

**EEDG/CE 6302**

**Microprocessors and Embedded Systems**

**Electrical and Computer Engineering**

**Erik Jonsson School of Engineering and Computer Science**

**The University of Texas at Dallas**

**Dr. Tooraj Nikoubin**

**Project 1 (Part 4): Data Hazard Stall and Branch Detection**

**Submitted By:**

**Muripa Uppaluri (MXU220008)**

**Chandanam Sai Nived (SXC210186)**

## Table of Contents

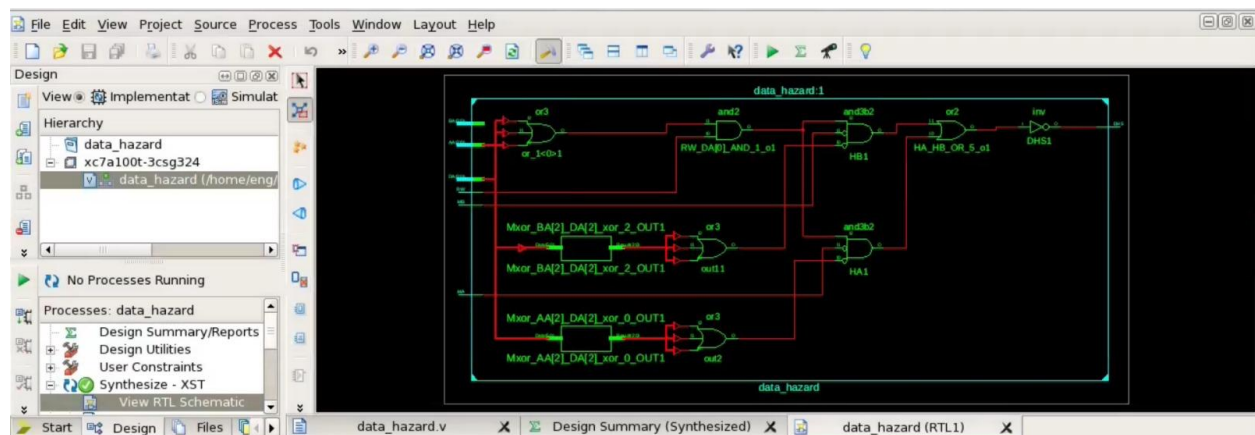
| <b>Content</b>                           | <b>Page Number</b> |
|--|--------------------|
| <b>Data Hazard Stall and Schematic</b>   | <b>3</b>           |
| <b>Schematic View and Waveform</b>       | <b>4</b>           |
| <b>Design Summary Report</b>             | <b>5</b>           |
| <b>Branch Detect</b>                     | <b>5</b>           |
| <b>Schematic View and Waveforms</b>      | <b>6</b>           |
| <b>Lab Handout Questions and Answers</b> | <b>7</b>           |

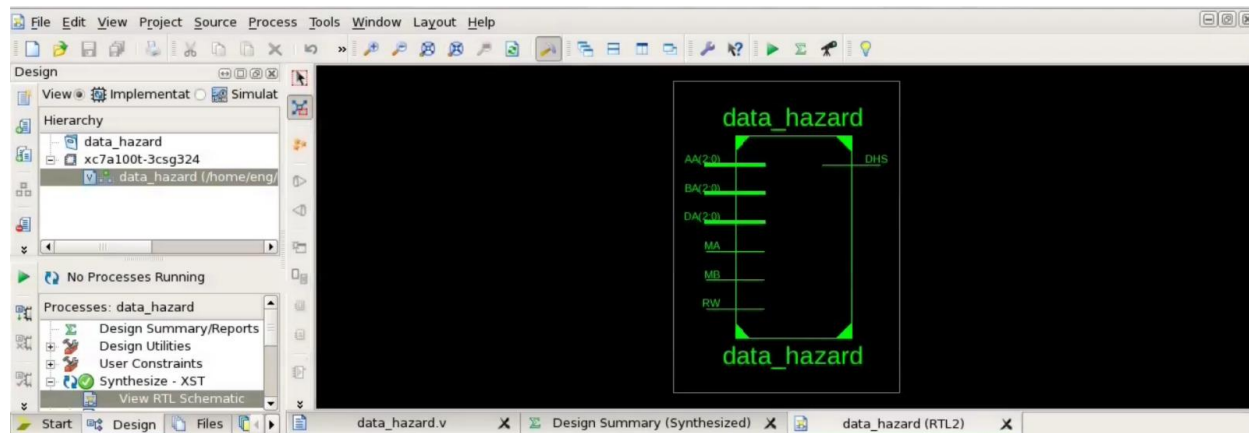
In this fourth part of the first lab project, we've designed a Data Hazard Stall (DHS) and Branch Detection(Br\_detect). We will be explaining about the implementation of each in the following sections.

### **Data Hazard Stall (DHS):**

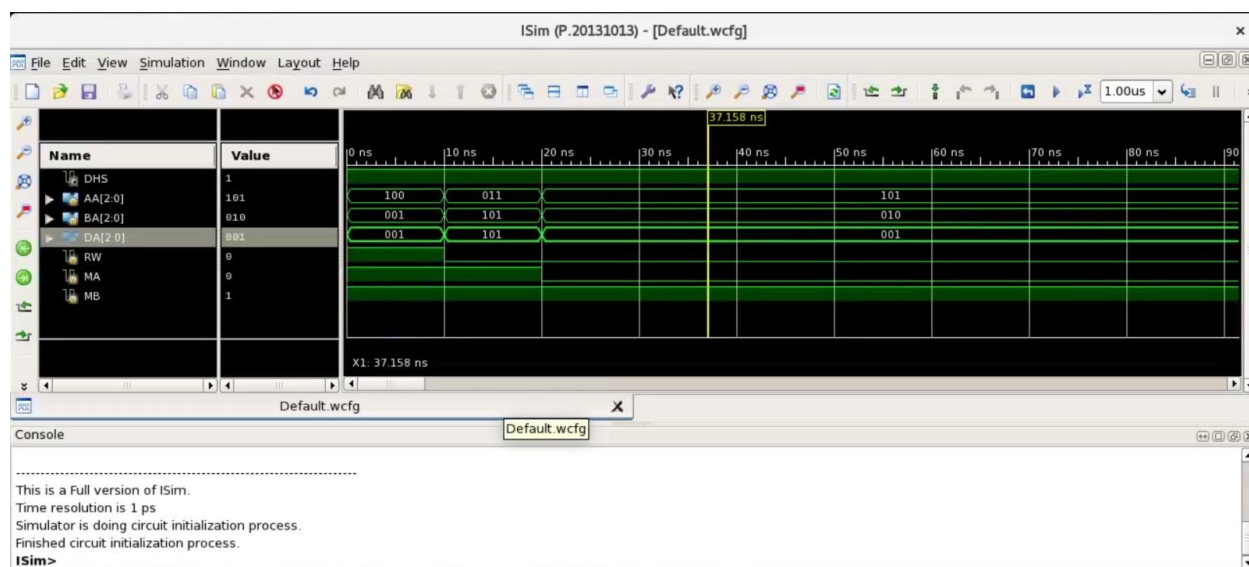
A Data Hazard Stall is a technique used to avoid data hazards in a processor pipeline. When a data hazard occurs, the pipeline must wait until the required data is available before it can proceed with execution. In a data hazard stall, the pipeline is stalled, or halted, until the required data is available. This is achieved by inserting bubbles, or NOP (no-operation) instructions, into the pipeline to delay execution until the required data is ready. This technique can help ensure that the processor produces correct results, but it may also reduce performance by increasing the overall execution time.

### **Schematic view of the Data Hazard Stall (DHS)**



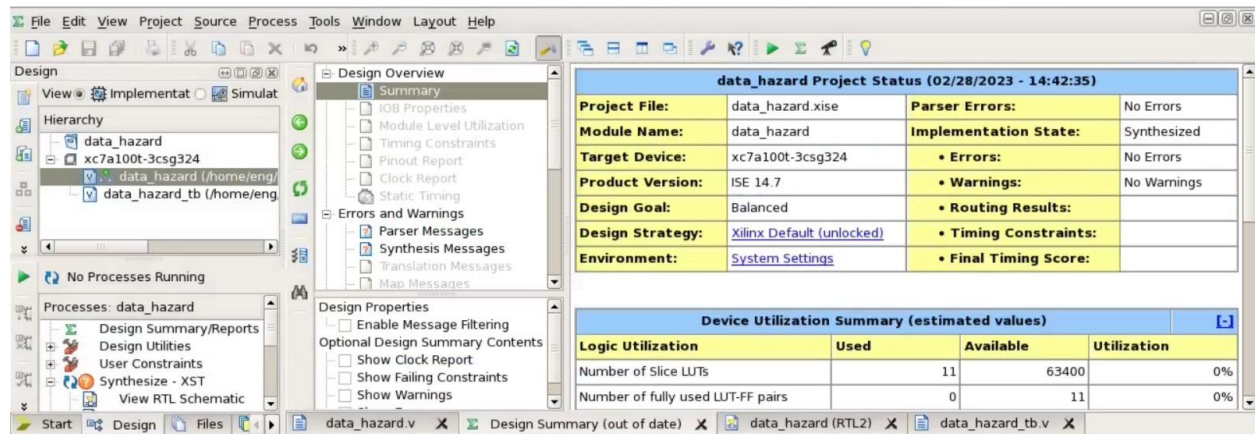


## Data Hazard Stall Waveform –



## Design Summary Report:

Since this is a pure combinational design, we don't have any Latches.



The screenshot displays the Xilinx ISE Design Suite interface. The main window shows the 'Design Summary' report for the project 'data\_hazard'. The report is titled 'data\_hazard Project Status (02/28/2023 - 14:42:35)'. It includes the following information:

- Project File:** data\_hazard.xise
- Module Name:** data\_hazard
- Target Device:** xc7a100t-3csg324
- Product Version:** ISE 14.7
- Design Goal:** Balanced
- Design Strategy:** Xilinx Default (unlocked)
- Environment:** System Settings
- Parser Errors:** No Errors
- Implementation State:** Synthesized
- Errors:** No Errors
- Warnings:** No Warnings
- Routing Results:**
- Timing Constraints:**
- Final Timing Score:**

Below the project status, there is a section for 'Device Utilization Summary (estimated values)'. It includes a table with the following data:

| Logic Utilization                 | Used | Available | Utilization |
|-----------------------------------|------|-----------|-------------|
| Number of Slice LUTs              | 11   | 63400     | 0%          |
| Number of fully used LUT-FF pairs | 0    | 11        | 0%          |

The interface also shows a 'Design Overview' pane on the left with a tree view of the design hierarchy, including 'data\_hazard' and 'data\_hazard\_tb'. The bottom status bar indicates that the design is 'out of date'.

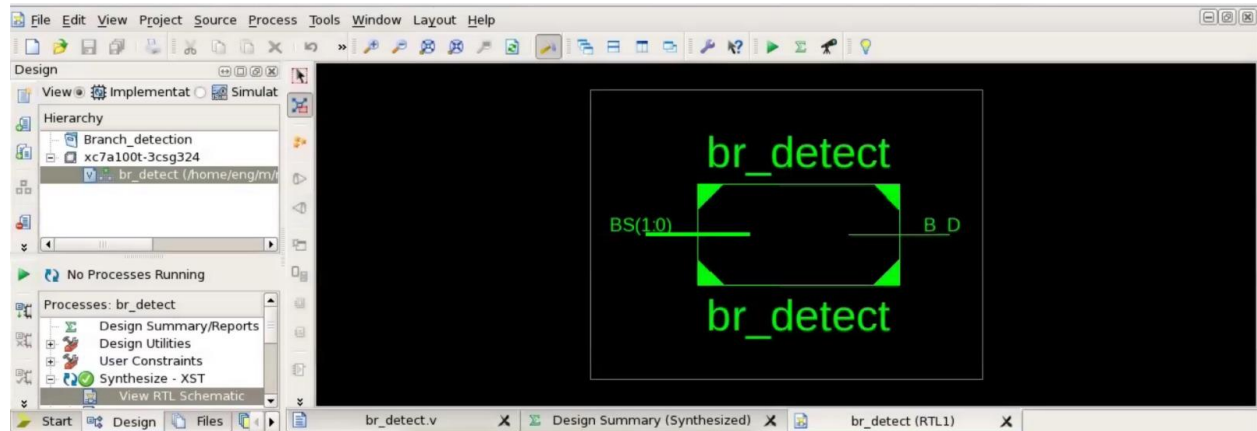
## Branch Detect Implementation:

During normal operation, when branches are not taken, instructions will be fetched and decoded, and then operands will be fetched based on the addition of 1 to the value of the PC. If the branch is taken, the instructions following the branch instruction need to be canceled. The cancellation is done by inserting bubbles into the execution and write-back stages for these instructions. These bubbles have the same effect as two NOP instructions in the pipeline.

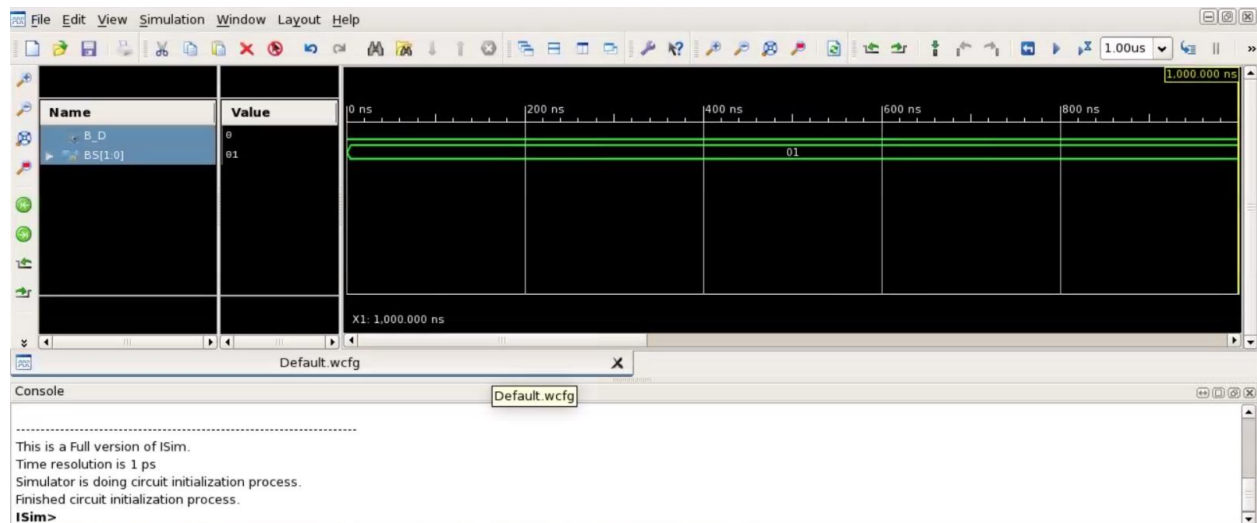
Br\_Detect requires a simpler logic and an adder. When a branch is taken, it flushes the value of IR by writing all zeros into this register and prevents running the previously fetched instruction. The MUXC will select the next correct value for the PC.

The MUXC has two selectors: S[1] and S[2]. S[1] is directly connected to BS[1] and S[0] requires extra logic to be created. S[1] and S[0] are ORed to create Br\_Detect which is inverted to produce Br\_Detect

## Schematic View of Br detect –



## Br Detect waveform –



## **Lab Handout Questions**

1. Active high and active low refer to the two possible states of a control signal. Active high control signals are signals that are considered "on" or "active" when their voltage level is high (logical 1). Active low control signals are signals that are considered "on" or "active" when their voltage level is low (logical 0).
2. Positive-edge-triggered control signals are activated on the rising edge of a clock signal, while negative-edge-triggered control signals are activated on the falling edge of a clock signal.
3. A latch is a type of sequential logic circuit that stores and outputs data based on its input signal. It has two stable states and can be built using a combination of logic gates. A flip-flop, on the other hand, is a type of latch that is clocked and can change state only on a clock edge.
4. Registers need reset signal to initialize their contents to a known state when the system starts up or when a reset event occurs. Reset is usually applied to microprocessors during power-up, or glitch in the power supply, or when a user-initiated reset is performed.
5. Yes, it is possible to have a "preset" signal instead of a reset signal in a microcontroller unit (MCU). The preset signal sets all the flip-flops or latches in the module to a known state, typically all ones. However, the reset signal is more commonly used in MCUs since it allows the system to start from a known state after power-on or when a critical error occurs.
6. Without any hazard detection and resolution techniques in place, running the given code may result in an incorrect result due to data hazards. The second and third instructions both depend on the result of the first instruction. However, in the pipeline, the instructions are being executed concurrently, which means the result of the first instruction may not be available by the time the second and third instructions execute, leading to incorrect results.
7. Data forwarding is a hardware technique used to resolve data hazards in a pipelined processor. It allows the result of an instruction in the execution stage to be forwarded directly to a subsequent instruction in the pipeline, bypassing the need to write the result to the register file and then read it back again. This reduces the number of pipeline stalls and improves performance. Data forwarding can be implemented using dedicated hardware circuits that detect and forward data as needed.