Sequential Logic and State

Output depends on inputs \textit{and} stored values. (vs. combinational logic: output depends only on inputs)

Elements to store values: latches, flip-flops, registers, memory
Processor: Data Path Components

Instruction Fetch and Decode → Registers → ALU → Memory
Unstable circuit

How can $Q = \overline{Q}$?

Have this issue with any odd number of inverters in a loop.
Bistable latches

Things are more sensible with an even number of inverters in a loop.

Suppose we somehow get a 1 (or a 0?) on here.
SR latch

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>$Q_{\text{prev}}$</th>
<th>$Q'_{\text{prev}}$</th>
<th>$Q_{\text{next}}$ (stable)</th>
<th>$Q'_{\text{next}}$ (stable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>any</td>
<td>any</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>any</td>
<td>any</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>any</td>
<td>any</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

violates invariant that $Q$ and $Q'$ are inverses!
SR latch
if \( C = 0 \), then SR latch stores current value of \( Q \).

if \( C = 1 \), then \( D \) flows to \( Q \):

- if \( D = 0 \), then \( R = 1 \) and \( S = 0 \), \( Q = 0 \)
- if \( D = 1 \), then \( R = 0 \) and \( S = 1 \), \( Q = 1 \)

**Notes:**
- Data bit \( D \) replaces \( S \) & \( R \): it’s the bit value we want to store when \( \text{Clock} = 1 \)
  - Internally, Data bit \( D \) prevents bad case of \( S = R = 1 \)
- This logic is **level-triggered**: as long as \( \text{Clock} = 1 \), changes to \( D \) have impact
Time matters!

Assume Q has an initial state of 0
Clocks

**Clock**: free-running signal with fixed *cycle time = clock period = T.*

**Clock frequency** = 1 / clock period

A clock controls when to update a sequential logic element's state.
Synchronous systems

Inputs to state elements must be valid on active clock edge.
D flip-flop with falling-edge trigger

leader stores
\( D = Q_{\text{next}} \) as \( E \)

follower stores
\( E = Q_{\text{next}} \) as \( Q \)

Assume
\( Q = Q_{\text{now}} \)

\( Q \) still = \( Q_{\text{now}} \)

\( Q \) is now \( Q_{\text{next}} \)
Time matters!

Assume Q and E have an initial state of 0
Reading and writing in the same cycle

Assume Q is initially 0.

Moral: It’s OK to use the current output Q of a flip-flop as part of the next data input D to the same flip-flop.
D flip-flop = one bit of storage

The bit value of D when C has a falling edge is remembered at Q until the next falling edge of C.
A 1-nybble* register

*Half a byte!

(a 4-bit hardware storage cell)

Write

Clock
Register file

Array of registers, with register selectors, write/read control, input port for writing data, output ports for reading data.

- Read ports
- Why 2?

- Write port
- Write register selector
- Write data

- Register address #1
- Read data 1

- Register address #2
- Read data 2

- Write?
- 0 = read
- 1 = write

- r = log₂ number of registers
- w = bits in word
Read ports

Register address #1
\((\log_2 k)\) bits

- Register 0
- Register 1
- \(\ldots\)
- Register \(k - 2\)
- Register \(k - 1\)

Data read from register address #1
\((n \text{ bits})\)

Register address #2
\((\log_2 k)\) bits

Data read from register address #2
\((n \text{ bits})\)
Read port: Register address #1
\((\log_2 k \text{ bits})\)

Register address #2
\((\log_2 k \text{ bits})\)

Data read from register address #1
\((n \text{ bits})\)

Data read from register address #2
\((n \text{ bits})\)
Write port (data in)

Write control

Clock

Address of register to write to ($log_2 k$ bits)

m-to-2^m decoder

Data to write ($n$ bits)

Register 0

Register 1

Register $k - 2$

Register $k - 1$
Write port (data in)

- **Write control**
- **Clock**
- **Address of register to write to** ($\log_2 k$ bits)
- **Data to write** ($n$ bits)

The diagram shows a block diagram of a write port. It includes:

- An input for $n$ bits of data to write.
- An input for the address of the register to write to, which is $k-2$ to $k-1$ bits.
- Outputs for registers 0 to $k-2$.

The diagram also includes an m-to-$2^m$ decoder that selects the appropriate register based on the address.
RAM (Random Access Memory)

- A is number of words in RAM
- Specify the desired word by an address of size $\log_2 A$
- B is the width of each word (in bits)

Similar to register file, except...
16 x 4 RAM

4-bit address
1101

4 to 16 decoder

data out