Team Ramsey.001

Kavon Ramsey

**List of Parts**

* Inputs
  + Clock 1-bit, feeds into the clock of the accumulator register
  + Input 1 (16-bits), the first binary integer used for input
  + Input 2 (16-bits), the second binary integer used for input
  + Opcode (4-bits), feeds into select of Multiplexer to select operation
* Outputs
* Output (16-bits), displays the results of the accumulator
* Interfaces (CIRC Diagram CHANNELS labeled from 0-15)
* Multiplexer (Mux16) (b) (16-bits), is the output of the multiplexer into the accumulator. Feeds into DFF16 (ACC) next.
* Multiplexer (Mux16) (S) (4-bits), is an input for the opcode to select multiplexer channels (opcode)
* Multiplexer (Mux16) (CH0) (16-bits), is a feedback loop input for current accumulator (modeNOOP)
* Multiplexer (Mux16) (CH1) (16-bits), input for the 16-bit adder results (add\_val)
* Multiplexer (Mux16) (CH2) (16-bits), input for the 16-bit subtractor results (sub\_val)
* Multiplexer (Mux16) (CH3) (16-bits), input for the 32-bit multiplier results (product)
* Multiplexer (Mux16) (CH4) (16-bits), input for the 16-bit divider results (quotient)
* Multiplexer (Mux16) (CH5) (16-bits), input for the 16-bit AND gate results (and\_val)
* Multiplexer (Mux16) (CH6) (16-bits), input for the 16-bit OR gate results (or\_val)
* Multiplexer (Mux16) (CH7) (16-bits), input for the 16-bit XOR gate results (xor\_val)
* Multiplexer (Mux16) (CH8) (16-bits), input for the 16-bit NOT gate results (not\_val)
* Multiplexer (Mux16) (CH9) (16-bits), unused. (muxPORT9) empty wire
* Multiplexer (Mux16) (CH10) (16-bits), unused. (muxPORT10) empty wire
* Multiplexer (Mux16) (CH11) (16-bits), unused. (muxPORT11) empty wire
* Multiplexer (Mux16) (CH12) (16-bits), unused. (muxPORT12) empty wire
* Multiplexer (Mux16) (CH13) (16-bits), unused. (muxPORT13) empty wire
* Multiplexer (Mux16) (CH14) (16-bits), unused. (muxPORT14) empty wire
* Multiplexer (Mux16) (CH15) (16-bits), 16-bit 0s used to reset accumulator (modeRESET)
* ADD adder (add\_val) (16-bits) feeds addition into Multiplexer (CH1)
* SUB subtractor (sub\_val) (16-bits) feeds subtraction into Multiplexer (CH2)
* MULT multiply (product) (32-bits TRUNC 16) feeds product into Multiplexer (CH3)
* DIV divider (quotient) (16-bits) feeds quotient into Multiplexer (CH4)
* AND ander (and\_val) (16-bits) feeds AND into Multiplexer (CH5)
* OR mor (or\_val) (16-bits) feeds OR into Multiplexer (CH6)
* XOR mxor (xor\_val) (16-bits) feeds XOR into Multiplexer (CH7)
* NOT mnot (not\_val) (16-bits) feeds NOT into Multiplexer (CH8)
* (Verilog DFF16) ACC (D) (16-bits), is the value being fed into the accumulator
* (Verilog DFF16) ACC (Q) (16-bits), is the current value held in the register
* Gates
* AND gate (16-bits), 2 channels take Input 1 and Input 2 to AND them. Output channel feeds result into (CH5) Multiplexer
* OR gate (16-bits), 2 channels take Input 1 and Input 2 to OR them. Output channel feeds result into (CH6) Multiplexer
* XOR gate (16-bits), 2 channels take Input 1 and Input 2 and exclusive ORs them into output channel (CH7) Multiplexer
* NOT gate (16-bits), 1 channel takes Input 1 and returns the result of the NOT operation. Channel (CH8) of Multiplexer is fed the output
* Arithmetic Components
* Adder (16-bit), arithmetic component to add 2 16-bit inputs. (**Note this is done behaviorally in Verilog in module ALU. Circuit diagram uses built in adder**)
* Subtractor (16-bit), arithmetic component to subtract 2 16-bit inputs. (**Note this is done behaviorally in Verilog in module ALU. Circuit diagram uses built in subtractor**)
* Multiplier (32-bit), arithmetic component to multiply 2 16-bit inputs (Need truncated to 16-bit). (**Note this is done behaviorally in Verilog in module ALU. Circuit diagram uses built in multiplier**)
* Divider (16-bit), arithmetic component to divide 2 16-bit inputs. (**Note this is done behaviorally in Verilog in module ALU. Circuit diagram uses built in divider**)
* Combinational Logic
* Multiplexer (16-bit) 16 channels, used to select desired operation and result (Verilog Mux16)
* Module ADD adder (16-bit) would be considered Combinational Logic. This module. (**Note this is done behaviorally in Verilog in module ALU. Circuit diagram uses built in adder**)
* Module SUB subtractor (16-bit) would be considered Combinational Logic. (**Note this is done behaviorally in Verilog in module ALU. Circuit diagram uses built in subtractor**)
* Module MULT multiply (32-bit) would be considered Combinational Logic. This module does behavioral multiplication (32-bit) truncated to 16-bit (product). (**Note this is done behaviorally in Verilog in module ALU. Circuit diagram uses built in multiplier**)
* Module DIV divider (16-bit). Module does behavioral division (quotient). (**Note this is done behaviorally in Verilog in module ALU. Circuit diagram uses built in divider**)
* Sequential Logic
* ACC (16-bit), D register that contains current system value (Verilog DFF16)
* Modules
* (Verilog DFF16) ACC. This module serves as the ACC accumulator
* (Verilog Mux16) choice. This module serves as the multiplexer to channel the selected operation into the ACC
* ADD adder (16-bit). This module does behavioral addition (add\_val)
* SUB subtractor (16-bit). This module does behavioral subtraction (sub\_val)
* MULT multiply (32-bit). This module does behavioral multiplication (32-bit) truncated to 16-bit (product)
* DIV divider (16-bit). Module does behavioral division (quotient)
* AND ander (16-bit). Module does an AND operation using AND gate (and\_val)
* OR mor (16-bit). Module does an OR operation using OR gate. (or\_val)
* XOR mxor (16-bit). Module does an XOR operation using XOR gate. (xor\_val)
* NOT mnot (16-bit). Module does NOT to input 1 using NOT. (not\_val)
* ALU. This module is to be the breadboard for the ALU where all components and functions are set. **All Logic and Math operations performed within this module**
* Testbench. This module runs the clock and stimulus

**OP-CODE**

|  |  |  |
| --- | --- | --- |
| **Command** | **Opcode** | **Description** |
| NO-OP | 0000 | No operation on this clock tick |
| ADD | 0001 | Feed adder into ACC |
| SUB | 0010 | Feed subtractor into ACC |
| MULT | 0011 | Feed multiplier into ACC |
| DIV | 0100 | Feed divider into ACC |
| AND | 0101 | Feed AND into ACC |
| OR | 0110 | Feed OR into ACC |
| XOR | 0111 | Feed XOR into ACC |
| NOT | 1000 | Feed NOT into ACC |
| RESET | 1111 | Reset ACC with 0s |

**MODE OF OPERATION**

|  |  |  |
| --- | --- | --- |
| **Mode** | **Code** | **Description** |
| Ready | 0000 | System is Idle |
| ADD | 0001 | Perform Addition |
| SUB | 0010 | Perform Subtraction |
| MULT | 0011 | Perform Multiplication |
| DIV | 0100 | Perform Division |
| AND | 0101 | Perform AND operation |
| OR | 0110 | Perform OR operation |
| XOR | 0111 | Perform XOR operation |
| NOT | 1000 | Perform NOT operation |
| RESET | 1111 | Reset ACC with 0s |

**STATE TABLE (SIMPLIFIED)**

|  |  |  |
| --- | --- | --- |
| **Current State** | **Command** | **Next State** |
| Ready | {ADD,SUB,MULT,DIV} | MATH |
| Ready | {AND,OR,XOR,NOT} | LOGIC |
| Ready | {NOOP,RESET} | Ready |
| MATH {ADD,SUB,MULT,DIV} | {NOOP,RESET} | Ready |
| MATH {ADD,SUB,MULT,DIV} | {ADD,SUB,MULT,DIV} | MATH |
| MATH {ADD,SUB,MULT,DIV} | {AND,OR,XOR,NOT} | LOGIC |
| LOGIC {AND,OR,XOR,NOT} | {NOOP,RESET} | Ready |
| LOGIC {AND,OR,XOR,NOT} | {AND,OR,XOR,NOT} | LOGIC |
| LOGIC {AND,OR,XOR,NOT} | {ADD,SUB,MULT,DIV} | MATH |

**State Diagram**

**A close up of text on a whiteboard

Description automatically generated**

**NOTE: My state diagram is a condensed/simplified version of what the state machine would appear. The output of my Verilog tries its best to simulate the transitions between the 3 condensed states. Doing the entire combination of states would be too much. My output shows the transition from READY to READY using NOOP and RESET, READY to MATH doing {NOOP->ADD}, MATH to MATH doing {ADD ->SUB}, {MULT -> DIV}. They can transition among each other, but I placed a NOOP to represent state transition from MATH to READY {SUB->NOOP}, {ADD->NOOP} and READY to MATH again {NOOP->MULT}. MATH to LOGIC {DIV->AND}, {ADD->NOT}. Again, MATH to LOGIC should do transitions to each math operation to logic operation. Including each transition would generate way too much output. LOGIC to LOGIC {AND->OR}, {XOR->NOT}. They can transition between all operations between each other. LOGIC to READY {OR->NOOP}, {NOT->RESET}. READY to LOGIC {NOOP->XOR}, {RESET->AND}. LOGIC to MATH {AND->ADD}, {NOT->ADD}. These should be able to transition among each other as I keep repeating but for my simplicity and yours, output is simplified to simplify the transition between the simplified 9.**

**CIRCUIT**

**Diagram, schematic

Description automatically generated**

**ONE key thing. Diagram Multiplexer shows 0-15 which in this System design shows CH 0-15**

**NOTE: I have tried to be consistent with names as possible. There are differences in naming conventions across the circuit and code and I have tried to link associated names among each the circuit and code as consistently as possible. This System design was designed from scratch after disbanding from the cohort. It does 16 bit integers and will store functions into the DFF ACC which can be seen using NOOP.**