**#ALU with Finite State Machine (FSM) Controller**

**##Project Overview 💻**

This project implements a fundamental \*\***Arithmetic Logic Unit (ALU)\*\*** controlled by a dedicated \*\***Finite State Machine (FSM)\*\***. The entire system is designed and synthesized using the \*\***VHDL\*\*** (VHSIC Hardware Description Language) and the \*\***Intel Quartus Prime\*\*** software environment. It serves as a comprehensive demonstration of designing and integrating basic computer architecture components, combining both data path elements (the ALU and registers) and control path logic (the FSM).

**##Key Components and Design Structure ⚙️**

The system is modular, with each core function implemented as a separate VHDL entity, as seen in the file structure:

**\*\*1. The Data Path (ALU and Registers)\*\***

The data path handles the actual data processing.

* **\*\*ALU (Arithmetic Logic Unit):\*\*** Performs arithmetic operations (e.g., addition, subtraction) and logical operations (e.g., AND, OR, XOR).
* **\*\*Registers (DFF\_Latch, DFF\_B, DFF\_A):**\*\*Store input data (A and B) and the output/accumulator (Acc) based on control signals from the FSM.
* **\*\*Multiplexers (MUX\_Gates):** \*\*Select data inputs for the ALU based on the FSM's control signals (SEL).

**\*\*2. The Control Path (Finite State Machine - FSM.vhd)\*\***

The FSM is the \*\***central control unit\*\*** of the entire system.

* **\*\*Function:\*\*** It dictates the sequence of operations by generating all necessary control signals.
* **\*\*Inputs:\*\*** clock, reset, start, accumulate (from the user/external environment).
* **\*\*Outputs:** **State\_Out** \*\*(the 3-bit control word),\*\* **WR\_A**, **WR\_B**, **WR\_C** \*\*(Write-Enable signals for the registers), and \*\***SEL\*\*** (Selector for the multiplexers).
* **\*\*States:\*\*** The FSM typically sequences through states like \*\***init**, **fetch**, **proc** \*\*(process/execute), and \*\***write\_ot\*\*** (write output). Each state corresponds to a specific step in an instruction cycle and sets a unique set of control output signals.

**##FSM State Sequence (High-Level) 🧭**

The FSM controls the system's operation cycle, which generally follows these steps:

1. **\*\*init (State 000):\*\*** The default idle state. Waits for a start signal.
2. **\*\*fetch (State 001):\*\*** Loads the initial operands from inputs into the A and B registers (controlled by WR\_A and WR\_B). The State\_Out is set to 001.
3. **\*\*proc (State 010):\*\*** Executes the ALU operation on the loaded data. The State\_Out is set to 010.
4. **\*\*write\_ot (State 100):\*\*** Writes the result from the ALU output into the \*\***Accumulator/Output Register** \*\*(Acc controlled by WR\_C). The State\_Out is set to 100. The FSM then transitions back to the init state or repeats based on a signal like accumulate.

**##Implementation Details and Technology 🛠️**

* **\*\*Language:\*\*** VHDL
* **\*\*Platform:\*\*** Intel Quartus Prime 64-bit
* **\*\*Design Flow:\*\*** The individual VHDL components are integrated using the Quartus Block Diagram/Schematic Editor (Final\_Project\_ALU.bdf) to form the complete top-level circuit, ensuring correct signal routing and component instantiations.

This project is a strong foundation for understanding the basic operational principles of a \*\***Central Processing Unit (CPU)\*\***, where a dedicated control unit (like this FSM) orchestrates the flow of data through the execution units (like the ALU and registers).