ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN-ĐIỆN TỬ
BỘ MÔN KỸ THUẬT ĐIỆN TỬ

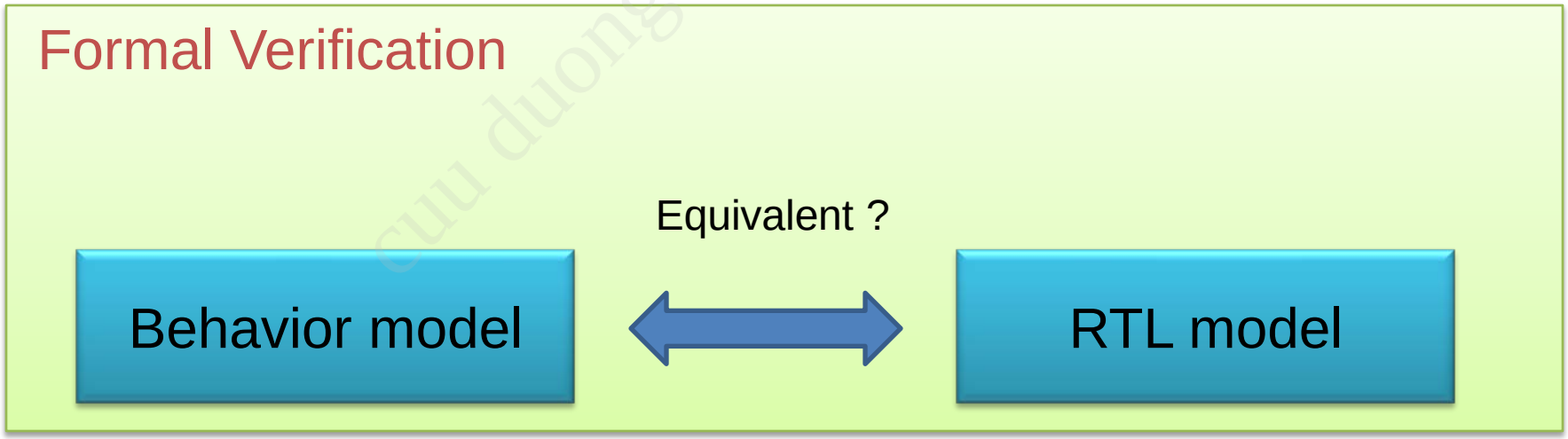
ASIC CHIP AND IP CORE DESIGN

Chapter 4: Verification Process (Part 2)

4.1 Test environment
4.2 Testbench design
4.3 Formal verification
4.4 Boundary-scan test

4.3 Formal Verification

- Prove the chip works
- No simulation
- Combinational Verification
  - Prove combinational circuit (no latches) equivalent to reference design
- Sequential Verification
  - Prove finite-state machine can't get into bad state, can't deadlock, etc



Issues in Simulation

- How to generate vectors ?
  - Based upon specification
  - designer's understanding
  - Random vectors
- How to evaluate simulation results ?
  - Use Golden Model
  - Use specifications
- When to stop simulating ?
  - Statement, condition and state coverage of the model
  - Feature coverage over specification
  - Time to market

Problems in simulation

- Simulation is **slow**, requires billions of vectors for large designs
- Exhaustive simulation **infeasible**
- Coverage of design functionality unknown
- Correctness of golden model **suspect**
- **Bugs** lurk deep in designs that get revealed after complex input sequences

Formal Verification

- Formally check a formal model of the system against a formal specification
- Formal : Mathematical, Precise, unambiguous
- Static analysis
- No test vector
- Exhaustive Verification
- Proves absence of bugs
- Complex and subtle bugs caught
- Early Detection

Simulation vs. Formal Verification

	SIMULATION	FORMAL VERIFICATION
EXHAUSTIVITY	•Not possible to simulate all possible states •Focus on scenarios and assertions to break the design	•Explores all possible states •Results in high quality RTL •Shifts focus on intent-correct functional behavior
CONTROLLABILITY	•Must conceive of vectors, scenarios to adequately simulate design •Likely to miss corner case scenarios	•No stimulus required •Start early in the design cycle •All corner case bugs caught
OBSERVABILITY	•Must propagate bugs to output pins •insert local assertions to expose bugs and aid debug	•Automatically isolate root cause of bugs •Visualize incorrect behaviors and fix them

Formal Verification Tools

- Commercial tools :
  - Formality (Synopsys,)
  - Incisive Formal Verifier (Cadence)
  - Design VERIFYer (Chrysalis Inc.)
  - Spin (Bell Labs.),
  - VIS (UCB),
  - SMV (CMU, Cadence)
  - In-house tools: Rule Base (IBM), Intel, SUN, Bingo (Fujitsu), etc

Three-step process

- Three steps are
  - Formal Specification
  - Formal Models
  - Verification

1. Formal specification

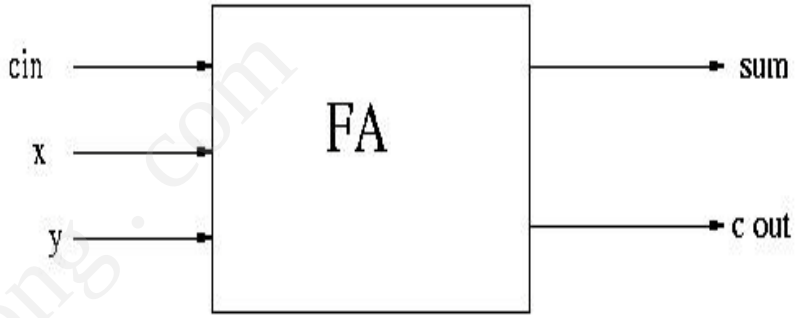
- Precise statement of properties
- System requirements and environmental constraints
- Logic, temporal logic
- Automata, labeled transition systems

Three-step process

2. Models
   - Flexible to model general to specific designs
   - Non-determinism, concurrency, fairness,
   - Transition systems, automata
3. Verification
   - Checking that model satisfies specification
   - Static and exhaustive checking
   - Automatic or semi-automatic

Formal Verification: Example

FULL ADDER:



- Formal Specification:
  - sum := (x ⊕ y) ⊕ cin
  - cout := (x ∧ y) ∨ (x ∧ cin) ∨ (y∧cin)
- Formal Model

  Verilog implementation of full adder
  - Assign sum = x xor y xor cin;
  - Assign cout = (x and y) or (x and cin) or (y and cin);
- Verification
  - Comparision of boolean expressions
  - Equivalence Checking
  - Theorem Proving, in general

Formal Verification Techniques

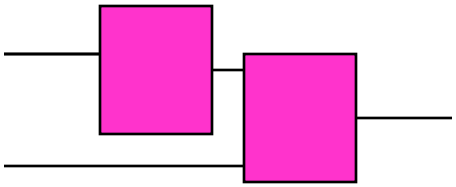
- Three major techniques
  - **Equivalence checking (HW)**
    - Compares optimized/synthesized model against original model
  - **Model checking(HW & SW)**
    - Checks if a model satisfies a given property
  - **Theorem proving(HW & SW)**
    - Proves implementation is equivalent to specification in some formalism

Equivalence Checking

- Combinational Equivalence Checking
  - Using **Binary Decision Diagrams** (BDDs)
    - The functions of the two circuits to be compared are converted into canonical representations of Boolean functions
    - The BDDs are then structurally compared.
    - Unable to compare the RTL model with a behavioral model.
  - Logic Equivalence Checking:
    - This is achieved by dividing the model into logic cones between registers, latches or black-boxes.
    - The corresponding logic cones are then compared between original and optimized models.
- Sequential Equivalence Checking
  - verifies the equivalence between two sequential designs at each valid state.
  - Can verify the equivalence between RTL and the behavioral model.

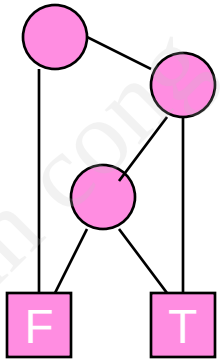
Combinational Equivalence Checking

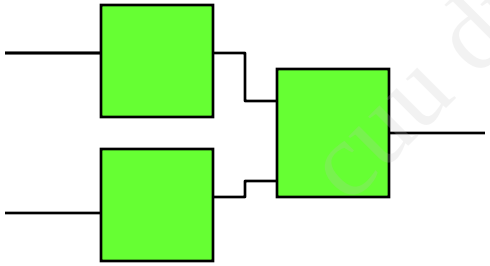
Circuit/Implementation
(comb. ckt)



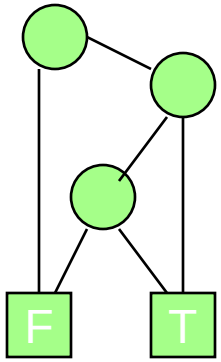
Circuit/Implementation A

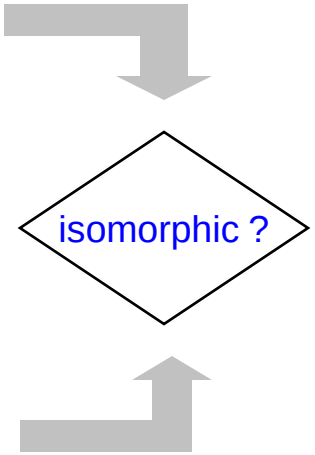
BDD
(Binary Decision Diagram)





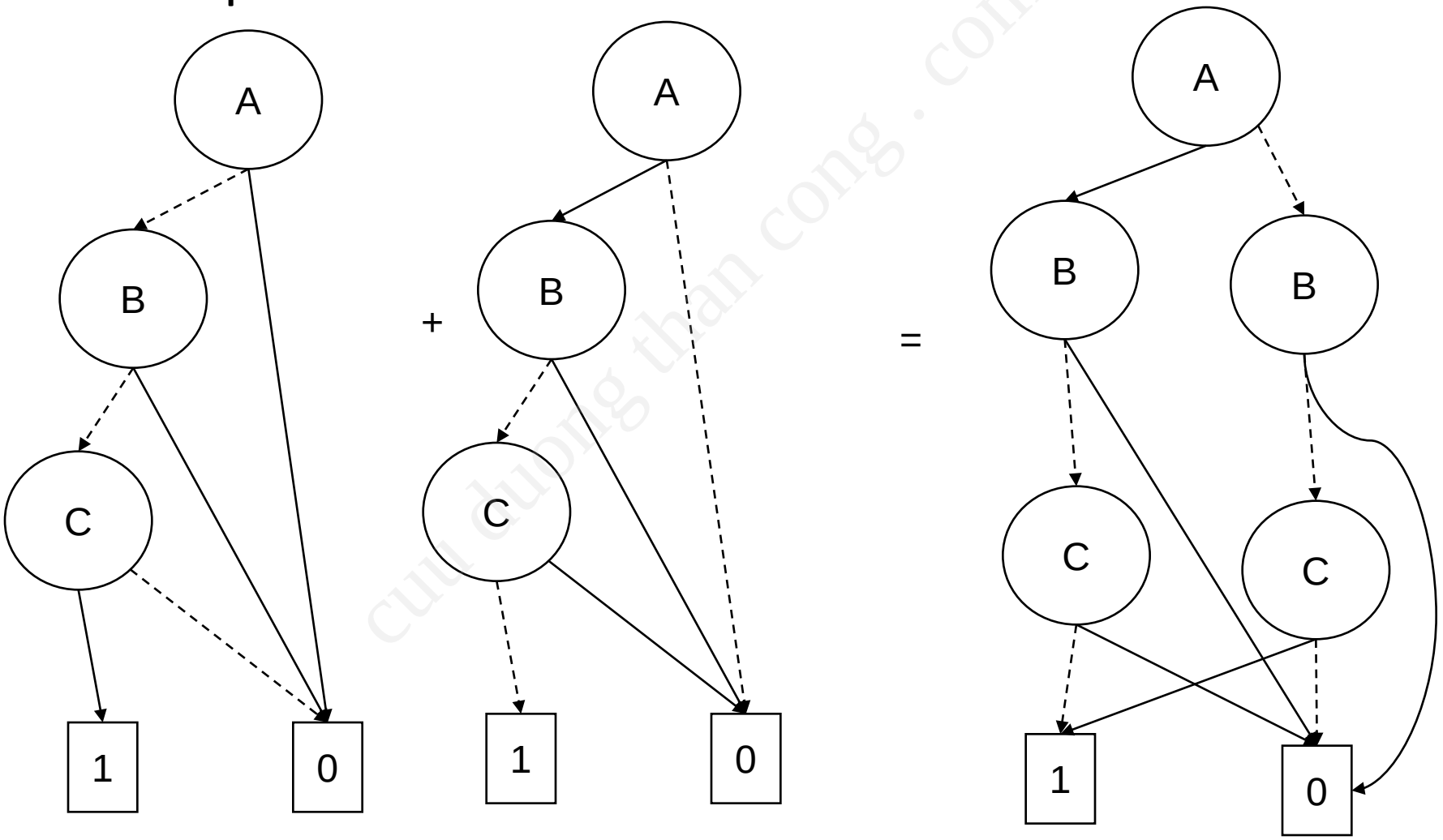
Circuit/Implementation B





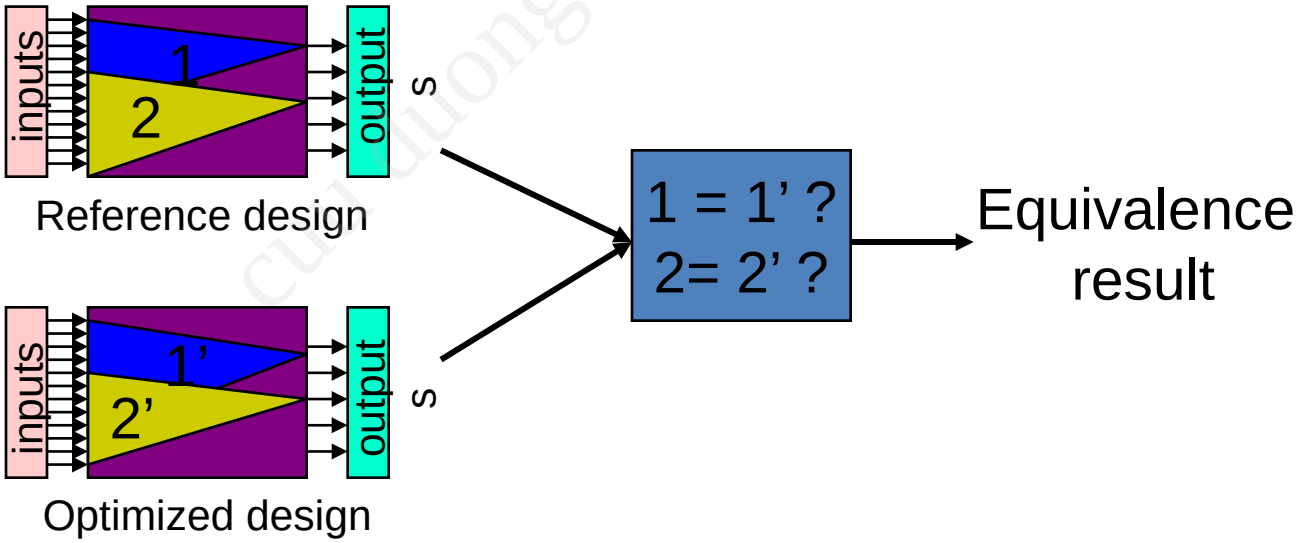
BDD Example

- Example: a’b’c + ab’c’



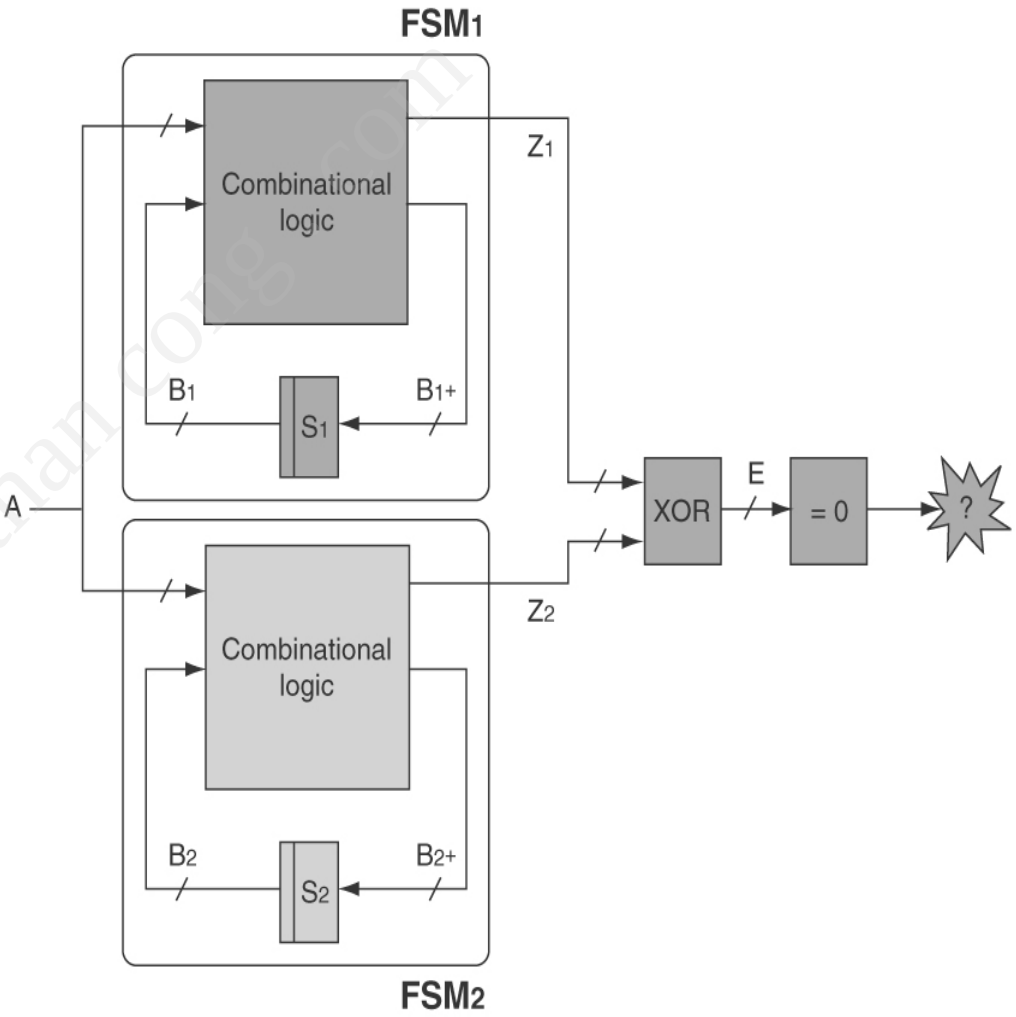
Combinational Equivalence Checking

- Task : Check functional equivalence of two designs
- Inputs
  - Reference (golden) design
  - Optimized (synthesized) design
  - Logic segments between registers, ports or black boxes
- Output
  - Matched logic segment equivalent/not equivalent



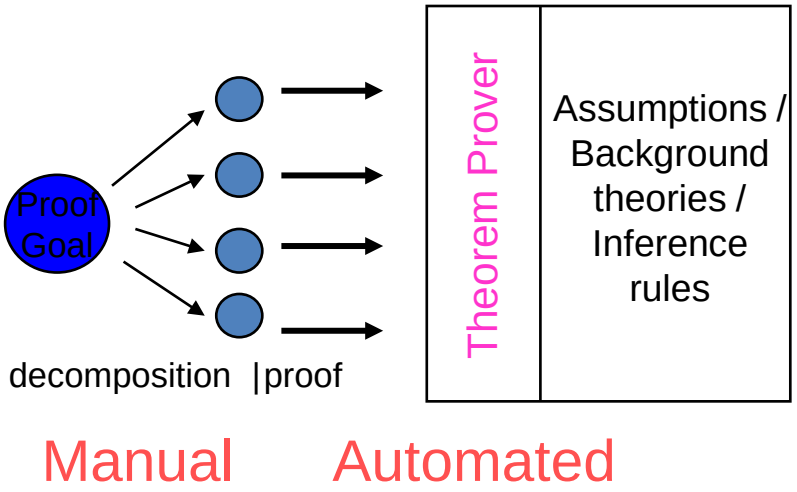
Sequential Equivalence Checking

- Common input A drives the two Models
- State Vectors are held in S1 and S2 and initialised to a common value – all zeroes for instance.
- The outputs Z1 and Z2 are compared by XORing them
- If E remains zero then the two models are equivalent
- The state space explosion rules out this approach



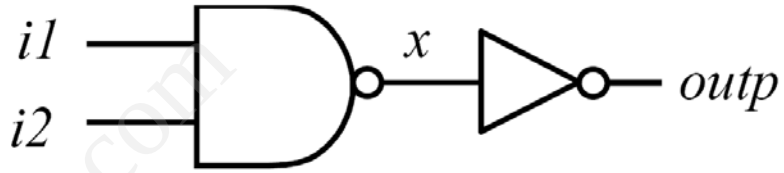
Theorem Proving

- Task :
  - Prove implementation is equivalent to spec in given logic
- Inputs
  - Formula for specification in given logic (spec)
  - Formula for implementation in given logic (impl)
  - Assumptions about the problem domain
    - Example : Vdd is logic value 1, Gnd is logic value 0
  - Background theory
    - Axioms, inference rules, already proven theorems
- Output
  - Proof for spec = impl



Theorem Proving

- Example
  - AND Logic



Implementation of AND

- 4 steps:
  - Specify the implementation of the AND gate,
  - Specify the behavioral models for the NAND and NOT gates,
  - Specify the intended behavior of the AND gate,
  - Prove that the implementation satisfies its intended behavior.

- Drawbacks of Theorem Proving
  - Not easy to deploy in industry
    - Most designers don't have background in math logic
    - Models must be expressed as logic formulas
  - Limited automation
    - Extensive manual guidance to derive proof sub-goals

Theorem Proving

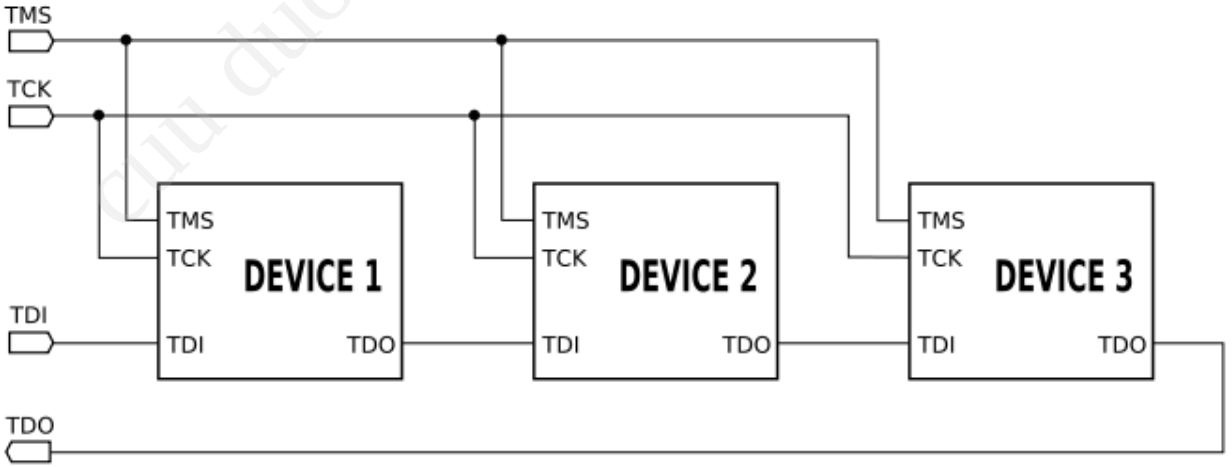
- Theorem prover is less automatic than a model checker (more of an assistance to the user):
  - User:
    1. assembles relevant info for the tool,
    2. sets up intermediate goals (i.e. lemmas)
  - Tool:
    - Attempts to achieve the intermediate goals based on input data.
- Effective use of it requires
  - a solid understanding of the internal operations of the tool and
  - familiarity with the mathematical proof process.

Class Assignments

1. Describe **Formal Specification** and **Formal Model** for the following designs:
   - Full subtractor
   - Excess-3 to bcd code converter
   - Binary to Gray code converter
2. Perform **Combinational Equivalence Checking** for the following designs:
   - Decoder 3-to-8
   - Multiplexor 8-to-1
3. Draw BDD for the following functions:
   - f = a'bc' + abc
   - f = a'b'c + ab'c' + a'bc' + abc

4.4 Boundary-Scan Test (BST)

- IEEE 1149.1 Boundary-Scan
  - Facilitates board testing
  - Provides an on-chip means of controlling and testing pads
  - Boundary-scan components can also be used for other test purposes:
    - Logic and RAM BIST control
    - Scan chain control
    - Scan wrapper config., test modes, etc.

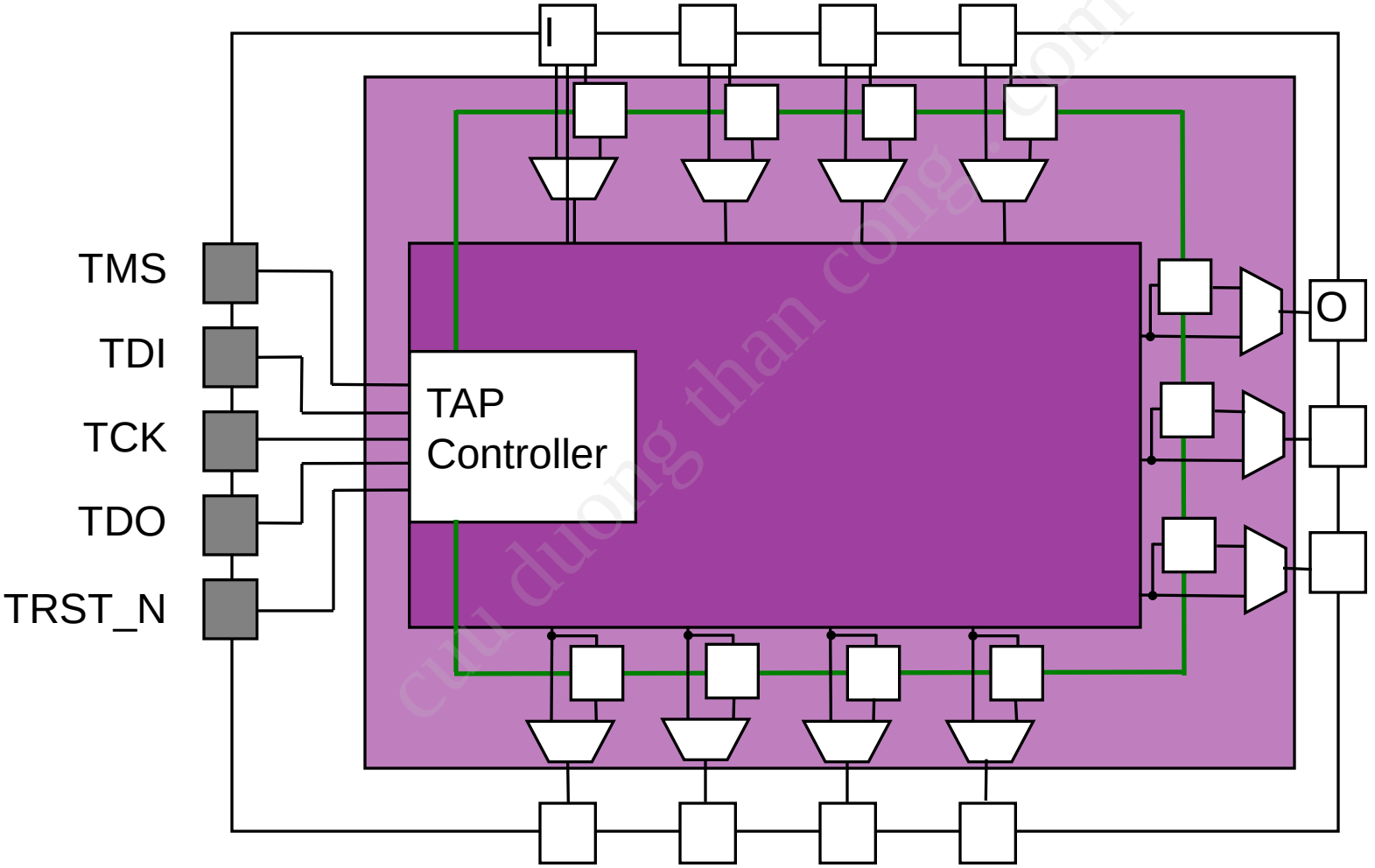


Board Test

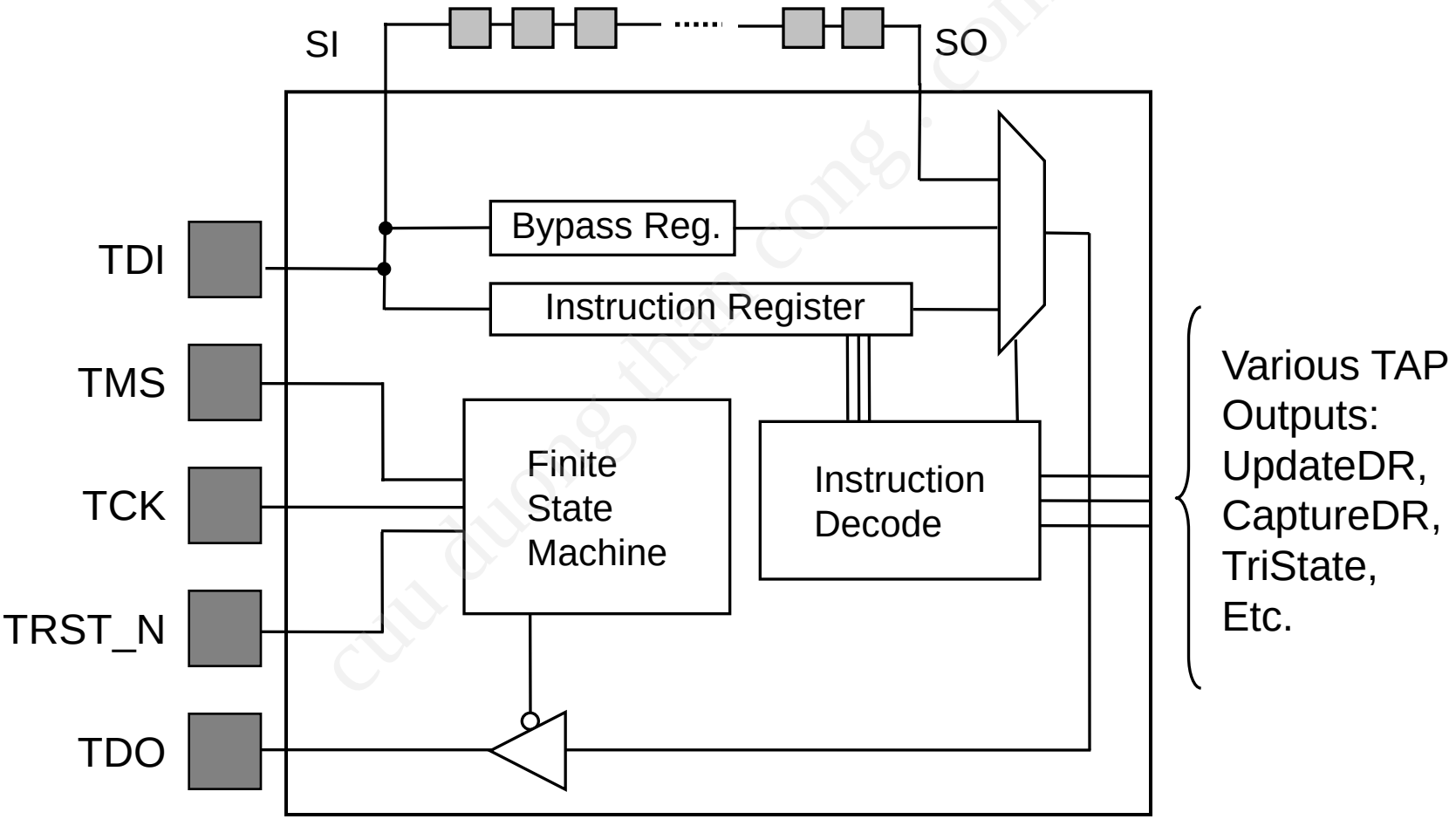
Boundary Scan Terminology

No.	Acronym	Meaning
1	BR	Bypass register
2	BSC	Boundary Scan cell
3	BSR	Boundary scan register
4	BST	Boundary scan test
5	IDCODE	Device identification register
6	IR	Instruction register
7	JTAG	Joint Test Action Group
8	TAP	Test access port
9	TCK	Test clock
10	TDI	Test data input
11	TDO	Test data output
12	TDR	Test data register
13	TMS	Test mode select
14	TRST / nTRST	Test reset input signal

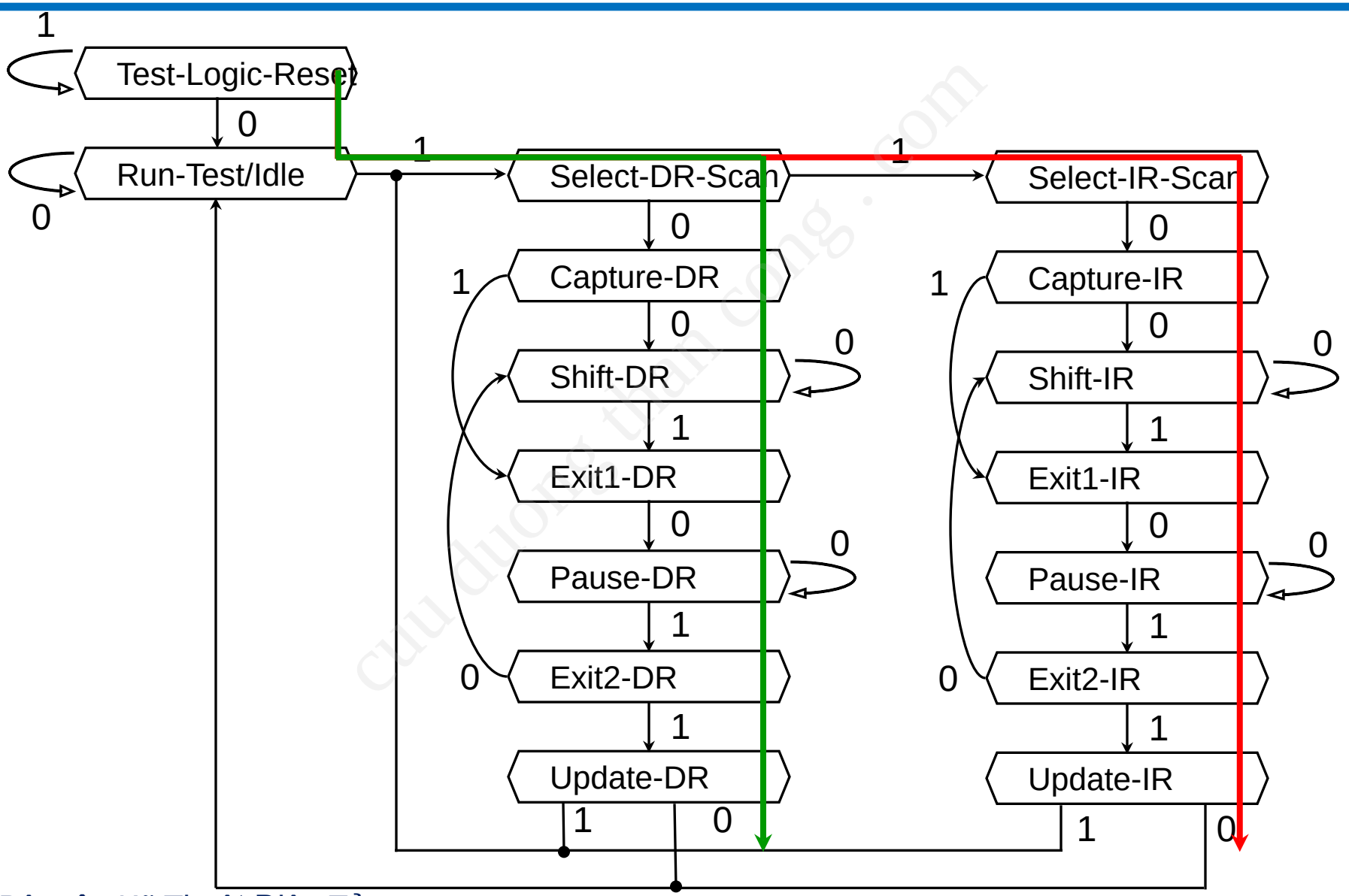
Boundary-Scan Components



TAP Controller Components



TAP Controller State Diagram

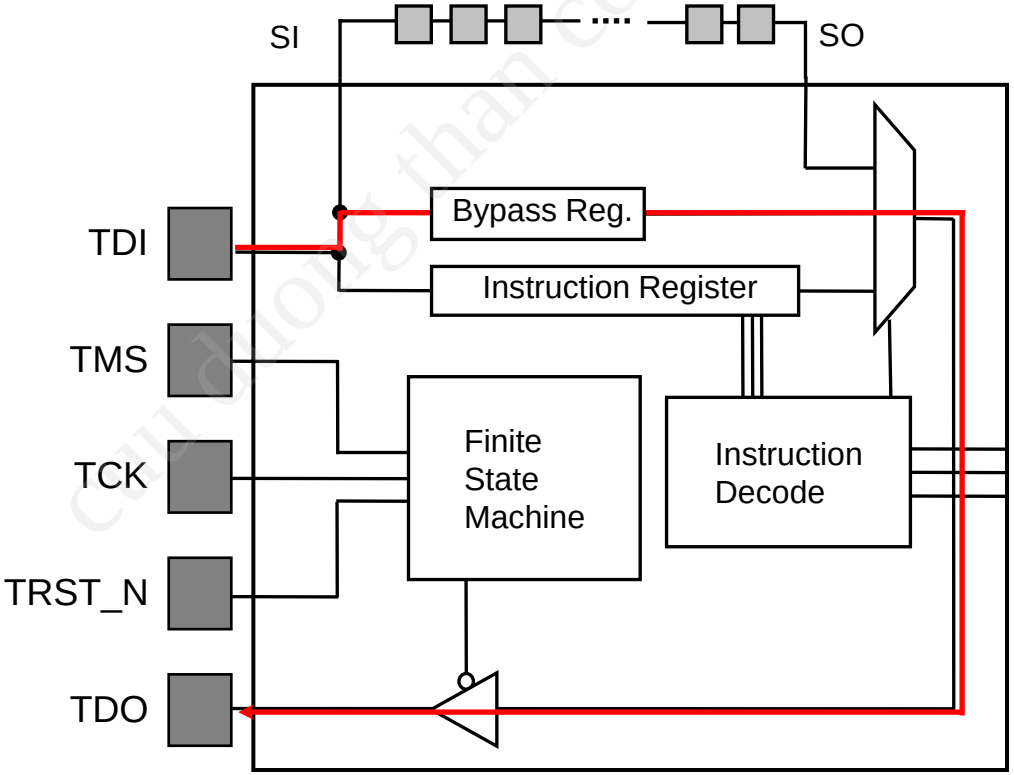


Boundary Scan Instruction

No.	Instruction	Status	Description
1	BYPASS	Mandatory	Bypass data from TDI to TDO through BYPASS register
2	SAMPLE / PRELOAD	Mandatory	Load data into boundary scan register prior to loading an EXTEST instruction.
3	EXTEST	Mandatory	selects the boundary scan register to be connected between TDI and TDO, allows the user to set and read pin states
4	INTEST	Optional	causes the TDI and TDO lines to be connected to the Boundary Scan Register
5	HIGHZ	Optional	forces all drivers into high impedance states.
6	IDCODE	Optional	causes the TDI and TDO to be connected to the IDCODE register.
7	USERCODE	Optional	enables a user-programmable 32 bit identification code to be shifted out for examination

BYPASS Instruction

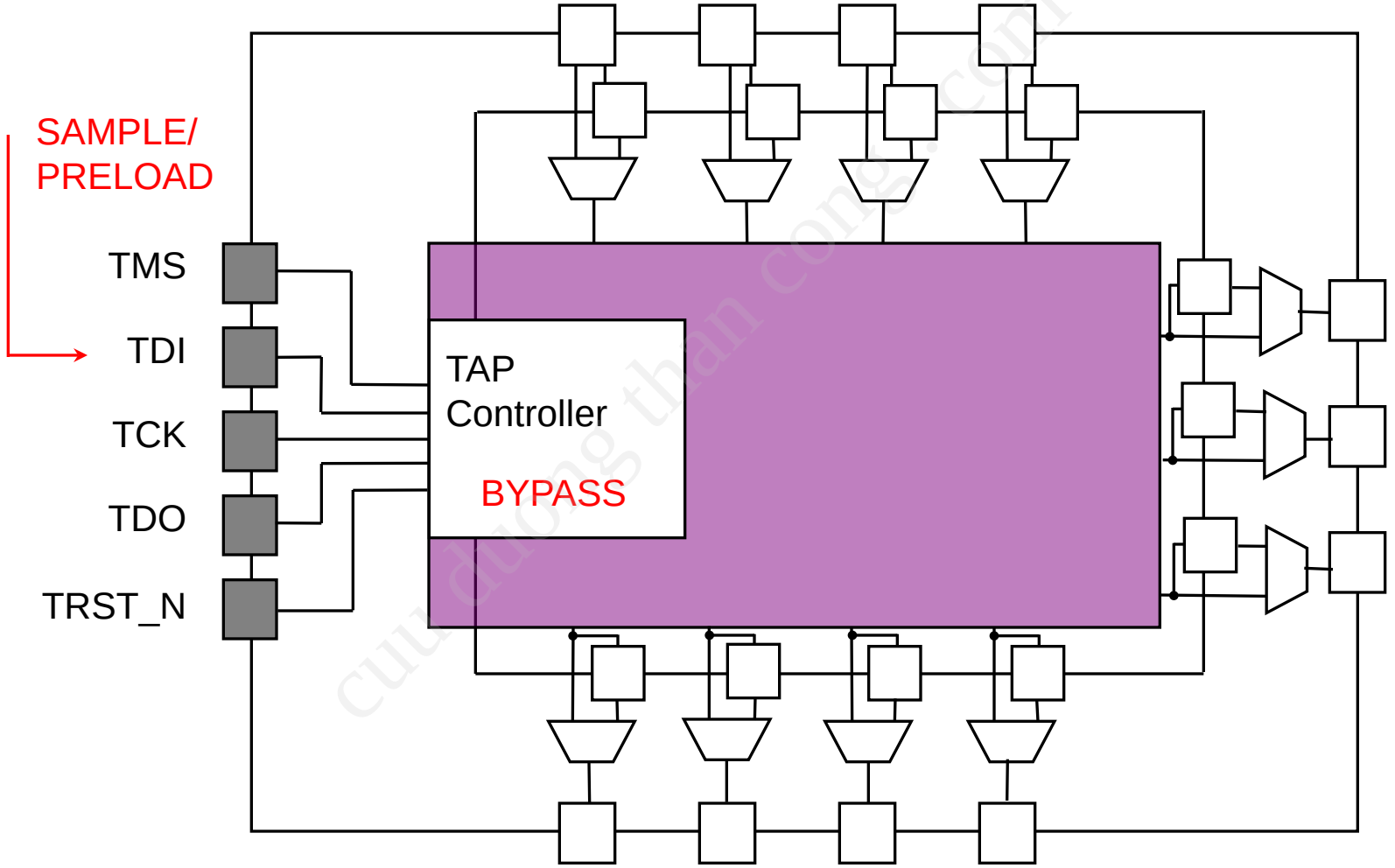
- A mandatory instruction
- The default instruction for TAPs with no IDCODE register
- Short scan path: 1 bit between TDI and TDO
- Usually loaded in chips that are idle while other chips on the board are being tested



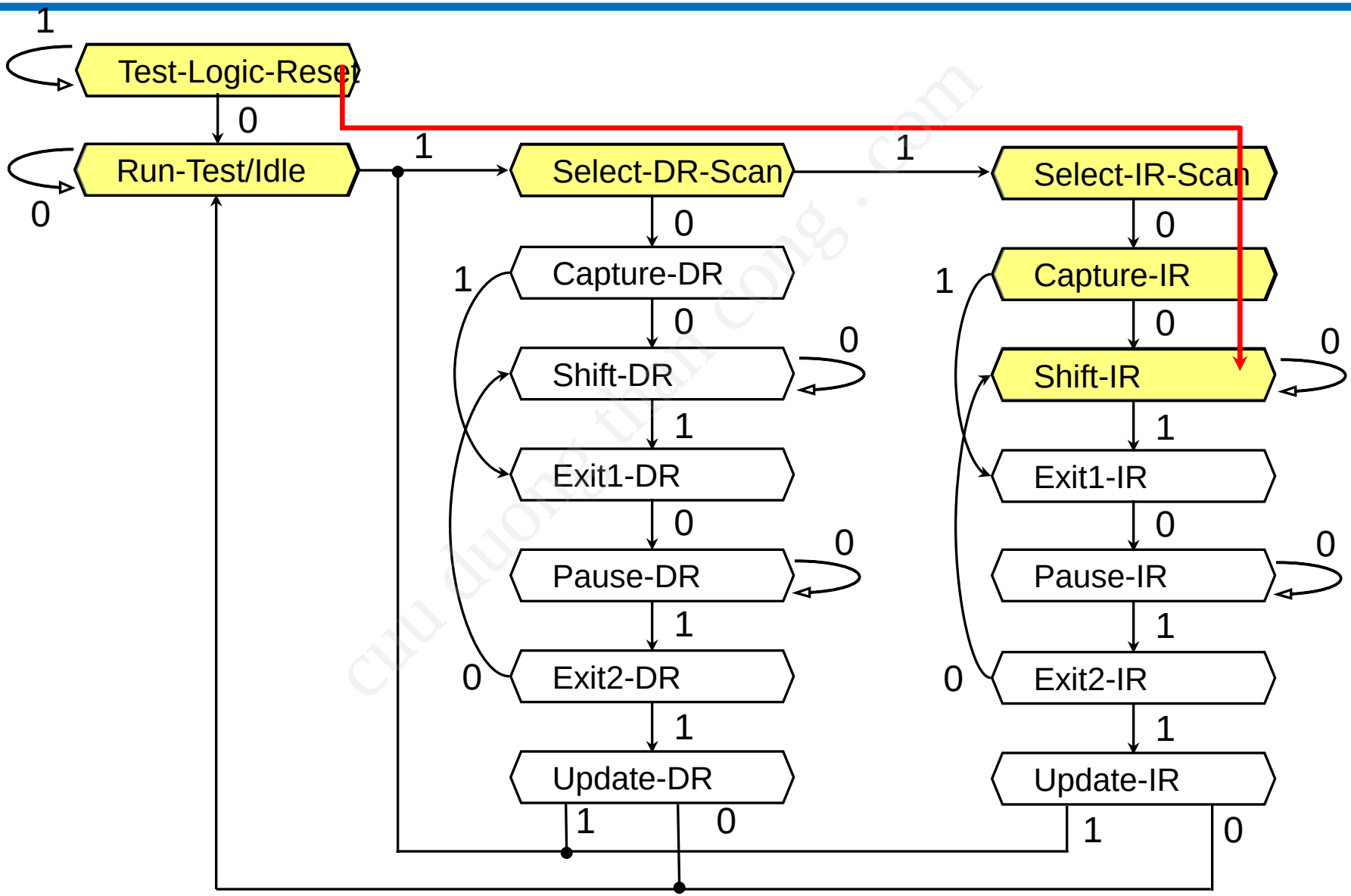
EXTEST & SAMPLE/PRELOAD

- EXTEST is the JTAG instruction
  - Sample (“Capture”) & Drive (“Update”) output signals
  - Sample & optionally drive input signals
- Data is first loaded into boundary register chain with SAMPLE/PRELOAD instruction
  - Samples inputs and outputs, pass-through
  - Loads boundary register with data

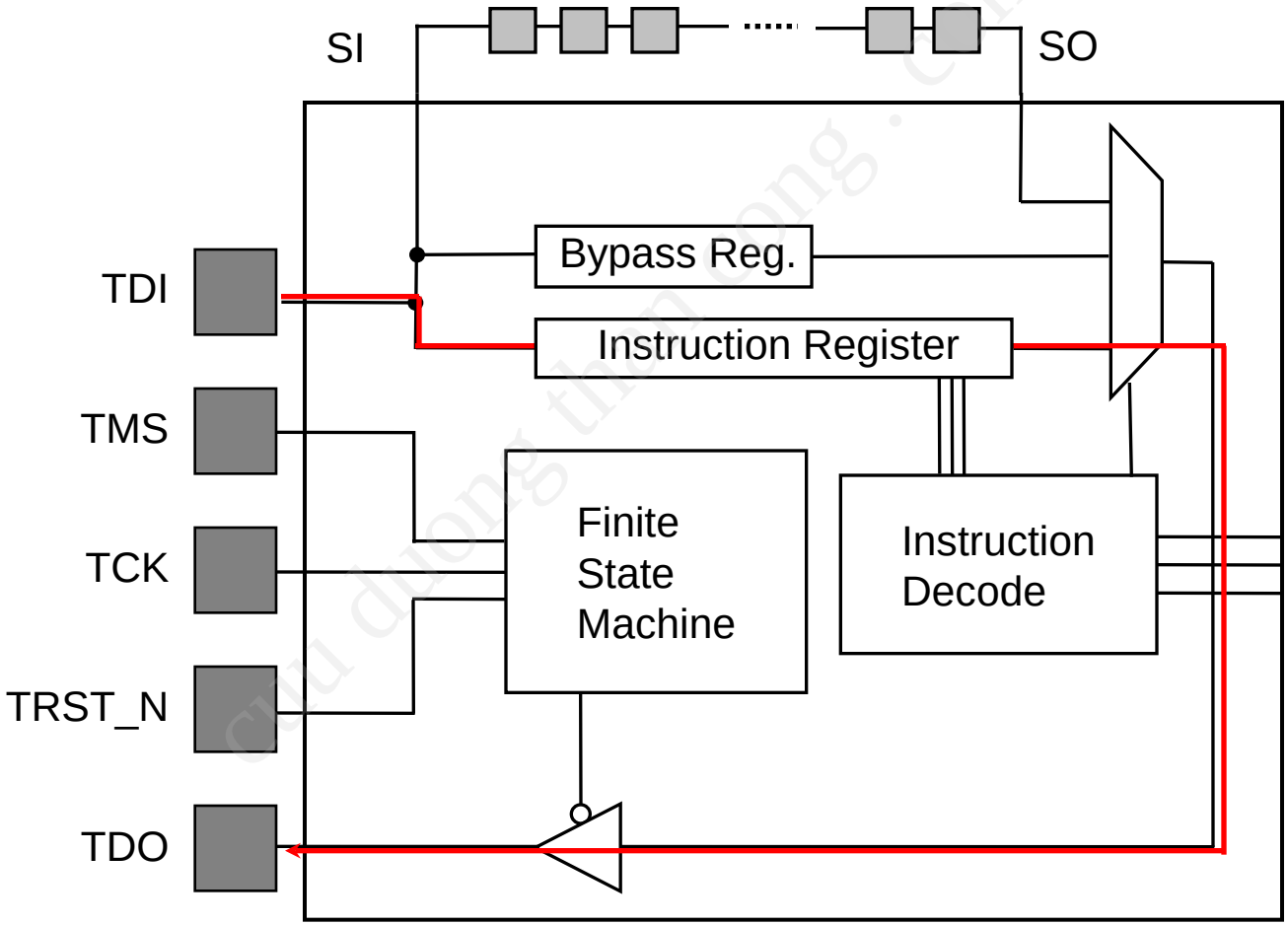
SAMPLE/PRELOAD: Start



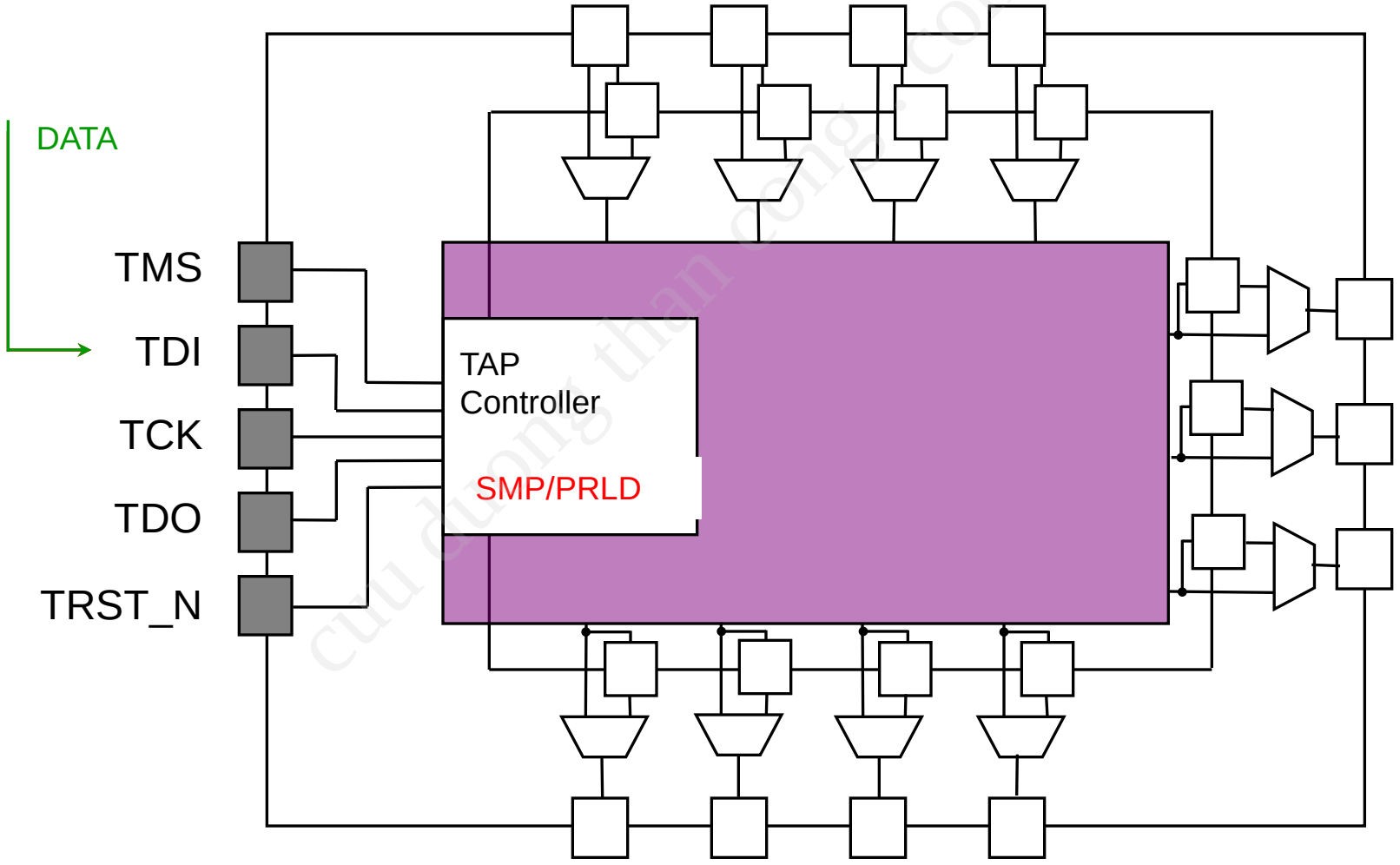
SAMPLE/PRELOAD: State diagram



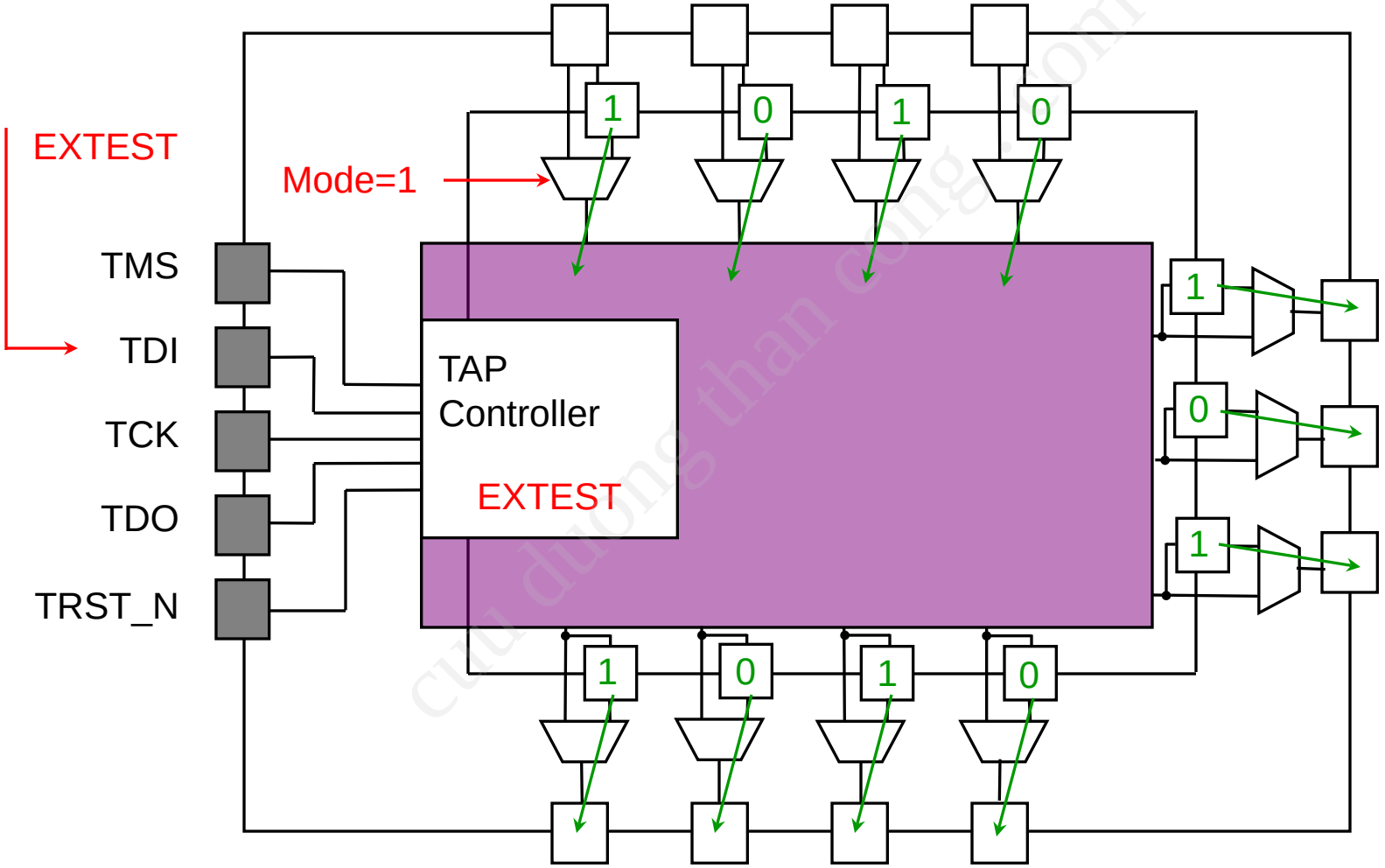
Instruction Register Data Path



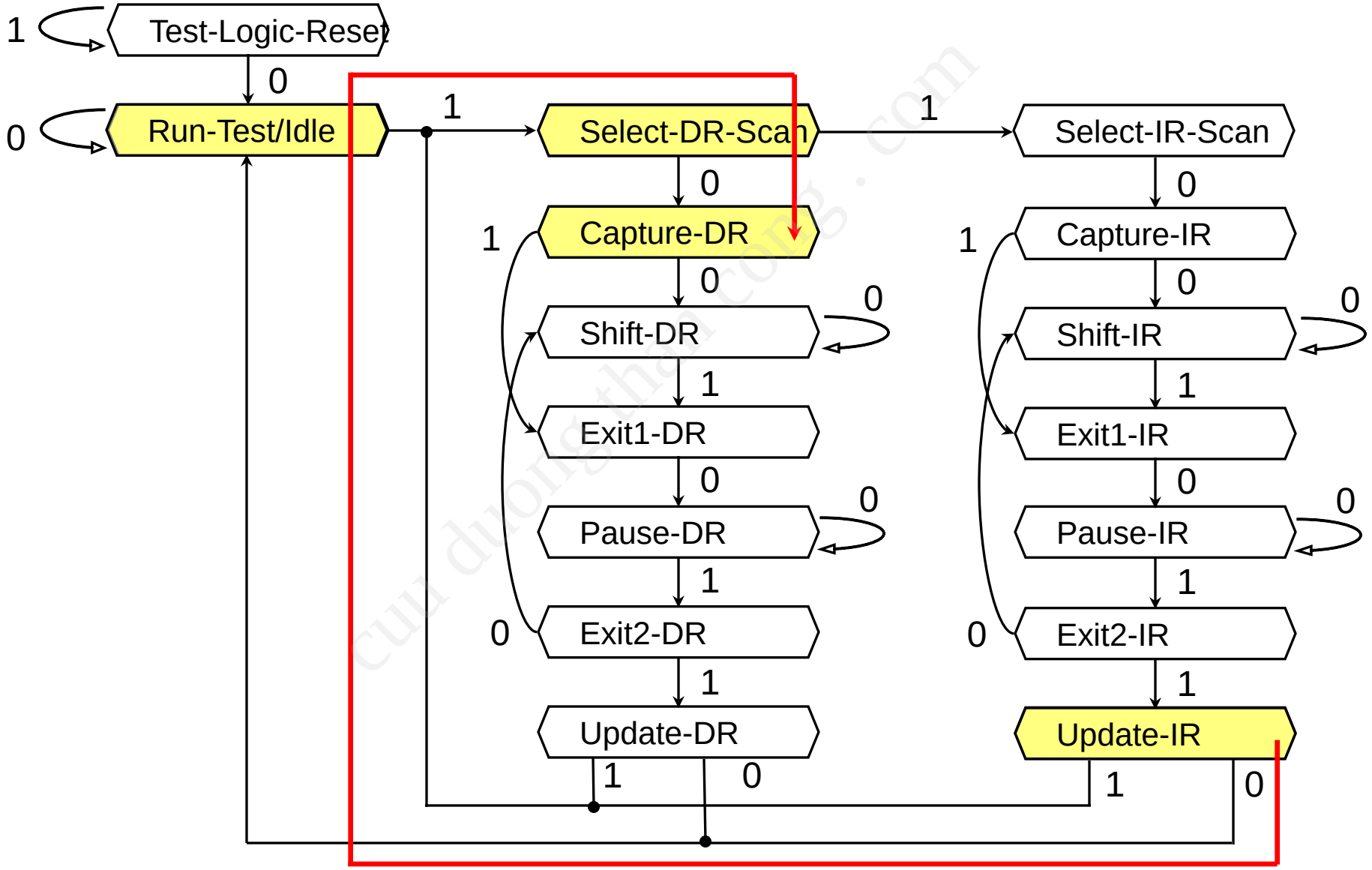
SAMPLE/PRELOAD: UpdateIR



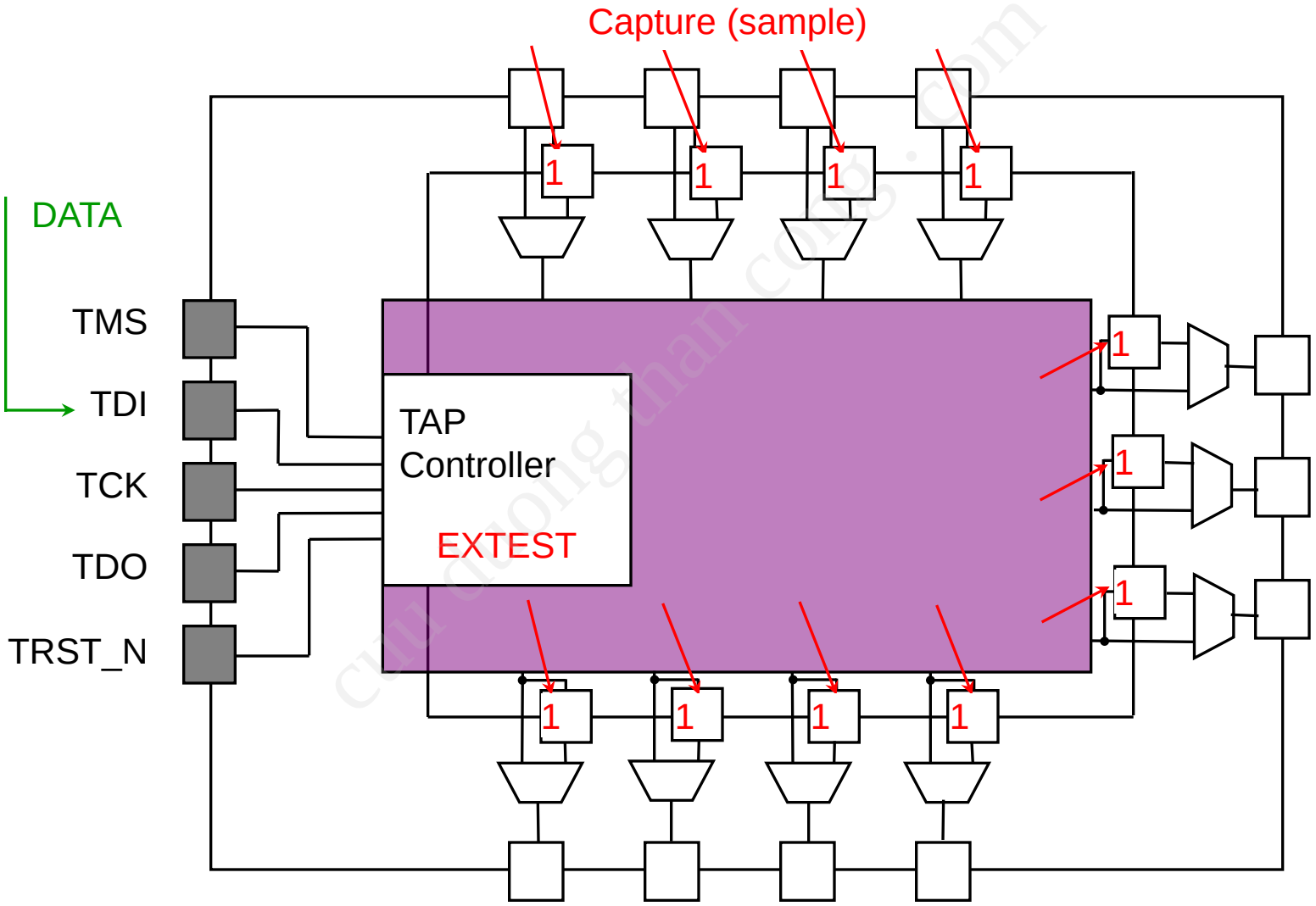
EXTEST: UpdateIR



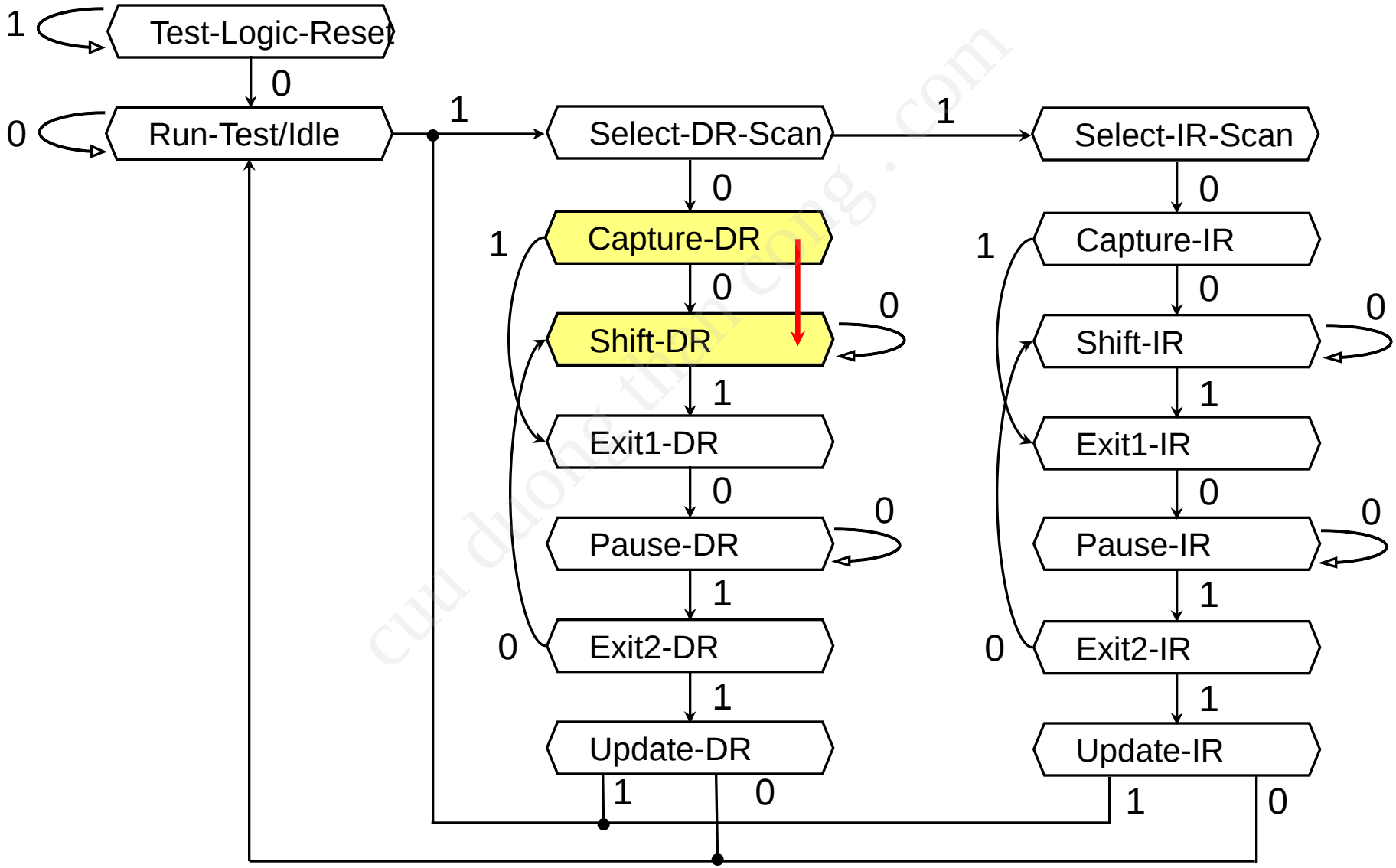
EXTEST: UpdateIR



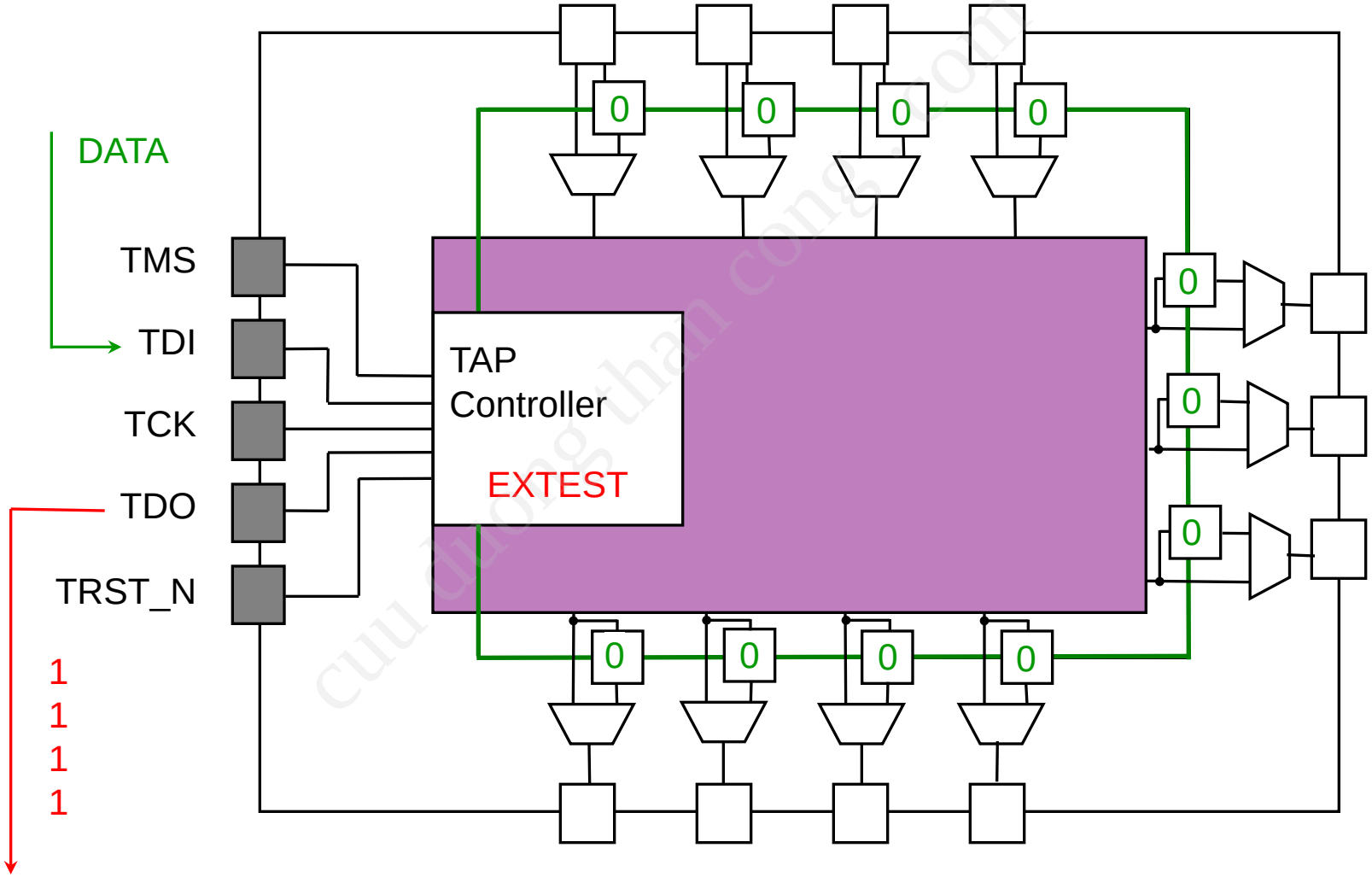
EXTEST: CaptureDR



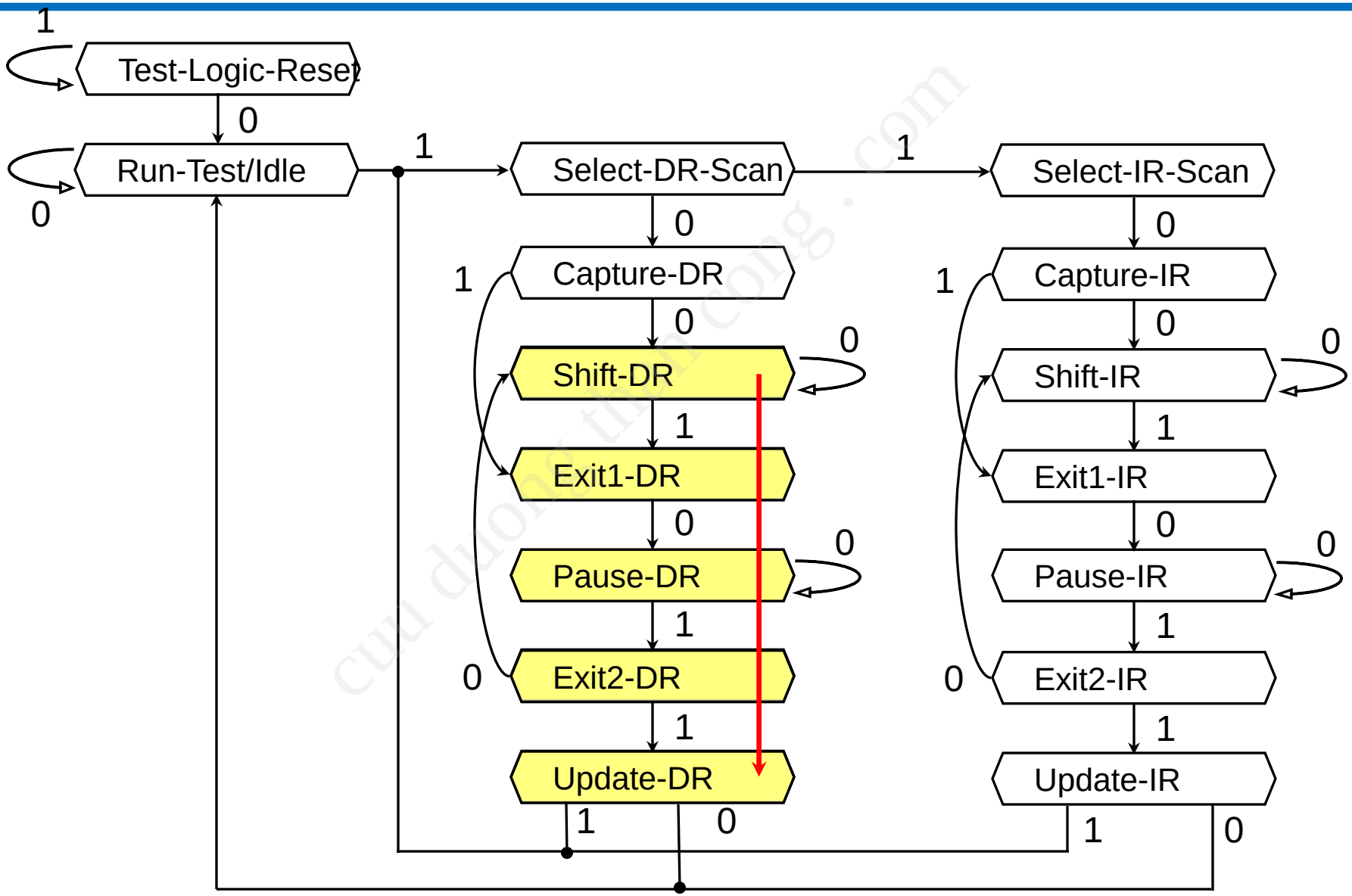
EXTEST: CaptureDR



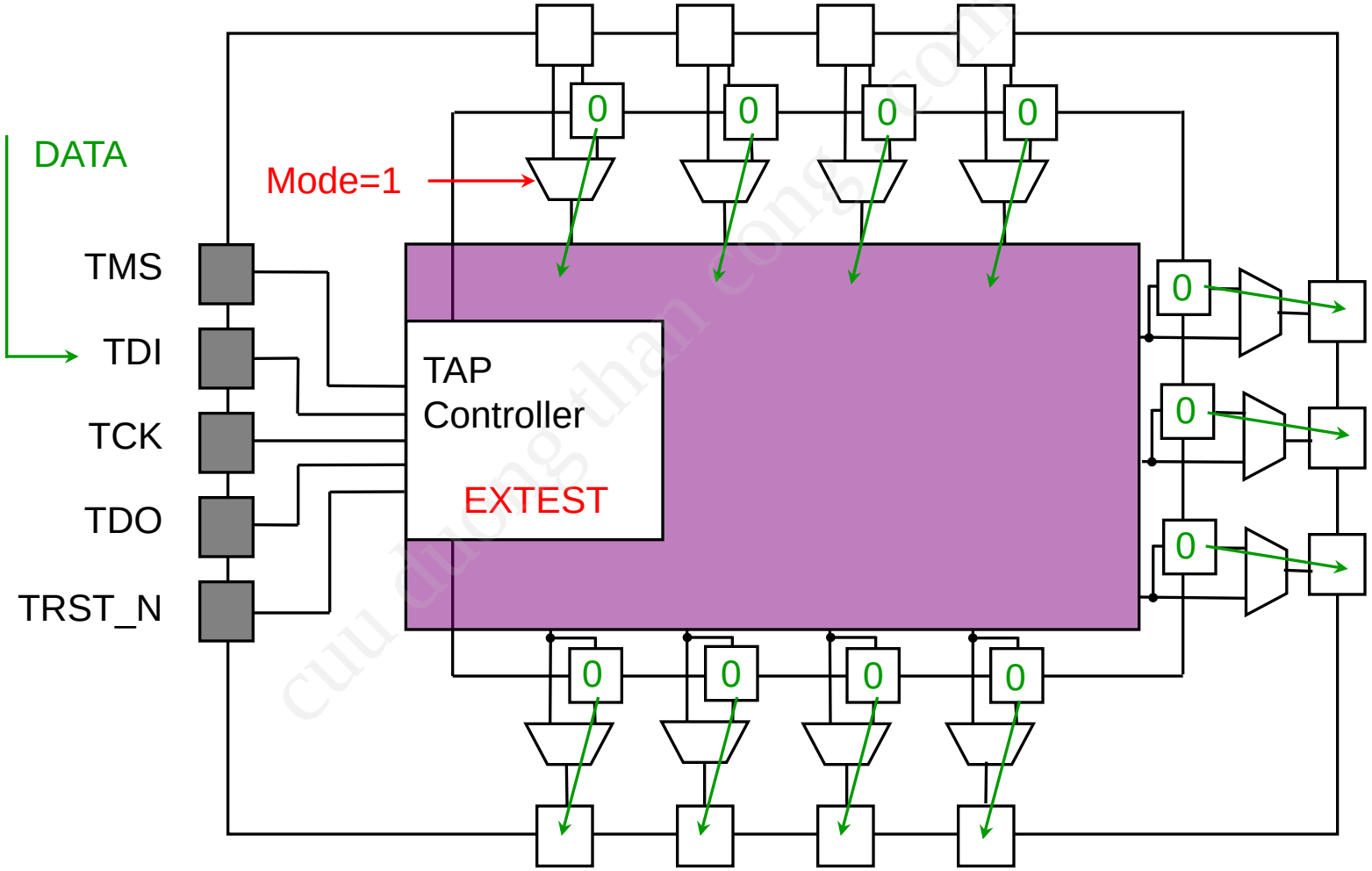
EXTEST: ShiftDR



EXTEST: ShiftDR



EXTEST: ShiftDR

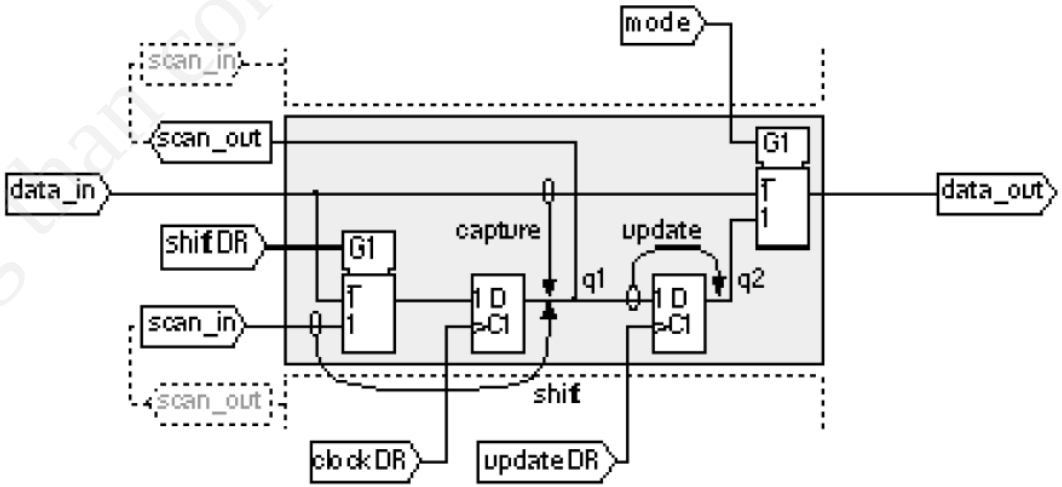


Boundary-Scan Documentation

- IEEE Standard:
  - IEEE Std 1149.1-1990 & 1149.1a-1993:
    "IEEE Standard Test Access Port and Boundary-Scan Architecture"
  - IEEE Std 1149.1b-1994:
    "Supplement to IEEE Std 1149.1-1990 …." (BDSL)
  - IEEE Std 1149.1-2001
- "The Boundary-Scan Handbook", Second Edition (1998), by Ken Parker

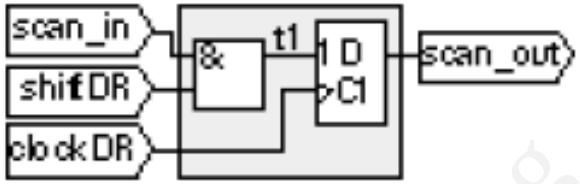
DR Cell

```
Entity DR_cell is
port (mode, data_in, shiftDR, scan_in, clockDR, updateDR: BIT;
        data_out, scan_out: out BIT);
End DR_cell;
Architecture behave of DR_cell is signal q1, q2: BIT;
Begin
  CAP: process (clockDR)   begin
    if clock_DR = '1' then
      if shiftDR = '0' then q1 <= data_in;
      else q1 <= scan_in;
      end if;
    end if;
End process;
UDP: process (updateDR)  begin
   if updateDR = '1' then q2 <= q1;
   end if;
End process;
Data_out <= data_in when mode = '0' else q2;
Scan_out <= q1;
End behave;
```

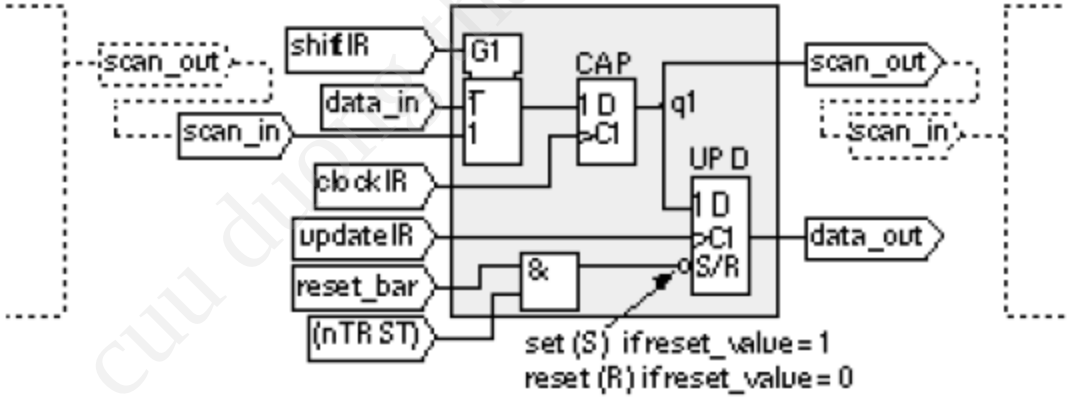


Class Assignment

1. Design Bypass (BR) cell in Verilog/VHDL

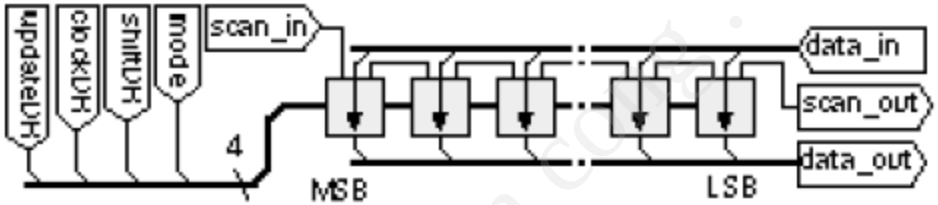


2. Design Instruction register cell (IR_cell) in Verilog/VHDL



Class Assignment

3. Design a boundary-scan register (BSR) as the below block diagram



4. Design TAP controller state machine in Verilog/VHDL