CSEE 6863 Formal Verification

Verification on an ALU_FIFO Module

Weijie Wang (ww2739)

Zhenqi Li (zl3508)

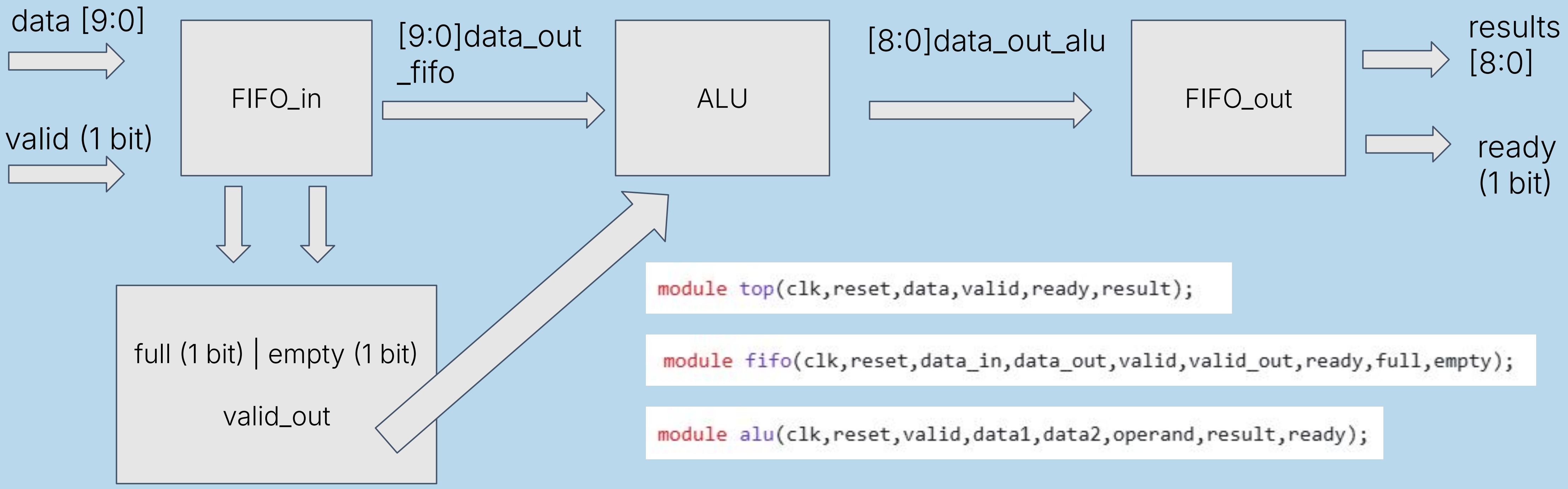
Siyuan Li (sl5590)

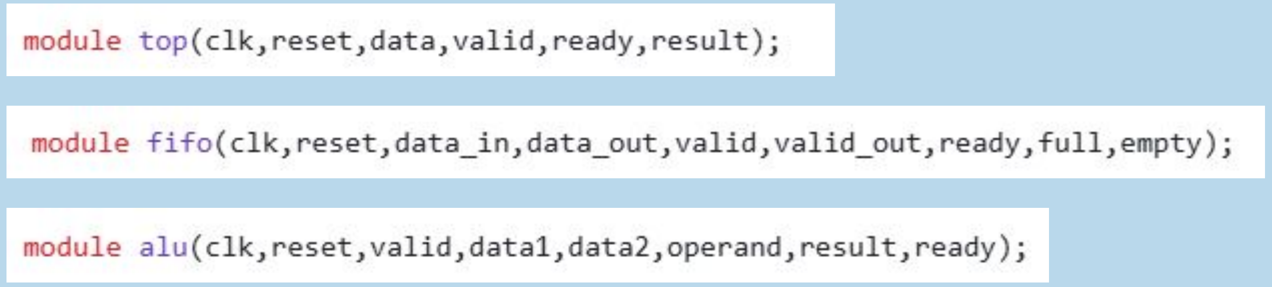
COLUMBIA ENGINEERING
The Fu Foundation School
of Engineering and Applied Science

CONTENT

- Module overview
- FIFO
- Alu

Module Overview





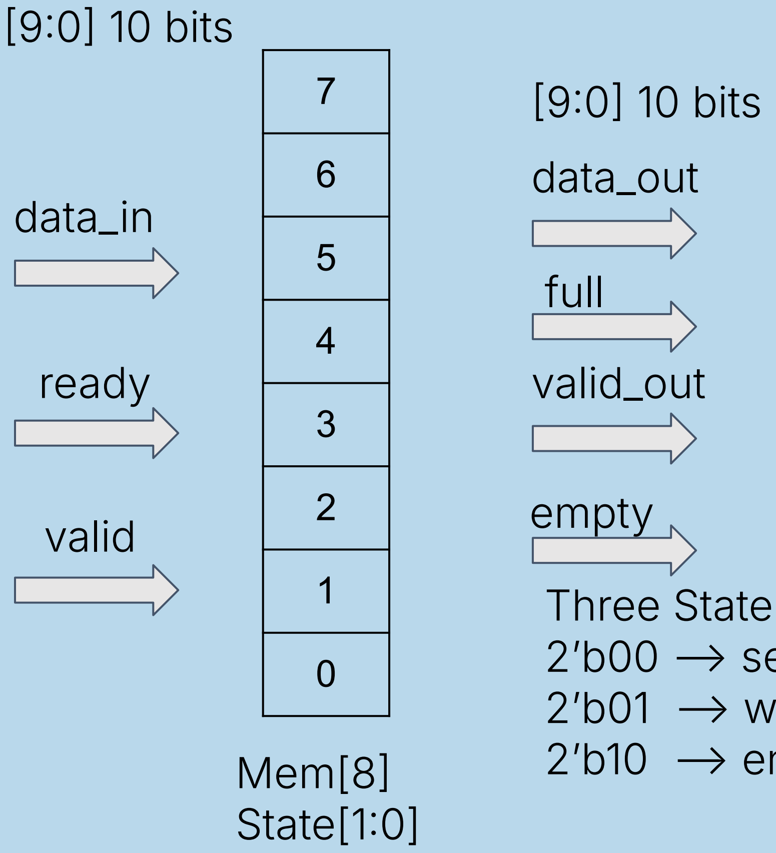
```
module top(clk,reset,data,valid,ready,result);
```

```
module fifo(clk,reset,data_in,data_out,valid,valid_out,ready,full,empty);
```

```
module alu(clk,reset,valid,data1,data2,operand,result,ready);
```

FIFO STRUCTURE

fifo (clk, reset, data_in, data_out, valid, valid_out, ready, full, empty)

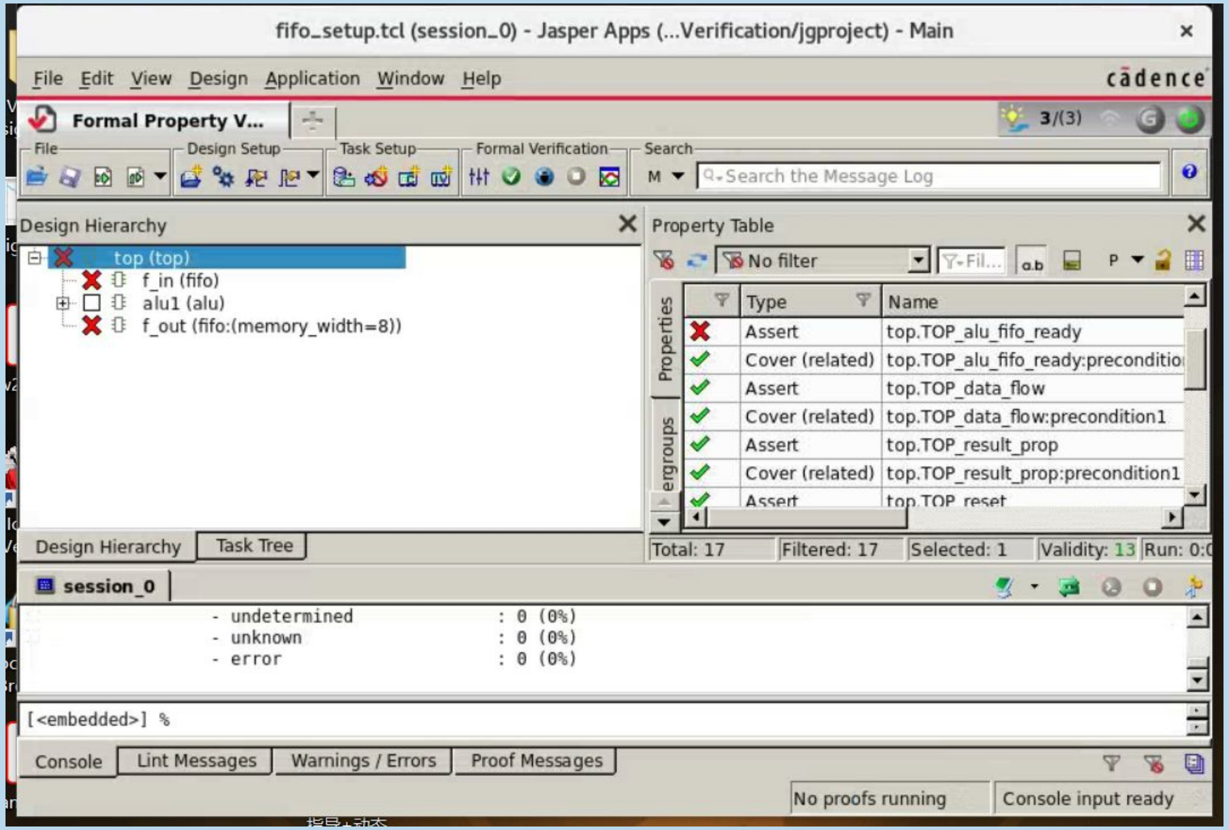


Three State:
2'b00 → send;
2'b01 → wait_ready;
2'b10 → end_tx;

Assertions in the Top Module

```
// Verify FIFO-ALU handshaking
TOP_fifo_alu_valid: assert property (@(posedge clk)
    f_in.valid_out |-> !alu1.ready);

// Verify ALU-FIFO handshaking
TOP_alu_fifo_ready: assert property (@(posedge clk)
    alu1.ready |-> !f_out.full);

// Verify data integrity between FIFOs and ALU
TOP_data_flow: assert property (@(posedge clk)
    f_in.valid_out |-> (f_in.data_out[3:0] == alu1.data1 &&
                        f_in.data_out[7:4] == alu1.data2 &&
                        f_in.data_out[9:8] == alu1.operand));
```

```
// Verify result propagation
TOP_result_prop: assert property (@(posedge clk)
    alu1.ready |-> (alu1.result == f_out.data_in));

TOP_reset: assert property (@(posedge reset)
    (f_in.empty && f_out.empty && alu1.ready));
```



Design Hierarchy:
- top (top)
  - f_in (fifo)
  - alu1 (alu)
  - f_out (fifo:(memory_width=8))

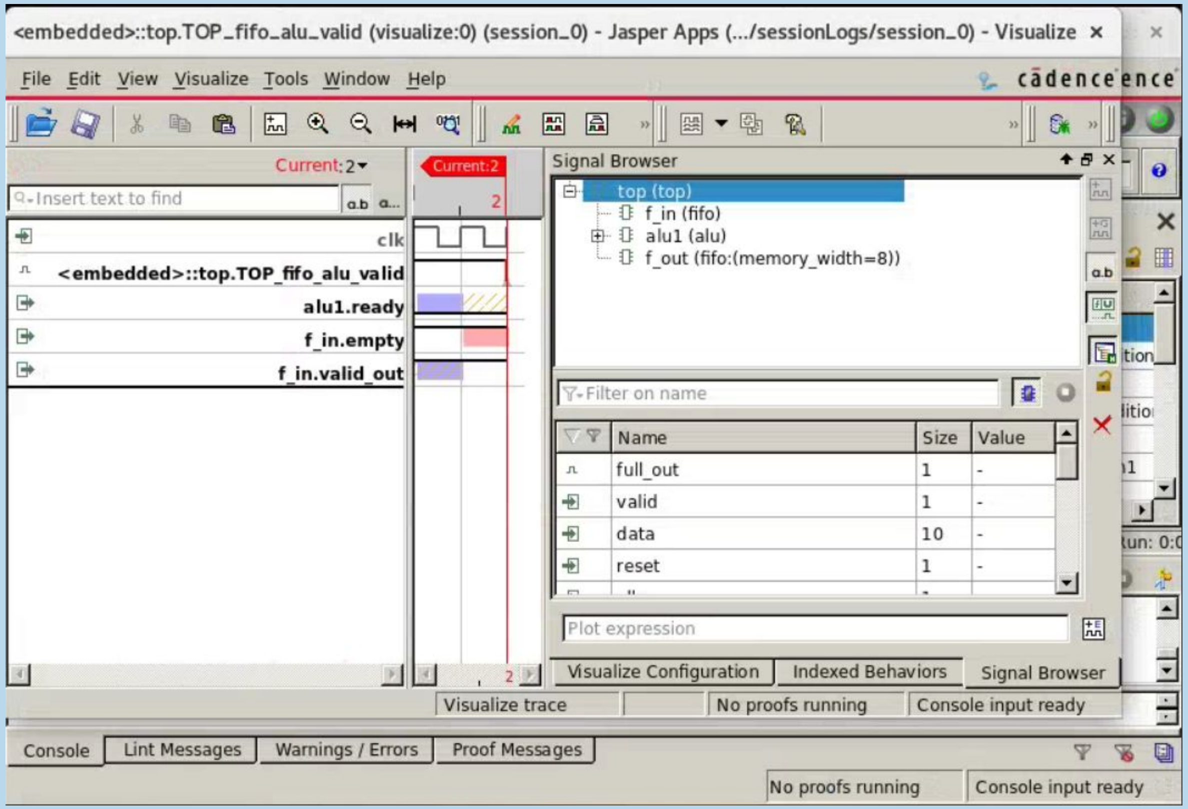
	Type	Name
✗	Assert	top.TOP_alu_fifo_ready
✓	Cover (related)	top.TOP_alu_fifo_ready:preconditio
✓	Assert	top.TOP_data_flow
✓	Cover (related)	top.TOP_data_flow:precondition1
✓	Assert	top.TOP_result_prop
✓	Cover (related)	top.TOP_result_prop:precondition1
✓	Assert	top.TOP_reset

Total: 17 Filtered: 17 Selected: 1 Validity: 13 Run: 0:0

```
           - undetermined             : 0 (0%)
           - unknown                  : 0 (0%)
           - error                    : 0 (0%)
[<embedded>] %
```

No proofs running Console input ready

Challenge We Faced



<embedded>::top.TOP_fifo_alu_valid (visualize:0) (session_0) - Jasper Apps (.../sessionLogs/session_0) - Visualize

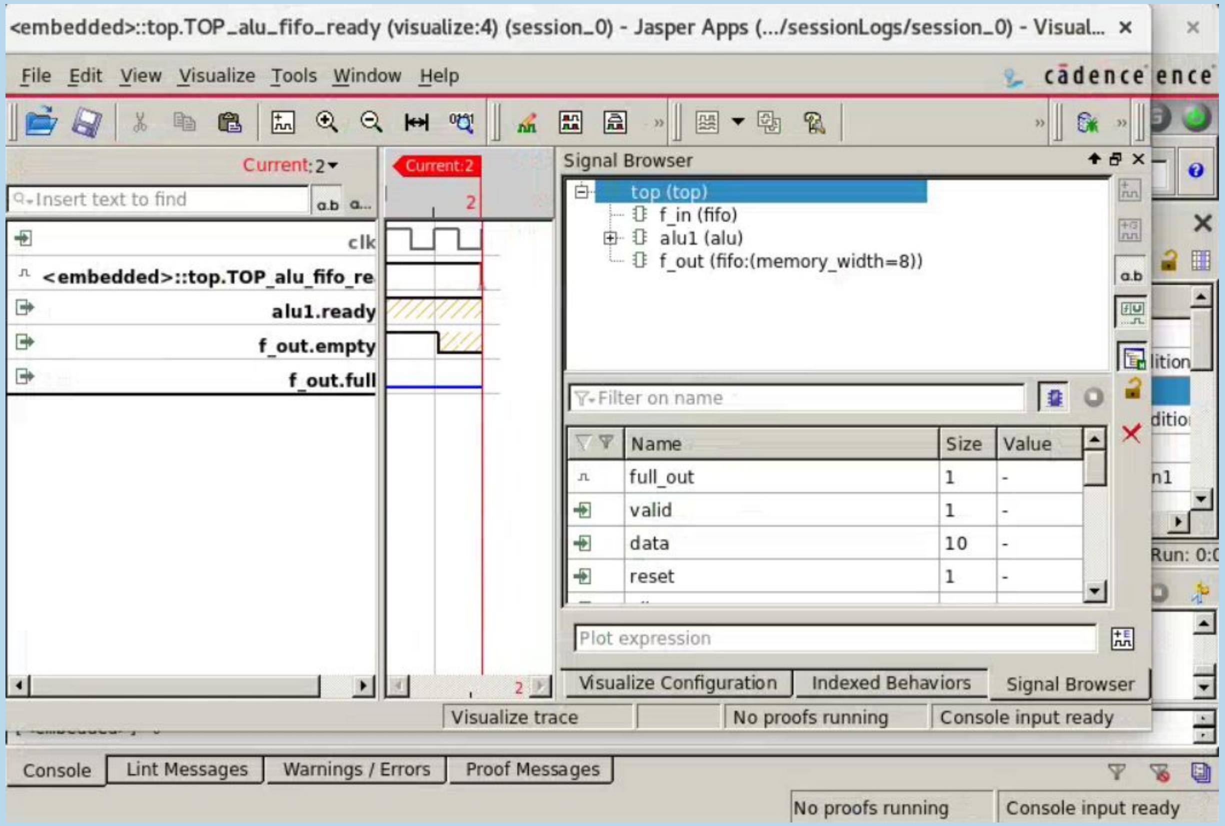
clk
<embedded>::top.TOP_fifo_alu_valid
alu1.ready
f_in.empty
f_in.valid_out

top (top)
f_in (fifo)
alu1 (alu)
f_out (fifo:(memory_width=8))

Name	Size	Value
full_out	1	-
valid	1	-
data	10	-
reset	1	-

Visualize trace | No proofs running | Console input ready

No proofs running | Console input ready



<embedded>::top.TOP_alu_fifo_ready (visualize:4) (session_0) - Jasper Apps (.../sessionLogs/session_0) - Visual...

clk
<embedded>::top.TOP_alu_fifo_re
alu1.ready
f_out.empty
f_out.full

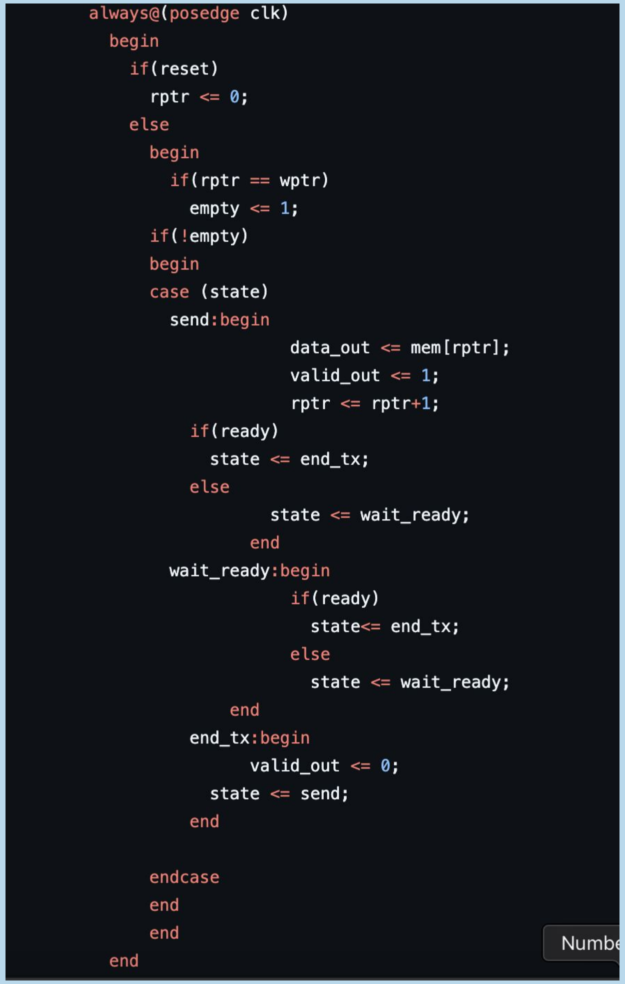
top (top)
f_in (fifo)
alu1 (alu)
f_out (fifo:(memory_width=8))

Name	Size	Value
full_out	1	-
valid	1	-
data	10	-
reset	1	-

Visualize trace | No proofs running | Console input ready

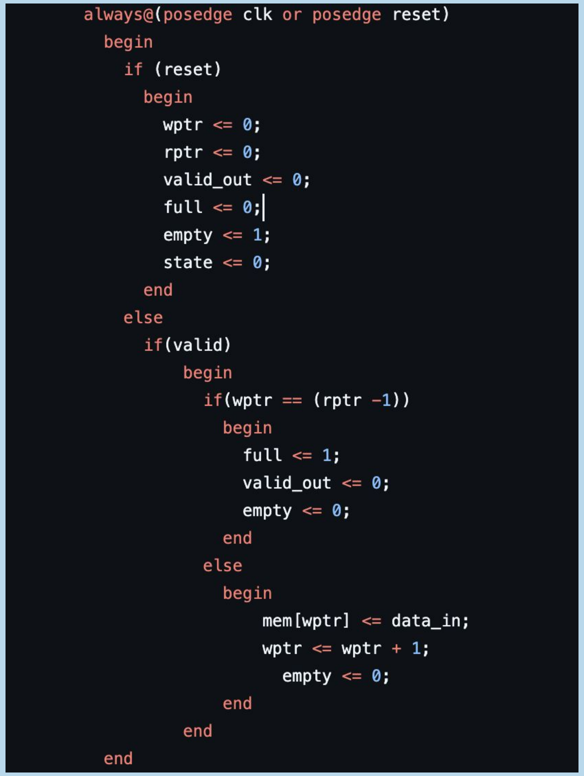
No proofs running | Console input ready

FIFO VERIFICATION



```
always@(posedge clk)
  begin
    if(reset)
      rptr <= 0;
    else
      begin
        if(rptr == wptr)
          empty <= 1;
      if(!empty)
      begin
      case (state)
        send:begin
                    data_out <= mem[rptr];
                    valid_out <= 1;
                    rptr <= rptr+1;
          if(ready)
            state <= end_tx;
          else
                  state <= wait_ready;
                end
        wait_ready:begin
                    if(ready)
                      state<= end_tx;
                    else
                      state <= wait_ready;
              end
          end_tx:begin
                valid_out <= 0;
            state <= send;
          end

      endcase
      end
      end
  end
```

fifo_read(original)



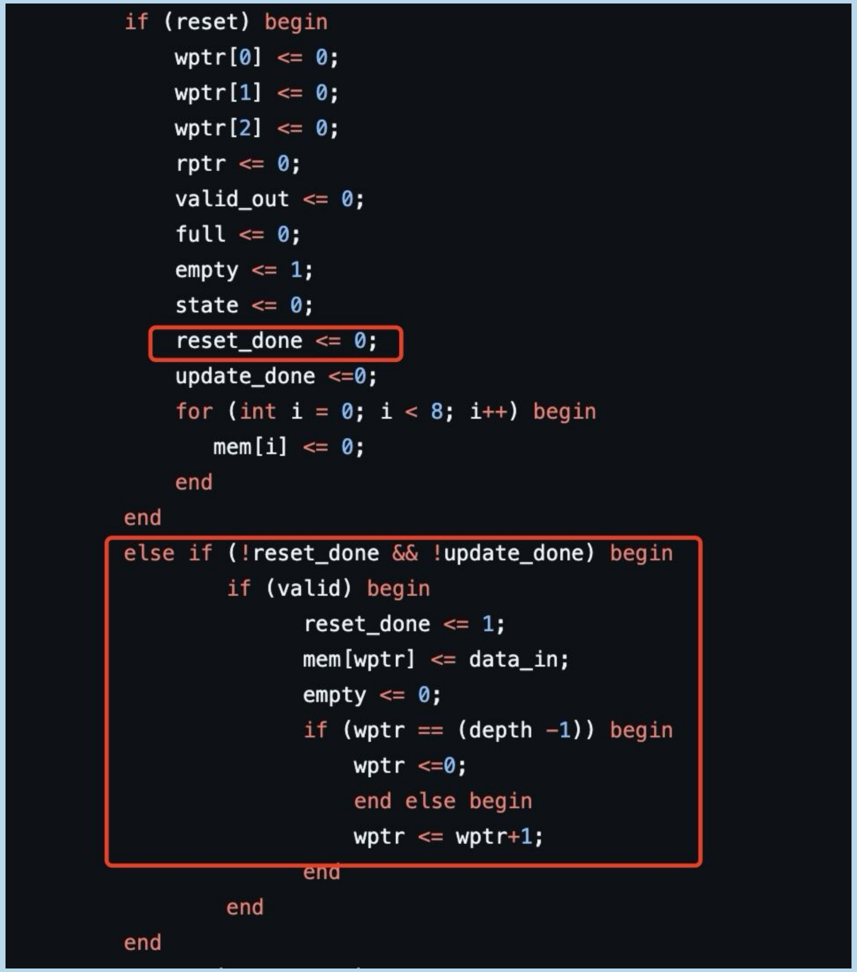
```
always@(posedge clk or posedge reset)
  begin
    if (reset)
      begin
        wptr <= 0;
        rptr <= 0;
        valid_out <= 0;
        full <= 0;
        empty <= 1;
        state <= 0;
      end
    else
      if(valid)
          begin
            if(wptr == (rptr -1))
              begin
                full <= 1;
                valid_out <= 0;
                empty <= 0;
              end
            else
              begin
                  mem[wptr] <= data_in;
                  wptr <= wptr + 1;
                    empty <= 0;
              end
          end
  end
```

fifo_write(original)

Some issues in the FIFO verification:

- **Multiple-Driven Conflict for empty:**
  - ❖ After reset, both wptr and rptr are initialized to 0, leading to different assignments to empty from two different always blocks at the same time.
  - ❖ Merged the empty and full state logic into a single always block to prevent multiple assignments.

FIFO VERIFICATION



```
if (reset) begin
    wptr[0] <= 0;
    wptr[1] <= 0;
    wptr[2] <= 0;
    rptr <= 0;
    valid_out <= 0;
    full <= 0;
    empty <= 1;
    state <= 0;
    reset_done <= 0;
    update_done <=0;
    for (int i = 0; i < 8; i++) begin
       mem[i] <= 0;
    end
end
else if (!reset_done && !update_done) begin
        if (valid) begin
              reset_done <= 1;
              mem[wptr] <= data_in;
              empty <= 0;
              if (wptr == (depth -1)) begin
                  wptr <=0;
                  end else begin
                  wptr <= wptr+1;
              end
        end
end
```

add reset_done

Some issues in the FIFO verification:

- **Deadlock after Reset:**
  - ❖ After reset, both wptr and rptr being 0 causes empty to remain active (never change), which prevents wptr and rptr from operating, leading to a deadlock.
  - ❖ Introduced a new internal variable reset_done to enforce wptr operations first.

- **Multiple-Driven Conflict for valid_out and rptr:**
  - ❖ Valid_out and rptr were assigned values in multiple places, causing multiple driver conflicts.
  - ❖ Refined the variable assignment across always blocks to ensure no multiple drivers.

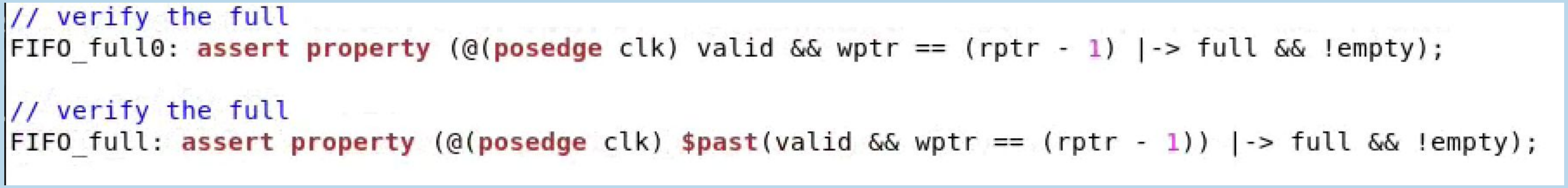
FIFO VERIFICATION

Some issues in the FIFO verification:

- **Delay between wptr, rptr updates, and empty/full state updates:**

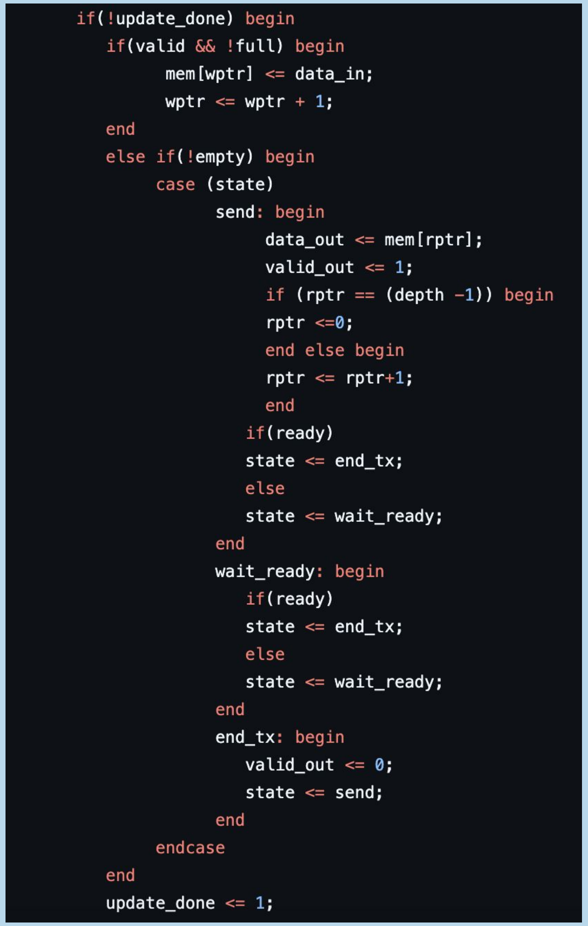
  ❖ Since pointer (wptr and rptr) updates happen on the clock's rising edge, the empty and full states take one additional clock cycle to reflect changes.

  ❖ We used the $past operator in assertions to account for the delay in state updates, ensuring the proper synchronization between pointer updates and state evaluations.

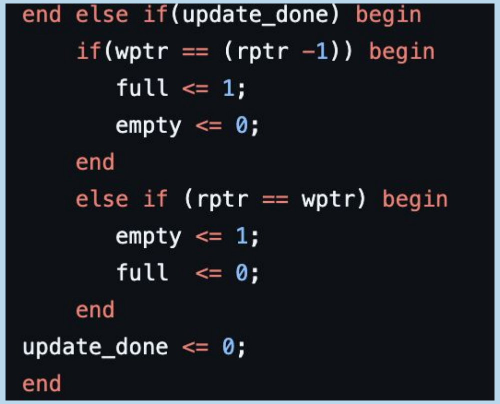


```
// verify the full
FIFO_full0: assert property (@(posedge clk) valid && wptr == (rptr - 1) |-> full && !empty);

// verify the full
FIFO_full: assert property (@(posedge clk) $past(valid && wptr == (rptr - 1)) |-> full && !empty);
```

FIFO VERIFICATION



```
if(!update_done) begin
   if(valid && !full) begin
         mem[wptr] <= data_in;
         wptr <= wptr + 1;
   end
   else if(!empty) begin
        case (state)
              send: begin
                   data_out <= mem[rptr];
                   valid_out <= 1;
                   if (rptr == (depth -1)) begin
                   rptr <=0;
                   end else begin
                   rptr <= rptr+1;
                   end
                 if(ready)
                 state <= end_tx;
                 else
                 state <= wait_ready;
              end
              wait_ready: begin
                 if(ready)
                 state <= end_tx;
                 else
                 state <= wait_ready;
              end
              end_tx: begin
                 valid_out <= 0;
                 state <= send;
              end
        endcase
   end
   update_done <= 1;
```



```
end else if(update_done) begin
    if(wptr == (rptr -1)) begin
       full <= 1;
       empty <= 0;
    end
    else if (rptr == wptr) begin
       empty <= 1;
       full  <= 0;
    end
update_done <= 0;
end
```

add update_done

Some issues in the FIFO verification:

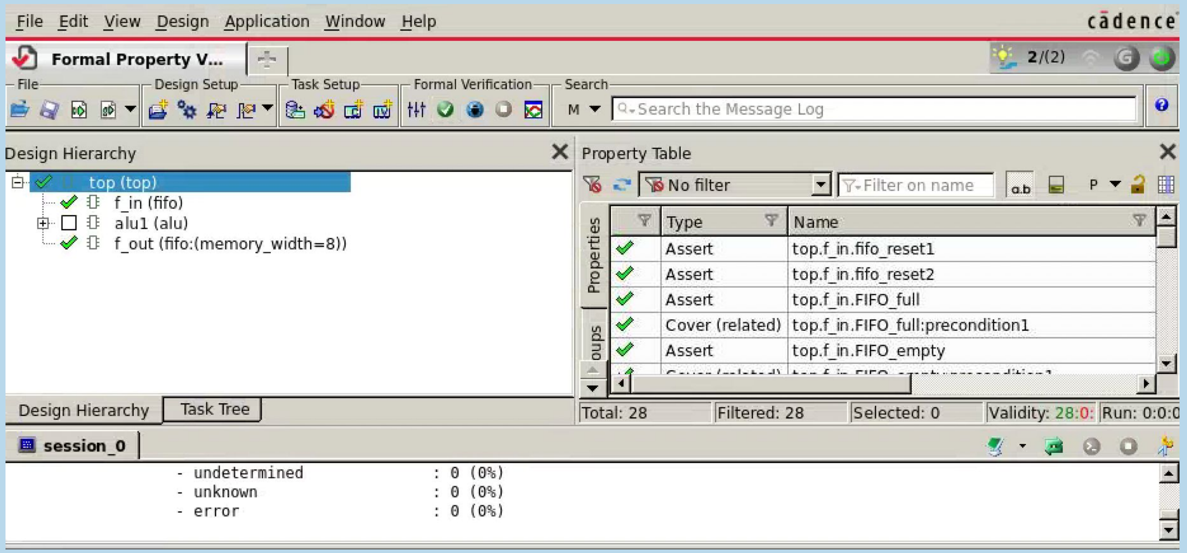
- **Logic error in rptr and state updates:**
  - ❖ rptr was updated in one always blocks, while empty, full, and wptr were handled in the other block, causing synchronization problems.
  - ❖ We merged the two always blocks into one, ensuring wptr and rptr are updated first. We added an internal variable update_done to ensure the empty and full states are only evaluated after the pointers are updated.

FIFO VERIFICATION

```
// reset: fifo ->reset
fifo_reset1: assert property (@(posedge reset) (empty && !full && (wptr == 0) && (rptr == 0) &&
(state == 0) && valid_out == 0));

// reset: fifo -> !valid  && !ready
fifo_reset2: assert property (@(posedge reset) (!valid && !ready));

// verify the full
FIFO_full: assert property (@(posedge clk) $past(valid && wptr == (rptr - 1)) |-> full && !empty);

// verify the empty
FIFO_empty: assert property (@(posedge clk) $past(valid && (wptr == rptr) && reset_done) |-> !full &&
empty);
```

```
//cover wptr add
FIFO_wptr: cover property (@(posedge clk)
    (wptr != 7) |-> (wptr == $past(wptr) + 1));

//cover rptr add
FIFO_rptr: cover property (@(posedge clk)
    (rptr != 7) |-> (rptr == $past(rptr) + 1));

//fifo full -> no change
FIFI_write_full: assert property (@(posedge clk) full && valid |-> (wptr == $past(wptr)));

// fifo empty -> no change
FIFO_read_emtpy: assert property (@(posedge clk) empty && ready |-> (rptr == $past(rptr)));

// fifo wptr_add
FIFO_wptr_add: assert property (@(posedge clk) $past(reset, 2) && valid && !empty && !full |-> (wptr
== $past(wptr) + 1));
```

Assertions and Covers



Design Hierarchy:
- ✓ top (top)
  - ✓ f_in (fifo)
  - alu1 (alu)
  - ✓ f_out (fifo:(memory_width=8))

Property Table:

	Type	Name
✓	Assert	top.f_in.fifo_reset1
✓	Assert	top.f_in.fifo_reset2
✓	Assert	top.f_in.FIFO_full
✓	Cover (related)	top.f_in.FIFO_full:precondition1
✓	Assert	top.f_in.FIFO_empty

Total: 28 | Filtered: 28 | Selected: 0 | Validity: 28:0: | Run: 0:0:0

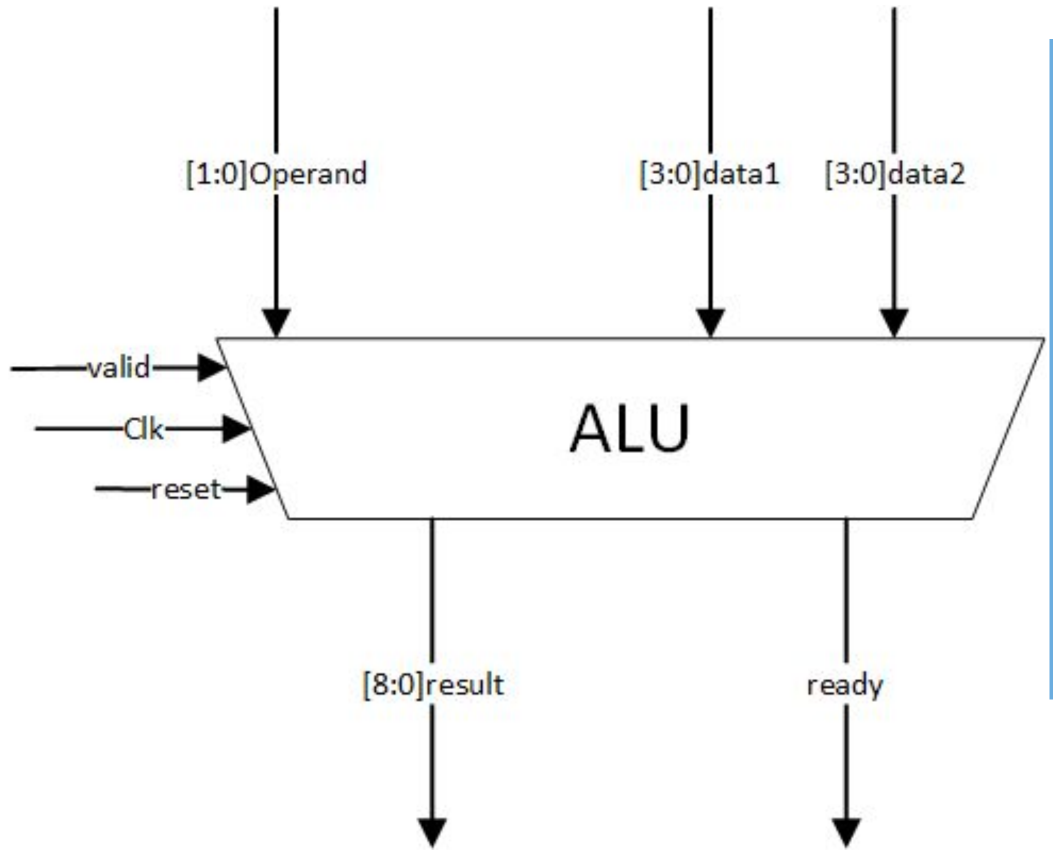
```
- undetermined          : 0 (0%)
- unknown               : 0 (0%)
- error                 : 0 (0%)
```

Screenshot of Results

Columbia Engineering

ALU

logic function



ALU Datapath

ALU Datapath Description

Operand : 2 bits Control Signal

ready : Output Enable Signal

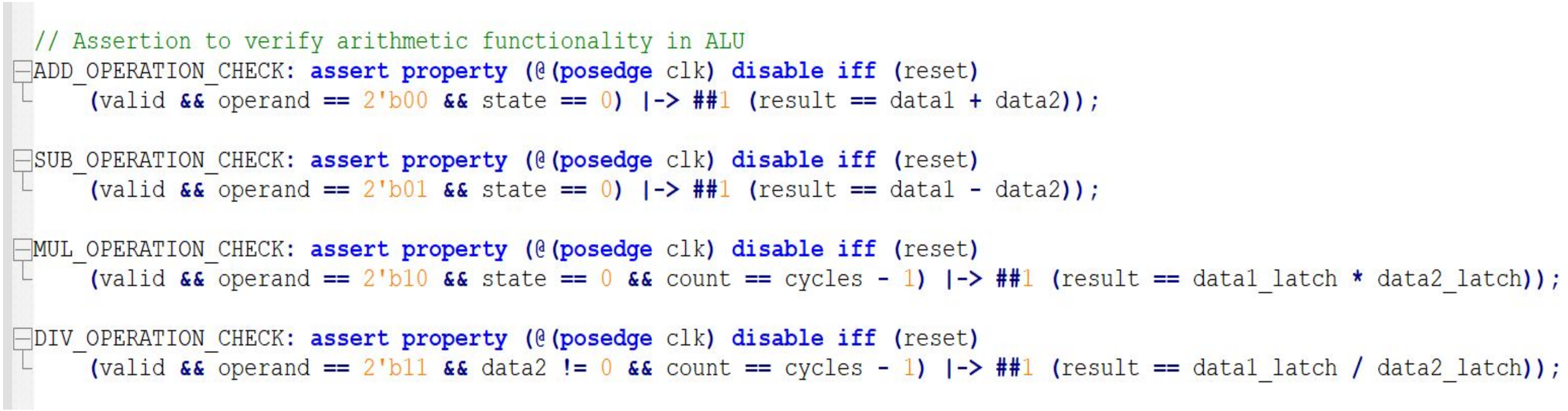
[8:0] result : 8 bits Arithmetic Result

[3:0] data1, [3:0] data2 : Two 4 bits Inputs

Clk : Clock Signal

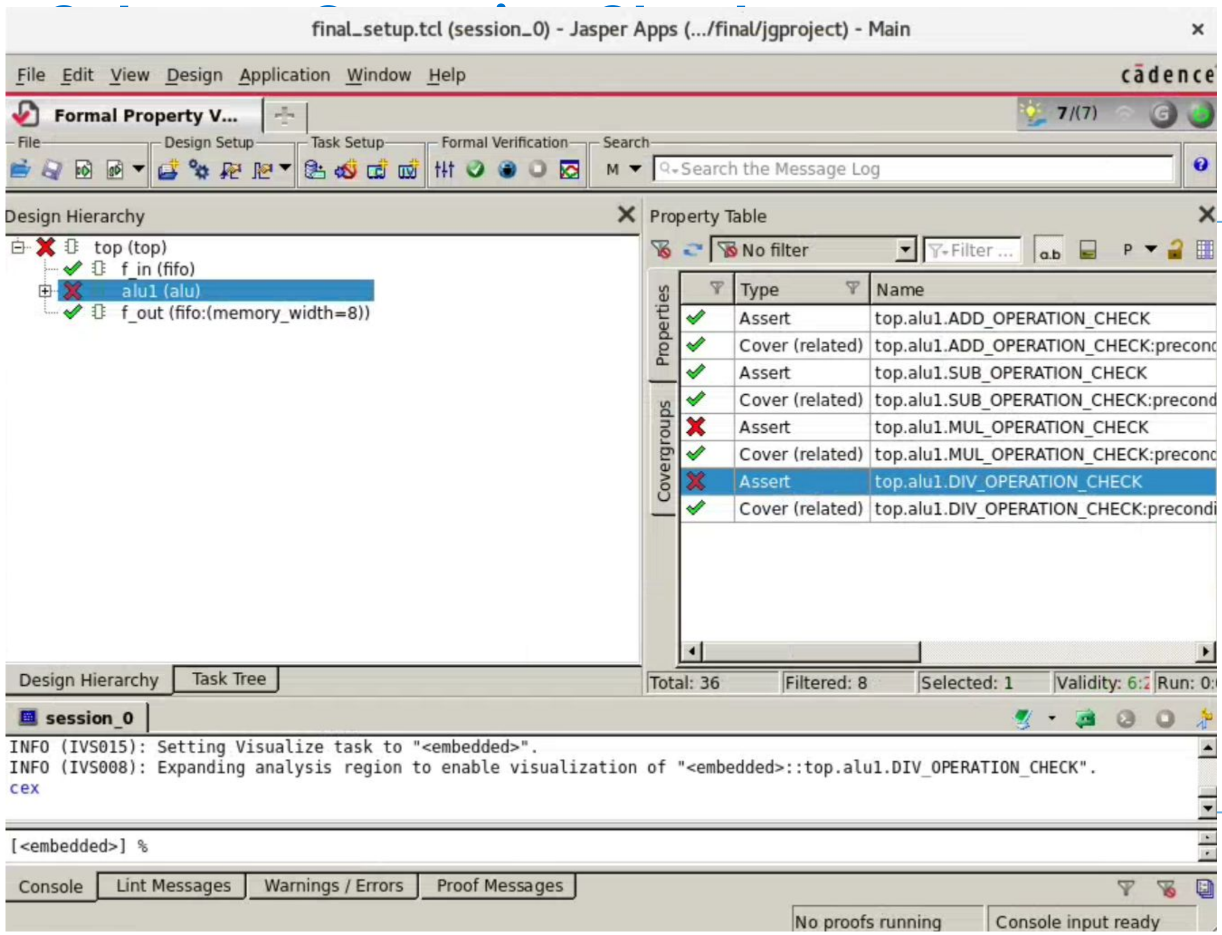
reset : Reset Signal

ALU Verification



```
// Assertion to verify arithmetic functionality in ALU
ADD_OPERATION_CHECK: assert property (@(posedge clk) disable iff (reset)
      (valid && operand == 2'b00 && state == 0) |-> ##1 (result == data1 + data2));

SUB_OPERATION_CHECK: assert property (@(posedge clk) disable iff (reset)
      (valid && operand == 2'b01 && state == 0) |-> ##1 (result == data1 - data2));

MUL_OPERATION_CHECK: assert property (@(posedge clk) disable iff (reset)
      (valid && operand == 2'b10 && state == 0 && count == cycles - 1) |-> ##1 (result == data1_latch * data2_latch));

DIV_OPERATION_CHECK: assert property (@(posedge clk) disable iff (reset)
      (valid && operand == 2'b11 && data2 != 0 && count == cycles - 1) |-> ##1 (result == data1_latch / data2_latch));
```

ALU Verification



	Type	Name
✓	Assert	top.alu1.ADD_OPERATION_CHECK
✓	Cover (related)	top.alu1.ADD_OPERATION_CHECK:precond
✓	Assert	top.alu1.SUB_OPERATION_CHECK
✓	Cover (related)	top.alu1.SUB_OPERATION_CHECK:precond
✗	Assert	top.alu1.MUL_OPERATION_CHECK
✓	Cover (related)	top.alu1.MUL_OPERATION_CHECK:precond
✗	Assert	top.alu1.DIV_OPERATION_CHECK
✓	Cover (related)	top.alu1.DIV_OPERATION_CHECK:precondi

Total: 36 | Filtered: 8 | Selected: 1 | Validity: 6:2 | Run: 0:

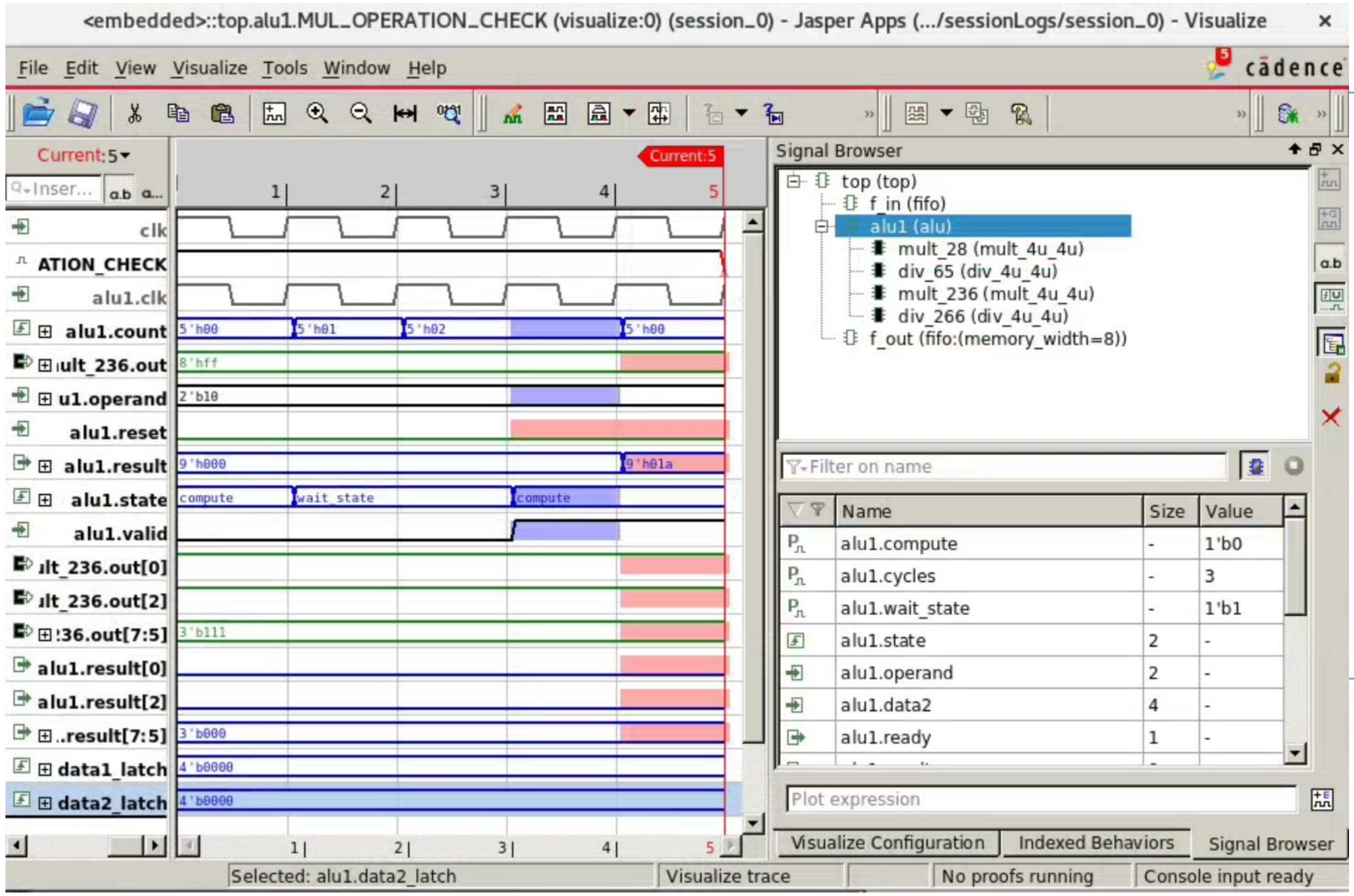
```
INFO (IVS015): Setting Visualize task to "<embedded>".
INFO (IVS008): Expanding analysis region to enable visualization of "<embedded>::top.alu1.DIV_OPERATION_CHECK".
cex

[<embedded>] %
```

Addition and Subtraction √

Multiplication and Division ×

ALU Verification

Multiplication Operation Check



<embedded>::top.alu1.MUL_OPERATION_CHECK (visualize:0) (session_0) - Jasper Apps (.../sessionLogs/session_0) - Visualize

Name	Size	Value
alu1.compute	-	1'b0
alu1.cycles	-	3
alu1.wait_state	-	1'b1
alu1.state	2	-
alu1.operand	2	-
alu1.data2	4	-
alu1.ready	1	-

Selected: alu1.data2_latch | Visualize trace | No proofs running | Console input ready

Wrong arithmetic result

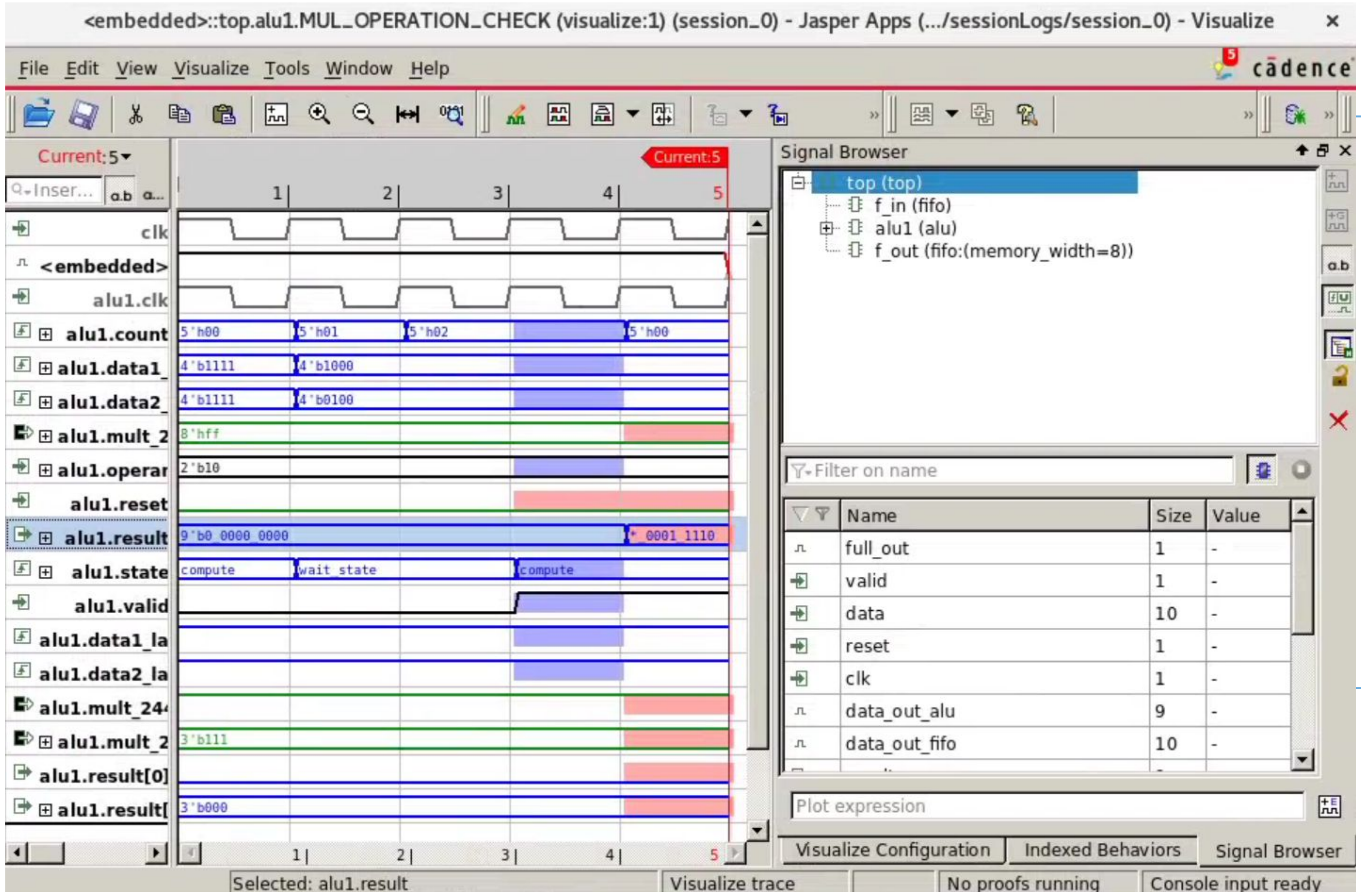
data1 : 4’b0000

data2: 4’b0000

result: 9’h01a

ALU Verification

Multiplication Operation Check



<embedded>::top.alu1.MUL_OPERATION_CHECK (visualize:1) (session_0) - Jasper Apps (.../sessionLogs/session_0) - Visualize

Name	Size	Value
full_out	1	-
valid	1	-
data	10	-
reset	1	-
clk	1	-
data_out_alu	9	-
data_out_fifo	10	-

Selected: alu1.result | Visualize trace | No proofs running | Console input ready

Wrong arithmetic result

data1 : 4’b0100

data2: 4’b0100

result: 9’b 0 0001 1110

Thank You