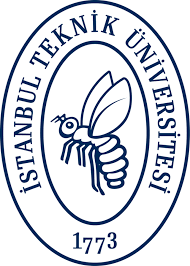
****

**VLSI Circuit Design II– EHB 425E**

**HOMEWORK VII**

**Yiğit Bektaş GÜRSOY**

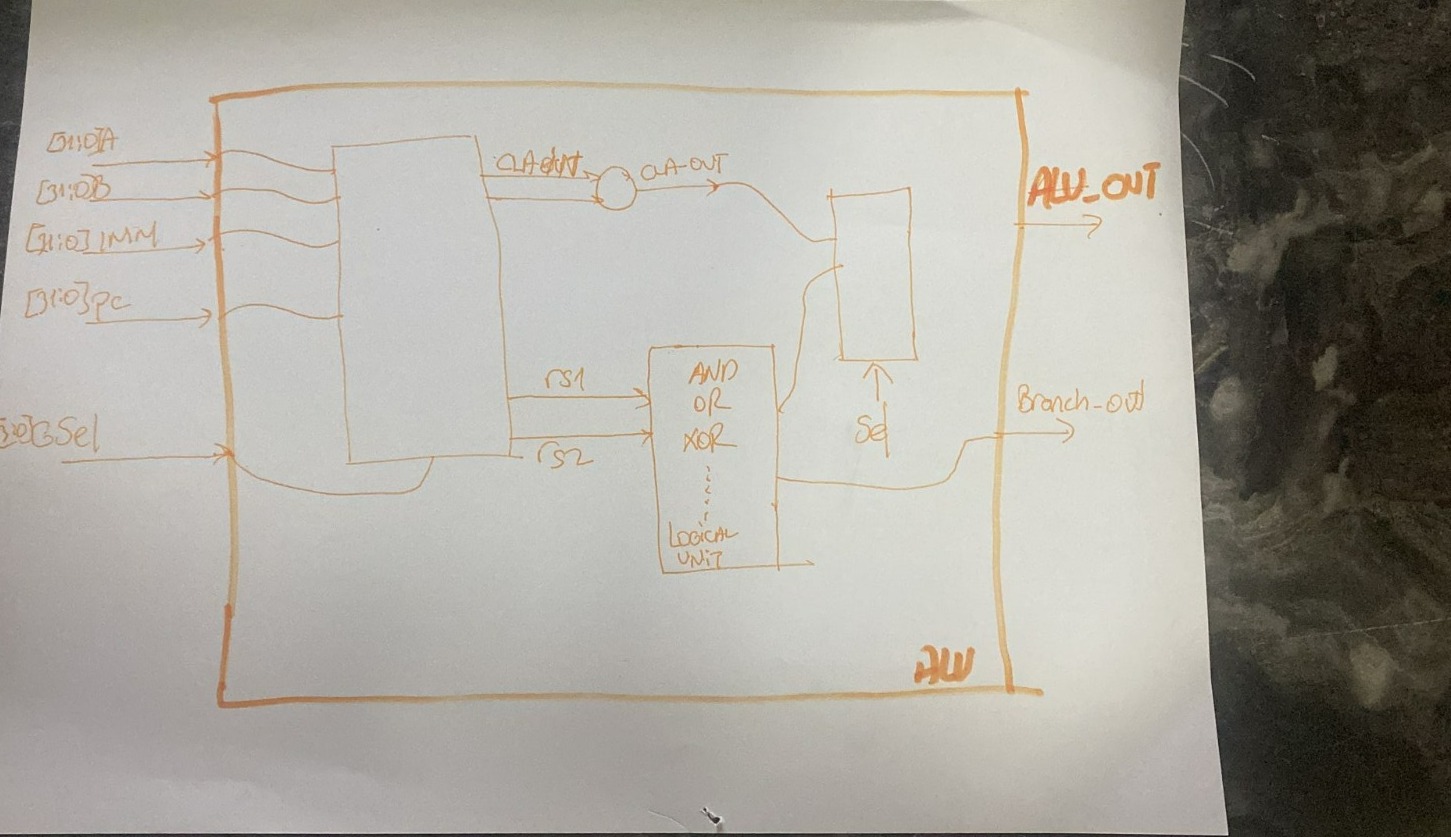
**040180063**

**Rana TİLKİ**

**040180741**

**Class Lecturer: Sıddıka Berna Örs Yalçın**

**Class Assistant:  
Yasin Fırat Kula**

1. **ALU Design**
2. 
3. Selection Signals
4. GSel (4 bits) [3:0]: This is the main selection signal used to control the operation of the ALU. The bits of GSel serve different purposes:

GSel[0]: This bit is used to select between using the B input or the immediate value (imm) as the second operand for the ALU operation.

* If GSel[0] is 1, the immediate value is used as the second operand.
* If GSel[0] is 0, the B input is used as the second operand.

GSel[3:1]: These 3 bits are used to select the specific operation to be performed by the ALU. The following cases are defined:

* 3'b000: AND operation
* 3'b001: OR operation
* 3'b010: XOR operation
* 3'b011: ADD/SUB operation
* 3'b100: Unsigned comparison and branch operations, or jump operations (JAL and JALR)
* 3'b101: Signed comparison and branch operations

1. branch\_in (3 bits) [2:0]: This signal is used when GSel[3:1] is set to 3'b100 (unsigned branch) or 3'b101 (signed branch). It is responsible for selecting a specific branch operation based on its value:

* 3'b000: Branch if less than (BLT)
* 3'b001: Branch if greater than or equal (BGE)
* 3'b010: Branch if equal (BEQ)
* 3'b011: Branch if not equal (BNE)
* 3'b100: Set less than (SLT)

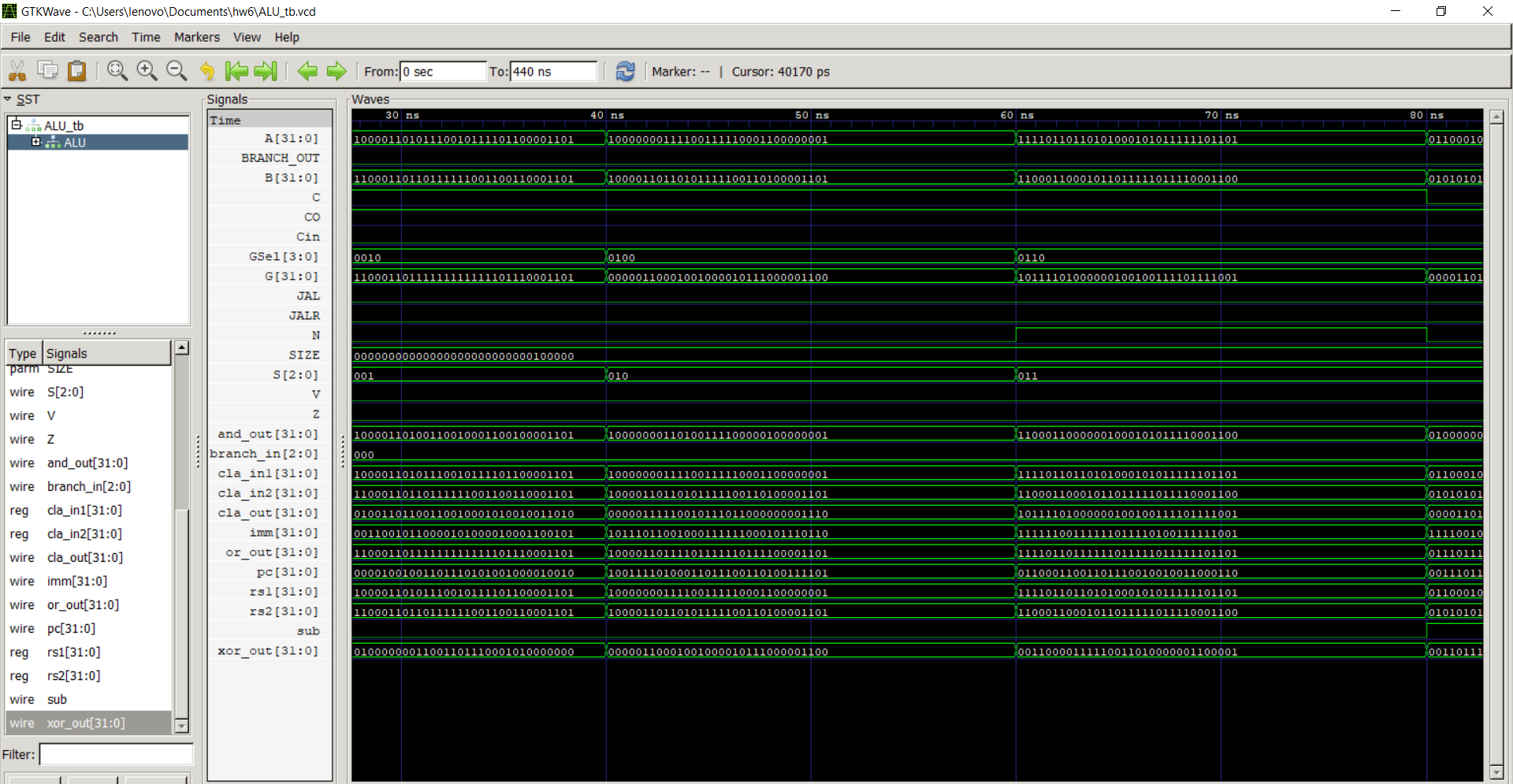
1. JAL: This signal indicates a Jump and Link operation, used for branching to a new location and saving the return address.
2. JALR: This signal indicates a Jump and Link Register operation, used for branching to a new location based on the value in a register and saving the return address.
3. sub: This signal is used to indicate whether the ALU should perform an addition or a subtraction operation when GSel[3:1] is set to 3'b011 (ADD/SUB operation). If 'sub' is 1, the ALU performs subtraction; if 'sub' is 0, the ALU performs addition.

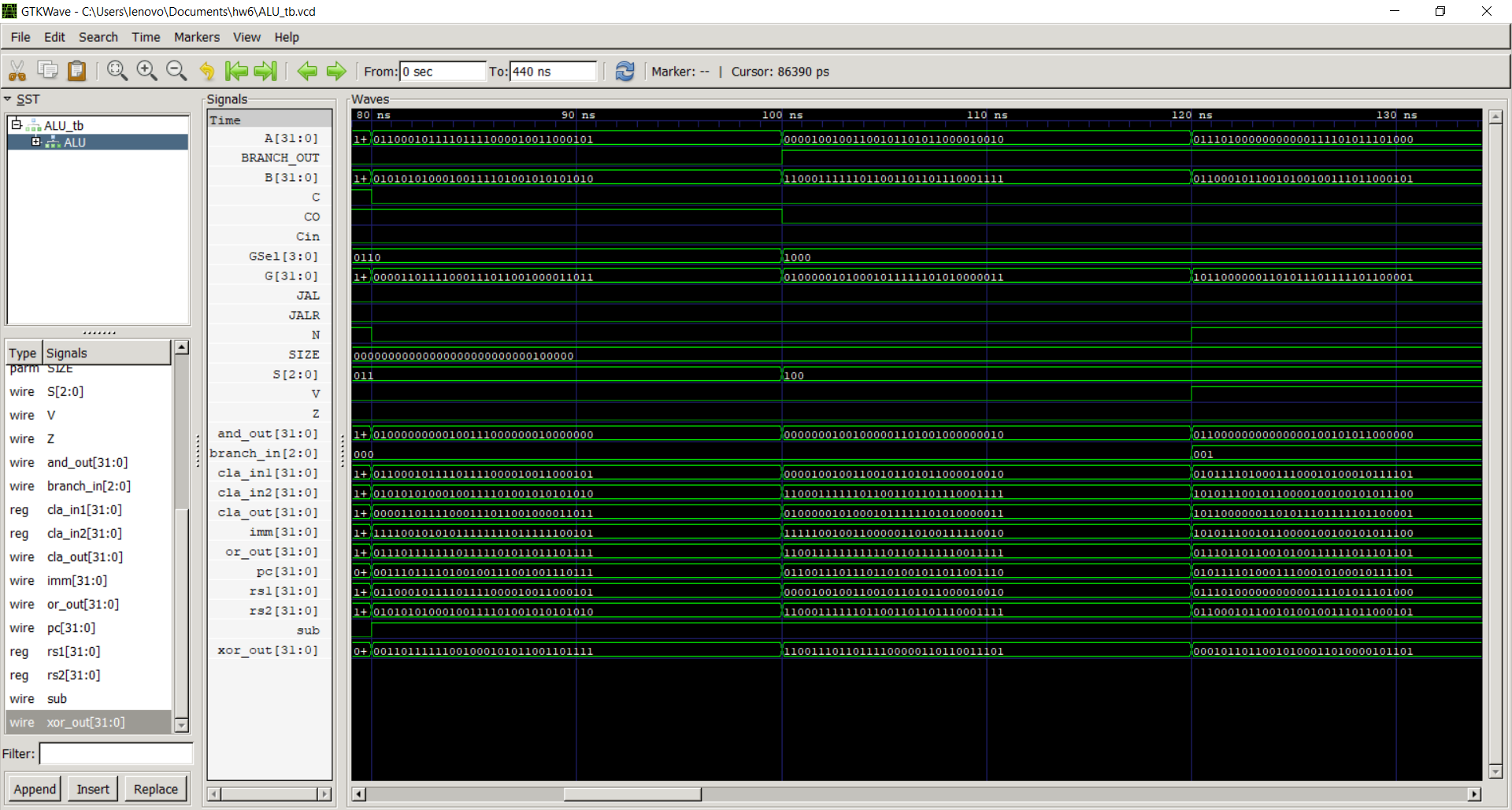
These selection signals are essential for controlling the functionality of the ALU, allowing it to perform a wide range of arithmetic, logic, and branching operations based on the input operands and the desired operation specified by the control signals.

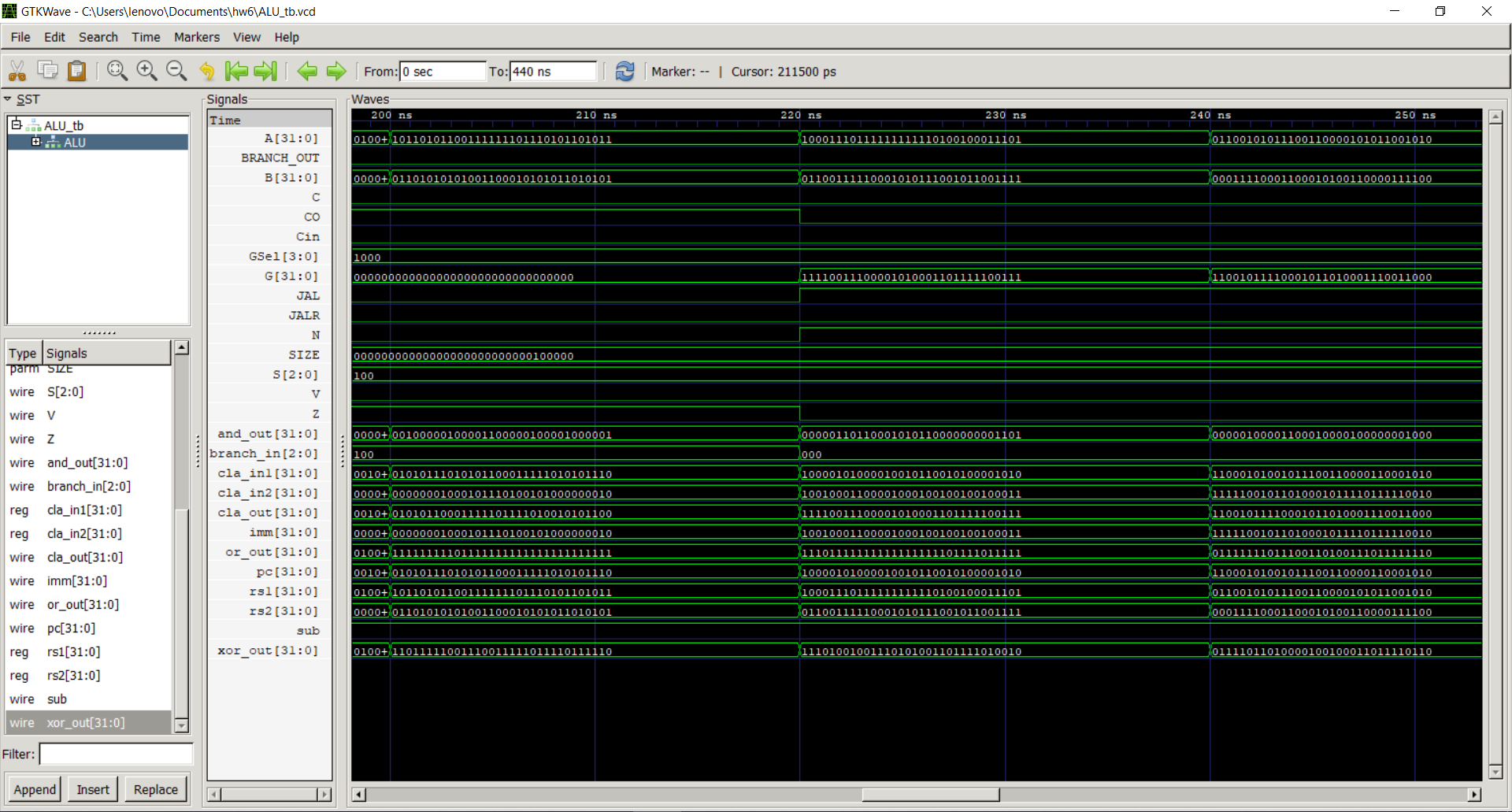
1. Status Signals
2. CO (Carry Out): This signal represents the carry-out generated during an addition or subtraction operation. It is set when there is a carry or borrow generated in the most significant bit (MSB) of the result. In unsigned arithmetic, this signal can be used to detect overflow.
3. V (Overflow): This signal indicates when an overflow occurs during a signed arithmetic operation (addition or subtraction). An overflow occurs when the result of the operation is too large (positive overflow) or too small (negative overflow) to be represented within the given bit-width (32 bits in this case).
4. C (Carry): This signal represents the carry generated during an addition or subtraction operation. It is set when there is a carry or borrow generated in the most significant bit (MSB) of the result. In unsigned arithmetic, this signal can be used to detect overflow. Note that the 'C' signal is functionally similar to the 'CO' signal, but is used in different contexts within the module.
5. N (Negative): This signal is set when the result of the ALU operation is negative, i.e., when the most significant bit (MSB) of the result is 1. It helps to determine the sign of the result for signed operations.
6. Z (Zero): This signal is set when the result of the ALU operation is zero. The ZERO\_DETECT module checks if all bits of the ALU output (G) are zero, and if so, it sets the 'Z' signal. This signal is useful for conditional branch instructions, such as BEQ (Branch if Equal) and BNE (Branch if Not Equal).

These status signals provide essential information about the outcome of the ALU operation, helping the control unit make decisions based on the result. They are used in various instructions like conditional branches, comparisons, and arithmetic operations to determine the flow of the program or to set the necessary flags in the processor's status register.

1. Purposes Sub-Blocks
2. AND: This sub-block performs a bitwise AND operation on the input operands (rs1 and rs2). It takes two inputs and produces an output where each bit is the result of a bitwise AND between the corresponding bits of the input operands. The AND operation is selected when GSel[3:1] is set to 3'b000.
3. OR: This sub-block performs a bitwise OR operation on the input operands (rs1 and rs2). It takes two inputs and produces an output where each bit is the result of a bitwise OR between the corresponding bits of the input operands. The OR operation is selected when GSel[3:1] is set to 3'b001.
4. XOR: This sub-block performs a bitwise XOR (exclusive OR) operation on the input operands (rs1 and rs2). It takes two inputs and produces an output where each bit is the result of a bitwise XOR between the corresponding bits of the input operands. The XOR operation is selected when GSel[3:1] is set to 3'b010.
5. CLA (Carry Lookahead Adder): This sub-block performs addition or subtraction on the input operands (cla\_in1 and cla\_in2) based on the 'sub' signal. The carry lookahead adder is an efficient implementation of an adder that can quickly propagate carry bits through the adder, reducing the overall delay. When GSel[3:1] is set to 3'b011, the CLA is used for ADD/SUB operations. In addition, the CLA is involved in executing branch and jump instructions (when GSel[3:1] is set to 3'b100 or 3'b101) as it calculates the new program counter (PC) value. The CLA sub-block also generates the status signals CO, V, C, and N.
6. ZERO\_DETECT: This sub-block checks if the result of the ALU operation (G) is zero. If all bits of the ALU output are zero, the 'Z' (Zero) status signal is set. The ZERO\_DETECT sub-block is involved in the execution of conditional branch instructions like BEQ (Branch if Equal) and BNE (Branch if Not Equal), as well as other instructions where the zero flag needs to be updated.
7. **ALU**
8. Behavioral Simulation Results

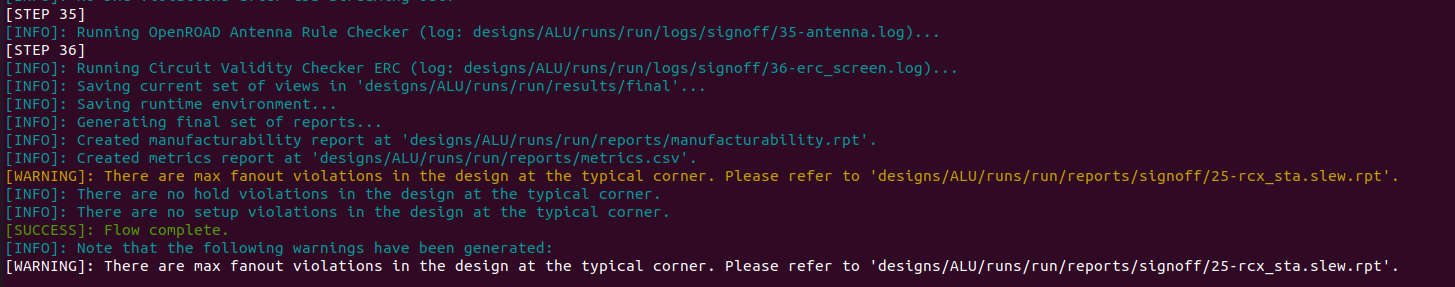




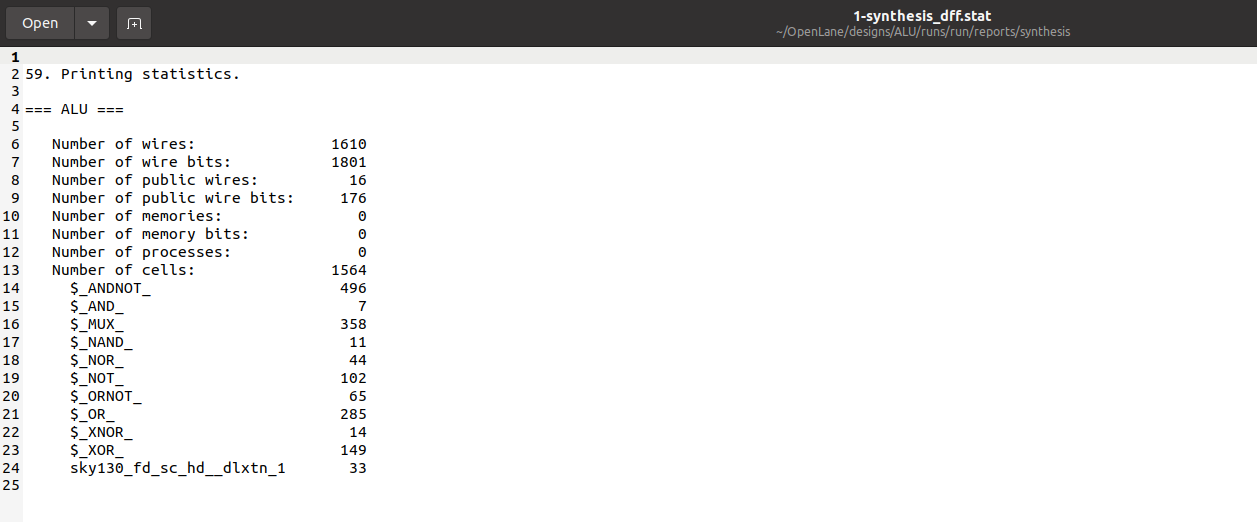


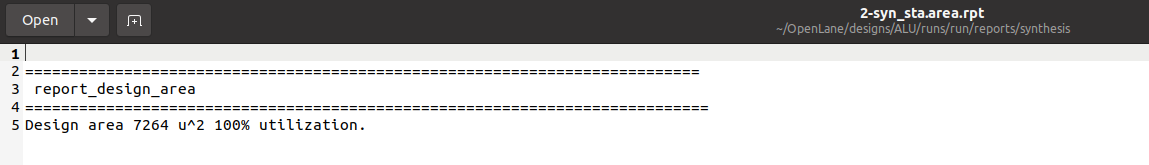
ALU block in testbench respectively AND, OR , XOR, ADDER, SUBSTRACTOR, branch A<B, branch A>=B, branch A=B, branch A!=B, G = (rs1 < rs2) ? 32'd1 : 32'd0, branch rs1<rs2 , branch rs1<rs2, JAL||JALR , branch, $signed(A) < $signed(B), $signed(A) >= $signed(B), G= ($signed(rs1) < $signed(rs2)) ? 32'd1 : It returns 32'd0. Looking at the graph and checking TCL from the console, it seems that all transactions are working correctly.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Constant\_in** | **Branch\_in** | **JAL** | **JALR** | **GSel** | **Function** | **Operation** |
| 0 | 00 | 0 | 0 | 0000 | ADD | A+B |
| 0 | 00 | 0 | 0 | 1000 | SUB | A-B |
| 1 | 00 | 0 | 0 | 0000 | ADDI | A+IMM |
| 0 | 00 | 0 | 0 | 0001 | XOR | A^B |
| 1 | 00 | 0 | 0 | 0001 | XORI | A^IMM |
| 0 | 00 | 0 | 0 | 0010 | OR | A|B |
| 1 | 00 | 0 | 0 | 0010 | ORI | A|IMM |
| 0 | 00 | 0 | 0 | 0011 | AND | A&B |
| 1 | 00 | 0 | 0 | 0011 | ANDI | A&IMM |
| 0 | 00 | 0 | 0 | 0101 | SLT | IF A<B, SET 1 |
| 1 | 00 | 0 | 0 | 0101 | SLTI | IF A<IMM, SET1 |
| 0 | 00 | 0 | 0 | 0100 | SLTU | IF A<B(UNSIGNED),  SET 1 |
| 1 | 00 | 0 | 0 | 0100 | SLTIU | IF A<IMM(UNSIGNED),  SET 1 |
| 0 | 01 | 0 | 0 | 0000 | BEQ | IF A == B, PC += IMM |
| 0 | 10 | 0 | 0 | 0000 | BNE | IF A != B, PC += IMM |
| 0 | 01 | 0 | 0 | 0101 | BLT | IF A < B, PC += IMM |
| 0 | 01 | 0 | 0 | 0100 | BLTU | IF A < B (UNSIGNED), PC += IMM |
| 0 | 10 | 0 | 0 | 0101 | BGE | IF A ≥ B, PC += IMM |
| 0 | 10 | 0 | 0 | 0100 | BGEU | IF A ≥ B (UNSIGNED), PC += IMM |
| 0 | 00 | 1 | 0 | 0000 | JAL | JUMP, PC += IMM |
| 0 | 00 | 0 | 1 | 0000 | JALR | JUMP, PC += IMM |

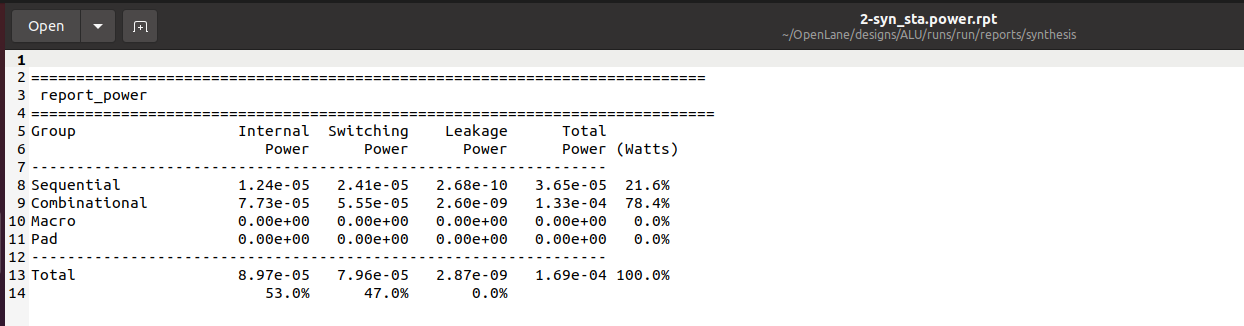
* Openlane Results

Cell Usage & Estimated Area

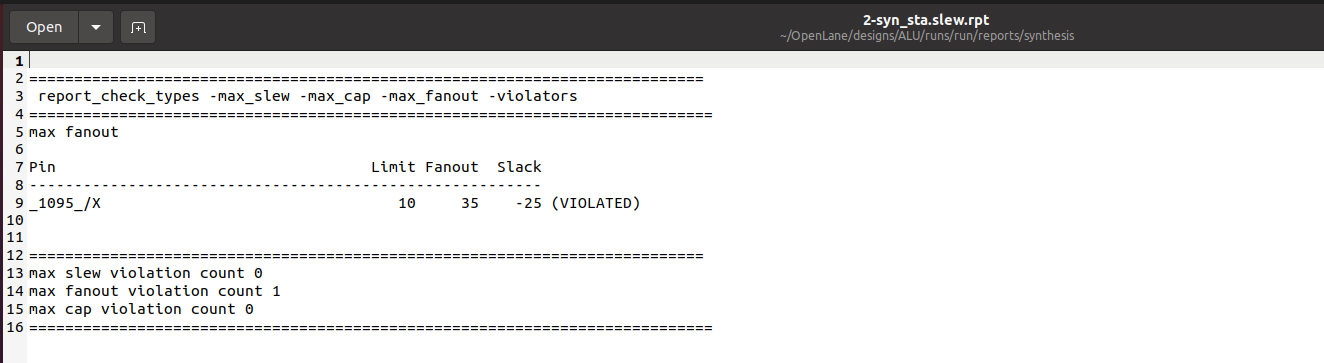


Estimated Area

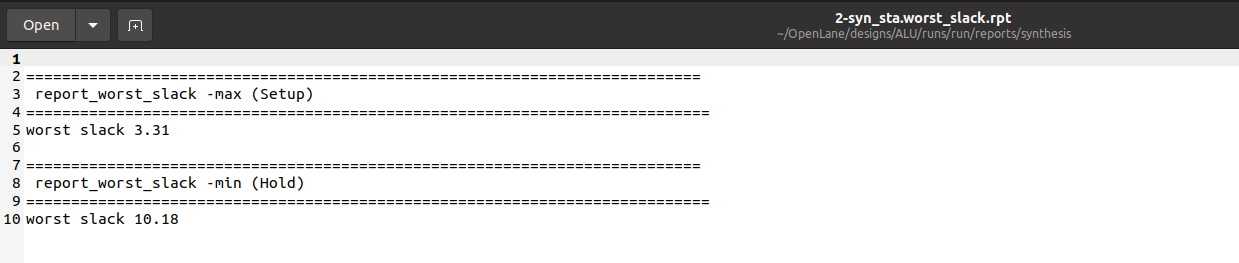
Power Consumption

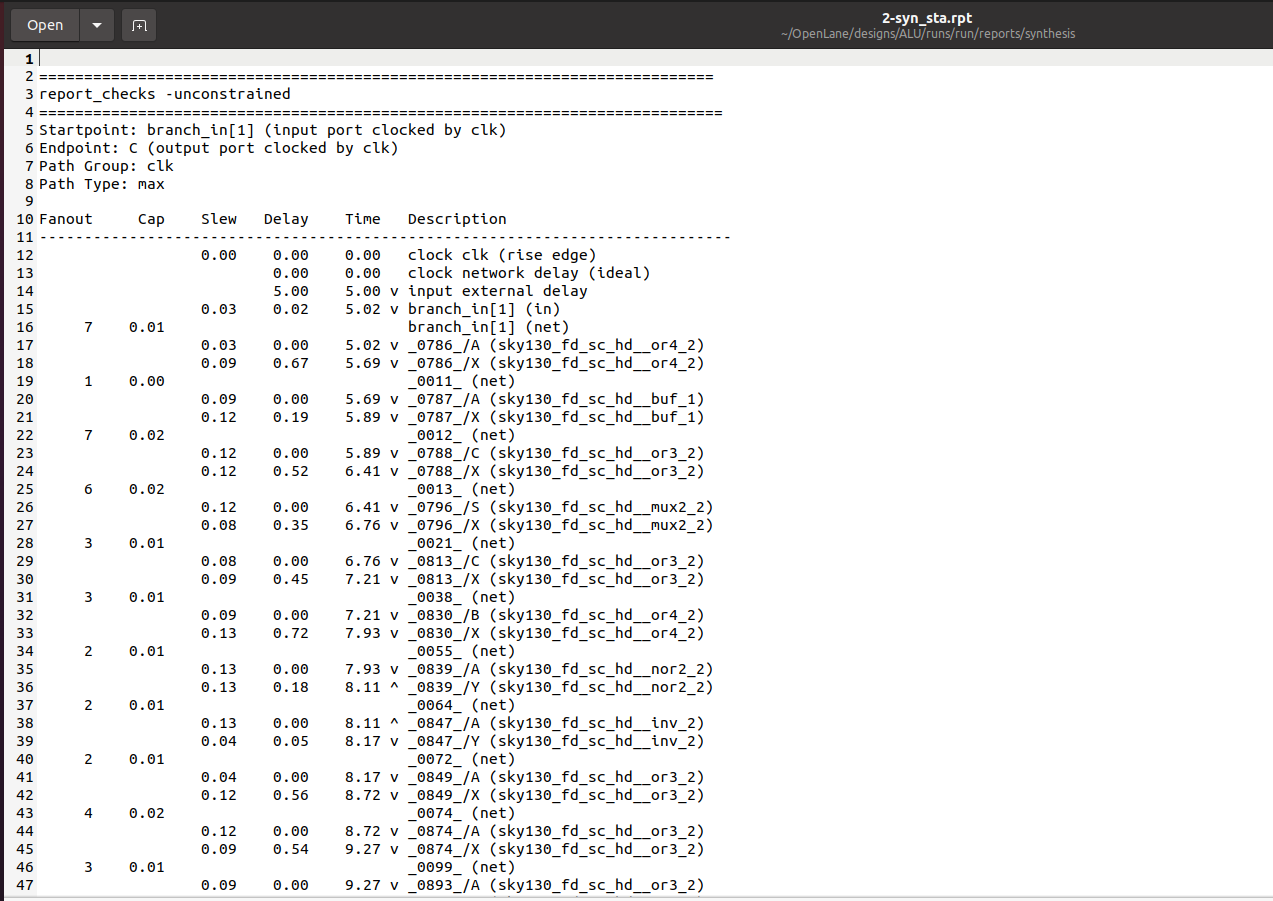


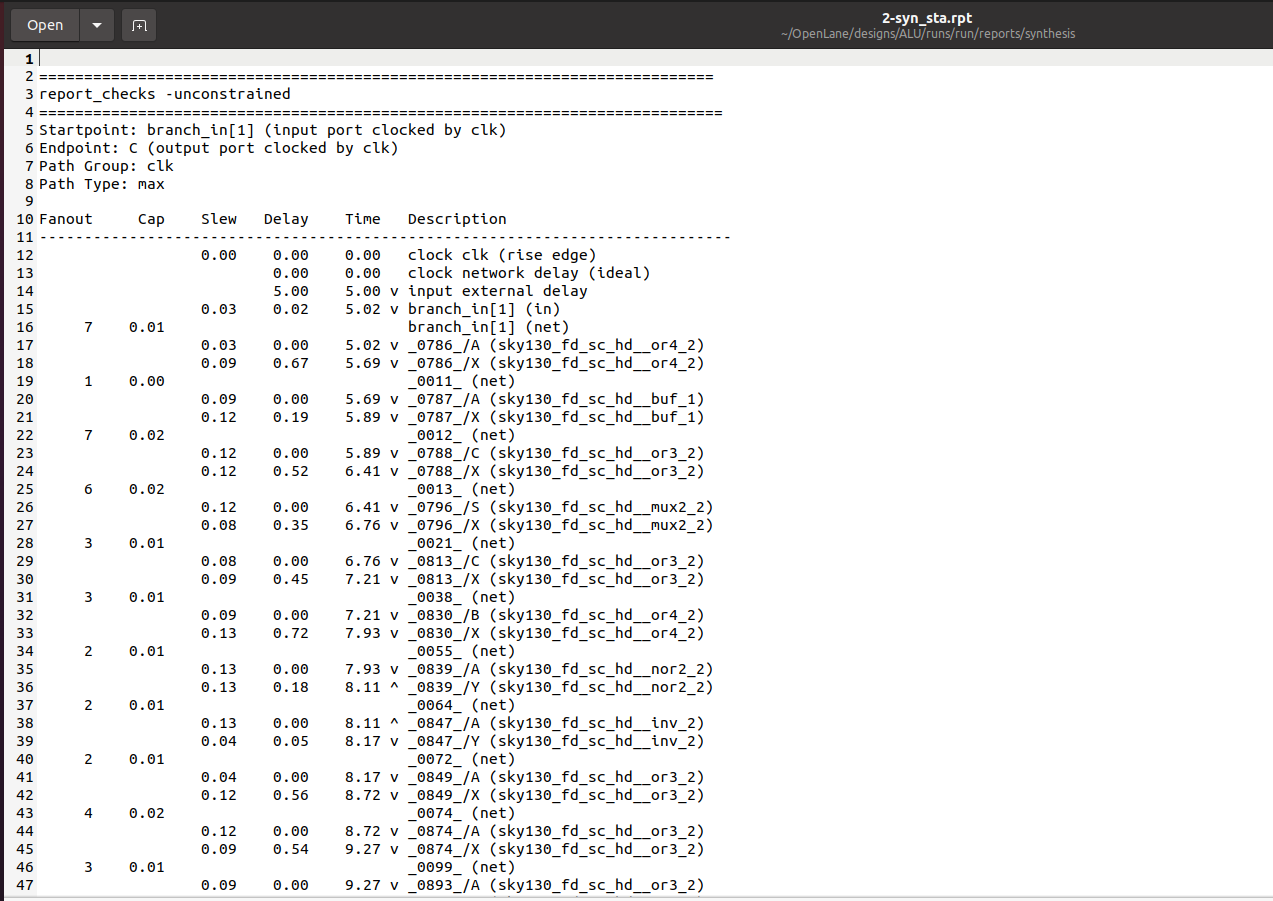
Fanout Report



Clock Report



Critical Path Delay



1. **SHIFTER**

* Purposes of control signals and selection signals

1. input [SIZE-1:0] A: This input signal represents the first operand, which will be shifted by the specified amount.
2. input [SIZE-1:0] B: This input signal represents the second operand, which can be used as the shift amount in some cases.
3. input [4:0] shamt: This input signal, short for "shift amount," specifies the number of positions to shift the operand A. This signal is used when an immediate value is provided for the shift amount.
4. input [1:0] shift\_type: This input signal is a 2-bit control signal that selects the type of shift operation to be performed. The shift operations and their corresponding shift\_type values are as follows:

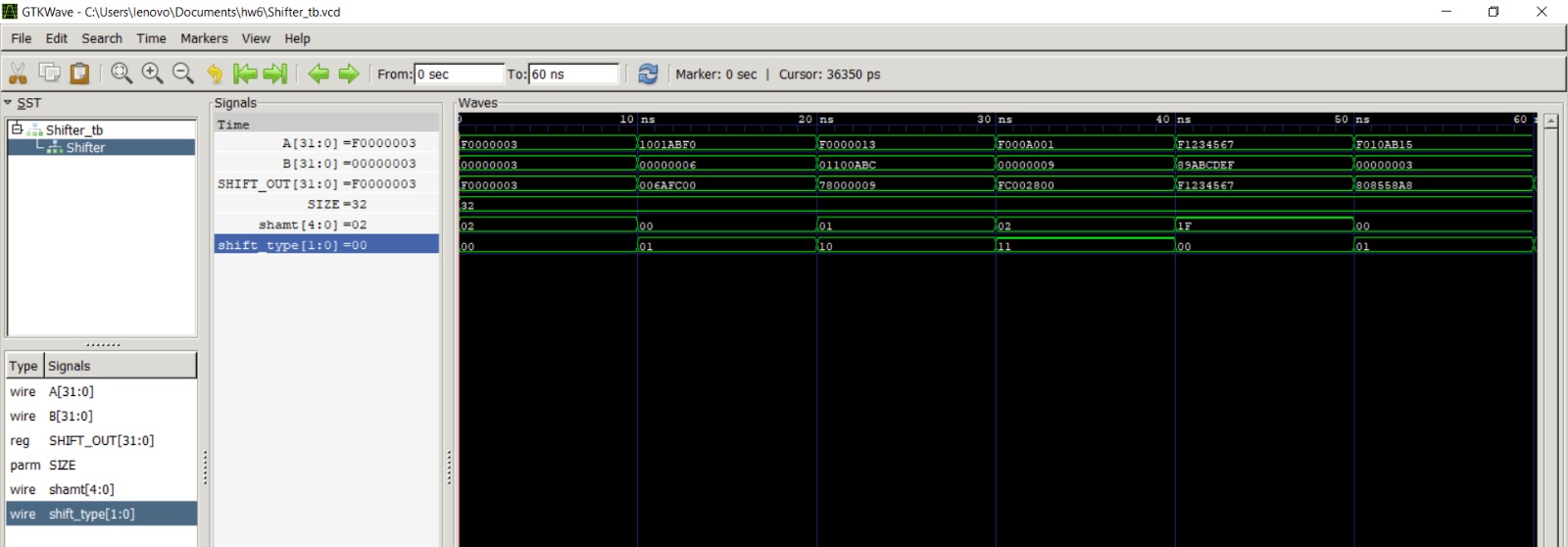
* 2'b00: No operation (default)
* 2'b01: Shift Left Logical (SLL) / Shift Left Logical Immediate (SLLI)
* 2'b10: Shift Right Logical (SRL) / Shift Right Logical Immediate (SRLI)
* 2'b11: Shift Right Arithmetic (SRA) / Shift Right Arithmetic Immediate (SRAI)

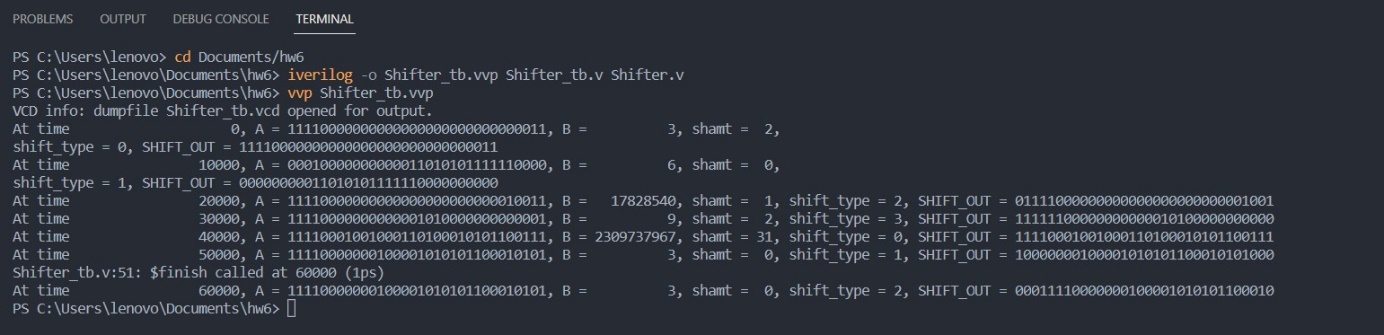
These control signals are used in the always block to determine the type of shift operation to perform on the input operands A and B. The shift\_type input signal is crucial in selecting the appropriate operation based on its value. The shamt input signal provides the shift amount, and depending on the value of shamt, the shift amount will either come from shamt or the lower 5 bits of input B.

|  |  |  |  |
| --- | --- | --- | --- |
| **SLL** | Shift Left Logical | rd 🡸 rs1 << rs2[4:0] | R-TYPE |
| **SLLI** | Shift Left Logical Immediate | rd 🡸 rs1 << shamt | I-TYPE |
| **SRL** | Shift Right Logical | rd 🡸 rs1 >> rs2[4:0] | R-TYPE |
| **SRLI** | Shift Right Logical Immediate | rd 🡸 rs1 >> shamt | I-TYPE |
| **SRA** | Shift Right Arithmetic | rd 🡸 rs1 >>> rs2[4:0] | R-TYPE |
| **SRAI** | Shift Right Arithme | rd 🡸 rs1 >>> shamt | I-TYPE |

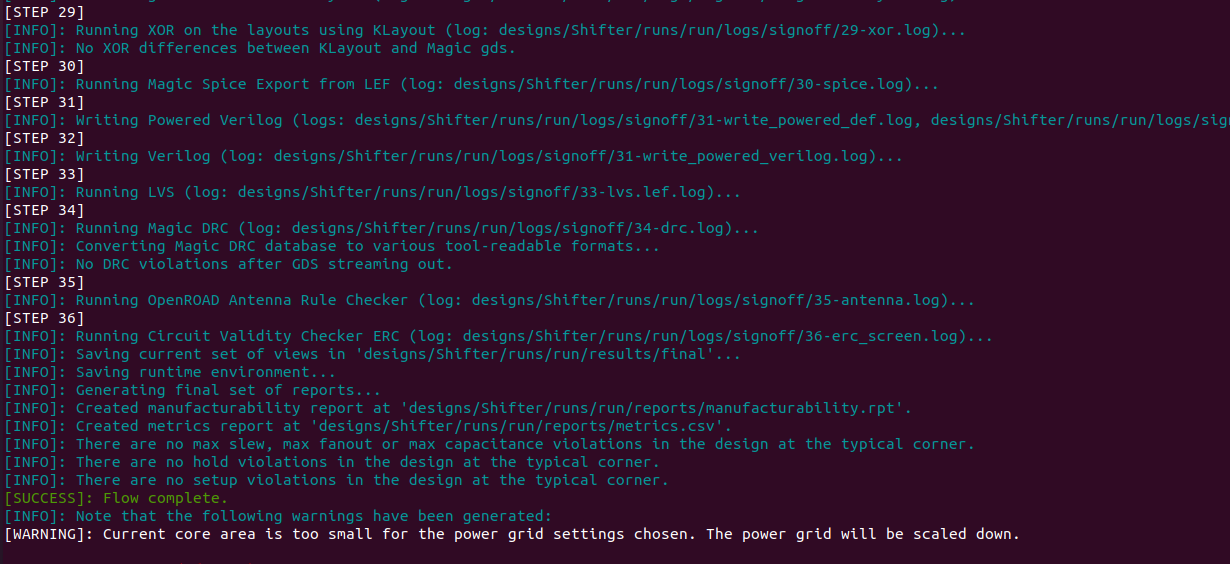
* Behavioral Simulation

As you can see, the shifter block is working successfully. First, shift\_type is selected, then if there is a shamt signal, that shift type is applied as much as the decimal value of the shamt, otherwise it is applied as the decimal value of the B value. Console output is given above. SLL shifts left, SRL shifts right, and SRA shifts arithmetically to the right, then fills in as much space as the most significant bit is.

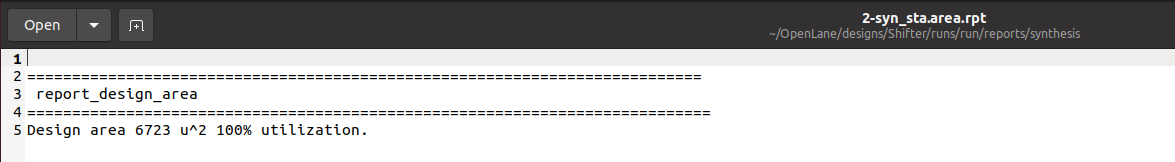


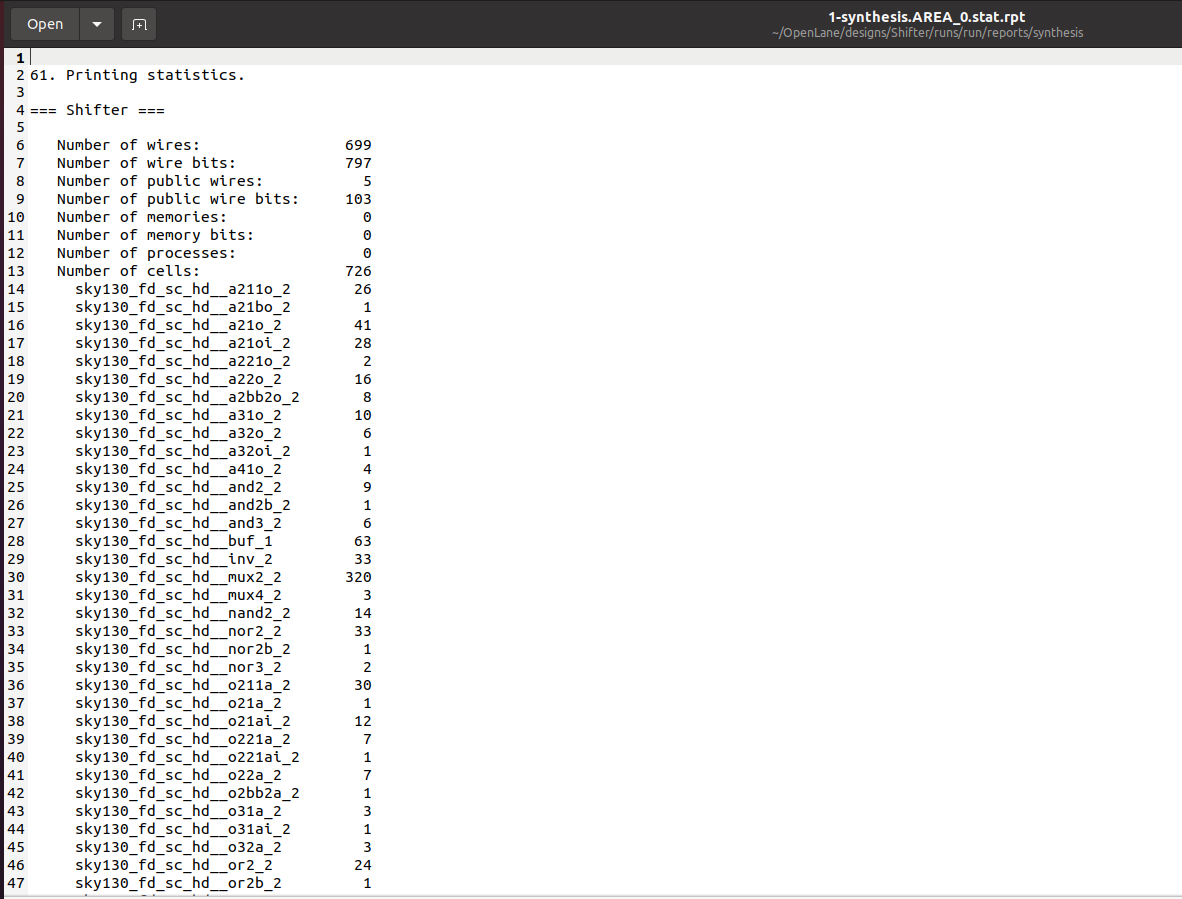


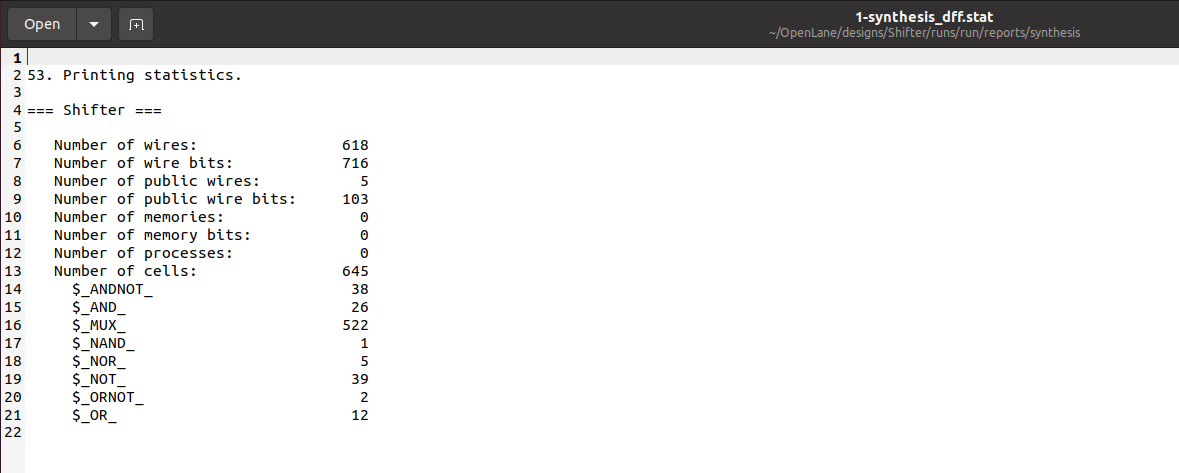
* Openlane results



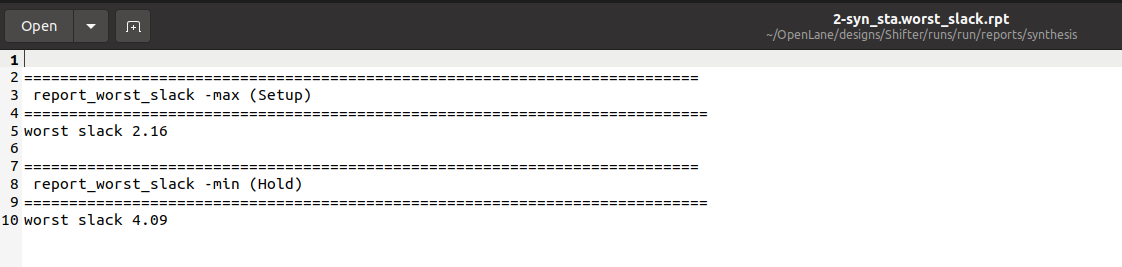
Estimated Area

Estimated Area

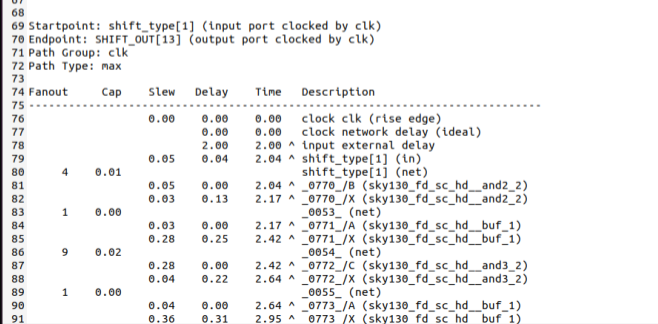
Cell Usage

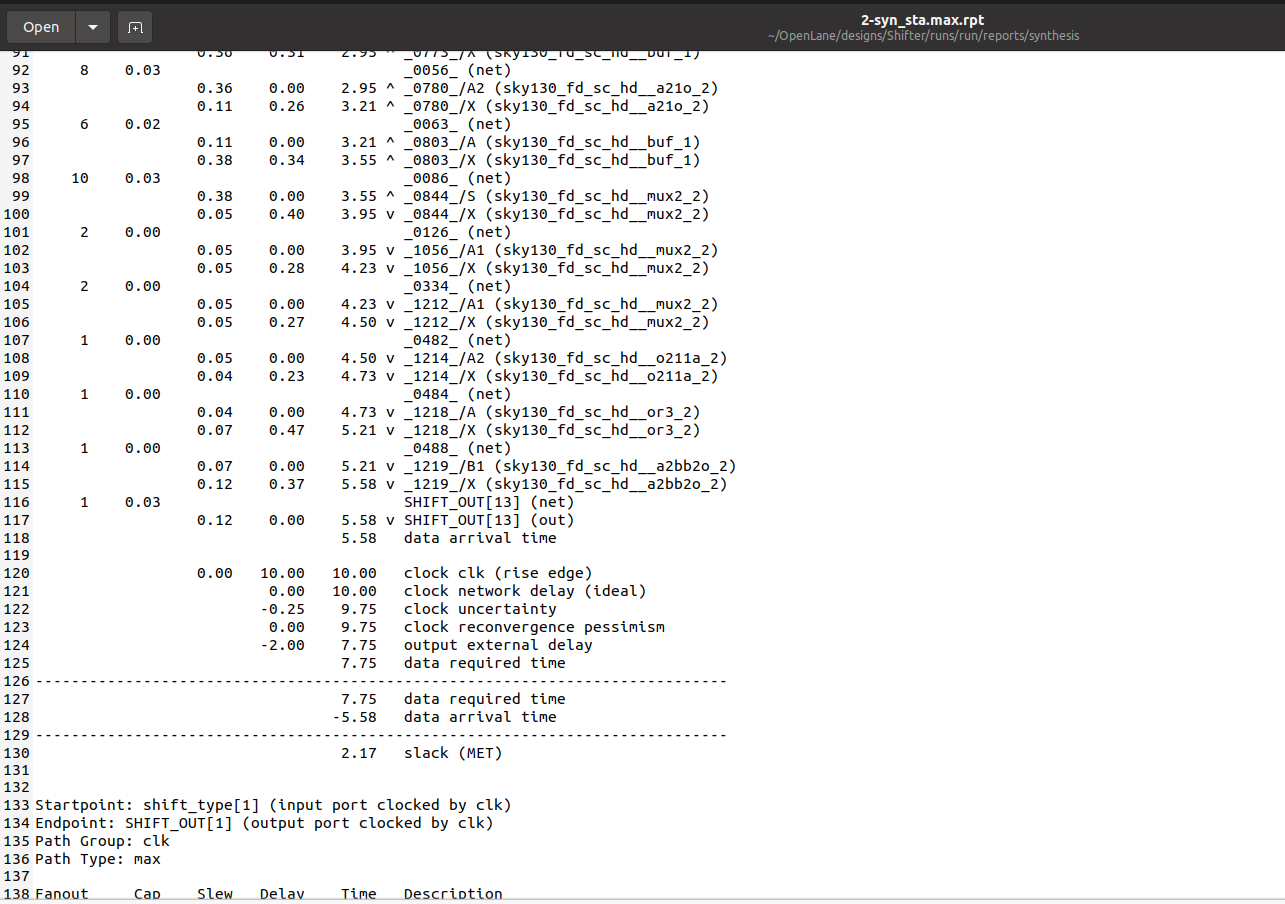


Clock Report

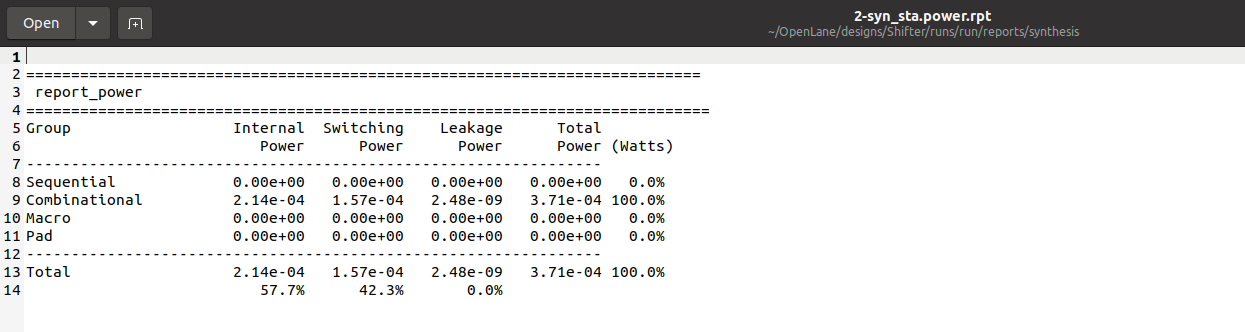


Critical Path





Power consumptioon



1. **FU**

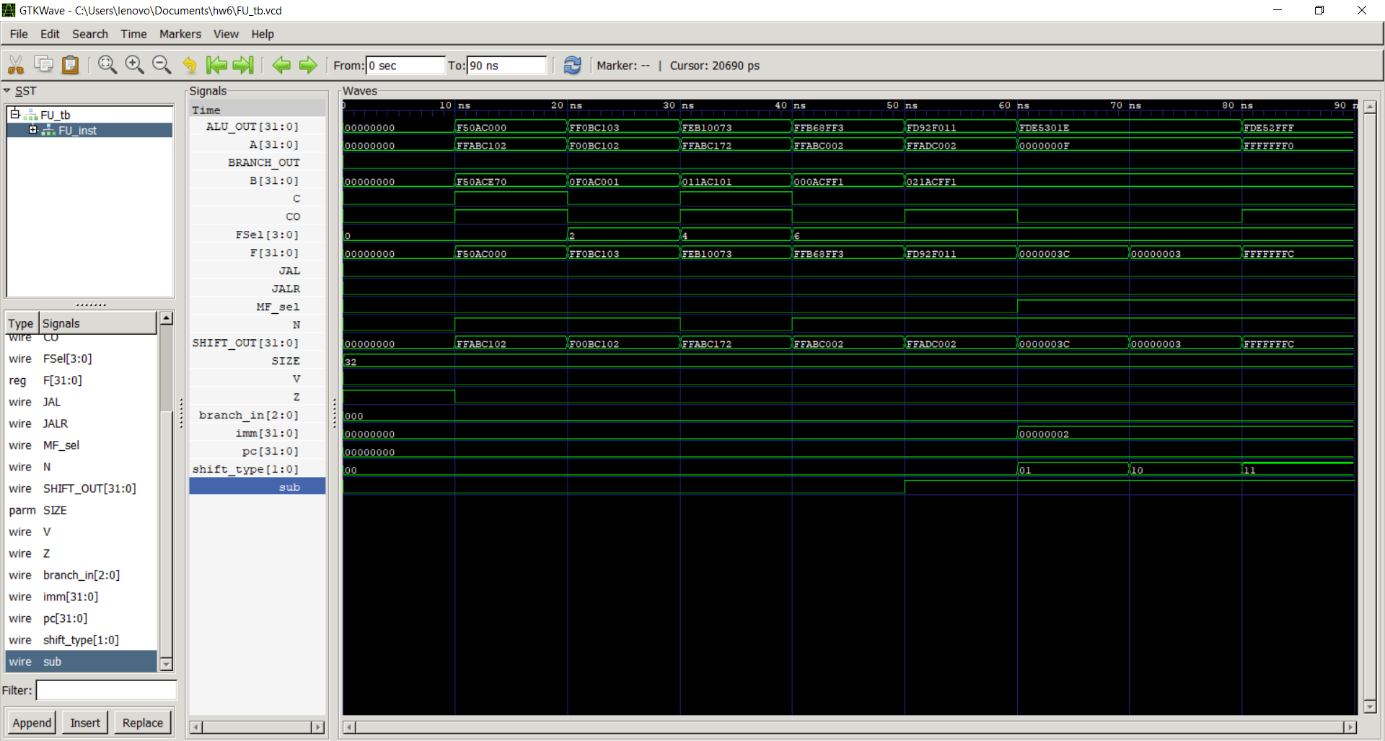
ALU module and shifter module are connected in this module. A and B inputs are similar, by defining one lot at the end of the house, it is aimed to leave the output of the ALU or to give the output of the shifter.

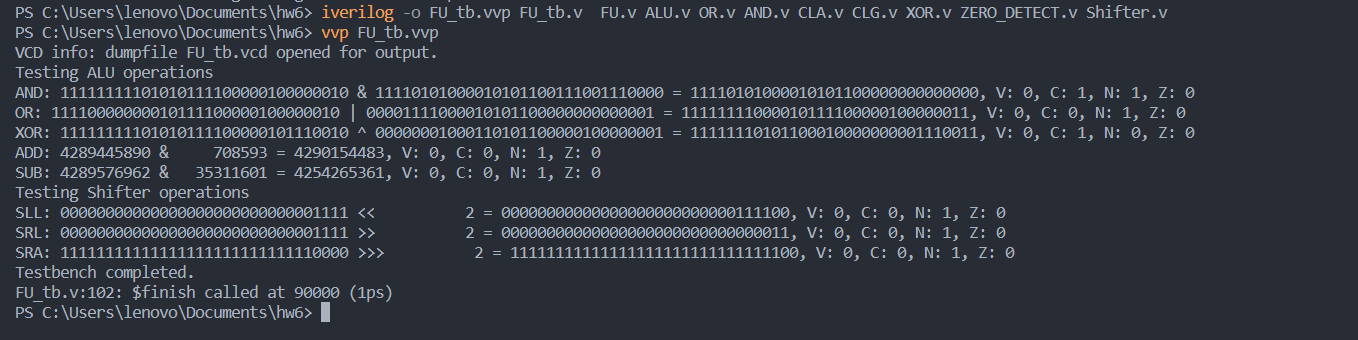
MUX 1'b0: ALU\_OUT

MUX 1'b1: SHIFT\_OUT

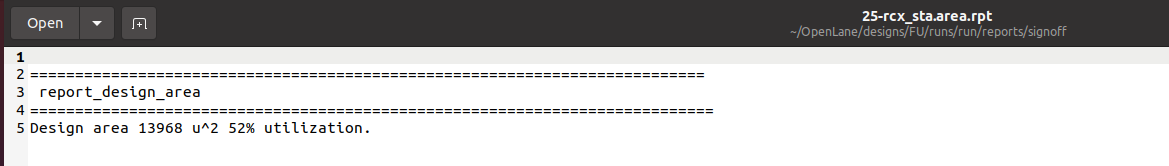
* Behavioral Simulation

As seen below, both shifter output and ALU output can be given. Shifter module and ALU module are connected here. As seen in the simulation, all operators are working successfully. Console output is given below.

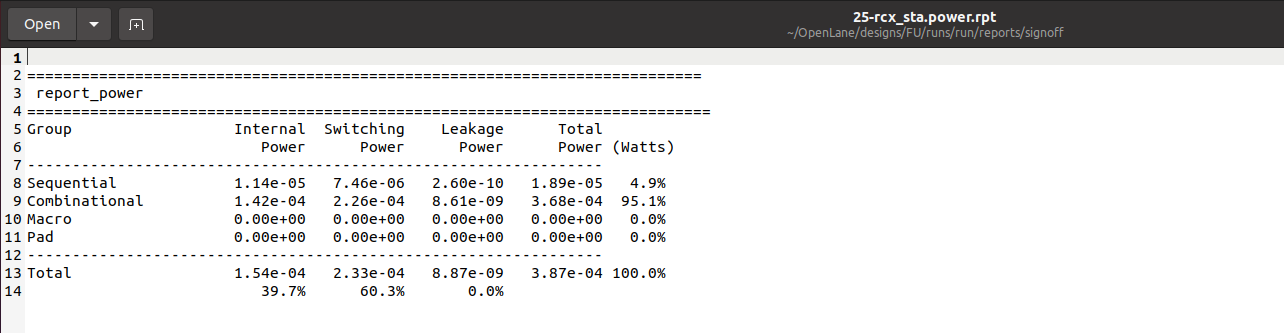


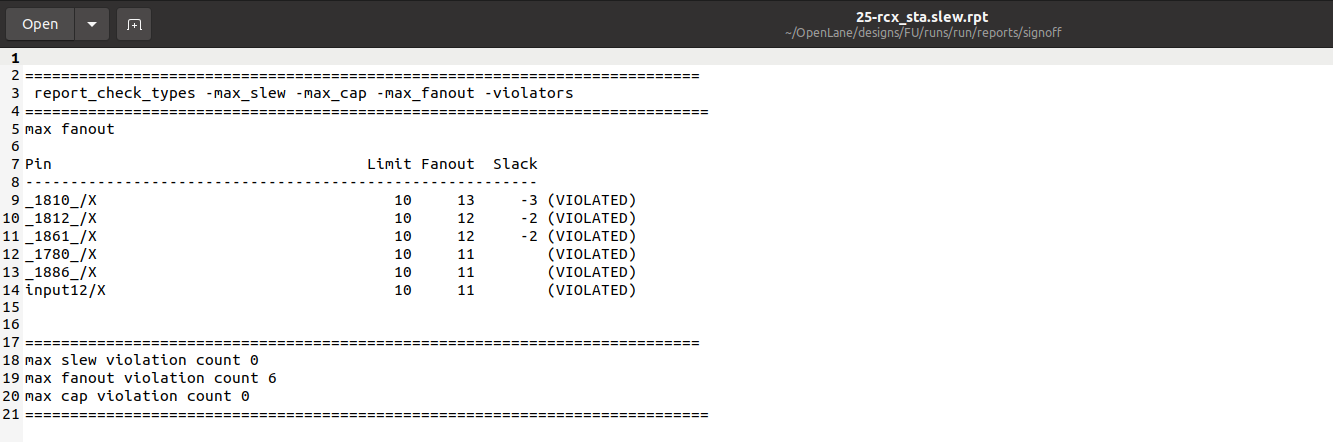


* Openlane Results

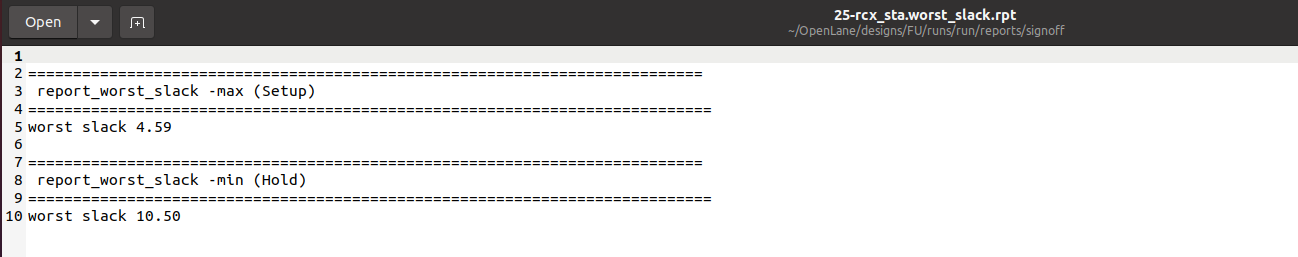
Estimated Area

Power Consumption



Fanout Report

Clock Report



Critical Path Delay

