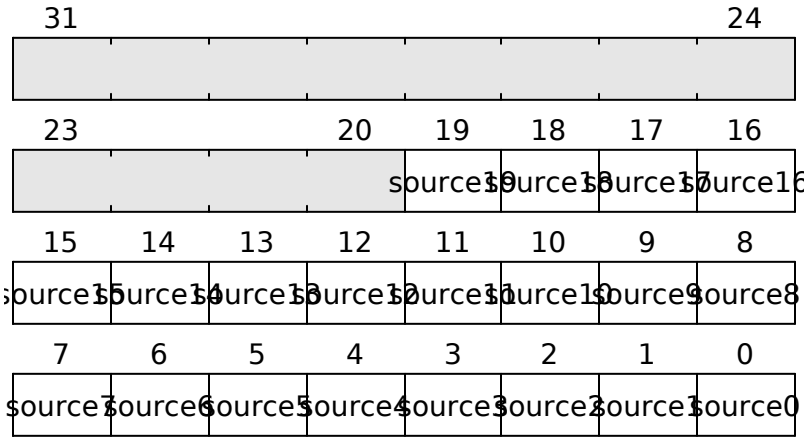

Fig. 92: IRQARRAY6_EV_PENDING

Field	Name	Description
[0]	SOURCE0	*1* when a source0 event occurs. This event uses an *EventSourceFlex* form of triggering
[1]	SOURCE1	*1* when a source1 event occurs. This event uses an *EventSourceFlex* form of triggering
[2]	SOURCE2	*1* when a source2 event occurs. This event uses an *EventSourceFlex* form of triggering
[3]	SOURCE3	*1* when a source3 event occurs. This event uses an *EventSourceFlex* form of triggering
[4]	SOURCE4	*1* when a source4 event occurs. This event uses an *EventSourceFlex* form of triggering
[5]	SOURCE5	*1* when a source5 event occurs. This event uses an *EventSourceFlex* form of triggering
[6]	SOURCE6	*1* when a source6 event occurs. This event uses an *EventSourceFlex* form of triggering
[7]	SOURCE7	*1* when a source7 event occurs. This event uses an *EventSourceFlex* form of triggering
[8]	SOURCE8	*1* when a source8 event occurs. This event uses an *EventSourceFlex* form of triggering
[9]	SOURCE9	*1* when a source9 event occurs. This event uses an *EventSourceFlex* form of triggering
[10]	SOURCE10	*1* when a source10 event occurs. This event uses an *EventSourceFlex* form of triggering
[11]	SOURCE11	*1* when a source11 event occurs. This event uses an *EventSourceFlex* form of triggering
[12]	SOURCE12	*1* when a source12 event occurs. This event uses an *EventSourceFlex* form of triggering
[13]	SOURCE13	*1* when a source13 event occurs. This event uses an *EventSourceFlex* form of triggering
[14]	SOURCE14	*1* when a source14 event occurs. This event uses an *EventSourceFlex* form of triggering
[15]	SOURCE15	*1* when a source15 event occurs. This event uses an *EventSourceFlex* form of triggering
[16]	SOURCE16	*1* when a source16 event occurs. This event uses an *EventSourceFlex* form of triggering
[17]	SOURCE17	*1* when a source17 event occurs. This event uses an *EventSourceFlex* form of triggering
[18]	SOURCE18	*1* when a source18 event occurs. This event uses an *EventSourceFlex* form of triggering
[19]	SOURCE19	*1* when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

IRQARRAY6_EV_ENABLE

Address: 0x58014000 + 0xc = 0x5801400c

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

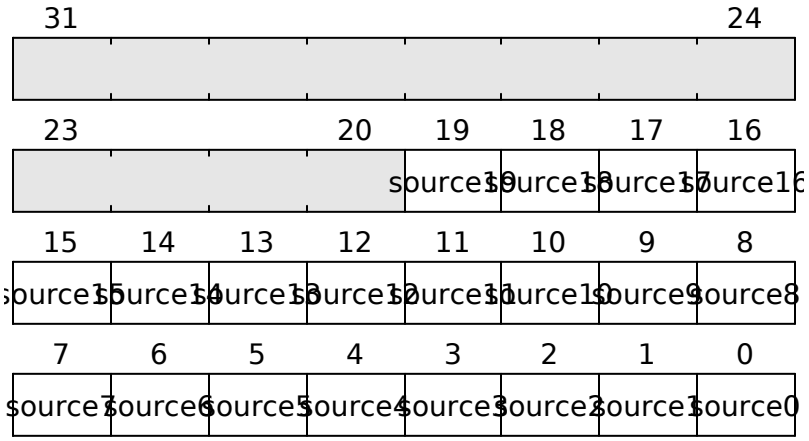

Fig. 93: IRQARRAY6_EV_ENABLE

Field	Name	Description
[0]	SOURCE0	Write a 1 to enable the `source0` Event
[1]	SOURCE1	Write a 1 to enable the `source1` Event
[2]	SOURCE2	Write a 1 to enable the `source2` Event
[3]	SOURCE3	Write a 1 to enable the `source3` Event
[4]	SOURCE4	Write a 1 to enable the `source4` Event
[5]	SOURCE5	Write a 1 to enable the `source5` Event
[6]	SOURCE6	Write a 1 to enable the `source6` Event
[7]	SOURCE7	Write a 1 to enable the `source7` Event
[8]	SOURCE8	Write a 1 to enable the `source8` Event
[9]	SOURCE9	Write a 1 to enable the `source9` Event
[10]	SOURCE10	Write a 1 to enable the `source10` Event
[11]	SOURCE11	Write a 1 to enable the `source11` Event
[12]	SOURCE12	Write a 1 to enable the `source12` Event
[13]	SOURCE13	Write a 1 to enable the `source13` Event
[14]	SOURCE14	Write a 1 to enable the `source14` Event
[15]	SOURCE15	Write a 1 to enable the `source15` Event
[16]	SOURCE16	Write a 1 to enable the `source16` Event
[17]	SOURCE17	Write a 1 to enable the `source17` Event
[18]	SOURCE18	Write a 1 to enable the `source18` Event
[19]	SOURCE19	Write a 1 to enable the `source19` Event

2.22 IRQARRAY7

IrqArray provides a large bank of interrupts for SoC integration. It is different from e.g. the NVIC or CLINT in that the register bank is structured along page boundaries, so that the interrupt handler CSRs can be owned by a specific virtual memory process, instead of bouncing through a common handler and forcing an inter-process message to be generated to route interrupts to their final destination.

The incoming interrupt signals are assumed to be synchronized to *aclk*.

Priorities are enforced entirely through software; the handler must read the *pending* bits and decide which ones should be handled first.

The *EventSource* is an *EventSourceFlex* which can handle pulses and levels, as well as software triggers.

The interrupt pending bit is latched when the trigger goes high, and stays high until software clears the event. The trigger takes precedence over clearing, so if the interrupt source is not cleared prior to clearing the interrupt pending bit, the interrupt will trigger again.

status reflects the instantaneous value of the trigger.

A separate input line is provided so that software can induce an interrupt by writing to a soft-trigger bit.

2.22.1 Register Listing for IRQARRAY7

Register	Address
IRQARRAY7_EV_SOFT	*0x58015000*
IRQARRAY7_EV_STATUS	*0x58015004*
IRQARRAY7_EV_PENDING	*0x58015008*
IRQARRAY7_EV_ENABLE	*0x5801500c*

IRQARRAY7_EV_SOFT

Address: 0x58015000 + 0x0 = 0x58015000

Software interrupt trigger register.

Bits set to *1* will trigger an interrupt. Interrupts trigger on write, but the value will persist in the register, allowing software to determine if a software interrupt was triggered by reading back the register.

Software is responsible for clearing the register to 0.

Repeated *1* writes without clearing will still trigger an interrupt.

Fig. 94: IRQARRAY7_EV_SOFT

Field	Name	Description
[19:0]	TRIGGER	Writing a 1 to this bit triggers the function.

IRQARRAY7_EV_STATUS

Address: 0x58015000 + 0x4 = 0x58015004

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

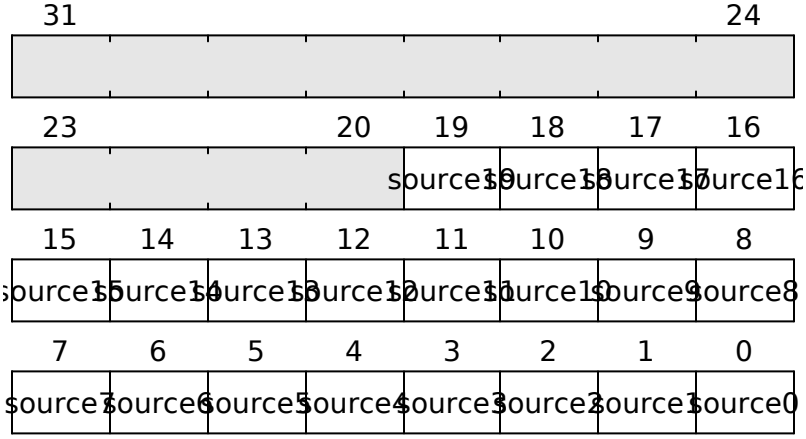

Fig. 95: IRQARRAY7_EV_STATUS

Field	Name	Description
[0]	SOURCE0	Level of the source0 event
[1]	SOURCE1	Level of the source1 event
[2]	SOURCE2	Level of the source2 event
[3]	SOURCE3	Level of the source3 event
[4]	SOURCE4	Level of the source4 event
[5]	SOURCE5	Level of the source5 event
[6]	SOURCE6	Level of the source6 event
[7]	SOURCE7	Level of the source7 event
[8]	SOURCE8	Level of the source8 event
[9]	SOURCE9	Level of the source9 event
[10]	SOURCE10	Level of the source10 event
[11]	SOURCE11	Level of the source11 event
[12]	SOURCE12	Level of the source12 event
[13]	SOURCE13	Level of the source13 event
[14]	SOURCE14	Level of the source14 event
[15]	SOURCE15	Level of the source15 event
[16]	SOURCE16	Level of the source16 event
[17]	SOURCE17	Level of the source17 event
[18]	SOURCE18	Level of the source18 event
[19]	SOURCE19	Level of the source19 event

IRQARRAY7_EV_PENDING

Address: 0x58015000 + 0x8 = 0x58015008

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

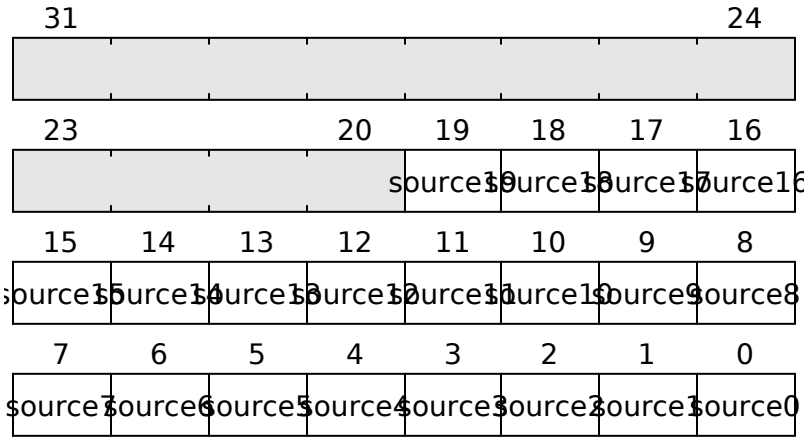

Fig. 96: IRQARRAY7_EV_PENDING

Field	Name	Description
[0]	SOURCE0	*1* when a source0 event occurs. This event uses an *EventSourceFlex* form of triggering
[1]	SOURCE1	*1* when a source1 event occurs. This event uses an *EventSourceFlex* form of triggering
[2]	SOURCE2	*1* when a source2 event occurs. This event uses an *EventSourceFlex* form of triggering
[3]	SOURCE3	*1* when a source3 event occurs. This event uses an *EventSourceFlex* form of triggering
[4]	SOURCE4	*1* when a source4 event occurs. This event uses an *EventSourceFlex* form of triggering
[5]	SOURCE5	*1* when a source5 event occurs. This event uses an *EventSourceFlex* form of triggering
[6]	SOURCE6	*1* when a source6 event occurs. This event uses an *EventSourceFlex* form of triggering
[7]	SOURCE7	*1* when a source7 event occurs. This event uses an *EventSourceFlex* form of triggering
[8]	SOURCE8	*1* when a source8 event occurs. This event uses an *EventSourceFlex* form of triggering
[9]	SOURCE9	*1* when a source9 event occurs. This event uses an *EventSourceFlex* form of triggering
[10]	SOURCE10	*1* when a source10 event occurs. This event uses an *EventSourceFlex* form of triggering
[11]	SOURCE11	*1* when a source11 event occurs. This event uses an *EventSourceFlex* form of triggering
[12]	SOURCE12	*1* when a source12 event occurs. This event uses an *EventSourceFlex* form of triggering
[13]	SOURCE13	*1* when a source13 event occurs. This event uses an *EventSourceFlex* form of triggering
[14]	SOURCE14	*1* when a source14 event occurs. This event uses an *EventSourceFlex* form of triggering
[15]	SOURCE15	*1* when a source15 event occurs. This event uses an *EventSourceFlex* form of triggering
[16]	SOURCE16	*1* when a source16 event occurs. This event uses an *EventSourceFlex* form of triggering
[17]	SOURCE17	*1* when a source17 event occurs. This event uses an *EventSourceFlex* form of triggering
[18]	SOURCE18	*1* when a source18 event occurs. This event uses an *EventSourceFlex* form of triggering
[19]	SOURCE19	*1* when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

IRQARRAY7_EV_ENABLE

Address: 0x58015000 + 0xc = 0x5801500c

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

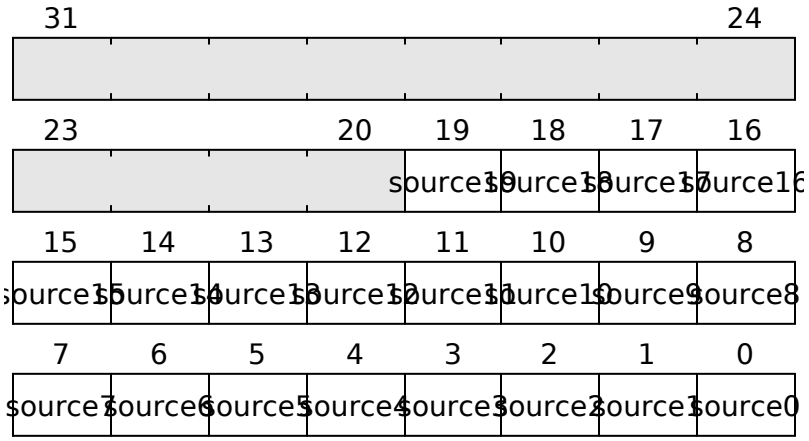

Fig. 97: IRQARRAY7_EV_ENABLE

Field	Name	Description
[0]	SOURCE0	Write a 1 to enable the `source0` Event
[1]	SOURCE1	Write a 1 to enable the `source1` Event
[2]	SOURCE2	Write a 1 to enable the `source2` Event
[3]	SOURCE3	Write a 1 to enable the `source3` Event
[4]	SOURCE4	Write a 1 to enable the `source4` Event
[5]	SOURCE5	Write a 1 to enable the `source5` Event
[6]	SOURCE6	Write a 1 to enable the `source6` Event
[7]	SOURCE7	Write a 1 to enable the `source7` Event
[8]	SOURCE8	Write a 1 to enable the `source8` Event
[9]	SOURCE9	Write a 1 to enable the `source9` Event
[10]	SOURCE10	Write a 1 to enable the `source10` Event
[11]	SOURCE11	Write a 1 to enable the `source11` Event
[12]	SOURCE12	Write a 1 to enable the `source12` Event
[13]	SOURCE13	Write a 1 to enable the `source13` Event
[14]	SOURCE14	Write a 1 to enable the `source14` Event
[15]	SOURCE15	Write a 1 to enable the `source15` Event
[16]	SOURCE16	Write a 1 to enable the `source16` Event
[17]	SOURCE17	Write a 1 to enable the `source17` Event
[18]	SOURCE18	Write a 1 to enable the `source18` Event
[19]	SOURCE19	Write a 1 to enable the `source19` Event

2.23 IRQARRAY8

IrqArray provides a large bank of interrupts for SoC integration. It is different from e.g. the NVIC or CLINT in that the register bank is structured along page boundaries, so that the interrupt handler CSRs can be owned by a specific virtual memory process, instead of bouncing through a common handler and forcing an inter-process message to be generated to route interrupts to their final destination.

The incoming interrupt signals are assumed to be synchronized to *aclk*.

Priorities are enforced entirely through software; the handler must read the *pending* bits and decide which ones should be handled first.

The *EventSource* is an *EventSourceFlex* which can handle pulses and levels, as well as software triggers.

The interrupt pending bit is latched when the trigger goes high, and stays high until software clears the event. The trigger takes precedence over clearing, so if the interrupt source is not cleared prior to clearing the interrupt pending bit, the interrupt will trigger again.

status reflects the instantaneous value of the trigger.

A separate input line is provided so that software can induce an interrupt by writing to a soft-trigger bit.

2.23.1 Register Listing for IRQARRAY8

Register	Address
IRQARRAY8_EV_SOFT	*0x58016000*
IRQARRAY8_EV_STATUS	*0x58016004*
IRQARRAY8_EV_PENDING	*0x58016008*
IRQARRAY8_EV_ENABLE	*0x5801600c*

IRQARRAY8_EV_SOFT

Address: 0x58016000 + 0x0 = 0x58016000

Software interrupt trigger register.

Bits set to *1* will trigger an interrupt. Interrupts trigger on write, but the value will persist in the register, allowing software to determine if a software interrupt was triggered by reading back the register.

Software is responsible for clearing the register to 0.

Repeated *1* writes without clearing will still trigger an interrupt.

Fig. 98: IRQARRAY8_EV_SOFT

Field	Name	Description
[19:0]	TRIGGER	Writing a 1 to this bit triggers the function.

IRQARRAY8_EV_STATUS

Address: 0x58016000 + 0x4 = 0x58016004

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

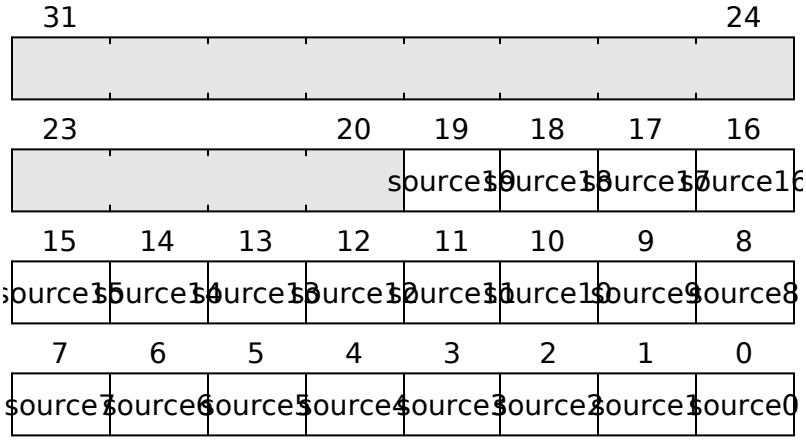

Fig. 99: IRQARRAY8_EV_STATUS

Field	Name	Description
[0]	SOURCE0	Level of the source0 event
[1]	SOURCE1	Level of the source1 event
[2]	SOURCE2	Level of the source2 event
[3]	SOURCE3	Level of the source3 event
[4]	SOURCE4	Level of the source4 event
[5]	SOURCE5	Level of the source5 event
[6]	SOURCE6	Level of the source6 event
[7]	SOURCE7	Level of the source7 event
[8]	SOURCE8	Level of the source8 event
[9]	SOURCE9	Level of the source9 event
[10]	SOURCE10	Level of the source10 event
[11]	SOURCE11	Level of the source11 event
[12]	SOURCE12	Level of the source12 event
[13]	SOURCE13	Level of the source13 event
[14]	SOURCE14	Level of the source14 event
[15]	SOURCE15	Level of the source15 event
[16]	SOURCE16	Level of the source16 event
[17]	SOURCE17	Level of the source17 event
[18]	SOURCE18	Level of the source18 event
[19]	SOURCE19	Level of the source19 event

IRQARRAY8_EV_PENDING

Address: 0x58016000 + 0x8 = 0x58016008

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

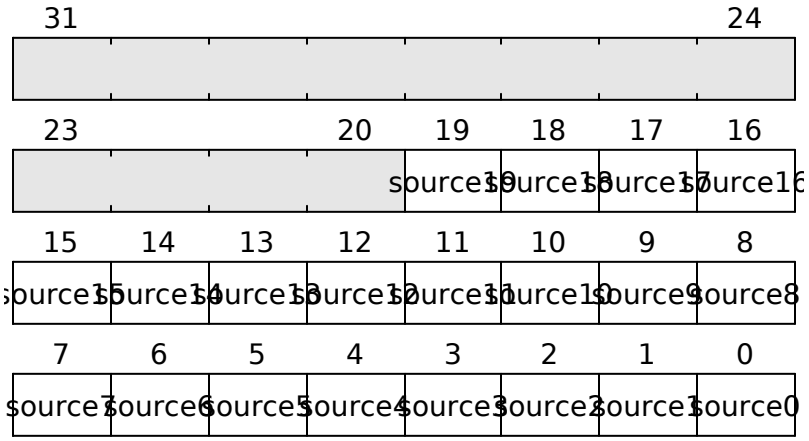

Fig. 100: IRQARRAY8_EV_PENDING

Field	Name	Description
[0]	SOURCE0	*1* when a source0 event occurs. This event uses an *EventSourceFlex* form of triggering
[1]	SOURCE1	*1* when a source1 event occurs. This event uses an *EventSourceFlex* form of triggering
[2]	SOURCE2	*1* when a source2 event occurs. This event uses an *EventSourceFlex* form of triggering
[3]	SOURCE3	*1* when a source3 event occurs. This event uses an *EventSourceFlex* form of triggering
[4]	SOURCE4	*1* when a source4 event occurs. This event uses an *EventSourceFlex* form of triggering
[5]	SOURCE5	*1* when a source5 event occurs. This event uses an *EventSourceFlex* form of triggering
[6]	SOURCE6	*1* when a source6 event occurs. This event uses an *EventSourceFlex* form of triggering
[7]	SOURCE7	*1* when a source7 event occurs. This event uses an *EventSourceFlex* form of triggering
[8]	SOURCE8	*1* when a source8 event occurs. This event uses an *EventSourceFlex* form of triggering
[9]	SOURCE9	*1* when a source9 event occurs. This event uses an *EventSourceFlex* form of triggering
[10]	SOURCE10	*1* when a source10 event occurs. This event uses an *EventSourceFlex* form of triggering
[11]	SOURCE11	*1* when a source11 event occurs. This event uses an *EventSourceFlex* form of triggering
[12]	SOURCE12	*1* when a source12 event occurs. This event uses an *EventSourceFlex* form of triggering
[13]	SOURCE13	*1* when a source13 event occurs. This event uses an *EventSourceFlex* form of triggering
[14]	SOURCE14	*1* when a source14 event occurs. This event uses an *EventSourceFlex* form of triggering
[15]	SOURCE15	*1* when a source15 event occurs. This event uses an *EventSourceFlex* form of triggering
[16]	SOURCE16	*1* when a source16 event occurs. This event uses an *EventSourceFlex* form of triggering
[17]	SOURCE17	*1* when a source17 event occurs. This event uses an *EventSourceFlex* form of triggering
[18]	SOURCE18	*1* when a source18 event occurs. This event uses an *EventSourceFlex* form of triggering
[19]	SOURCE19	*1* when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

IRQARRAY8_EV_ENABLE

Address: 0x58016000 + 0xc = 0x5801600c

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

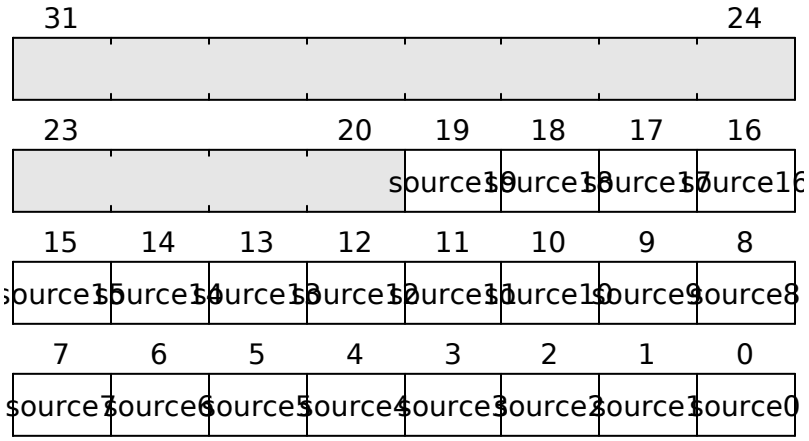

Fig. 101: IRQARRAY8_EV_ENABLE

Field	Name	Description
[0]	SOURCE0	Write a 1 to enable the `source0` Event
[1]	SOURCE1	Write a 1 to enable the `source1` Event
[2]	SOURCE2	Write a 1 to enable the `source2` Event
[3]	SOURCE3	Write a 1 to enable the `source3` Event
[4]	SOURCE4	Write a 1 to enable the `source4` Event
[5]	SOURCE5	Write a 1 to enable the `source5` Event
[6]	SOURCE6	Write a 1 to enable the `source6` Event
[7]	SOURCE7	Write a 1 to enable the `source7` Event
[8]	SOURCE8	Write a 1 to enable the `source8` Event
[9]	SOURCE9	Write a 1 to enable the `source9` Event
[10]	SOURCE10	Write a 1 to enable the `source10` Event
[11]	SOURCE11	Write a 1 to enable the `source11` Event
[12]	SOURCE12	Write a 1 to enable the `source12` Event
[13]	SOURCE13	Write a 1 to enable the `source13` Event
[14]	SOURCE14	Write a 1 to enable the `source14` Event
[15]	SOURCE15	Write a 1 to enable the `source15` Event
[16]	SOURCE16	Write a 1 to enable the `source16` Event
[17]	SOURCE17	Write a 1 to enable the `source17` Event
[18]	SOURCE18	Write a 1 to enable the `source18` Event
[19]	SOURCE19	Write a 1 to enable the `source19` Event

2.24 IRQARRAY9

IrqArray provides a large bank of interrupts for SoC integration. It is different from e.g. the NVIC or CLINT in that the register bank is structured along page boundaries, so that the interrupt handler CSRs can be owned by a specific virtual memory process, instead of bouncing through a common handler and forcing an inter-process message to be generated to route interrupts to their final destination.

The incoming interrupt signals are assumed to be synchronized to *aclk*.

Priorities are enforced entirely through software; the handler must read the *pending* bits and decide which ones should be handled first.

The *EventSource* is an *EventSourceFlex* which can handle pulses and levels, as well as software triggers.

The interrupt pending bit is latched when the trigger goes high, and stays high until software clears the event. The trigger takes precedence over clearing, so if the interrupt source is not cleared prior to clearing the interrupt pending bit, the interrupt will trigger again.

status reflects the instantaneous value of the trigger.

A separate input line is provided so that software can induce an interrupt by writing to a soft-trigger bit.

2.24.1 Register Listing for IRQARRAY9

Register	Address
IRQARRAY9_EV_SOFT	*0x58017000*
IRQARRAY9_EV_STATUS	*0x58017004*
IRQARRAY9_EV_PENDING	*0x58017008*
IRQARRAY9_EV_ENABLE	*0x5801700c*

IRQARRAY9_EV_SOFT

Address: 0x58017000 + 0x0 = 0x58017000

Software interrupt trigger register.

Bits set to *1* will trigger an interrupt. Interrupts trigger on write, but the value will persist in the register, allowing software to determine if a software interrupt was triggered by reading back the register.

Software is responsible for clearing the register to 0.

Repeated *1* writes without clearing will still trigger an interrupt.

Fig. 102: IRQARRAY9_EV_SOFT

Field	Name	Description
[19:0]	TRIGGER	Writing a 1 to this bit triggers the function.

IRQARRAY9_EV_STATUS

Address: 0x58017000 + 0x4 = 0x58017004

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

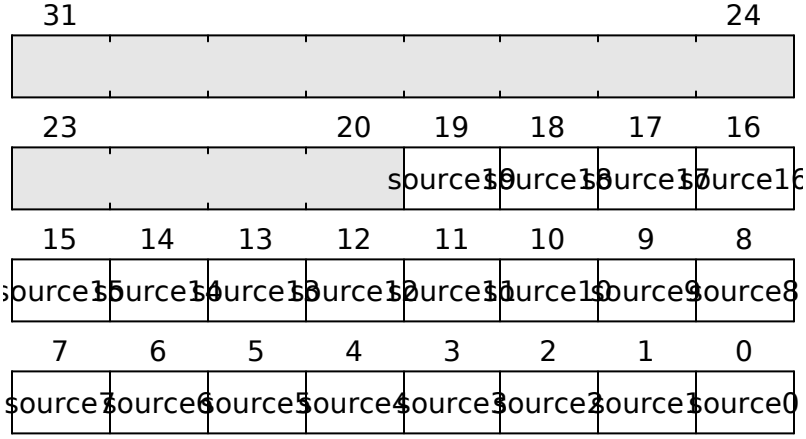

Fig. 103: IRQARRAY9_EV_STATUS

Field	Name	Description
[0]	SOURCE0	Level of the source0 event
[1]	SOURCE1	Level of the source1 event
[2]	SOURCE2	Level of the source2 event
[3]	SOURCE3	Level of the source3 event
[4]	SOURCE4	Level of the source4 event
[5]	SOURCE5	Level of the source5 event
[6]	SOURCE6	Level of the source6 event
[7]	SOURCE7	Level of the source7 event
[8]	SOURCE8	Level of the source8 event
[9]	SOURCE9	Level of the source9 event
[10]	SOURCE10	Level of the source10 event
[11]	SOURCE11	Level of the source11 event
[12]	SOURCE12	Level of the source12 event
[13]	SOURCE13	Level of the source13 event
[14]	SOURCE14	Level of the source14 event
[15]	SOURCE15	Level of the source15 event
[16]	SOURCE16	Level of the source16 event
[17]	SOURCE17	Level of the source17 event
[18]	SOURCE18	Level of the source18 event
[19]	SOURCE19	Level of the source19 event

IRQARRAY9_EV_PENDING

Address: 0x58017000 + 0x8 = 0x58017008

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

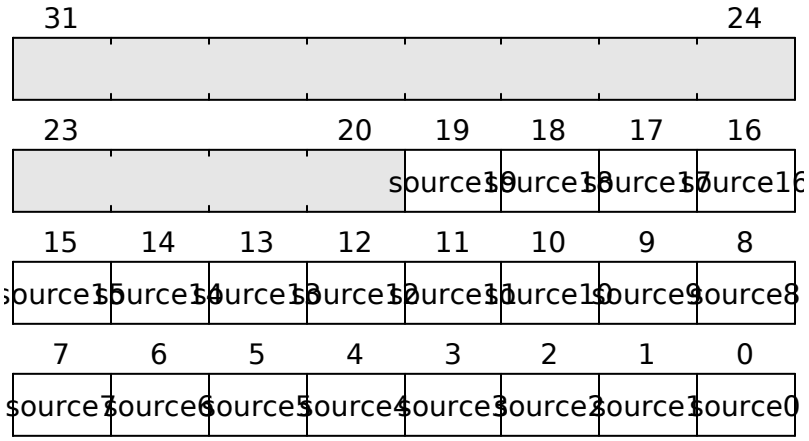

Fig. 104: IRQARRAY9_EV_PENDING

Field	Name	Description
[0]	SOURCE0	*1* when a source0 event occurs. This event uses an *EventSourceFlex* form of triggering
[1]	SOURCE1	*1* when a source1 event occurs. This event uses an *EventSourceFlex* form of triggering
[2]	SOURCE2	*1* when a source2 event occurs. This event uses an *EventSourceFlex* form of triggering
[3]	SOURCE3	*1* when a source3 event occurs. This event uses an *EventSourceFlex* form of triggering
[4]	SOURCE4	*1* when a source4 event occurs. This event uses an *EventSourceFlex* form of triggering
[5]	SOURCE5	*1* when a source5 event occurs. This event uses an *EventSourceFlex* form of triggering
[6]	SOURCE6	*1* when a source6 event occurs. This event uses an *EventSourceFlex* form of triggering
[7]	SOURCE7	*1* when a source7 event occurs. This event uses an *EventSourceFlex* form of triggering
[8]	SOURCE8	*1* when a source8 event occurs. This event uses an *EventSourceFlex* form of triggering
[9]	SOURCE9	*1* when a source9 event occurs. This event uses an *EventSourceFlex* form of triggering
[10]	SOURCE10	*1* when a source10 event occurs. This event uses an *EventSourceFlex* form of triggering
[11]	SOURCE11	*1* when a source11 event occurs. This event uses an *EventSourceFlex* form of triggering
[12]	SOURCE12	*1* when a source12 event occurs. This event uses an *EventSourceFlex* form of triggering
[13]	SOURCE13	*1* when a source13 event occurs. This event uses an *EventSourceFlex* form of triggering
[14]	SOURCE14	*1* when a source14 event occurs. This event uses an *EventSourceFlex* form of triggering
[15]	SOURCE15	*1* when a source15 event occurs. This event uses an *EventSourceFlex* form of triggering
[16]	SOURCE16	*1* when a source16 event occurs. This event uses an *EventSourceFlex* form of triggering
[17]	SOURCE17	*1* when a source17 event occurs. This event uses an *EventSourceFlex* form of triggering
[18]	SOURCE18	*1* when a source18 event occurs. This event uses an *EventSourceFlex* form of triggering
[19]	SOURCE19	*1* when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

IRQARRAY9_EV_ENABLE

Address: 0x58017000 + 0xc = 0x5801700c

1 when a source19 event occurs. This event uses an *EventSourceFlex* form of triggering

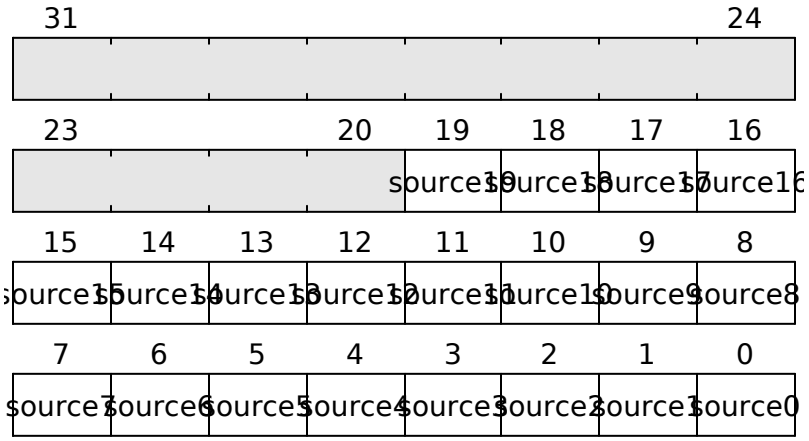

Fig. 105: IRQARRAY9_EV_ENABLE

Field	Name	Description
[0]	SOURCE0	Write a 1 to enable the source0 Event
[1]	SOURCE1	Write a 1 to enable the source1 Event
[2]	SOURCE2	Write a 1 to enable the source2 Event
[3]	SOURCE3	Write a 1 to enable the source3 Event
[4]	SOURCE4	Write a 1 to enable the source4 Event
[5]	SOURCE5	Write a 1 to enable the source5 Event
[6]	SOURCE6	Write a 1 to enable the source6 Event
[7]	SOURCE7	Write a 1 to enable the source7 Event
[8]	SOURCE8	Write a 1 to enable the source8 Event
[9]	SOURCE9	Write a 1 to enable the source9 Event
[10]	SOURCE10	Write a 1 to enable the source10 Event
[11]	SOURCE11	Write a 1 to enable the source11 Event
[12]	SOURCE12	Write a 1 to enable the source12 Event
[13]	SOURCE13	Write a 1 to enable the source13 Event
[14]	SOURCE14	Write a 1 to enable the source14 Event
[15]	SOURCE15	Write a 1 to enable the source15 Event
[16]	SOURCE16	Write a 1 to enable the source16 Event
[17]	SOURCE17	Write a 1 to enable the source17 Event
[18]	SOURCE18	Write a 1 to enable the source18 Event
[19]	SOURCE19	Write a 1 to enable the source19 Event

2.25 MAILBOX

2.25.1 Mailbox: An inter-CPU mailbox

The *Mailbox* is a bi-directional, inter-CPU mailbox for delivering messages between CPUs without requiring shared memory.

A single message consists of a packet up to 1024 words long, where each word is 32 bits in length.

Both CPUs are considered as "peers"; each can initiate a packet at-will.

The bus signal layout is as follows:

```
layout = [
    # data going to the peer. `valid` indicates data is ready to be written;
    # `ready` acknowledges the current write
    ("w_dat", 32, DIR_M_TO_S),
    ("w_valid", 1, DIR_M_TO_S),
    ("w_ready", 1, DIR_S_TO_M),
    # Interrupt signal to peer.
    # A single pulse used to indicate when the full packet is in the FIFO.
    ("w_done", 1, DIR_M_TO_S),
    # data coming from the peer
    ("r_dat", 32, DIR_S_TO_M),
    ("r_valid", 1, DIR_S_TO_M),
    ("r_ready", 1, DIR_M_TO_S),
    # Interrupt signal from peer.
    # A single pulse used to indicate when the full packet is in the FIFO.
    ("r_done", 1, DIR_S_TO_M),
```

(continues on next page)

(continued from previous page)

```
    # Bi-directional sync signal. This can be used at any time to recover the protocol
    # to a known state.
    # The signal is cross-wired, e.g. `w_abort` on one peer connects to `r_abort` on
    # the other. Either peer can assert `w_abort`, and it must stay asserted until
    # `r_abort` is pulsed to acknowledge the abort.
    # Asserting `w_abort` immediately clears the sender's FIFO, and blocks new data
    # from being loaded until `r_abort` is asserted.
    # In the case that both happen to simultaneously assert `w_abort`,
    # the protocol completes in one cycle.
    ("w_abort", 1, DIR_M_TO_S),
    ("r_abort", 1, DIR_S_TO_M),
]
```

2.25.2 Data Transfer Protocol

The protocol has two levels, one at a MAC level, and one at an APP level.

The MAC level protocol controls synchronization of data transfer, and the transfer of single, fully-formed packets between the devices. The MAC protocol is implemented by this hardware block.

The APP protocol is managed by the operating system, and can be considered advisory as just one of many ways to use this system to communicate between CPUs. However, it helps to ground the protocol in an APP framework as some details of the MAC impact the APP framework, especially around synchronization and conflict avoidance.

Each peer has a channel to write data to the other peer, using 32 bits *dat*, one *valid* to indicate when data is available, and *ready* to indicate when the data has been latched by the corresponding peer's hardware FIFO. Generally, *valid*/*ready* is managed exclusively by hardware state machines and the host CPUs are not aware of these signals; they mainly exist to avoid overflowing the FIFO in the case that one is pipelining multiple packets through the interface.

There is an additional *done* signal which is asserted for exactly one cycle, and it indicates to the other peer that the sender has finished writing all the data for a given packet. The *done* signal is provided so that the corresponding peer does not need to busy-monitor the FIFO depth.

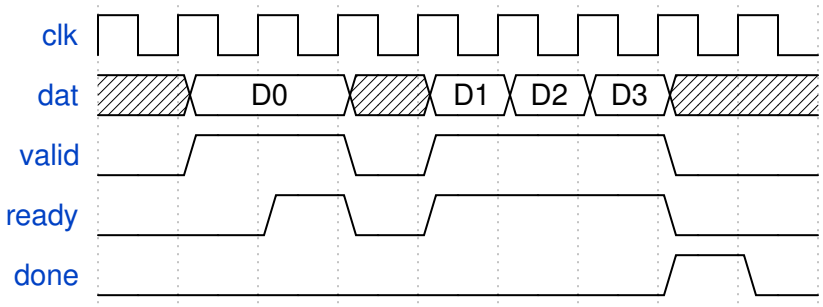

Fig. 106: Sending four words of data, followed by a *done*.

The above example shows a packet with a length of four words being transmitted. The first word takes an extra cycle to be acknowledged; the remaining three are immediately accepted. The *done* signal could come as early as simultaneously with the last *ready*, but in practice it comes a couple cycles later since it would be triggered by a write from the CPU to the *done* register.

The data transfer protocol is symmetric across the peers.

2.25.3 Abort Protocol

The abort protocol is used to recover the protocol to a known state: all FIFOs empty, and both hosts state machines in an idle state. This is accomplished by cross-wiring *w_abort* on the sending peer to *r_abort* on the corresponding peer. Either peer can assert *w_abort*, and it must stay asserted until *r_abort* is pulsed to acknowledged the abort condition. At the conclusion of the protocol, both FIFOs are empty and their protocol state machines are idle.

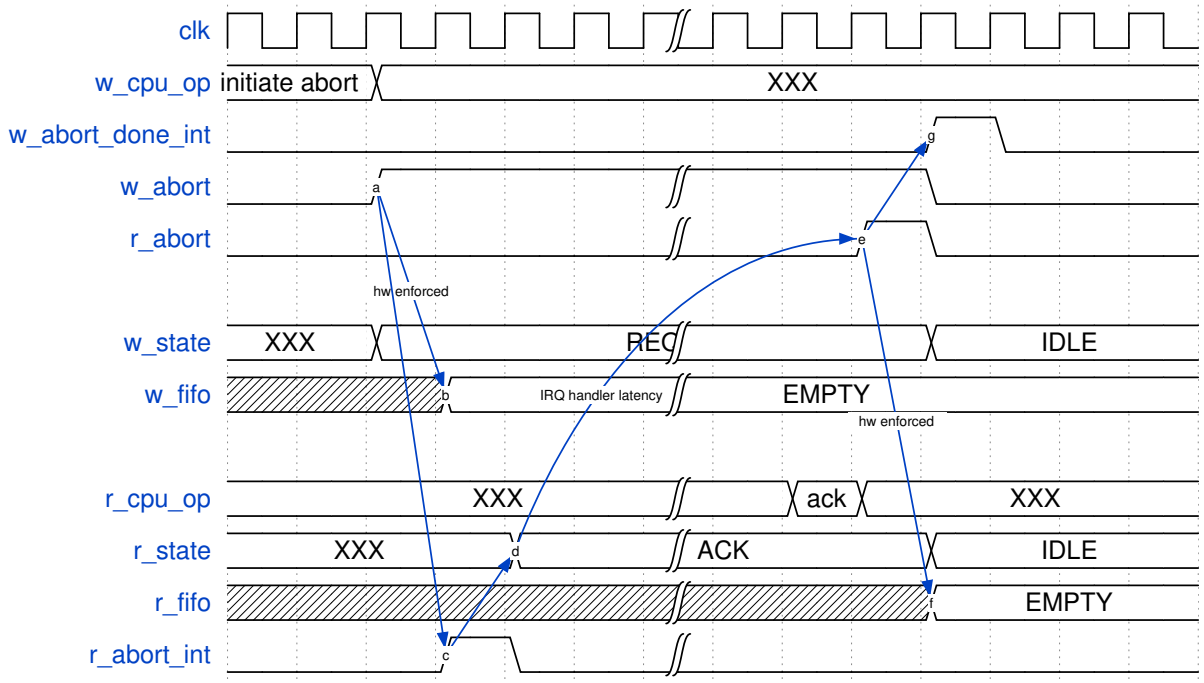

Fig. 107: Normal abort

In the diagram above, the initiating peer is the *w_* signal set, and the corresponding peer is the *r_* signal set. Here, the *w_* CPU issues a write operation by writing *1* to the *control* CSR's *abort* bit. This results in *w_abort* being asserted and held, while simultaneously both the receive and send FIFOs being cleared and refusing to accept any further data. The assertion of *w_abort* is received by the corresponding peer, which triggers an interrupt (rendered as a single pulse *r_abort_int*; but the *pending* bit is sticky until cleared).

The link stays in this state until the receiver's main loop or IRQ handler runs and acknowledges the abort condition by writing to its *control* CSR *abort* bit. Note that the IRQ handler has to be written such that any in-progress operation is truly aborted. Thus, a peer's FIFO interaction code should probably be written as follows:

1. Main loop decides it needs to interact with the FIFO
2. Disable abort response IRQ
3. Interact with the FIFO
4. Re-enable abort response IRQ; at which point an IRQ would fire triggering the abort response
5. Inside the abort response IRQ, side-effect any state machine variables back to an initial state
6. Resume main loop code, which should now check & handle any residual clean-up from an abort

At this point, both sides drop their *abort* signals, both state machines return to an *IDLE* state, and all FIFOs are empty. An *abort_done* interrupt is triggered, but it may be masked and polled if the initiating CPU prefers to monitor the abort by polling.

In order to make the case work where both peers attempt to initiate an abort at the same time, the initiator guarantees that on asserting *w_abort* it is immediately ready to act on an *r_abort* pulse. This means the hardware guarantees two things:

- All FIFOs are cleared by the request
- The incoming *abort* response line is prevented from generating an interrupt

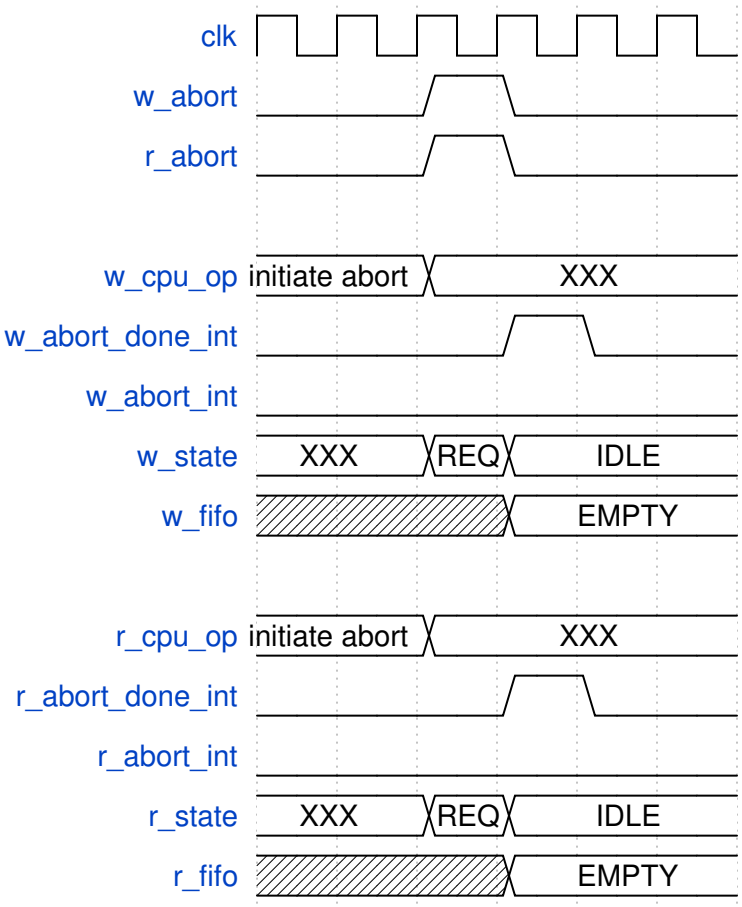

Fig. 108: Edge case: simultaneous abort

Above is the rare edge case of a cycle-perfect simultaneous abort request. It "just works", and both devices immediately transition from *REQ* -> *IDLE*, without either going through *ACK*.

Above is the more common edge case where one peer has initiated an abort, and the other is preparing to initiate at the same time, but is perhaps a cycle or two later. In this case, the late peer would have an interrupt initiated simultaneously with an abort initiation, which would result in the *HANDLER* code running, in this case, the **abort initiator** handler code (not the **abort done** handler).

A naive implementation would re-issue the *abort* bit, triggering the first peer to respond, and the two could ping-pong back and forth in an infinite cycle.

In order to break the cycle, an additional "abort acknowledged" (*abort_ack*) signal is provided, which is set in the case that the respective peer is responding to a request (thus, it would be set for both peers in the above case of the "perfectly aligned" abort request; but more typically it is cleared by the first initiator, and set for the later initiator). The

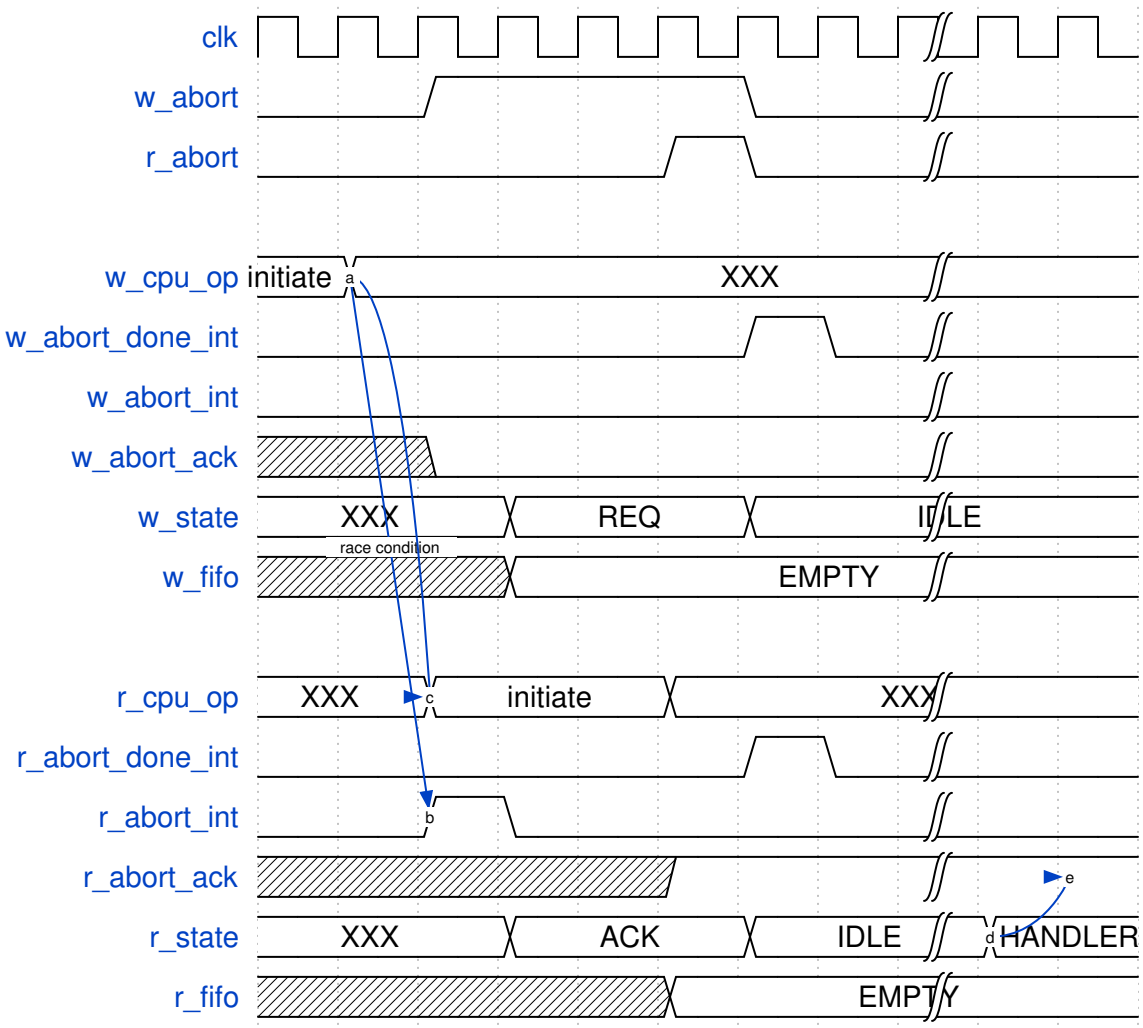

Fig. 109: Edge case: semi-simultaneous abort

abort handler thus shall always check the *abort_ack* signal, and in the case that it is set, it will not re-acknowledge a previously acknowledged abort, and avoiding an abort storm.

2.25.4 Application Protocol

The application protocol wraps a packet format around each packet. The general format of a packet is as follows:

- Word 0
  - Bit 31 - set if a response; cleared if initiating
  - Bit 30:16 - sequence number
  - Bit 15:10 - tag
  - Bit 9:0 - length in words of the packet, excluding word 0

The sequence number allows responses to occur out of order with respect to requests.

The tag encodes the operation intended by the packet. Within the tag, further meaning may be ascribed to later fields in the packet. As an example, a *tag* of 0 could indicate an RPC, and in this case *word 1* would encode the desired system call, and then the subsequent words would encode arguments to that system call. After processing the data, the response to this system call would be returned to the corresponding peer, using the same *tag* and *sequence number*, but with the *response* bit set.

Further definition of the protocol would extend from here, for example, a *send* of data could use a tag of *1*, and the response would be with the same tag and sequence number to acknowledge that the sent data was accepted, with the length field specifying the number of words that were accepted.

2.25.5 Register Listing for MAILBOX

Register	Address
MAILBOX_WDATA	*0x58018000*
MAILBOX_RDATA	*0x58018004*
MAILBOX_EV_STATUS	*0x58018008*
MAILBOX_EV_PENDING	*0x5801800c*
MAILBOX_EV_ENABLE	*0x58018010*
MAILBOX_STATUS	*0x58018014*
MAILBOX_CONTROL	*0x58018018*
MAILBOX_DONE	*0x5801801c*

MAILBOX_WDATA

Address: 0x58018000 + 0x0 = 0x58018000

Write data to outgoing FIFO.

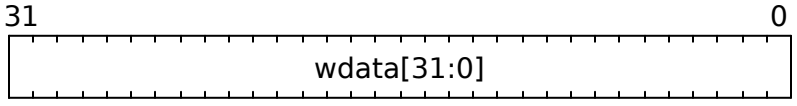

Fig. 110: MAILBOX_WDATA

MAILBOX_RDATA

Address: 0x58018000 + 0x4 = 0x58018004

Read data from incoming FIFO.

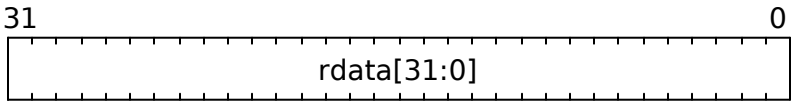

Fig. 111: MAILBOX_RDATA

MAILBOX_EV_STATUS

Address: 0x58018000 + 0x8 = 0x58018008

Triggers if either *tx_err* or *rx_err* are asserted

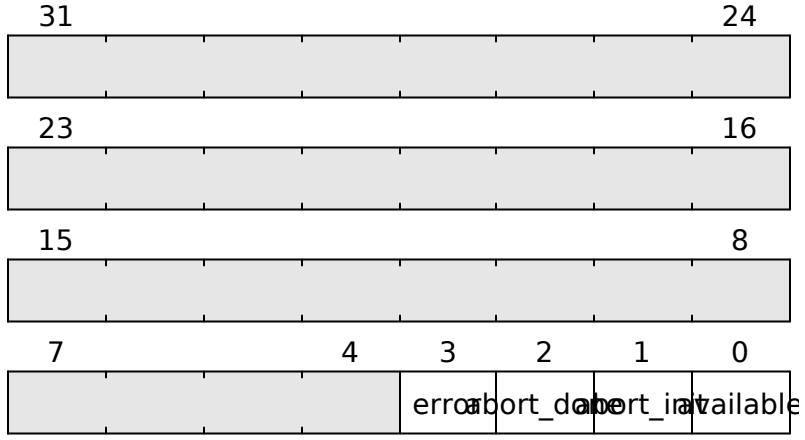

Fig. 112: MAILBOX_EV_STATUS

Field	Name	Description
[0]	AVAILABLE	Level of the `available` event
[1]	ABORT_INIT	Level of the `abort_init` event
[2]	ABORT_DONE	Level of the `abort_done` event
[3]	ERROR	Level of the `error` event

MAILBOX_EV_PENDING

Address: 0x58018000 + 0xc = 0x5801800c

Triggers if either *tx_err* or *rx_err* are asserted

Field	Name	Description
[0]	AVAILABLE	Triggers when the *done* signal was asserted by the corresponding peer
[1]	ABORT_INIT	Triggers when abort is asserted by the peer, and there is currently no abort in progress
[2]	ABORT_DONE	Triggers when a previously initiated abort is acknowledged by peer
[3]	ERROR	Triggers if either *tx_err* or *rx_err* are asserted

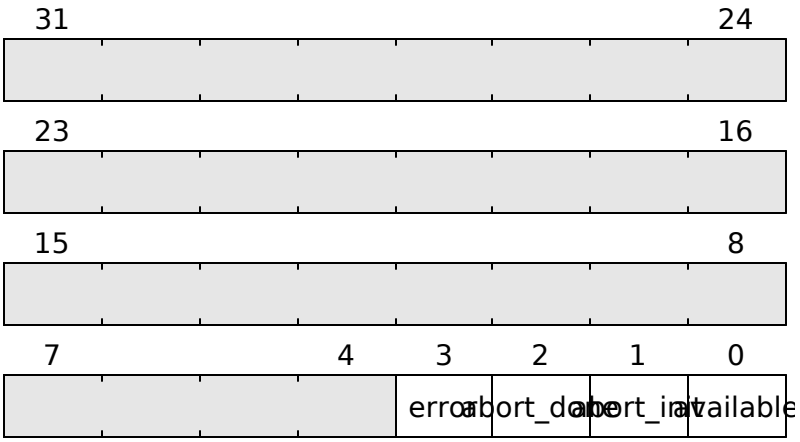

Fig. 113: MAILBOX_EV_PENDING

MAILBOX_EV_ENABLE

Address: 0x58018000 + 0x10 = 0x58018010

Triggers if either *tx_err* or *rx_err* are asserted

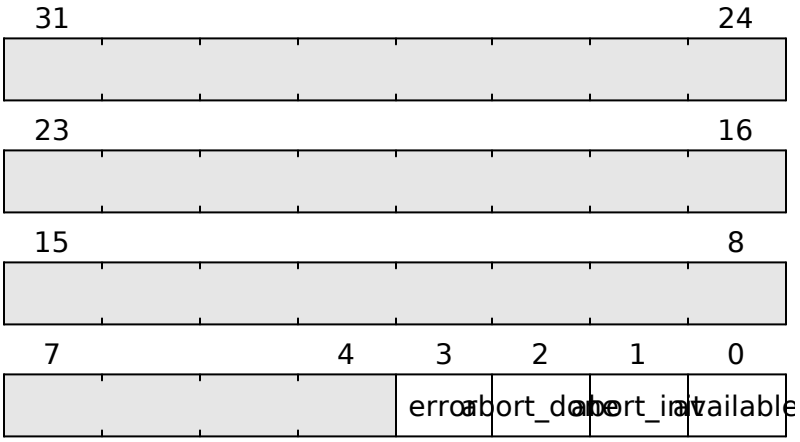

Fig. 114: MAILBOX_EV_ENABLE

Field	Name	Description
[0]	AVAILABLE	Write a 1 to enable the `available` Event
[1]	ABORT_INIT	Write a 1 to enable the `abort_init` Event
[2]	ABORT_DONE	Write a 1 to enable the `abort_done` Event
[3]	ERROR	Write a 1 to enable the `error` Event

MAILBOX_STATUS

Address: 0x58018000 + 0x14 = 0x58018014

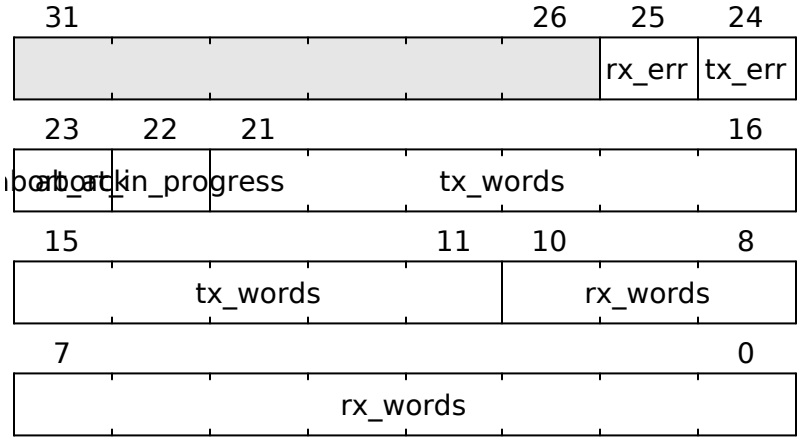

Fig. 115: MAILBOX_STATUS

Field	Name	Description
[10:0]	RX_WORDS	Number of words available to read
[21:11]	TX_WORDS	Number of words pending in write FIFO. Free space is 1024 - *tx_avail*
[22]	ABORT_IN_PROGRESS	This bit is set if an *aborting* event was initiated and is still in progress.
[23]	ABORT_ACK	This bit is set by the peer that acknowledged the incoming abort (the later of the two, in case of an imperfect race condition). The abort response handler should check this bit; if it is set, no new acknowledgement shall be issued. The bit is cleared when an initiator initiates a new abort. The initiator shall also ignore the state of this bit if it is intending to initiate a new abort cycle.
[24]	TX_ERR	Set if the write FIFO overflowed because we wrote too much data. Cleared on register read.
[25]	RX_ERR	Set if read FIFO underflowed because we read too much data. Cleared on register read.

MAILBOX_CONTROL

Address: 0x58018000 + 0x18 = 0x58018018

Field	Name	Description
[0]	ABORT	Write *1* to this field to both initiate and acknowledge an abort. Empties both FIFOs, asserts *aborting*, and prevents an interrupt from being generated by an incoming abort request. New reads & writes are ignored until *aborted* is asserted from the peer. Writing a 1 to this bit triggers the function.

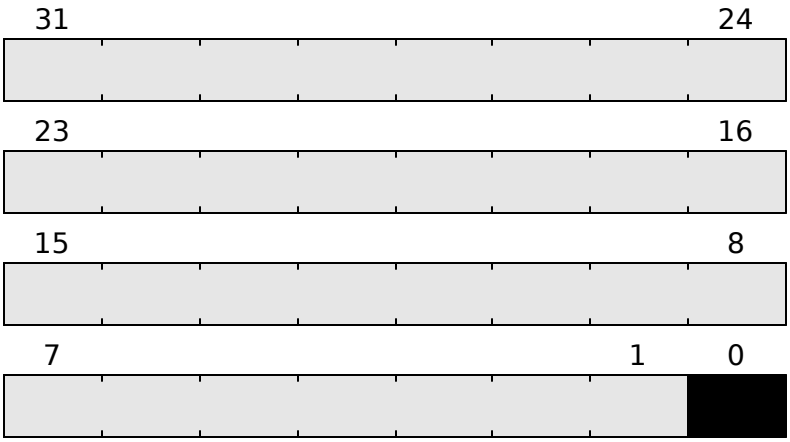

Fig. 116: MAILBOX_CONTROL

MAILBOX_DONE

Address: 0x58018000 + 0x1c = 0x5801801c

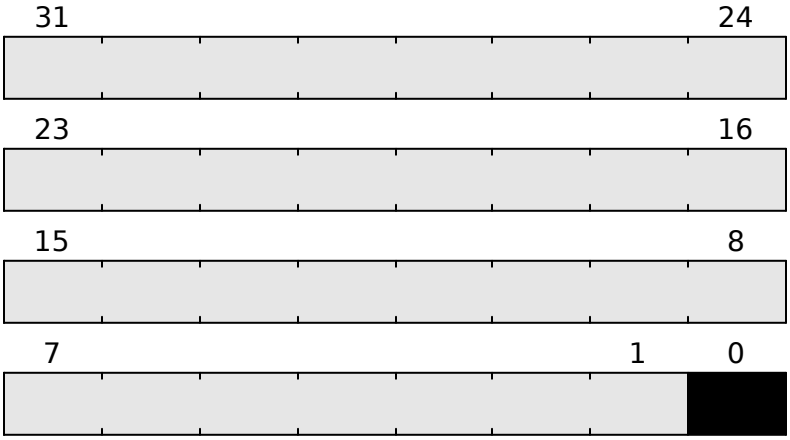

Fig. 117: MAILBOX_DONE

Field	Name	Description
[0]	DONE	Writing a *1* to this field indicates to the corresponding peer that a full packet is done loading. There is no need to clear this register after writing. Writing a 1 to this bit triggers the function.

2.26 MB_CLIENT

2.26.1 Thin Mailbox Client

This is a "minimal" mailbox client which has no FIFO of its own. It relies entirely on the other side's FIFO for the protocol to be efficient.

2.26.2 Register Listing for MB_CLIENT

Register	Address
MB_CLIENT_WDATA	*0x58019000*
MB_CLIENT_RDATA	*0x58019004*
MB_CLIENT_EV_STATUS	*0x58019008*
MB_CLIENT_EV_PENDING	*0x5801900c*
MB_CLIENT_EV_ENABLE	*0x58019010*
MB_CLIENT_STATUS	*0x58019014*
MB_CLIENT_CONTROL	*0x58019018*
MB_CLIENT_DONE	*0x5801901c*

MB_CLIENT_WDATA

Address: 0x58019000 + 0x0 = 0x58019000

Write data to outgoing FIFO.

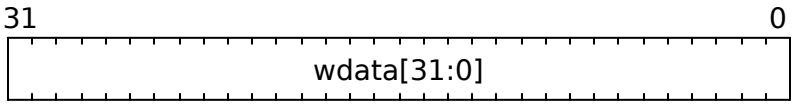

Fig. 118: MB_CLIENT_WDATA

MB_CLIENT_RDATA

Address: 0x58019000 + 0x4 = 0x58019004

Read data from incoming FIFO.

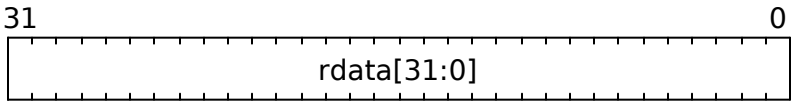

Fig. 119: MB_CLIENT_RDATA

MB_CLIENT_EV_STATUS

Address: 0x58019000 + 0x8 = 0x58019008

Triggers if either *tx_err* or *rx_err* are asserted

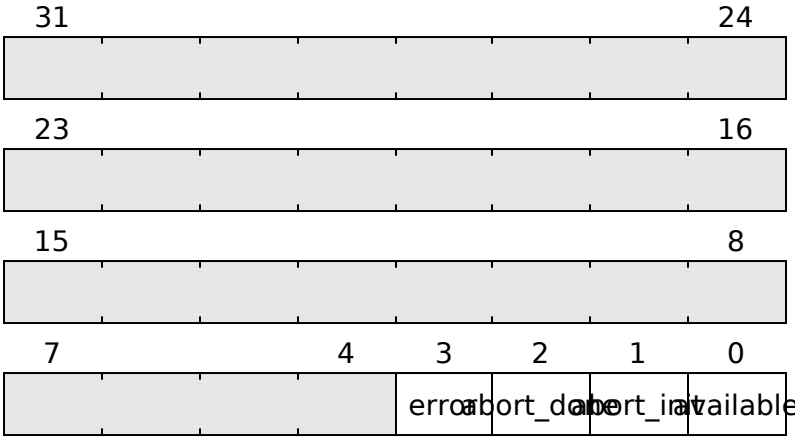

Fig. 120: MB_CLIENT_EV_STATUS

Field	Name	Description
[0]	AVAILABLE	Level of the `available` event
[1]	ABORT_INIT	Level of the `abort_init` event
[2]	ABORT_DONE	Level of the `abort_done` event
[3]	ERROR	Level of the `error` event

MB_CLIENT_EV_PENDING

Address: 0x58019000 + 0xc = 0x5801900c

Triggers if either *tx_err* or *rx_err* are asserted

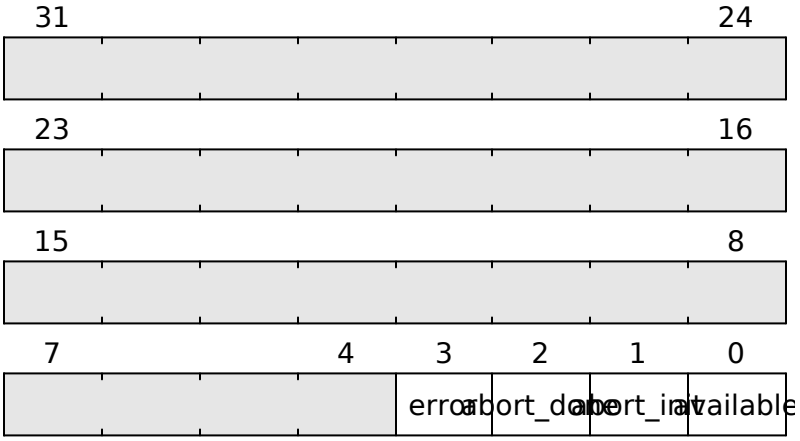

Fig. 121: MB_CLIENT_EV_PENDING

Field	Name	Description
[0]	AVAILABLE	Triggers when the *done* signal was asserted by the corresponding peer
[1]	ABORT_INIT	Triggers when abort is asserted by the peer, and there is currently no abort in progress
[2]	ABORT_DONE	Triggers when a previously initiated abort is acknowledged by peer
[3]	ERROR	Triggers if either *tx_err* or *rx_err* are asserted

MB_CLIENT_EV_ENABLE

Address: 0x58019000 + 0x10 = 0x58019010

Triggers if either *tx_err* or *rx_err* are asserted

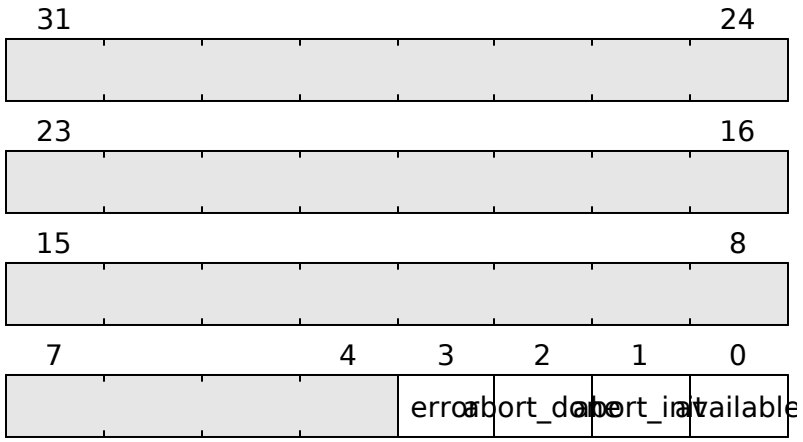

Fig. 122: MB_CLIENT_EV_ENABLE

Field	Name	Description
[0]	AVAILABLE	Write a 1 to enable the `available` Event
[1]	ABORT_INIT	Write a 1 to enable the `abort_init` Event
[2]	ABORT_DONE	Write a 1 to enable the `abort_done` Event
[3]	ERROR	Write a 1 to enable the `error` Event

MB_CLIENT_STATUS

Address: 0x58019000 + 0x14 = 0x58019014

Field	Name	Description
[0]	RX_AVAIL	Rx data is available
[1]	TX_FREE	Tx register can be written
[2]	ABORT_IN_PROGRESS	This bit is set if an *aborting* event was initiated and is still in progress.
[3]	ABORT_ACK	ACK bit is set by the peer that acknowledged the incoming abort (the later of the two, in case of an imperfect race condition). The abort response handler should check this bit; if it is set, no new acknowledgement shall be issued. The bit is cleared when an initiator initiates a new abort. The initiator shall also ignore the state of this bit if it is intending to initiate a new abort cycle.
[4]	TX_ERR	Set if the recipient was not ready for the data. Cleared on read.
[5]	RX_ERR	Set if the recipient didn't have data available for a read. Cleared on read.

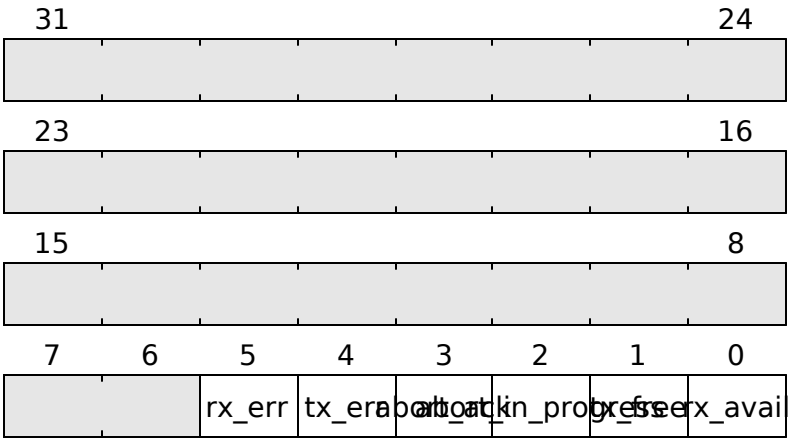

Fig. 123: MB_CLIENT_STATUS

MB_CLIENT_CONTROL

Address: 0x58019000 + 0x18 = 0x58019018

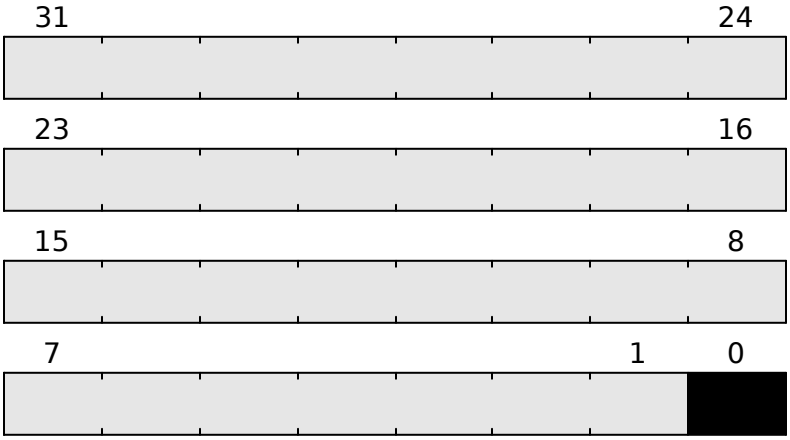

Fig. 124: MB_CLIENT_CONTROL

Field	Name	Description
[0]	ABORT	Write *1* to this field to both initiate and acknowledge an abort. Empties both FIFOs, asserts *aborting*, and prevents an interrupt from being generated by an incoming abort request. New reads & writes are ignored until *aborted* is asserted from the peer. Writing a 1 to this bit triggers the function.

MB_CLIENT_DONE

Address: 0x58019000 + 0x1c = 0x5801901c

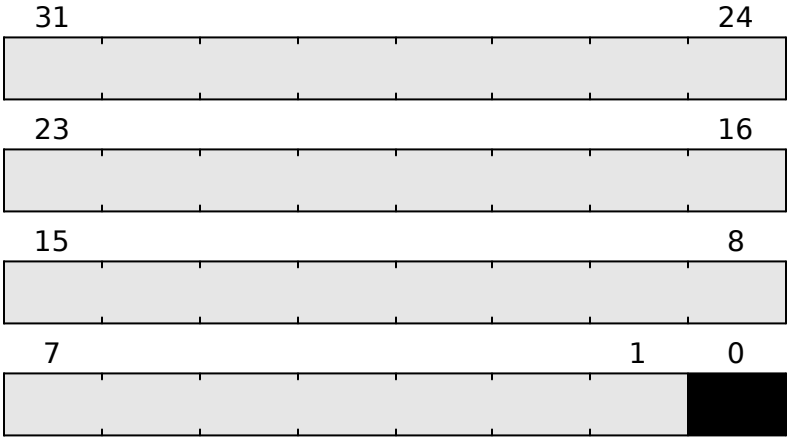

Fig. 125: MB_CLIENT_DONE

Field	Name	Description
[0]	DONE	Writing a *1* to this field indicates to the corresponding peer that a full packet is done loading. There is no need to clear this register after writing. Writing a 1 to this bit triggers the function.

2.27 RESETVALUE

ResetValue captures the actual reset value present at a reset event. The reason this is necessary is because the reset value could either be that built into the silicon, or it could come from a "trimming value" that is programmed via ReRAM bits. This vector can be read back to confirm that the reset vector is, in fact, where we expected it to be.

default_value specifies what the value would be if the *trimming_reset* ReRAM bits are not enabled with *trimming_reset_ena*.

2.27.1 Register Listing for RESETVALUE

Register	Address
RESETVALUE_PC	*0x5801a000*

RESETVALUE_PC

Address: 0x5801a000 + 0x0 = 0x5801a000

Latched value for PC on reset

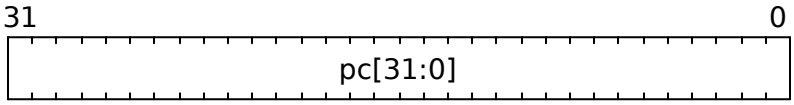

Fig. 126: RESETVALUE_PC

2.28 TICKTIMER

2.28.1 TickTimer: A practical systick timer.

TIMER0 in the system gives a high-resolution, sysclk-speed timer which overflows very quickly and requires OS overhead to convert it into a practically usable time source which counts off in systicks, instead of sysclks.

The hardware parameter to the block is the divisor of sysclk, and sysclk. So if the divisor is 1000, then the increment for a tick is 1ms. If the divisor is 2000, the increment for a tick is 0.5ms.

Note to self: substantial area savings could be hand by being smarter about the synchronization between the always-on and the TickTimer domains. Right now about 1.8% of the chip is eaten up by ~1100 synchronization registers to cross the 64-bit values between the clock domains. Since the values move rarely, a slightly smarter method would be to create a lock-out around a read pulse and then create some false_path rules around the datapaths to keep the place/route from getting distracted by the cross-domain clocks.

2.28.2 Configuration

This timer was configured with 64 bits, which rolls over in 584942417.36 years, with each bit giving 1.0ms resolution

2.28.3 msleep extension

The msleep extension is a Xous-specific add-on to aid the implementation of the msleep server.

msleep fires an interrupt when the requested time is less than or equal to the current elapsed time in systicks. The interrupt remains active until a new target is set, or masked.

There is a slight slip in time (~200ns) from when the msleep timer is set before it can take effect. This is because it takes many CPU clock cycles to transfer this data into the always-on clock domain, which runs at a much slower rate than the CPU clock.

2.28.4 Register Listing for TICKTIMER

Register	Address
TICKTIMER_CONTROL	*0x5801b000*
TICKTIMER_TIME1	*0x5801b004*
TICKTIMER_TIME0	*0x5801b008*
TICKTIMER_MSLEEP_TARGET1	*0x5801b00c*
TICKTIMER_MSLEEP_TARGET0	*0x5801b010*
TICKTIMER_EV_STATUS	*0x5801b014*
TICKTIMER_EV_PENDING	*0x5801b018*
TICKTIMER_EV_ENABLE	*0x5801b01c*

TICKTIMER_CONTROL

Address: 0x5801b000 + 0x0 = 0x5801b000

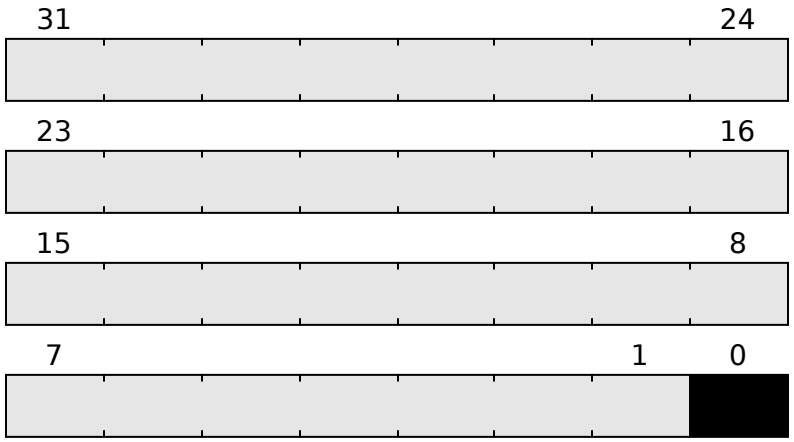

Fig. 127: TICKTIMER_CONTROL

Field	Name	Description
[0]	RESET	Write a *1* to this bit to reset the count to 0. This bit has priority over all other requests. Writing a 1 to this bit triggers the function.

TICKTIMER_TIME1

Address: 0x5801b000 + 0x4 = 0x5801b004

Bits 32-63 of *TICKTIMER_TIME*. Elapsed time in systicks

Fig. 128: TICKTIMER_TIME1

TICKTIMER_TIME0

Address: 0x5801b000 + 0x8 = 0x5801b008

Bits 0-31 of *TICKTIMER_TIME*.

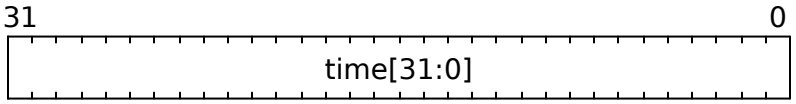

Fig. 129: TICKTIMER_TIME0

TICKTIMER_MSLEEP_TARGET1

Address: 0x5801b000 + 0xc = 0x5801b00c

Bits 32-63 of *TICKTIMER_MSLEEP_TARGET*. Target time in 1.0ms ticks

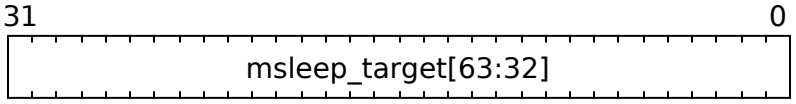

Fig. 130: TICKTIMER_MSLEEP_TARGET1

TICKTIMER_MSLEEP_TARGET0

Address: 0x5801b000 + 0x10 = 0x5801b010

Bits 0-31 of *TICKTIMER_MSLEEP_TARGET*.

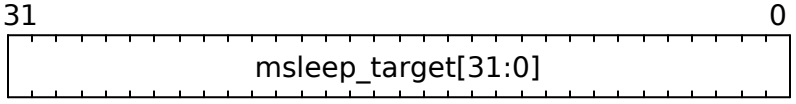

Fig. 131: TICKTIMER_MSLEEP_TARGET0

TICKTIMER_EV_STATUS

Address: 0x5801b000 + 0x14 = 0x5801b014

This register contains the current raw level of the alarm event trigger. Writes to this register have no effect.

Field	Name	Description
[0]	ALARM	Level of the `alarm` event

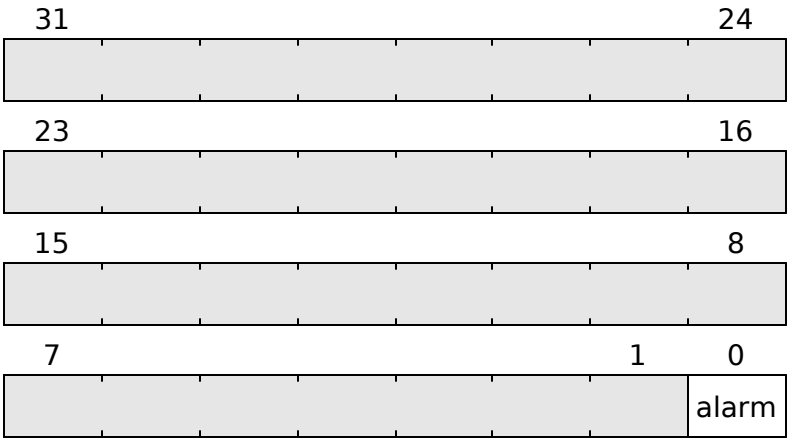

Fig. 132: TICKTIMER_EV_STATUS

TICKTIMER_EV_PENDING

Address: 0x5801b000 + 0x18 = 0x5801b018

When a alarm event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.

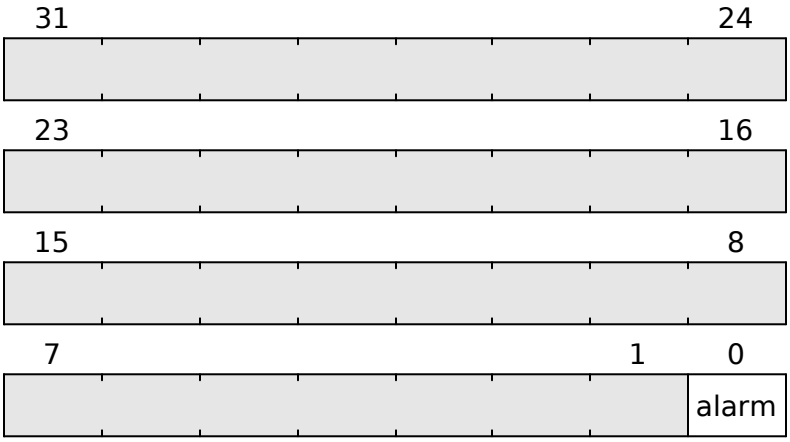

Fig. 133: TICKTIMER_EV_PENDING

Field	Name	Description
[0]	ALARM	*1* if a *alarm* event occurred. This Event is **level triggered** when the signal is **high**.

TICKTIMER_EV_ENABLE

Address: 0x5801b000 + 0x1c = 0x5801b01c

This register enables the corresponding alarm events. Write a `0` to this register to disable individual events.

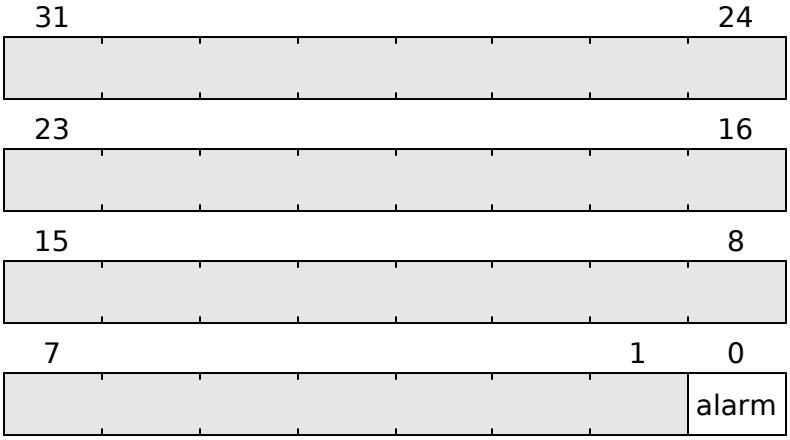

Fig. 134: TICKTIMER_EV_ENABLE

Field	Name	Description
[0]	ALARM	Write a 1 to enable the `alarm` Event

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search