Section 2

Overview

- Finite state machine
- Homework 2 preparation
- Q&A

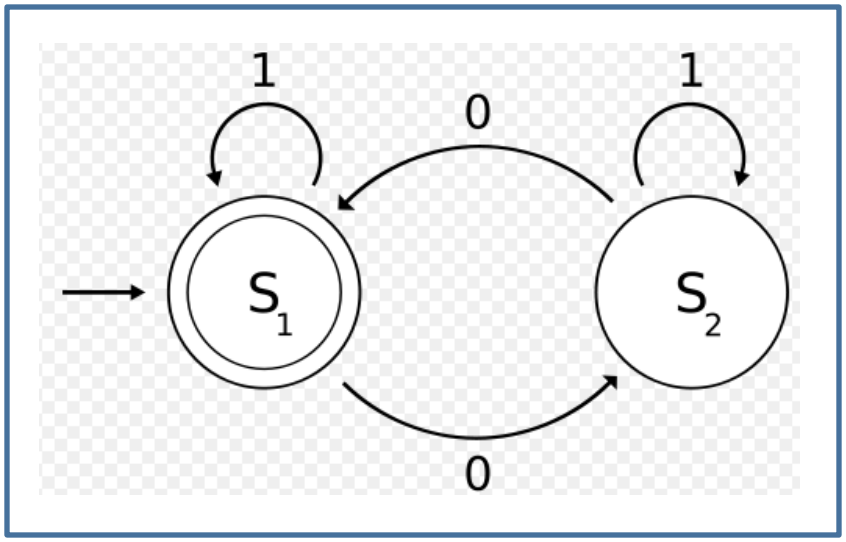
Finite State Machine

FSM

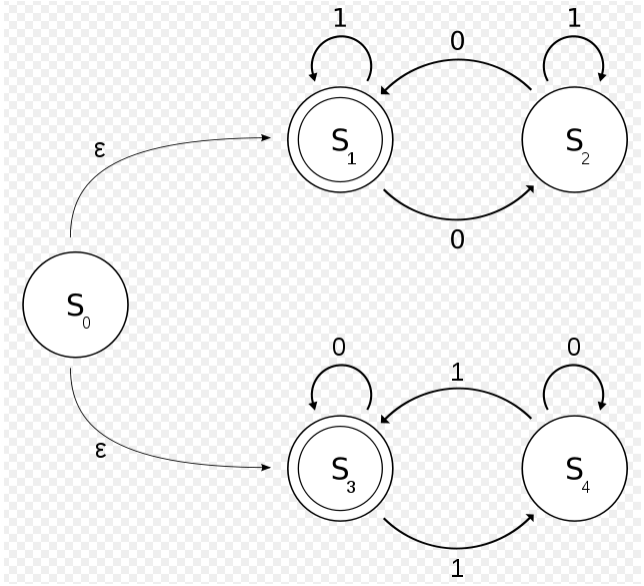
- Finite automata / State machines
- Mathematical abstraction
- Wide application
  - Digital design
  - Communication protocol
  - Language parsing
  - Neural System modeling
  - …

DFA and NFA

- Deterministic and Non-deterministic
- Deterministic means there is only one outcome for each state transition



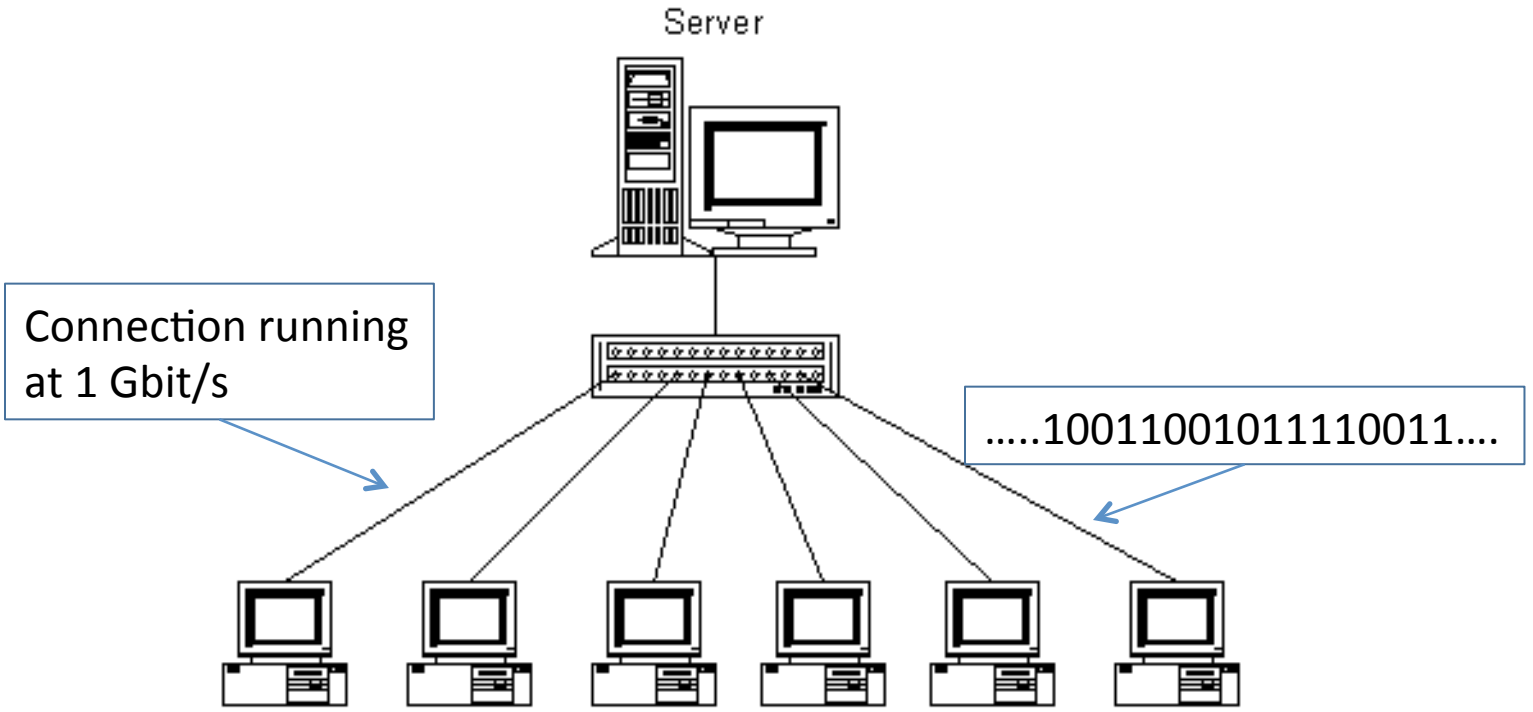
Deterministic



Non-deterministic

Design an FSM

- We will design a synchronization circuitry that is used in Gigabit Ethernet.



FSM example: sync_fsm

- FSM that looks for a specific sequence of bits, called *comma character*.
- The Gigabit Ethernet standard specifies a few comma characters,
  - 0011111001
  - 0011111010
  - 0011111000
- in this example, we will design for this pattern: 0011111xxx.
- The leading zero is the first bit received.

Example Bit Stream

- For example, if the incoming bit streams is 000110011100010011100011100111000111001110011100111000011001110001110000 ……
- The synchronization circuitry will catch the patterns in red, and flags the output sync bit to 1, (0 for not-in-sync).

Similar to …

- Remember when you learn about regular expression?
- ab*(c|ε) denotes the set of strings starting with *a*, then zero or more *b*s and finally optionally a *c*: {*a*, *ac*, *ab*, *abc*, *abb*, *abbc*, ...}
- The circuit we design operates on 0 or 1 instead of a, b, c…

Elements of FSM

- Combinational logic that computes next state
- Memory element that remembers the current state
- Input and output

State Machine Design Process

- 1. Determine the inputs and outputs
- sync_fsm
  - Input: bit_in , 1-bit
  - Output: comma_detect, 1-bit

State Machine Design Process

- 2. Determine the machine states
- sync_fsm, 0011111xxx
  - S0, initial state
  - S1, received first 0
  - S2, received second 0
  - S3 ~ S7, received 1s
  - S8 ~ S10, received either 0 or 1.
  - In S10, signal comma_detect

State Machine Design Process

- 3. Create state/bubble diagram – Mealy or Moore?

State Machine Design Process

- 4. State assignment – give each state a particular value.
  - We have 11 states
    - Needs at least 4 bits to encode (compact)
    - One hot encoding (minimize decoding logic)
  - We use the 4 bit encode.

State Machine Design Process

- 5. Create Transition/Output Table

Current State	Next State (Input == 0)	Next State (Input == 1)	Output
0000			0
0001			0
0010			0
0011			0
0100			0
0101			0
0110			0

Current State	Next State (Input == 0)	Next State (Input == 1)	Output
0111			0
1000			0
1001			0
1010			1

State Machine Design Process

- 6. Derive Next State Logic for each state element.

State Machine Design Process

- Derive Output Logic
- Ouput = S[3] & ~S[2] & S[1] & ~S[0]

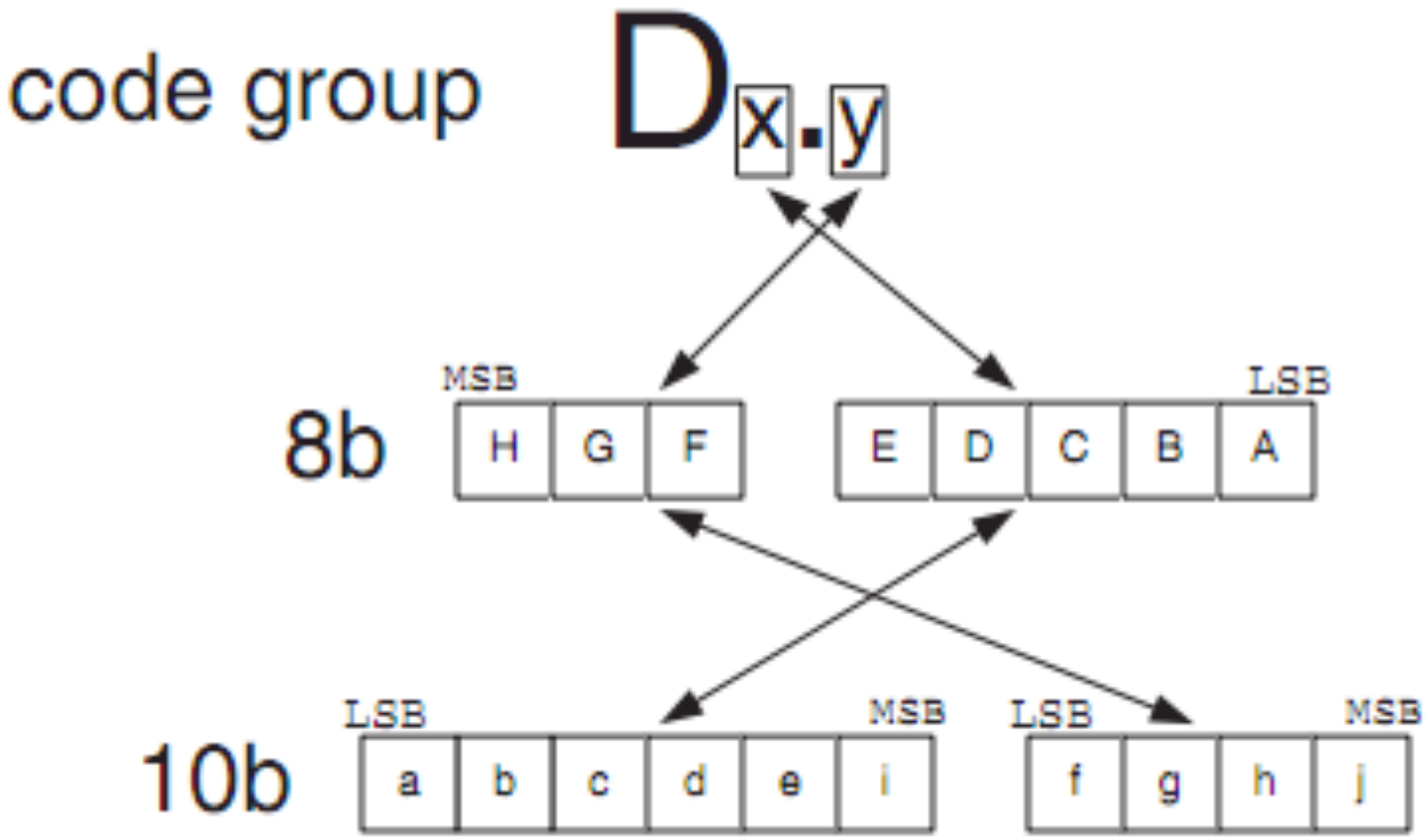
State Machine Design Process

- 8. Implement in Logisim

Homework 2 preparation

- Use tunnel
- Demo

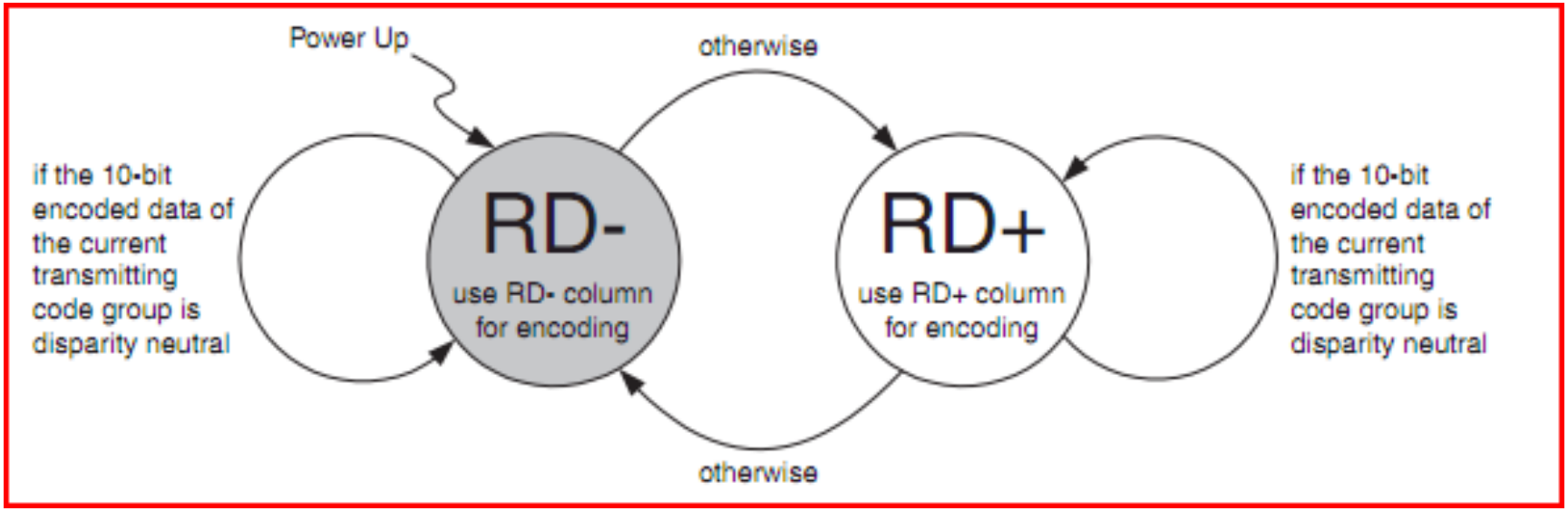
- 8-bit input => 10-bit output
- 8-bit input divided into
  - 3-bit blocks (HGF)
  - 5-bit blocks (EDCBA)
- 10-bit output divided into
  - 4-bit blocks (fghj)
  - 6-bit blocks (abcdei)


code group
Dx.y
8b
MSB
LSB
H
G
F
E
D
C
B
A
10b
LSB
MSB
LSB
MSB
a
b
c
d
e
i
f
g
h
j

Running Disparity

- The difference between the number of 1s transmitted and the number of 0s transmitted is always limited to ±2, and at the end of each state, it is either +1 or −1. This difference is known as the *running disparity* (RD).

- This scheme only needs two states for running disparity of +1 and −1. It starts at −1.


Power Up
otherwise
if the 10-bit encoded data of the current transmitting code group is disparity neutral
RD-
use RD- column for encoding
RD+
use RD+ column for encoding
if the 10-bit encoded data of the current transmitting code group is disparity neutral
otherwise

- Difference between RD- and RD+ for an encoding is XOR.
- Can minimize to figure out when to XOR and when not.

- Split the circuit into smaller components
  - Cleaner design and easier for testing
  - 5b/6b encoder, 3b/4b encoder, circuit to figure out more 1 or zero, fsm
- Find resource (or ask for help) to understand how the encoding works
  - Make sure any resource you find is the same as what we have on the appendix (Wikipedia isn't always right)

Table 1. 3-Bit to 4-Bit Encoding Values

3b Decimal	3b Binary (HGF)	4b Binary (fghi)
0	000	0100 or 1011
1	001	1001
2	010	0101
3	011	0011 or 1100
4	100	0010 or 1101
5	101	1010
6	110	0110
7	111	0001 or 1110 or 1000 or 0111

Table 2. 5-Bit to 6-Bit Encoding Values

5b Decimal	5b Binary (EDCBA)	6b Binary (abcdei)
0	00000	100111 or 011000
1	00001	011101 or 100010
2	00010	101101 or 010010
3	00011	110001
4	00100	110101 or 001010
5	00101	101001
6	00110	011001
7	00111	111000 or 000111
8	01000	111001 or 000110
9	01001	100101
10	01010	010101
11	01011	110100
12	01100	001101
13	01101	101100
14	01110	011100
15	01111	010111 or 101000
16	10000	011011 or 100100
17	10001	100011
18	10010	010011
19	10011	110010
20	10100	001011
21	10101	101010
22	10110	011010
23	10111	111010 or 000101
24	11000	110011 or 001100
25	11001	100110
26	11010	010110
27	11011	110110 or 001001
28	11100	001110
29	11101	101110 or 010001
30	11110	011110 or 100001
31	11111	101011 or 010100

Hints

- Split the circuit into smaller components
  - Cleaner design and easier for testing
  - 5b/6b encoder, 3b/4b encoder, circuit to figure out more 1 or zero, fsm
- Find resource (or ask for help) to understand how the encoding works
  - Make sure any resource you find is the same as what we have on the appendix (Wikipedia isn't always right)