Eccelerators Library IP specification

Table of Contents

List of Figures

List of Tables

1. EventCatcherIfc

1.1. Event Catcher Interface (EventCatcherIfc)

Interface containing a basic Event Catcher block.

Table 1.1. Blocks of Event Catcher Interface

Blocks of Event Catcher Interface		
Block Address	ID	Block Name
`0x00`	EventCatcherBlk	Event Catcher Block

Table 1.2. Resets of Event Catcher Interface

Resets of Registers of Event Catcher Interface	
ID	Reset Name
Async	BusReset: Asynchronous Bus Reset

1.1.1. Event Catcher Block (EventCatcherBlk)

This block defines a basic event catcher e.g. to convert **egde triggered** interrupt sources to **level triggered** interrupt sources. Its purpose is to be inherited by implementations of e.g., GPIO or Timer controllers utilizing this function to provide event information to SW for polling or as a prestage to transport events to an interrupt controller. Usually the events can be activated or deactivated at the HW block sourcing them e.g. timer is activated or not. The events must be deliverered as a pulse of exactly one clock duration.

Event Catcher details:

Figure 1.1. Event Catcher details

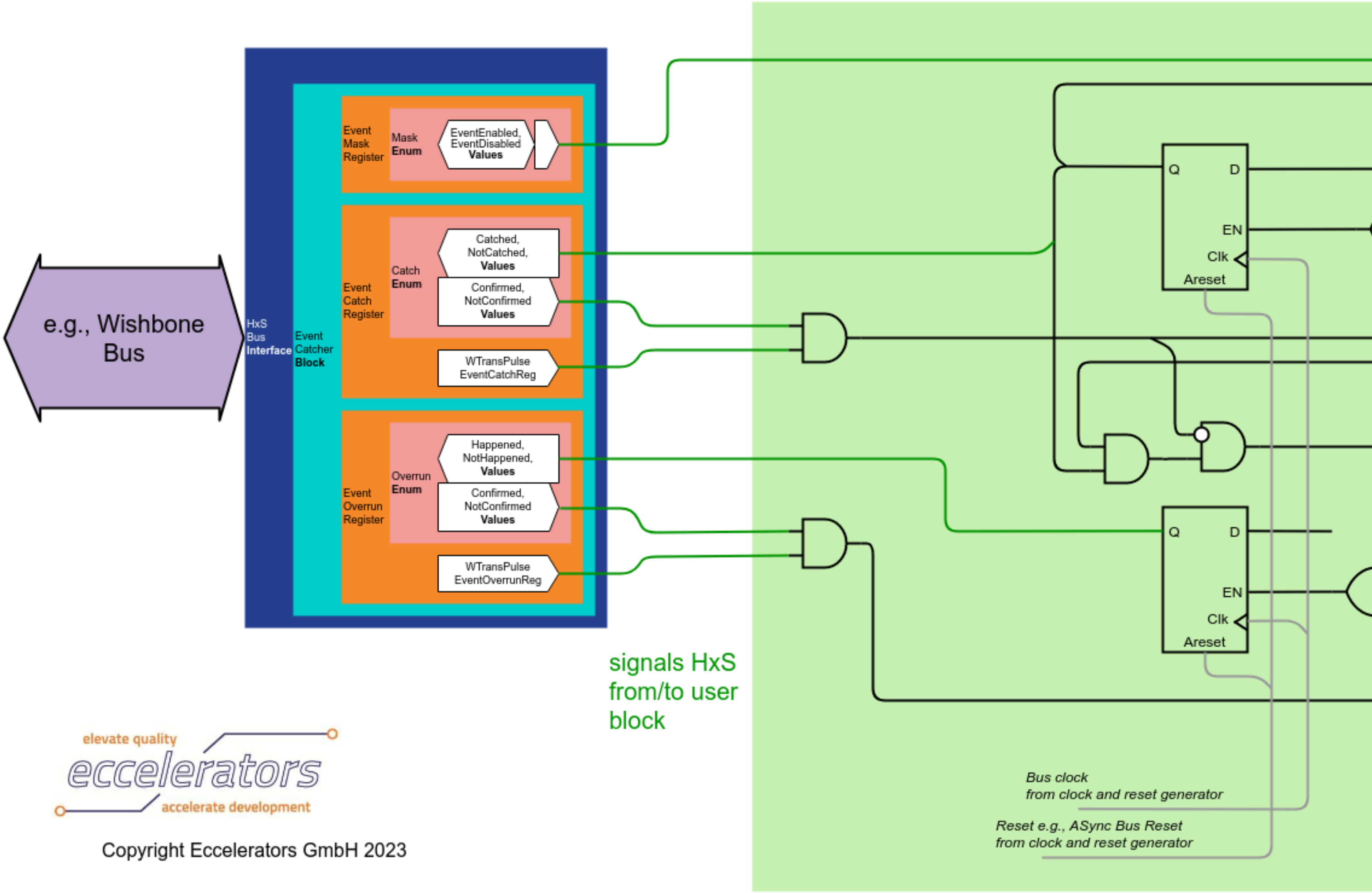
e.g., Wishbone Bus
HxS Bus Interface
Event Catcher Block
Event Mask Register
Mask Enum
EventEnabled, EventDisabled Values
Event Catch Register
Catch Enum
Catched, NotCatched, Values
Confirmed, NotConfirmed Values
WTransPulse EventCatchReg
Event Overrun Register
Overrun Enum
Happened, NotHappened, Values
Confirmed, NotConfirmed Values
WTransPulse EventOverrunReg
signals HxS from/to user block
Q
D
EN
Clk
Areset
Q
D
EN
Clk
Areset
Bus clock from clock and reset generator
Reset e.g., ASync Bus Reset from clock and reset generator
elevate quality
eccelerators
accelerate development
Copyright Eccelerators GmbH 2023

Table 1.3. Registers or Delegates of Event Catcher Block

Registers or Delegates of Event Catcher Block		

`0x00`
`..`
`0x0b`
		Event Catcher Block

Address	ID	Name
`0x00`	EventMaskReg	Event Mask Register
`0x04`	EventCatchReg	Event Catch Register
`0x08`	EventOverrun-Reg	Event Overrun Register

1.1.1.1. Event Mask Register (EventMaskReg)

Table 1.4. Bits of Event Mask Register

Bits of Event Mask Register			
`0x00`			Event Mask Register (EventMaskReg)
Bits	ID	Type	Description
03 .. 00	Mask	RW	**Table 1.5. Values of Mask**

Value	ID	Type	Description
`0x1`	EventEnabled	RW	Event is catched and forwarded.
`0x0`	EventDis-abled	RW	Event is ignored.

<table>
<tr><td></td><td></td><td></td><td><b>Table 1.6. Resets of Mask</b>
<table>
<tr><td>0x0</td><td>BusReset</td><td>RW</td><td>Default Bus Reset</td></tr>
</table>
</td></tr>
</table>

1.1.1.2. Event Catch Register (EventCatchReg)

Table 1.7. Bits of Event Catch Register

<table>
<tr><td colspan="4">Bits of Event Catch Register</td></tr>
<tr><td colspan="3">0x04</td><td>Event Catch Register (EventCatchReg)</td></tr>
<tr><td>Bits</td><td>ID</td><td>Type</td><td>Description</td></tr>
<tr><td>03
..
00</td><td>Catch</td><td>R/W</td><td><b>Table 1.8. Values of Catch</b>
<table>
<tr><td>Value</td><td>ID</td><td>Type</td><td>Description</td></tr>
<tr><td>0x1</td><td>Catched</td><td>R</td><td>An event has been catched and is waiting to be processed.</td></tr>
<tr><td>0x0</td><td>NotCatched</td><td>R</td><td>No event has been catched.</td></tr>
<tr><td>0x1</td><td>Confirmed</td><td>W</td><td>Notifies HW that a catched event has been processed by SW.
SW writes Confirmed to confirm to HW that a catched event has been processed. The value isn’t stored, thus there is no need to reset it to ‘0’ again. Solely the write action is sufficient.</td></tr>
<tr><td>0x0</td><td>InEffective</td><td>W</td><td>Writing this value has no effect.</td></tr>
</table>
<b>Table 1.9. Resets of Catch</b>
<table>
<tr><td>Init Value</td><td>ID</td><td>Impact</td><td>Description</td></tr>
<tr><td>0b0000</td><td>Async</td><td>R</td><td>BusReset: Asynchronous Bus Reset
The bit Catch is set to NotCatched after reset in the usually attached EventCatcher HW block.</td></tr>
</table>
</td></tr>
</table>

1.1.1.3. Event Overrun Register (EventOverrunReg)

Table 1.10. Bits of Event Overrun Register

<table>
<tr><td colspan="4">Bits of Event Overrun Register</td></tr>
<tr><td colspan="3">0x08</td><td>Event Overrun Register (EventOverrunReg)</td></tr>
<tr><td>Bits</td><td>ID</td><td>Type</td><td>Description</td></tr>
<tr><td>03
..
00</td><td>Overrun</td><td>R/W</td><td><b>Table 1.11. Values of Overrun</b>
<table>
<tr><td>Value</td><td>ID</td><td>Type</td><td>Description</td></tr>
<tr><td>0x1</td><td>Happened</td><td>R</td><td>There has been a new event before a catched event had been processed thus an overrun has happened and at least one event has been lost.</td></tr>
<tr><td>0x0</td><td>NotHappened</td><td>R</td><td>No event has been lost.</td></tr>
<tr><td>0x1</td><td>Confirmed</td><td>W</td><td>Notifies HW that a event overrun has been processed by SW.
SW writes Confirmed to confirm to HW that a event overrun has been processed. The value isn’t stored, thus there is no need to reset it to ‘0’ again. Solely the write action is sufficient.</td></tr>
<tr><td>0x0</td><td>InEffective</td><td>W</td><td>Writing this value has no effect.</td></tr>
</table>
<b>Table 1.12. Resets of Overrun</b>
<table>
<tr><td>Init Value</td><td>ID</td><td>Impact</td><td>Description</td></tr>
<tr><td>0b0000</td><td>Async</td><td>R</td><td>BusReset: Asynchronous Bus Reset
The bit Overrun is set to NotHappened after reset in the usually attached EventCatcher HW block.</td></tr>
</table>
</td></tr>
</table>