PMU_counters specification version v1

Guillem Cabo Pitarch

August 29, 2022

CONTENTS

1 General purpose of the module

Submodule of the PMU to handle event counters. This module contains generic adders with reset and simple control logic to ease interface with other modules.

New values for the counters is provided through regs_i and output of the counters is provided through regs_o.

Counter results are internally registered.

2 Design placement

This modules is meant to be instantiated by the interface agnostic PMU (PMU_raw.sv). Only one instance of this module is required. The application-specific interface shall take care of the write enable of this module.

3 Parameters

This module requires two integer parameters.REG_WIDTH and N_COUNTERS.
REG_WIDTH defines the number of bits that a counter has.
N_COUNTERS defines the amount of counters that are available in the PMU.

Given the previous parameters the unit shall generate correct RTL without any manual modification to the source code.

4 Interface

Port Name	Direction	Width	Index	Comment	Comment Source
clk_i	INPUT	1	-	Global Clock Signal	module port
rstn_i	INPUT	1	-	Global Reset Signal. This Signal...	module port
softrst_i	INPUT	1	-	Active HIGH	module port
en_i	INPUT	1	-	Active HIGH	module port
we_i	INPUT	1	-	TODO: Consider if is worth addin...	module port
regs_i	INPUT	288	[0:8][31:0]	registers	module port
regs_o	OUTPUT	288	[0:8][31:0]	-	-
events_i	INPUT	9	[8:0]	external signals from Soc events	module port

Table 4.1: Ports of module PMU_counters

Interface signals of the module are listed in table **??**

5 RESET BEHAVIOR

The module contains two reset signals. One synchronous active high reset enabled by software called softrst_i, the other is the global reset, called rstn_i it is asynchronous and active low.

Any reset when active shall clear any internal register and set them to 0. Interruptions shall become inactive while the unit is in reset.

6 WHAT COULD NOT HAPPEN

When a write is performed to any of the counter values fo the PMU regs_o shall show the most up to date value of the counters, rather than the internally registered value of the counters. Otherwise the value of the counter may oscillate between the last value and new one.

7 BEHAVIOR

The module uses generate loops to parametrically implement the amount of counters required. The output of the adders is stored in slv_reg. the adder takes the last value of the counter (slv_reg[n]) and adds one to it if the unit is enabled and the event n is active. Events and counters share the same index, meaning that event[0] will map to slv_reg[0].

The outputs of the adders are stored in slv_reg. If counters need to be updated externally the write enable signal (we_i) will be high. When we_i is high the input regs_i will be bypassed to regs_o ignoring the current value of the counters, also each slv_reg will be updated with regs_i rather than the ouput of the adders.

7.1 PACKAGES AND STRUCTURES

No packages and structures are used in this module.

8 SPECIAL CASES, CORNER CASES

No special cases or corner cases have been considered.