Digital Design & Computer Arch.

Lecture 7a: Sequential Logic Design II

Prof. Onur Mutlu

ETH Zürich

Spring 2020

12 March 2020

Agenda for This Week

- Today
  - Wrap up Sequential Logic
  - Hardware Description Languages and Verilog
    - Combinational Logic
    - Sequential Logic
- Tomorrow
  - Timing and Verification

Agenda for Next Week

- Thursday
  - Von Neumann Model of Execution
  - Instruction Set Architecture
    - LC-3 and MIPS
- Friday
  - ISA and Assembly Programming

Assignment: Required Lecture Video

- Why study computer architecture?
- Why is it important?
- **Future Computing Architectures**

- **Required Assignment**
  - **Watch** Prof. Mutlu's inaugural lecture at ETH and understand it
  - https://www.youtube.com/watch?v=kgiZlSOcGFM

- **Optional Assignment – for 1% extra credit**
  - **Write a 1-page summary** of the lecture and email us
    - What are your key takeaways?
    - What did you learn?
    - What did you like or dislike?
    - Submit your summary to Moodle – Deadline: April 1

Extra Assignment: Moore's Law (I)

- **Paper review**
- G.E. Moore. "Cramming more components onto integrated circuits," Electronics magazine, 1965

- **Optional Assignment – for 1% extra credit**
  - **Write a 1-page review**
  - Upload PDF file to Moodle – Deadline: April 1

- I strongly recommend that you follow my guidelines for (paper) review (see next slide)

Extra Assignment: Moore's Law (II)

- **Guidelines on how to review papers critically**
  - Guideline slides: pdf ppt
  - Video: https://www.youtube.com/watch?v=tOL6FANAJ8c
  - Example reviews on "Main Memory Scaling: Challenges and Solution Directions" (link to the paper)
    - Review 1
    - Review 2
  - Example review on "Staged memory scheduling: Achieving high performance and scalability in heterogeneous systems" (link to the paper)
    - Review 1

Required Readings (This Week)

- Hardware Description Languages and Verilog
  - H&H Chapter 4 in full

- Timing and Verification
  - H&H Chapters 2.9 and 3.5 + (start Chapter 5)

- By tomorrow, make sure you are done with
  - **P&P Chapters 1-3 + H&H Chapters 1-4**

Required Readings (Next Week)

- Von Neumann Model, LC-3, and MIPS
  - P&P, Chapters 4, 5
  - H&H, Chapter 6
  - P&P, Appendices A and C (ISA and microarchitecture of LC-3)
  - H&H, Appendix B (MIPS instructions)

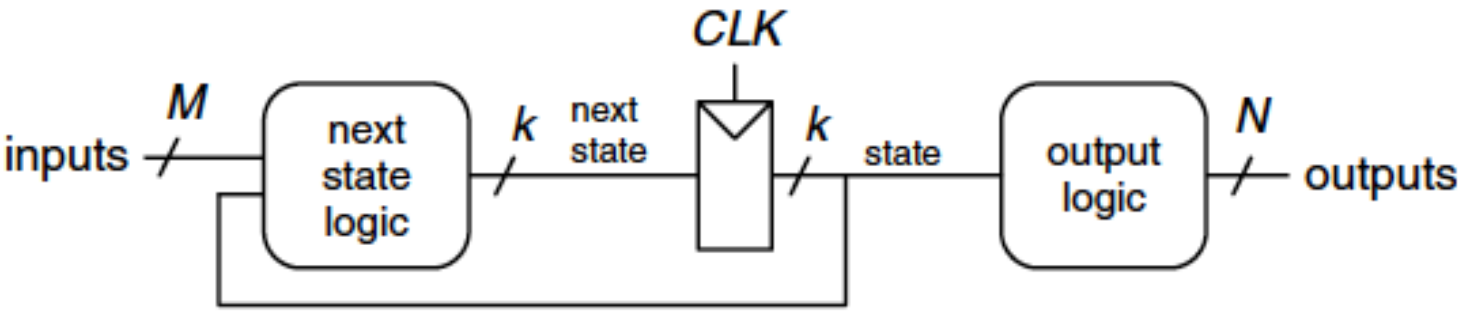
- Programming
  - P&P, Chapter 6

- **Recommended:** Digital Building Blocks
  - H&H, Chapter 5

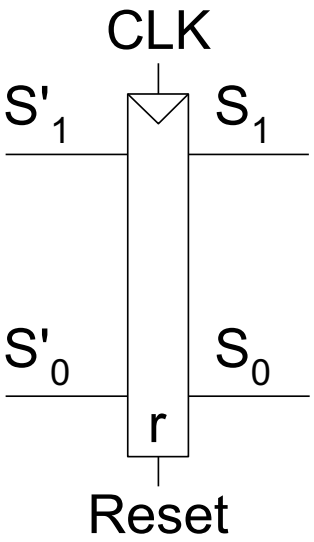
Wrap-Up Sequential Logic Circuits and Design

Finite State Machine: Schematic

FSM Schematic: State Register

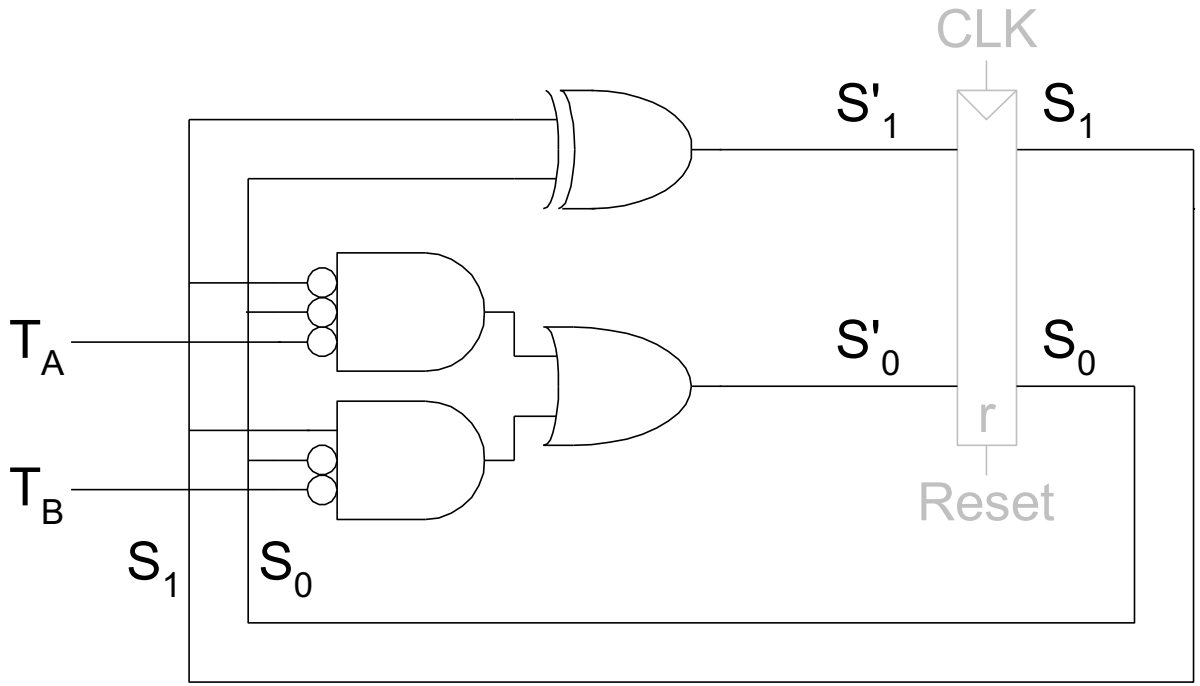


FSM Schematic: State Register



state register

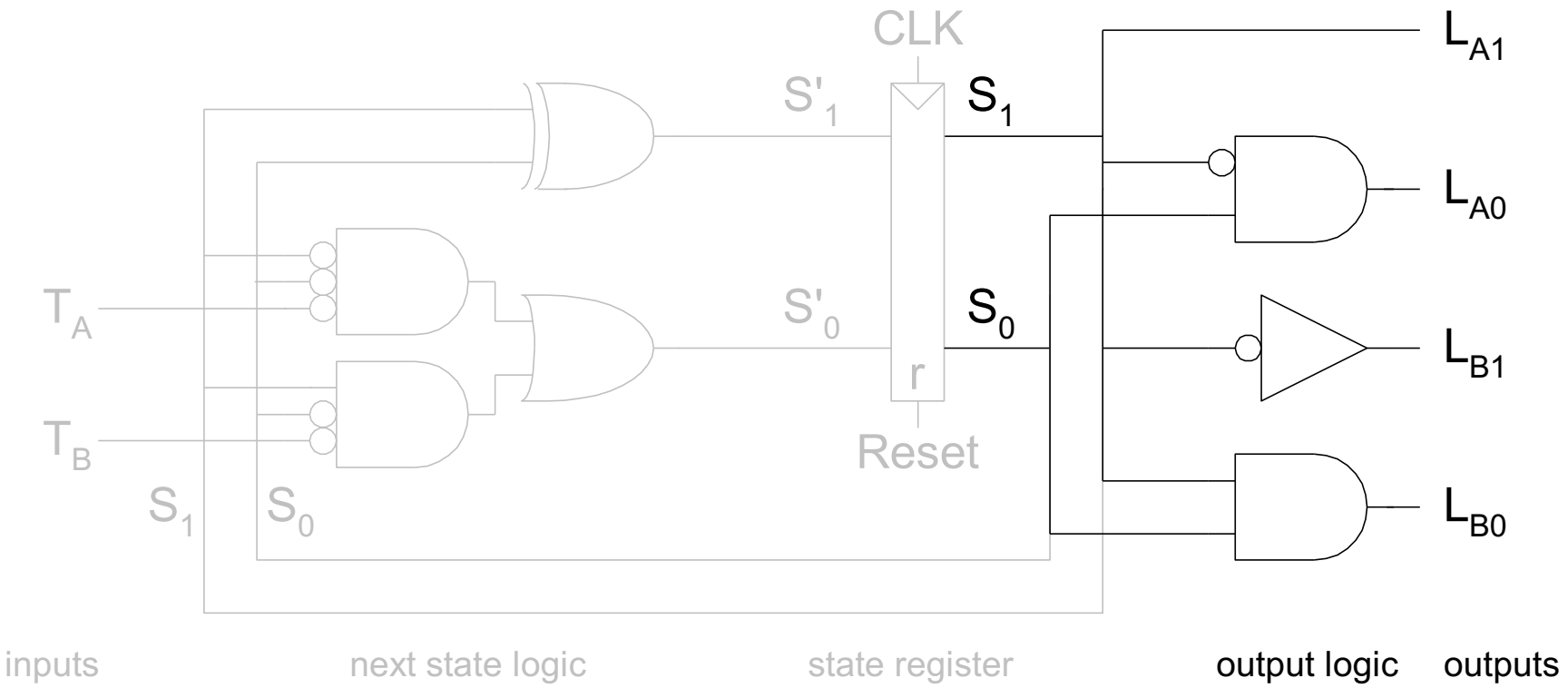
FSM Schematic: Next State Logic



$$S'_1 = S_1 \text{ xor } S_0$$

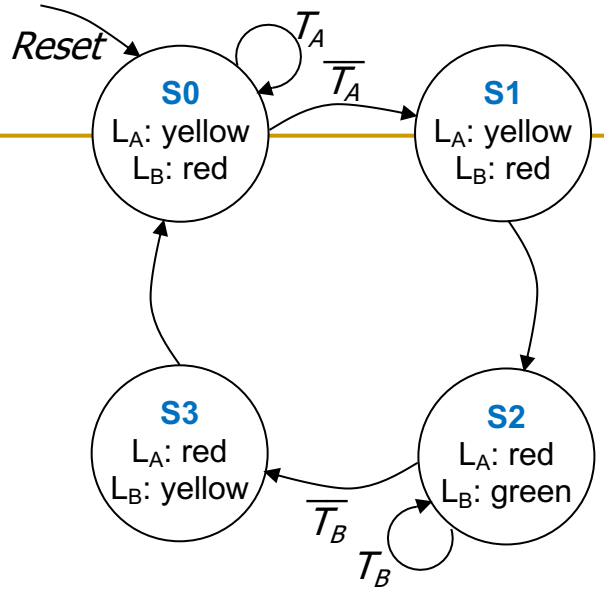
$$S'_0 = (\overline{S}_1 \cdot \overline{S}_0 \cdot \overline{T}_A) + (S_1 \cdot \overline{S}_0 \cdot \overline{T}_B)$$

FSM Schematic: Output Logic

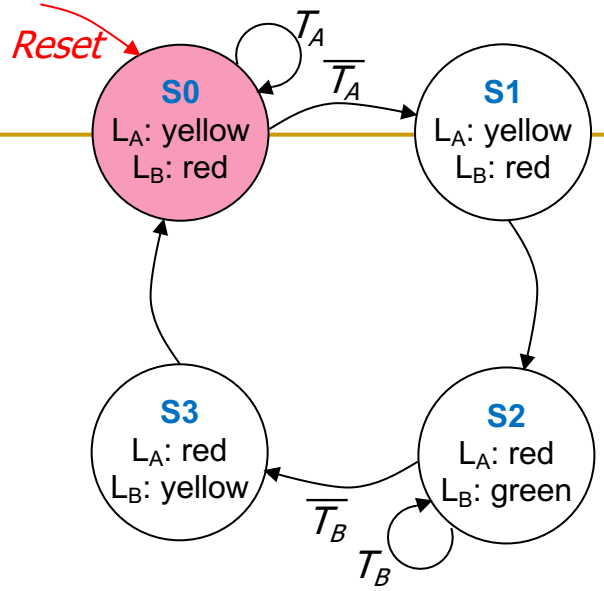


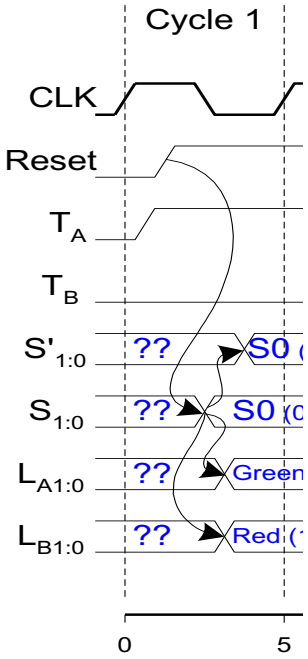
$$L_{A1} = S_1$$
$$L_{A0} = \overline{S_1} \cdot S_0$$
$$L_{B1} = \overline{S_1}$$
$$L_{B0} = S_1 \cdot S_0$$

FSM Timing Diagram

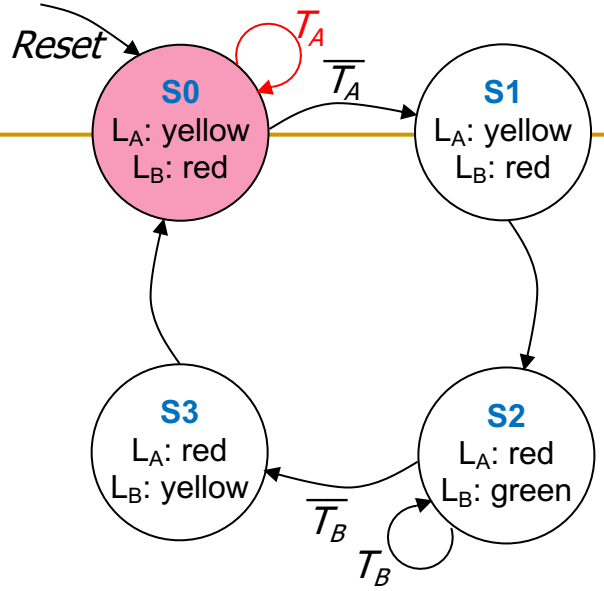


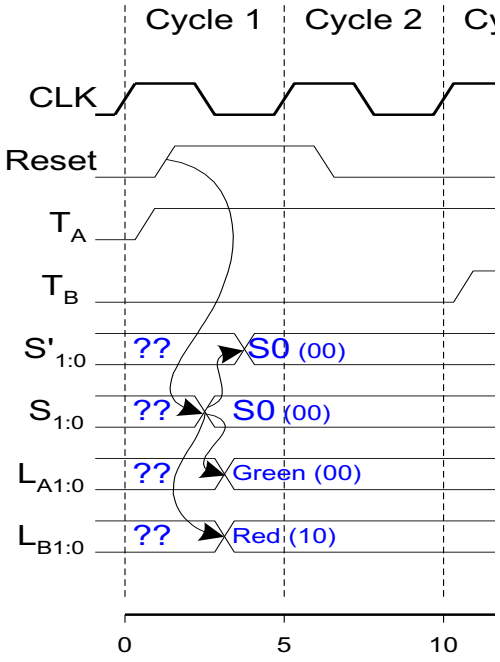
FSM Timing Diagram



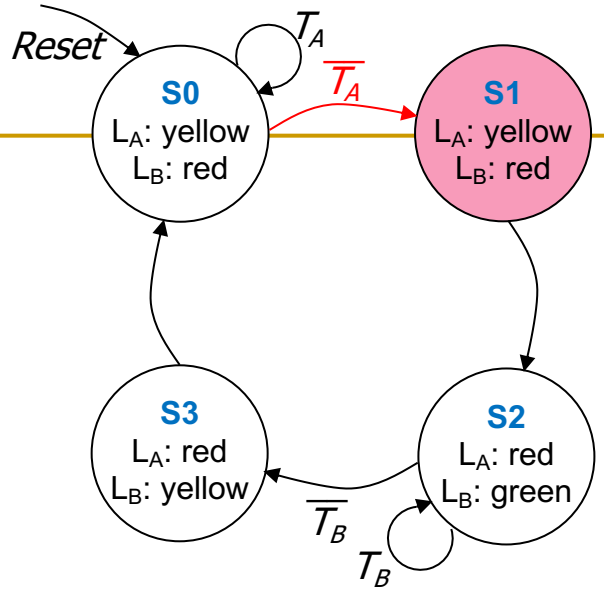


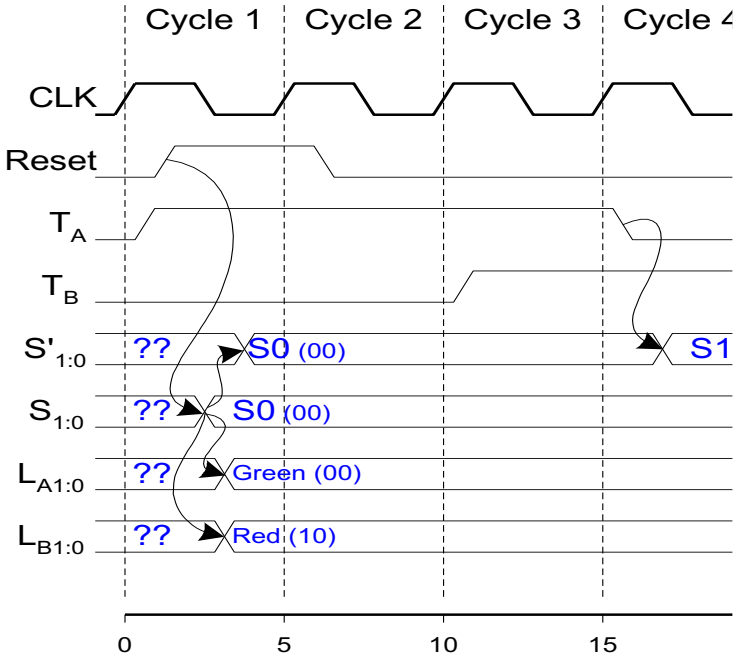
FSM Timing Diagram



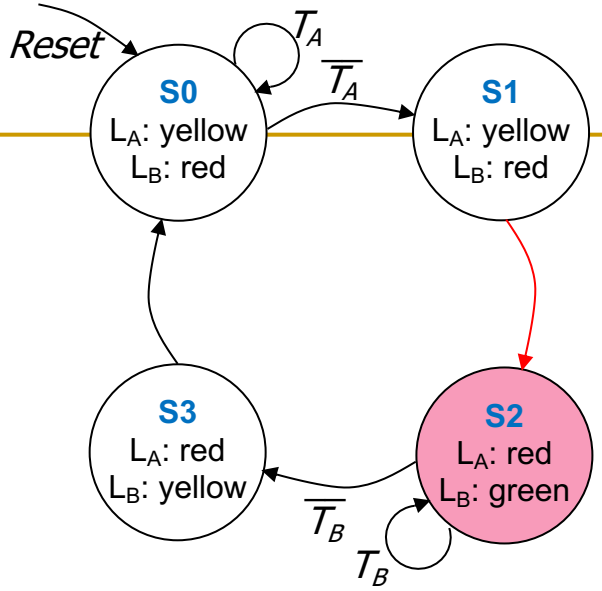


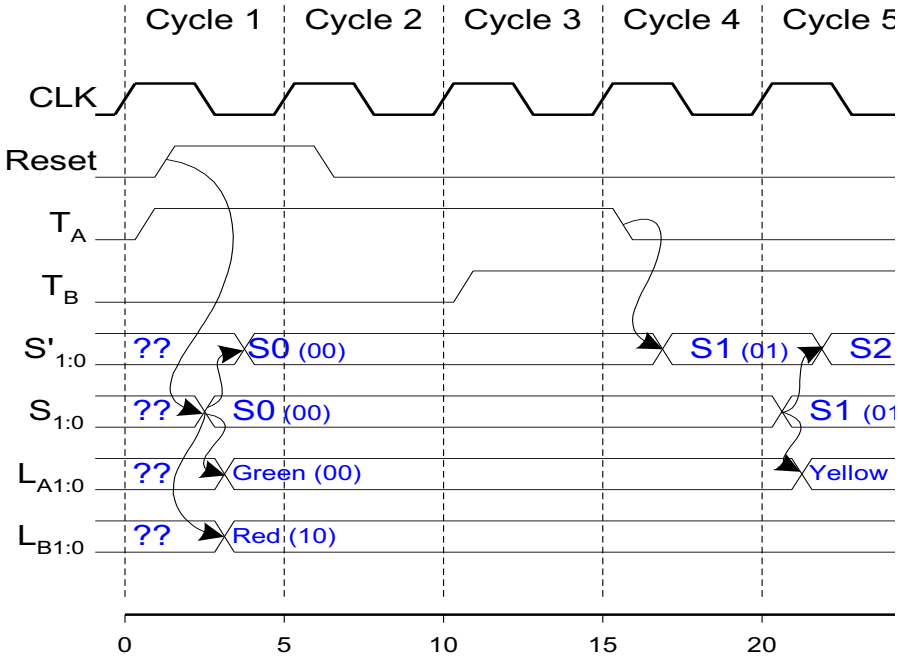
FSM Timing Diagram



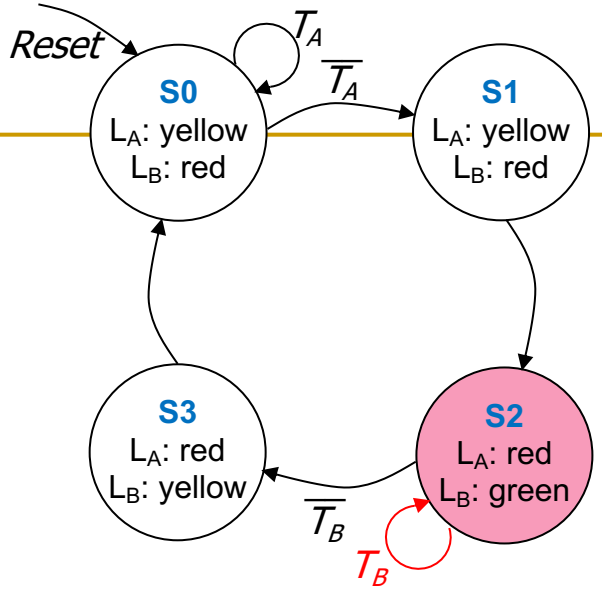


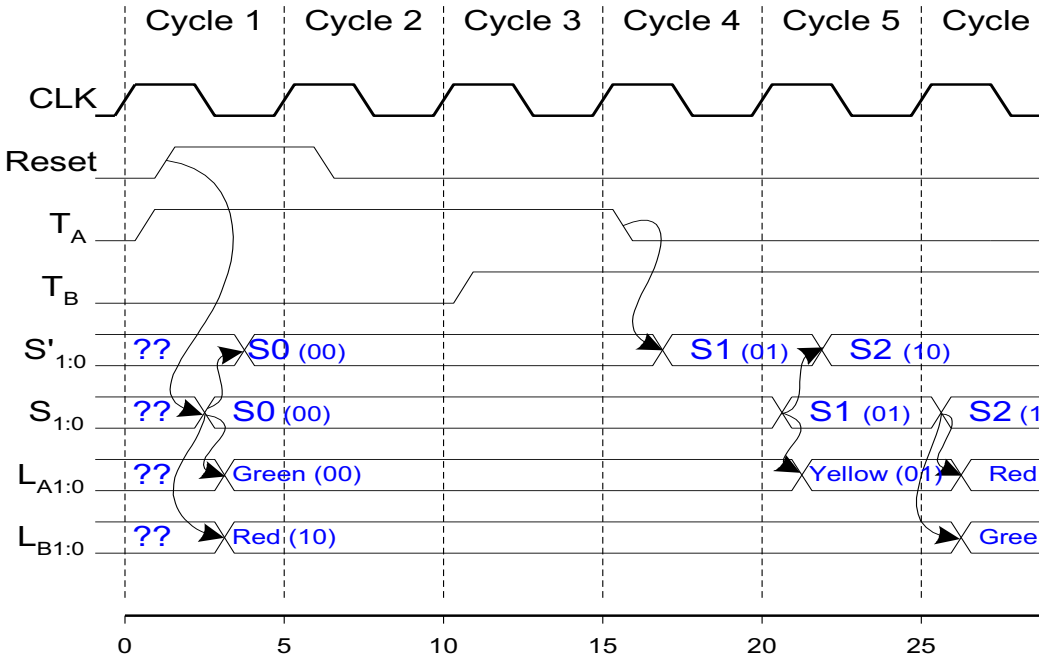
FSM Timing Diagram



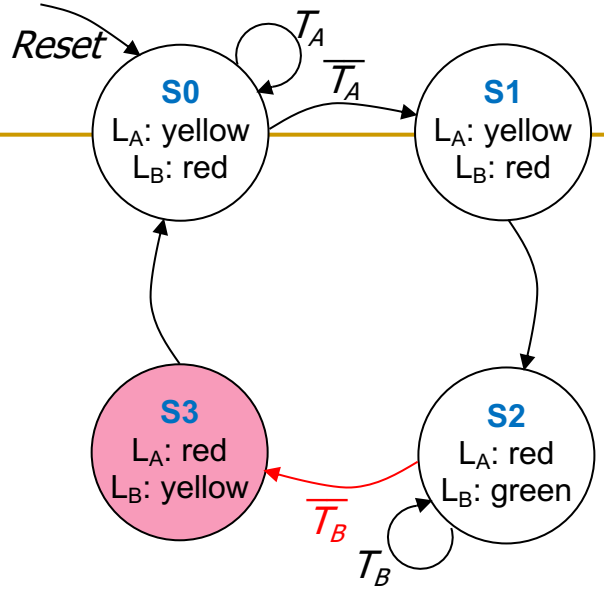


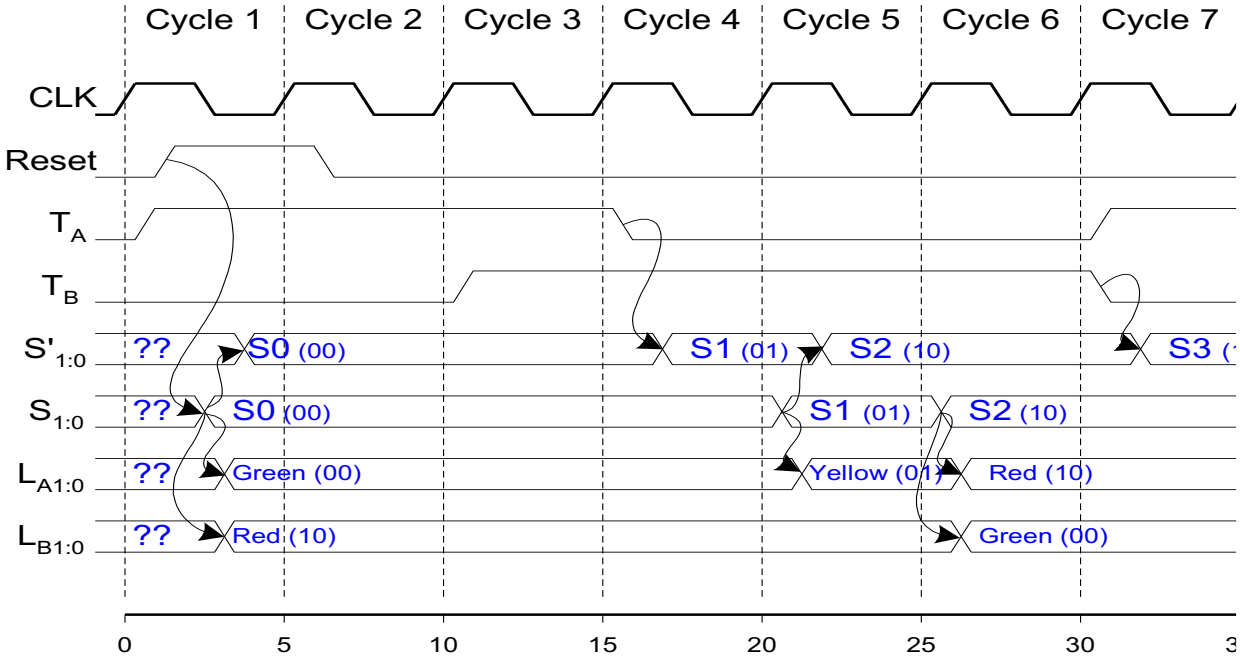
FSM Timing Diagram



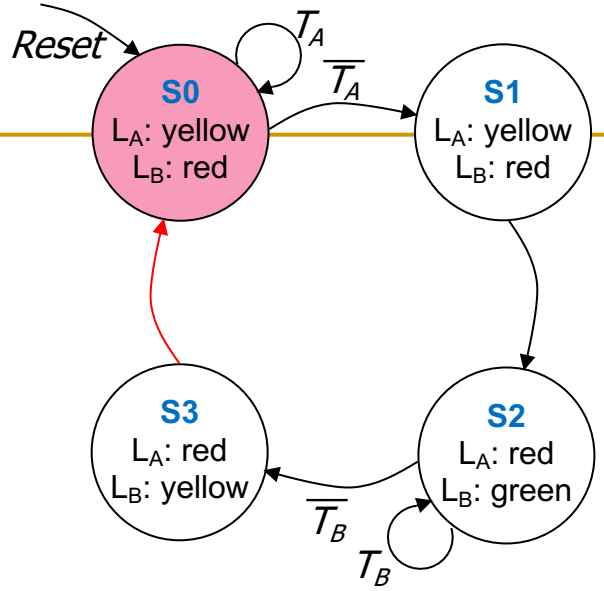


FSM Timing Diagram

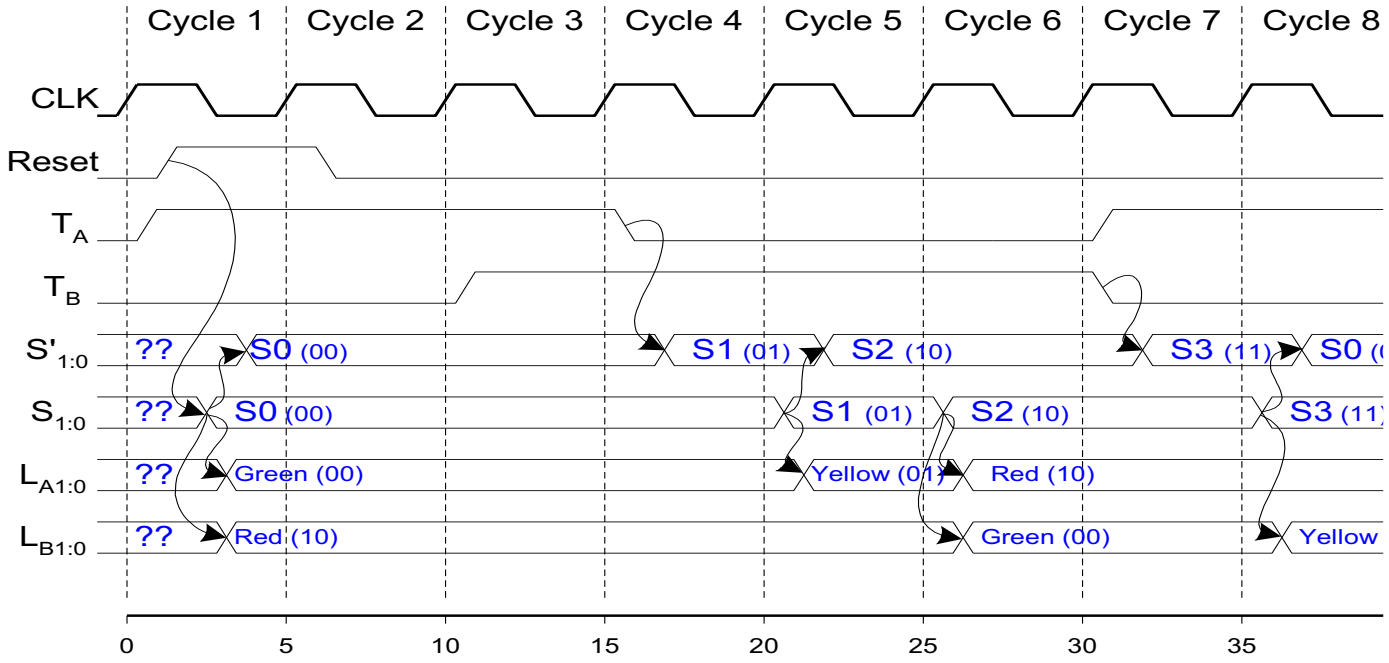




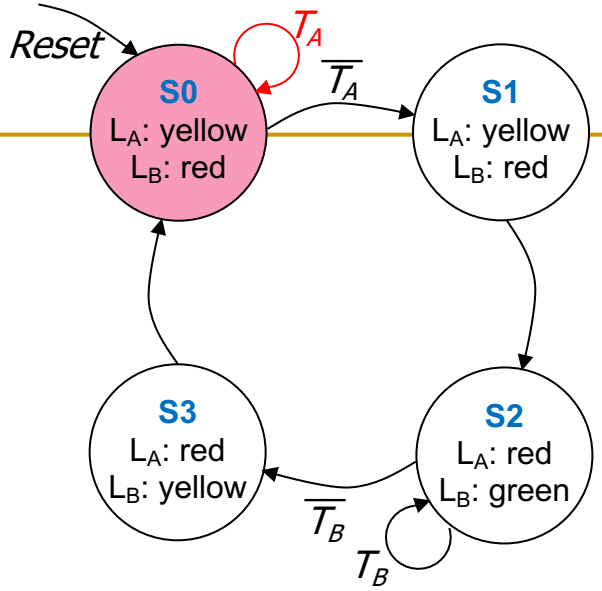
FSM Timing Diagram

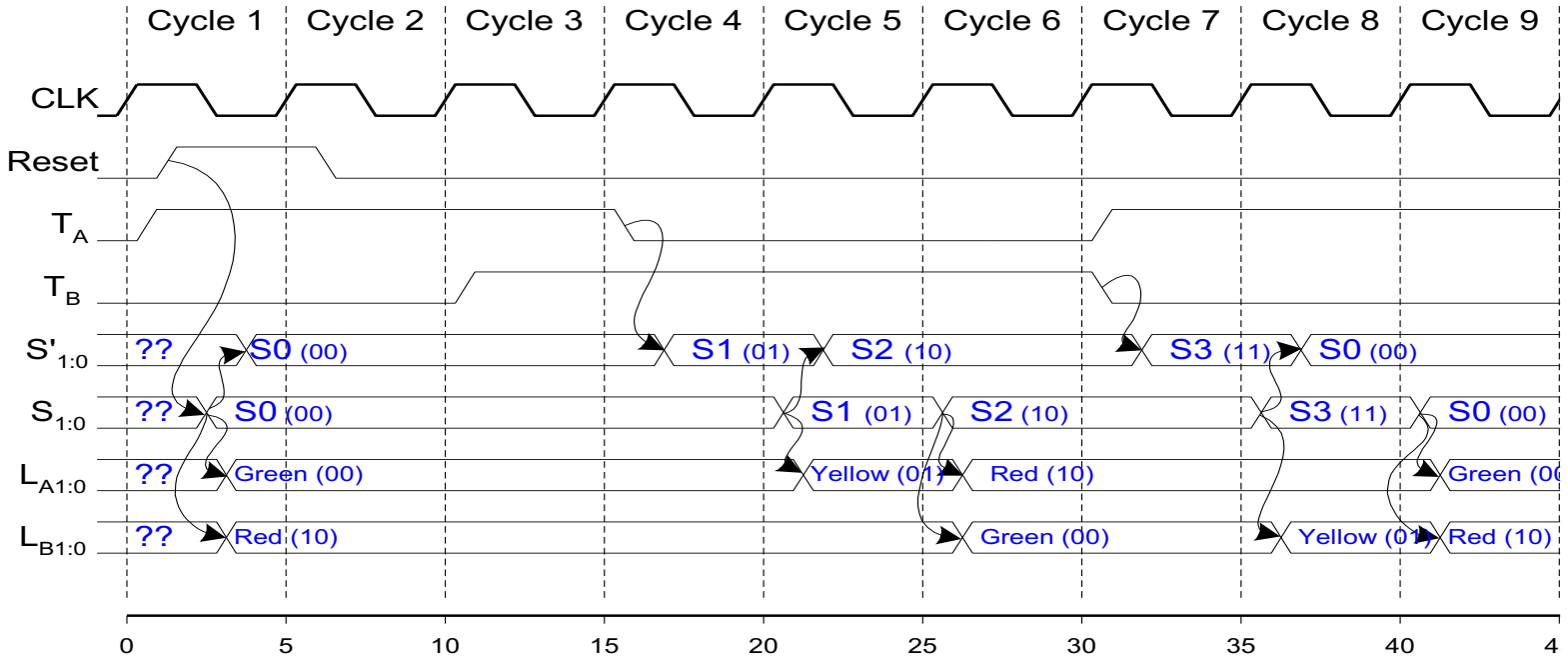


This is from H&H Section 3.4.1

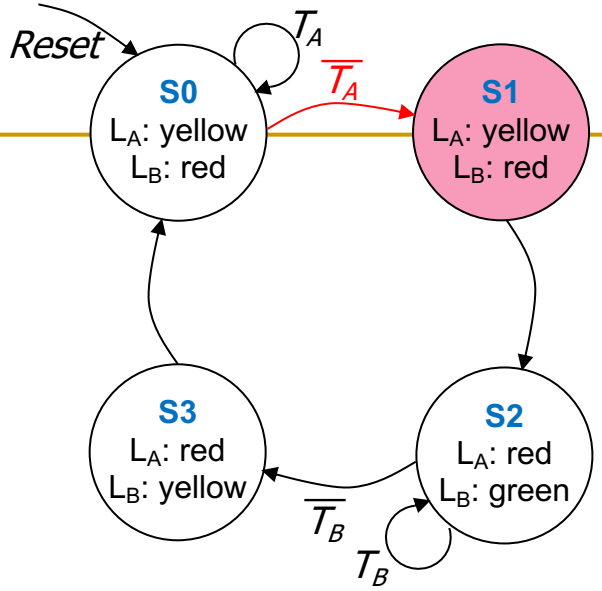


FSM Timing Diagram

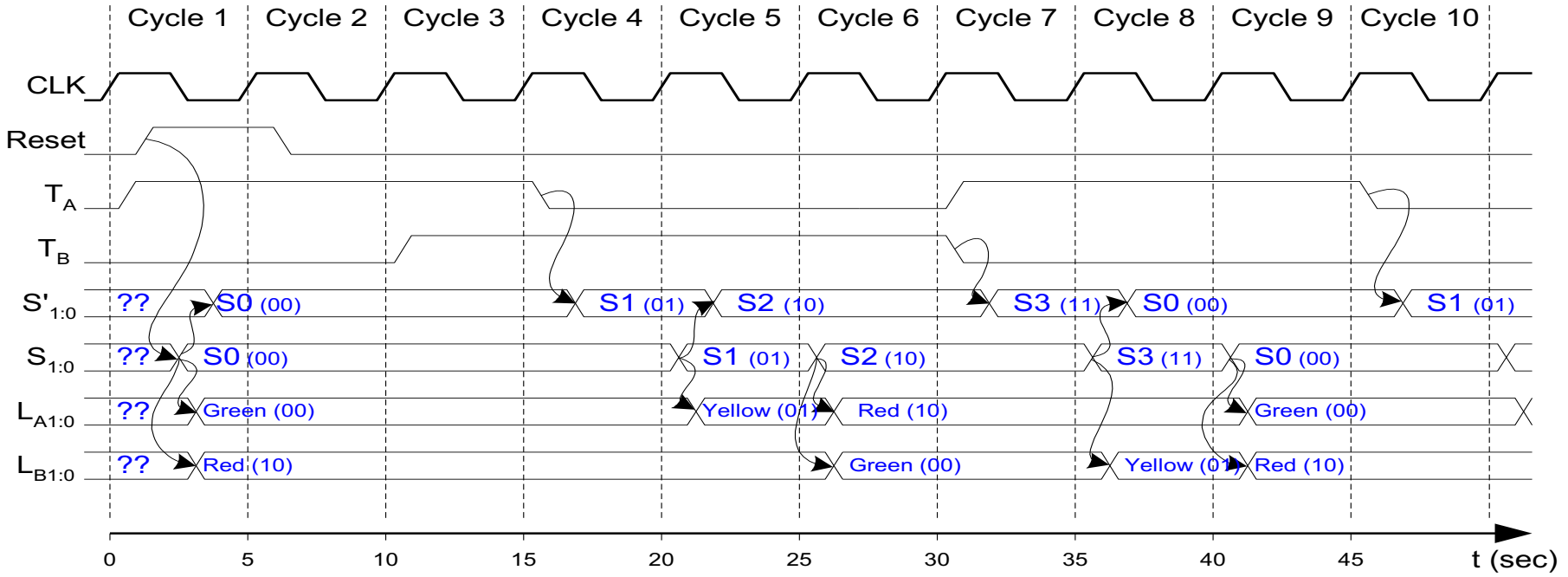




FSM Timing Diagram



See H&H Chapter 3.4

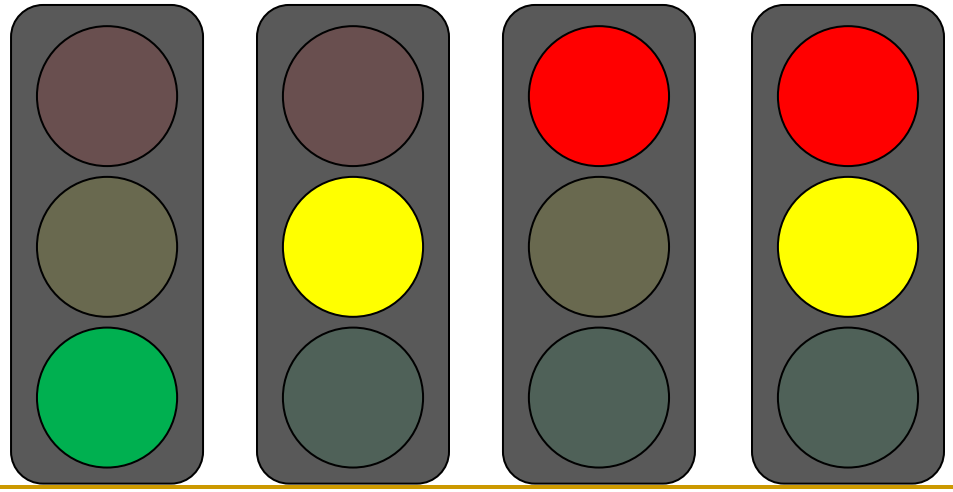


Finite State Machine: State Encoding

FSM State Encoding

- How do we encode the state bits?
  - Three common state binary encodings with different tradeoffs
    1. **Fully Encoded**
    2. **1-Hot Encoded**
    3. **Output Encoded**

- Let's see an example **Swiss** traffic light with 4 states
  - Green, Yellow, Red, Yellow+Red



FSM State Encoding (II)

1. **Binary Encoding (Full Encoding):**
   - Use the minimum number of bits used to encode all states
     - Use $log_2(num\_states)$ bits to represent the states
   - *Example states:* 00, 01, 10, 11
   - **Minimizes** # flip-flops, but not necessarily output logic or next state logic

2. **One-Hot Encoding:**
   - Each bit encodes a different state
     - Uses *num_states* bits to represent the states
     - Exactly 1 bit is "hot" for a given state
   - *Example states:* 0001, 0010, 0100, 1000
   - **Simplest design process** – very automatable
   - **Maximizes** # flip-flops, **minimizes** next state logic

FSM State Encoding (III)

3. Output Encoding:

- Outputs are **directly accessible** in the state encoding
- For example, since we have **3 outputs** (light color), encode state with **3 bits**, where each bit represents a color
- *Example states:* 001, 010, 100, 110
  - Bit_0 encodes **green** light output,
  - Bit_1 encodes **yellow** light output
  - Bit_2 encodes **red** light output
- **Minimizes** output logic
- Only works for Moore Machines (output function of state)

FSM State Encoding (III)

3. **Output Encoding:**
   - Outputs are **directly accessible** in the state encoding

The **designer** must **carefully** choose an encoding scheme to **optimize** the design under given constraints

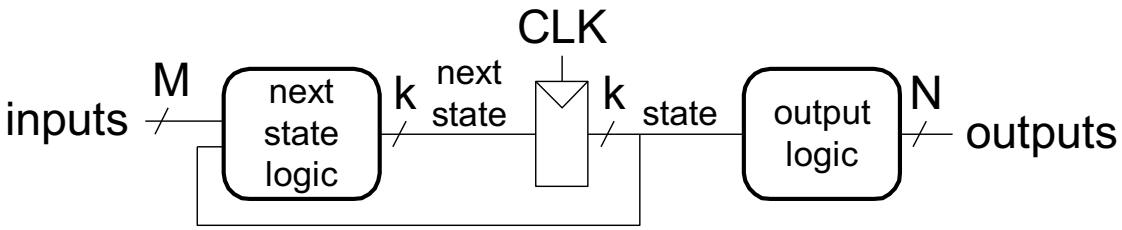
   - **Minimizes** output logic
   - Only works for Moore Machines (output function of state)

Moore vs. Mealy Machines

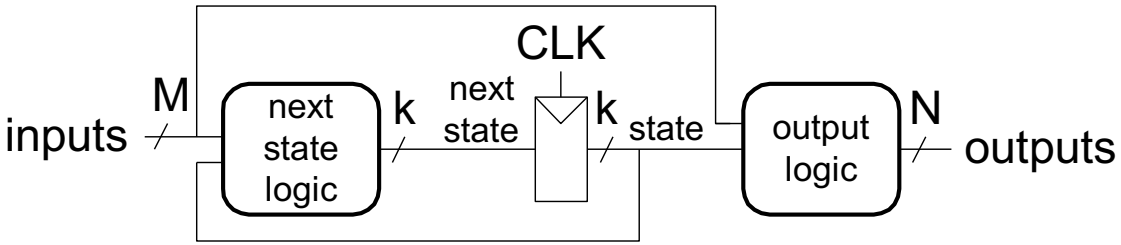
Recall: Moore vs. Mealy FSMs

- Next state is determined by the current state and the inputs
- Two types of finite state machines differ in the **output logic**:
  - **Moore FSM**: outputs depend only on the current state
  - **Mealy FSM**: outputs depend on the current state and the inputs

Moore FSM

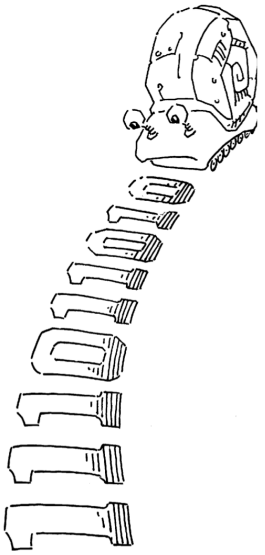


Mealy FSM

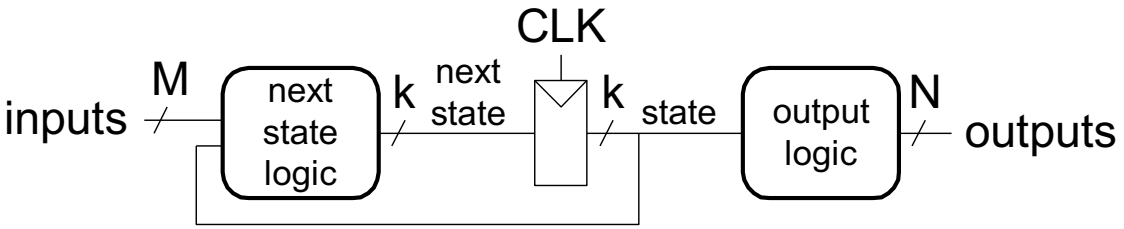


Moore vs. Mealy FSM Examples

- Alyssa P. Hacker has a snail that crawls down a paper tape with 1's and 0's on it.
- The snail smiles whenever the last four digits it has crawled over are 1101.
- Design Moore and Mealy FSMs of the snail's brain.

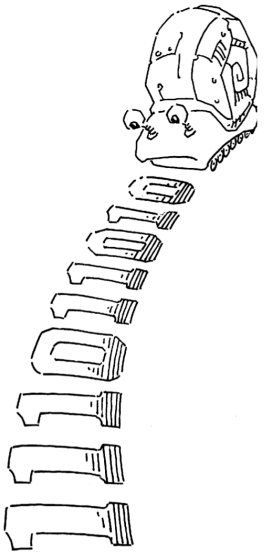


Moore FSM

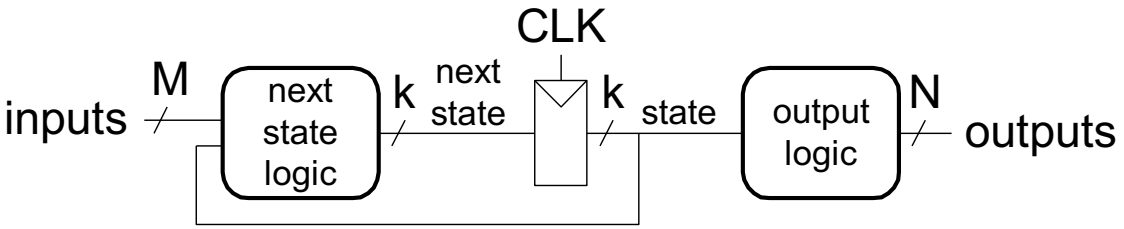


Moore vs. Mealy FSM Examples

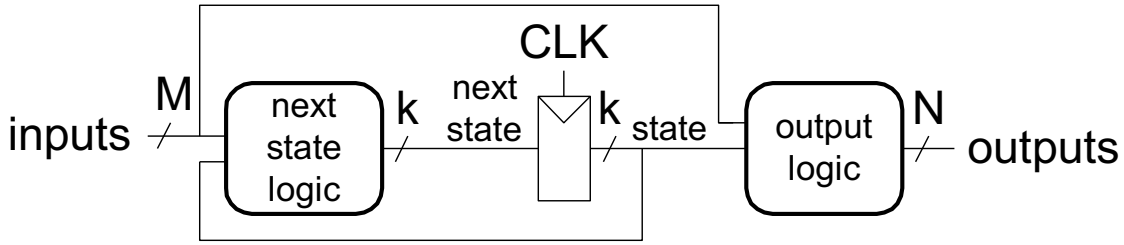
- Alyssa P. Hacker has a snail that crawls down a paper tape with 1's and 0's on it.
- The snail smiles whenever the last four digits it has crawled over are 1101.
- Design Moore and Mealy FSMs of the snail's brain.



Moore FSM

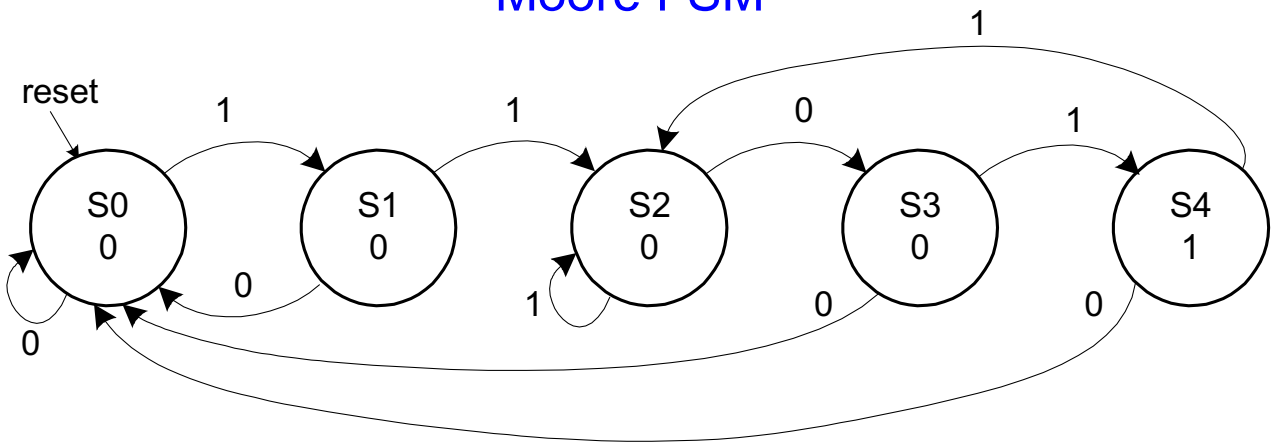


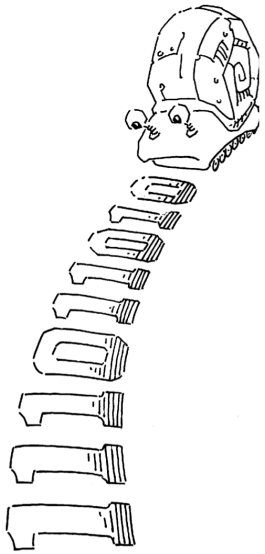
Mealy FSM



State Transition Diagrams

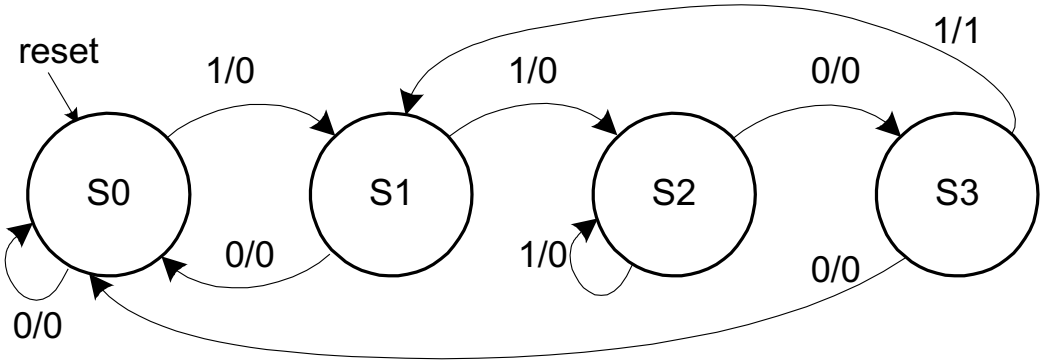
Moore FSM





What are the tradeoffs?

Mealy FSM



FSM Design Procedure

- **Determine** all possible states of your machine
- **Develop** a **state transition diagram**
  - Generally this is done from a textual description
  - You need to 1) determine the **inputs** and **outputs** for each **state** and 2) figure out how to get from one state to another
- **Approach**
  - Start by defining the **reset state** and what happens from it – this is typically an easy point to start from
  - Then continue to add **transitions** and **states**
  - Picking **good state names** is very important
  - Building an FSM is **like** programming (but it *is not* programming!)
    - An FSM has a sequential "control-flow" like a program with conditionals and goto's
    - The if-then-else construct is controlled by one or more inputs
    - The outputs are controlled by the state or the inputs
  - In hardware, we typically have many concurrent FSMs

What is to Come: LC-3 Processor

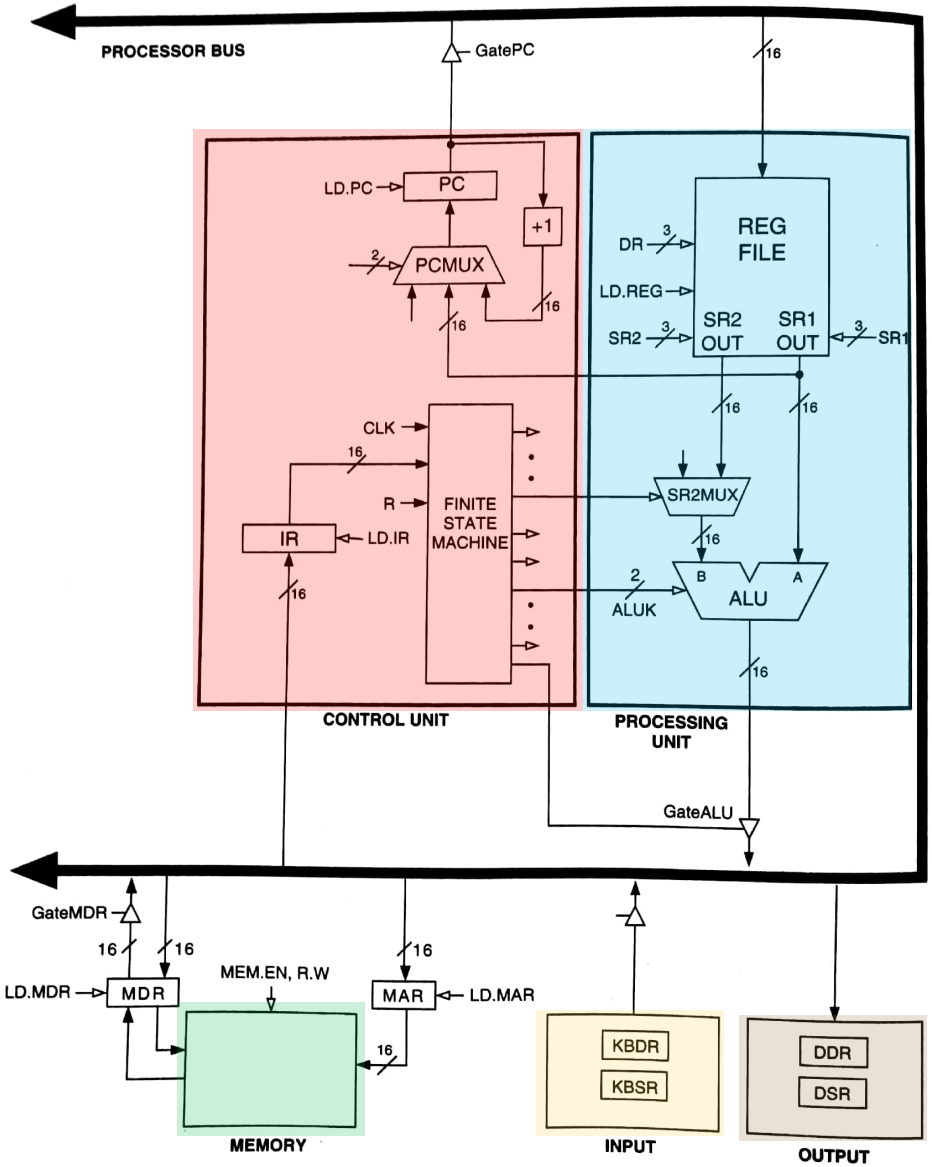
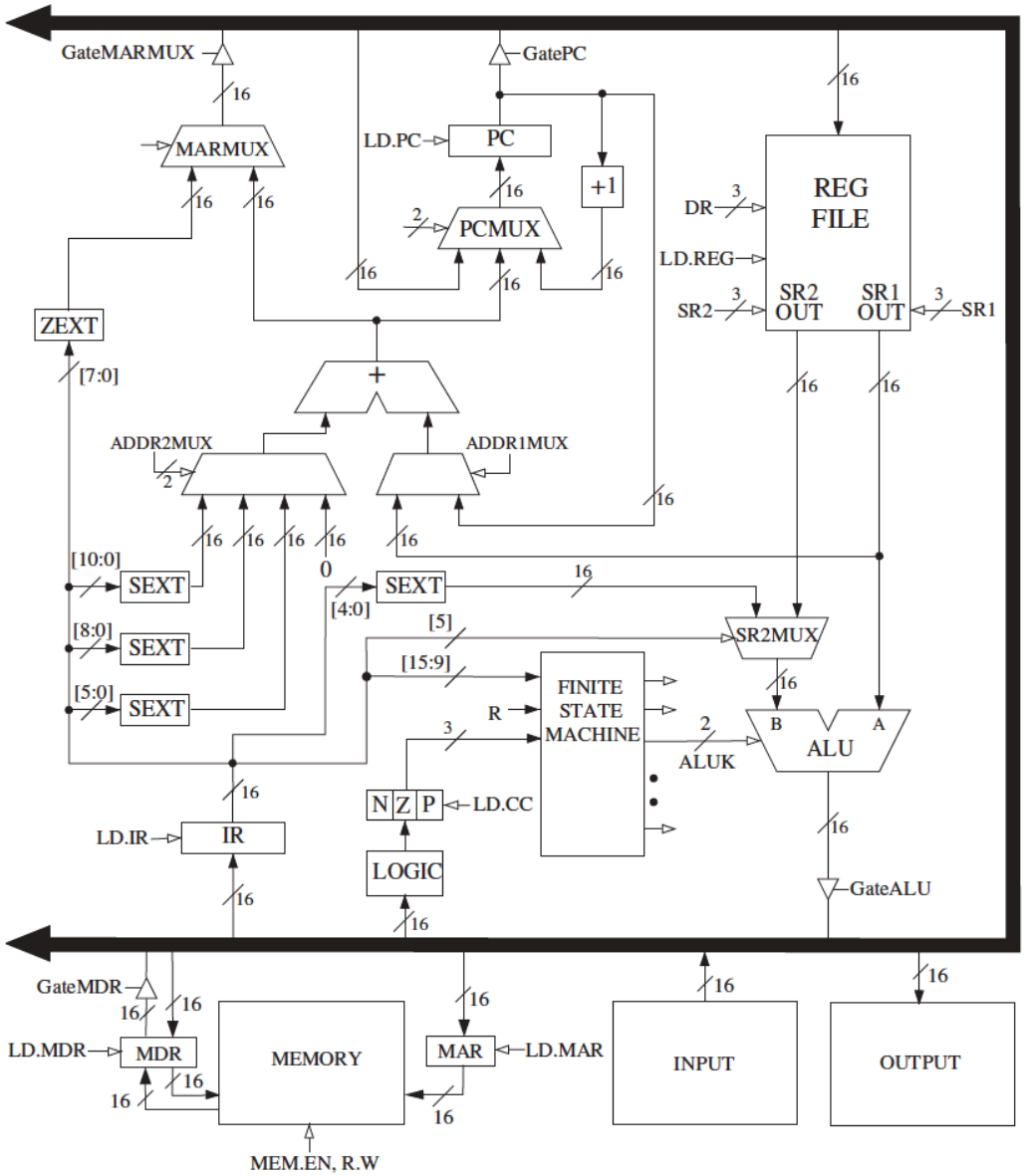


Figure 4.3 The LC-3 as an example of the von Neumann model

What is to Come: LC-3 Datapath



Digital Design & Computer Arch.

Lecture 7a: Sequential Logic Design II

Prof. Onur Mutlu

ETH Zürich

Spring 2020

12 March 2020