Automatic Formal Verification of Clock Domain Crossing Signals

Bing Li, Chris Kwok

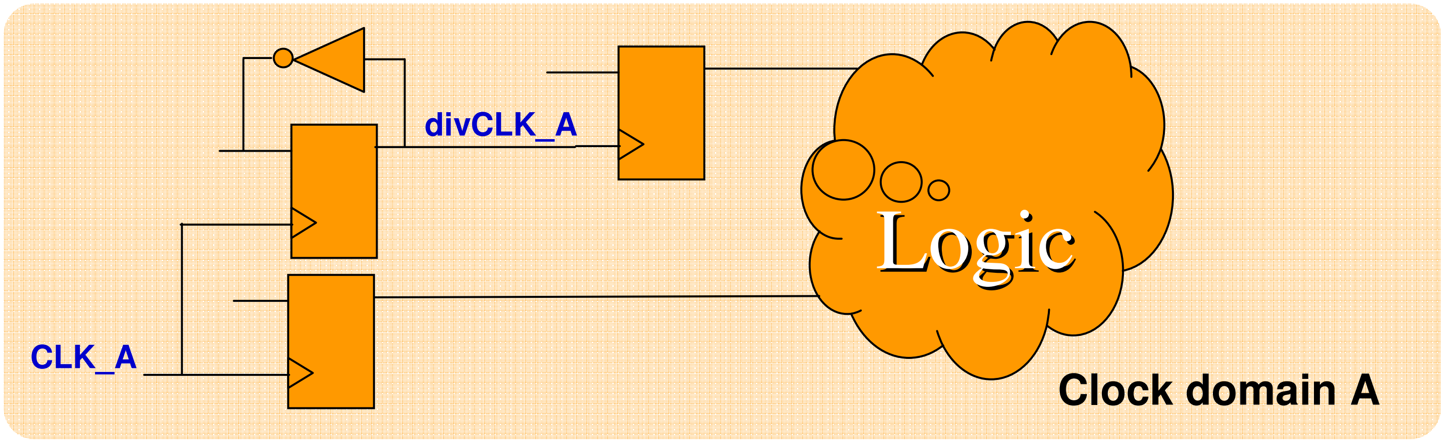
Mentor Graphics®

Outline

- CDC Problem Overview
- Current Solutions and Limitations
- Proposed Solution
- Experimental Results
- Conclusion

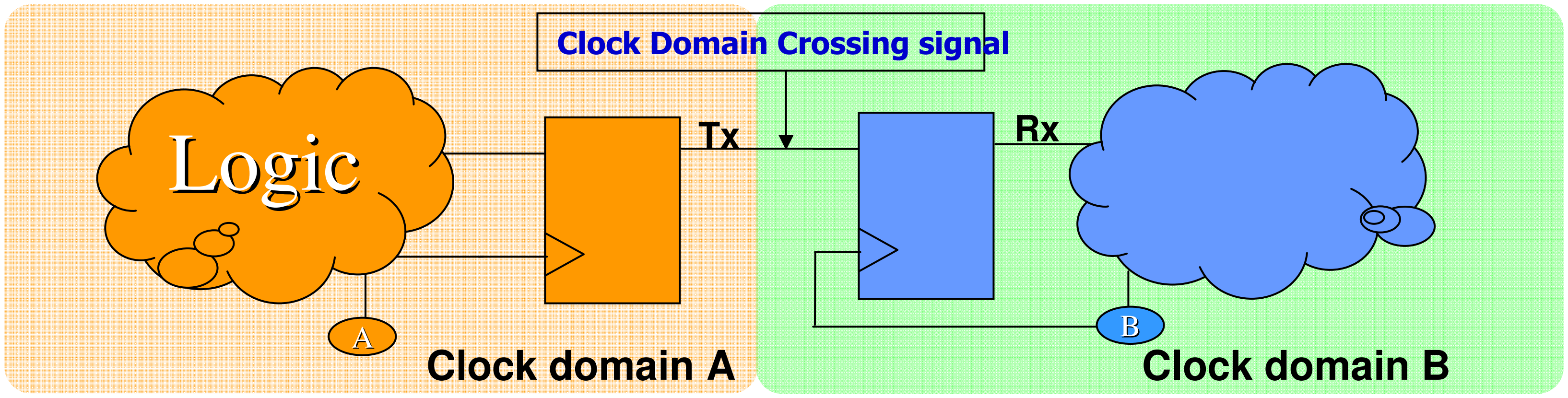
What are Clock Domains?

- All sequential logic is controlled by a *clock*
- All logic that is driven by a single clock (or inversions or divisions of that clock) defines a *clock domain*



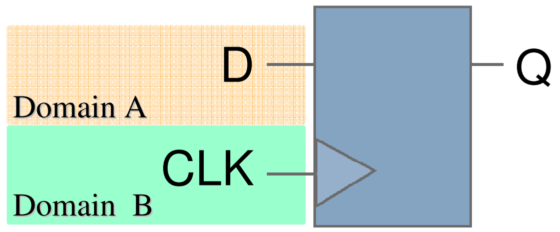
What are Clock Domain Crossings?

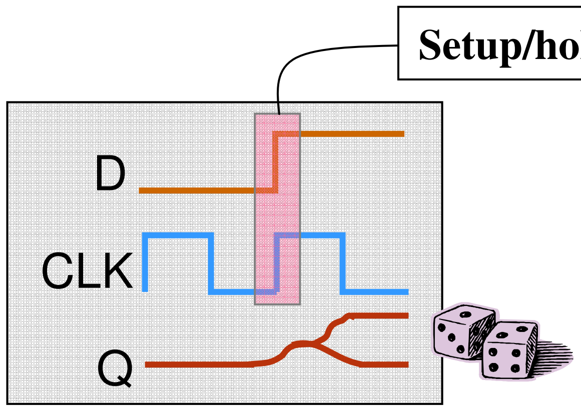
- **Signals that connect clock domains (to transfer data from one domain into the other) are called clock-domain crossings or *CDC*s**



What are the Problems with CDCs?

- **Setup/hold violations occur across clock domains**
- **When setup/hold conditions are violated, the output of a storage element becomes unpredictable**



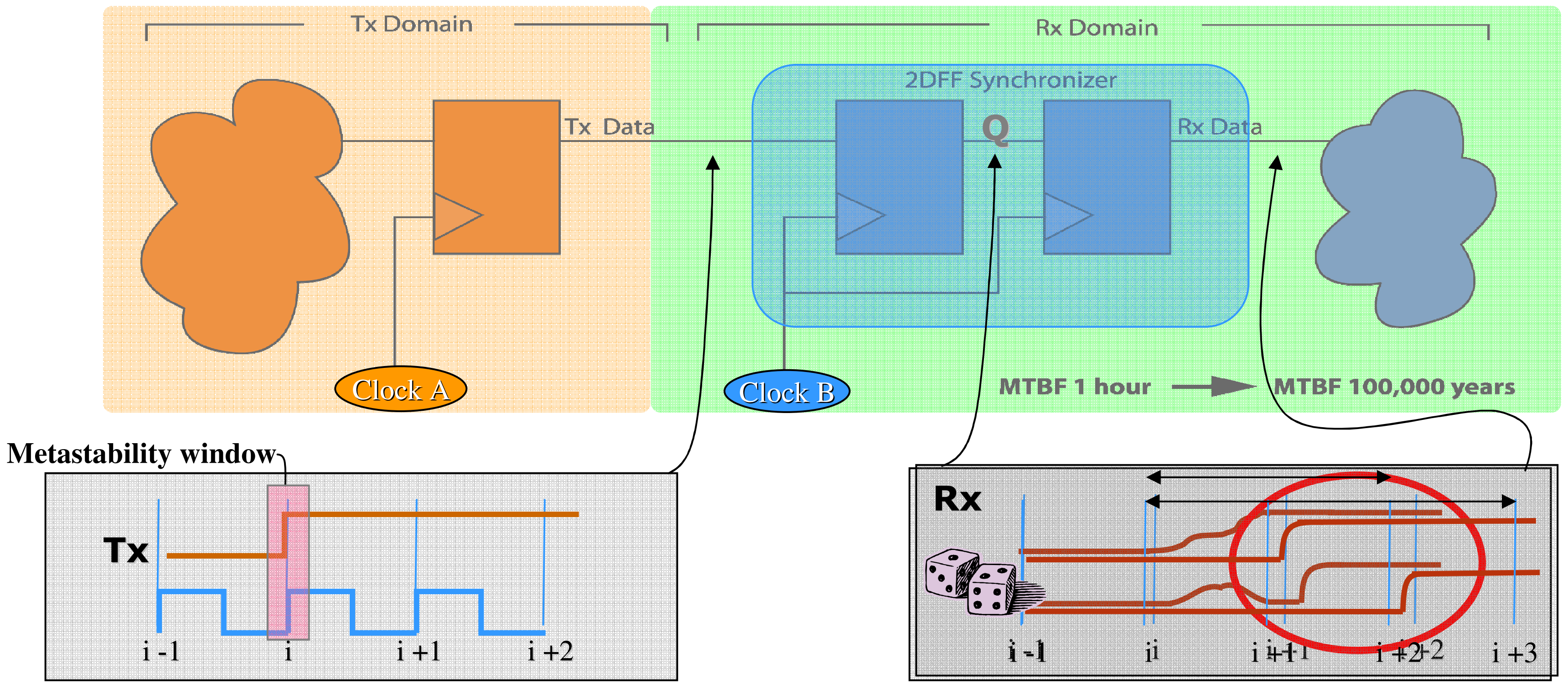


$$MTBF = \frac{1}{f_{clk} \times f_{in} \times t_d}$$

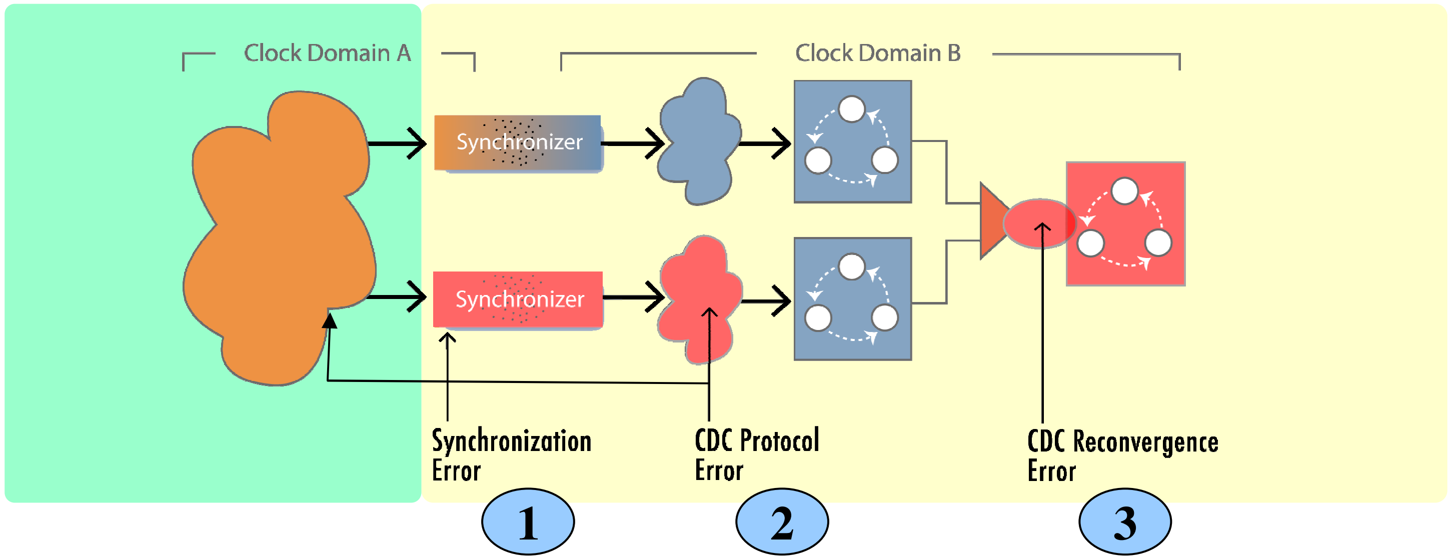
f_{clk}	= Clock Frequency
f_{in}	= Input Signal Frequency
t_d	= Duration of critical time window

- **This non-deterministic effect is called *metastability***

Designers Use Synchronizers to Isolate Metastability



What Can Go Wrong with CDCs?



1. **Missing or incorrect synchronizers**
2. **Incorrectly implemented CDC protocols**
3. **Design does not account for nondeterministic delay through synchronizers (a.k.a. reconvergence error)**

Current Solutions

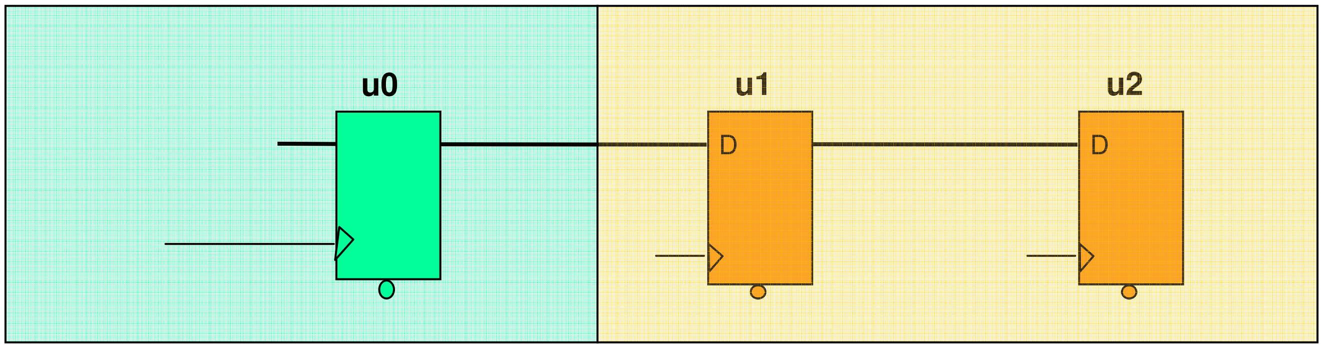
- Static Timing Analysis
  - Manual Inspection
- Structural detection
  - Not checking for protocol
- Simulation
  - Very good way to find problem,
  - Cannot prove the CDC is good by design
- Post-CDC Formal Analysis
  - Capacity and runtime problem – usually runs on block only
  - Not fully automatic

Proposed Solution

- Fully Automatic CDC formal solution that can run on chip level
- Formal analysis is run alongside with the structural analysis – no need for a post-CDC analysis step
- There are 2 parts
  - Commonly seen CDC protocols and the checks necessary to prove the protocol
  - Assertion Synthesis Algorithms

Problem #1 : Synchronizing Multiple bit signals

- Even with synchronizers, multi-bit CDC signals are not guaranteed to be correctly received by the receiving domain

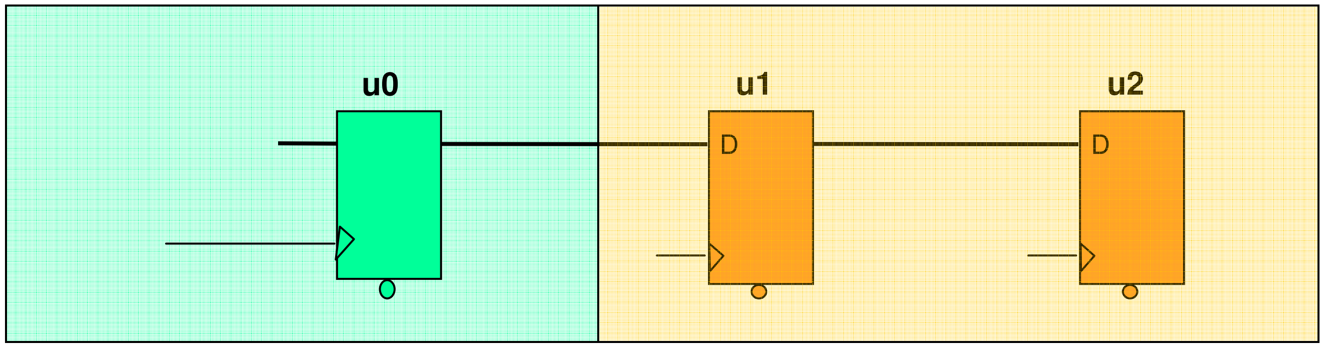


000 -> 101

000 -> 000 -> 101
000 -> 001 -> 101
000 -> 100 -> 101
000 -> 101 -> 101

Solution

- Signal needs to be gray-coded
- Gray coding the signal ensures that only valid signal reaches the receiving domain



000 -> 100

000 -> 000 -> 100
000 -> 100 -> 100

Gray Coding Check

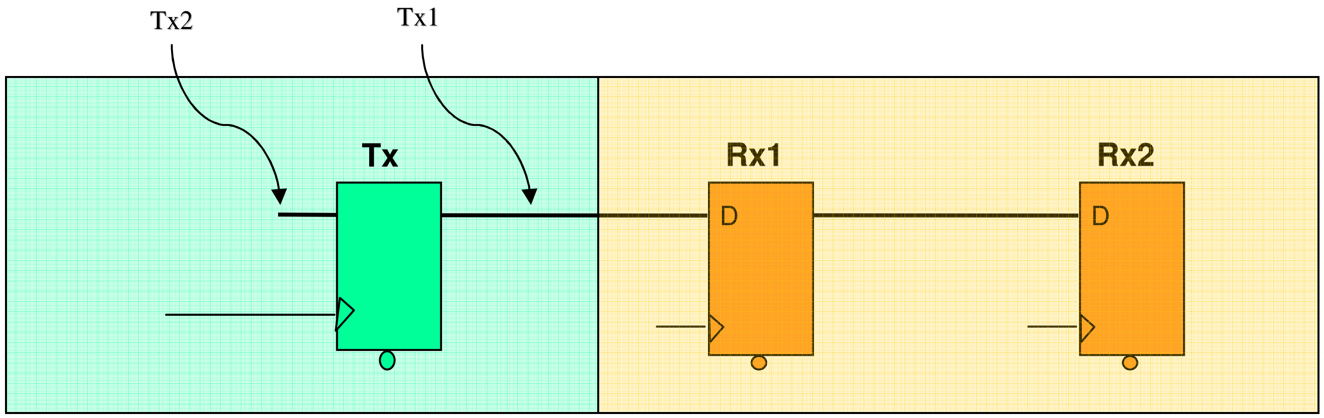
- **Verify that at most one bit changes for each transmitting clock cycle**

!(|(e & ~((~e)+ 1)))==TRUE (1)

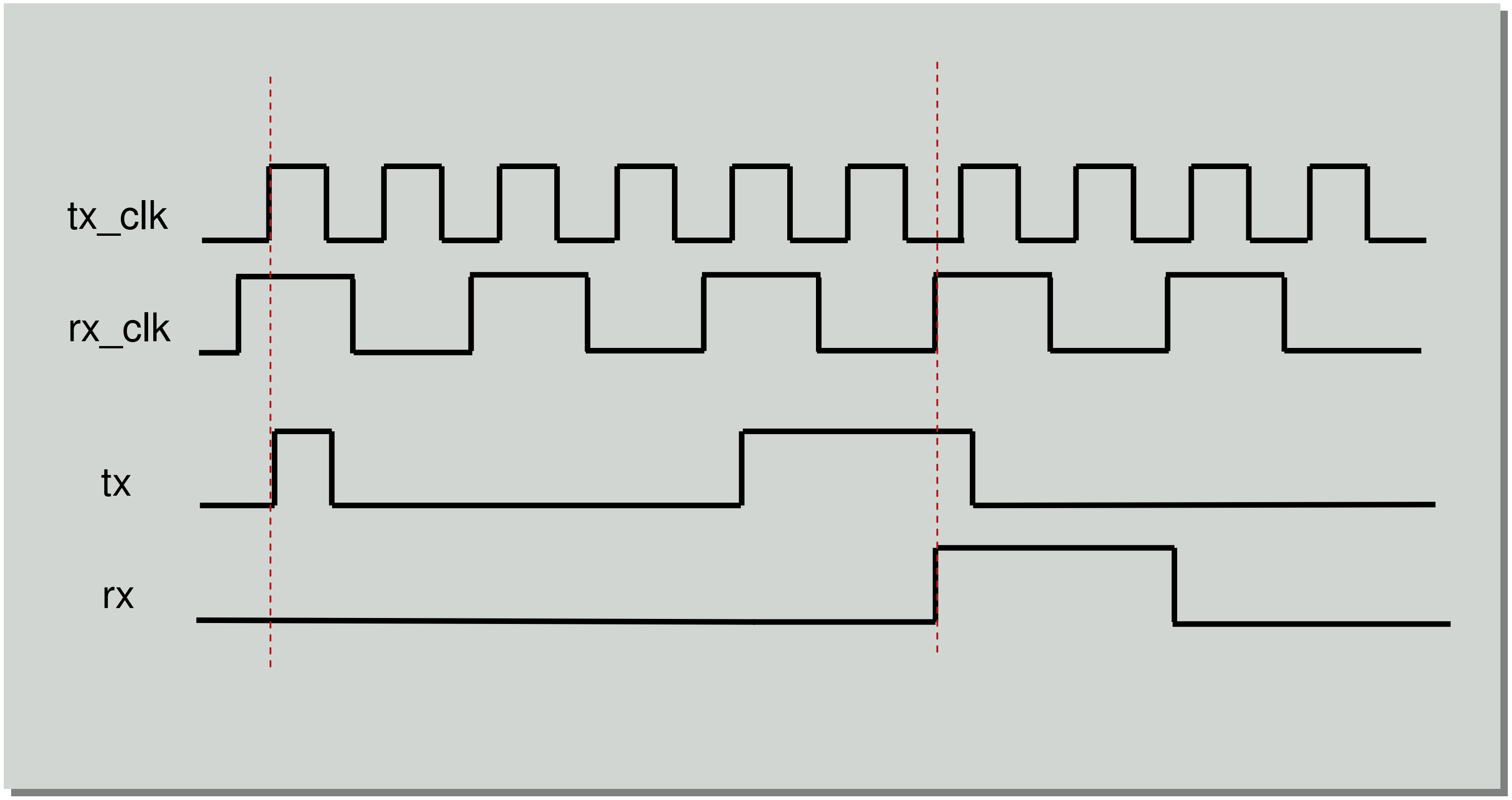
In which e = Tx1 ^ Tx2

Tx1 & Tx2 are the values of the transmitting register for 2 consecutive timeframes

(1) Is necessary & sufficient condition



Problem #2: Verify signal got sampled



Stability Check

- To guarantee that the receiving domain register samples the correct value
- Necessary for signals going from fast to slow clocks
- The transmitting signal must be stable for N clock cycles

$$N = \left\lfloor \frac{P_{rx}}{P_{tx}} \right\rfloor + 1$$

- N : number of stable cycles the transmitting signal must be stablized
- P_{rx} : Period of RX clock
- P_{tx} : Period of TX clock

Proving Stability Check

- **Definition 1.** ***If a signal remains at least N clock cycles stable for each new value, it is called an N-cycle-stable signal.***

- **Lemma 1.** ***For a CDC signal e, the sufficient and necessary condition for it to be a N-cycle-stable signal is that for any N consecutive cycles, e only changes no more than once.***

Proof #1

- **Direct translation of lemma 1 gives you:**

$$\left(\bigwedge_{i=0}^{N-1} S_i = S_{i+1}\right) \vee \left\{\bigvee_{i=0}^{N-1} \left[(S_i \neq S_{i+1}) \wedge \bigwedge_{j=0,\, j\neq i}^{N-1} (S_j = S_{j+1})\right]\right\} \quad (3)$$

Where :
S : CDC Signal
S_i : Value at time i

Complexity : O(N^2)

Proof #2

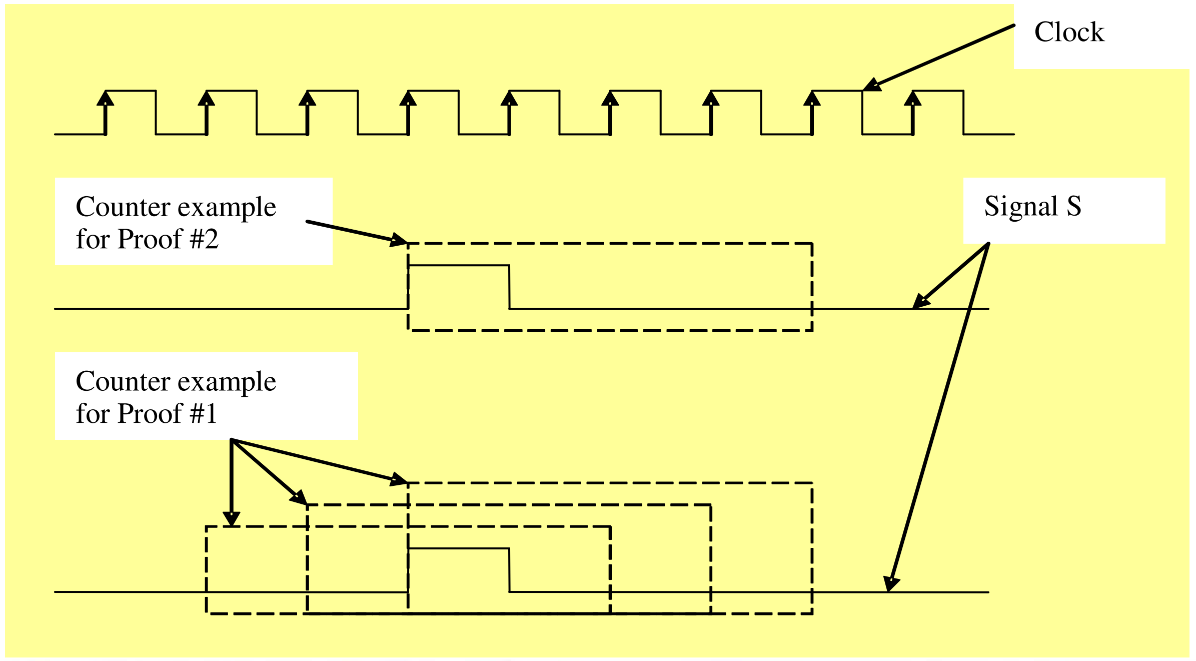
- **Force the value change happens at the beginning of N consecutive clock cycles, we got the formula below**

$$(S_0 \neq S_1) \rightarrow \bigwedge_{i=1}^{N-1} (S_i = S_{i+1})$$

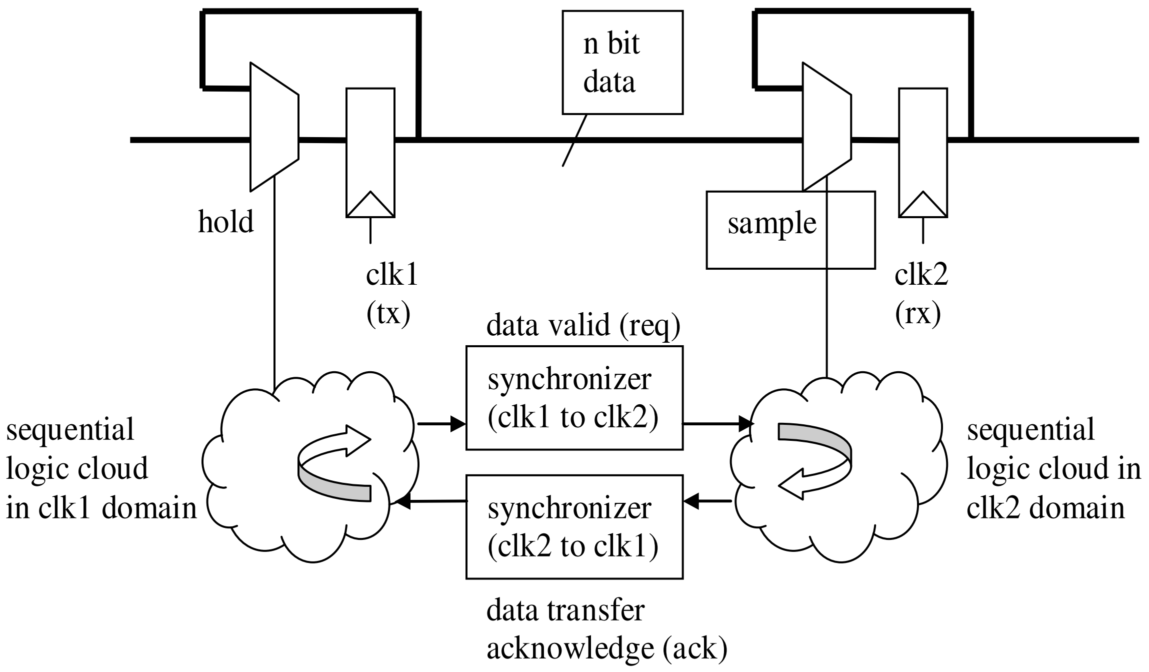
Complexity : O(N)

Proof #1 vs Proof #2

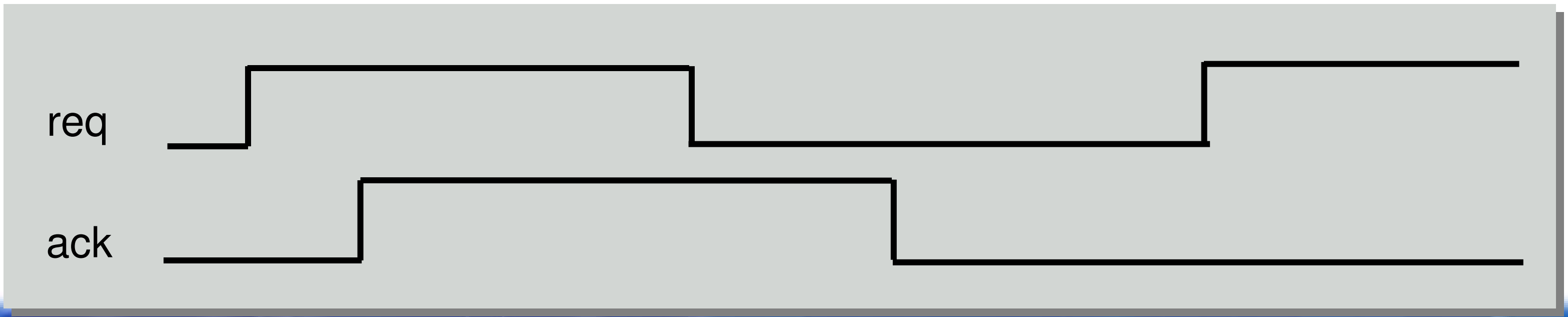
- **Proof #1 catches more bugs than Prove #2**
  - **Proof #1 catches 3 windows**
  - **Proof #2 catches 1 window**



Problem #3: Handshake Scheme

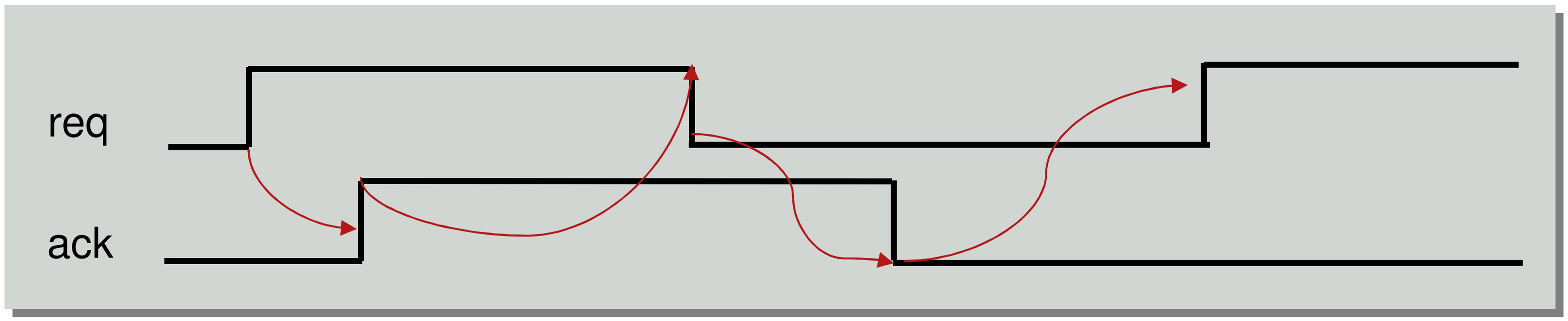


n bit data
hold
clk1 (tx)
sample
clk2 (rx)
data valid (req)
synchronizer (clk1 to clk2)
synchronizer (clk2 to clk1)
sequential logic cloud in clk1 domain
sequential logic cloud in clk2 domain
data transfer acknowledge (ack)



Handshake Check

- Checks necessary for handshake scheme
  - Once signal *req* gets asserted, it remains asserted until signal *ack* is asserted
  - Once signal *ack* gets asserted, it remains asserted until signal *req* gets deasserted
  - Signal *req* doesn't assert again until *ack* gets deasserted
  - During the assertion of *req* and *ack*, the data has to remain stable



Assertion Logic Synthesis

- **Protocols described above can be synthesized into assertion logic to be used by formal**
  - **Static-timeframe check**
  - **Dynamic-timeframe check**
- **Static timeframe check**
  - **Difference between ending timeframe and starting timeframe is constant**
- **Dynamic timeframe check**
  - **Difference between ending timeframe and starting timeframe is changing**

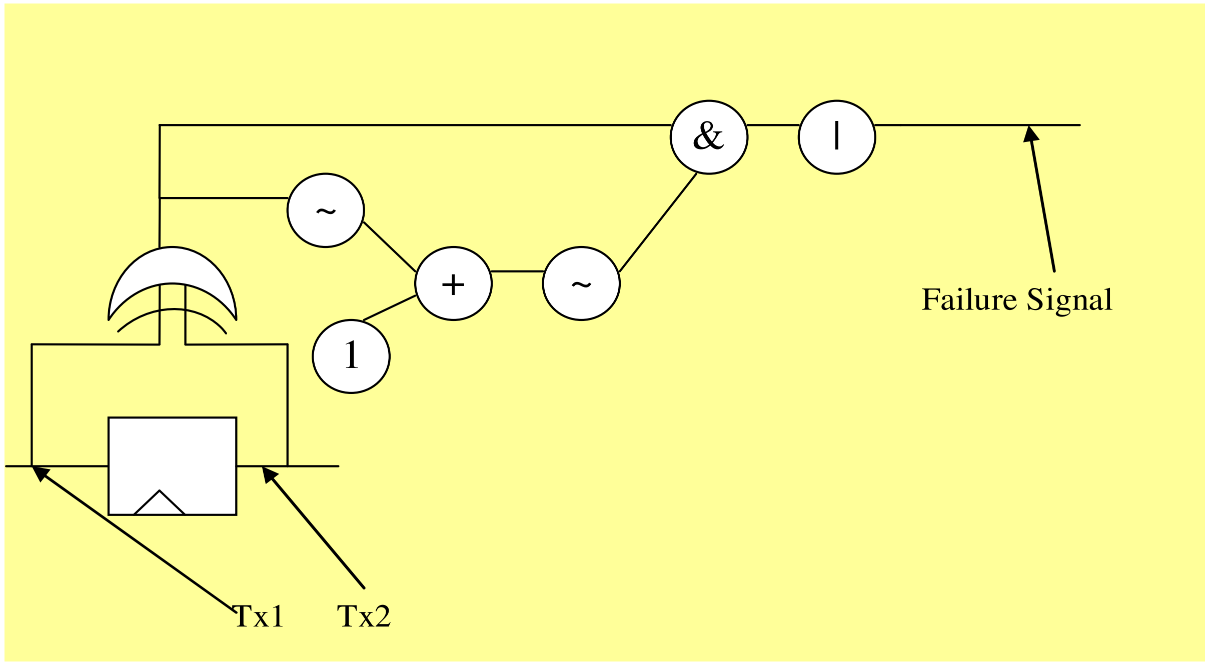
Examples

- Examples of static-timeframe checks
  - Gray-code check
  - Stability check
- Examples of Dynamic-timeframe checks
  - Handshake

Static-timeframe Assertion Synthesis

- **For each signal in different timeframes, we add registers to represent the delay between timeframes**
- **Compare these delayed signals using combinational logic**

Synthesized Assertion for Gray-Coding Check

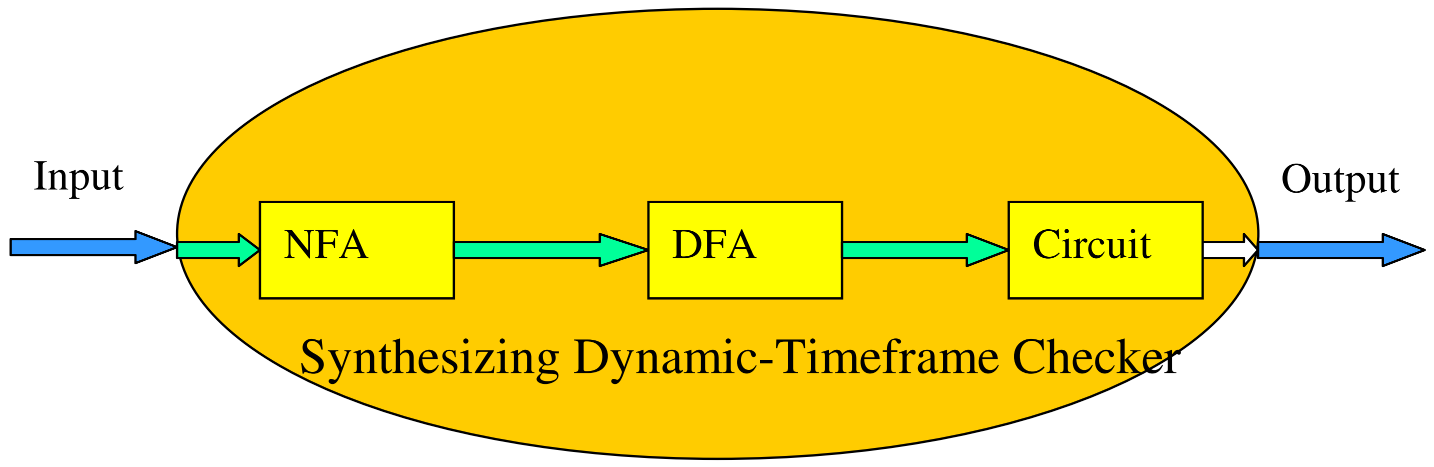


!(|(e & ~((~e)+ 1’b1)))==TRUE   (1)

In which e = Tx1 ^ Tx2

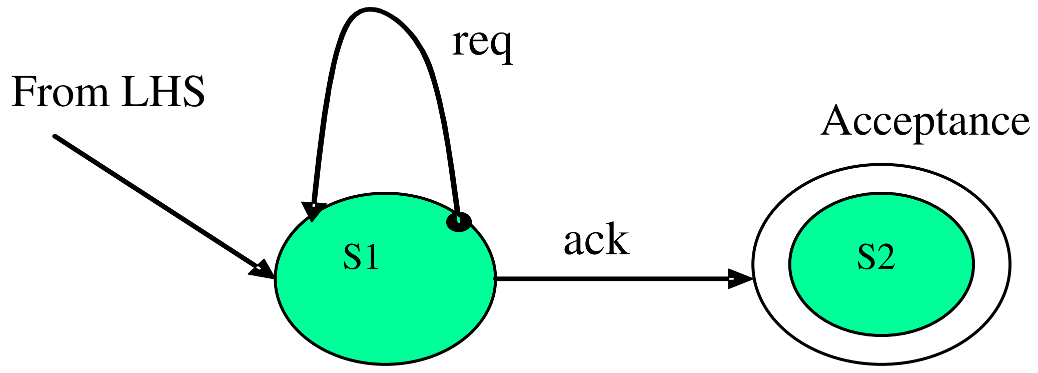
Tx1 & Tx2 are the values of the transmitting register for 2 consecutive timeframes

Dynamic-timeframe Assertion Synthesis



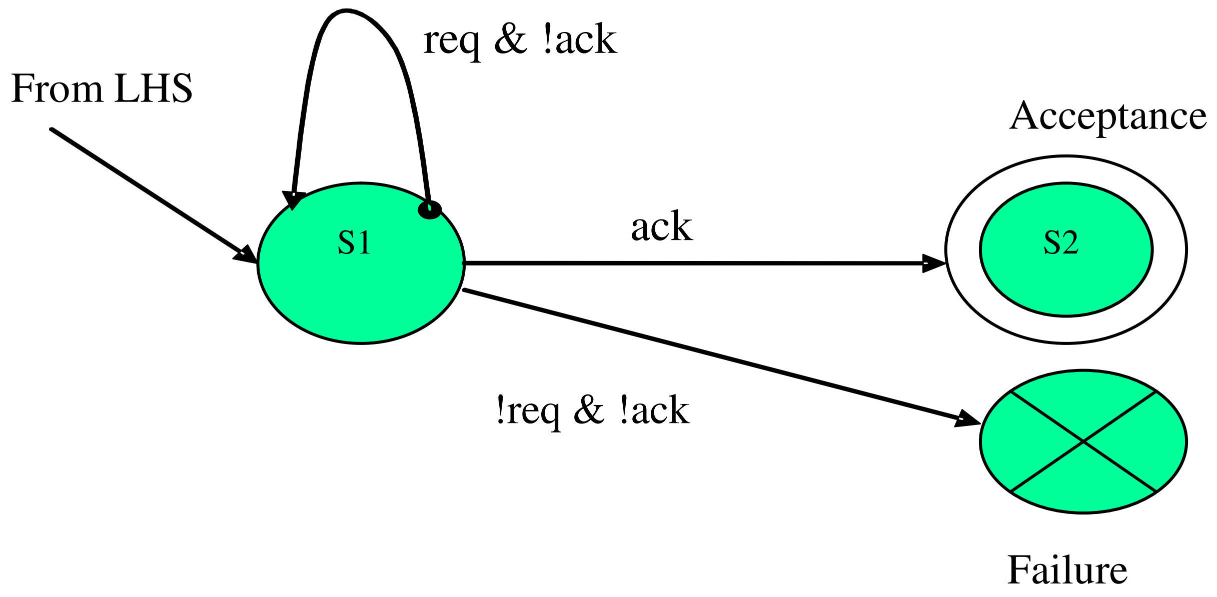
- **Transform the checks into NFA**
- **Transform the NFA into DFA**
- **Synthesize DFA into a circuit**

NFA for handshake



- **Once signal *req* gets asserted, it remains asserted until signal *ack* is asserted**

DFA for handshake check



Traditional Formal Analysis

- Flow
  - Perform static CDC analysis
  - Collect all the assertions after CDC analysis
  - Synthesizing all the assertions together with the design to form formal netlist
  - Run formal algorithm
- Problems
  - Capacity
  - Performance

Automatic Formal CDC Flow

- **During CDC static analysis, create a local circuit for each CDC property at interest**
- **Keep bring in larger circuit until budget is used up or the property is proven**
- **Any unproven property will generate assertions to run for simulation**

Automatic Formal CDC Flow

```
Formal_CDC (B, P) {
  For_each_CDC_boundary(B, $\beta$) {
      Proven = FALSE;
      If ($\beta$ needs to be verified) {
            E = Extract_local_circuit_as_an_abstraction($\beta$);
            C = Create_formal_netlist(E);
            Proven = Formal_verify(C);
      }
    If (Proven == FALSE)
           insert(P, $\beta$);
  }
}

CDC_Analysis($\Omega$) {
    B =  Structure_analysis($\Omega$);
    Formal_CDC(B, P);
    Checker_promotion(P);
}
```

Ω : Flattened Netlist

B : CDC boundary

P : empty queue, will contain all the non-proven CDC boundaries

β : CDC boundary

Experimental Results

	Post-CDC Formal		Auto formal CDC			
Testcase	**Time**	**Proven**	**Time**	**Proven**	**Total**	**Size**
testcase1	16	1	6	1	3	160
testcase2	21	5	9	5	8	176
testcase3	25	9	8	7	12	194
testcase4	89	5	13	1	7	302
testcase5	526	3	125	2	5	414
testcase6	51	6	14	1	8	315
testcase7	385	128	187	106	370	3554
testcase8	92	10	31	7	21	6997
testcase9	12509	48	2586	23	71	7648
testcase10	912	42	100	6	166	11286
Total	**14626**	**257**	**3079**	**159**	**671**	**31046**

20% 62%

Conclusion

- Discussed various CDC protocols
- Proposed a fully automatic approach to formally verify CDC protocols at chip level
- Experiments showed our new approach can prove a large portion of the assertions in a much shorter time