PMU_ahb specification version v1

Guillem Cabo Pitarch & Sergi Alcaide

October 25, 2022

CONTENTS

1 General purpose of the module

This module is the top level for the application specific PMU. It implements an AHB slave interface that connects with any compliant master with a 32 wide amba bus. Wider buses can be used with minimal modifications, for more detail on this matter check AMBA AHB 5 specification [**?**], section 6.3.1.

The module has a set of registers that are synchronized between the interface agnostic PMU and the SoC. Writes to the registers are updated to internal registers of each PMU feature if needed.

2 Design placement

This modules is meant to be used as a top level. It will instance PMU_raw, and only one instance is required.

3 Parameters

This unit uses several parameters. **REG_WIDTH** defines the size of the data registers.

HDATA_WIDTH and **HADDR_WIDTH** define the width of the data and address signals.

MCCU_N_CORES, **N_REGS**, **PMU_CFG** and **PMU_COUNTERS** define the parameters of the PMU that will be used.

4 Interface

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.
At reset internal registers shall be set to 0.

6 What could not happen

The clock of this module (**clk_i**) shall be the same than the clock of the modules that generate the events. If the clock does not come from the same source events are not guaranteed to be registered. If the clock is faster events will be duplicated. If the clock is slower the events will

Port Name	Direction	Width	Index	Comment	Comment Source
rstn_i	INPUT	1	Clock	Width of registers data bus	module port
clk_i	INPUT	1	-	-	-
hsel_i	INPUT	1	-	Slave select	module port
hreadyi_i	INPUT	1	-	Previous transfer done	module port
haddr_i	INPUT	32	[31:0]	Address bus	module port
hwrite_i	INPUT	1	-	Read/write	module port
htrans_i	INPUT	2	[1:0]	Transfer type	module port
hsize_i	INPUT	3	[2:0]	Transfer size	module port
hburst_i	INPUT	3	[2:0]	Burst type	module port
hwdata_i	INPUT	32	[31:0]	Write data bus	module port
hprot_i	INPUT	4	[3:0]	Portection control	module port
hmastlock_i	INPUT	1	-	Locked access	module port
hreadyo_o	OUTPUT	1	-	Trasfer done	module port
hresp_o	OUTPUT	2	[1:0]	response type	module port
hrdata_o	OUTPUT	32	[31:0]	Read data bus	module port
events_i	INPUT	24	[23:0]	PMU input events6	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 interruptions	module port
intr_RDC_o	OUTPUT	1	-	RDC interruption	module port

Table 4.1: Ports of module pmu_ahb

not register.

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

7 BEHAVIOR

7.1 AHB REGISTERS

The internal registers are generated parametrically based in **REG_WIDTH** and **N_REGS**. They are named **slv_reg** and the input and output of such registers are named **slv_reg_D** and **slv_reg_Q**. At reset the values become 0.

7.2 AHB CONTROL LOGIC

The control logic is divided in several elements. Since AHB is a pipelined protocol we have two phases, named address and data. While in address phase the transaction gets started by the slave, on data phase the content of the writes or reads is transferred.

In address phase, we check **htrans_i** in combinational logic to determine if the state of the bus transfer (**IDLE, BUSY, NONSEQUENTIAL, SEQUENTIAL**) and if the slave is selected. The output of this logic is the signal named **next**.Then a small state machine that registers the required inputs to handle a transfer. The state of the machine is given by **next**. The signals from the bus are stored several registers defined as the packed structure **address_phase**. It

has a select, write and address_phase fields.

In the data phase of the transference we have combinational logic to determine the memory position from our register bank that needs to be accessed, and assign the data required to **dread_slave** or **dwrite_slave** depending on the transaction. Reads are taken from **slv_reg_Q**. Bussy and IDLE states put fix values in the dwrite_slave and dread_slave signals to ease detection of any bug, since such values shall never be used unless there is a bug in the module.

7.3 SLAVE REGISTERS UPDATE

The slave registers need to be sync between the PMU_raw submodule and the SoC. If there where no logic to manage conflicts between updates of the counters by the PMU features and the incoming writes the register values may become inconsistent. In order to prevent this issue, when there is an incoming write **slv_reg_Q** and **slv_reg_D slv_reg_D**, allowing to register the new value and passing to the submodules the most updated value for such register. If no write transaction is active the pmu updates the registers with the signal **pmu_regs_int**.

7.4 PACKAGES AND STRUCTURES

No packages are used.

A single structure is used. It is named pmu_regs_int and it contains one bit to record bus select, bus writes, and the address for the incoming request.

Structures and typedef could be used more extensively to handle Types of bursts, Type of transfers, Type of Ready outputs, etc...

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.