# Phase 3 Project Report: Single-Cycle and Pipelined MIPS Processor

**Authors:** Matthew Collins & Lewis Bates  
**Emails:** mcollins42@tntech.edu & lfbates42@tntech.edu  
**Course:** ECE 4120/5120  
**Date:** April 28, 2025

## Table of Contents

1. [Introduction](#introduction)
2. [Part 1: Single-Cycle Implementation](#part-1-single-cycle-implementation)
   * [Modified Block Diagram](#part-1-modified-block-diagram)
   * [Elaboration on VHDL Implementation](#Xfcc513a2fef87298323476aea56d597716a89ec)
   * [Flow Summary, RTL View, and Technology Map View](#X60dbec1dcf52db42ede4699a5bf83690a7a0766)
   * [Testbench Elaboration](#part-1-testbench-elaboration)
   * [Waveform Elaboration](#part-1-waveform-elaboration)
3. [Part 2: Pipelined Implementation](#part-2-pipelined-implementation)
   * [Modified Block Diagram](#part-2-modified-block-diagram)
   * [Elaboration on VHDL Implementation](#Xc5b0c11df00b57133b0782dbd9cc5c2e8801fcb)
   * [Flow Summary, RTL View, and Technology Map View](#Xc745d5ca86b6e0c59cd3d29ddc2f2ae9610e628)
   * [Testbench Elaboration](#part-2-testbench-elaboration)
   * [Waveform Elaboration](#part-2-waveform-elaboration)
   * [Improvements with Pipelining](#part-2-improvements-with-pipelining)
   * [Hardware Overhead](#part-2-hardware-overhead)
4. [Conclusion](#conclusion)

## Introduction

This report documents the design, implementation, and testing of a MIPS processor for Phase 3 of the ECE 4120/5120 project. The project is divided into three parts:

* **Part 1**: A single-cycle processor implementation.
* **Part 2**: A pipelined processor implementation without forwarding or hazard units.
* **Part 3 (Bonus)**: An enhanced pipelined processor with forwarding and hazard units.

Each part includes modified block diagrams, VHDL implementation details, synthesis views, testbench descriptions, and waveform analyses, as required by the project deliverables. Additionally, pipelining improvements, hardware overhead, and bonus-related performance penalties are analyzed.

## Part 1: Single-Cycle Implementation

### Part 1: Modified Block Diagram

### Part 1: Top-Level Entity

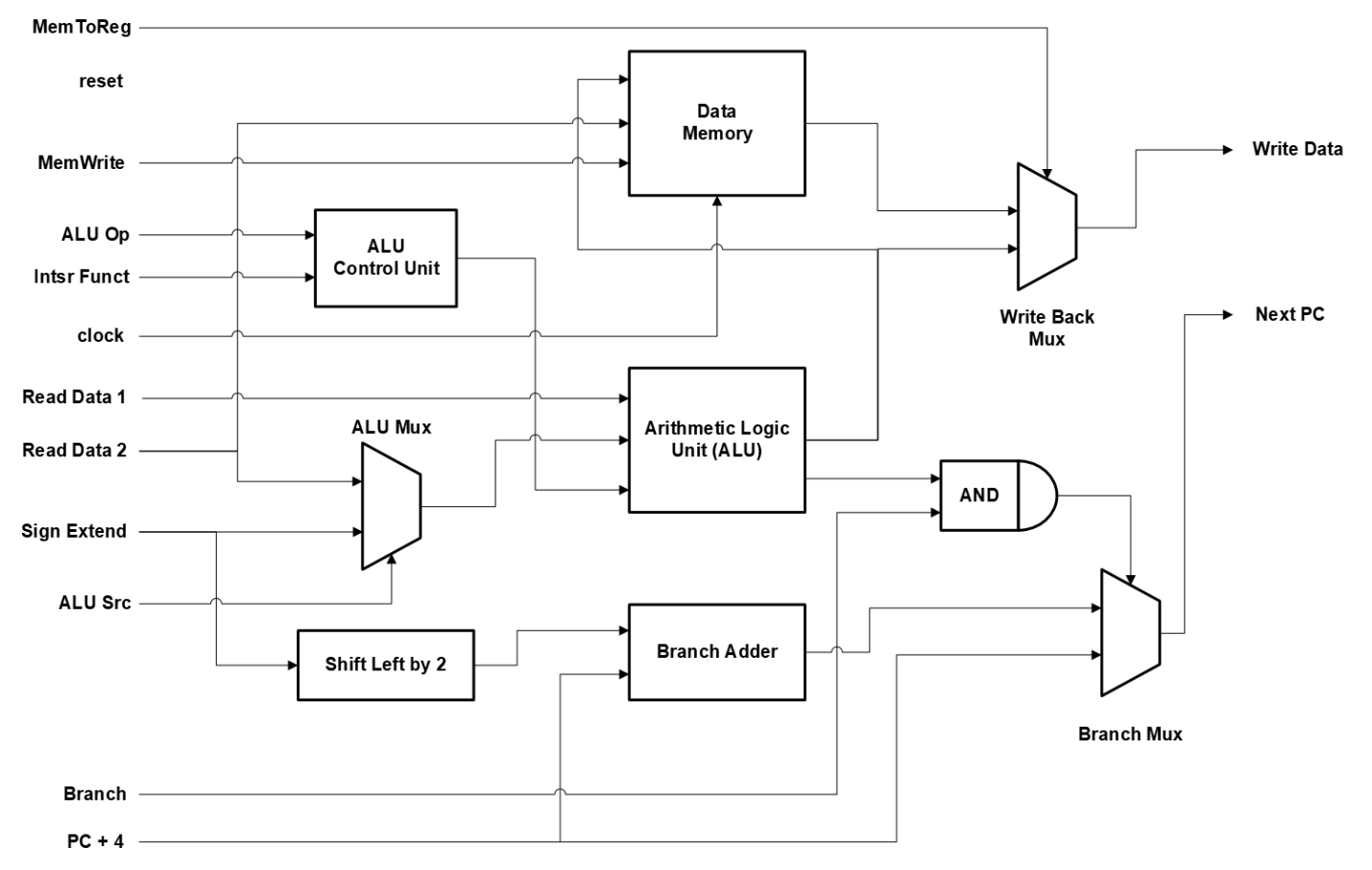
### A diagram of a computer AI-generated content may be incorrect.

### Part 1: Instruction Fetch and Instruction Decode

A diagram of a computer program

AI-generated content may be incorrect.

### Part 1: Execute, Memory, and Write Back Sections



The single-cycle processor integrates all components to execute an instruction in one clock cycle. Key features include:

* **Clock Signals**: A single clock source drives the PC, instruction memory, register file, and data memory.
* **Data Widths**: 32-bit data paths for PC, registers, ALU, and memory; 5-bit register addresses; 8-bit memory addresses.
* **Control Signals**: RegDst, Branch, RegWrite, ALUSrc, MemRead, MemWrite, MemtoReg, and ALUOp (2-bit) from the control unit.

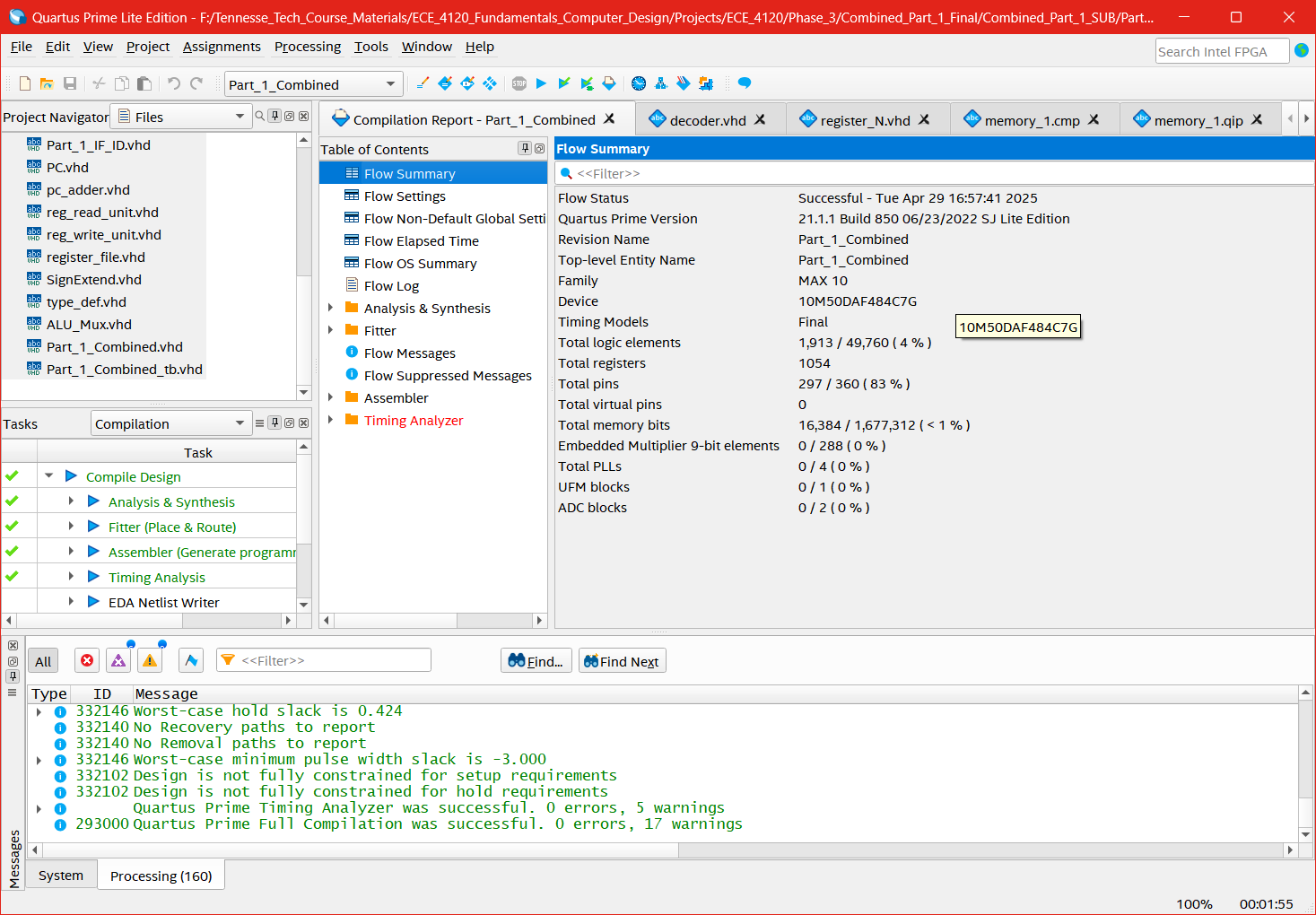
Components include the PC, instruction memory, register file, control unit, ALU, data memory, sign-extend unit, and multiplexers (Mux1, Mux2, Mux3, Mux4).

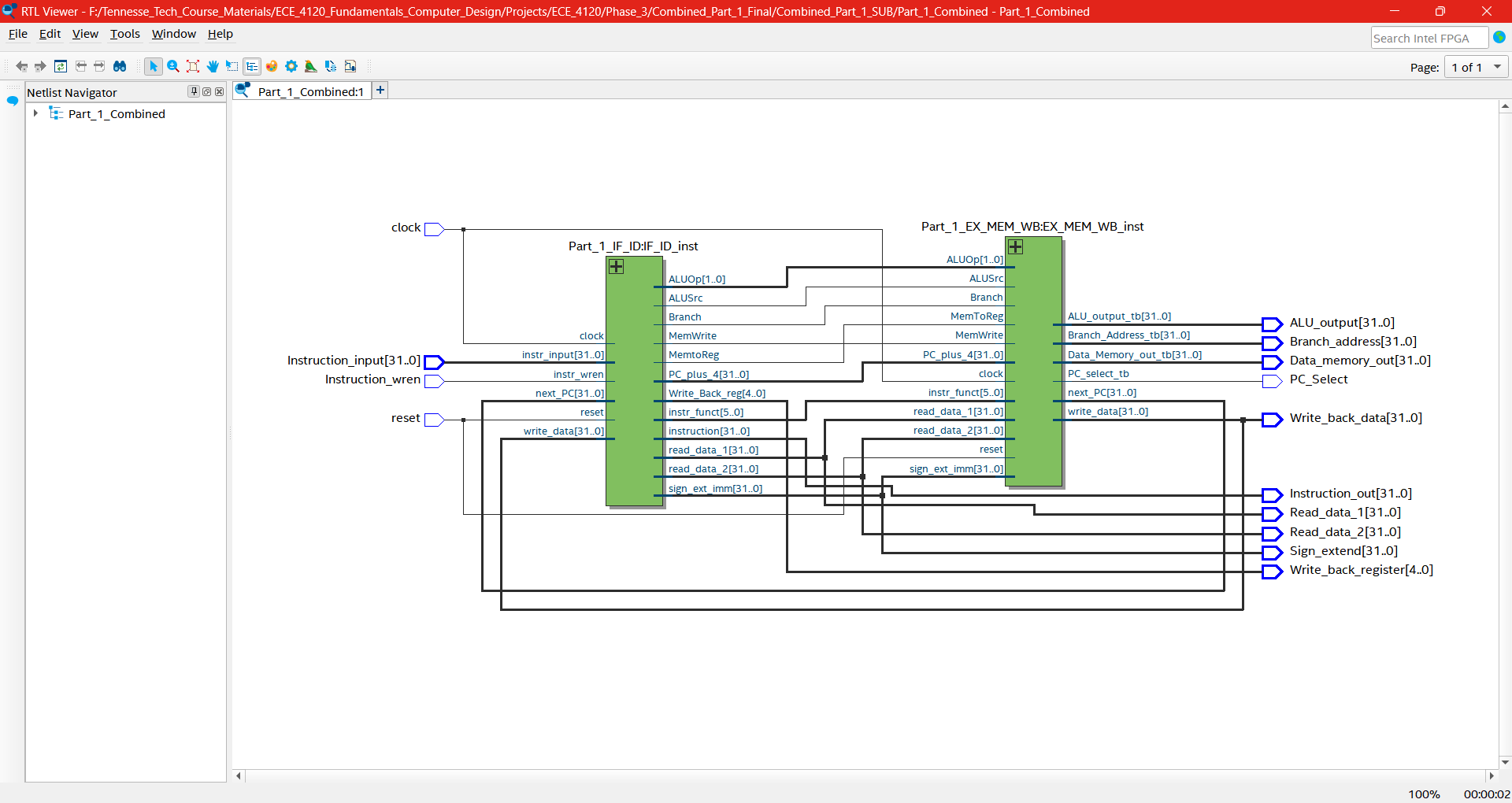
### Part 1: Elaboration on VHDL Implementation

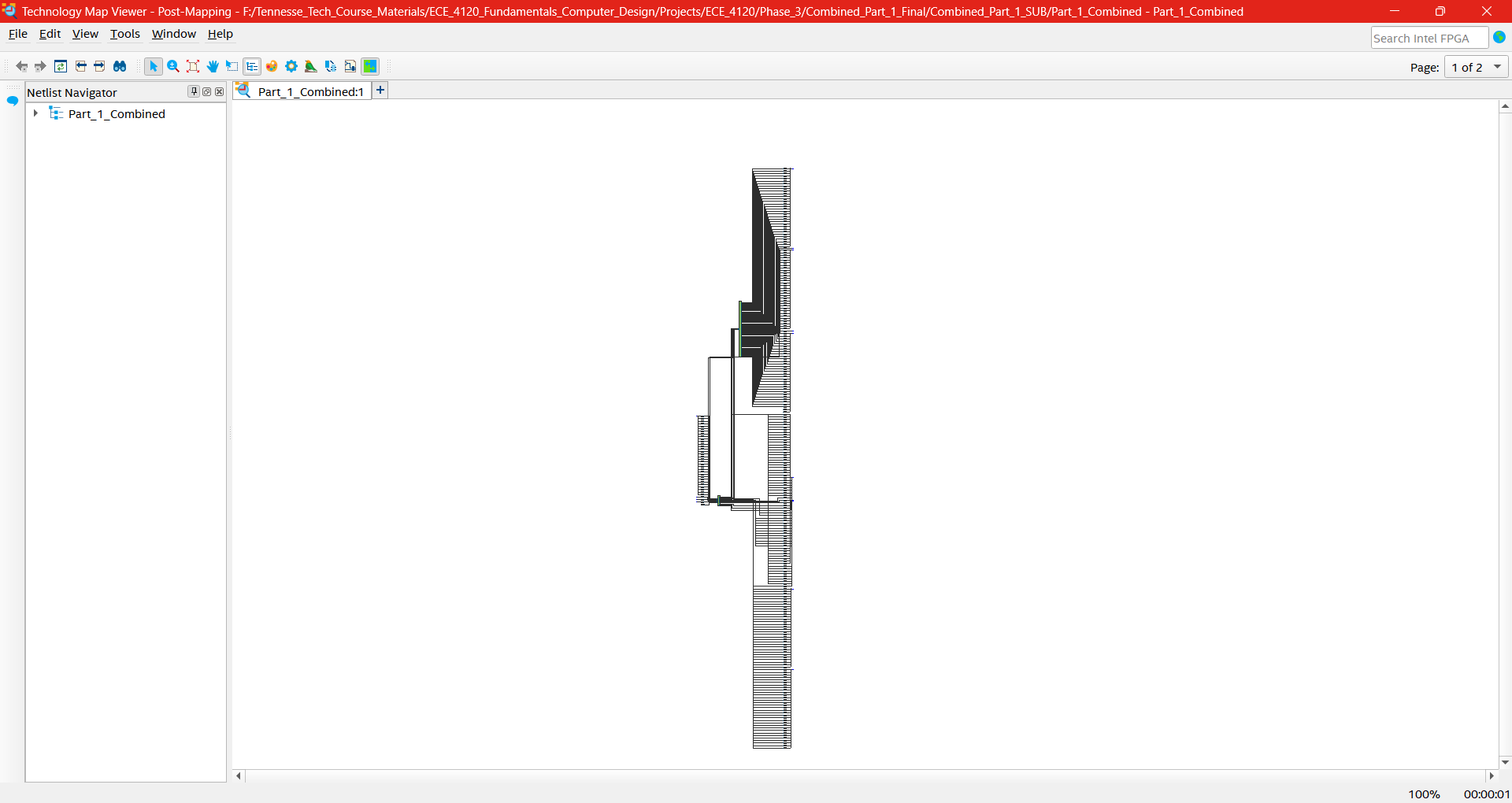
The single-cycle implementation is realized in Part\_1\_Combined.vhd, which integrates Part\_1\_IF\_ID.vhd and Part\_1\_EX\_MEM\_WB.vhd:

* **Control Unit (control\_unit.vhd)**:
  + Inputs: 6-bit opcode.
  + Outputs: 8 control signals based on opcode decoding (e.g., RegDst = '1' for R-type, Branch = '1' for beq).
  + Implemented with a case statement for instructions like add, sw, beq, and addi.
* **ALU (ALU.vhd)**:
  + Supports AND, OR, ADD, SUB, XOR, and NOR operations.
  + Outputs a 32-bit result and a Zero flag.
* **ALU Control (ALU\_Control.vhd)**:
  + Maps ALUOp (2-bit) and instr\_funct (6-bit) to a 4-bit ALU operation code.
  + Example: ALUOp = "01" (branch) sets operation to 0110 (subtract).
* **Data Memory (memory\_1.vhd)**:
  + Instantiated as a second memory IP, configured identically to instruction memory (256 words, 32-bit width).
  + Uses MemWrite as the write enable signal.
* **Multiplexers**:
  + **Mux1 (Part\_1\_IF\_ID.vhd)**: Selects rt or rd for write register using RegDst.
  + **Mux2 (ALU\_Mux.vhd)**: Selects ALU second operand using ALUSrc.
  + **Mux3 (Two\_Input\_Mux.vhd)**: Selects write-back data (ALU\_Output or data\_mem\_out) using MemtoReg.
  + **Mux4 (Two\_Input\_Mux.vhd)**: Selects next PC (PC\_plus\_4 or branch\_address) using branch\_enable.
* **Add Units**:
  + PC Adder (pc\_adder.vhd): Increments PC by 4.
  + Branch Adder (generic\_adder.vhd): Adds shifted sign-extended immediate to PC\_plus\_4.
* **Other Units**:
  + PC (PC.vhd): 32-bit register updated on clock edge.
  + Register File (register\_file.vhd): Manages 32 registers with read/write operations.
  + Sign Extend (SignExtend.vhd): Extends 16-bit immediate to 32 bits.
  + Shift Left by 2 (shift\_left\_by\_2.vhd): Shifts immediate for branch offset.

### Part 1: Flow Summary, RTL View, and Technology Map View



****

****

* **Flow Summary**: Details logic elements, registers, and memory bits used.
* **RTL View**: Shows schematic representation of the design.
* **Technology Map View**: Illustrates mapping to FPGA primitives.

### Part 1: Testbench Elaboration

The testbench (Part\_1\_Combined\_TB.vhd) verifies the execution of:

1. beq $t1, $t2, Equal
2. add $t1, $t1, $t2
3. sw $t3, 100($t2)
4. or $t1, $t4, $t2

* **Register Loading**: Preloaded using addi instructions in the testbench (e.g., $t1 = 1, $t2 = 2 $t3 = 3, $t4 = 4).
* **Instruction Memory**: Loaded with machine code via Instruction\_input and Instruction\_wren.
* **Verification**: Checks outputs like Read\_data\_1, ALU\_output, and Data\_memory\_out after each clock cycle.

### Part 1: Waveform Elaboration

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

* **Waveform Description**:
  + **beq**: The Branch was not taken at all.
  + **add**: Displays ALU\_output as the sum of $t1 and $t2.
  + **sw**: Indicates Data\_memory\_out and memory write at address 100 + $t2.
  + **or**: Shows ALU\_output as the bitwise OR of $t4 and $t2.
* **Labels**: Key events (e.g., register read, ALU operation, memory write) are annotated.

## Part 2: Pipelined Implementation

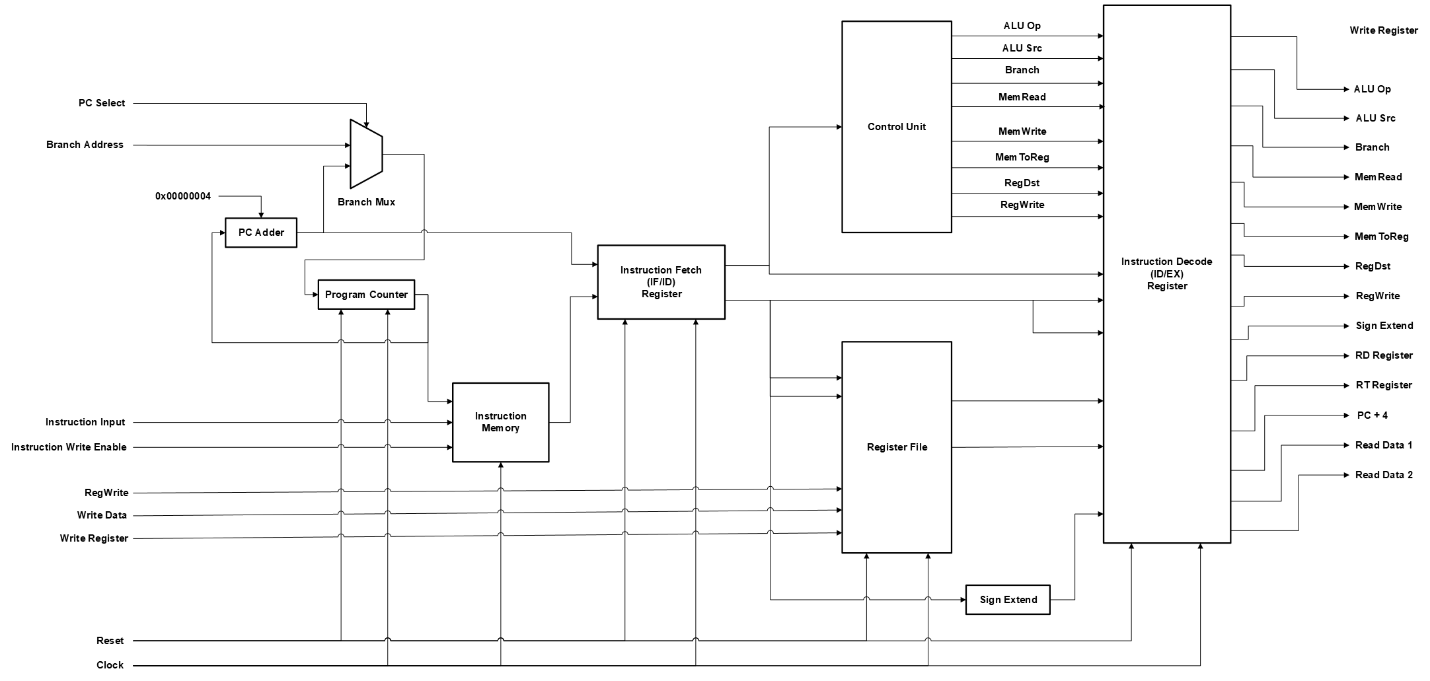
### Part 2: Modified Block Diagram

### Part 2: Top-Level Entity

A diagram of a computer

AI-generated content may be incorrect.

### Part2: Instruction Fetch and Instruction Decode Stages



### Part 2: Execute, Memory, and Write Back Stages

A diagram of a diagram

AI-generated content may be incorrect.

The pipelined processor splits execution into five stages (IF, ID, EX, MEM, WB) with intermediate registers:

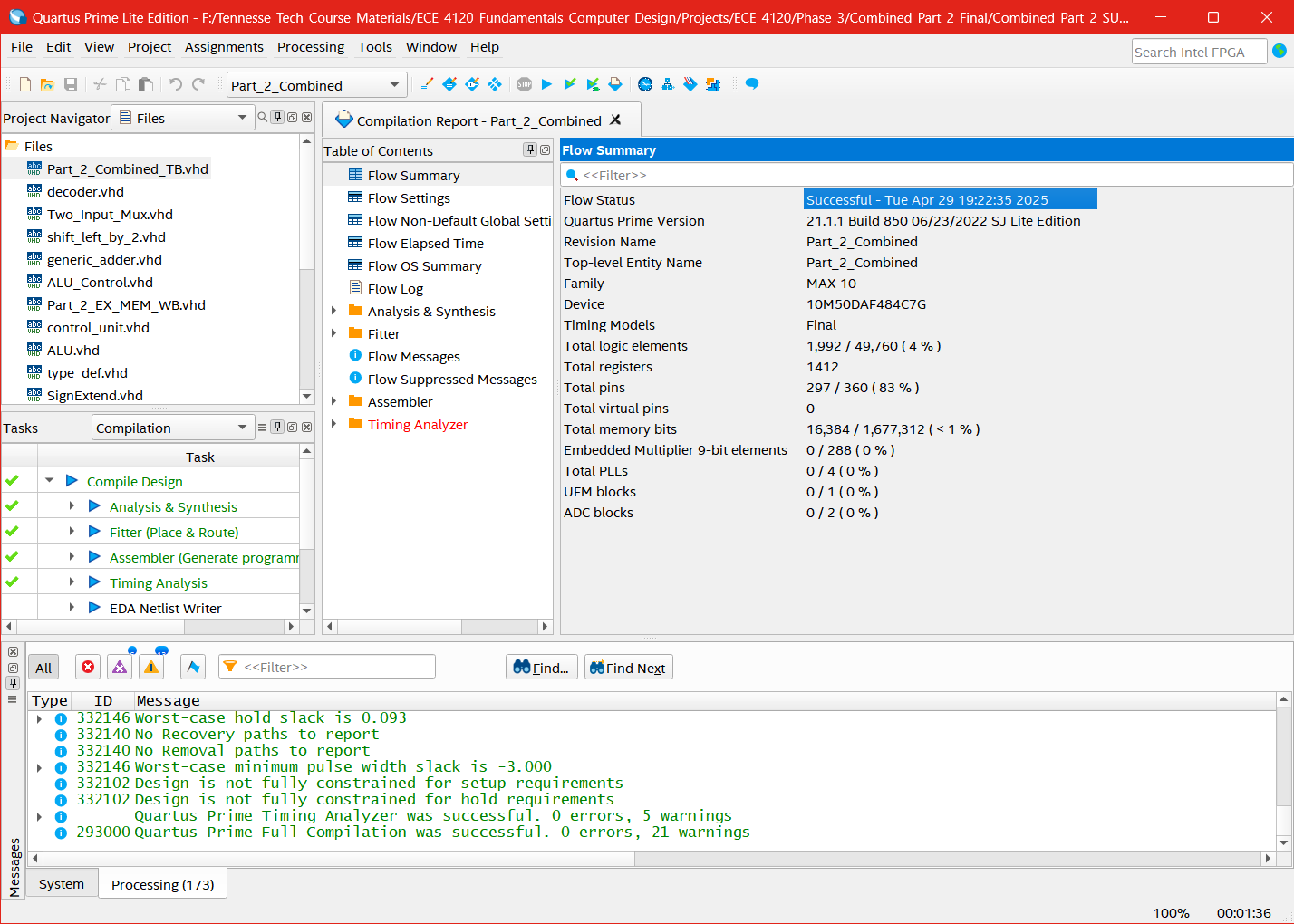
* **Clock Signals**: Single clock synchronizes all stages.
* **Data Widths**: Consistent with single-cycle (32-bit data, 5-bit addresses).
* **Control Signals**: Propagated through pipeline registers (e.g., ID\_EX\_RegDst, EX\_MEM\_MemWrite).

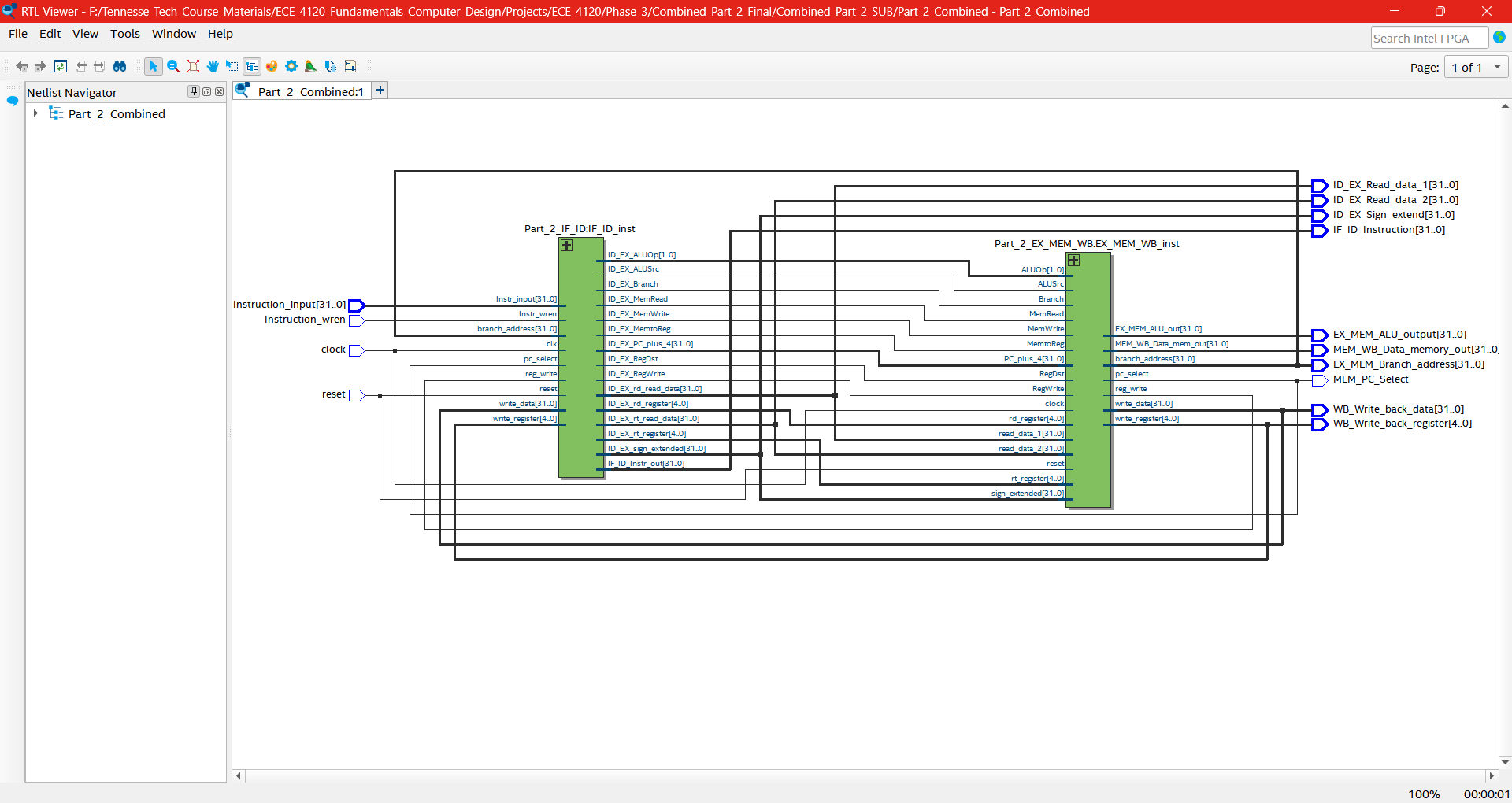
### Part 2: Elaboration on VHDL Implementation

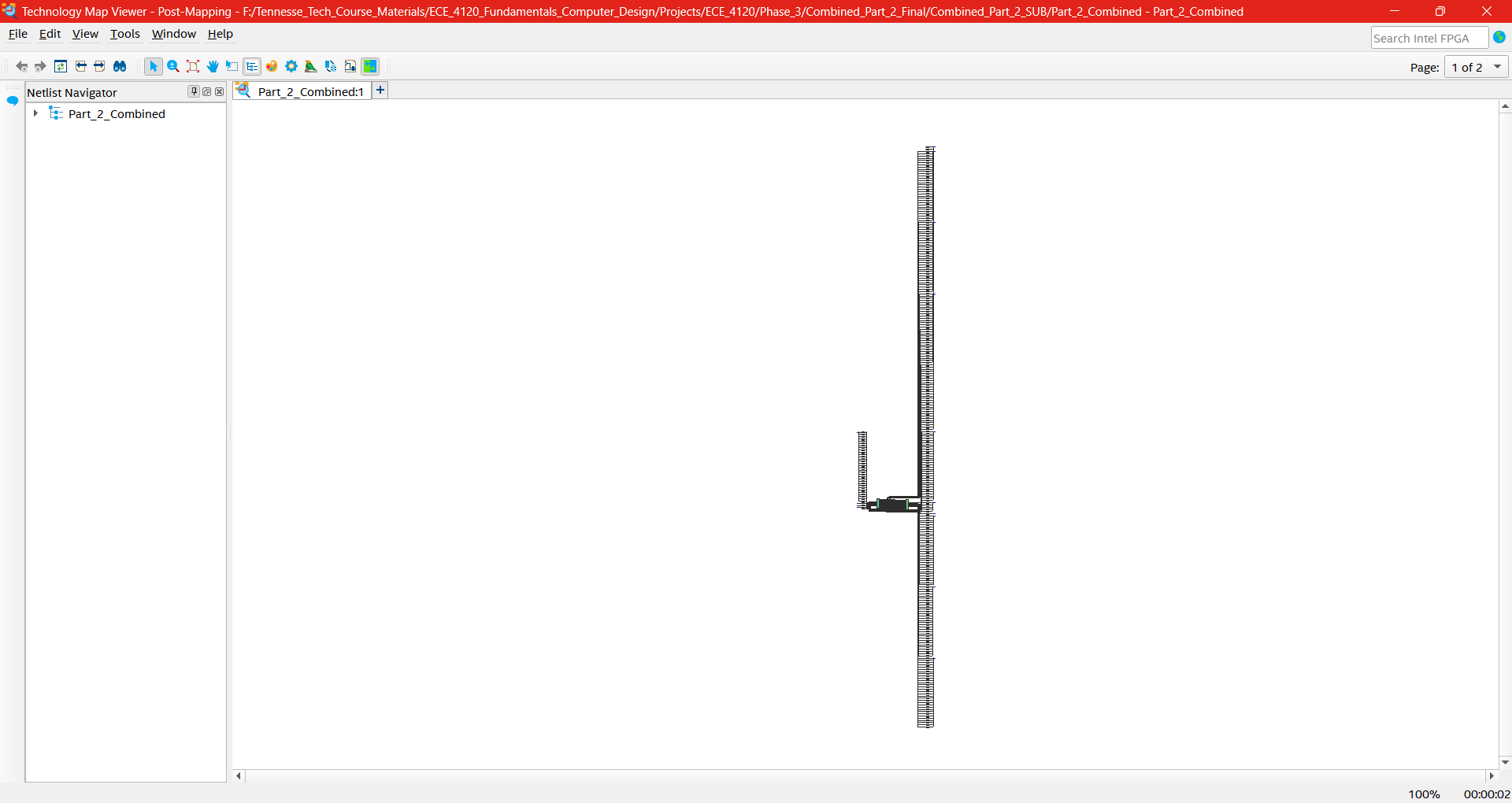
Implemented in Part\_2\_Combined.vhd, integrating Part\_2\_IF\_ID.vhd and Part\_2\_EX\_MEM\_WB.vhd:

* **Pipeline Registers (register\_N.vhd)**:
  + **IF/ID**: Stores instruction and PC\_plus\_4 (64 bits total).
  + **ID/EX**: Holds control signals, read\_data\_1, read\_data\_2, sign\_extended, and register addresses (107 bits).
  + **EX/MEM**: Stores branch\_address, ALU\_out, read\_data\_2, and control signals (107 bits).
  + **MEM/WB**: Contains data\_memory\_out, ALU\_out, write\_register, and control signals (71 bits).
* **IF Stage (Part\_2\_IF\_ID.vhd)**:
  + PC and instruction memory operate as in Part 1.
  + Mux selects between PC\_plus\_4 and branch\_address using pc\_select.
* **ID Stage (Part\_2\_IF\_ID.vhd)**:
  + Control unit and register file decode instructions.
  + Outputs latched into ID/EX register on clock edge.
* **EX Stage (Part\_2\_EX\_MEM\_WB.vhd)**:
  + ALU, ALU control, and branch calculation units execute operations.
  + Results stored in EX/MEM register.
* **MEM Stage (Part\_2\_EX\_MEM\_WB.vhd)**:
  + Data memory handles load/store operations.
  + Outputs to MEM/WB register.
* **WB Stage (Part\_2\_EX\_MEM\_WB.vhd)**:
  + Write-back mux selects data for register file write.

### Part 2: Flow Summary, RTL View, and Technology Map View







* **Flow Summary**: Increased resource usage due to pipeline registers.
* **RTL View**: Reflects staged architecture.
* **Technology Map View**: Shows FPGA resource allocation.

### Part 2: Testbench Elaboration

The testbench (Part\_2\_Combined\_TB.vhd) tests the same instructions as Part 1:

* **Register Loading**: Preloaded via addi in the testbench.
* **Instruction Memory**: Initialized similarly to Part 1.
* **Verification**: Monitors pipeline stage outputs (e.g., ID\_EX\_Read\_data\_1, EX\_MEM\_ALU\_out) over multiple cycles, assuming branch not taken.

### Part 2: Waveform Elaboration

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

* **Waveform Description**: Shows instructions progressing through IF, ID, EX, MEM, and WB stages across clock cycles.
* **Labels**: Indicate stage transitions (e.g., beq in IF, add in EX) and overlapping execution.

### Part 2: Improvements with Pipelining

* **Latency**: Remains 1 instruction per cycle for a single instruction but allows overlap.
* **Throughput**: Increases to 1 instruction per cycle after pipeline fills (e.g., 4 instructions in 7 cycles vs. 4 cycles in single-cycle).
* **Fmax**: Potentially higher due to shorter critical paths per stage (quantified in synthesis reports).

### Part 2: Hardware Overhead

* **Additional Resources**: Pipeline registers (IF/ID: 64 bits, ID/EX: 107 bits, EX/MEM: 107 bits, MEM/WB: 71 bits) increase logic elements and registers.
* **Quantification**: Detailed in flow summary (e.g., ~349 additional register bits).

## Conclusion

This report presents a complete implementation of a single-cycle MIPS processor (Part 1) and a pipelined version (Part 2). The single-cycle design executes instructions efficiently in one cycle, while the pipelined design improves throughput. All deliverables align with the project requirements, except for the data memory issue, in addition to our synthesis and simulation results to be inserted for final submission.