22.03.21 11:57

CSCI 150
Introduction to Digital and Computer System Design
Lecture 4: Sequential Circuit IV

Jetic Gū

Overview

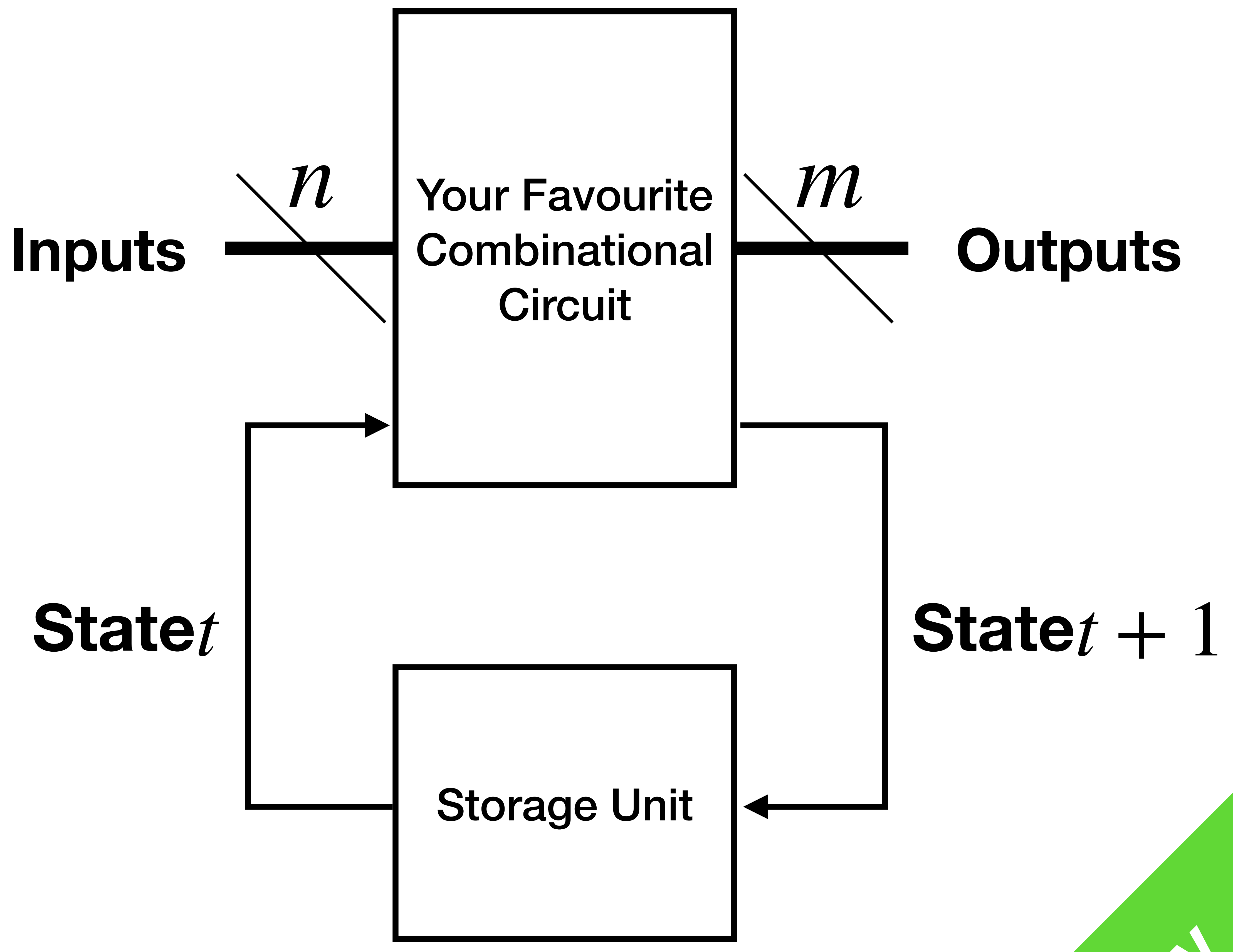
- Focus: Basic Information Retaining Blocks
- Architecture: Sequential Circuit
- Textbook v4: Ch5 5.3, 5.4; v5: Ch4 4.2 4.3
- Core Ideas:

  1. Sequential Circuit Design Procedures

P0
Review

Definitions

1. **Storage Elements**
   circuits that can store binary information
2. **State**
   partial results, instructions, etc.
3. **Synchronous Sequential Circuit**
   Signals arrive at discrete instants of time, outputs at next time step
4. **Asynchronous Sequential Circuit**
   Signals arrive at any instant of time, outputs when ready



Review

P0
Review

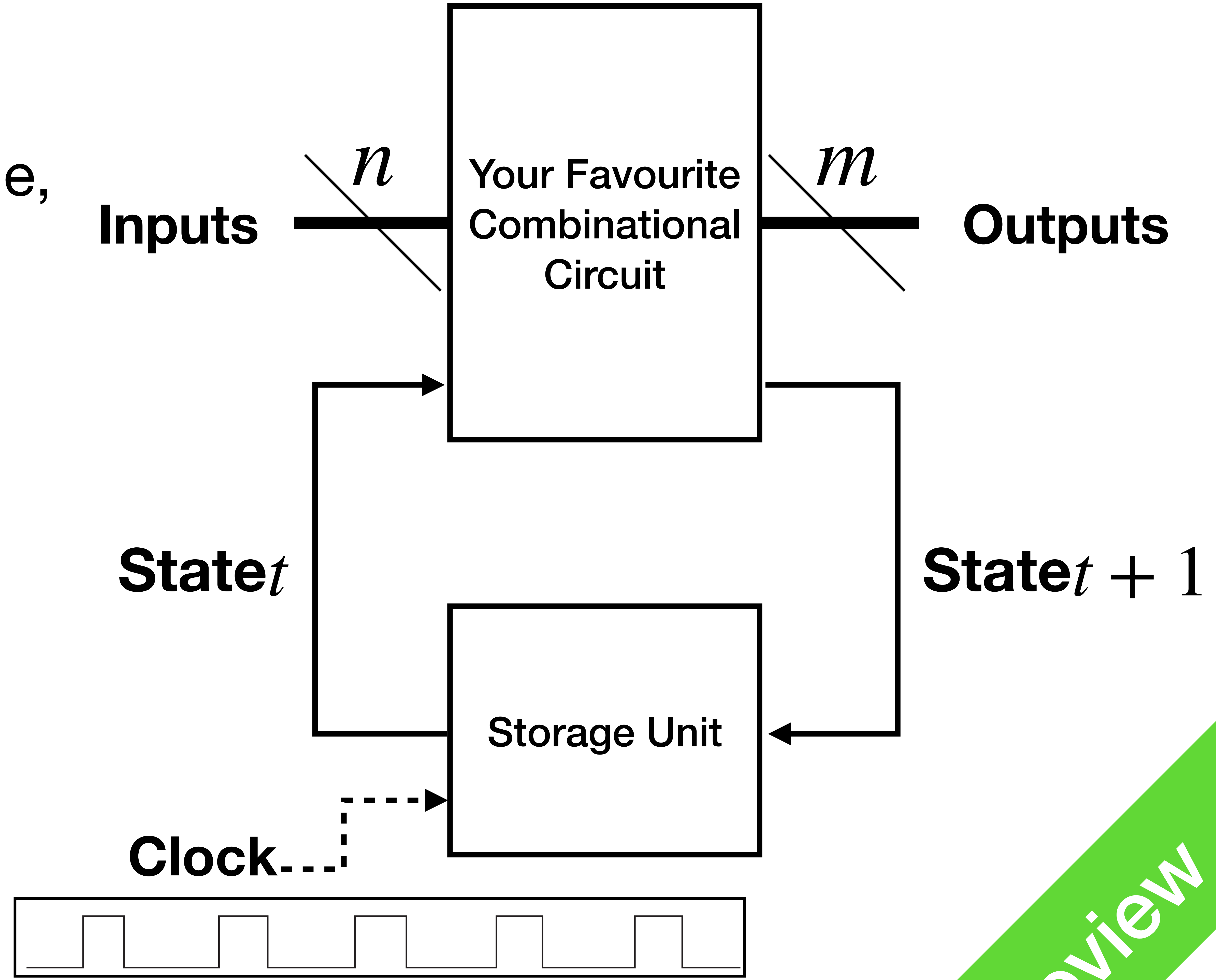
Definitions

3. **Synchronous Sequential Circuit**
   Signals arrive at discrete instants of time, outputs at next time step

- Has Clock

4. **Asynchronous Sequential Circuit**
   Signals arrive at any instant of time, outputs when ready

- May not have Clock

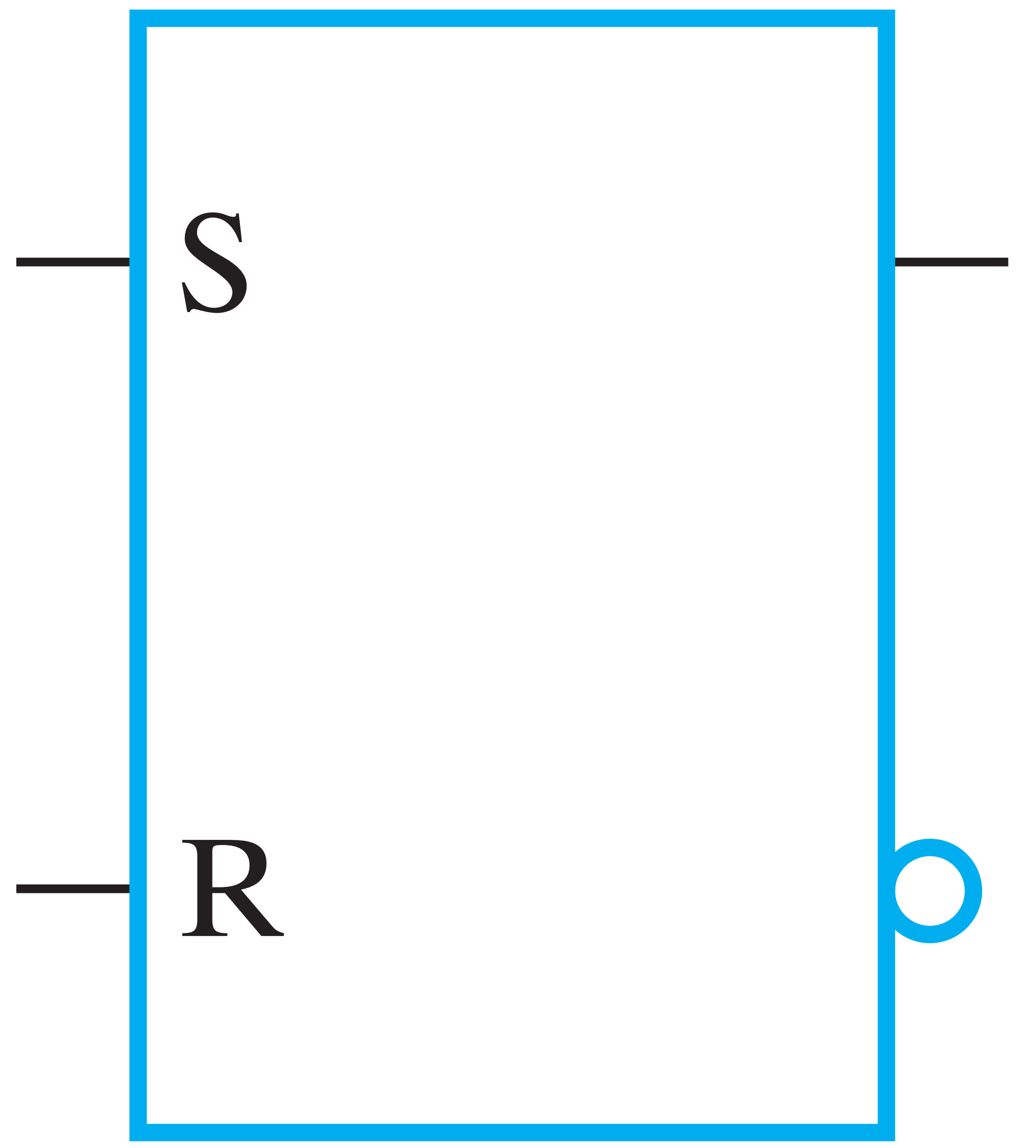


Review

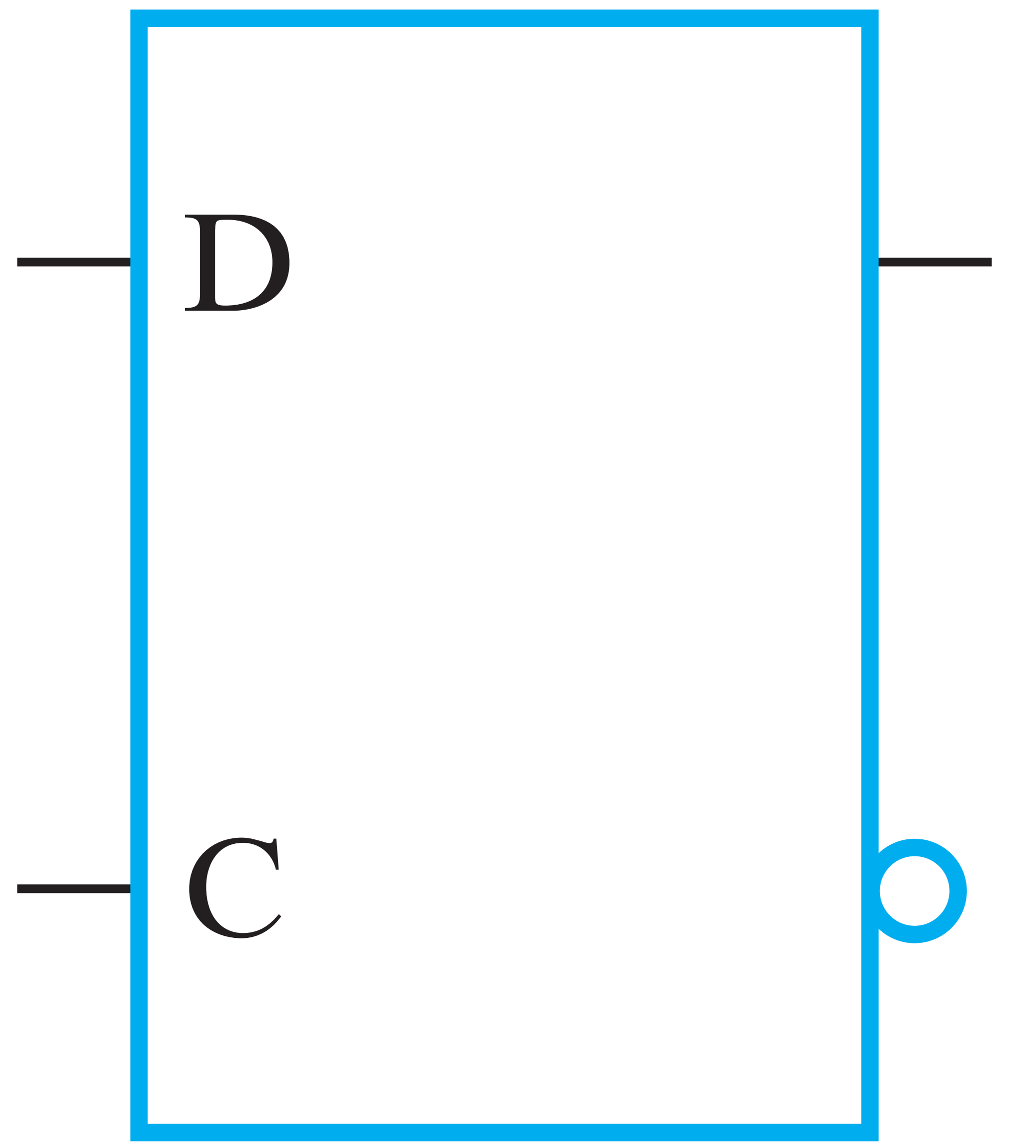
P0
Review

Summary

Latches

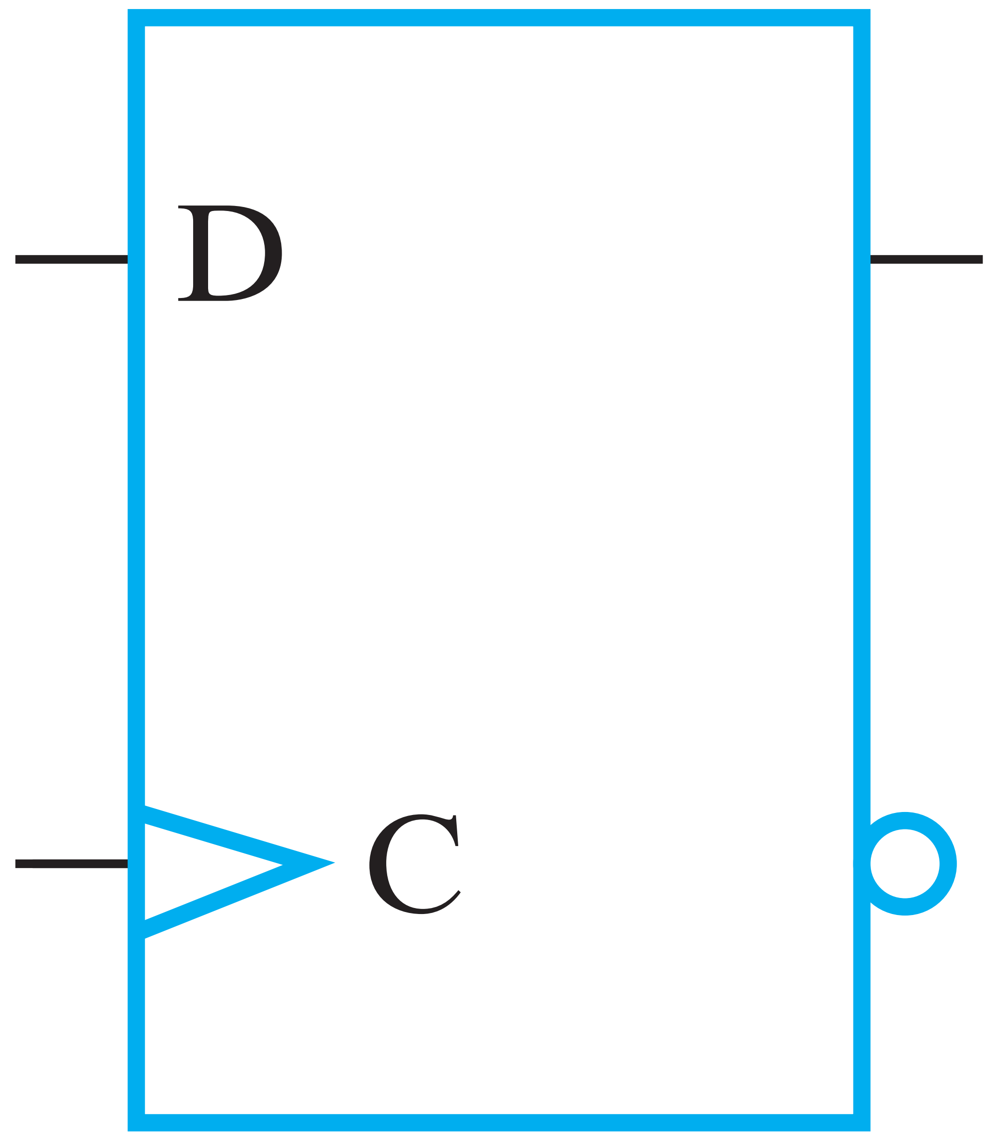


SR



D with 1 Control

Flip-Flops



⎾ Triggered D

Demo

P1
Design

Sequential Circuit Design I

8 Step Design Procedures; Formulation

Summary

P1
Design

Systematic Design Procedures Combinational Circuits

1. **Specification**
2. **Formulation**
3. **Optimisation**
4. **Technology Mapping**
5. **Verification**

Review

P1
Design

Systematic Design Procedures Sequential Circuits

1. **Specification**
2. **Formulation**
   e.g. using state table or state diagram

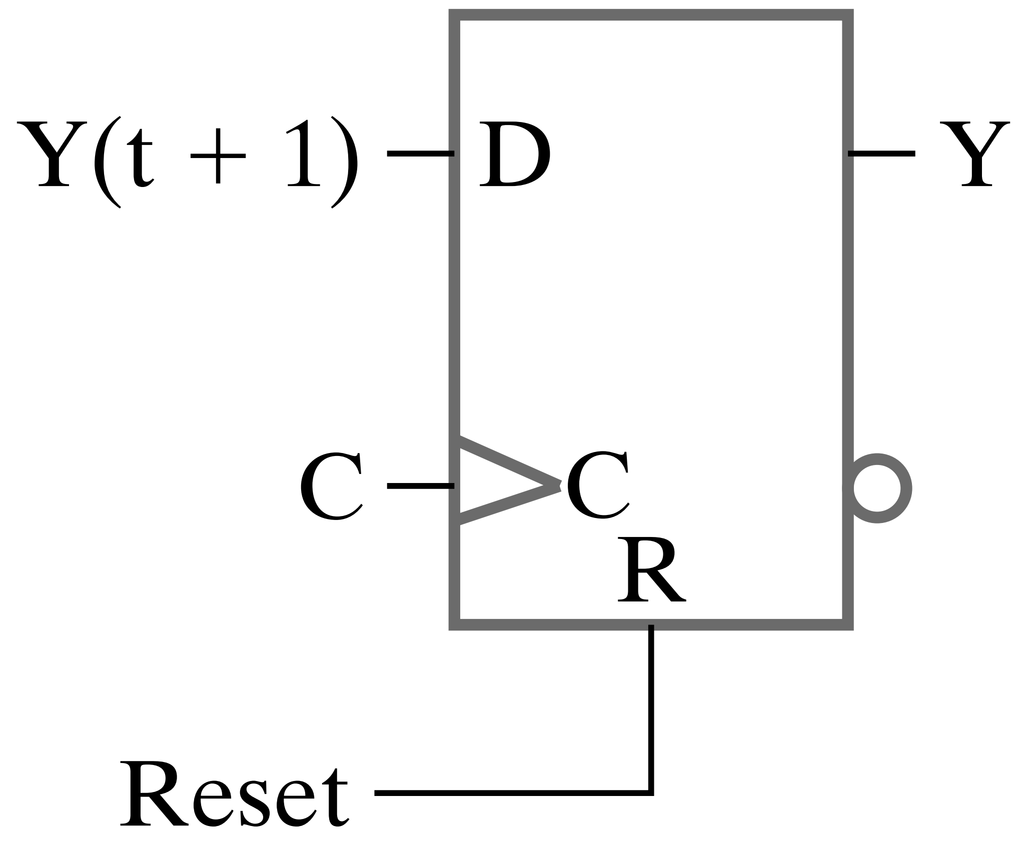
   TODAY'S FOCUS
3. **State Assignment**: assign binary codes to states
4. **Flip-Flop Input Equation Determination**: Select flip-flop types, derive input equations from next-state entries
5. **Output Equation Determination**: Derive output equations from the output entries
6. **Optimisation**
7. **Technology Mapping**
8. **Verification**

Concept

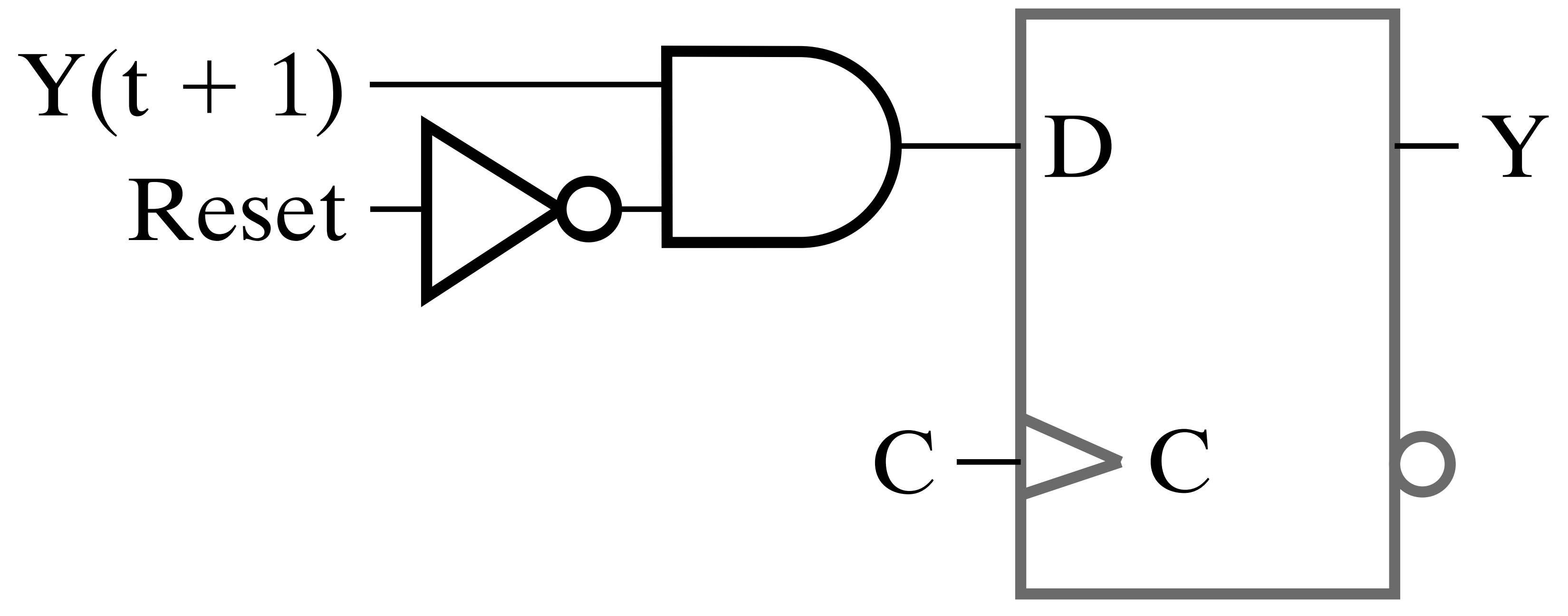
0. Reset

- When the power was first turned on, the states in flip-flops are all unknown
  This requires resetting!

- Flip-Flops with Reset



(a) Asynchronous Reset



(b) Synchronous Reset

-

0. Reset

- In reality, all storage devices in a computer has a Reset mode for easy reseting to all 1s or all 0s
- e.g. C: `memset( void* ptr, int value, size_t num);`
  Fill blocks of memory

P1
Design

1. Specification

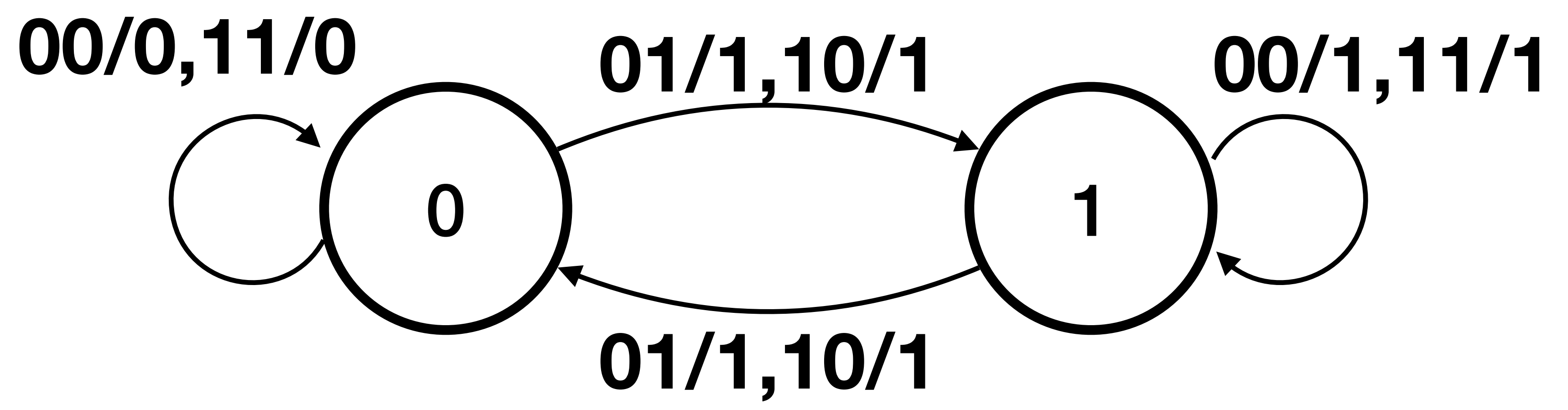
- Same as combinational

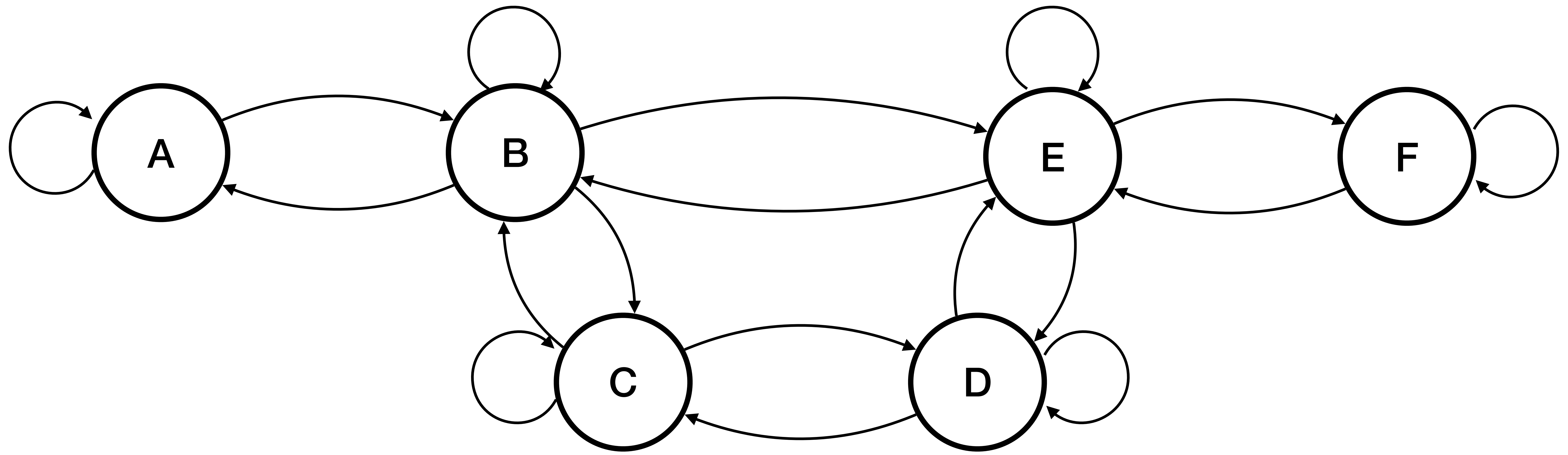
Concept

P1
Design

2. Formulation

- Sometimes it is more intuitive to describe state transitions then defining the states





Concept

2. Formulation

- Incrementer: perform +1 operation every `CLK` on 3-bit

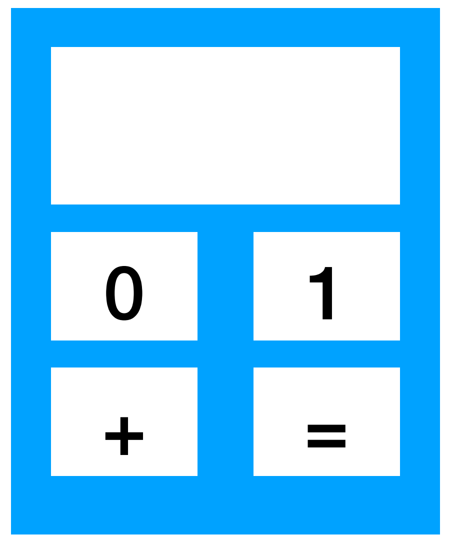
$$\begin{tikzcd}
000 \arrow[r] & 001 \arrow[r] & 010 \arrow[r] & 011 \arrow[d] \\
111 \arrow[u] & 110 \arrow[l] & 101 \arrow[l] & 100 \arrow[l]
\end{tikzcd}$$

Example

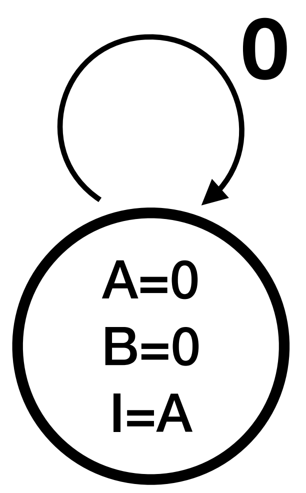
P1
Design

2. Formulation

- Simple 1-bit binary calculator
- Input: 0 button, 1 button, Add button, Equ button
- Storage: A, B: variables; I: operator



First Input



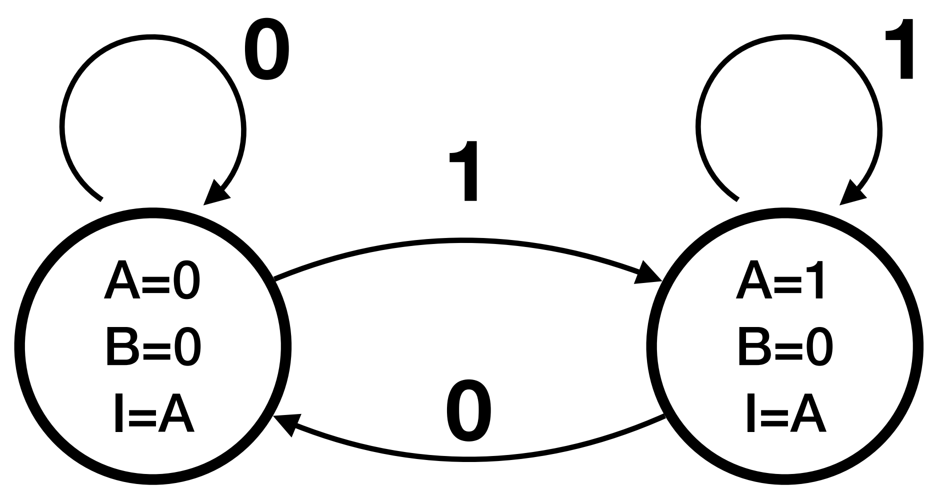
Example

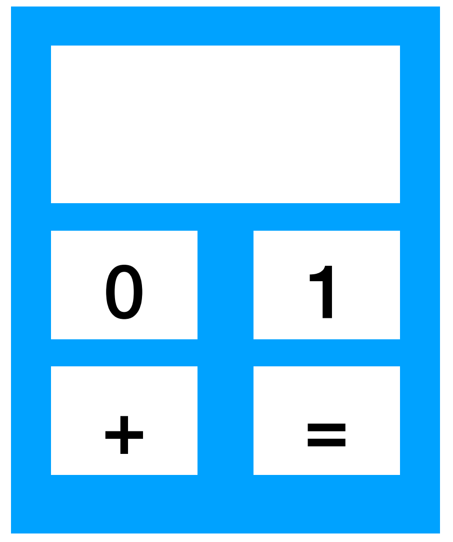
P1
Design

2. Formulation

- Simple 1-bit binary calculator
- Input: 0 button, 1 button, Add button, Equu button
- Storage: A, B: variables; I: operator

First Input



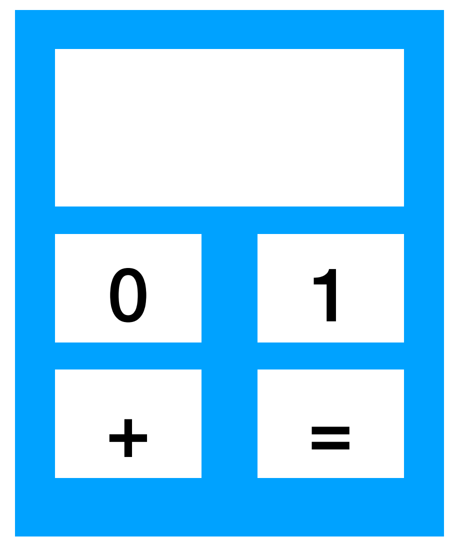


Example

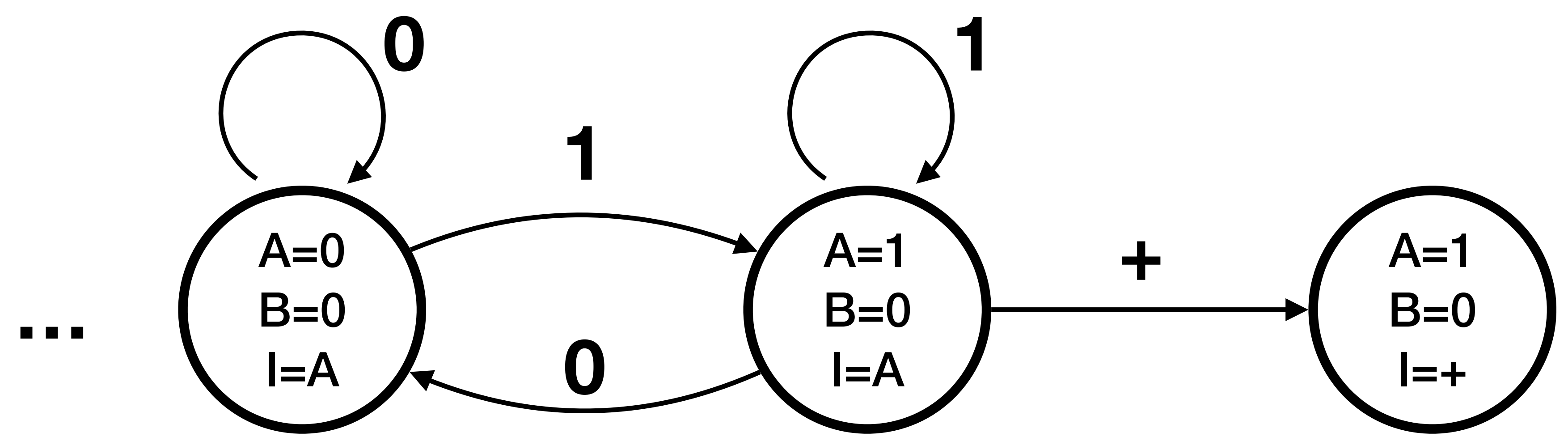
P1
Design

2. Formulation

- Simple 1-bit binary calculator
- Input: 0 button, 1 button, Add button, Equ button
- Storage: A, B: variables; I: operator



After first input, press add

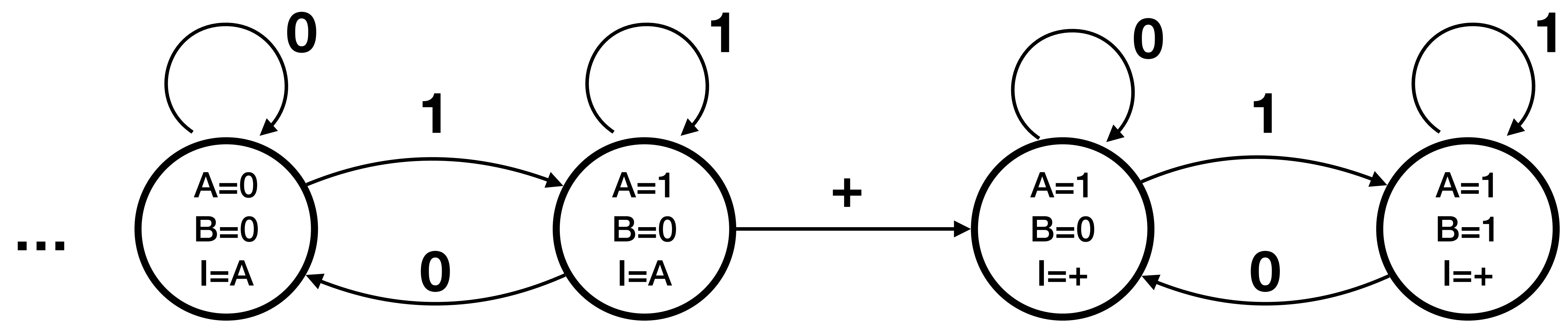


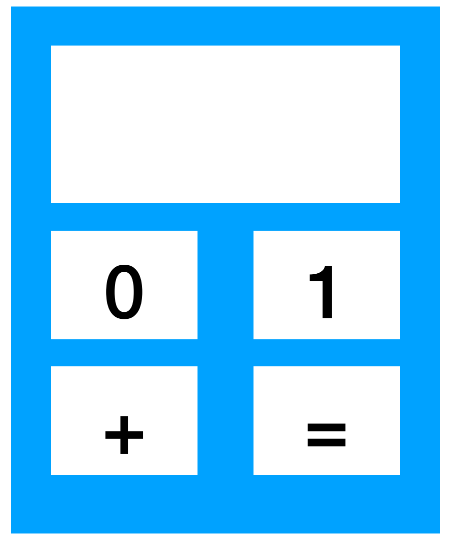
Example

2. Formulation

- Simple 1-bit binary calculator
- Input: 0 button, 1 button, Add button, Equ button
- Storage: A, B: variables; I: operator

Second input



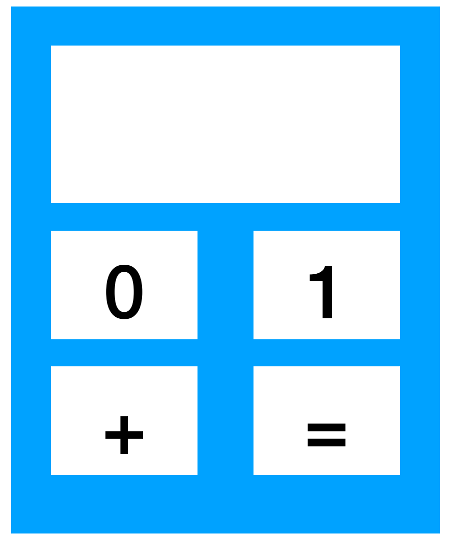


Example

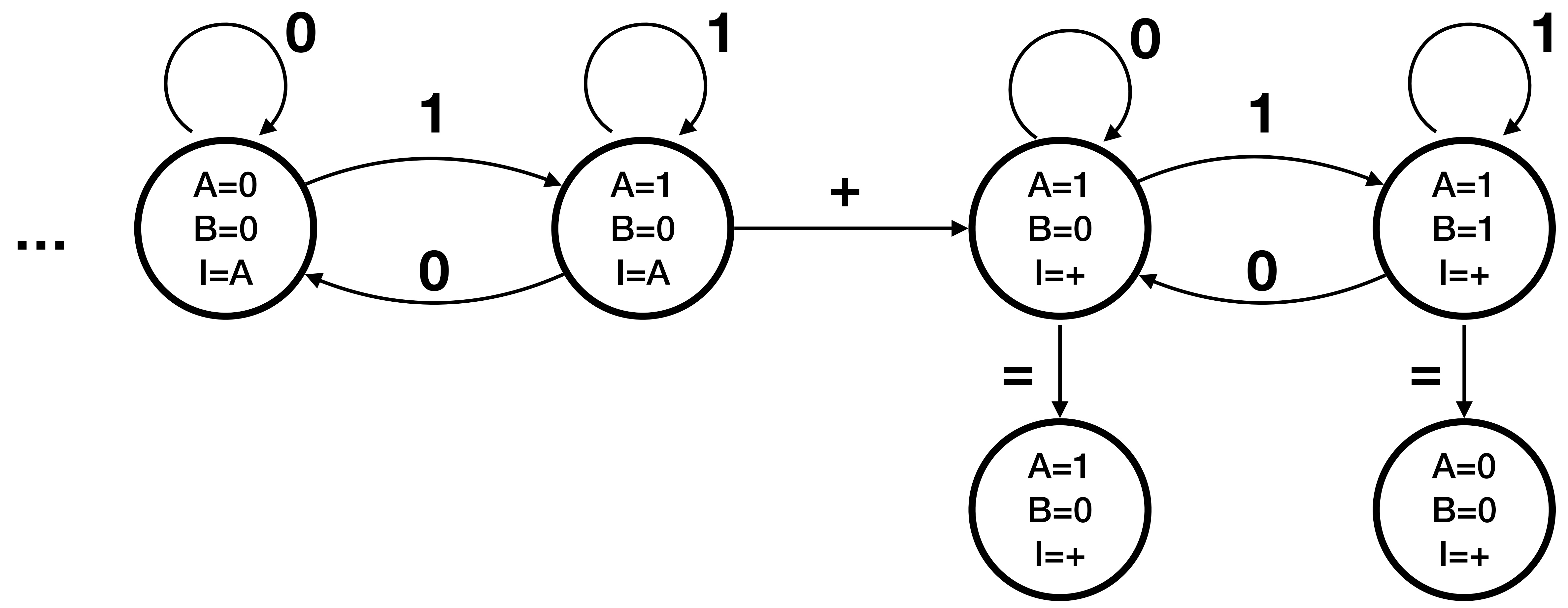
P1
Design

2. Formulation

- Simple 1-bit binary calculator
- Input: 0 button, 1 button, Add button, Equ button
- Storage: A, B: variables; I: operator



Get result

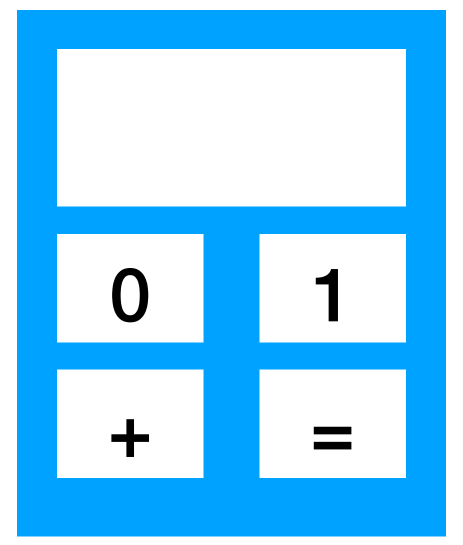


Example

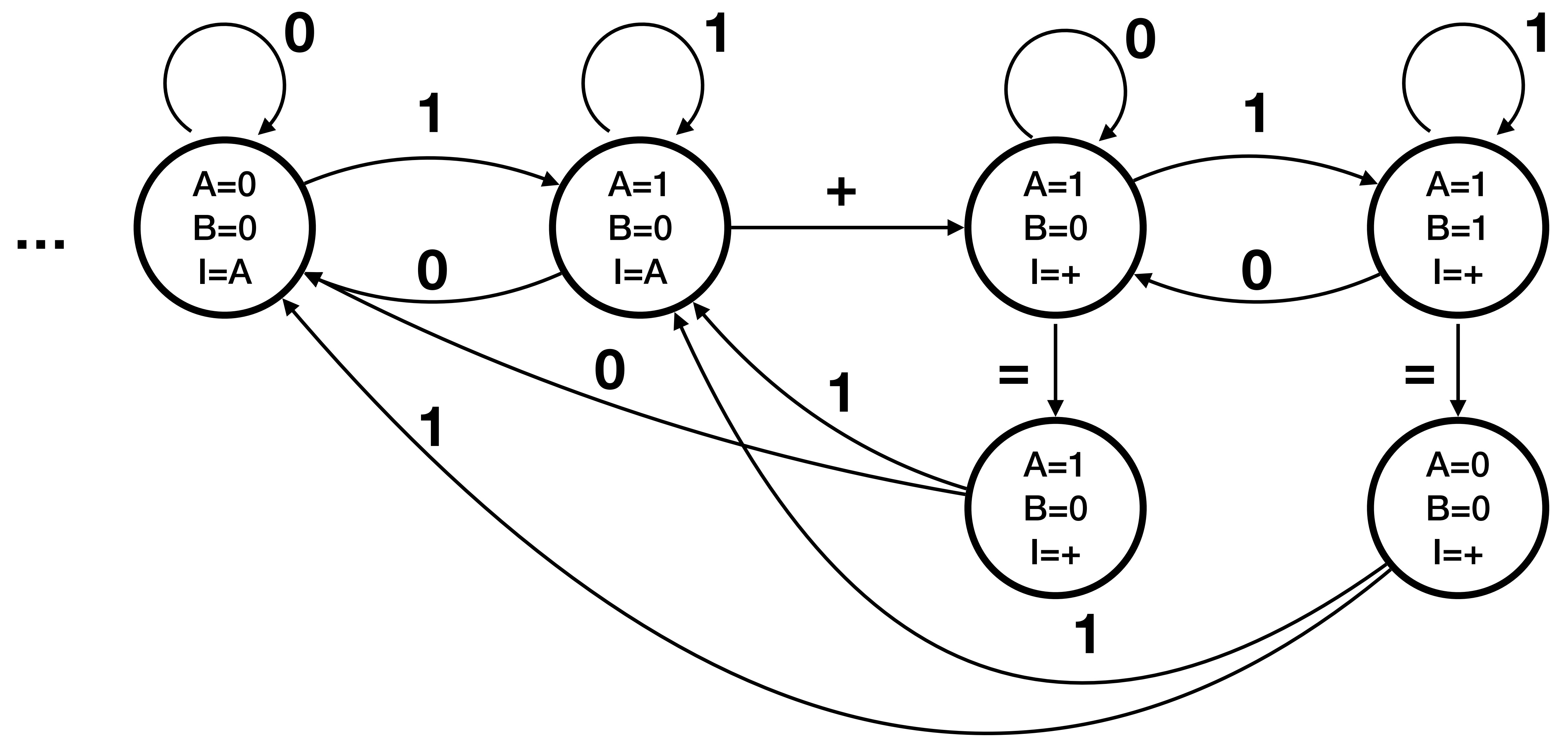
P1
Design

2. Formulation

- Simple 1-bit binary calculator
- Input: 0 button, 1 button, Add button, Equ button
- Storage: A, B: variables; I: operator



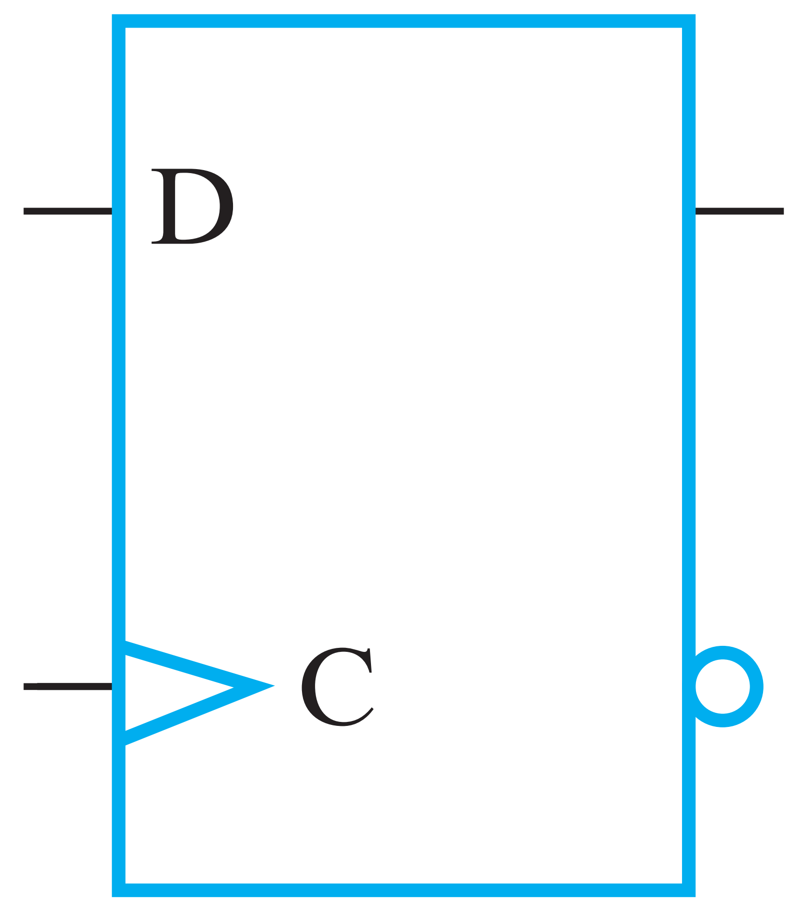
Next calculation



Example

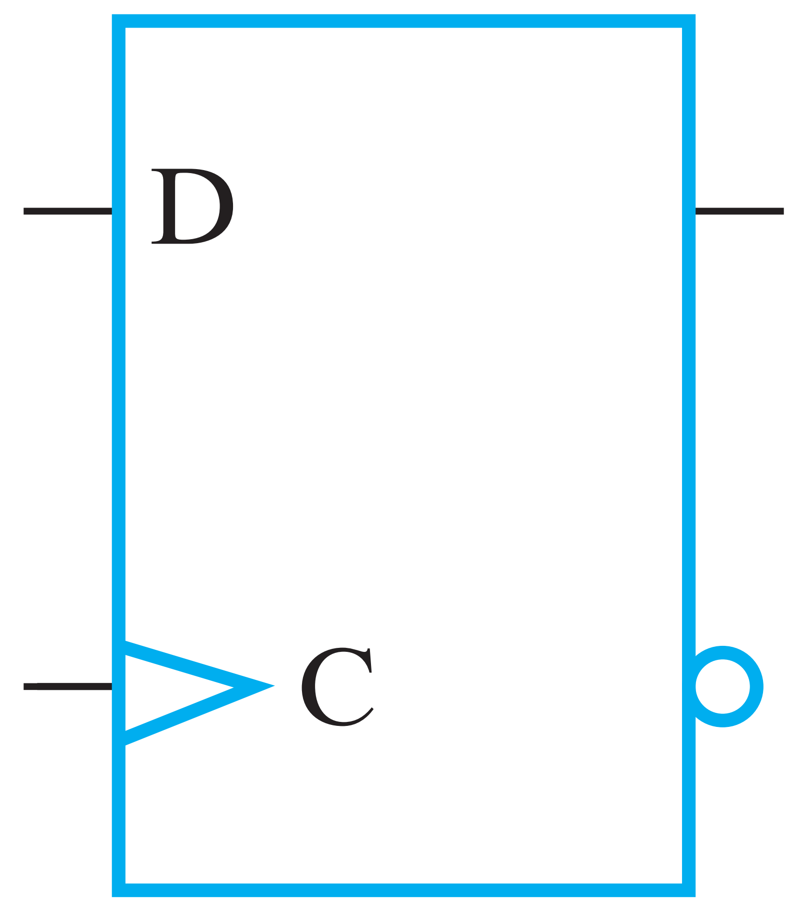
Exercise

- Draw the state diagram of 3-bit incrementer/decrementer
  - Input X: 0 for increment, 1 for decrement
- Do the state table



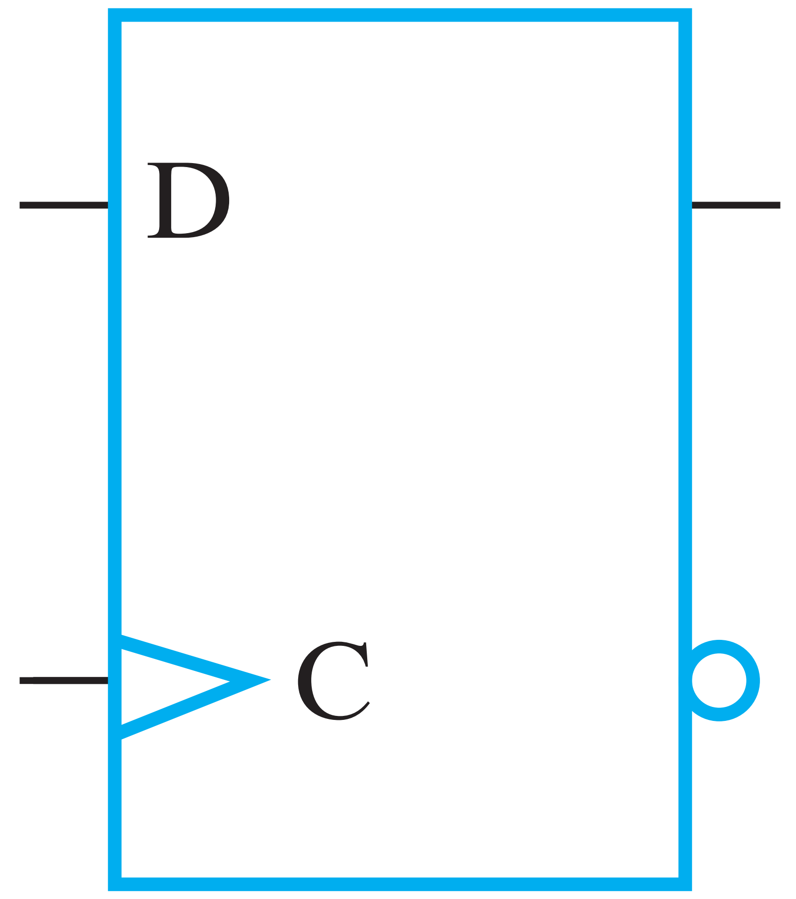
Exercise

- Draw the state diagram of rotator
  - Start state $X_3X_2X_1X_0$: original 4-bit
  - Input Y:
    0 for left rotation (output $X_2X_1X_0X_3$);
    1 for right rotation (output $X_0X_3X_2X_1$);
  - Every CLK triggers a shift



Exercise

- Draw the state diagram of rotator
  - Start state $X_3X_2X_1X_0$: original 4-bit
  - Input Y:
    0 for left rotation (output $X_2X_1X_0X_3$);
    1 for right rotation (output $X_0X_3X_2X_1$);
  - Every CLK triggers a shift
- Try to write down the equations for each flip-flop, treat X_i as constants. Implement in LogicWorks



P1
Design

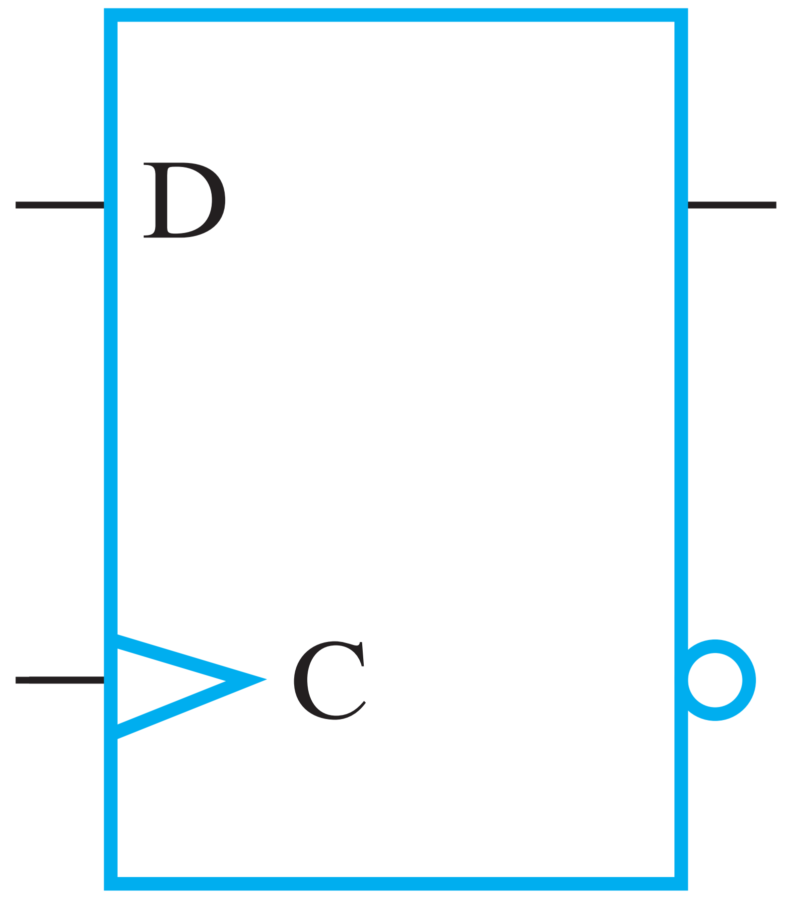
LogicWorks Exercise

- Implement D flip flop using D latch and SR latch
  Save it as a component in your library

- Implement circuit $D_S = X \oplus Y \oplus S$, where D_S is a D flip flop

- Implement $D_A = \overline{X}A + XY$, $D_B = \overline{X}B + XA$, $Z = XB$

- Draw the state table and diagram, and verify your table with LogicWorks



Exercise