Research Paper

Paper No.: 2023-JL-08
Online: ISSN: 2583-3707

ARAI Journal of Mobility Technology
https://araijournal.com/index.php/arai
Volume 3 • Issue 4 • October – December, 2023, pp. 856-882
DOI:10.37285/ajmt.3.4.8

Field Programmable Functional Safety Mechanisms Implementation using *FPGA*

Author's Affiliations

Priyank Sharma
STMicroelectronics Private LTD., Plot No.1, Knowledge Park III, Greater Noida 201308, India

***Corresponding Author: Ms. Priyank Sharma,** STMicroelectronics Private LTD., Plot No.1, Knowledge Park III, Greater Noida 201308, India.
Email: priyanks91@gmail.com

Article History
Received 27/03/2023
Accepted 23/04/2023
Published 15/12/2023

Keywords

ASIC, FPGA, Functional Safety, Diagnostic Coverage, Safety Mechanisms, ISO26262

Cite this paper as: Priyank Sharma (2023) "Field Programmable Functional Safety Mechanisms Implementation using *FPGA*, ARAI Journal of Mobility Technology, 3(4), pp. 856-882.
Available at: https://doi.org/10.37285/ajmt.3.4.8

Abstract

The topic of this report is to build an IC in form of the FPGA, consisting of different types of safety mechanisms, which depending on the application of the design, could help achieve either QM, ASIL B or ASIL D diagnostic coverage compliance. The selection of the Safety mechanisms could be made through configuration bits written in a register.

Tools and devices used

Software

- VITIS Model Composer
- MATLAB
- Simulink
- XILINX Toolbox for System generator
- VIVADO 2022 Suite

Hardware

- XILINX BASYS 3 Board
  - Artix – FPGA

Introduction

Until very recently, the global economy faced a crippling shortage of semiconductor devices, and the automotive market was especially affected. One of the key issues for the automotive market was that ASICs typically formed the bulk of the supplies, which meant that each application (there are hundreds of applications in a modern automobile) required a chip device built specifically for it. While this delivered great area, power and performance (commonly called PPA) statistics, it meant substantial manufacturing and design investments were required to deliver them, thus the true potential of 'economies of scale' was not realised. Worse, in light of global medical emergencies, it became difficult to develop such a large variety of ASICs.

FPGAs could generally solve some issues but in the interest of PPA, they are generally not favoured. But considering how the supply chain issues have affected operations, a new approach is being presented here, w herein an FPGA device is proposed, with configurable parameters that a customer may choose, so as to enable supporting various applications via same hardware.

One of the key performance benchmarks of a modern-day automotive ECU is Functional Safety. An ECU, primarily built of a SoC plus some other peripheral devices, which is expected to perform in key critical vehicle operations (braking system for e.g.), is expected to meet all functional safety compliance requirements, as outlined in the ISO26262, the de-facto standard for automotive Functional Safety.

One of the most stringent requirements as outlined in the Part-5 of the ISO26262 is that a HW module expected to fulfil ASIL D compliance shall have a PMHF (Probabilistic Metrics of HW faults) below 10 FIT, or 1 in 100M hours.

On the other hand, if an ECU is expected to fulfil a lower level of ASIL compliance, (ASIL B for e.g.), it is expected to have a PMHF below 100 FIT, or 1 in 10M hours.

Thus, to achieve the threshold of acceptance of risk (PMHF), the devices are built with a variety of design measures, otherwise also called as Functional Safety Mechanisms. These are SW or HW implementations that mitigate the effect of a failure of any part of the SoC. A Typical list of these is provided in the Part 11 of the ISO26262, Edition 2018. Of course, depending on the required ASIL compliance, the safety mechanisms are designed in a way to have lowest possible PPA penalties, while meeting the specified Functional Safety compliance requirements.

Via this report, an FPGA is proposed that can be used in 2 applications with same functional design but different 'PMHF requirements', i.e. implementation of Functional Safety Mechanisms is expected to be different. A customer can use the same FPGA HW and simply configure the applicability of safety mechanisms using a configuration setting, so that it can be used in 2 different safety criticality applications.

Abbreviations: FSM: Functional Safety Mechanism; PPA: Performance, Power, Area; ASIC: Application Specific Integrated Chip; FPGA: Field Programmable Gate Arrays; FIT: Failure in Time (1 FIT=1 in 109 hours); SoC: System on Chip; PMHF: Probabilistic Metric of Hardware Failures; DC: Diagnostic Coverage; FMEDA: Failure Mode Effects & Diagnostics Analysis

In addition to the above innovation, another novelty shall be the use of Vitis Model Composer to design and implement the hardware; with the assumption of no competence in any HDL.

The complete ASIC design flow shall be implemented.

Theory

Sources of Failure

The 2 major sources of failures in hardware are:

1. Random failures
2. Systematic Failures

Random Failures in ICs are caused by high energy cosmic particles, but can also be caused by trace amounts of charged particles emitted in chip material. [4] The probability of random failures is a function of the total number of transistors used and thus, higher the number of transistors in a chip, the higher the base failure rate. [5]

PMHF Computation

In the part 11 of the ISO26262, the standard gives the methodology for PMHF computation. The following figure explains that methodology.

PMHF is computed as per,

$$PMHF_{ite\_d} = \Sigma_{safety\ -\ Related\ hardware(HW)\ elements} (\lambda_{SPF} + \lambda_{RF} + \lambda_{DPF,L}) \quad [6]$$

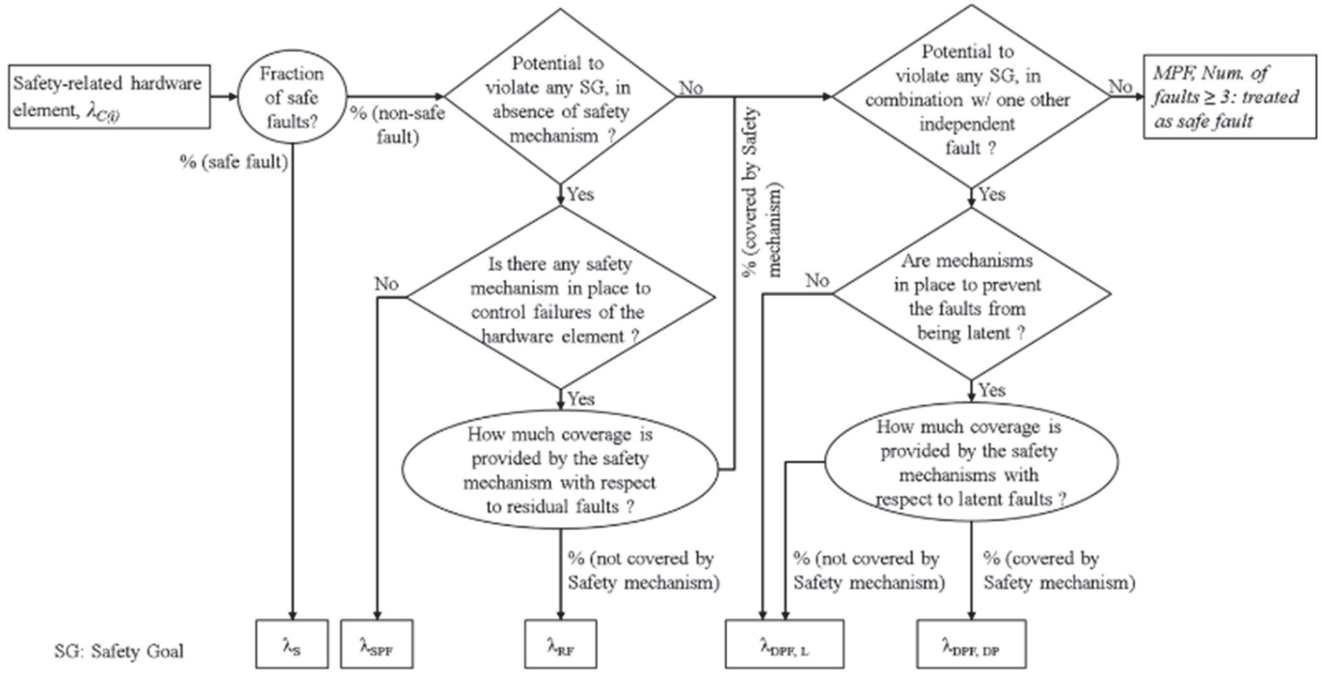
Diagnostic Coverage

Diagnostic coverage is a measure of effectiveness of the diagnostics implemented in the system. Mathematically, it is the ratio of the failures detected and/or controlled by a Safety mechanism to the total failures) in the element.

Goal

During design phase, certain hardware safety mechanisms are selected. While this approach works well in case of ASIC (Application Specific ICs), where the IC is designed for specific use cases, it makes it difficult for the same ASIC to be used in a different ASIL application, since the choice of the safety mechanisms would invariably be different, thus leading to a different design of silicon altogether.

Instead, the topic of this report is to build an IC in form of the FPGA, consisting of different types of safety mechanisms, which depending on the application of the design, could help achieve either ASIL B or ASIL D compliance.



System Design

A representative diagram of the system logic IP is as per follows:

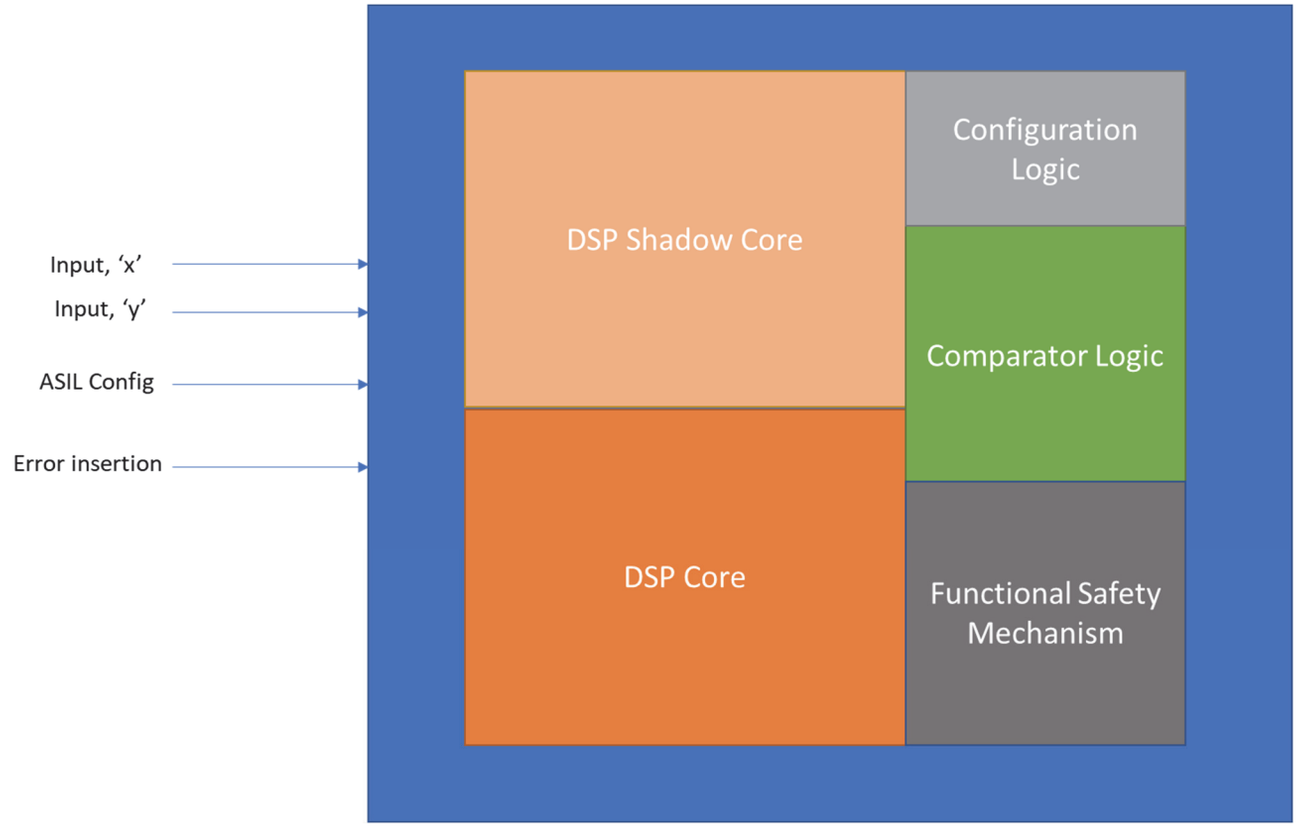
DSP Processor IP

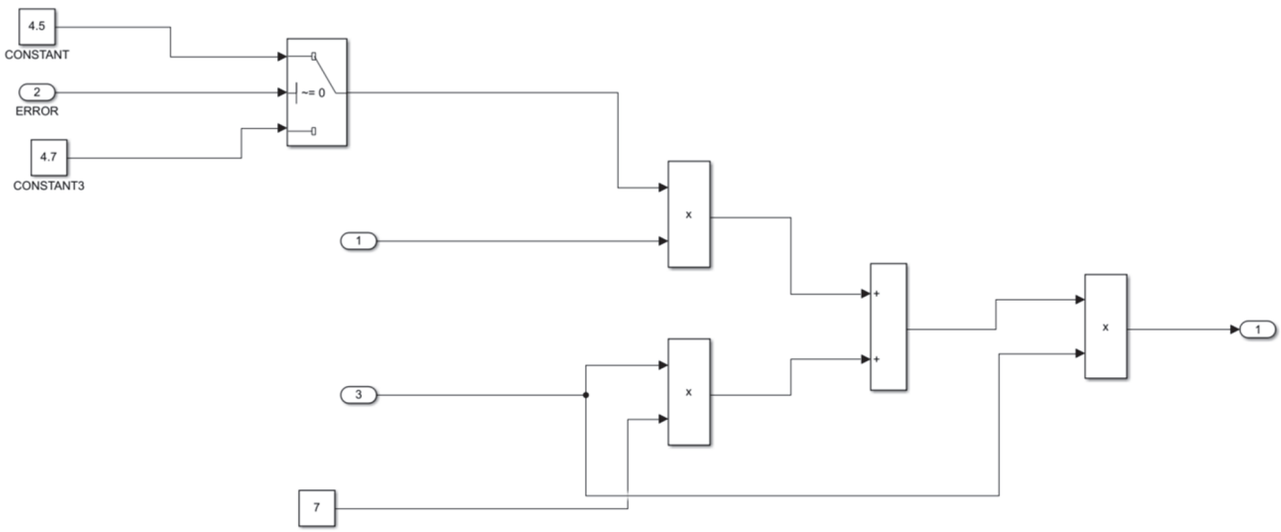
A digital signal processor (DSP) IP is a specialized microprocessor chip, with its architecture optimized for the operational needs of digital signal processing.

The IP developed for this report, implements the following mathematical operation,

$$F(x, y) = y(4.5x + 7y)$$

Its implementation in Simulink is as follows:





Shadow DSP

A shadow DSP IP implements the exact same DSP function, by fully independently running the same operations as the main DSP IP, but on physically redundant & separate digital logic. It helps to ensure the functional correctness of the overall DSP operation. This feature is also called 'Lockstep,' as explained later.

Its implementation in Simulink is as follows:

Lockstep Comparator logic

An additional comparator logic compares the output of both the main DSP and shadow DSP IP, and in the event of a difference, outputs a value of '0', which can basically be assumed to be the safe state. This is similar to a failing braking or powertrain system, where in the event of a failure, the system is expected to output a value of zero (in addition to ofcourse, the warnings).

Its implementation in Simulink is as follows:

Configuration logic

This module allows the configurability of the device to suit different safety criticality needs. For example, 1 of the configurations may allow activation of the strictest FSMs (to achieve ASIL D level) while another configuration may allow activations of somewhat weaker FSMs (to achieve ASIL B level)

Overall System Design

The overall system design is as per the following Simulink implementation:

System Functional Safety Analysis

This section shall provide an analytical Estimation of Device failure rate & diagnostic coverage.

FSM1 – Lockstep

The lockstep logic for the DSP IP has been implemented as follows. This implementation implies that each transaction is simultaneously processed by both the main and shadow IP. In case of a random error in either of the DSPs, the lockstep comparator logic would recognise the difference and immediately set the output signal value to '0', which is considered a safe state in this case.

Diagnostic Coverage

The diagnostic coverage (DC) can be calculated as follows:

$$DC = \frac{\text{Percentage of failures detected by lockstep logic}}{\text{Total failures in system}}$$

As can be expected, practically all failures can be handled by this logic, and thus, the DC can be estimated to be >99%. Infact, the ISO26262 part 11 provides certain estimates that conform to this estimate.

Cost

Since the lockstep requires a fully redundant computation path, the cost for lockstep is very high in terms of PPA penalties, and thus is avoided if possible.

FSM2 – Duplicated Multiplicand Blocks

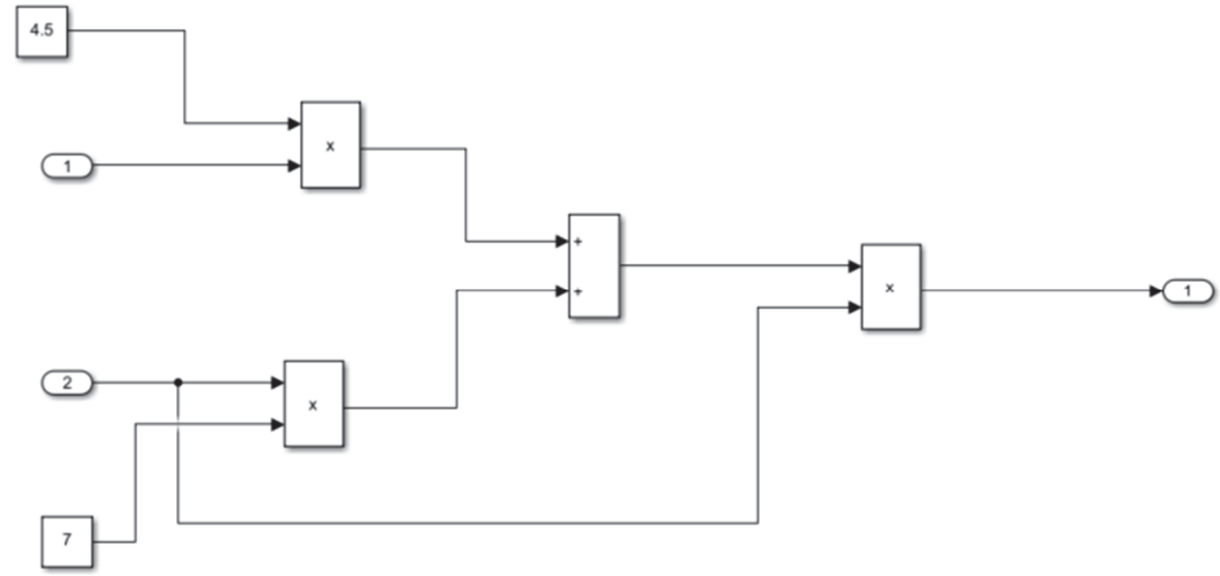
An alternate FSM being designed for this implementation, is for the multiplier blocks. Similar to Lockstep, in this case, only the 'multiplicand' operation is lock-stepped.

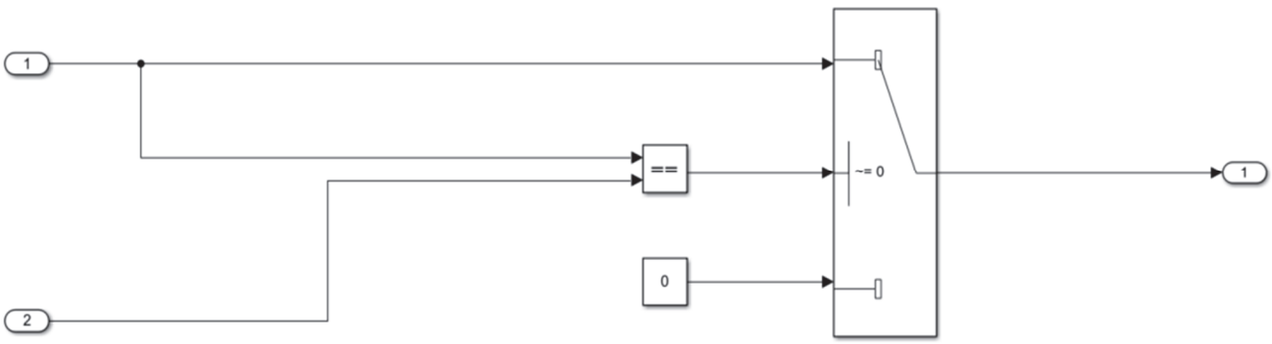
Diagnostic Coverage

It is expected that the DC of this implementation shall be lesser that that of lockstep, because it covers a small part of the logic, i.e. only the multiplicands.

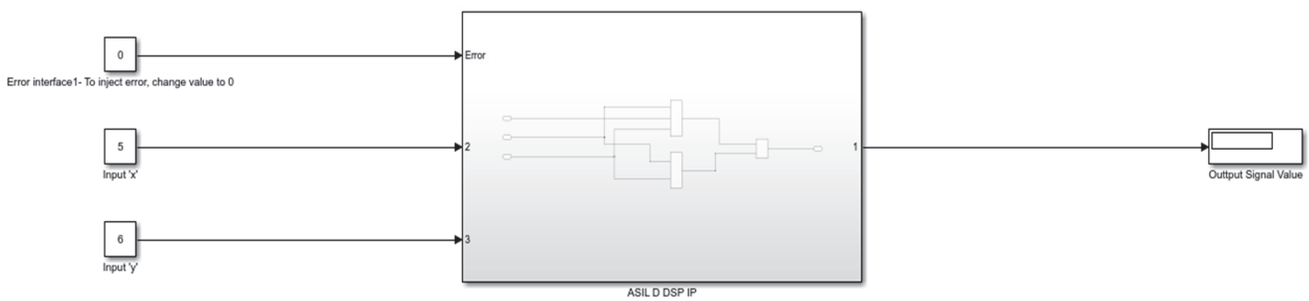
Cost

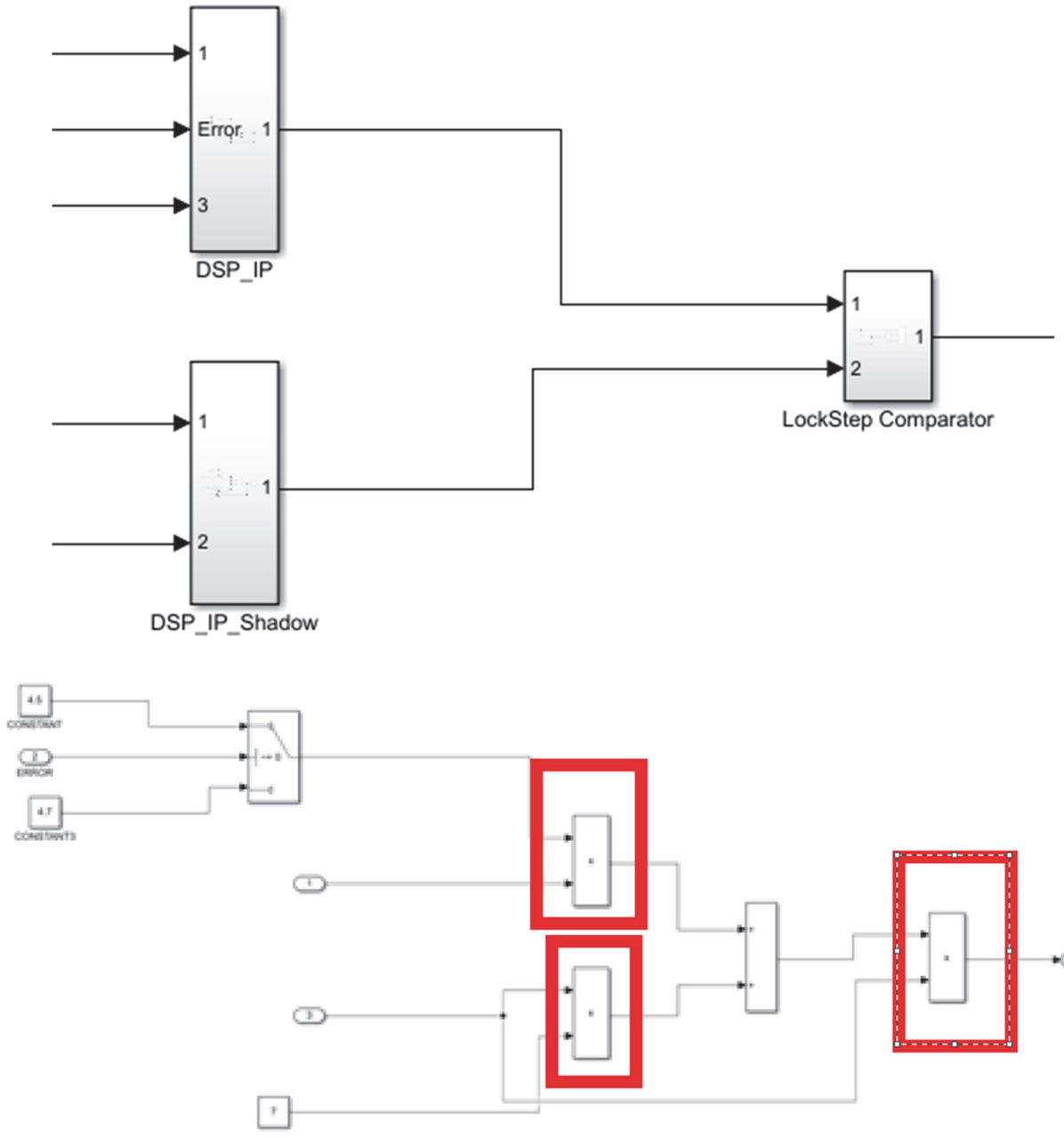
It is expected to have a much lower cost, compared to a lockstep logic, since only a small section is lock-stepped.


4.5
1
x
+
+
x
1
2
x
7


1
==
~= 0
1
0
2


System Generator
System
Generator


0
Error interface1- To inject error, change value to 0
5
Input 'x'
6
Input 'y'
Error
2
3
1
ASIL D DSP IP
Outtput Signal Value



Complete ASIC Design Flow using CAD/ EDA tools

Simulink Design

- Complete System developments as shown in earlier sections

VITIS Model Composer/ HDL Workflow Advisor

HDL Workflow Advisor facilitates RTL code (VHDL/Verilog) and testbench generation from a subsystem, performs synthesis tasks by invoking a supported third party synthesis tool, and annotates critical path information back to the system.

- **Output:** Generated RTL, test Bench, Validation Models, Synthesis Report

VIVADO

The complete ASIC Design flow was lso run in Vivado 2022.1 suite

- **Output:** Packaged DSP IP

Conclusions

This report demonstrates that an IP has been built, which allows tuning of FSMs via a configuration setting.

- This allows for the same HW (semiconductor chip) to be used in different applications of varying ASIL levels
- This eventually results in lower design and development costs for the development of said HW, while catering to a large variety of applications.
- The report also demonstrates how effective the VITIS Model Composes/ Simulink HDL Coder are, to allow ASIC/ FPGA development from a Systems' perspective. A system engineer with no HDL experience can still develop semiconductor designs with the aid of such tools.

Future Scope

This report work only demonstrates 'in theory' the configurability of Safety mechanisms in a Field Programmable Device.

This work, though initial, is very promising, and following avenues of research shall be pursued upon, by interested parties:

1. Validating PMHF Metrics via Fault Injections
   - It remains to be validated if the empirical values of assumed diagnostic coverages are reflected in the safety analysis
   - Sophisticated tools may be required to run fault injection campaigns to test the actual Diagnostic coverages
2. Validating the lower Power consumption assumptions
   - It remains to be validated if the increased power penalties due to increased digital logic of configuration registers are still lower than a higher ASIL compliant device catering to a lower ASIL application
   - It would have to be calculated on actual hardware while monitoring power consumption.
3. Validating lower cost assumptions
   - It remains to be validated if the design, development & manufacturing costs for 2 ASICs catering to different applications is still higher than 1 FPGA with higher complexity

References

[1] https://www.statista.com/statistics/277931/automotive-electronics-cost-as-a-share-of-total-car-cost-worldwide/
[2] https://fuse.wikichip.org/news/2207/tsmc-starts-5-nanometer-risk-production/
[3] https://www.iso.org/standard/68383.html
[4] https://tezzaron.com/media/soft_errors_1_1_secure.pdf
[5] https://www.iso.org/standard/69604.html
[6] https://www.mdpi.com/2076-3417/12/11/5456

Appendix 1: HDL Code for the Sub-system

```
-- -------------------------------------------------------------
--
-- File Name: hdl_prj\hdlsrc\DSP\ASIL_D_DSP_IP.vhd
-- Created: 2023-04-23 14:35:58
--
-- Generated by MATLAB 9.11 and HDL Coder 3.19
--
--
-- -------------------------------------------------------------
-- Rate and Clocking Details
-- -------------------------------------------------------------
-- Model base rate: 0.1
-- Target subsystem base rate: 0.1
--
--
-- Clock Enable  Sample Time
-- -------------------------------------------------------------
-- ce_out        0.1
-- -------------------------------------------------------------
--
--
-- Output Signal                 Clock Enable  Sample Time
-- -------------------------------------------------------------
-- OUTPUT1                       ce_out        0.1
-- -------------------------------------------------------------
--
-- -------------------------------------------------------------

-- -------------------------------------------------------------
--
-- Module: ASIL_D_DSP_IP
-- Source Path: DSP/ASIL D DSP IP
-- Hierarchy Level: 0
--
-- -------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY ASIL_D_DSP_IP IS
  PORT( clk               :   IN    std_logic;
        rstx              :   IN    std_logic;
        clk_enable        :   IN    std_logic;
        ERROR_rsvd        :   IN    std_logic_vector(63 DOWNTO 0);  -- double
        INPUT2            :   IN    std_logic_vector(63 DOWNTO 0);  -- double
        INPUT3            :   IN    std_logic_vector(63 DOWNTO 0);  -- double
        ce_out            :   OUT   std_logic;
        OUTPUT1           :   OUT   std_logic_vector(63 DOWNTO 0)  -- double
        );
END ASIL_D_DSP_IP;
```

```
ARCHITECTURE rtl OF ASIL_D_DSP_IP IS

  -- Component Declarations
  COMPONENT DSP_IP
    PORT( clk             :   IN    std_logic;
          rstx            :   IN    std_logic;
          enb             :   IN    std_logic;
          In1             :   IN    std_logic_vector(63 DOWNTO 0);  -- double
          ERROR_rsvd      :   IN    std_logic_vector(63 DOWNTO 0);  -- double
          In2             :   IN    std_logic_vector(63 DOWNTO 0);  -- double
          Out1            :   OUT   std_logic_vector(63 DOWNTO 0)  -- double
          );
  END COMPONENT;

  COMPONENT DSP_IP_SHADOW
    PORT( clk             :   IN    std_logic;
          rstx            :   IN    std_logic;
          enb              :   IN    std_logic;
          In1              :   IN    std_logic_vector(63 DOWNTO 0);  -- double
          In2              :   IN    std_logic_vector(63 DOWNTO 0);  -- double
          Out1             :   OUT   std_logic_vector(63 DOWNTO 0)  -- double
          );
  END COMPONENT;

  COMPONENT LOCKSTEP_COMPARATOR1
    PORT( clk               :   IN    std_logic;
          rstx              :   IN    std_logic;
          enb               :   IN    std_logic;
          In1               :   IN    std_logic_vector(63 DOWNTO 0);  -- double
          In2               :   IN    std_logic_vector(63 DOWNTO 0);  -- double
          Out1              :   OUT   std_logic_vector(63 DOWNTO 0)  -- double
          );
  END COMPONENT;

  -- Component Configuration Statements
  FOR ALL : DSP_IP
    USE ENTITY work.DSP_IP(rtl);

  FOR ALL : DSP_IP_SHADOW
    USE ENTITY work.DSP_IP_SHADOW(rtl);

  FOR ALL : LOCKSTEP_COMPARATOR1
    USE ENTITY work.LOCKSTEP_COMPARATOR1(rtl);

  -- Signals
  SIGNAL DSP_IP_out1            : std_logic_vector(63 DOWNTO 0);  -- ufix64
  SIGNAL DSP_IP_SHADOW_out1     : std_logic_vector(63 DOWNTO 0);  -- ufix64
  SIGNAL LOCKSTEP_COMPARATOR1_out1 : std_logic_vector(63 DOWNTO 0);  -- ufix64
```

```
BEGIN
  u_DSP_IP : DSP_IP
    PORT MAP( clk => clk,
              rstx => rstx,
              enb => clk_enable,
              In1 => INPUT2,  -- double
              ERROR_rsvd => ERROR_rsvd,  -- double
              In2 => INPUT3,  -- double
              Out1 => DSP_IP_out1  -- double
              );

  u_DSP_IP_SHADOW : DSP_IP_SHADOW
    PORT MAP( clk => clk,
              rstx => rstx,
              enb => clk_enable,
              In1 => INPUT2,  -- double
              In2 => INPUT3,  -- double
              Out1 => DSP_IP_SHADOW_out1  -- double
              );
  u_LOCKSTEP_COMPARATOR1 : LOCKSTEP_COMPARATOR1
    PORT MAP( clk => clk,
              rstx => rstx,
              enb => clk_enable,
              In1 => DSP_IP_out1,  -- double
              In2 => DSP_IP_SHADOW_out1,  -- double
              Out1 => LOCKSTEP_COMPARATOR1_out1  -- double
              );

  ce_out <= clk_enable;

  OUTPUT1 <= LOCKSTEP_COMPARATOR1_out1;

END rtl;
```

Appendix 2: Package File

```
-- -------------------------------------------------------------
--
-- File Name: hdl_prj\hdlsrc\DSP\ASIL_D_DSP_IP_pac.vhd
-- Created: 2023-04-23 14:35:58
--
-- Generated by MATLAB 9.11 and HDL Coder 3.19
--
-- -------------------------------------------------------------


LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
```

```
PACKAGE ASIL_D_DSP_IP_pac IS
  TYPE vector_of_unsigned11 IS ARRAY (NATURAL RANGE <>) OF unsigned(10 DOWNTO 0);
  TYPE vector_of_signed57 IS ARRAY (NATURAL RANGE <>) OF signed(56 DOWNTO 0);
  TYPE vector_of_unsigned56 IS ARRAY (NATURAL RANGE <>) OF unsigned(55 DOWNTO 0);
  TYPE vector_of_unsigned52 IS ARRAY (NATURAL RANGE <>) OF unsigned(51 DOWNTO 0);
  TYPE vector_of_unsigned106 IS ARRAY (NATURAL RANGE <>) OF unsigned(105 DOWNTO
0);
  TYPE vector_of_signed14 IS ARRAY (NATURAL RANGE <>) OF signed(13 DOWNTO 0);
  TYPE vector_of_std_logic_vector64 IS ARRAY (NATURAL RANGE <>) OF
std_logic_vector(63 DOWNTO 0);
END ASIL_D_DSP_IP_pac;
```

Appendix 3: HDL Code Generation Check report

HDL Code Generation Check Report for 'DSP/ASIL D DSP IP' open model 'DSP/ASIL D DSP IP'
Generated on 2023-04-23 14:35:59

HDL check for 'DSP' complete with 0 errors, 0 warnings, and 2 messages.

The following table describes blocks for which errors, warnings or messages were reported.

Simulink Blocks and resources	Level	Description
DSP	Message	'AdaptivePipelining' is set to 'Off' for the model. 'AdaptivePipelining' can improve the achievable clock frequency and reduce the area usage on FPGA boards. To enable adaptive pipelining, set the option to 'On'. When adaptive pipelining is enabled, it inserts pipeline registers to create patterns that efficiently map blocks to DSP units on the target FPGA device.
DSP	Message	'LUTMapToRAM' is set to 'On' for the model. This option is used to map lookup tables to a block RAM in hardware. To disable pipeline insertion for mapping lookup tables to RAM, set the option to 'Off'.

Appendix 4: HDL Code for Test Bench

```
-- -------------------------------------------------------------
--
-- File Name: hdl_prj\hdlsrc\DSP\ASIL_D_DSP_IP_tb.vhd
-- Created: 2023-04-23 14:36:04
--
-- Generated by MATLAB 9.11 and HDL Coder 3.19
--
--
```

```
-- -------------------------------------------------------------
-- Rate and Clocking Details
-- -------------------------------------------------------------
-- Model base rate: 0.1
-- Target subsystem base rate: 0.1
--
--
-- Clock Enable  Sample Time
-- -------------------------------------------------------------
-- ce_out        0.1
-- -------------------------------------------------------------
--
--
-- Output Signal                 Clock Enable  Sample Time
-- -------------------------------------------------------------
-- OUTPUT1                       ce_out        0.1
-- -------------------------------------------------------------
--
-- -------------------------------------------------------------


-- -------------------------------------------------------------
--
-- Module: ASIL_D_DSP_IP_tb
-- Source Path:
-- Hierarchy Level: 0
--
-- -------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_textio.ALL;
USE IEEE.float_pkg.ALL;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
LIBRARY STD;
USE STD.textio.ALL;
LIBRARY work;
USE work.ASIL_D_DSP_IP_pac.ALL;
USE work.ASIL_D_DSP_IP_tb_pac.ALL;

ENTITY ASIL_D_DSP_IP_tb IS
END ASIL_D_DSP_IP_tb;


ARCHITECTURE rtl OF ASIL_D_DSP_IP_tb IS

  -- Component Declarations
  COMPONENT ASIL_D_DSP_IP
    PORT( clk               :   IN    std_logic;
          rstx              :   IN    std_logic;
          clk_enable        :   IN    std_logic;
          ERROR_rsvd        :   IN    std_logic_vector(63 DOWNTO 0);  -- double
```

```
        INPUT2            :   IN    std_logic_vector(63 DOWNTO 0);  -- double
        INPUT3            :   IN    std_logic_vector(63 DOWNTO 0);  -- double
        ce_out            :   OUT   std_logic;
        OUTPUT1           :   OUT   std_logic_vector(63 DOWNTO 0)  -- double
        );
END COMPONENT;

-- Component Configuration Statements
FOR ALL : ASIL_D_DSP_IP
  USE ENTITY work.ASIL_D_DSP_IP(rtl);

-- Signals
SIGNAL clk                : std_logic;
SIGNAL rstx               : std_logic;
SIGNAL clk_enable         : std_logic;
SIGNAL rawData_ERROR_rsvd  : std_logic_vector(63 DOWNTO 0);  -- ufix64
SIGNAL rawData_INPUT2      : std_logic_vector(63 DOWNTO 0);  -- ufix64
SIGNAL rawData_INPUT3      : std_logic_vector(63 DOWNTO 0);  -- ufix64
SIGNAL OUTPUT1_done        : std_logic;  -- ufix1
SIGNAL rdEnb               : std_logic;
SIGNAL OUTPUT1_done_enb    : std_logic;  -- ufix1
SIGNAL OUTPUT1_addr        : unsigned(6 DOWNTO 0);  -- ufix7
SIGNAL OUTPUT1_active      : std_logic;  -- ufix1
SIGNAL holdData_INPUT3     : std_logic_vector(63 DOWNTO 0);  -- ufix64
SIGNAL INPUT3_offset       : std_logic_vector(63 DOWNTO 0);  -- ufix64
SIGNAL INPUT3_1            : std_logic_vector(63 DOWNTO 0);  -- ufix64
SIGNAL holdData_INPUT2     : std_logic_vector(63 DOWNTO 0);  -- ufix64
SIGNAL INPUT2_offset       : std_logic_vector(63 DOWNTO 0);  -- ufix64
SIGNAL INPUT2_1            : std_logic_vector(63 DOWNTO 0);  -- ufix64
SIGNAL holdData_ERROR_rsvd: std_logic_vector(63 DOWNTO 0);  -- ufix64
SIGNAL ERROR_rsvd_offset  : std_logic_vector(63 DOWNTO 0);  -- ufix64
SIGNAL ERROR_rsvd_1       : std_logic_vector(63 DOWNTO 0);  -- ufix64
SIGNAL check1_done        : std_logic;  -- ufix1
SIGNAL snkDonen           : std_logic;
SIGNAL resetn             : std_logic;
SIGNAL tb_enb             : std_logic;
SIGNAL tb_enb_delay       : std_logic;
SIGNAL ce_out             : std_logic;
SIGNAL OUTPUT1            : std_logic_vector(63 DOWNTO 0);  -- ufix64
SIGNAL OUTPUT1_enb        : std_logic;  -- ufix1
SIGNAL OUTPUT1_lastAddr   : std_logic;  -- ufix1
SIGNAL OUTPUT1_chkcnt     : unsigned(5 DOWNTO 0);  -- ufix6
SIGNAL OUTPUT1_ignCntDone : std_logic;  -- ufix1
SIGNAL OUTPUT1_needToCount: std_logic;  -- ufix1
SIGNAL OUTPUT1_chkenb     : std_logic;  -- ufix1
SIGNAL OUTPUT1_chkdata    : std_logic;  -- ufix1
SIGNAL OUTPUT1_addr_delay_1 : unsigned(6 DOWNTO 0);  -- ufix7
SIGNAL OUTPUT1_expected     : std_logic_vector(63 DOWNTO 0);  -- ufix64
SIGNAL OUTPUT1_ref          : std_logic_vector(63 DOWNTO 0);  -- ufix64
SIGNAL OUTPUT1_testFailure  : std_logic;  -- ufix1
```

```
BEGIN
  u_ASIL_D_DSP_IP : ASIL_D_DSP_IP
    PORT MAP( clk => clk,
              rstx => rstx,
              clk_enable => clk_enable,
              ERROR_rsvd => ERROR_rsvd_1,  -- double
              INPUT2 => INPUT2_1,  -- double
              INPUT3 => INPUT3_1,  -- double
              ce_out => ce_out,
              OUTPUT1 => OUTPUT1  -- double
              );

  -- Data source for ERROR_rsvd
  rawData_ERROR_rsvd <= X"3fb999999999999a";

  -- Data source for INPUT2
  rawData_INPUT2 <= X"4014000000000000";

  -- Data source for INPUT3
  rawData_INPUT3 <= X"4018000000000000";

  OUTPUT1_done_enb <= OUTPUT1_done AND rdEnb;


  OUTPUT1_active <= '1' WHEN OUTPUT1_addr /= to_unsigned(16#64#, 7) ELSE
      '0';

  -- holdData reg for INPUT_Y_out1
  stimuli_INPUT_Y_out1_process: PROCESS (clk)
  BEGIN
    IF clk'event AND clk = '1' THEN
      IF rstx = '1' THEN
        holdData_INPUT3 <= (OTHERS => 'X');
      ELSE
        holdData_INPUT3 <= rawData_INPUT3;
      END IF;
    END IF;
  END PROCESS stimuli_INPUT_Y_out1_process;

  stimuli_INPUT_Y_out1_1: PROCESS (rawData_INPUT3, rdEnb)
  BEGIN
    IF rdEnb = '0' THEN
      INPUT3_offset <= holdData_INPUT3;
    ELSE
      INPUT3_offset <= rawData_INPUT3;
    END IF;
  END PROCESS stimuli_INPUT_Y_out1_1;

  INPUT3_1 <= INPUT3_offset AFTER 2 ns;

  -- holdData reg for INPUT_X_out1
```

```
  stimuli_INPUT_X_out1_process: PROCESS (clk)
  BEGIN
    IF clk'event AND clk = '1' THEN
      IF rstx = '1' THEN
        holdData_INPUT2 <= (OTHERS => 'X');
      ELSE
        holdData_INPUT2 <= rawData_INPUT2;
      END IF;
    END IF;
  END PROCESS stimuli_INPUT_X_out1_process;

  stimuli_INPUT_X_out1_1: PROCESS (rawData_INPUT2, rdEnb)
  BEGIN
    IF rdEnb = '0' THEN
      INPUT2_offset <= holdData_INPUT2;
    ELSE
      INPUT2_offset <= rawData_INPUT2;
    END IF;
  END PROCESS stimuli_INPUT_X_out1_1;

  INPUT2_1 <= INPUT2_offset AFTER 2 ns;

  -- holdData reg for ERROR_INTERFACE_1_TO_INJECT_ERROR_CHANGE_VALUE_TO_ZERO_out1
  stimuli_ERROR_INTERFACE_1_TO_INJECT_ERROR_CHANGE_VALUE_TO_ZERO_out1_process:
PROCESS (clk)
  BEGIN
    IF clk'event AND clk = '1' THEN
      IF rstx = '1' THEN
        holdData_ERROR_rsvd <= (OTHERS => 'X');
      ELSE
        holdData_ERROR_rsvd <= rawData_ERROR_rsvd;
      END IF;
    END IF;
  END PROCESS
stimuli_ERROR_INTERFACE_1_TO_INJECT_ERROR_CHANGE_VALUE_TO_ZERO_out1_process;

  stimuli_ERROR_INTERFACE_1_TO_INJECT_ERROR_CHANGE_VALUE_TO_ZERO_out1_1: PROCESS
(rawData_ERROR_rsvd, rdEnb)
  BEGIN
    IF rdEnb = '0' THEN
      ERROR_rsvd_offset <= holdData_ERROR_rsvd;
    ELSE
      ERROR_rsvd_offset <= rawData_ERROR_rsvd;
    END IF;
  END PROCESS
stimuli_ERROR_INTERFACE_1_TO_INJECT_ERROR_CHANGE_VALUE_TO_ZERO_out1_1;

  ERROR_rsvd_1 <= ERROR_rsvd_offset AFTER 2 ns;

  snkDonen <=  NOT check1_done;
```

```
  resetn <=  NOT rstx;

  tb_enb <= resetn AND snkDonen;

  -- Delay inside enable generation: register depth 1
  u_enable_delay_process: PROCESS (clk)
  BEGIN
    IF clk'event AND clk = '1' THEN
      IF rstx = '1' THEN
        tb_enb_delay <= '0';
      ELSE
        tb_enb_delay <= tb_enb;
      END IF;
    END IF;
  END PROCESS u_enable_delay_process;


  rdEnb <= tb_enb_delay WHEN check1_done = '0' ELSE
      '0';

  clk_enable <= rdEnb AFTER 2 ns;

  rstx_gen: PROCESS
  BEGIN
    rstx <= '1';
    WAIT FOR 20 ns;
    WAIT UNTIL clk'event AND clk = '1';
    WAIT FOR 2 ns;
    rstx <= '0';
    WAIT;
  END PROCESS rstx_gen;

  clk_gen: PROCESS
  BEGIN
    clk <= '1';
    WAIT FOR 5 ns;
    clk <= '0';
    WAIT FOR 5 ns;
    IF check1_done = '1' THEN
      clk <= '1';
      WAIT FOR 5 ns;
      clk <= '0';
      WAIT FOR 5 ns;
      WAIT;
    END IF;
  END PROCESS clk_gen;

  OUTPUT1_enb <= ce_out AND OUTPUT1_active;

  -- Count limited, Unsigned Counter
  --  initial value   = 0
```

```
--  step value      = 1
--  count to value  = 100
c_4_process : PROCESS (clk)
BEGIN
  IF clk'EVENT AND clk = '1' THEN
    IF rstx = '1' THEN
      OUTPUT1_addr <= to_unsigned(16#00#, 7);
    ELSIF OUTPUT1_enb = '1' THEN
      IF OUTPUT1_addr >= to_unsigned(16#64#, 7) THEN
        OUTPUT1_addr <= to_unsigned(16#00#, 7);
      ELSE
        OUTPUT1_addr <= OUTPUT1_addr + to_unsigned(16#01#, 7);
      END IF;
    END IF;
  END IF;
END PROCESS c_4_process;


OUTPUT1_lastAddr <= '1' WHEN OUTPUT1_addr >= to_unsigned(16#64#, 7) ELSE
    '0';

OUTPUT1_done <= OUTPUT1_lastAddr AND resetn;

-- Delay to allow last sim cycle to complete
checkDone_1_process: PROCESS (clk)
BEGIN
  IF clk'event AND clk = '1' THEN
    IF rstx = '1' THEN
      check1_done <= '0';
    ELSIF OUTPUT1_done_enb = '1' THEN
      check1_done <= OUTPUT1_done;
    END IF;
  END IF;
END PROCESS checkDone_1_process;


OUTPUT1_ignCntDone <= '1' WHEN OUTPUT1_chkcnt /= to_unsigned(16#20#, 6) ELSE
    '0';

OUTPUT1_needToCount <= ce_out AND OUTPUT1_ignCntDone;

-- Count limited, Unsigned Counter
--  initial value   = 0
--  step value      = 1
--  count to value  = 32
OUTPUT1_IgnoreDataChecking_process : PROCESS (clk)
BEGIN
  IF clk'EVENT AND clk = '1' THEN
    IF rstx = '1' THEN
      OUTPUT1_chkcnt <= to_unsigned(16#00#, 6);
    ELSIF OUTPUT1_needToCount = '1' THEN
```

```
        IF OUTPUT1_chkcnt >= to_unsigned(16#20#, 6) THEN
          OUTPUT1_chkcnt <= to_unsigned(16#00#, 6);
        ELSE
          OUTPUT1_chkcnt <= OUTPUT1_chkcnt + to_unsigned(16#01#, 6);
        END IF;
      END IF;
    END IF;
  END PROCESS OUTPUT1_IgnoreDataChecking_process;



  OUTPUT1_chkenb <= '1' WHEN OUTPUT1_chkcnt = to_unsigned(16#20#, 6) ELSE
      '0';

  OUTPUT1_chkdata <= ce_out AND OUTPUT1_chkenb;

  OUTPUT1_addr_delay_1 <= OUTPUT1_addr AFTER 1 ns;

  -- Data source for OUTPUT1_expected
  OUTPUT1_expected_fileread: PROCESS (OUTPUT1_addr_delay_1, tb_enb_delay, ce_out)
    FILE fp: TEXT open READ_MODE is "OUTPUT1_expected.dat";
    VARIABLE l: LINE;
    VARIABLE read_data: std_logic_vector(63 DOWNTO 0);

  BEGIN
    IF tb_enb_delay /= '1' THEN
    ELSIF ce_out = '1' AND NOT ENDFILE(fp) THEN
      READLINE(fp, l);
      HREAD(l, read_data);
    END IF;
    OUTPUT1_expected <= std_logic_vector(read_data(63 DOWNTO 0));
  END PROCESS OUTPUT1_expected_fileread;

  OUTPUT1_ref <= OUTPUT1_expected;

  OUTPUT1_checker: PROCESS (clk, rstx)
  BEGIN
    IF rstx = '1' THEN
      OUTPUT1_testFailure <= '0';
    ELSIF clk'event AND clk = '1' THEN
      IF OUTPUT1_chkdata = '1' AND NOT isFloatDoubleEqual(OUTPUT1, OUTPUT1_ref,
9.9999999999999995e-08) THEN
        OUTPUT1_testFailure <= '1';
        ASSERT FALSE
          REPORT "Error in OUTPUT1: Expected " & to_hex(OUTPUT1_ref) & (" Actual
" & to_hex(OUTPUT1))
          SEVERITY ERROR;
      END IF;
    END IF;
  END PROCESS OUTPUT1_checker;
```

```
  completed_msg: PROCESS (clk)
  BEGIN
    IF clk'event AND clk = '1' THEN
      IF check1_done = '1' THEN
        IF OUTPUT1_testFailure = '0' THEN
          ASSERT FALSE
            REPORT "**************TEST COMPLETED (PASSED)**************"
            SEVERITY NOTE;
        ELSE
          ASSERT FALSE
            REPORT "**************TEST COMPLETED (FAILED)**************"
            SEVERITY NOTE;
        END IF;
      END IF;
    END IF;
  END PROCESS completed_msg;

END rtl;
```

Appendix 5: Package File for Test Bench

```
-- -------------------------------------------------------------
--
-- File Name: hdl_prj\hdlsrc\DSP\ASIL_D_DSP_IP_tb_pac.vhd
-- Created: 2023-04-23 14:36:04
--
-- Generated by MATLAB 9.11 and HDL Coder 3.19
--
-- -------------------------------------------------------------


LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
LIBRARY IEEE;
USE IEEE.std_logic_textio.ALL;
USE IEEE.float_pkg.ALL;
LIBRARY STD;
USE STD.textio.ALL;
LIBRARY work;
USE work.ASIL_D_DSP_IP_pac.ALL;

PACKAGE ASIL_D_DSP_IP_tb_pac IS
  -- Functions
  FUNCTION to_hex(x : IN std_logic) RETURN string;
  FUNCTION to_hex(x : IN std_logic_vector) RETURN string;
  FUNCTION to_hex(x : IN signed) RETURN string;
  FUNCTION to_hex(x : IN unsigned) RETURN string;
  FUNCTION to_hex(x : IN real) RETURN string;
  FUNCTION isFloatEqual(x : IN std_logic_vector; y : IN std_logic_vector;
```

```
                        eps: IN real; exp_len : IN natural; mantissa_len : IN
natural) RETURN boolean;
  FUNCTION isFloatDoubleEqual(x : IN std_logic_vector; y : IN std_logic_vector;
eps: IN real) RETURN boolean;
END ASIL_D_DSP_IP_tb_pac;


PACKAGE BODY ASIL_D_DSP_IP_tb_pac IS
  FUNCTION to_hex(x : IN std_logic_vector) RETURN string IS
    VARIABLE result : STRING(1 TO 256);
    VARIABLE i      : INTEGER;
    VARIABLE imod   : INTEGER;
    VARIABLE j      : INTEGER;
    VARIABLE jinc   : INTEGER;
    VARIABLE newx   : std_logic_vector(1023 DOWNTO 0);
  BEGIN
    newx := (OTHERS => '0');
    IF x'LEFT > x'RIGHT THEN
      j := x'LENGTH - 1;
      jinc := -1;
    ELSE
      j := 0;
      jinc := 1;
    END IF;
    FOR i IN x'RANGE LOOP
      newx(j) := x(i);
      j := j + jinc;
    END LOOP;
    i := x'LENGTH - 1;
    imod := x'LENGTH MOD 4;
    IF    imod = 1 THEN i := i + 3;
    ELSIF imod = 2 THEN i := i + 2;
    ELSIF imod = 3 THEN i := i + 1;
    END IF;
    j := 1;
    WHILE i >= 3 LOOP
      IF    newx(i DOWNTO (i-3)) = "0000" THEN result(j) := '0';
      ELSIF newx(i DOWNTO (i-3)) = "0001" THEN result(j) := '1';
      ELSIF newx(i DOWNTO (i-3)) = "0010" THEN result(j) := '2';
      ELSIF newx(i DOWNTO (i-3)) = "0011" THEN result(j) := '3';
      ELSIF newx(i DOWNTO (i-3)) = "0100" THEN result(j) := '4';
      ELSIF newx(i DOWNTO (i-3)) = "0101" THEN result(j) := '5';
      ELSIF newx(i DOWNTO (i-3)) = "0110" THEN result(j) := '6';
      ELSIF newx(i DOWNTO (i-3)) = "0111" THEN result(j) := '7';
      ELSIF newx(i DOWNTO (i-3)) = "1000" THEN result(j) := '8';
      ELSIF newx(i DOWNTO (i-3)) = "1001" THEN result(j) := '9';
      ELSIF newx(i DOWNTO (i-3)) = "1010" THEN result(j) := 'A';
      ELSIF newx(i DOWNTO (i-3)) = "1011" THEN result(j) := 'B';
      ELSIF newx(i DOWNTO (i-3)) = "1100" THEN result(j) := 'C';
      ELSIF newx(i DOWNTO (i-3)) = "1101" THEN result(j) := 'D';
      ELSIF newx(i DOWNTO (i-3)) = "1110" THEN result(j) := 'E';
```

```
      ELSIF newx(i DOWNTO (i-3)) = "1111" THEN result(j) := 'F';
      ELSE result(j) := 'X';
      END IF;
      i := i - 4;
      j := j + 1;
    END LOOP;
    RETURN result(1 TO j - 1);
  END;

  FUNCTION to_hex(x : IN std_logic) RETURN string IS
  BEGIN
    RETURN std_logic'image(x);
  END;

  FUNCTION to_hex(x : IN signed) RETURN string IS
  BEGIN
    RETURN to_hex(std_logic_vector(x));
  END;

  FUNCTION to_hex(x : IN unsigned) RETURN string IS
  BEGIN
    RETURN to_hex(std_logic_vector(x));
  END;

  FUNCTION to_hex(x : IN real) RETURN string IS
  BEGIN
    RETURN real'image(x);
  END;

  FUNCTION isFloatEqual(x : IN std_logic_vector; y : IN std_logic_vector;
                        eps : IN real; exp_len : IN natural;
                        mantissa_len : IN natural) RETURN boolean IS
    VARIABLE absdiff : real;
    VARIABLE a : real;
    VARIABLE b : real;
  BEGIN
    a := to_real(to_float(std_ulogic_vector(x), exp_len, mantissa_len));
    b := to_real(to_float(std_ulogic_vector(y), exp_len, mantissa_len));
    absdiff := abs(a - b);
    IF absdiff < eps THEN -- absolute error check
      RETURN TRUE;
    ELSIF a = b THEN -- check infinities
      RETURN TRUE;
    ELSIF a*b = 0.0 THEN -- either is zero
      RETURN absdiff < eps;
    ELSIF (abs(a) < abs(b)) THEN -- relative error check
      RETURN absdiff/abs(b) < eps;
    ELSE
      RETURN absdiff/abs(a) < eps;
    END IF;
  END;
```

```
  FUNCTION isFloatDoubleEqual(x : IN std_logic_vector;
                              y : IN std_logic_vector;
                              eps : IN real) RETURN boolean IS
  VARIABLE a : std_logic_vector(63 downto 0);
  VARIABLE b : std_logic_vector(63 downto 0);
  VARIABLE zrEx : std_logic_vector(10 downto 0) := b"111" & x"FF";
  VARIABLE zrMt : std_logic_vector(51 downto 0) := x"0000000000000";
  BEGIN
    a := x;
    b := y;
    IF (a(62 downto 52) = zrEx AND a(51 downto 0) /= zrMt) THEN
      a(63) := '0';
      a(51 downto 0) := x"0000000000001";
    END IF;
    IF (b(62 downto 52) = zrEx AND b(51 downto 0) /= zrMt) THEN
      b(63) := '0';
      b(51 downto 0) := x"0000000000001";
    END IF;
      RETURN isFloatEqual(a, b, eps, 11, 52);
    END;

END ASIL_D_DSP_IP_tb_pac;
```

Appendix 6: Log File for FPGA Synthesis

```
Task "Create Project" successful.
Generated logfile:

****** Vivado v2022.1 (64-bit)
  **** SW Build 3526262 on Mon Apr 18 15:48:16 MDT 2022
  **** IP Build 3524634 on Mon Apr 18 20:55:01 MDT 2022
    ** Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.

source ASIL_D_DSP_IP_Xilinx_Vivado_run.tcl -notrace
### Create new Xilinx Vivado 2022.1 project <a
href="matlab:downstream.handle('Model','DSP').openTargetTool;">hdl_prj\vivado_prj
\ASIL_D_DSP_IP_vivado.xpr</a>
create_project: Time (s): cpu = 00:00:02 ; elapsed = 00:00:09 . Memory (MB): peak
= 1637.656 ; gain = 0.000
### Set Xilinx Vivado 2022.1 project properties
### Update Xilinx Vivado 2022.1 project with HDL source files
WARNING: [Vivado 12-818] No files matched '*.tcl'
### Close Xilinx Vivado 2022.1 project.
INFO: [Common 17-206] Exiting Vivado at Sun Apr 23 07:16:41 2023...

Elapsed time is 98.6648 seconds.
```

Appendix 7: Passed Synthesis Report

```
Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
------------------------------------------------------------------------------------
------------------------------------
| Tool Version : Vivado v.2022.1 (win64) Build 3526262 Mon Apr 18 15:48:16 MDT
2022
| Date         : Sun Apr 23 07:21:20 2023
| Host         : LAPTOP-3H9IEFKL running 64-bit major release  (build 9200)
| Command      : report_utilization -file ASIL_D_DSP_IP_utilization_synth.rpt -pb
ASIL_D_DSP_IP_utilization_synth.pb
| Design       : ASIL_D_DSP_IP
| Device       : xa7a100tcsg324-1I
| Speed File   : -1I
| Design State : Synthesized
------------------------------------------------------------------------------------
------------------------------------
Utilization Design Information

Table of Contents
-----------------
1. Slice Logic
1.1 Summary of Registers by Type
2. Memory
3. DSP
4. IO and GT Specific
5. Clocking
6. Specific Feature
7. Primitives
8. Black Boxes
9. Instantiated Netlists

1. Slice Logic
--------------
+----------------------------+------+-------+------------+-----------+-------+
|          Site Type         | Used | Fixed | Prohibited | Available | Util% |
+----------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*                | 5030 |     0 |          0 |     63400 |  7.93 |
|   LUT as Logic             | 4486 |     0 |          0 |     63400 |  7.08 |
|   LUT as Memory            |  544 |     0 |          0 |     19000 |  2.86 |
|     LUT as Distributed RAM |    0 |     0 |            |           |       |
|     LUT as Shift Register  |  544 |     0 |            |           |       |
| Slice Registers            | 7409 |     0 |          0 |    126800 |  5.84 |
|   Register as Flip Flop    | 7409 |     0 |          0 |    126800 |  5.84 |
|   Register as Latch        |    0 |     0 |          0 |    126800 |  0.00 |
| F7 Muxes                   |    2 |     0 |          0 |     31700 | <0.01 |
| F8 Muxes                   |    0 |     0 |          0 |     15850 |  0.00 |
+----------------------------+------+-------+------------+-----------+-------+
* Warning! The Final LUT count, after physical optimizations and full implementation, is
typically lower. Run opt_design after synthesis, if not already completed, for a more
realistic count.
```

1.1 Summary of Registers by Type

Total	Clock Enable	Synchronous	Asynchronous
0	_	-	-
0	_	-	Set
0	_	-	Reset
0	_	Set	-
0	_	Reset	-
0	Yes	-	-
0	Yes	-	Set
0	Yes	-	Reset
0	Yes	Set	-
7409	Yes	Reset	-

2. Memory

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	0	0	0	135	0.00
RAMB36/FIFO*	0	0	0	135	0.00
RAMB18	0	0	0	270	0.00

* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one FIFO18E1. However, if a FIFO18E1 occupies a Block RAM Tile, that tile can still accommodate a RAMB18E1

3. DSP

Site Type	Used	Fixed	Prohibited	Available	Util%
DSPs	54	0	0	240	22.50
DSP48E1 only	54				

4. IO and GT Specific

Site Type	Used	Fixed	Prohibited	Available	Util%
Bonded IOB	0	0	0	210	0.00
Bonded IPADs	0	0	0	2	0.00
PHY_CONTROL	0	0	0	6	0.00
PHASER_REF	0	0	0	6	0.00
OUT_FIFO	0	0	0	24	0.00
IN_FIFO	0	0	0	24	0.00
IDELAYCTRL	0	0	0	6	0.00
IBUFDS	0	0	0	202	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	0	24	0.00
PHASER_IN/PHASER_IN_PHY	0	0	0	24	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	0	300	0.00
ILOGIC	0	0	0	210	0.00
OLOGIC	0	0	0	210	0.00

5. Clocking

Site Type	Used	Fixed	Prohibited	Available	Util%
BUFGCTRL	0	0	0	32	0.00
BUFIO	0	0	0	24	0.00
MMCME2_ADV	0	0	0	6	0.00
PLLE2_ADV	0	0	0	6	0.00
BUFMRCE	0	0	0	12	0.00
BUFHCE	0	0	0	96	0.00
BUFR	0	0	0	24	0.00

6. Specific Feature

Site Type	Used	Fixed	Prohibited	Available	Util%
BSCANE2	0	0	0	4	0.00
CAPTUREE2	0	0	0	1	0.00
DNA_PORT	0	0	0	1	0.00
EFUSE_USR	0	0	0	1	0.00
FRAME_ECCE2	0	0	0	1	0.00
ICAPE2	0	0	0	2	0.00
PCIE_2_1	0	0	0	1	0.00
STARTUPE2	0	0	0	1	0.00
XADC	0	0	0	1	0.00

```
7. Primitives
-------------

+----------+------+---------------------+
| Ref Name | Used | Functional Category |
+----------+------+---------------------+
| FDRE     | 7409 |        Flop & Latch |
| LUT2     | 1519 |                 LUT |
| LUT6     | 1510 |                 LUT |
| LUT3     |  831 |                 LUT |
| LUT5     |  763 |                 LUT |
| LUT4     |  605 |                 LUT |
| SRL16E   |  481 |  Distributed Memory |
| CARRY4   |  442 |          CarryLogic |
| LUT1     |  161 |                 LUT |
| SRLC32E  |   63 |  Distributed Memory |
| DSP48E1  |   54 |    Block Arithmetic |
| MUXF7    |    2 |               MuxFx |
+----------+------+---------------------+


8. Black Boxes
--------------

+----------+------+
| Ref Name | Used |
+----------+------+


9. Instantiated Netlists
------------------------

+----------+------+
| Ref Name | Used |
+----------+------+
```

Appendix 8: Resource & Timing Summary

Timing summary	
	Value
Requirement	10000 ns
Data Path Delay	5.394 ns
Slack	9994.5 ns
Clock Frequency	181.09 MHz