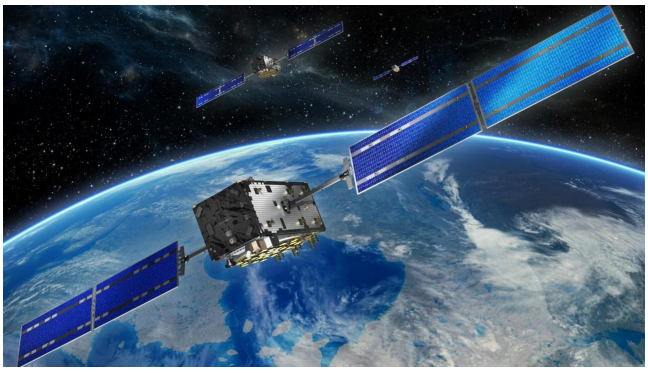
Final Project: Enhancing the Reliability of an FPGA-based RISC-V Embedded Processor

Problem Statement

Space-G is developing a telecommunications payload for the International Space Station (ISS). The payload is based on a Zynq-7000 SoC, integrating a NEORV32 (RISC-V) soft-core processor for executing matrix multiplication tasks used in telecommunications (e.g., FFT). In LEO, soft errors due to radiation-induced bit flips are a significant concern. Therefore, to improve the dependability of the NEORV32, two soft-error mitigation techniques—Triple Modular Redundancy (TMR) and Error Correction Codes (ECC)—must be applied selectively. You are tasked with implementing these SEE mitigation schemes in the `neorv32_cpu_alu.vhd` file and evaluating the system's post-routing metrics and reliability for a 15-year LEO mission.



Milestone 1: SEE Mitigation

Task 1: Baseline (unhardened) NEORV32 Design Metrics

Objective: Gather the baseline design metrics of the unmodified NEORV32.

Metrics to Obtain:

1. Worst Negative Slack (WNS) and Maximum Frequency
2. Resource Utilisation:
   - Total LUTs, Logic LUTs, LUTRAMs, SRLs, FFs, RAMB36, RAMB18, DSP48 Blocks for both:
     - The entire NEORV32 design (top module)
     - The `neorv32_cpu_alu` module
3. Power Consumption:
   - Total, Dynamic, and Static Power of the whole design

Hints:

Open Vivado GUI. Run Implementation → Open Implemented Design.

- To get utilization: Tools → Report → Report Utilization
- To get WNS: Tools → Timing → Report Timing Summary
- To get power metrics: Tools → Report → Power Report

Task 2: Selective TMR of NEORV32 ALU

Objective: Apply Triple Modular Redundancy (TMR) selectively to enhance the reliability of specific NEORV32 ALU components.

Steps: Modify the `neorv32_cpu_alu.vhd` file to apply TMR to:

1. Line 104: Apply TMR to the comparator unit (responsible for conditional branches).
2. Line 134: Apply TMR to the adder core.
3. Lines 197-225: Apply TMR to the Co-processor 1 (integer multiplication/division unit - 'M' Extension).

Task 3: Implement Hamming ECC on Execute Engine FSM

Objective: Improve the reliability of the NEORV32 execution unit by adding a Hamming Error Correction Code (ECC) in the Finite State Machine (FSM) of the Execute Engine.

The Execute Engine FSM is implemented by two processes:

- (a) `execute_engine_fsm_sync`
- (b) `execute_engine_fsm_comb`

In process (a), the signal `execute_engine.state` (current FSM state) is assigned the signal `execute_engine.state_nxt` (next FSM state), while in process (b), the signal `execute_engine.state_nxt` is calculated according to the current FSM state and the current instruction. The type of both signals is `execute_engine_state_t`; the type has 12 values (states), and thus the signals are 4-bit vectors. The Hamming ECC will be a (7,4) Hamming Single Error Correction Single Error Detection (SECSED) code, i.e., 4 bits for the FSM state code and 3 bits for the parities. See the presentations: (a) Lecture: Error Detection Codes and (b) Lab Exercise: Safe FSM Encoding.

Hints:

- The parity bits will be assigned values along with the `execute_engine.state` in the process `execute_engine_fsm_sync` (line 697) and will be calculated based on the `execute_engine.state_nxt`.
- For calculating the parity bits and the syndromes, use the equations from the (7,4) Hamming SECSED code presented in the lectures.
- Calculate the `state_corrected` signal from the `execute_engine.state` and the syndromes.
- Decide whether the parity bits, the syndrome, and the `state_corrected` signal will be defined as part of the `execute_engine_t` record, or they will be standalone signals.
- Replace the `execute_engine.state` signal with the `state_corrected` signal (i.e., the original current FSM state with the corrected FSM state) in all the points of the VHDL code where the `execute_engine.state` signal is read (checked).

Task 4: unhardened and hardened

Maximum frequency and initialisation Metrics:

Compare the following design metrics between the unhardened and hardened versions of NEORV32:

1. Maximum Frequency
2. Resource Utilisation:
   - Whole design
   - Only the ALU module (use `report_utilization -hierarchical)`
3. Power Consumption: Total, Dynamic, and Static Power

Milestone 2: Reliablity Analysis

Task 1: Fault Injection Experiments

Use FREtZ to conduct fault injection experiments on the `Co-Processor 1 - Integer Multiplication/Division Unit` when running a matrix multiplication application.

After power on, the FPGA should start the application which enters in an infinite loop, waiting for input from the UART interface. When the character 'X' is received, it starts the matrix multiplication process between two 28x28 matrices. The matrices mat1 and mat2 are initialized such that each row is filled with the row index + 1. After initialisation, the programme multiplies the two matrices and outputs the result via UART, printing each element of the result matrix using neorv32_uart0_printf().

Use the software code from the CDS206_Final_Project.zip file from Eclass(`https://thales.cs.unipi.gr/modules/document/file.php/CDS126/Project/CDS206_Final_Project_Files.zip`).

Report

1. # of Silent Data Corruptions (SDCs), i.e., if the result of mult is not correct.
2. # of Processor Crashes, i.e., the processor becomes unresponsive (crashes).
3. Architectural Vulnerability Factor (AVF):

$$AVF = \frac{SDC + Crashes}{\text{Total Faults Injected}}$$

Task 2: Calculate Reliability for a One-Year LEO Mission

Objective: Estimate the reliability of the NEORV32 ALU for a 15-year mission.

$R(t = 15 \text{ year}) = e^{-\lambda t}$,

where λ = AVF $\times$ CRAM-upset$_{\text{rate}}$. Assume an CRAM-upset$_{\text{rate}}$ = 4.48×10^{-6} per device per day.

Requirements

- Vivado version: 2016.4
- Target board: Zybo Z7-10 or Zybo.

- Vivado board support files can be installed using `https://digilent.com/reference/software/vivado/board-files?redirect=1`.

Source code

- `Download the CDS206_Final_Project.zip` from Eclass `https://thales.cs.unipi.gr/modules/document/file.php/CDS126/Project/CDS206_Final_Project_Files.zip`.
- Inside the zip file you will find:
  - A `UserApplication.py` file for the FREtZ framework. This `UserApplication.py` can be used to inject faults into the mul/div component of NEORV32. Copy this file into your FREtZ project's UI folder (example:`/home/fretz/wsp/sysyfos-fretz-host-sw/src/UI`)
  - Software code for a matrix multiplication application (`NEORV32_SW/main.c`). Ideally, you should configure RISC-V to execute this software after the SoC-FPGA is configured (i.e. do not use the UART bootloader).
  - A Vivado placement constraints file (`NEORV32_SW/Zybo-Z7-Master.xdc`) in order to implement the mul/div into a specific area of the SoC-FPGA (i.e. into a Pblock)
  - The `frames_app.txt` contains the configuration frame addresses of the mul/div Pblock. Move this file into the FREtZ project folder.
- General Hints:
  - To perform fault injections you need to copy the `.bit, .ll, .ebd, .ebc, .msk, frames_app.txt` files into you FREtZ folder.
  - After executing the fault injection, you can find the results in file called results.txt into you FREtZ folder. Each mmult result from each fault injection iteration (inject a fault, run mmult, save the result in results.txt file) should be compared with a *golden* reference results (i.e. a results that you captured by running mmult but without inserting a fault). Each mismatch between the golden result and the captured result

Deliverables

1. Final Report:
   - Unhardened and hardened design metrics.
   - Fault injection results and calculated AVF.
   - Reliability analysis of the 15-year LEO mission.

2. Project Files:
    - Vivado: Unhardened and hardened NEORV32 designs.
    - The results.txt file.

Deadline and Progress Meetings

You should deliver this project on **03/11/2024**.

Progress Meetings

We will have progress meetings on the following dates:

- 14/10/2024
- 21/10/2024
- 28/10/2024

All meetings will take place from **18:30 - 19:30** via Google Meet: `https://meet.google.com/run-jxfs-bna`

During our meetings, we will solve any problems that you may encounter and answer any questions you have about this project.

Final Examination

On **04/11/2024**, you will be examined face-to-face at **Themistokleio, Nikaia**.