PMU_MCCU specification version v1

Guillem Cabo Pitarch

August 29, 2022

Contents

1 General purpose of the module 3

2 Design placement 3

3 Parameters 3

4 Interface 4

5 Reset behavior 4

6 What could not happen 4

7 Behavior 4

7.1 Quota update 4

7.2 IDLE state 5

7.3 Active without update 5

7.4 Active without update 5

7.5 Interruptions 5

7.6 Addition of weights 5

7.7 Packages and structures 5

8 Special cases, corner cases 6

1 General purpose of the module

MCCU stands for maximum-contention control unit, and aims to measure resource access an enforce the maximum contention produced for different cores in a multi-core system. Further details about the initial concept are provided by the paper Maximum-Contention Control Unit (MCCU): Resource Access Count and Contention Time Enforcement, implementation may vary.

The MCCU is allowed to assign individual quotas for each core. The quota value represent the number of clock cycles that the corresponding core is allowed to cause contention over other cores. At setup the user assigns weights to each one of the MCCU signals, the content of the weight would be the average or worst contention that a given event can cause over other cores, is up to the user to set the appropiate value for the end application. While the unit is enabled, at each clock cycle checks all the active events and adds its corresponding weights. In the same cycle the result of the weight addition is subtracted from the core quota. When the quota is about to reach 0 an interrupt is risen.

Interruptions can be handled individually for each core, rather than relaying on a monitor core, but that is dependent of the target platform.

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.

3 Parameters

This unit uses several parameters. **DATA_WIDTH** defines the size of the data registers. **WEIGHTS_WIDTH** defines the size of the registers that store the estimated contention that a given event causes. **N_CORES** is used to configured to replicate the internal logic of the RDC to support several cores. **CORE_EVENTS** sets the number of events that are track for each core.

Internal parameters are used to perform padding between signals of different width when need to be used together. Such local parameter are **O_D_0PAD, D_W_0PAD, O_W_0PAD** further details are provided in the source code comments.

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	-	Width of data registers	module port
rstn_i	INPUT	1	-	Active low syncronous reset. It...	module port
enable_i	INPUT	1	-	can be generated	module port
events_i	INPUT	8	[0:1][3:0]	Monitored events that can genera...	module port
quota_i	INPUT	64	[0:1][31:0]	Quota for each of the cores, int...	module port
update_quota_i	INPUT	2	[0:1]	Update quota. Set quota_int to t...	module port
quota_o	OUTPUT	64	[0:1][31:0]	Internal quota available	module port
events_weights_i	INPUT	64	[0:1][0:3][7:0]	internally registered, set by so...	module port

Table 4.1: Ports of module MCCU

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.

When reset is active shall any internal register shall be set to 0. Specially the current weight addition (**ccc_suma_int**) and current quota (**quota_int**)

Interruption shall become inactive while the unit is in reset or disabled.

6 WHAT COULD NOT HAPPEN

DEBUG and define shall be undefined for synthesis.

SYNT shall be defined for synthesis.

7 BEHAVIOR

ext subsections describe the different functional states of the unit and interruption capabilities.

7.1 QUOTA UPDATE

The unit requires a synchronization mechanism with the wrapper since the available quota is registered at the top level and internally. Such synchronization is done through an update signal (**update_quota_i**) provided by the module that instances the MCCU. Each quota for each core has an individual update bit. **If the unit is active and the update bit is active** the interal quota (**quota_int**) takes the input quota (quota_int). Having an external quota that remains constant and an internal quota allows the developers to see the actual quota consumed and the initial quota without having to keep track in software.

7.2 IDLE STATE

If MCCU is inactive and the update bit is not active **(!enable_i && update_quota_i[i])** the internal quota remains the same than in the previous cycle.

7.3 ACTIVE WITHOUT UPDATE

When the unit is active state and no quota update is required **(enable_i && !update_quota_i[i])** the internal quota (**quota_int**) is set to either the current quota minus the addition of weights of the active events (**ccc_suma_int**) or if an underflow is going to be produced, it is set to 0.

7.4 ACTIVE WITHOUT UPDATE

If the unit is active but the update bit is set high (**enable_i && update_quota_i[i]**) the same operation of section 7.3 is performed, but instead of subtracting to the last quota, the incoming quota (quota_i[i]) is used instead.

7.5 INTERRUPTIONS

Interruptions are generated when the unit is enabled and not in reset. Even if the reset is not explicitly guarded, while the unit is in reset the comparison among the current addition of active weights (**ccc_suma_int**) and the internal quota (**quota_int**) will not cause an interrupt. If ccc_suma_int[i] is larger than quota_int[i] the interruption will rise for core i.

7.6 ADDITION OF WEIGHTS

The addition of weights is done in two steps, first the event will set if the weight needs to be added or not. To do so an internal signal called **events_weights_int** is used. If the event is active events_weights_int will contain the weight forwarded for the value in **events_weights_i** otherwise the value will be 0. events_weights_int is always added together in a single cycle and the result is finally stored in **ccc_suma_int**.

7.7 PACKAGES AND STRUCTURES

No packages and structures are used in this module.

8 SPECIAL CASES, CORNER CASES

Depending on the number of signals and the size of the weights the addition of events may result in overflow. Is important to ensure that **ccc_suma_int** is wide enough to store the addition of the maximum value of all the weights.

The user shall ensure that the update bit is set and released to avoid missing interruptions and allow normal operation og the unit.