SafeDE User's Manual

Francisco Bas

August 18, 2021

Contents

1 Overview

SafeDE is a flexible Diversity Enforcement hardware module that provides a solution for light-lockstepping execution, an alternative to conventional lockstep. In contrast to the traditional lockstep solution, in the light-lockstep approach, there are two independent cores that can execute either critical tasks in lockstepped execution or non-critical tasks independently. For that purpose, SafeDE is instantiated in the SoC as an APB slave[1] and couples two cores, forcing them to keep some staggering [2] and introducing time diversity to avoid common cause faults.

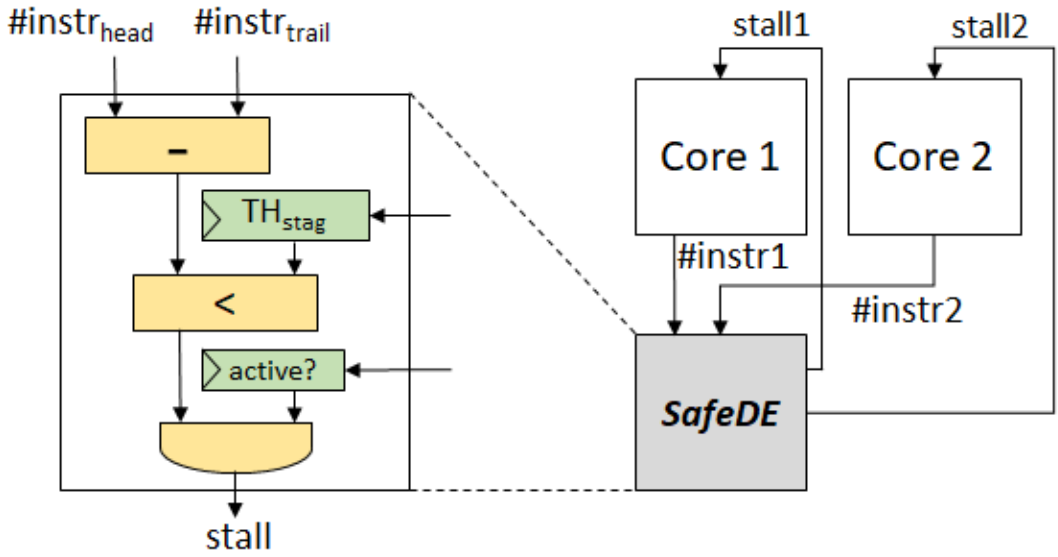


Figure 1.1: Simplified light-lockstep scheme with SafeDE.

In figure 1.1 it is shown a simplified light-lockstep scheme with SafeDE. When active, SafeDE counts the instructions executed by both cores. Each cycle, SafeDE computes the staggering substrating those instructions executed by the trail core to the ones executed by the head core. If that difference (staggering) is smaller than a given threshold, the trail core is stalled.

A light-lockstep execution implemented with SafeDE offers some advantages in contrast with the classical lockstep approach:

- **Low-cost:** SafeDE employs few hardware resources.
- **Flexibility:** SafeDE can be enabled and disable at will. This allows the possibility of using a couple of cores to execute either two tasks with SafeDE disabled or one task

[1] Based on AMBA 2.0 specification
[2] Difference of executed instructions between two cores executing the same software

with SafeDE enabled, depending on its criticality.

- **Low intrusiveness:** SafeDE can be implemented in a SoC performing only a few modifications in the design. Namely, SafeDE needs only the instruction counters and two signals to stall the cores.

When working in light-lockstep mode, both cores have to execute exactly the same instructions so that software does not diverge. For that reason, they need to execute redundant independent processes allocated in different segments of memory. Even though SafeDE guarantees time diversity, a software mechanism to detect the discrepancies in the results of both cores has to be implemented. SafeDE is devised to work in bare metal since an operating system could make the paths of the redundant cores diverge (e.g. interruptions, exceptions).

2 SIGNAL DESCRIPTIONS

Table 2.1 shows the interface of the core (VHDL ports).

Table 2.1: Signal descriptions (VHDL ports)

Signal name	Field	Type	Function	Active
RSTN		Input	Reset	Low
CLK		Input	AHB master bus clock	-
ICNT1_I		Input	Input to count the instructions executed by core1	High
ICNT2_I		Input	Input to count the instructions executed by core2	High
STALL1_O		Output	Output to stall the pipeline of core1	High
STALL2_O		Output	Output to stall the pipeline of core2	High
APBI_I	*	Input	APB slave output signals, includes interrupts	-
APBO_O	*	Output	APB slave output signals, includes interrupts	-

Signals ICNT1_I and ICNT2_I come from the pipelines of the cores. These signals have two bits each, one bit per lane. When one instruction is committed in one of the lanes, the corresponding bit will be set to 1 during one cycle.

Signals STALL1_O and STALL2_O come from SafeDE to the caches controllers and pipelines of the cores. These signals hold the pipelines of the cores when they are set to 1.

SafeDE is designed to work with two cores. The pair of signals ICNT_1 and STALL_1 have to be wired to the pipeline of the same core. This core will be considered the core1 and will have to write the critical section 1 register 4.2 explained in the section 4. The same reasoning is applicable to core2.

3 CONFIGURATION OPTIONS

Table 3.1 shows the configuration parameters (VDHL generics) exposed by *apb_wrapper.vhd*.

Table 3.1: Configuration options (VHDL ports)

Generic	**Function**	**Allowed range**	**Default**
PADDR	APB base address	0 to 16#fff#	0
PMASK	APB address mask	0 to 16#fff#	16#fff#
PINDEX	APB slave index	0 to 16#fff#	16#fff#
PIRQ	APB interrupt index	0 to 16#fff#	16#fff#
LANES_NUMBER	Lanes of the processor	1 to 5	2
REGISTER_OUTPUT	Number of registers at the stall ouput signals (if 0 no registers)	-	0
REGISTER_INPUT	Number of registers at the icnt input signals (if 0 no registers)	-	0
MIN_STAGGERING_INIT	If the minimum staggering threshold is not changed through the software, its value will be this parameter	5 to 32740	20
EN_CYCLES_LIMIT	Maximum number of cycles allowed between setting to 1 one of the critical section registers 4.2 4.3 and the other before raising an interrupt	ALL	500

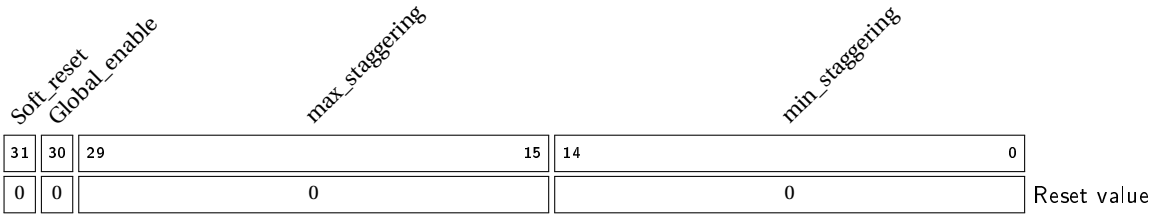
4 OPERATION

4.1 GENERAL

SafeDE is controlled through three internal registers:

The output signals used to stall the cores 2.1 are anded with the bit 30 of the configuration register 4.1. If this bit is set to 0, SafeDE will be able to stop neither of the cores. If bit 31 "soft_reset" is set to 1, all SafeDE registers are set to its reset value except the configuration register. In the same register, there are 15 bits to configure the maximum and the minimum staggering allowed. When the staggering is smaller or bigger than these thresholds, the trail or the head core will be stalled until the staggering is within limits again. If the value of "max_staggering" is kept to 0, the value of the maximum staggering will be 32750 to avoid overflows. If "min_staggering" is kept to 0, the value of the maximum staggering will be the value of the VHDL generics min_staggering_init explained in table 3.1.

Register 4.1: SAFEDE CONFIGURATION REGISTER 0 (0x00)

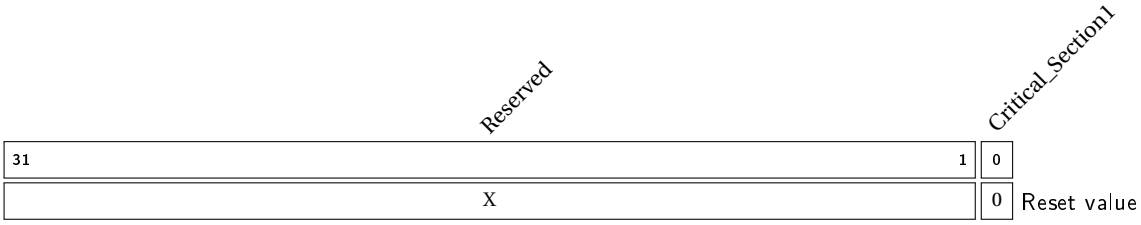


To choose the correct value for maximum and minimum staggering is important to take into account the configurations options in section 3. If SafeDE is configured to work with a two-lanes core, it means that in each cycle, two instructions can be committed. If we choose a minimum staggering of 15 instructions, that means that each time that the staggering is equal to or smaller than 15, the trail core will be stalled. However, at some point, the staggering could be 16, and the trail core could commit two instructions. In this scenario, the staggering would become 14, and the minimum staggering threshold would be exceeded. Suppose the VHDL generic "register_output" in table 3.1 is set to 1 for timing issues. In that case, the minimum staggering threshold can be exceeded by three instructions instead of one due to the delay between the assertion of the stall signal and the time that it takes effect in the pipeline. In conclusion, depending on the configuration, the threshold can be exceeded by a few instructions. In case a low minimum staggering threshold is chosen, this could cause SafeDE malfunction if the staggering gets 0 or lower.

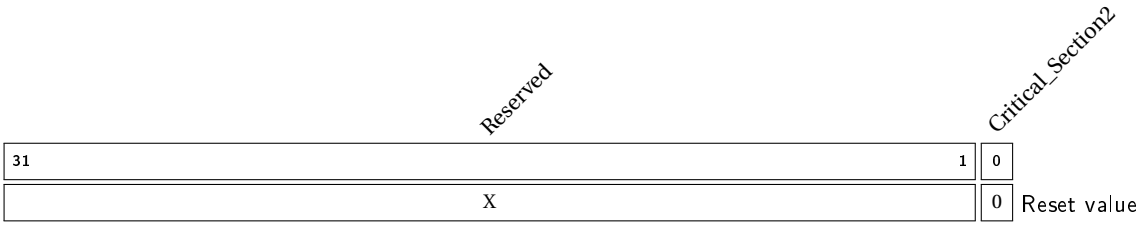
Apart from the configuration register, SafeDE has two other registers 4.2 and 4.3. The function of these registers is to indicate SafeDE that the core has started the critical task. Each core has assigned one critical section register, and it has to set its first bit before starting the critical section. Once the register is set to '1', SafeDE will start counting the executed instructions

by that core. In this way, the instructions are counted starting from the same point. It could happen that the write instruction in the "Critical_section" register had some delay because of the write buffer. Suppose the write buffers of the two cores are not equally filled when the write instruction is fetched. In that case, both cores could execute a different number of instructions, and the staggering computed by SafeDE could vary some instructions from the real value. This can be avoided by reading the critical section register right after writing it (in the next instruction).

Register 4.2: CRITICAL SECTION 1 REGISTER 1 (0x04)



Register 4.3: CRITICAL SECTION 2 REGISTER 2 (0x08)



It doesn't matter which core enters first the critical section (core1 or core2). The first core entering the critical section will take the role of head core and vice-versa. A mechanism also raises an interrupt if one core enters its critical section and the other doesn't. VHDL generic EN_CYCLES_LIMIT described in table 3.1 determine how many cycles can elapse between one core and the other enter their critical section.

4.2 STATISTICS

In addition to the configuration registers, SafeDE also has some registers that store statistics of the execution for debug purpuses. All these registers are shown in figure 4.1.

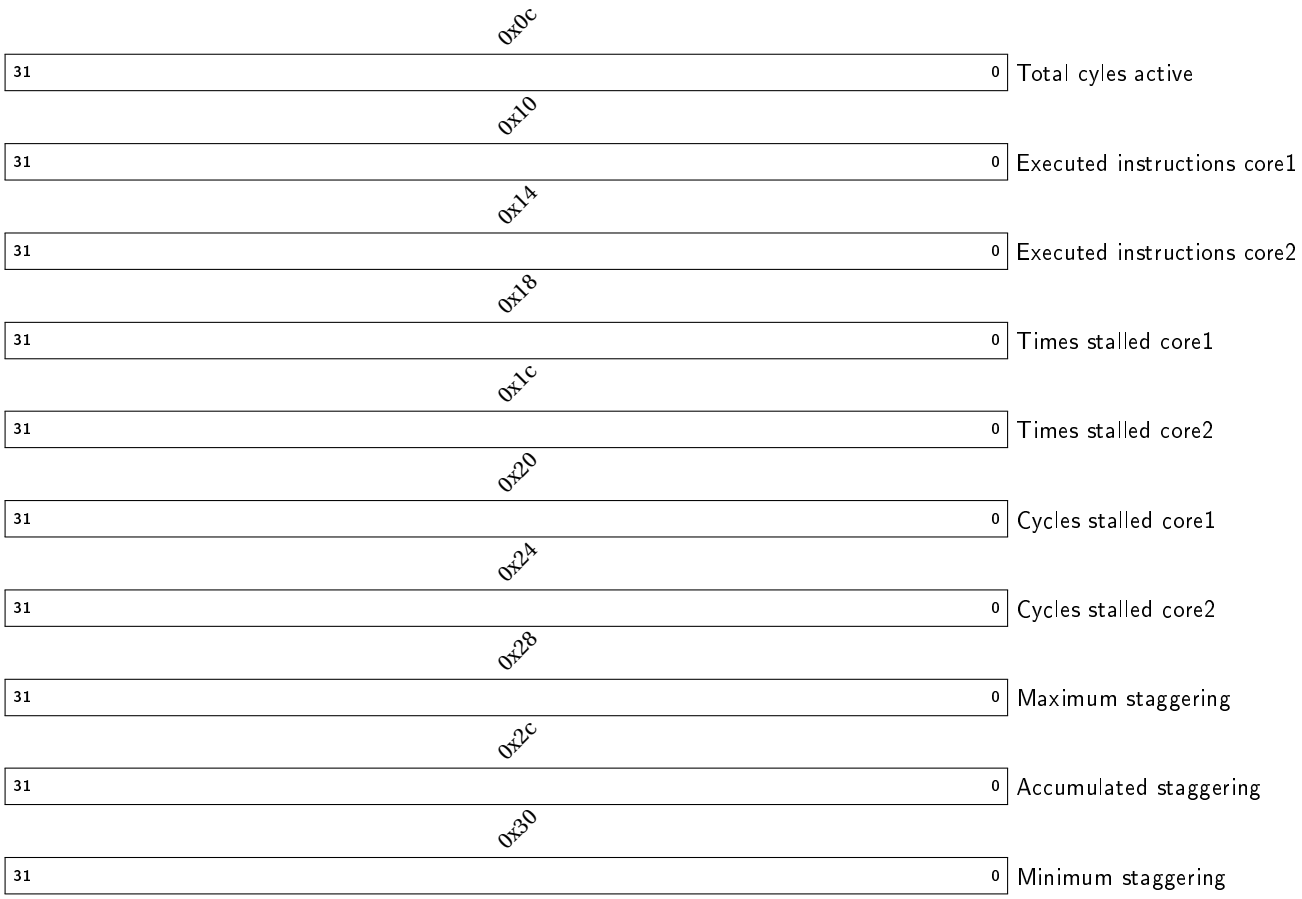


Figure 4.1: Statistics registers

The maximum staggering register (0x28) stores the maximum staggering achieved during the execution. The minimum staggering register (0x30) stores the minimum staggering achieved during the execution. However, this register is only updated once both cores have entered the critical section and the staggering is equal to or bigger than the minimum threshold. Each cycle is added to the value of the Accumulated staggering register (0x2C) the value of the staggering. The average staggering can be derived by dividing the value of this register by the number of cycles active.

All statistic registers except the minimum staggering register are updated once one core has entered its critical section (head core) and until the head core has finished its critical section.

To reset these registers, the soft_reset bit in the configuration register 4.1 has to be set to 1. None of these registers have any mechanism to prevent overflow.

4.3 SOFTWARE SUPPORT

The unit can be configured by the user at a low level following the description of previous sections. In addition we provide a small bare-metal driver under the path *grlib/software/noelv/BSC_tests/BSC_libraries/light-lockstep*.

The driver is composed of three files.

- *lightlock_vars.h*: Defines a set of constants with the RTL parameters and the memory position of the lockstep within the memory map of the SoC.
- *lightlock.h*: It contains the prototypes of each function of the driver.
- *lightlock.c*: Contains the definition of each of the functions.

The functions defined in the driver are:

- *lightlock_enable()*: Sets to 1 the 31 bit of the configuration register 4.1. It has to be called by any core to make SafeDE work.
- *lightlock_disable()*: Sets to 0 the 31 bit of the configuration register 4.1. SafeDE won't work independently of the value of the other registers.
- *lightlock_min_threshold(int min_threshold)*: It changes the value of the minimum staggering threshold.
- *lightlock_min_max_thresholds(int min_threshold, max_threshold)*: It changes the value of the minimum and the maximum staggering thresholds.
- *lightlock_start_criticalSec(int core)*: It has to be called right before entering the critical section. The argument of the function must be 1 or 2, for core1 or core2 respectively.
- *lightlock_finish_criticalSec(int core)*: It has to be called right after finishing the critical section. The argument of the function must be 1 or 2, for core1 or core2 respectively.
- *lightlock_report()*: It prints the values stored in the statistic registers.
- *lightlock_softreset()*: It resets all the registers in the module except the configuration register 4.1.

5 Library dependences

Table 5.1 shows the VHDL libraries used when instantiating SafeDE.

Table 5.1: Library dependencies

Library	Package	Imported units	Description
IEEE	std_logic_1164	Types	Standard logic types
IEEE	numeric_std	Functions and types	Arithmetic functions and signed and unsigned types
IEEE	math_real	Constants and functions	Mathematical functions and constants
GRLIB	amba	Signals	AMBA signal definitions
GRLIB	devices	Types	Device names and vendors
BSC	ligthlock_pkg	Instances and signals	Instances and type definitions for SafeDE
BSC	lightlock_module	Instance	Instance of SafeDE top module

6 Instantiation

An example design is provided in the context of De-RISC.

Listing 1: SafePMU instance example for gpp_sys

```
-- Include BSC library
library bsc;
use bsc.lightlock_module.all;

--Provide non-overlaping APB parameters
constant psidx_safede : integer := 5;
constant paddr_safede : integer := 6;
-- Chose minimum length: 256 bytes;
constant pmask_safede : integer := 16#fff#;
constant pirq_safede : integer := 14;

--Declare signals with as many bits as lanes are.
--Each bit of each lane has to be set to '1' during one
--cycle when one instruction is commited in that lane
singal icnt1 : std_logic_vector(1 downto 0);
singal icnt2 : std_logic_vector(1 downto 0);

--Declare signals that are able to stall the pipelines
```

```
signal lockstep_stall1 : std_logic;  -- For core1
signal lockstep_stall2 : std_logic;  -- For core2


--Instance of the unit
apb_lockstep_inst : apb_wrapper_lockstep
  generic map(
      pindex    => psidx_safede,
      paddr     => paddr_safede,
      pmask     => pmask_safede,
      pirq      => pirq_safede,
      lanes_number        => 2,
      register_output     => 0,
      register_input      => 0,
      en_cycles_limit     => 500,
      min_staggering_init => 20
      )
  port map(
      rstn     => rstn,
      clk      => clkm,
      apbi_i   => apbi(psidx_safede),
      apbo_o   => apbo(psidx_safede),
      icnt1_i  => icnt1,
      icnt2_i  => icnt2,
      stall1_o => lockstep_stall1,
      stall2_o => lockstep_stall2
      );
```