Sequential Logic

Programming Language Analogy:

- One statement = Combinational Logic
  \[ \text{eg. } \text{Sum} = f(A, B, C) \]
  \[ \text{Sum} = A + B + C \]

- Multiple statements = Sequential Logic
  \[ \text{eg. } \text{Sum} = A + B; \]
  \[ \text{Sum} = \text{Sum} + C; \]
  \[ \text{Sum} = \text{Sum} + D; \]

Time/Space Tradeoff:

Combinational: fast, expensive
Sequential: slower, cheaper

* Next way to save output of one step to be used on the next step.

\[ \Rightarrow \text{Register} \]

Abstraction

\[ \text{clk} \]

\[ \text{step } 0, \text{ step } 1, \text{ step } 2, \text{ step } 3 \ldots \]
Register

\[ D \rightarrow Q \]

\[ CLK \]

Close-up of clock edge

\[ tsu = \text{setup time} \] \{ must obey \}
\[ th = \text{hold time} \] \{ must obey \}
\[ tpd = \text{"clk-to-Q"}, \text{propagation delay} \]

\[ * \Rightarrow \text{All registers sample inputs simultaneously} \]

Example:

\[ \text{Step 0: 0} \]
\[ 1: 3 \]
\[ 2: 5 \]
\[ 3: 2 \]
\[ 4: 1 \]

You complex shift registers:

* PARALLEL LOAD
* SHIFT/NO SHIFT
* RIGHT/LEFT
* PARALLEL OUTPUT
Example

\[ \text{in} \rightarrow \text{next} \rightarrow \text{current} \rightarrow \text{out} \]

Clock ↑ causes current ← next

State Diagram

Step
0 1 2 3 4 5 6

n-bit register

\[ D[n-1:0] \xrightarrow{m} D \xrightarrow{n} D[n-1:0] \]

Register with Reset

\[ D \xrightarrow{\text{reset}} \]

\[ \text{out} \]
Enabled Register

Register samples on CLK 1 only when CE = 1.

Example

<table>
<thead>
<tr>
<th>CLR</th>
<th>EN</th>
<th>CURRENT</th>
<th>NEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
"Counters"

1-Bit Binary Counter

2-Bit Binary Counter

3-Bit Binary Counter

n-Bit Binary Counter
Binary Counters

1 Bit

2 Bits

3 Bits

<table>
<thead>
<tr>
<th>Q^t</th>
<th>Q^{t+1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
State Diagrams

Graphical representation of a Sequential Circuit. Registers contain the **STATE**. In registers $\Rightarrow 2^n$ states. Circuit may do something different in each different state.

Inputs $\rightarrow$ CL $\rightarrow$ Outputs

Combination

Inputs $\rightarrow$ CL $\rightarrow$ Outputs

Current State $\rightarrow$ Next State

State Diagram

Inputs/Outputs

State Transition

Arrows: Change in state caused by clock tick. Each state has one outgoing arrow.
States Diagram (cont)

Give input order/ output order
Reduce # of areas by using X's for inputs

Example

\[\text{State Diagram}\]

\[\text{State Table}\]
Example

State Table

<table>
<thead>
<tr>
<th>Current in</th>
<th>S_i, S_o</th>
<th>next s'_i, s'_o</th>
<th>out</th>
</tr>
</thead>
</table>

State Diagram
Design Problem

A circuit has two inputs A, B, and one output F. A, B, F start at 0.
The circuit waits for A=1, then B=1, and then turns on F.
If B=1 happens before A=1, F is never turned on.
MEMORY (RAM) (1-D Array, Vector)

- \(N\) locations, \(M\) bits per location (word size)
- Address: \(\log N\) bits
- \(\Rightarrow\) \(N\) \(M\)-bit registers
- E.g., 4 locations, 8-bit words.

WRITE:
- \(i_f(Write)\) \(\text{MEM}[Write\ Address] = Write\ Data;\)
- Each clock we can write a value into 1 location.
- Write Address \(\rightarrow\) \(\log N\) specifies which register.
- Write Data \(\rightarrow\) \(M\) gives the data to be stored.
- Write \(\rightarrow\) Control specifies whether to write.

How do we implement the WRITE function?

READ:
- \(\text{Read Data} = \text{MEM}[Read\ Address]\):
- We can read one location at a time. (only look at it)
- Read Address \(\rightarrow\) \(\log N\) specifies which register.
- Read Data \(\leftarrow\) \(M\) Data in register.
- Non-Destructive \(\Rightarrow\) No Read Control.

How do we implement the READ function?
Example: \(4 \times 8\) Memory

WORD SIZE

Write Data \(\times 8\)

Write Address \(\times 8\)

Write

Read Data

Read Address

Extend to any size memory!

---

**PROCESSOR**

Each instruction performs an operation on data.

\[ A = B + C; \]

\[ Z = X - Y; \]

\[ i = i + 1; \]

*Variables are stored in memory.*

*Each variable is allocated to a location.*

- Compiler does this assignment:
  \[ A = B + C \Rightarrow M[1] \leftarrow M[3] + M[5]; \]
  1 Machine instruction

For each instruction:

- Read 2 values from memory
- Perform operation
- Write result back to memory