

L11: Major/Minor FSMs



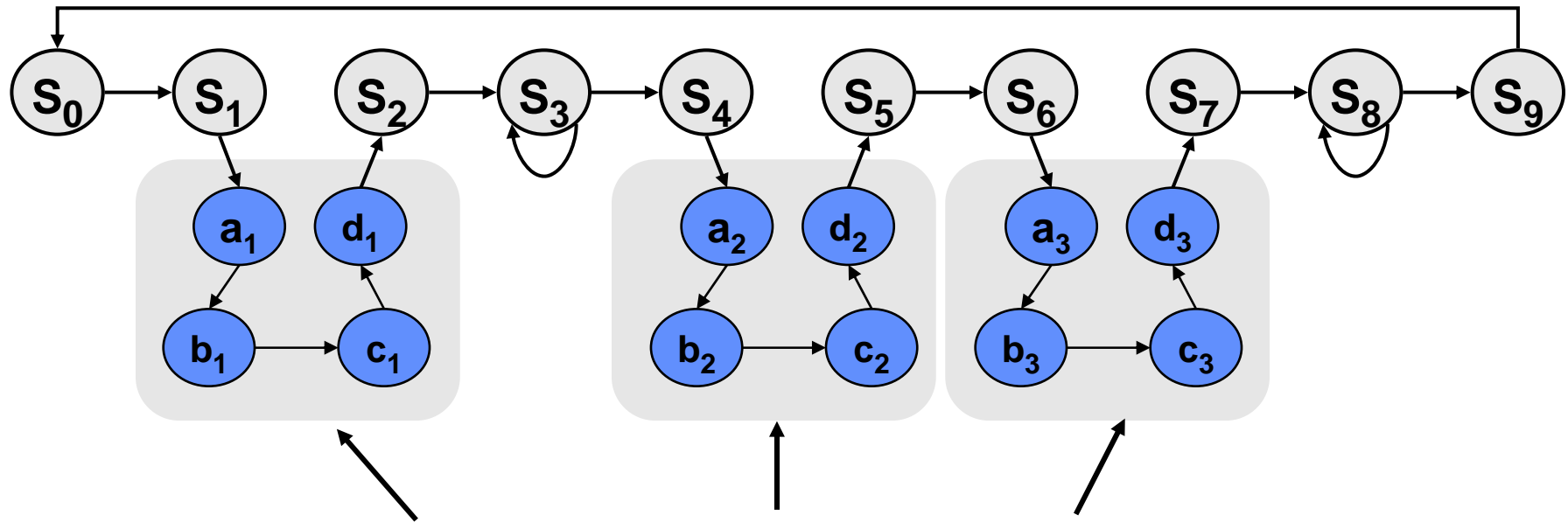
Acknowledgements: Rex Min

Lecture Notes prepared by Professor Anantha Chadrakasan

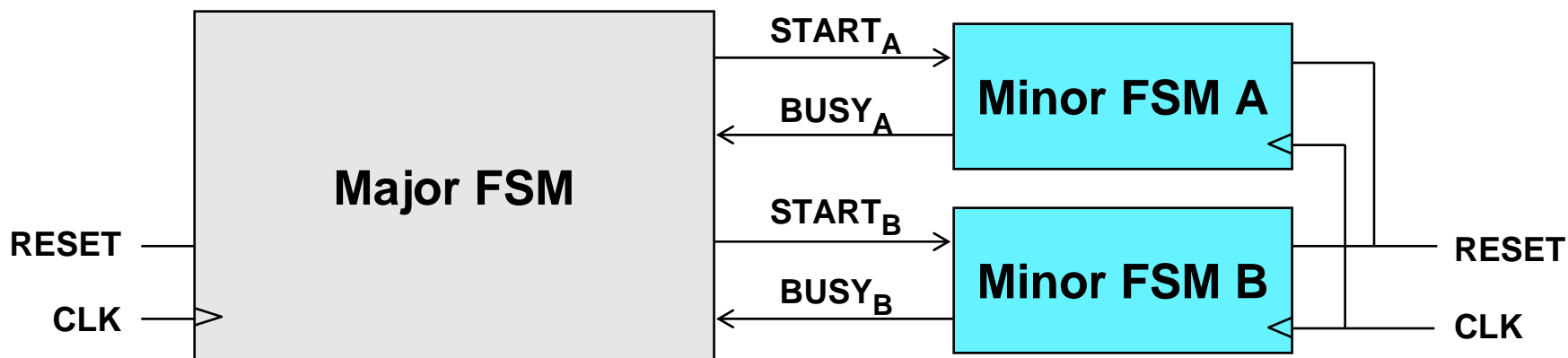
- Quiz will be **Closed Book Tuesday, March 18, 2008**, 7:30pm-9:30pm in 32-155 Covers Problem Sets 1-3, Lectures 1-9 (through Arithmetic), Labs 1-3
- Some of the topics to be covered include
 - Combinational Logic: Boolean Algebra, Karnaugh Maps, MSP, MPS, dealing with don't cares
 - Latches and Edge Triggered Registers/Flip-flops
 - Understand the difference between latches, registers and unclocked memory elements (e.g., SR-Flip Flop)
 - Different memory types: SR, D, JK, T
 - Understand setup/hold/propagation delay and how they are computed
 - System Timing (minimum clock period and hold time constraint)
 - Impact of Clock skew on timing
 - Counters and simple FSMs (understand how the '163 and '393 work)
 - FSM design (Mealy/Moore, dealing with glitches)
 - Combinational and sequential Verilog coding
 - Continuous assignments, blocking vs. non-blocking, etc.

- **Tri-states basics**
- **Dealing with glitches**
 - When are glitches OK?
 - How do you deal with glitches in digital system design? (registered outputs, appropriate techniques to gate a clock, etc.)
- **Memory Basics**
 - Understand differences between DRAM vs. SRAM vs. EEPROM
 - Understand timing and interfacing to the 6264
- **Arithmetic**
 - Number representation: sign – magnitude, Ones complement, Twos complement
 - Adder Structures: Ripple carry, Carry Bypass Adder, Carry Lookahead Adder
 - False Paths and Delay Estimation
 - Shift/add multiplier, Baugh-Wooley Multiplier (Twos complement multiplication)

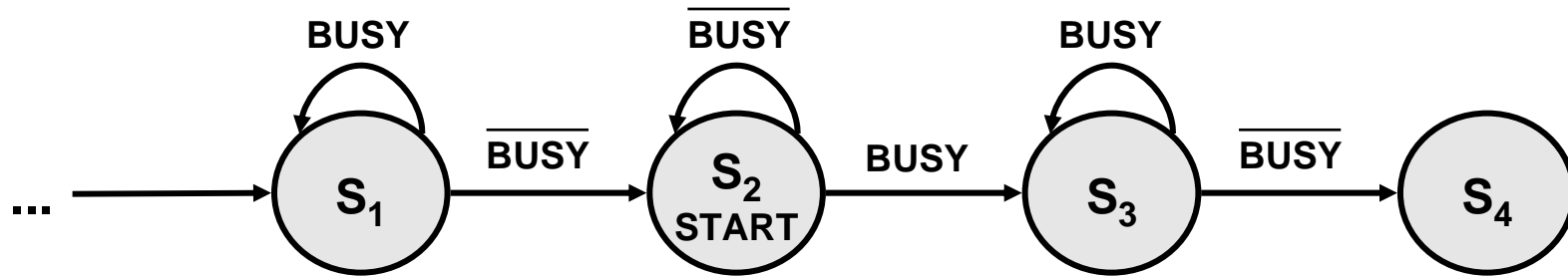
- Consider the following abstract FSM:



- Suppose that each set of states $a_x \dots d_x$ is a “sub-FSM” that produces exactly the same outputs.
- Can we simplify the FSM by removing equivalent states?
No! The outputs may be the same, but the next-state transitions are not.
- This situation closely resembles a **procedure call** or **function call** in software...how can we apply this concept to FSMs?



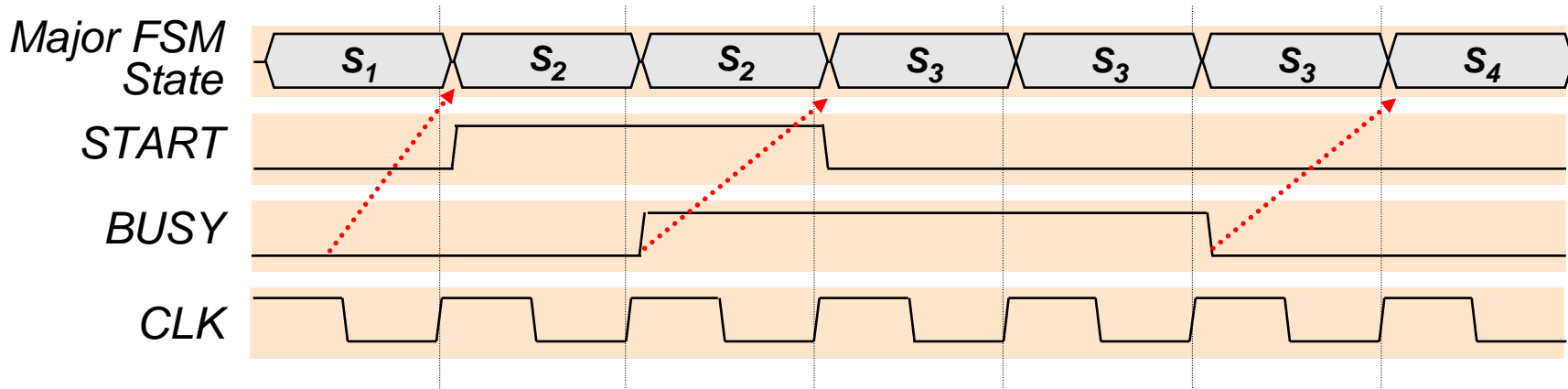
- Subtasks are encapsulated in **minor FSMs** with common reset and clock
- Simple communication abstraction:
 - START: tells the minor FSM to begin operation (the call)
 - BUSY: tells the major FSM whether the minor is done (the return)
- The major/minor abstraction is great for...
 - Modular designs (*always* a good thing)
 - Tasks that occur often but in different contexts
 - Tasks that require a variable/unknown period of time
 - Event-driven systems



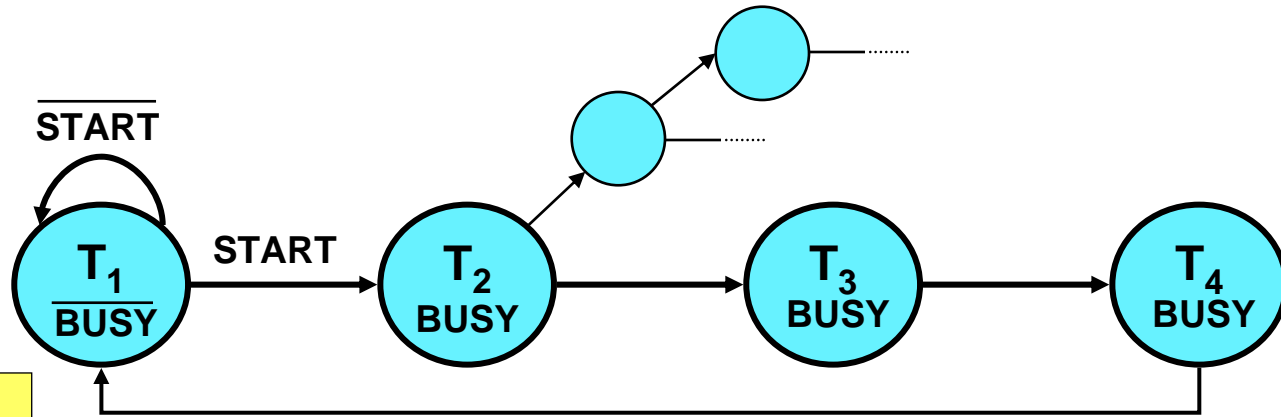
1. Wait until the minor FSM is ready

2. Trigger the minor FSM (and make sure it's started)

3. Wait until the minor FSM is done

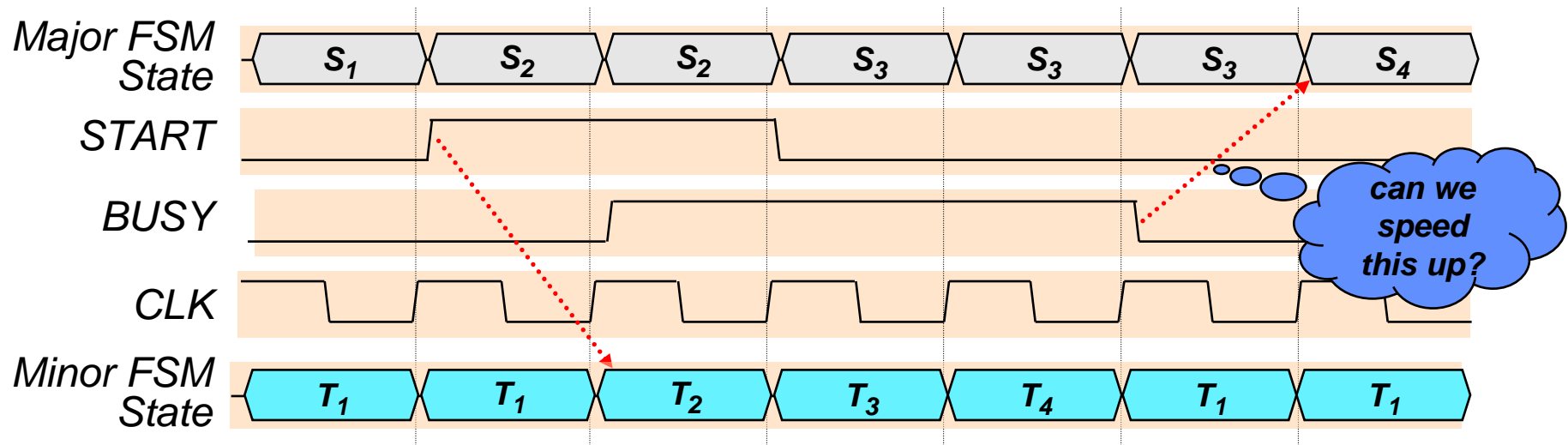


1. Wait for a trigger from the major FSM

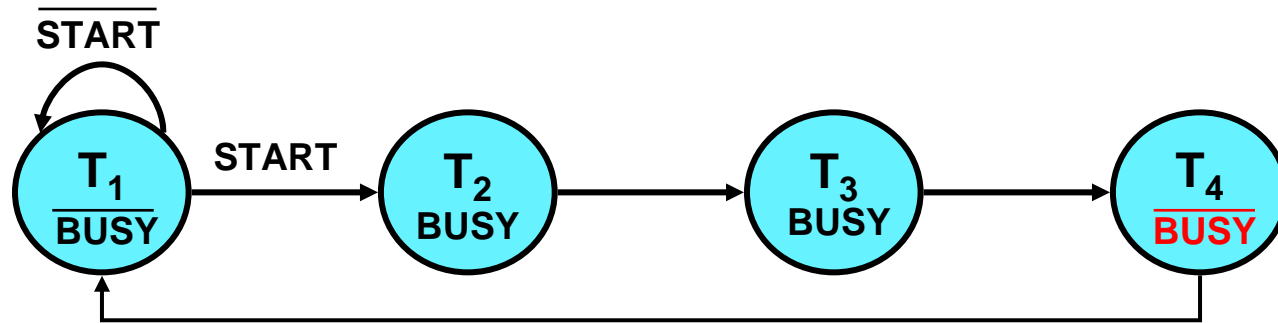


3. Signal to the major FSM that work is done

2. Do some useful work

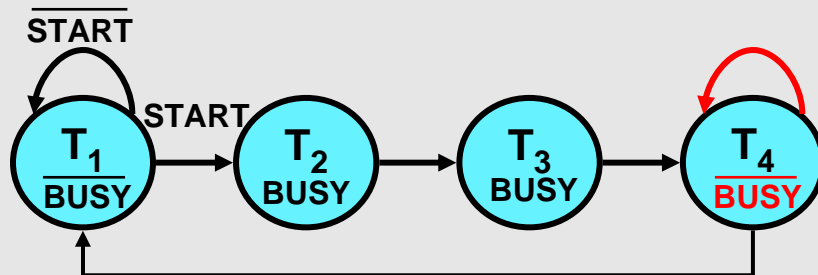


Good idea: de-assert BUSY one cycle early



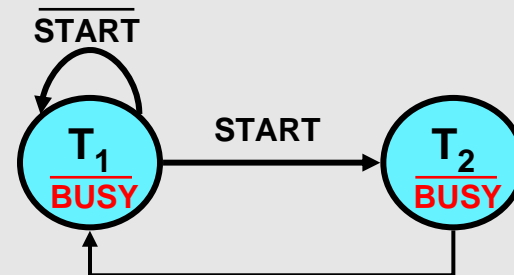
Bad idea #1:

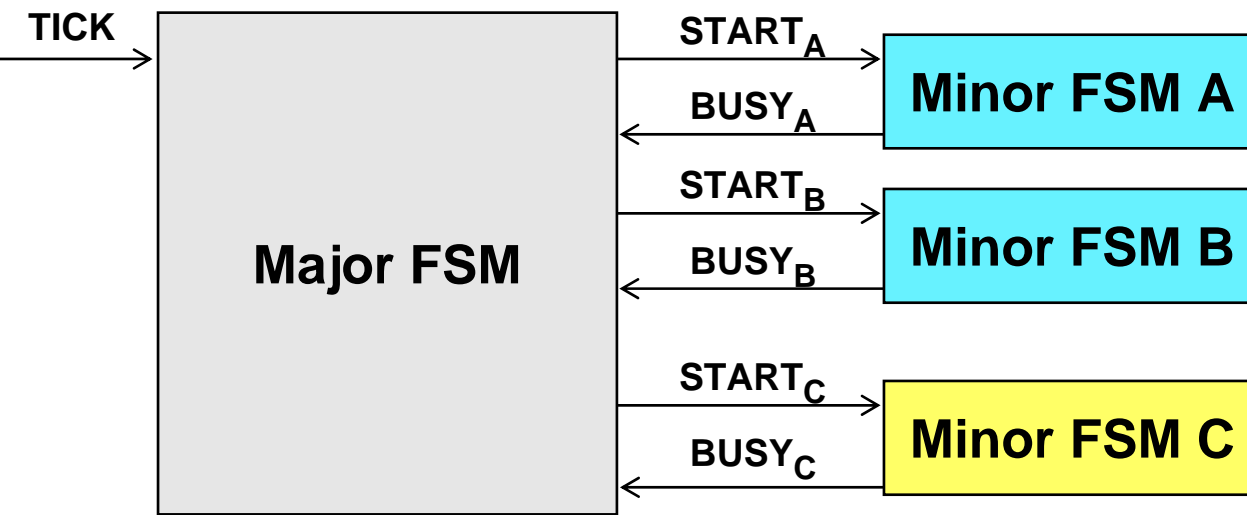
T₄ may not immediately return to T₁



Bad idea #2:

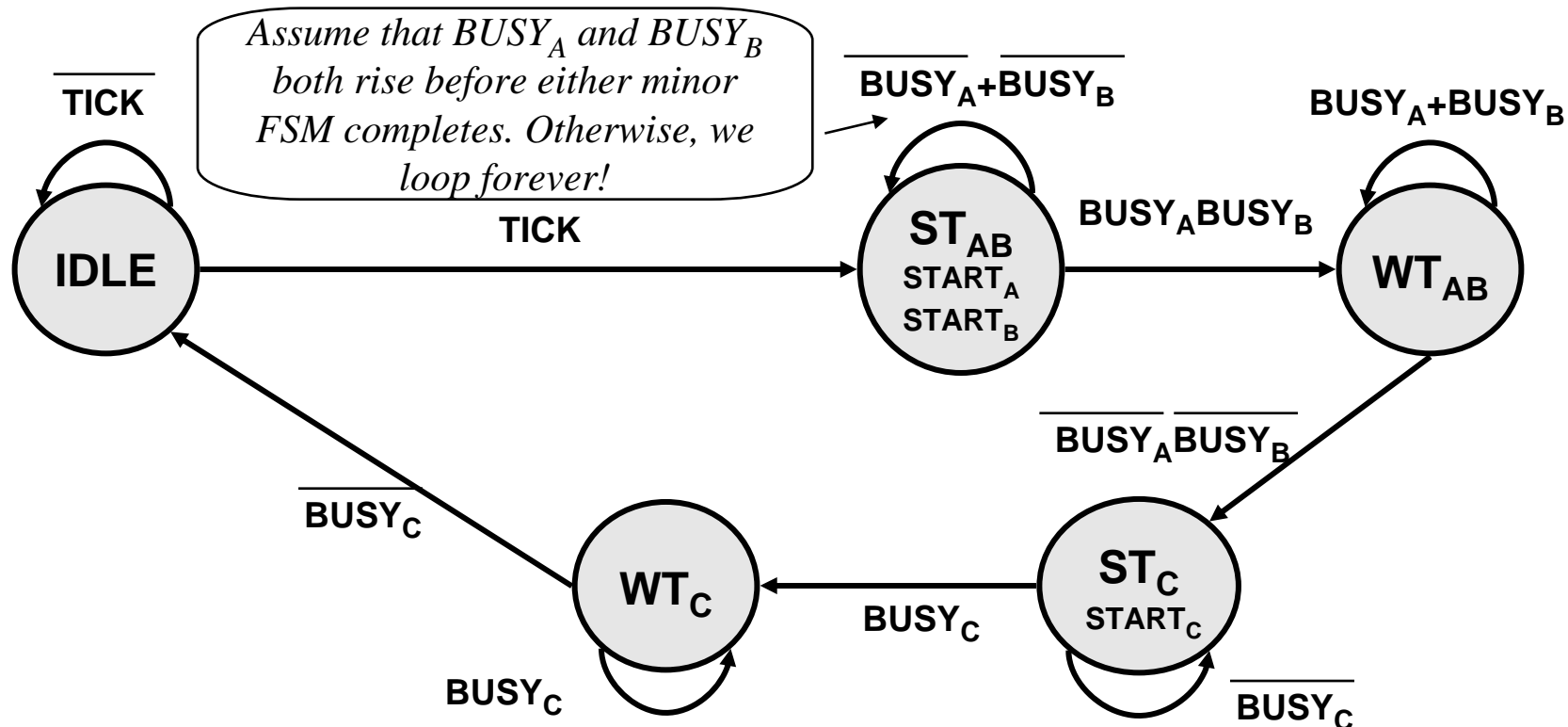
BUSY never asserts!





Operating Scenario:

- Major FSM is triggered by TICK
- Minors A and B are started simultaneously
- Minor C is started once both A and B complete
- TICKs arriving before the completion of C are ignored



Four-FSM Sample Waveform

