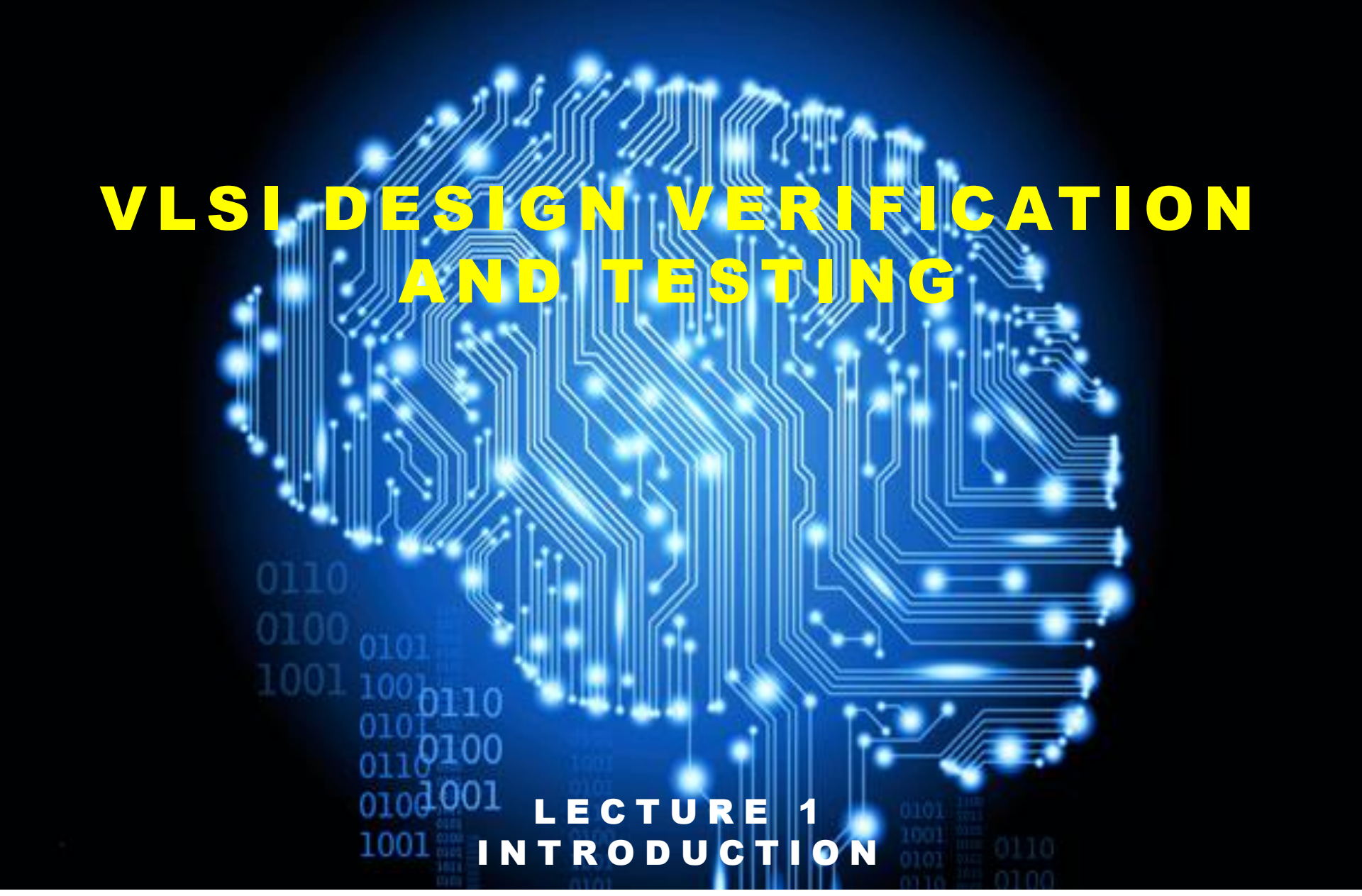


VLSI DESIGN VERIFICATION AND TESTING

LECTURE 1
INTRODUCTION

CSE 412

Ayman wahba

Course Information

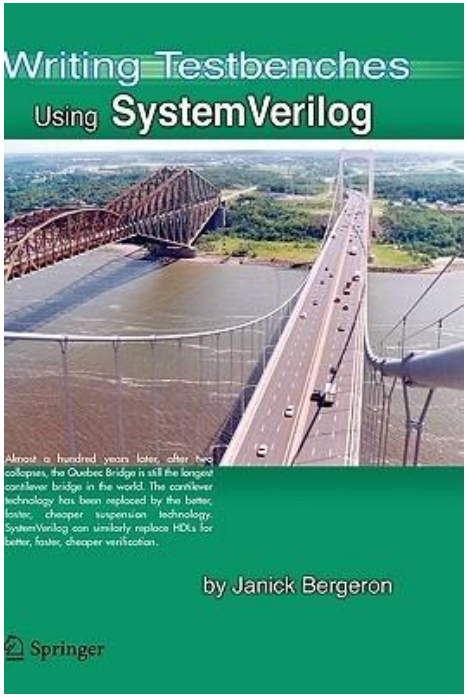
- **Instructor**
  - Ayman M. Wahba, Ph.D. ayman.wahba@eng.asu.edu.eg

- **Lecture Time and Location**
  - Saturday 9:00 – 12:00 Online Weekly

- **Office Hours**
  - Available through Microsoft Teams all days of the week

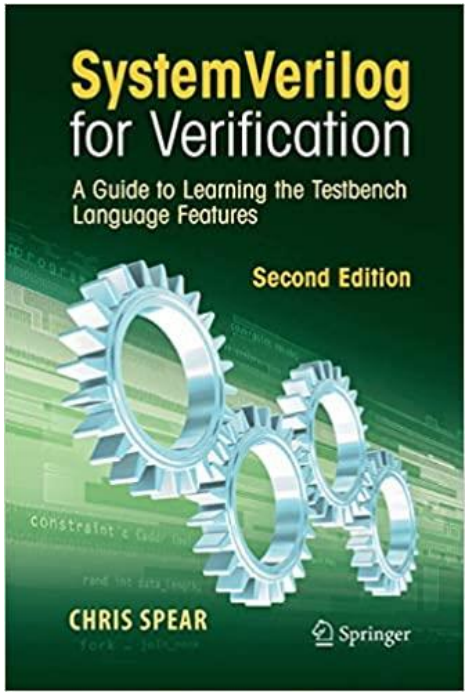
Textbook

- Janick Bergeron, Writing Testbenches using System Verilog, Springer, 2006.



Textbook

- Chris Spear, System Verilog for Verification: A Guide to Learning the Testbench Language Features, *2nd* Edition, Springer, 2008.



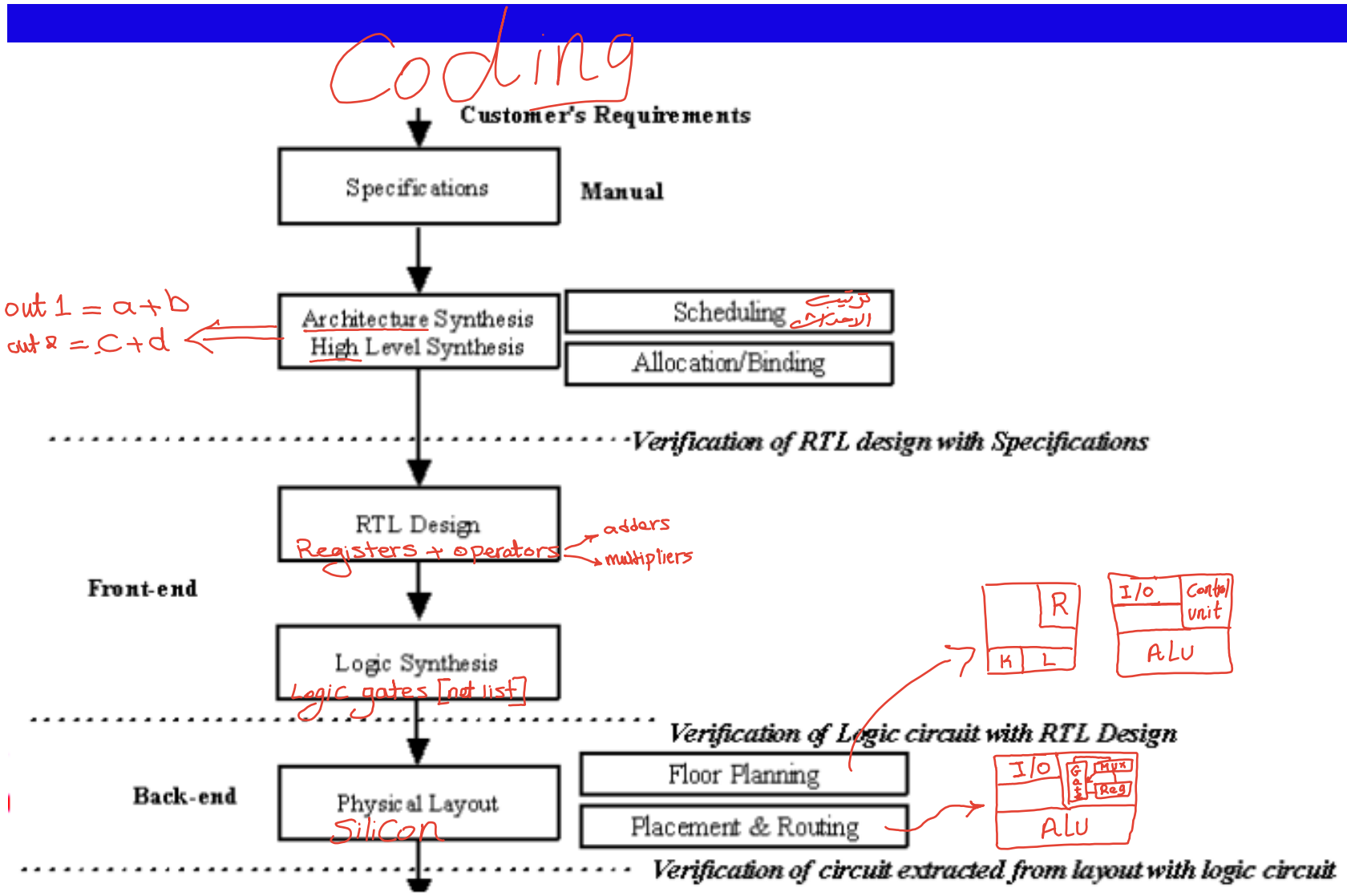
"Everyone knows debugging is twice as hard as writing a program in the first place"

- Brian Kernighan

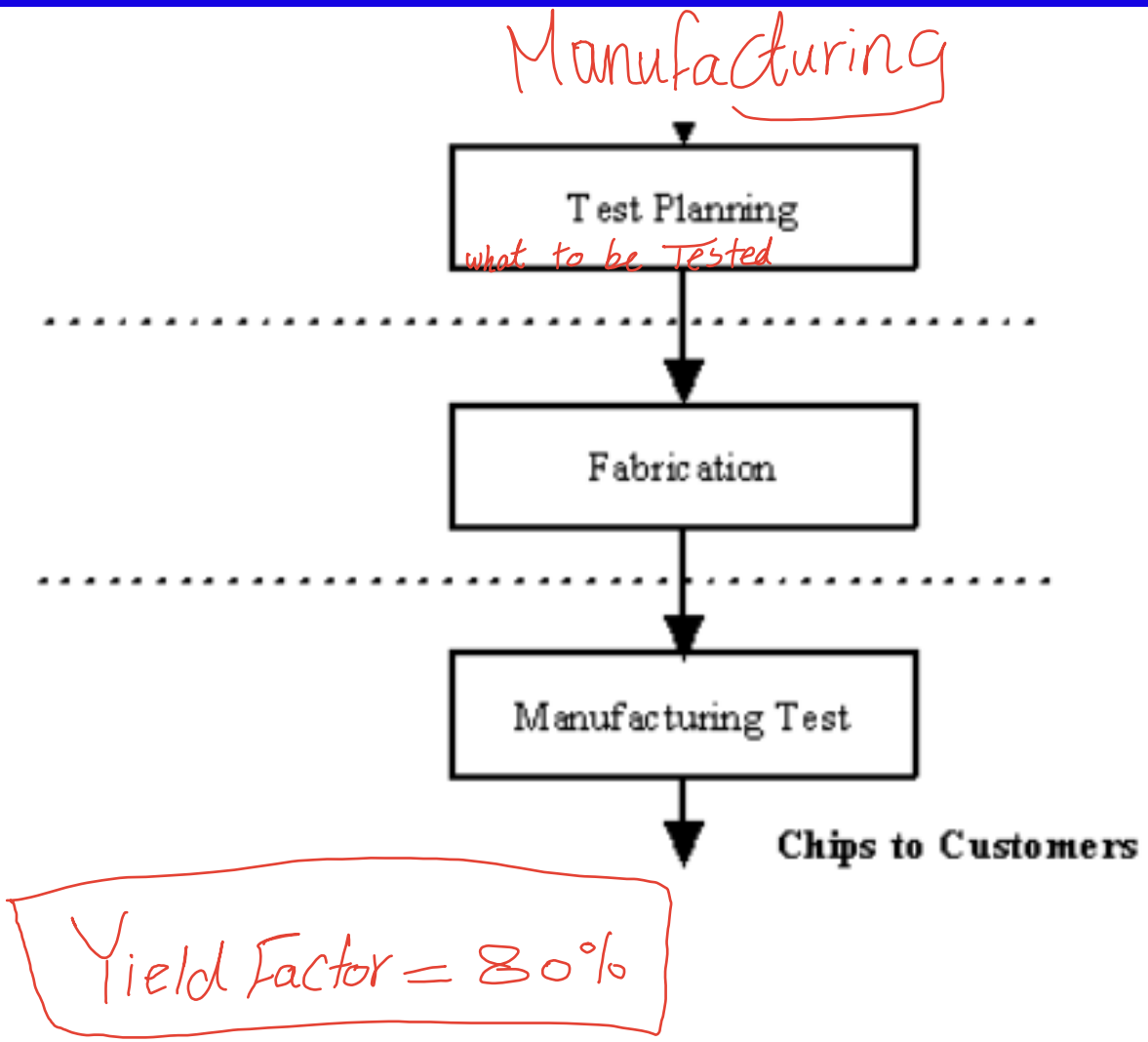
"*Elements of Programming Style*"

1974

Design Flow


Coding
Customer's Requirements
Specifications
Manual
out 1 = a+b
out 2 = c+d
Architecture Synthesis
High Level Synthesis
Scheduling
Allocation/Binding
Verification of RTL design with Specifications
RTL Design
Registers + operators
adders
multipliers
Front-end
Logic Synthesis
Logic gates [net list]
Verification of Logic circuit with RTL Design
Back-end
Physical Layout
Silicon
Floor Planning
Placement & Routing
Verification of circuit extracted from layout with logic circuit
R
K
L
I/O
Control Unit
ALU
Mux
Reg

Design Flow (continued)



1. Specification

Step1: Specification Design

In a typical VLSI flow, we start with system specifications, which is nothing but technical representation of design intent. To explain the flow, the following example will be used through this section.

Example:

Specification: out1=a+b; out2=c+d; where a,b,c,d are single bit inputs and out1,out2 are two bit outputs (sum and carry).

2. High Level Synthesis

Step 2: High level Synthesis

High-level synthesis (HLS) algorithms are used to convert specifications into Register Transfer Level (RTL) circuits.

•HLS, sometimes referred to as architectural synthesis is an automated design procedure that interprets an algorithmic description of the design intent and creates hardware at RTL that implements that behavior.

•The input to a HLS tool is design intent written in some high level hardware definition language like SystemC, System Verilog etc.

•The HLS tool first schedules the computations (required to meet the specifications) at different control steps.

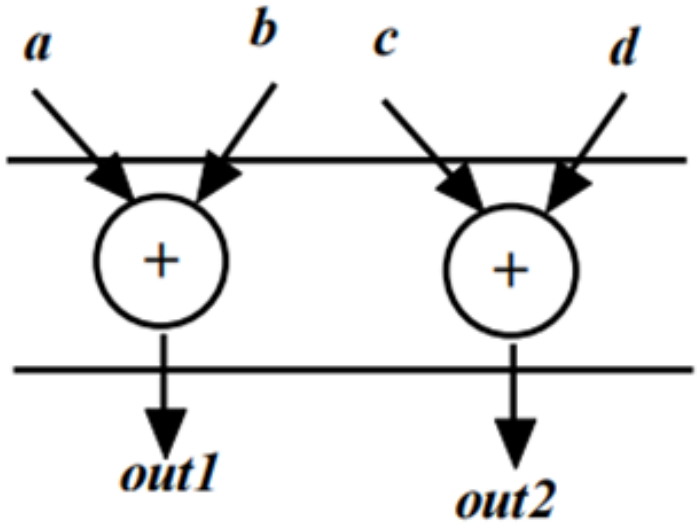
•Following that, depending on availability of hardware units and time constraints, the scheduled computations (comprising instructions and variables) are allocated and binded to the hardware units like adders, multipliers, multiplexors, registers, wires etc.

HLS Example

HLS Example:

In the example there are two operations (addition of single bit numbers) and none of them depend on each other. So both the operations can be scheduled in a single control step. However, if there are dependencies e.g., out1=a+b; out2=out1+d; then "out1=a+b;" is scheduled in 1st control step whereas "out2=out1+d;" is scheduled in 2nd control step.

2 ADDers
6 Registers

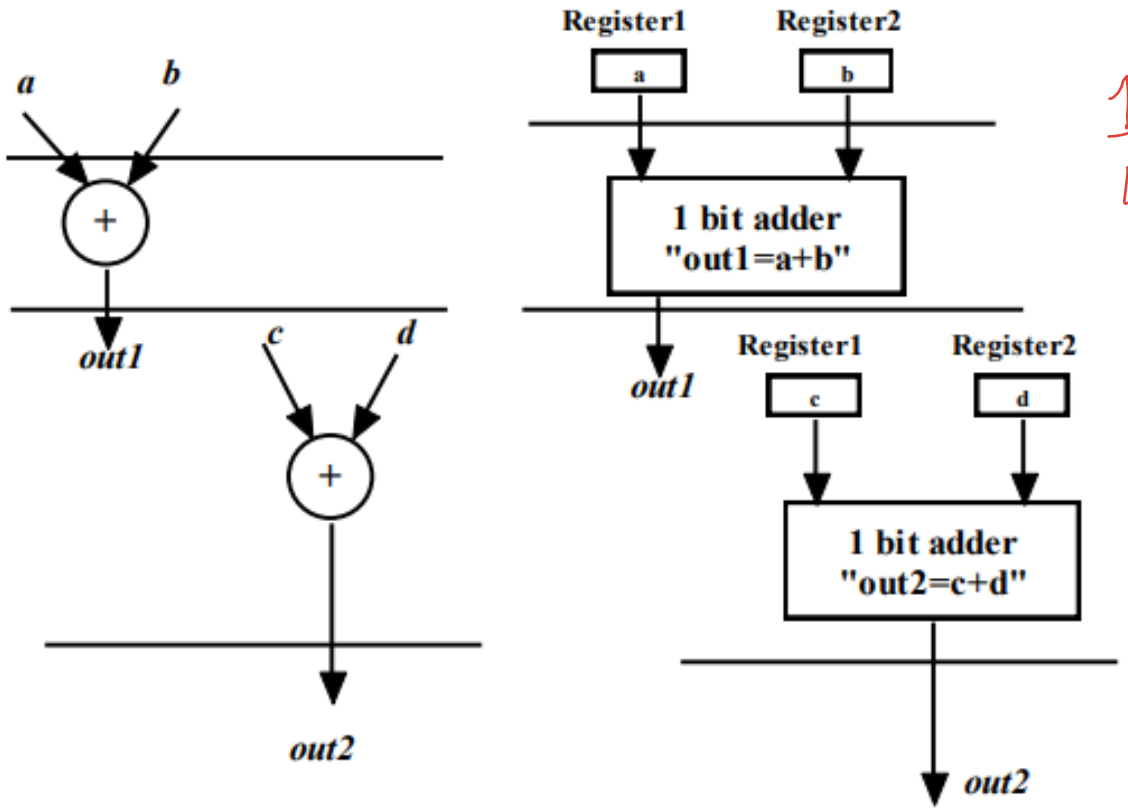


1 step
[Fast]

HLS Example (continued)

•Depending on availability of hardware resources and time constraints the scheduled operators and variables are allocated and binded to hardware units.

Let there be one adder and two registers in the library.



2 steps
[slow]

1 ADDer
4 Registers

HLS Example (continued)

There is one adder and two registers in the library. So the two operations (addition) of the example, even if scheduled in one control step, cannot be allocated to the single adder. Similarly, the four variables cannot be allocated to two registers.

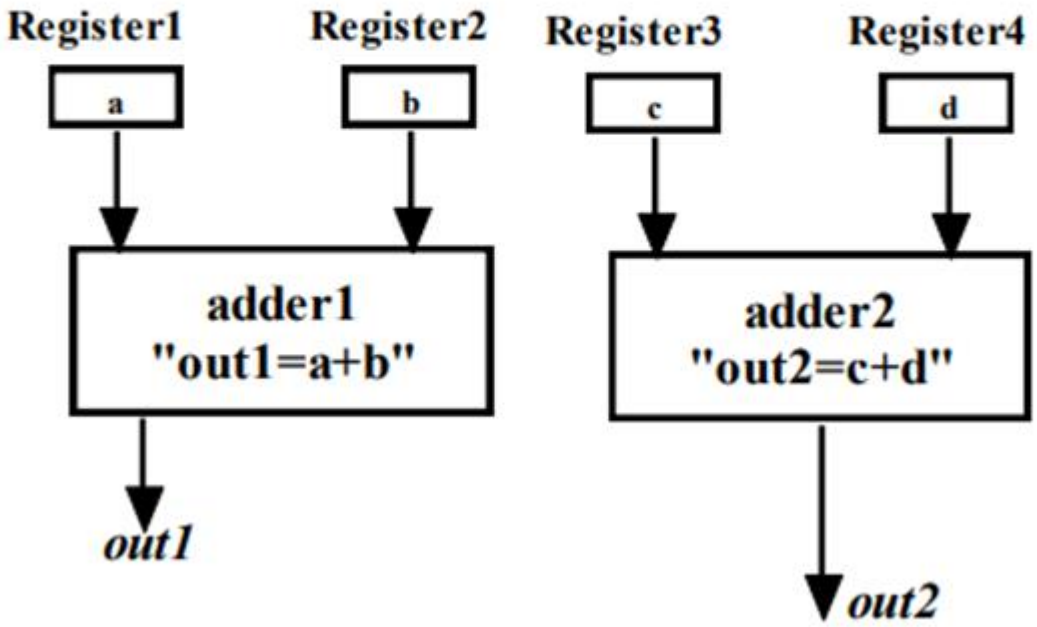
In the running example with the given resource constraints, the two operations can be done in two control steps:

Step 1- variable a is allocated to Register1, variable b is allocated to Register2 and operation “out1=Register1+Register2;” is allocated to adder;

Step 2- variable c is allocated to Register1, variable d is allocated to Register2 and operation “out2=Register1+Register2;” is allocated to adder.

HLS Example (continued)

However, if there are two adders and four registers in the library then both the operations can be carried out in one control step.



HLS Example (continued)

- Finally, based on allocation and binding, the control unit is to be designed (at high level).

- If the allocation/binding is according to (2 adders + 4 registers), the control is trivial.

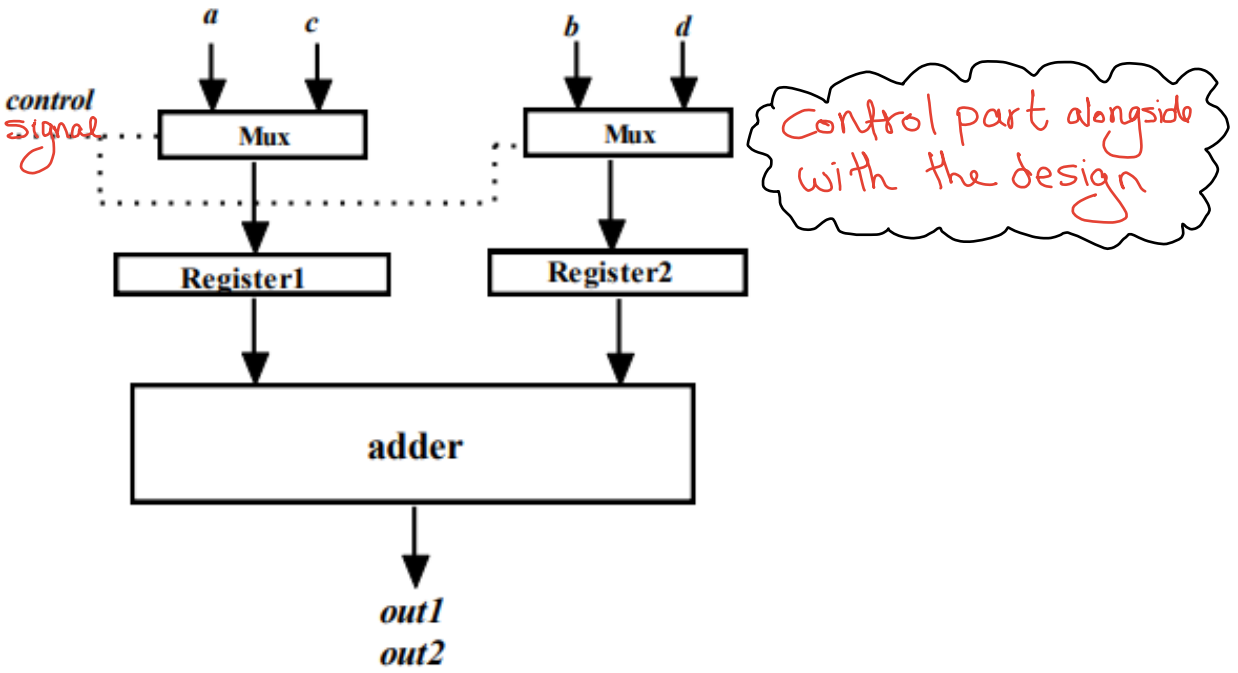
- However, if the allocation is according to (1 adder + 2 registers), then the control circuit needs to provide signals that can do multiplexing between *a* and *c*, *b* and *d*;

  - In 1st control step, *a* should be fed to Register1 and *b* should be fed to Register2,
  - In 2nd control step, *c* should be fed to Register1 and *d* should be fed to Register2.

HLS Example (continued)

•Figure below illustrates the block diagram where control modules are added after allocation and binding (1 adder + 2 registers).

•It may be noted that *control* signal is not available as an external pin which can be controlled by the user. "*Control*" is connected to some signal generated by the system, which alternates in every control step thereby making its value 0 in 1st step and 1 in the 2nd.



HLS Example (continued)

The HLS tool generates output comprising,
(i) operations-variables allocated-binded to hardware units and
(ii) control modules.

The output of HLS tool is called Register Transfer Level (RTL) circuit because data flow, data operations and control flow are captured between registers.

After HLS, RTL circuits are transformed into logic gate level implementation; the step is called logic synthesis.

Verification (Specs vs. RTL)

Before the staring of logic synthesis, one needs to verify if the RTL is equivalent to the specifications.

In the running example, we can verify by applying all possible input conditions of a,b,c,d (along with control) to the RTL and checking if out1 and out2 are as expected.

However, if the RTL has about hundreds of inputs then exercising all possible inputs is impossible because of the exponential complexity (i.e., if there are n inputs then all possible input combinations are 2^n).

So we need to have formal verification methods which verify equivalence of RTL with input specifications.

Verification (continued)

Broadly speaking, for formal verification we need to model the RTL circuit and the specifications using some formal modeling techniques and verify that both of them are equivalent.

In other words, equivalence is determined without applying inputs.

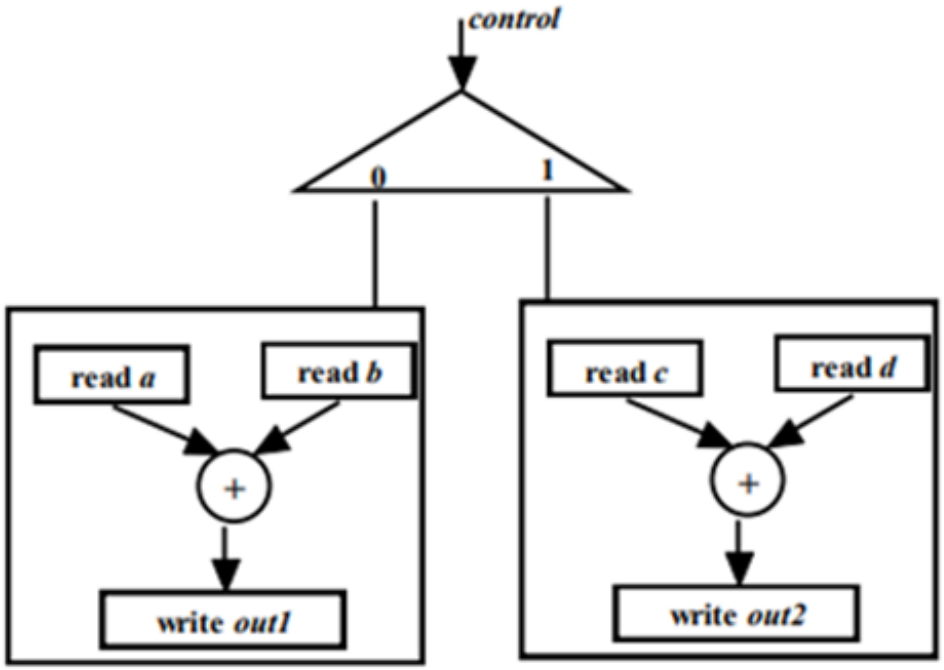
Control and Data Flow Diagram (CDFG), a formal modeling, to capture the RTL. Finite State Machine (FSM) to model the control logic.

This example being very simple, we can see that both specifications and the model are equivalent. Formal techniques for checking equivalence can be will be elaborated in "VERIFICATION" section of the course.

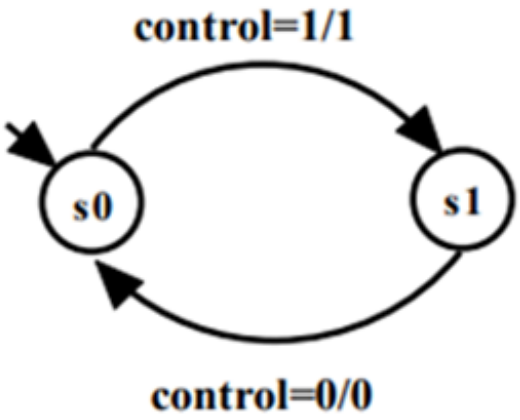
Verification (continued)

This example being very simple, we can see that both specifications and the model are equivalent.

Adders part converted to net list (RTL)



Control part converted to FSM



3. Logic Synthesis

- After the RTL is verified to be equivalent to system specification, logic synthesis is performed by CAD tools.
- In logic synthesis all blocks of the RTL circuit is transformed into logic gates and flip-flops.
  - For the running example all the blocks namely, adder, multiplexers, control logic etc. need to be synthesized to logic gates.

Will illustrate synthesis only for the adder module and for the rest, similar procedure holds.

We first determine the Boolean function of the adder module, in terms of mean terms.

a	*b*	*Out1(sum)*	*Out1(carry)*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

3. Logic Synthesis (continued)

•From the table we have Boolean equations for
Out1(sum)= and Out1(carry)=a.b

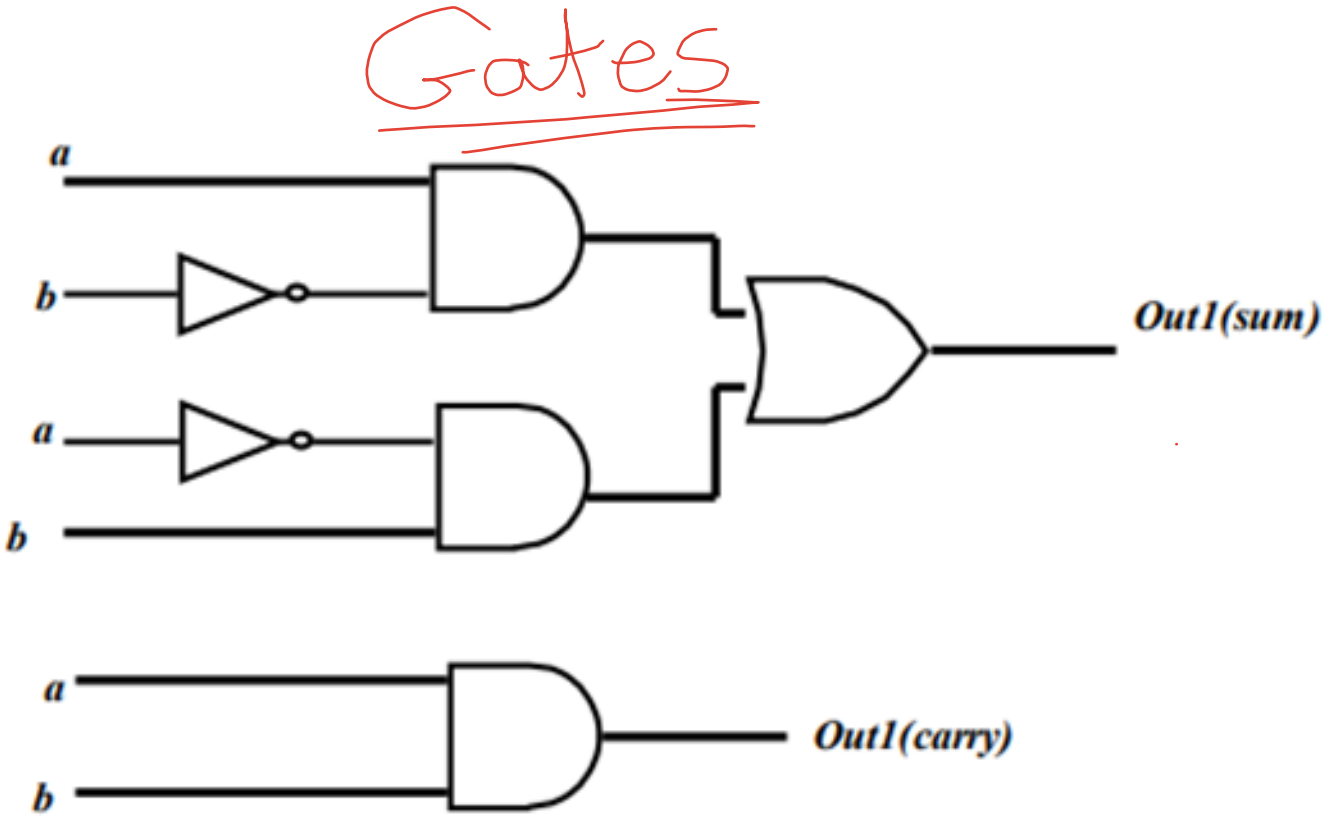
•After the equations are obtained they need to be minimized so that the circuit can be implemented using minimal number of gates. Karnaugh map, Quine–McCluskey algorithm etc. are some standard techniques to minimize Boolean functions.

•In this example of the adder, the equations are already minimized and can be directly converted to Boolean gate implementation as shown.

•Karnaugh map and Quine–McCluskey techniques work well if the number of inputs is less. However, in case of practical VLSI circuits the number of inputs are in orders of hundreds, so minimization is carried out using heuristics techniques.

•Again equivalence of logic synthesis output should be established with RTL design.

3. Logic Synthesis (continued)



4. Backend

•Once the logic level output of the circuit is obtained we move to backend phase of the design process.

In backend we start with a soft version of the silicon die where the chip will be finally fabricated.

- •Broad plan regarding placement of gates, flip-flops etc. (output of logic synthesis) in appropriate places in the soft representation of the chip; this process is called Floorplan.
- •Exact locations in the die (software representation) where the circuit components are placed; this is called Placement.
- •Required interconnections (as given in the logic circuit) among the gates that are placed in exact positions in the die; this process is called routing.

Again equivalence of output of Backend process should be established with logic design.

5. Test

•In VLSI designs millions of transistors are packed into a single chip, thereby leading to manufacturing defects. So all chips need to be physically tested by providing input signals from a pattern generator and comparing responses using a logic analyzer.

• As in the case of verification, testing by applying all possible input combinations is prohibitive, due to curse of dimensionality problem.

•The testing problem is more time hungry than verification because all chips need to be tested while only "one" design is to be verified.

•Testing by applying all possible input combinations is called exhaustive functional testing, which is avoided because of prohibitive time requirements.

5. Test (continued)

•Testing is therefore done based on "structure" of the circuit and is called structural testing.

•In structural testing we first decide on set of faults that can occur, called Fault Models; stuck-at, bridging etc. are some well known fault models.

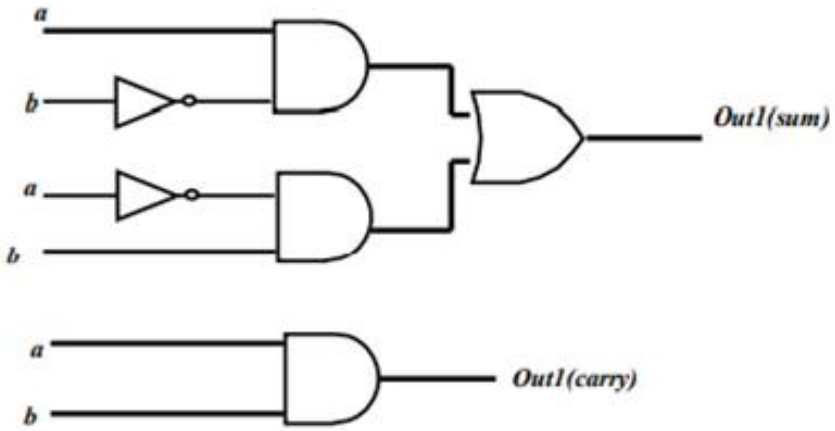
•Then we apply only those inputs which are required to validate that faults (as per fault model) are not present.

•Number of patterns required to perform structural testing is exponentially lower than that required for exhaustive functional testing.

• In Test Planning step, given a logic level circuit and fault model, we generate patterns, which when applied to a circuit determines that no fault from the fault model exists in the circuit.

5. Test (continued)

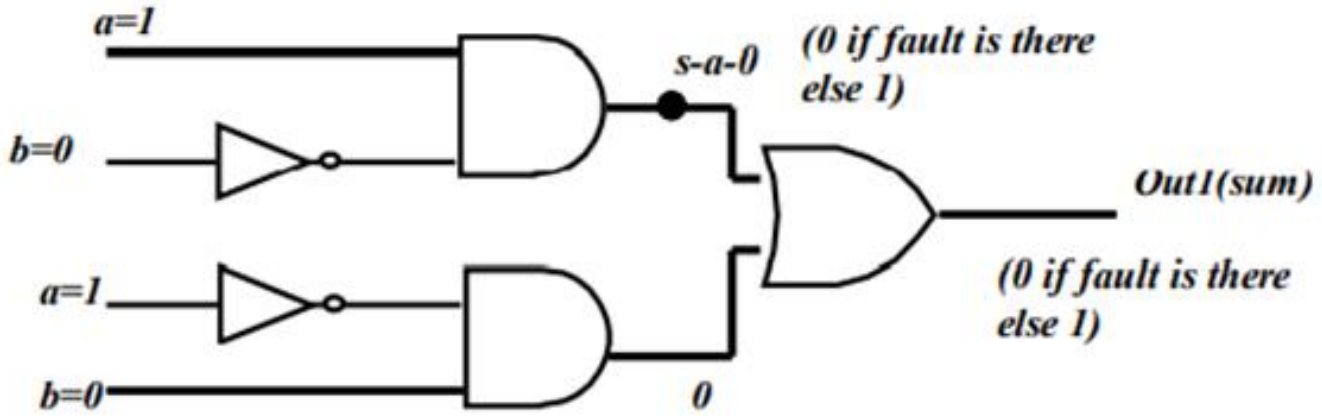
- Test planning for the adder module of the example assuming that fault model is "stuck-at".
  - In "stuck-at" fault model each line of the circuit is assumed to have two types of faults i.e., s-a-0 and s-a-0.
  - So if there are n lines in a circuit then in all there can be 2n stuck-at faults in the circuit.
- In test planning we need to find input patterns which can determine that none of the stuck-at faults are present.
- In the circuit of Figure as there are 12 lines (9 lines in circuit for "sum" and 3 lines in the circuit for "carry"), there can be 24 stuck-at faults.



5. Test (continued)

•Here we will illustrate for only one fault and the same holds for all the other 23 faults. Let there be a stuck-at-0 fault in the output of one AND gate of the circuit for "sum".

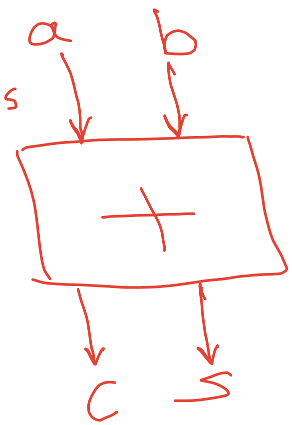
•If a=1 and b=0 is applied as inputs, then "output1(sum)" is 0 if fault is present, 1 otherwise. So a=1 and b=0 can verify the absence of fault by comparing output with 1.



Verification : For design steps before fabrication ⟹ ① Random Tests (1000 tests)
② Constrained Random Tests (CRT)
↳ موجهة

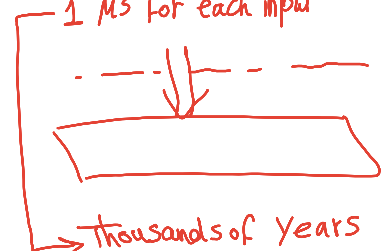
Test : test for fabrication errors

2^4 input possibilities



2^{100} input possibilities

1 Ms for each input



→ Thousands of years [Exhaustive Testing]

عشان اعرف مين عنده مش عايز يطلع

Test Plan

CRT

① add
② MUL in 3 clock cycles
③ Arbiter

1, 2, 3, 4, 5, 6, ..., 10
✓ ✓ ✓ ✓ ✓ ✓

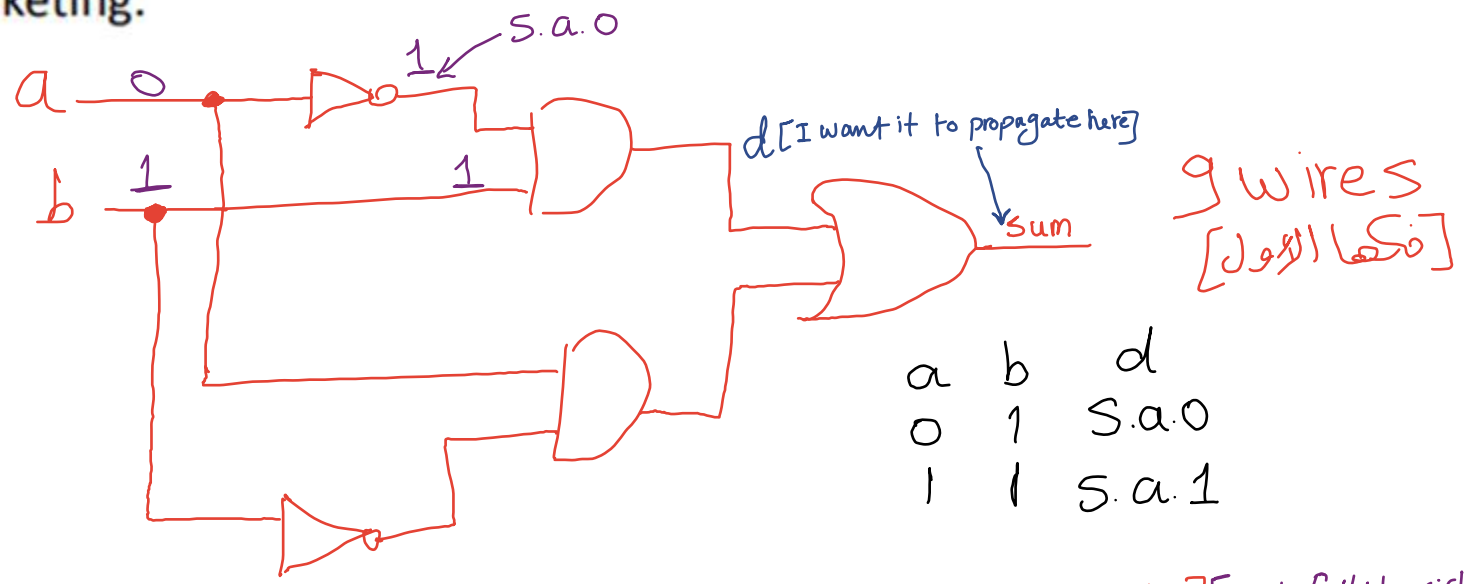
Directed tests

Test Coverage

كام Func من اللي انا عايز اعملهم test تم اختبارهم

6. Fabrication, Test and Marketing

Once all steps are completed and verification after each level of transformations are done, the chips are fabricated, physically tested and fault free chips are sent for marketing.

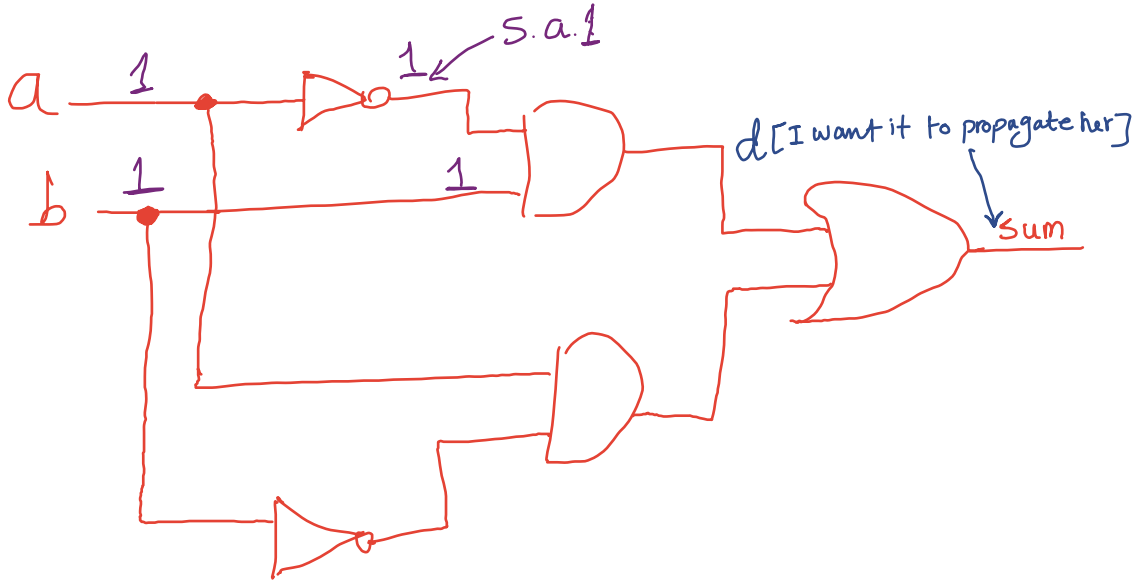


Fault model: e.g stuck at fault [some wire stuck at certain value] [due to faulty transistor or wire]

Test: ① error exitation [use input to generate the opposite of stuck value]

② justification [what primary inputs to cause excitation]

③ propagation



2^{100}

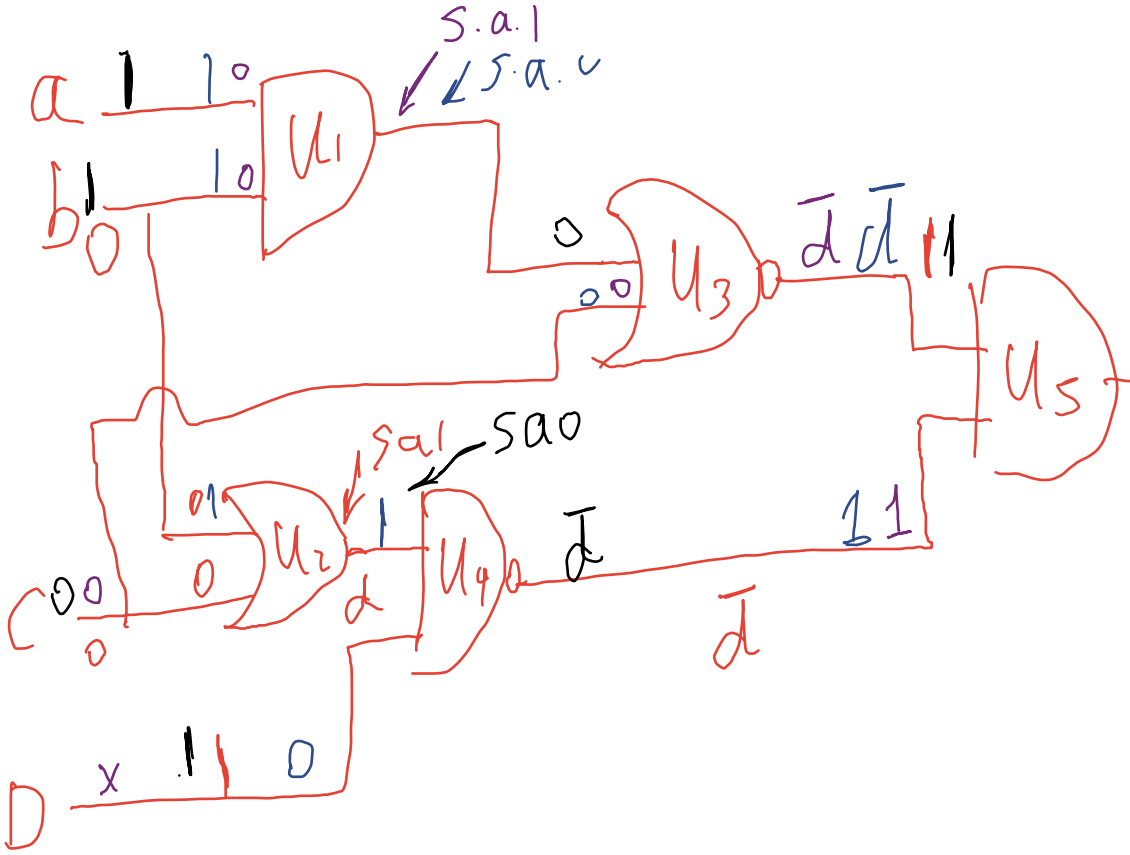
exponential with # of inputs

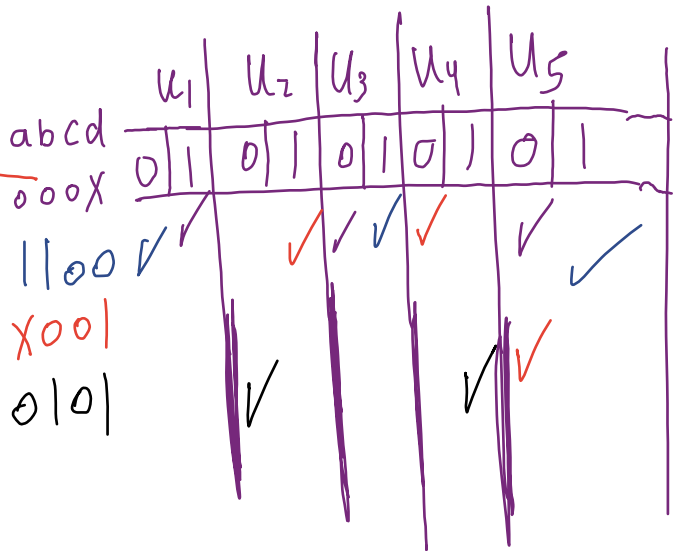
60 s.a. fault

⇓

60 pattern at max

Linear with # of gates


S.a.1
S.a.0
a
b
c
D
U1
U2
U3
U4
U5
sa1
sa0
x


U1
U2
U3
U4
U5
abcd
000X
1100
X001
0101