Bit-Wise Operators
Bit-Wise Operators

If A and B are bytes, what does this code mean?

\[ C = A \& B; \]
Bit-Wise Operators

If A and B are bytes, what does this code mean?

\[ C = A \& B; \]

The corresponding bits of A and B are ANDed together.
Bit-Wise AND

\[
\begin{align*}
01011110 & \quad A \\
10011011 & \quad B \\
\hline \\
? & \quad C = A \& B
\end{align*}
\]
Bit-Wise AND

0 1 0 1 1 1 1 0

1 0 0 1 1 0 1 1

C = A & B
Bit-Wise AND

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

\[C = A \& B\]
Bit-Wise AND

0 1 0 1 1 1 1 0  A
1 0 0 1 1 0 1 1  B

C = A & B
Bit-Wise AND

\[
\begin{align*}
A &= 01011110 \\
B &= 10011011 \\
C &= A \& B = 00011010
\end{align*}
\]
Logical AND

\[ 0 \ 1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 0 \quad A \]
\[ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \quad B \]

\[ \text{C} = A \&\& B \]
Logical AND

0 1 0 1 1 1 1 0  \text{A} \text{true}

1 0 0 1 1 0 1 1  \text{B}

??? \quad \text{C} = \text{A} \& \& \text{B}
Logical AND

\[ C = A \land B \]

\[ \begin{array}{cccccccc}
0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
\end{array} \]

\[ A = \text{true} \]

\[ B = \text{true} \]

\[ C = \text{true} \]
Logical AND

\[ A = 01011110 \quad \text{true} \]
\[ B = 10011011 \quad \text{true} \]
\[ C = A \&\& B \quad \text{true} \]
Logical AND

\[
\begin{array}{cccccccc}
0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{array}
\]

\[
C = A \&\& B
\]

NOTE: we are assuming an 8-bit value
Representing Logical Values

Most of the time, we represent logical values using a multi-bit value. (e.g., using 8 or 16 bits). The rules are:

- A value of zero is interpreted as \textit{false}
- A non-zero value is interpreted as \textit{true}
Representing Logical Values

A logical operator will give a result of **true** or **false**:

- **false** is represented with a value of zero (0)
- **true** is represented with a value of one (1)
### Other Operators

<table>
<thead>
<tr>
<th>LOGICAL</th>
<th>Bit-Wise</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OR:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>NOT:</strong></td>
<td>!</td>
</tr>
<tr>
<td><strong>XOR:</strong></td>
<td>^</td>
</tr>
<tr>
<td><strong>Shift left:</strong></td>
<td>&lt;&lt;</td>
</tr>
<tr>
<td><strong>Shift right:</strong></td>
<td>&gt;&gt;</td>
</tr>
</tbody>
</table>

When coding: keep this distinction straight
Putting the Bit-Wise Operators to Work: Bit Manipulation

Assume a variable A is declared as such:

```c
uint8_t A;
```

What is the code that allows us to set bit 2 of A to 1? (we start counting bits from 0)
Bit Manipulation

What is the code that allows us to set bit 2 of A to 1? (we start counting bits from 0)

\[ A = A | 4; \]
Bit Manipulation

What is the code that allows us to set bit 2 of A to 0?
Bit Manipulation

What is the code that allows us to set bit 2 of A to 0?

\[ A = A \& 0xFB; \]

or

\[ A = A \& \sim 4; \]
Bit Shifting

```c
uint8_t A = 0x5A;
uint8_t B = A << 2;
uint8_t C = A >> 5;
```

What are the values of B and C?
What mathematical operations have we performed?
Example

Suppose a sensor is connected to pins 4 and 5 of port E:
• Fill in the following code so that variable “state” will have one of the following values: 0,1,2,3

```c
#include <type.h>

uint8_t state;

:\
state = ????
```
Suppose a sensor is connected to pins 4 and 5 of port E:
• Fill in the following code so that variable “state” will have one of the following values: 0, 1, 2, 3

```c
uint8_t state;
:
state = (GPIOE_PDIR & 0x30) >> 4;
```
Example (with only 8 bits)

GPIOE_PDIR: E7 E6 E5 E4 E3 E2 E1 E0
GPIOE_PDIR&0x30:
Example (cont)

GPIOE_PDIR : E7 E6 E5 E4 E3 E2 E1 E0
GPIOE_PDIR&0x30: 0 0 E5 E4 0 0 0 0
() >> 4:
Example (cont)

GPIOE_PDIR : E7 E6 E5 E4 E3 E2 E1 E0
GPIOE_PDIR & 0x30: 0 0 E5 E4 0 0 0 0
() >> 4: 0 0 0 0 0 0 0 0 0 E5 E4
... Back to Digital I/O