PMU_RDC specification version v1

Guillem Cabo Pitarch

August 29, 2022

Contents

1 General purpose of the module

RDC stands by request duration counter. The purpose of this unit is track the length of a given event. At the core we assume that there are two types of events that are feed in to the PMU. On one hand there are single clock events, where each cycle that they are high means that a new event has occurred, and example of it would be instruction count for instance. On the other hand we have multiple-cycle events, in this case the event is risen at the begging of a given condition and remains high until the condition has finished, counting effectively the cycles that such condition is high rather than the number of events. An example of a multiple-cycle event would be the cache hold signal, that indicates the time the cache is waiting to get a response from memory.

The RDC implements a mechanism to measure the length of multiple-cycle events and control that the length of a given event is not exceeded. If an event exceeds the set length an interruption is risen. A single interruption is provided for the whole unit.

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 lengths of the requests and the maximum requests allowed. **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.

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

4 Interface

Interface signals of the module are listed in table 4.1

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	[0:3][1:0]	Monitored events that can genera...	module port
events_weights_i	INPUT	64	[0:3][0:1][7:0]	internally registered, set by so...	module port
interruption_rdc_o	OUTPUT	1	-	Event longer than specified weig...	module port
interruption_vector_rdc_o	OUTPUT	8	[0:3][1:0]	Interruption vector to indicate ...	module port
watermark_o	OUTPUT	64	[0:3][0:1][7:0]	High watermark for each event of...	module port

Table 4.1: Ports of module RDC

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

6 What could not happen

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

7 Behavior

7.1 Pulse counters

The unit has a generate loop that will create for each input event of each core one counter, a max value hold register and update logic. While rstn_is low the hold registers (max_value) are set to 0. If the unit is enabled and the event assigned to the counter is active the counter will increase by one and the result will be stored in the hold register. If the event is 0 the hold register is reset to 0.

Each counter has overflow protection.

7.2 Watermark

The watermark logic assigns one register for each event of each core called watermark_int. At reset the register is set to 0. If the unit is not enabled the values of watermark_int are keept unchanged. If the unit is active the current value of watermark_int is compared against the current value of the pulse counters (watermark_int). When max_value is larger than watermark_int, the second is updated with the new value of the pulse counter (max_value).

7.3 Interruption

The RDC provides one interruption (interruption_vector_rdc_o) for the whole unit an one interruption vector (interruption_vector_int) for each core. The interruption vector encodes in one hot encoding the signal exceeding the pulse width, the LSB of the vector is maps to the signal 0 of each core, and the remaining bits are filled in ascending order with the following signals.

The interruption logic checks for each signal if its corresponding max_value exceeds the respective events_weights_i. In other words if the current pulse been measured exceeds the weight currently assigned to it a bit in the interrupt vector of the corresponding core will be set to high.

Interruptions are hold over time, until the init is disabled or reset. That is achieved by registering the output interruption vector (interruption_vector_rdc_o) and applying an or operation between its past value and the current cycle internal interruption vector (interruption_vector_int).

To decide if the RDC interruption needs to be risen all the interruption vectors are or-reduced along a bit that stores the previous state of the interrupt.

7.4 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.