(Design Vulnerabilities in Sequential Circuit)

Contents

- Trust in System Design
- Vulnerabilities in Sequential Circuit Design
- Simple Attacks on FSM
  - Random walk attack for finding backdoor
  - Countermeasures to Prevent hardware Trojan

Trust in System Design

- **Trust can be defined as follows:**
  - When a sequential system is specified as an FSM, it is trusted if the FSM makes correct transitions from the current state to the next state and produces correct outputs based on the input values.
  - Consequently, if an FSM is trusted then all of its equivalent FSMs will also be trusted.

- It is observed from the sequential circuit design that additional states and transitions will be introduced when the design has don't care terms for next state or an output is not defined.

- Logic design tools uses these don't care conditions to optimize the design for performance improvement, area reduction and power efficiency.

- The circuit implementation will generate deterministic next states and output for each of the don’t care conditions which is a cause of untrust as the deterministic states may have access to protected states.

Trust in Circuit/System Design(contd..)

- An implicit violation of trust also comes from the nature of digital logic implementation.

- When the original FSM has ***n*** states after state minimization, it will need a minimum of ***k = [log2(n)]*** bits to encode these states. Some encoding schemes that target other design objectives (such as testability and power) may use even longer codes. When ***n*** is not a power of ***2***, which happens most of the time, those unused codes will introduce extra states into the system.

- All transitions from those extra states will be treated as don't care transitions during logic synthesis, introducing uncertainty about the trust of the design and implementation of the FSM.

- **A sequential system will have a trusted logic implementation from the traditional synthesis flow if and only if:**
  - The system is completely specified.
  - The number of states after state reduction is a power of 2.
  - Code with the minimal length is used for state encoding.

Trust in Circuit/System Design: Example

- Consider the given FSM as shown in the figure below with controlling input ***x***:

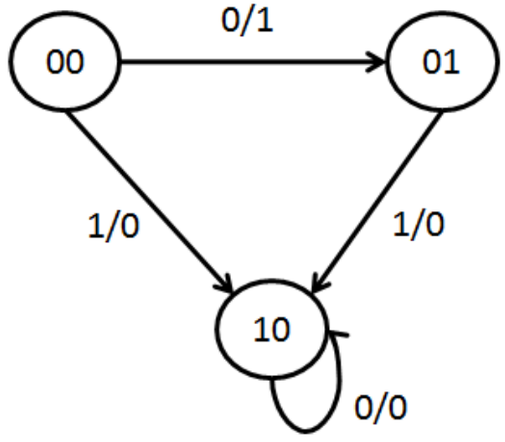

Fig. Required FSM

Current State		Input	Next State	
A	B	x	A	B
0	0	0	0	1
0	0	1	1	0
0	1	0	---	---
0	1	1	1	0
1	0	0	1	0
1	0	1	---	---

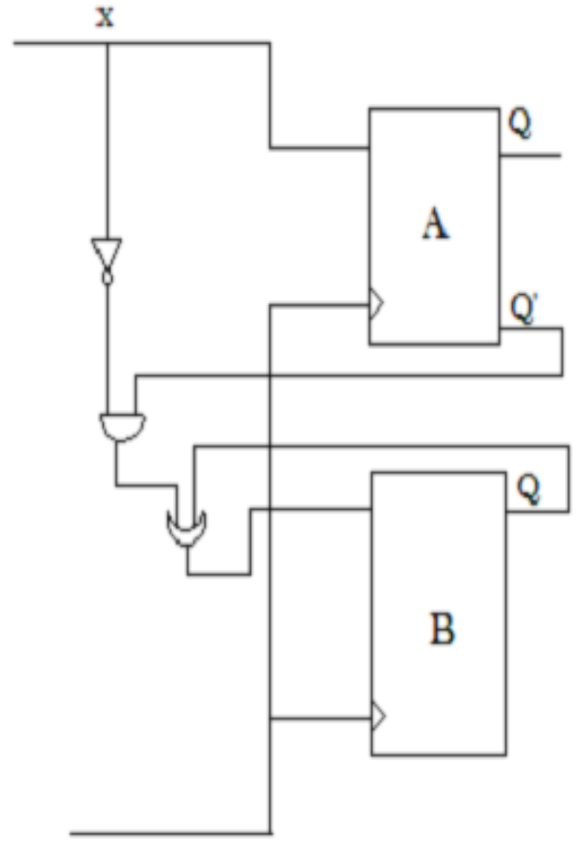

Fig. Delivered design

What is the next state for 0 1 0 and 1 0 1 ?

What happened when Input of Flip flops is 1 1 ?

Trust in Circuit/System Design: Attack Scenarios

- **Two attack scenarios are possible on sequential system depending upon what the adversary can access.**
  - **Case-I:** The attacker can only access the logic implementation of the system. The attacking objective is to gain access to the states that are not accessible as specified in the original specification.
  - This attack can be simplified as finding an equivalent FSM to find paths to inaccessible states in original FSM.
  - **Case-II:** The adversary gets hold of the original system specification, in the format of FSM and wants to establish a path to reach a certain unreachable state without being detected.
  - In this case, the attacker can implement such a path into the design. However, the challenge is how to disguise the secret path.

Trust in Circuit/System Design: Vulnerabilities Example

- **Let us fill in the blanks of the state transition table using the logic expression/design obtained after synthesizing original FSM design.**

Current State		Input	Next State		Flip-flop Input		Output
A	B	x	A	B	TA	TB	out
0	0	0	0	1	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	0	1	1
0	1	1	1	0	1	1	0
1	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0

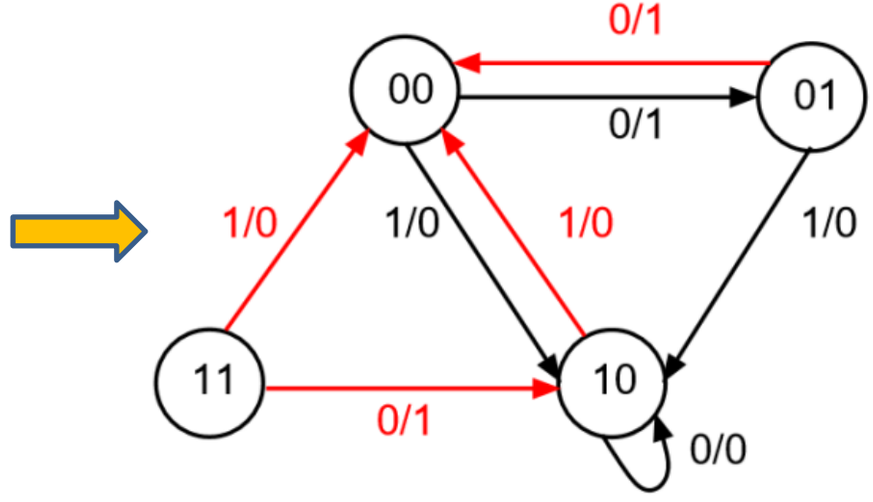

Fig. Untrusted delivered design

There are backdoors,
They create vulnerabilities

Trust in Circuit/System Design: Vulnerabilities Example

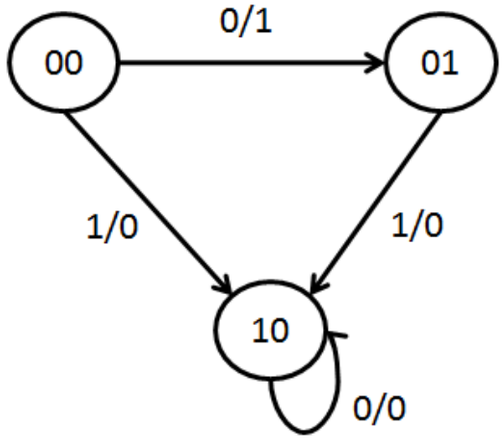

Fig. Required FSM

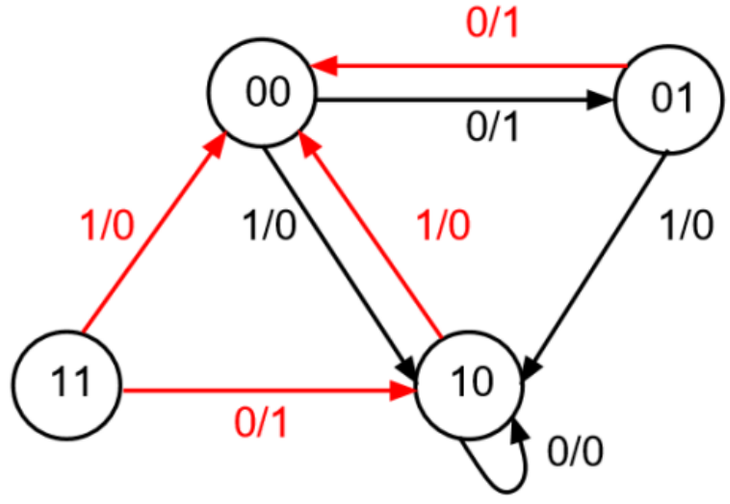

Fig. Untrusted delivered design

- The **state 00** which was previously in original FSM **inaccessible** by all other states, but now it is accessible by 01 and 10 (possible backdoors).

- The **state 11** has **complete access** to 00 and rest of the states of the FSM through 10(possible place for Trojan)

Simple Attacks on FSM

- The adversary is aware of the vulnerability of the logic implementation of the FSM following the traditional design flow. Therefore, he can launch the **random walk attack** and to gain unauthorized access to states.

- In this attack, the adversary will try random input sequences. If it leads to the discovery of a previously safe state (a state that cannot be reached by the adversary according to the design specification), the attack will be successful.

- This is possible because the FSM synthesis tools will assign values to the don't care transitions in order to optimize design objectives such as delay, area, or power.

- These added transitions may make some of the safe states reachable from states that do not belong to their starting states set and therefore making them unsafe.

Finding Backdoors in a Design: Example

- **Who can reach state 00 ?**
  - Required: S(00) = ϕ
  - Designed: S(00) = {**00, 01, 10, 11**};
    (Every one can get access to state **00**)

- **Random Walk Attack**

  In the given design/system:

  1. *Start from a random state*
  2. *Give a random input*
  3. ***if*** *(new state == 00)*
     - *successful attack; break;*
  4. ***else***
     - *goto step 2.*

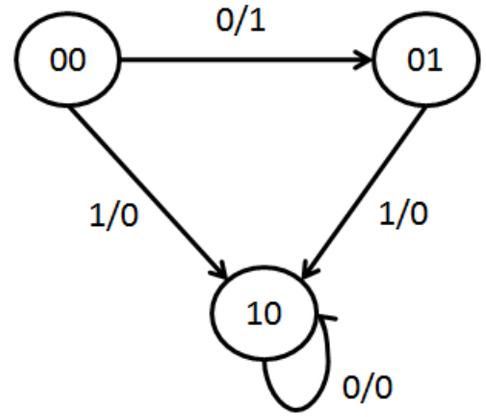

Fig. Original FSM

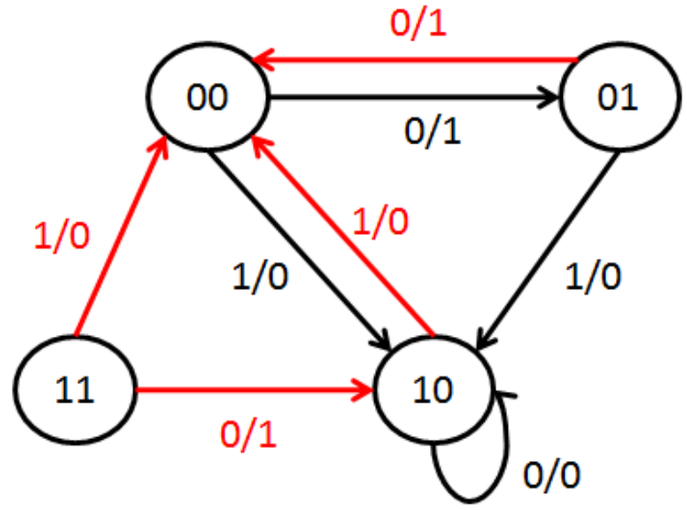

Fig. Delivered FSM

Simple Attacks on FSM (Cont..)

- The adversary has the original FSM specification of the system before it is synthesized. They want to access a **protected state(v)** from an unauthorized additionally **added state(u).**

- **The adversary can simply do the following:-**
  - Check whether there is any don't care transition from **u**; if so, attacker simply makes **v** as the next state for that transition. This will give attacker an unauthorized access to state **v** in the logic implementation of the system.
  - If the state transitions from state u are all specified, attacker can check whether there are any don't care transitions from a vertex/state that belongs to **R(u)** or reachable sates of **u**, and try to connect that state to **v** to create a path from **u** to **v**.
  - If this also fails, then state **v** in the system is safe with respect to state **u** in the sense that one can never reach state **v** from state **u**. In this case, the attack fails.

Hardware Trojan and Countermeasures: Example

- **Hardware Trojan horse:** Adding hidden access to 00.

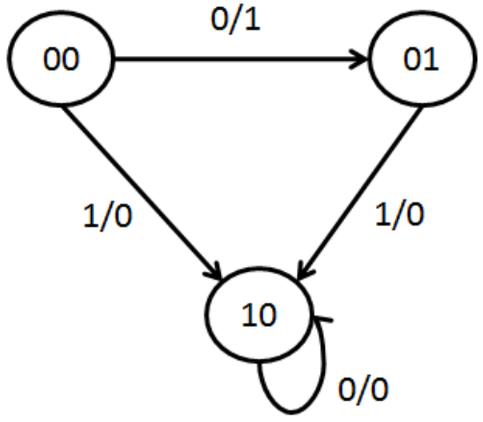

Fig. Original FSM

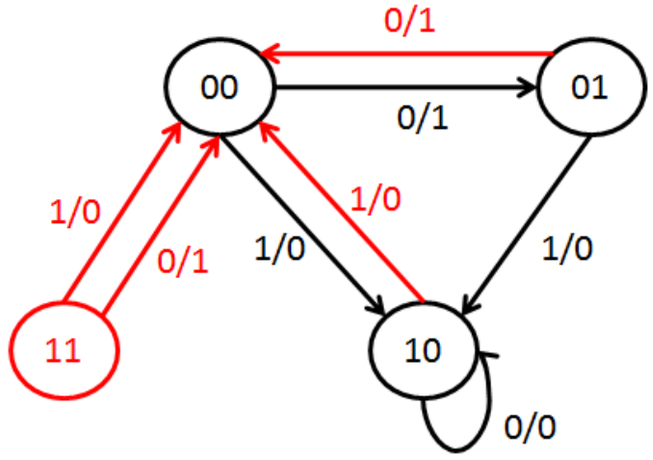

Fig. Best case scenario for Trojan

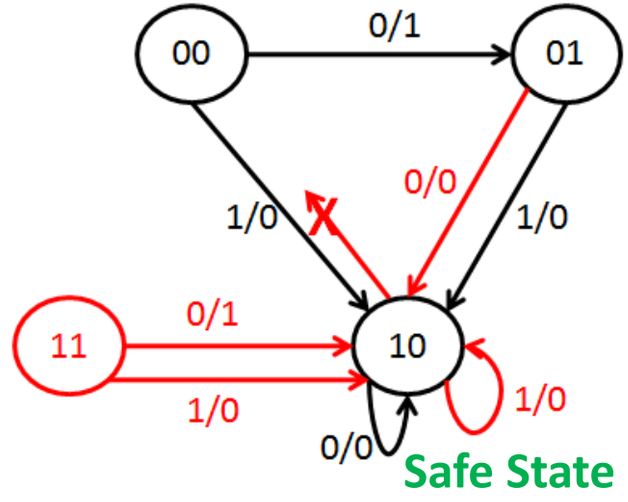

Fig. Countermeasures to Prevent from HT

- The best-case scenario for a HT is it can get access to protected states for all cases of inputs. In the figure above the best-case scenario is when 11 can access 00 for both 0 and 1.

- To prevent the HT based attacks the FSM must be encode such that any transitions from unused states must all fall into a protected state and the safe state cannot access other protected states.
- To thwart HT based attacks, all transitions from state 11 must go to state 10, and 10 now doesn't have any transition to 00.

Summary of The Class

- Trust in System Design
- Vulnerabilities in Sequential Circuit Design
- Simple Attacks on FSM
  - Random walk attack for finding backdoor
  - Countermeasures to Prevent hardware Trojan

Thank You