Chapter 7

Storage Components
Chapter preview

- Boolean algebra
- Logic gates and flip-flops
- Finite-state machine
- Logic design techniques
- Sequential design techniques
- Binary system and data representation
- Combinational components
- Storage components
- Generalized finite-state machines
- Register-transfer design
- Processor components
Storage components

- Storage components store data and perform some simple operations.

- Storage components include:
  - registers
  - counters
  - register files
  - queues
  - stacks

- Combinatorial and storage components are used for construction of:
  - datapaths
  - controllers

which are main subsystems of modern processors and other microchips.
Registers

- Registers are bit wise extensions of flip-flops.
- Registers store one dataword.

$Q_i = I_i$ when $Clk = \uparrow$
Registers with asynchronous set and reset

- Asynchronous setting and resetting is independent of clock signal.
- Asynchronous inputs are used to initialize the register.

![Register schematic](Image)

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Clk</th>
<th>Qᵢ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>Iᵢ</td>
</tr>
</tbody>
</table>
Parallel load register can hold data indefinitely. It can also load new data when load signal is 1.
Serial-in, parallel-out shift register

- Serial-in, parallel-out register converts serial data stream into parallel data stream.

![Register schematic](image)

<table>
<thead>
<tr>
<th>Present state</th>
<th>Next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift</td>
<td>Q_3 Q_2 Q_1 Q_0</td>
</tr>
<tr>
<td>0</td>
<td>No change</td>
</tr>
<tr>
<td>1</td>
<td>I_L Q_3 Q_2 Q_1</td>
</tr>
</tbody>
</table>

Graphic symbol

Operation table

I_L
Shift
Clk

Q_0 Q_1 Q_2 Q_3
Shift Register

1 0 Selector
D_3 Q_3

1 0 Selector
D_2 Q_2

1 0 Selector
D_1 Q_1

1 0 Selector
D_0 Q_0

Y_3 Y_2 Y_1 Y_0

Register schematic
Shift register with parallel load

**Graphic symbol**

**Present state**

<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$S_0$</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>No change</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Load input</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Shift left</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Shift right</td>
</tr>
</tbody>
</table>

**Next state**

<table>
<thead>
<tr>
<th>$Q_3$</th>
<th>$Q_2$</th>
<th>$Q_1$</th>
<th>$Q_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_3$</td>
<td>$I_2$</td>
<td>$I_1$</td>
<td>$I_0$</td>
</tr>
<tr>
<td>$Q_2$</td>
<td>$Q_1$</td>
<td>$Q_0$</td>
<td>$I_R$</td>
</tr>
<tr>
<td>$I_L$</td>
<td>$Q_3$</td>
<td>$Q_2$</td>
<td>$Q_1$</td>
</tr>
</tbody>
</table>

**Operation table**

- $D_0 = S_1'S_0'Q_0 + S_1'S_0I_0 + S_1S_0'I_R + S_1S_0Q_1$
- $D_i = S_1'S_0'Q_i + S_1'S_0I_i + S_1S_0'I_{i-1} + S_1S_0Q_{i+1}$
- $D_3 = S_1'S_0'Q_3 + S_1'S_0I_3 + S_1S_0'Q_2 + S_1S_0I_L$

**Register schematic**
4-bit binary counter

- Counters increment (decrement) their content when enabled.

**Operation table**

<table>
<thead>
<tr>
<th>$E$</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No change</td>
</tr>
<tr>
<td>1</td>
<td>Count</td>
</tr>
</tbody>
</table>

**HA truth table**

<table>
<thead>
<tr>
<th>$Q_i$</th>
<th>$C_i$</th>
<th>$C_{i+1}$</th>
<th>$D_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Counter schematic**

- $D_i = Q_i \oplus C_i$
- $C_{i+1} = Q_i C_i$
4-bit up/down binary counter

**Graphic symbol**

**Operation table**

<table>
<thead>
<tr>
<th>E</th>
<th>D</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>No change</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Count up</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Count down</td>
</tr>
</tbody>
</table>

**HAS truth table**

<table>
<thead>
<tr>
<th>E</th>
<th>D</th>
<th>Q_i</th>
<th>C_i</th>
<th>C_{i+1}</th>
<th>D_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Counter schematic**

\[ D_i = Q_i \oplus C_i \]

\[ C_{i+1} = D'Q_iC_i + DQ_i'C_i \]
4-bit up/down binary counter with parallel load

- This counter is sometimes called presettable counter.

<table>
<thead>
<tr>
<th>Load</th>
<th>E</th>
<th>D</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>No change</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Count up</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Count down</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>Load the input</td>
</tr>
</tbody>
</table>

Graphic symbol

Operation table

Register schematic
**BCD counters**

- Up sequence: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, ....
- Down sequence: 0, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 9, ....

- Up counter loads 0 when counter content is 9 (1001)
- Up/down counter loads 0 when counter content is 9 (1001) and direction bit D=0
- Up/down counter loads 9 when counter content is 0 (0000) and direction bit D=1
Asynchronous counters

- Each FF in synchronous counters changes its output at the same time.

- FFs in asynchronous counters change values at different times.

- Advantage of asynchronous counters is low cost (less gates).

- Weakness of asynchronous counters is longer delays in comparison with synchronous CLA counters.
4-bit asynchronous up counter

Graphic symbol: 
- Clk (Clock)
- E (Enable)
- Reset

Logic schematic:
- T3 Q3
- T2 Q2
- T1 Q1
- T0 Q0

States:
- Q3
- Q2
- Q1
- Q0

Timing diagram:
- Clk: 0 1 2 3 4 5 6 7 8
- Q3: 0 1 2 3
- Q2: 0 1 2
- Q1: 0 1 2
- Q0: 0 1

Transition times:
- t0
- t1
- t2
- t3
- t4
- t5
- t6
- t7

Copyright © 2004-2005 by Daniel D. Gajski
Slides by Zheng Zhao, University of California, Irvine
Mixed-mode counter consists of
(1) asynchronous counters connected synchronously
(2) synchronous counters connected asynchronously
Register-file

- Register-file is used as fast temporary storage.
Register-file with 1 write port and 2 read ports

- This register-file is used for reading two operands and writing one result in each clock cycle.
Random access memory (RAM)

<table>
<thead>
<tr>
<th>MEMORY ADDRESS</th>
<th>MEMORY CONTENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>Decimal</td>
</tr>
<tr>
<td>0...000</td>
<td>0</td>
</tr>
<tr>
<td>0...001</td>
<td>1</td>
</tr>
<tr>
<td>0...010</td>
<td>2</td>
</tr>
<tr>
<td>0...011</td>
<td>3</td>
</tr>
<tr>
<td>0...100</td>
<td>4</td>
</tr>
<tr>
<td>0...101</td>
<td>5</td>
</tr>
<tr>
<td>0...110</td>
<td>6</td>
</tr>
<tr>
<td>0...111</td>
<td>7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1...110</td>
<td>$2^{n-2}$</td>
</tr>
<tr>
<td>1...111</td>
<td>$2^{n-1}$</td>
</tr>
</tbody>
</table>

Memory address and content

Graphic symbols

Copyright © 2004-2005 by Daniel D. Gajski
RAM organization

- Ram memory cells can be static or dynamic.
- Static RAM’s do not lose data with time.
- Dynamic RAM’s must be refreshed.

Memory schematic
RAM timing

Read-cycle timing:
- **Address:** Valid address
- **Data:** Valid data
- **Output-enable time:** $t_1 - t_2$
- **Access time:** $t_2 - t_3$
- **Output-disable time:** $t_4 - t_5$

Write-cycle timing:
- **Address:** Valid address
- **Data:** Valid data
- **Address-hold time:** $t_0 - t_1$
- **Data-setup time:** $t_1 - t_2$
- **Write pulse width:** $t_2 - t_3$
- **Address-hold time:** $t_4 - t_5$
16K x 32 RAM design with 16K x 8 RAMs
64K x 8 RAM design with 16K x 8 RAMs

Addresses

0
...
$2^{14} - 1$

$2^{14}$
...

$2^{15} - 1$

$2^{15}$
...

$2^{15} + 2^{14} + 1$

$2^{15} + 2^{14}$
...

$2^{16} - 1$
Push-down stack principle

Stack content before 45 is pushed down
Stack content after 45 is pushed down
Stack content before 45 is popped up
4-Word push-down stack

<table>
<thead>
<tr>
<th>Push/Pop</th>
<th>Enable</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0</td>
<td>No change</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Push</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Pop</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Push/Pop Enable</th>
<th>Shift register controls</th>
<th>Counter controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0 0</td>
<td>X 0</td>
</tr>
<tr>
<td>0</td>
<td>1 1</td>
<td>0 1</td>
</tr>
<tr>
<td>1</td>
<td>1 0</td>
<td>0 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Counter outputs</th>
<th>Empty</th>
<th>Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q2   Q1   Q0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0    0    0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0    0    1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0    1    0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0    1    1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1    0    0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Operations:
- Push = Shift right
- Pop = Shift left

Control logic
- Reset
- Enable
- Push/pop
- Output logic

Stack schematic
Push-down stack with a 1K RAM

Symbolic design:

<table>
<thead>
<tr>
<th>Push/Pop</th>
<th>Enable</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0</td>
<td>No change</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Push</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Pop</td>
</tr>
</tbody>
</table>

Operation table:

<table>
<thead>
<tr>
<th>Push/Pop Enable</th>
<th>Selector control</th>
<th>Memory controls</th>
<th>Counter controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Control table:

<table>
<thead>
<tr>
<th>Push/Pop Enable</th>
<th>Selector control</th>
<th>Memory controls</th>
<th>Counter controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Stack schematic:

- Push: Data \(\rightarrow\) RAM (TOP); Increment Top, Top 1
- Pop: RAM (Top-1) \(\rightarrow\) Date; Decrement Top, Top-1
- Stack is full when Top=1023; Stack is empty when Top=0
- Location with address 1023 is never loaded
FIFO queue principles

- Queue is used for irregular “bursty” communication

```
Top
empty
empty
empty
Top-1
empty
34
Top-2
34
Top-3
23
```

Queue content before 45 is stored

```
empty
empty
empty
```

Queue content after 45 is stored

```
empty
45
empty
```

Queue content after 23 is read

```
empty
empty
23
```

Copyright © 2004-2005 by Daniel D. Gajski

Slides by Zheng Zhao, University of California, Irvine
4-word FIFO queue

<table>
<thead>
<tr>
<th>READ/WRITE ENABLE</th>
<th>OPERATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>No change</td>
</tr>
<tr>
<td>0</td>
<td>Read</td>
</tr>
<tr>
<td>1</td>
<td>Write</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>READ/WRITE ENABLE</th>
<th>S₁</th>
<th>S₀</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Operation table

Control table

Queue schematic
FIFO queue implemented with a 1K RAM
Simple datapath with one accumulator

- Datapath are used for temporary variable storage and operation execution

**Datapath schematic**

**Input select**  | **ALU controls**  | **Shift values**  | **Accumulator controls**  | **Output enable**
---|---|---|---|---

Control word: 29
Datapath with 3 port register-file

<table>
<thead>
<tr>
<th>M</th>
<th>S_1</th>
<th>S_0</th>
<th>ALU Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Complement A</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>AND</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>EX-OR</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>OR</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Decrement A</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Add</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Subtract</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Increment A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S_2</th>
<th>S_1</th>
<th>S_0</th>
<th>Shift Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Pass</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Pass</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Not used</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Not used</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Shift left</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Rotate left</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Shift right</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Rotate right</td>
</tr>
</tbody>
</table>

Datapath schematic

Control word

Copyright © 2004-2005 by Daniel D. Gajski

Slides by Zheng Zhao, University of California, Irvine
One’s-count algorithm

Example: One’s-counter implementation

Problem: Using a datapath with a 3 port register-file, design a one’s counter that will count the number of ones in an input dataword, and return the result after completion.

1. Data := Input
2. Ocount := 0
3. Mask := 1
   while Data ≠ 0 repeat
   4. Temp := Data AND Mask
   5. Ocount := Ocount + Temp
   6. Data := Data >> 1
   end while
7. Output := Ocount

Basic algorithm for one’s counter

Register assignment

<table>
<thead>
<tr>
<th>Control words</th>
<th>IE</th>
<th>Write address</th>
<th>Read address A</th>
<th>Read address B</th>
<th>ALU operation</th>
<th>Shifter operation</th>
<th>OE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>R₁</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>R₃</td>
<td>0</td>
<td>0</td>
<td>Add</td>
<td>Pass</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>R₂</td>
<td>0</td>
<td>X</td>
<td>Increment</td>
<td>Pass</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>R₄</td>
<td>R₁</td>
<td>R₂</td>
<td>AND</td>
<td>Pass</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>R₃</td>
<td>R₃</td>
<td>R₄</td>
<td>Add</td>
<td>Pass</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>R₁</td>
<td>R₁</td>
<td>0</td>
<td>Add</td>
<td>Shift right</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>None</td>
<td>R₃</td>
<td>0</td>
<td>Add</td>
<td>Pass</td>
<td>0</td>
</tr>
</tbody>
</table>

Control words for one’s counter

Repeated while Data ≠ 0
FSM representation of One’s-counter

- State lasts for a clock cycle.
- In each state the datapath executes the statement indicated on its right side.

Start = 0

- Start = 0

Start = 1

- Data = Input

Done = 1

- Ocoun = 0

- Mask = 1

Data ≠ 0

- Temp = Data AND Mask

Data = 0

- Ocoun = Ocoun + Temp

Data = Data >> 1 (shift right)

Output = Ocoun
Next-state logic for One’s-counter

<table>
<thead>
<tr>
<th>States</th>
<th>$Q_2Q_1Q_0$</th>
<th>$Q_2Q_1Q_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>0 0 0</td>
<td>000 000 001 001</td>
</tr>
<tr>
<td>$s_1$</td>
<td>0 0 1</td>
<td>010 010 010 010</td>
</tr>
<tr>
<td>$s_2$</td>
<td>0 1 0</td>
<td>011 011 011 011</td>
</tr>
<tr>
<td>$s_3$</td>
<td>0 1 1</td>
<td>100 100 100 100</td>
</tr>
<tr>
<td>$s_4$</td>
<td>1 0 0</td>
<td>101 101 101 101</td>
</tr>
<tr>
<td>$s_5$</td>
<td>1 0 1</td>
<td>110 110 110 110</td>
</tr>
<tr>
<td>$s_6$</td>
<td>1 1 0</td>
<td>100 111 100 111</td>
</tr>
<tr>
<td>$s_7$</td>
<td>1 1 1</td>
<td>000 000 000 000</td>
</tr>
</tbody>
</table>

Next-state table

$Q_2(next) = Q_2'Q_1Q_0 + Q_2Q_1'(Data\neq0)Q_2Q_0'$

$Q_1(next) = Q_1'Q_0 + Q_2'Q_1Q_0'(Data\neq0)Q_1Q_0'$

$Q_0(next) = Q_2'Q_1Q_0' + Q_2Q_1'Q_0' + Start Q_2'Q_0'(Data\neq0)Q_2Q_0'$

Next-state equations

Karnaugh map
Output logic for One’s-counter controller

Output logic table

\[
\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|}
\hline
\text{State} & Q_2Q_1Q_0 & \text{IE} & \text{WA}_2 \text{WA}_1 \text{WA}_0 & \text{WE} & \text{RAA}_2 \text{RAA}_1 \text{RAA}_0 & \text{REA} & \text{RAB}_2 \text{RAB}_1 \text{RAB}_0 & \text{REB} & M \text{ S}_1 \text{ S}_0 & \text{S}_2 \text{ S}_1 \text{ S}_0 & \text{OE} \\
\hline
s_0 & 000 & 0 & X X X X 0 & X X X X 0 & X X X X 0 & \text{X X X X} \\
\hline
d_1 & 001 & 1 & 0 0 1 1 & X X X 0 & X X X 0 & \text{X X X X} \\
\hline
d_2 & 010 & 0 & 0 1 1 1 & X X X 0 & X X X 0 & 1 0 1 \\
\hline
d_3 & 011 & 0 & 0 1 0 1 & X X X 0 & X X X 0 & 1 1 1 \\
\hline
d_4 & 100 & 0 & 1 0 0 1 & 0 0 1 1 & 0 1 0 1 & 0 0 1 \\
\hline
d_5 & 101 & 0 & 0 1 1 1 & 0 1 1 1 & 1 0 0 1 & 1 0 1 \\
\hline
d_6 & 110 & 0 & 0 0 1 1 & 0 0 1 1 & X X X 0 & 1 0 1 \\
\hline
ds_7 & 111 & 0 & X X X 0 & 0 1 1 1 & X X X 0 & 1 0 1 \\
\hline
\end{array}
\]

Output logic equations

\[
\begin{align*}
\text{IE} &= Q_2' Q_1' Q_0 \\
\text{WA}_2 &= Q_1' Q_0' \\
\text{WA}_1 &= Q_2 Q_0 + Q_2 Q_1' \\
\text{WA}_0 &= Q_1' Q_0 + Q_1 Q_0' \\
\text{WE} &= Q_2 Q_1' + Q_2' Q_0 + Q_1 Q_0 \\
\text{RAA}_2 &= 0 \\
\text{RAA}_1 &= Q_0 \\
\text{RAA}_0 &= 1 \\
\text{REA} &= Q_1 \\
\text{RAB}_2 &= Q_0 \\
\text{RAB}_1 &= Q_0' \\
\text{RAB}_0 &= 0 \\
\text{REB} &= Q_2 Q_1 \\
\text{M} &= Q_1 + Q_0 \\
\text{S}_1 &= Q_2' Q_0 \\
\text{S}_0 &= 1 \\
\text{S}_2 &= Q_1 Q_0 \\
\text{OE} &= Q_2 Q_1 Q_0
\end{align*}
\]
One's-counter schematic

Start

Q2 Q2' Q1 Q1' Q0 Q0

Clk

Done

Next-state logic

Data = 0

Data ≠ 0

Control unit

Start

Clk

Done

Output logic

Data = 0

Data ≠ 0

Output

Import

Result Bus

IE

WA2

WA1

WAb

WE

RAA2

RAA1

RAAb

REA

RAB2

RAB1

RABB

RBB

M

S1

S0

S1

S0

OE

Output
Parallel datapath

- Datapath allows 2 operations in each clock cycle.
- Choice of operations performed simultaneously is limited.
An example of a custom datapath

- Uses latches as temporary storage on the input and output of functional units.
- Latches allow shorter clock cycles and more concurrency.
Control-unit implementation styles

Control unit model
- Control inputs
- Status signals
- Datapath control signals
- Output logic
- Next-state logic
- State register

Control unit with state-register and decoder
- Control inputs
- Status signals
- Datapath control signals
- Output logic
- Next-state logic
- State register

Control unit with counter
- Control inputs
- Status signals
- Datapath control signals
- Output logic
- Next-state logic
- State register

Control unit with state-register and push-down stack
- Control inputs
- Status signals
- Datapath control signals
- Output logic
- Next-state logic
- State register
- Push-down stack
- External branch
- Incrementer
Control-unit implementation styles

Control unit with state-register, ROM and push-down stack (microprogrammed control unit)
Chapter Summary

- We introduced sequential components
  - Simple (registers, counters)
  - Storage (register-files, memories)
  - Complex (stacks, queues, datapaths and control units)

- Datapath and control units are used to implement standard processors and application specific integrated circuits (One’s count example)