PMU_raw specification version v1

Guillem Cabo Pitarch

August 29, 2022

CONTENTS

1 General purpose of the module

PMU_raw is an interface agnostic module that integrates all the features of the PMU based on the instance parameters. This module will instance the sub-modules that implement the MCCU, RDC, quota, counters and overflow when needed. Based on the input parameters it will generate the memory map of the PMU and each feature. This will allow to automatically connect most of the signals and extract the values of each parameter after elaboration to generate C drivers for the unit.

Given that the unit will act as a middle module between the features and the top level that will implement the control logic for the bus protocol of the SoC or Core that integrates the PMU, this module will pass through most of the signals.

In addition to the routing the unit implements a self-test feature that allows to bypass the event input and help to check that the software and hardware behave correctly.

2 Design placement

This modules is meant to be instantiated by the interface specific PMU (pmu_ahb.sv for instance). Only one instance of this module is required.

3 Parameters

This unit has few external parameters.**REG_WIDTH**, **N_COUNTERS** and **N_CONF_REGS** are the basic parameters to configure the memory map of the PMU and route the signals. This parameters are provided by the module instancing the unit. A set of local parameters select the active features (**OVERFLOW**, **QUOTA**, **MCCU**, **RDC**).

Given the previous parameters all the memory map is generates as local parameters as well. For each group of signals associated with a feature function there is a local parameter with the name of the feature and function prefixed by **BASE_**, **N_** and **END_**. This signals have the first register of the feature function, the number of used registers and the last register of the feature, respectively. The total of registers is stored in **TOTAL_NREGS** and it adds up all the intermediate parameters with the registers used by each feature.

Is key to generate this parameters following a consistent method and build the values of the next parameters based in the last values to prevent inconsistencies. Furthermore all the parameters are important since the C libraries will be generated out of the parameter report generated by spyglass.

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
regs_i	INPUT	896	[0:27] [31:0]	Input registers	module port
regs_o	OUTPUT	896	[0:27] [31:0]	Output registers	-
wrapper_we_i	INPUT	1	-	Write counters from regs_i	module port
events_i	INPUT	9	[8:0]	Event signals	module port
intr_overflow_o	OUTPUT	1	-	Overflow interruption	module port
intr_quota_o	OUTPUT	1	-	Quota interruption	module port
intr_MCCU_o	OUTPUT	4	[3:0]	MCCU interruption	module port
intr_RDC_o	OUTPUT	1	-	RDC interruption	module port

Table 4.1: Ports of module PMU_raw and default widths

Interface signals of the module are listed in table 4.1

5 RESET BEHAVIOR

The module contains a single 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.

This unit will filter the reset signals of the MCCU, avoiding any glitch but making the reset synchronous.

6 WHAT COULD NOT HAPPEN

No special restrictions have been considered other than specified in section 7.

ILA_DEBUG_PMU_RAW shall be disabled unless the unit is under FPGA debug on a Xilinx platform.

FORMAL shall never be defined when doing synthesis.

7 BEHAVIOR

7.1 SELF-TEST

While the unit can be easily verified on simulations by feeding specific test patterns, on a hardware implementation is much harder to do without some hardware support. Self-test is important to verify that the software behaves correctly, but also to assure that there are no discrepancies with the simulation.

To perform a simple self-test the unit uses two bits of the configuration registers to change between four operation modes. **NO_SELF_TEST** (0b00), **ALL_ACTIVE** (0b01), **ALL_OFF** (0b10), **ONE_ON** (0b11). This configuration modes allow, respectively, to pass through the events of the SoC, active all events, disable all events or active only event 0.

7.2 Counters

Events an I/O signals are passed through without changes.

7.3 Overflow

Only active signals are passed through. If N_COUNTERS larger than REG_WIDTH it will need modifications to route **overflow_intr_mask_i** and **overflow_intr_vect_o**.

7.4 Quota

Only active signals are passed through. If N_COUNTERS larger than REG_WIDTH it will need modifications to route **quota_mask_i**.

7.5 MCCU

Events are routed through an internal signal called **MCCU_events_int**. The MCCU will route part of the signals of the SoC, so the number of the counters can be smaller. It is defined by **MCCU_N_CORES** and **MCCU_N_EVENTS**.
Since the signal**MCCU_softrst** may have glitches, **MCCU_rstn** is synchronized to avoid such conditions and unpredictable resets on MCCU.

MCCU_update_quota_int,**MCCU_events_intMCCU_events_weights_int** and **intr_MCCU_o** are not parametric. If a change is performed to MCCU_N_CORES or MCCU_N_EVENTS minor adjustmens will be required.

7.6 RDC

Only active signals are passed through. If N_COUNTERS larger than REG_WIDTH it will need modifications to route **interruption_rdc_o**.

The events and weight signals are shared with the MCCU. Therefore both functions must be present in the PMU or the MCCU code to route **MCCU_events_weights_int** and **MCCU_events_weights_int** shall be duplicated.

7.7 Packages and structures

No packages and structures are used in this module.

8 Special cases, corner cases

The interrupts shall be generated if the current pulse value is equal or larger than the event weigth. Otherwise if the event weight is set to the maximum value of the register, due to the overflow protection of the counters will prevent the interrupt to trigger, producing a non intuitive outcome.