SafeDE User's Manual

Francisco Bas

July 20, 2022

CONTENTS

1 Overview

1.1 SafeDM description

SafeDM is a hardware Diversity Monitor that quantifies the diversity between two redundant processors to guarantee that CCF will not go unnoticed, and without needing to deploy lock-stepped cores. SafeDM takes advantage of the fact that diversity naturally exists due to the complexity of the system to develop the safety concept. Each cycle SafeDM computes one signature per core summarizing their internal state. Both signatures are compared and if they coincide, lack of diversity is reported.

SafeDM is instantiated in the SoC as an APB slave[1]. SafeDM raises an output each cycle that diversity is missed between cores. SafeDM also has an internal counter that is increased by one each cycle that there is lack of diversity. The value of this counter can be retrieved through the APB bus reading the appropriate register. SafeDM only reports a lack of diversity in the system and it is up to the user to determine the proper actions in the event of missing diversity.

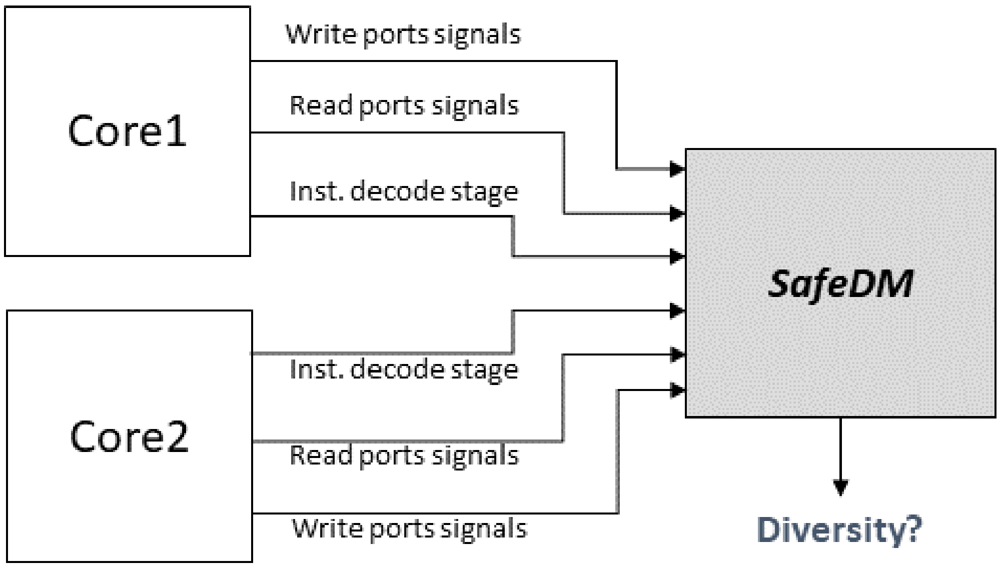


Figure 1.1: SafeDM simplified scheme.

Figure 1.1 shows a simplified scheme of SafeDM connected to two redundant processors.

[1]Based on AMBA 2.0 specification

Internally, SafeDM generates two different signatures that later are concatenated into a single one: instruction and data signatures. Some internal signals from the pipeline and the register file are required to calculate those signatures.

The signatures are generated by storing the last instructions processed in the decode stage (Figure 1.2) and the last values read from the register file ports in FIFOs (Figure 1.3).

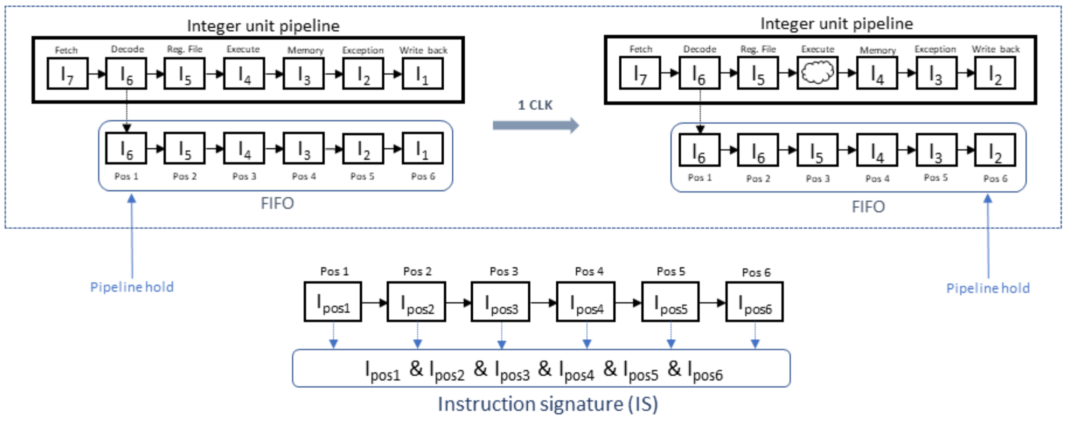


Figure 1.2: Instructions signature scheme.

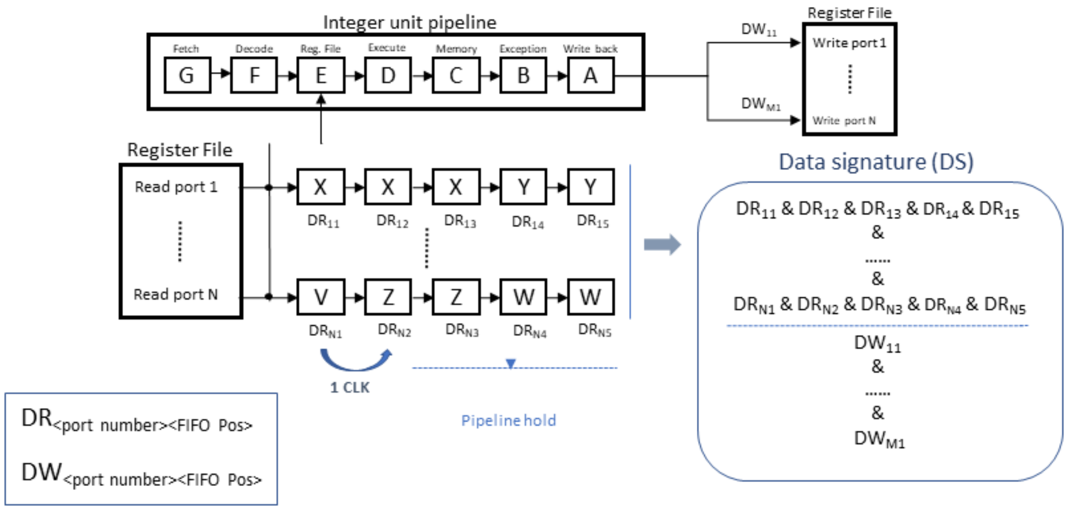


Figure 1.3: Registers signature scheme.

SafeDM offers some advantages in comparison to lockstep approaches:

- **Performance:** Classical tight-lockstep approach employs two cores that are seen as one by the user, thus halving the performance. A pair of redundant cores monitored by SafeDM can execute different tasks at any time.
- **Instructions stream:** Other lockstep approaches as lightlockstep avoid the performance penalty of the tightlockstep but require identical instructions streams. This prevents this solution from executing parallel applications or handling the system's inputs and outputs. SafeDM overcomes this limitation.
- **Low intrusiveness:** SafeDM can be implemented in a SoC performing only a few modifications in the design. SafeDM only needs a few signals to calculate the signatures.

1.2 LICENSING

This BSC IP core module is freely provided under the terms of the MIT License.

2 PRODUCT SPECIFICATION

2.1 CONFIGURATION OPTIONS

Several SafeDM parameters have to be correctly configured for SafeDM to adapt to the concrete architecture of the replicated cores. Table 2.1 shows SafeDM VHDL generics that allow configuring the hardware design.

Table 2.1 shows SafeDM configuration parameters (VDHL generics).

Table 2.1: Configuration options (VHDL ports)

Generic	Description	Allowed range	Default
CODING_METHOD	When it is set to 1, ECC (Error Correction Codes) redundant bits are generated for the instructions and register value inputs. SafeDM stores the ECCgenerated bits instead of storing all instruction and register bits saving FPGA resources. All the original input bits are stored if this generic is set to 0.	0 to 1	0
CODING_BITS_REG	Register FIFO width in bits. It depends on the selected coding method and the size of the registers of the cores.	-	64
CODING_BITS_INST	Instruction FIFO width in bits. It depends on the selected coding method and the size of the instructions of the cores.	-	32
REGS_FIFO_POS	Register FIFO number of positions.	-	5
INST_FIFO_POS	Instruction FIFO number of positions.	-	6

When the CODING_METHOD generic is set to 1, ECC are calculated applying the extended hamming code method. For inputs that are a power of 2, the output is $output\_bits = log2(input\_bits)+2$. For instance, if the instruction length of the cores is 32, the output length after applying the extended hamming code method will have $5+2=7$ bits. Therefore, CODING_BITS_INST generic must be set to 7.

As explained before in section 1.1, SafeDM generates two signatures storing the last instructions from the decode stage, and the last values read and written in the register file. To generate the signatures, it employs two FIFOs, one for each signature. The Instruction FIFO depth is configured using the INST_FIFO_POS generic. Its value must be equal to the number of pipeline stages from decode to the end of the pipeline including decode. The register FIFO depth is configured using the REGS_FIFO_POS generic. Its value must be equal to the number of pipeline stages from the register file stage to the end of the pipeline including the register file stage.

Different core architectures can have a different number of read and write ports in the register file or they can be single or multipleissue with a different number of lanes. This parameters can be configured in the file containing the type definitions *diversity_types_pkg.vhd* changing the value of the constants *lanes_number* and *read_ports*. Notice that SafeDM is not devised to work with outoforder superscalar processors.

2.2 Port Descriptions

Table 2.2 shows the interface of the IP core (VHDL ports).

Table 2.2: Signal descriptions (VHDL ports)

Signal name	Type	I/O	Description	Active
RSTN	STD_LOGIC	I	Reset	Low
CLK	STD_LOGIC	I	AHB master bus clock	-
APBI_PSEL_I	STD_LOGIC	I	APB slave selection signal	-
APBI_PADDR_I	STD_LOGIC_VECTOR	I	APB address signal	-
APBI_PENABLE_I	STD_LOGIC	I	APB enable signal	-
APBI_PWRITE_I	STD_LOGIC	I	APB read/write selection signal	-
APBI_PWDATA_I	STD_LOGIC_VECTOR	I	APB write data signal	-
APBI_PRDATA_O	STD_LOGIC_VECTOR	O	APB read data signal	-
INSTRUCTIONS_I	INSTRUCTION_TYPE_VECTOR	I	This type definition is a two-vector positions (one for each core) of the type *instruction_type*. This type includes an instruction value and a bit indicating if that instruction is valid per lane. The instructions value signals should come from the decode stage.	-
REGISTERS_I	REGISTER_TYPE_VECTOR	I	This type definition is a two-vector positions (one for each core) of the type *register_type*.This type definition includes a register value and a bit indicating if that register is read per register file read port. The registers value should come from the register file.	-
HOLD_I	STD_LOGIC_VECTOR	I	This signal is a two-bits vector containing the hold signal of each core. The hold signal is set to 1 when the core pipeline is held.	High
DIVERSISTY_LACK_O	STD_LOGIC	O	This signal rises when the signatures from both cores coincide meaning that there is no diversity between both core replicas.	High

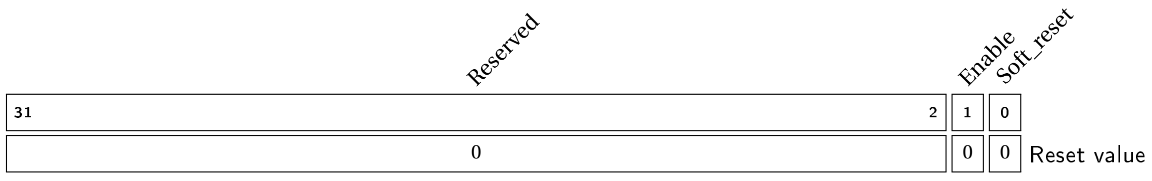
As explained later in section 4, SafeDM top VHDL file should be instantiated in an APB wrapper to drive the APB signals to SafeDM top. This wrapper should also define the input signals of the types *instruction_type_vector* and *register_type_vector* and wire those signals with the ones coming from the cores. This kind of design allows configuring the number of lanes and read ports without modifying the architecture ports.

2.3 Register space

SafeDM has only two registers that can be accessed through normal store and load operations.

The first register is used to reset SafeDM and start SafeDM operation:

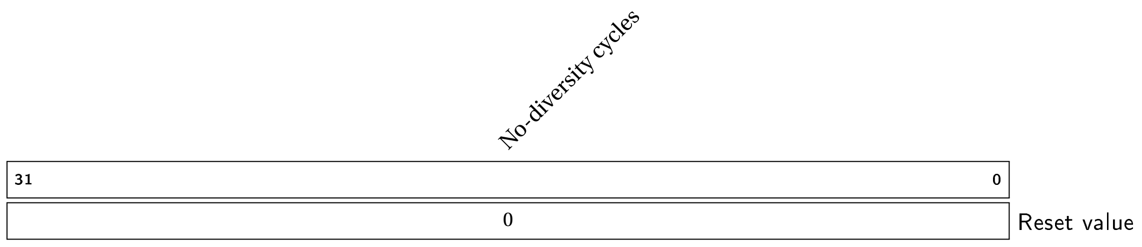
Register 2.1: SAFEDM CONFIGURATION REGISTER (0x00)



Once the enable bit is set to 1, SafeDM will start monitoring the diversity.

The second register contains the number of cycles in which SafeDM detected lack of diversity.

Register 2.2: NO-DIVERSITY CYCLES COUNTER (0x04)



To reset the count SafeDM has to be reset.

2.4 RESOURCES

3 CLOCKING AND RESET

3.1 CLOCKING

SafeDM has a single input clock. You should connect the appropriate AHB bus clock to this clock input.

3.2 RESETS

SafeDM reset *rstn* is an active-Low synchronous reset to the core. All registers are reset to power-on conditions. Apart from the external reset signal, SafeDM can be reset through the configuration APB register 2.1.

4 Implementing the IP core

As mention in the section 2.2, to implement SafeDM we need a wrapper. The function of this wrapper is to adapt the input signals. Below is an example of a wrapper used in a SoC designed by Cobham Gaisler. The SoC cores are Noel-V 64-bits cores. The noel-V core is dual-issue and has four read register file ports.

As you can see in the example, the APB signals are a custom type defined by gaisler that has to be converted into *std_logic* and *std_logic_vector* types. On the other hand the *std_logic* and *std_logic_vector* types of the instructions and registers inputs have to be converted into the *instruction_type_vector* and *register_type_vector types.*

Listing 1: SafeDM instance example

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;
-- AMBA 2.0 Cobham Gaisler's library
-- It defines AHB and APB specific types
library grlib;
use grlib.amba.all;
use grlib.devices.all;
-- Library containing SafeDM
library bsc;
use bsc.diversity_types_pkg.all;
use bsc.diversity_components_pkg.SafeDM_top;


entity apb_wrapper_diversity is
  generic (
    -- APB generics
    pindex : integer := 0;
    paddr  : integer := 0;
    pmask  : integer := 16#fff#;
    -- SafeDM configuration
    coding_method     : integer := 1;
    coding_bits_reg   : integer := 64;
    coding_bits_inst  : integer := 32;
    regs_fifo_pos     : integer := 5;
    inst_fifo_pos     : integer := 6
  );
  port (
    rstn    : in std_logic;
    clk     : in std_logic;
    -- Apb signals
    apbi_i : in apb_slv_in_type;
    apbo_o : out apb_slv_out_type;
    -- Instructions sign signals (both cores)
    -- Core 1
    lane1_inst_value_1_i : in std_logic_vector(31 downto 0);
    lane2_inst_value_1_i : in std_logic_vector(31 downto 0);
    inst_valid_1_i     : in std_logic_vector(1 downto 0);
    -- Core 2
    lane1_inst_value_2_i : in std_logic_vector(31 downto 0);
    lane2_inst_value_2_i : in std_logic_vector(31 downto 0);
    inst_valid_2_i     : in std_logic_vector(1 downto 0);
    -- Registers sign signals (both cores)
    -- Core 1
    port1_ren_1_i    : in std_logic;
    port2_ren_1_i    : in std_logic;
    port3_ren_1_i    : in std_logic;
    port4_ren_1_i    : in std_logic;
    port1_rdata_1_i  : in std_logic_vector(63 downto 0);
    port2_rdata_1_i  : in std_logic_vector(63 downto 0);
    port3_rdata_1_i  : in std_logic_vector(63 downto 0);
    port4_rdata_1_i  : in std_logic_vector(63 downto 0);
    -- Core 2
    port1_ren_2_i    : in std_logic;
    port2_ren_2_i    : in std_logic;
    port3_ren_2_i    : in std_logic;
    port4_ren_2_i    : in std_logic;
    port1_rdata_2_i  : in std_logic_vector(63 downto 0);
    port2_rdata_2_i  : in std_logic_vector(63 downto 0);
    port3_rdata_2_i  : in std_logic_vector(63 downto 0);
    port4_rdata_2_i  : in std_logic_vector(63 downto 0);
    -- Hold signals
    hold_1_i : in std_logic;
    hold_2_i : in std_logic;
    -- Lack of diversity
    diversity_lack_o : out std_logic
  );
  end;

architecture structural of apb_wrapper_diversity is
```

```
  -- APB configuration ----------------------------------------
  constant REVISION  : integer := 0;
  constant VENDOR_ID : integer := 16#0e#;
  constant DEVICE_ID : integer := 16#002#;

  constant PCONFIG : apb_config_type := (
  0 => ahb_device_reg (VENDOR_ID, DEVICE_ID, 0, REVISION, 0),
  1 => apb_iobar(paddr, pmask));
  -------------------------------------------------------------


  -- Inputs to SafeDM
  signal instructions : instruction_type_vector;
  signal registers    : register_type_vector;
  signal hold : std_logic_vector(1 downto 0);

begin

  -- SafeDM inputs are driven to the input register_type_vector and
  -- instruction_type_vector types.
  -- INSTRUCTIONS
  -- Core 1 assignation
  instructions(0).inst_value(0)  <= lane1_inst_value_1_i;
  instructions(0).inst_value(1)  <= lane2_inst_value_1_i;
  instructions(0).valid      <= inst_valid_1_i;
  -- Core 2 assignation
  instructions(1).inst_value(0)  <= lane1_inst_value_2_i;
  instructions(1).inst_value(1)  <= lane2_inst_value_2_i;
  instructions(1).valid      <= inst_valid_2_i;
  -- REGISTERS
  -- Core 1 assignement
  registers(0).value(0)  <= port1_rdata_1_i;
  registers(0).value(1)  <= port2_rdata_1_i;
  registers(0).value(2)  <= port3_rdata_1_i;
  registers(0).value(3)  <= port4_rdata_1_i;
  registers(0).ren(0)    <= port1_ren_1_i;
  registers(0).ren(1)    <= port2_ren_1_i;
  registers(0).ren(2)    <= port3_ren_1_i;
  registers(0).ren(3)    <= port4_ren_1_i;
  -- Core 2 assignement
  registers(1).value(0)  <= port1_rdata_2_i;
  registers(1).value(1)  <= port2_rdata_2_i;
  registers(1).value(2)  <= port3_rdata_2_i;
  registers(1).value(3)  <= port4_rdata_2_i;
registers(1).ren(0)   <= port1_ren_2_i;
registers(1).ren(1)   <= port2_ren_2_i;
registers(1).ren(2)   <= port3_ren_2_i;
registers(1).ren(3)   <= port4_ren_2_i;
-- Hold signals
hold <= hold_2_i & hold_1_i;


apb_diversity_inst : SafeDM_top
generic map(
  coding_method    => coding_method,
  coding_bits_reg  => coding_bits_reg,
  coding_bits_inst => coding_bits_inst,
  regs_fifo_pos    => regs_fifo_pos,
  inst_fifo_pos    => inst_fifo_pos
  )
port map(
  rstn           => rstn,
  clk            => clk,
  -- Apb signals: From Cobham Gailer's types
  -- to std_logic and std_logic_vectors
  apbi_psel_i      => apbi_i.psel(pindex),
  apbi_paddr_i     => apbi_i.paddr,
  apbi_penable_i   => apbi_i.penable,
  apbi_pwrite_i    => apbi_i.pwrite,
  apbi_pwdata_i    => apbi_i.pwdata,
  apbo_prdata_o    => apbo_o.prdata,
  -- Singals to calculate sigantures
  -- Instructions signature inputs
  instructions_i => instructions,
  -- Registers signatures inputs
  registers_i => registers,
  -- Hold signals
  hold => hold,
  -- Lack of diversity flag
  diversity_lack_o => diversity_lack_o
  );


-- APB bus output signals
apbo_o.pirq    <= (others => '0');
apbo_o.pindex  <= pindex;
apbo_o.pconfig <= PCONFIG;
```

```
end;
```

5 Library dependences

Table 5.1 shows the VHDL libraries used when instantiating SafeDM.

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
BSC	diversity_types_pkg	Types and constants	Custom SafeDM types and configuration constants
BSC	diversity_components_pkg	SafeDM components	SafeDM component definitions

6 Simulation

6.1 Test Bench

The design also contains a test bench. First, the test bench resets and enables SafeDM writing in the configuration register through the APB interface. The instructions and registers values fed as inputs to SafeDM are randomly generated using pseudo-random number generators. At some point, generated instructions and registers values of both cores coincide, simulating a lack of diversity scenario. Later, the counter storing the cycles where the system lacked diversity is read and compared with the expected value. The test bench will fail if the expected and obtained values do not coincide.

Changing the constants *lanes_number* and *read_ports* in the VHDL package *diversity_type_pkg.vhd* will also modify the test bench to generate the proper inputs for the SafeDM module.

6.2 Performing simulation

We have performed the simulation using *Questa Sim 10.7c*. You can reproduce the simulation using the Makefile inside the folder *tb*. Several recipes:

- **compile:** It creates the work and safety libraries and compiles all the VHDL files.
- **vsim-launch:** It compiles and launches the QuestaSim graphical user interface simulation.
- **vsim:** It compiles and runs the simulation in batch mode.
- **launch-tb:** It compiles, runs the simulation in batch mode and analyzes the test-bench results.
- **clean:** It removes temporal files.