Bit-sliced design:
- Design for one bit-slice
- Replicate the design for the entire bit pattern

- P: input bits
- M bits from prev. bit-slice
- Q: O/P bits
- M bits to next bit-slice
**Key idea:**

- Replication: have same bit-slice do the task.
- Pass information from one bit-slice to the next.

**Q:** Can we do this differently?

**Serialization**

- Use FF to store information for next use
- Use FF to store information for next use
- Same bit slice do the task repeatedly.
* P input, Q output, M bits of information (just like bit-sliced)
* M bits need to go back to the bit-slice (feedback)
* Accommodate initial values using select logic
Comparison of Serialized and bit-sliced design:

1. Area (number of gates)
1. Area (number of gates)
   For operations involving a small number of bits \textit{bit-slice} is smaller
   For operations involving a large number of bits \textit{serialized} is smaller

2. Performance (delay through the critical path):
   \textit{serialized} is slower because result needs to wait for the clock (FP) for every bit-slice operation.

\underline{Counters}\n\underline{Clocked} \underline{Sequential} \underline{circuit}
\cdot \text{Goes through sequence of distinct states with the}
- Goes through sequence of clock pulse
- Sequence of states can be user-defined
- All FF use same clock - synchronous
don't use same clock - asynchronous

Synchronous counter
Down-counter
7 → 6 → 5 → 4 → 3 → 2 → 1 → 0 → 7
8 states, 3 FF
Next-state table

<table>
<thead>
<tr>
<th>$S_2 S_1 S_0$</th>
<th>$S^+_2 S^+_1 S^+_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1</td>
<td>1 1 0</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 0 1</td>
</tr>
</tbody>
</table>
$S_2^+, S_1^+, S_0^+$

$S_2 = S_2' S_1 S_0 + S_2 S_0 + S_2 S_1$
\[ s_1^+ \]

\[ s_2^+ \]

\[ \begin{array}{c|cc}
0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 \\
\end{array} \]

\[ s_1 s_0 \]

\[ s_0 \]

\[ \begin{array}{c|cc}
0 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 \\
\end{array} \]

\[ s_0^+ = s_0 \]

\[ s_1^+ = s_1 s_0^+ + s_1 s_0 \]
Asynchronous

Ripple counter

Output of FF triggers remaining flip-flops.

3-bit binary counter

000 → 001 → 010 → 011 → 100 → 101 → 110 → 111