CS4HS @ MU
Logic and Math in Computers

• How do we get from silicon crystals to computers?
Logic and Math in Computers

• How do we get from silicon crystals to computers?
• Silicon crystals $\rightarrow$ switches called “transistors”
Logic and Math in Computers

• How do we get from silicon crystals to computers?
• Silicon crystals → switches called “transistors”
• Transistors → Boolean logic blocks
Logic and Math in Computers

- How do we get from silicon crystals to computers?
- Silicon crystals $\rightarrow$ switches called “transistors”
- Transistors $\rightarrow$ Boolean logic blocks
- Modern computers are made up of millions of Boolean logic blocks.
The Logic of Simple Switches
A
off

B
off

Light
off
A  B  Light
off  off  off
off  on  on
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>on</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>on</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>Light</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td>off</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>on</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>on</td>
</tr>
<tr>
<td>on</td>
<td>on</td>
<td>on</td>
</tr>
</tbody>
</table>
Light is on if A is on OR B is on

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>on</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>on</td>
</tr>
<tr>
<td>on</td>
<td>on</td>
<td>on</td>
</tr>
</tbody>
</table>
Another Logic Block
Another Logic Block

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>Light</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>off</td>
<td>off</td>
<td>off</td>
</tr>
</tbody>
</table>
### Another Logic Block

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>off</td>
</tr>
</tbody>
</table>

- Switch A is in the off position.
- Switch B is in the on position.
- The light remains off.

This diagram illustrates a simple logic block where the light is only on if both switches are in the on position.
Another Logic Block

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>off</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>off</td>
</tr>
</tbody>
</table>
Another Logic Block

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>off</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>on</td>
<td>on</td>
<td>on</td>
</tr>
</tbody>
</table>
**Light on if A is on AND B is on**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>off</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>on</td>
<td>on</td>
<td>on</td>
</tr>
</tbody>
</table>

The diagram illustrates a circuit where the light is on if both switches A and B are on.
The Inverter Block

<table>
<thead>
<tr>
<th>A</th>
<th>Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>on</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
</tr>
</tbody>
</table>

NOT
Combining Logic Blocks

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
</tr>
<tr>
<td>on</td>
<td>on</td>
</tr>
</tbody>
</table>
Combinational Logic

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F output</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>off</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>on</td>
<td>on</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Combinational Logic

A \quad \text{NOT} \quad \rightarrow \quad C \quad \text{AND} \quad \rightarrow \quad E \quad \text{NOT} \quad \rightarrow \quad F

B \quad \text{NOT} \quad \rightarrow \quad D

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>\text{C NOT(A)}</th>
<th>D</th>
<th>E</th>
<th>F output</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td></td>
<td></td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td></td>
<td></td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td></td>
<td></td>
</tr>
<tr>
<td>on</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Combinational Logic

A
\[ \text{NOT} \]
B
\[ \text{NOT} \]
C
D
\[ \text{AND} \]
E
\[ \text{NOT} \]
F

\begin{array}{|c|c|c|c|c|}
\hline
A & B & \text{NOT}(A) & \text{NOT}(B) & \text{F output} \\
\hline
\text{off} & \text{off} & \text{on} & \text{on} & \\
\text{off} & \text{on} & \text{on} & \text{off} & \\
\text{on} & \text{off} & \text{off} & \text{on} & \\
\text{on} & \text{on} & \text{off} & \text{off} & \\
\hline
\end{array}
Combinational Logic

The diagram shows a combinational logic circuit with inputs A, B, and outputs C, D, E, and F. The logic gates and connections are as follows:

- A and B are fed into NOT gates.
- The outputs of the NOT gates are connected to an AND gate.
- The output of the AND gate is fed into another NOT gate.

The truth table for the circuit is as follows:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C (NOT(A))</th>
<th>D (NOT(B))</th>
<th>E (AND(C,D))</th>
<th>F output</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>on</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>on</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>off</td>
</tr>
</tbody>
</table>
Combinational Logic

\[ \text{NOT}(A) \land \text{NOT}(B) \land \text{AND}(C,D) \land \text{NOT}(E) \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F output NOT(E)</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>off</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>on</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>on</td>
</tr>
<tr>
<td>on</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>on</td>
</tr>
</tbody>
</table>
The Logic Gate Game

A
\[ \text{NOT} \]
B
\[ \text{NOT} \]
\[ \text{AND} \]
F
\[ \text{NOT} \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>F output</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>on</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>on</td>
</tr>
<tr>
<td>on</td>
<td>on</td>
<td>on</td>
</tr>
</tbody>
</table>
De Morgan's Law

• OR(A,B) == NOT(AND(NOT(A), NOT(B))))
De Morgan's Law

- OR(A, B) == NOT(AND(NOT(A), NOT(B)))
- NOT(OR(A, B)) == AND(NOT(A), NOT(B))
- NOT(OR(NOT(A), B)) == AND(A, NOT(B))
- NOT(OR(NOT(A), NOT(B))) == AND(A, B)

These identities have many applications and parallels in logic, mathematics, and computer science.
One More Logic Block Puzzle...

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th></th>
<th></th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td></td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td></td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td></td>
</tr>
</tbody>
</table>
One More Logic Block Puzzle...

A

B

A

B

C

NOT(A)

off off on
off on on
on off off
on on off

NOT

AND

OR

C

A

B

C

AND

NOT

A

B

C

NOT(A)

off off on
off on on
on off off
on on off

NOT

AND

OR

C
One More Logic Block Puzzle...

A  B  C  NOT(A)  D  NOT(B)  
off off on  on  on  
off on  on  off  
on off  off  on  
on on  off  off
One More Logic Block Puzzle...

A
\[ \text{NOT} \]
B
\[ \text{AND} \]
C
\[ \text{NOT} \]
D
\[ \text{AND} \]
E
\[ \text{OR} \]

\begin{array}{c|c|c|c}
A & B & C & D \\
\hline
\text{off} & \text{off} & \text{on} & \text{on} \Rightarrow \text{off} \\
\text{off} & \text{on} & \text{on} & \text{off} \Rightarrow \text{off} \\
\text{on} & \text{off} & \text{off} & \text{on} \Rightarrow \text{on} \\
\text{on} & \text{on} & \text{off} & \text{off} \Rightarrow \text{off} \\
\end{array}
One More Logic Block Puzzle...

A | B | C | D | E | F
---|---|---|---|---|---
off | off | on | on | off | off
off | on | on | off | off | on
on | off | off | on | on | off
on | on | off | off | off | off
One More Logic Block Puzzle...

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>on</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>on</td>
</tr>
<tr>
<td>on</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>off</td>
</tr>
</tbody>
</table>
One More Logic Block Puzzle...

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>on</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>on</td>
</tr>
<tr>
<td>on</td>
<td>on</td>
<td>off</td>
</tr>
</tbody>
</table>
Logic Blocks – Collect Them All

• There are 16 \( (2^4) \) distinct Boolean logic functions (Logic Blocks) over two input variables.
Logical Completeness

• Any Boolean logic function can be built entirely out of AND and NOT blocks,
Logical Completeness

• Any Boolean logic function can be built entirely out of AND and NOT blocks,
• or out of OR and NOT blocks,
• or out of NAND blocks,
• or out of NOR blocks, etc., etc.
Logical Completeness

• Any Boolean logic function can be built entirely out of AND and NOT blocks,
• or out of OR and NOT blocks,
• or out of NAND blocks,
• or out of NOR blocks, etc., etc.

• SO, what does the Logic Block Game have to do with the fundamental ideas linking mathematics and computer science?
Simple Addition of Binary Digits

0 + 0 = 0
Simple Addition of Binary Digits

\[ 0 + 0 = 0 \]
\[ 0 + 1 = 1 \]
Simple Addition of Binary Digits

\[
\begin{align*}
0 + 0 &= 0 \\
0 + 1 &= 1 \\
1 + 0 &= 1
\end{align*}
\]
Simple Addition of Binary Digits

\[
\begin{align*}
0 + 0 &= 0 \\
0 + 1 &= 1 \\
1 + 0 &= 1 \\
1 + 1 &= 10
\end{align*}
\]
Simple Addition of Binary Digits

\[
\begin{align*}
0 + 0 &= 00 \\
0 + 1 &= 01 \\
1 + 0 &= 01 \\
1 + 1 &= 10
\end{align*}
\]
## Simple Addition of Binary Digits

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</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>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>
\[ C = \text{AND}(A, B) \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</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>
\[ D = \text{XOR}(A, B) \]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</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>0</td>
</tr>
</tbody>
</table>
Two-Bit Logic Block Computer

\[
\begin{array}{c|c|c|c}
A & B & C & D \\
\hline
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 \\
\end{array}
\]

C = AND(A,B)
D = XOR(A,B)
Two-Bit Logic Block Computer

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>S</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>

\( C = \text{AND}(A,B) \)
\( S = \text{XOR}(A,B) \)

S is for “Sum”
C is for “Carry”
Bigger Blocks – The Half Adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>S</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>

C = AND(A,B)
S = XOR(A,B)

S is for "Sum"
C is for "Carry"
More Than Two Bits

A_3
C S
A_2
C S
A_1
C S
A_0
C S

A B
A B
A B
A B

A B
A B
A B
A B

C out
S_3
S_2
S_1
S_0

1
More Than Two Bits

A_3 \rightarrow A \rightarrow A_2 \rightarrow A_1 \rightarrow A_0 \rightarrow 1

C_{\text{out}} \rightarrow C \rightarrow C_2 \rightarrow C_1 \rightarrow C_0 \rightarrow 0

S_3 \rightarrow S \rightarrow S_2 \rightarrow S_1 \rightarrow S_0 \rightarrow 0

A_3 A_2 A_1 A_0 | C_{\text{out}} S_3 S_2 S_1 S_0

0 0 0 0 | 0
More Than Two Bits

A3

A2

A1

A0

C

S

S

S

S

A

A

A

A

C

S

S

S

S

0 0 0 0 0

1

1 1 1 1
More Than Two Bits

A diagram of a circuit with inputs A₃, A₂, A₁, A₀ and outputs Cₜₐₓₜ, S₃, S₂, S₁, S₀. The values shown are:

- A₃ = 0, A₂ = 0, A₁ = 0, A₀ = 1
- Cₜₐₓₜ = 0, S₃ = 0, S₂ = 0, S₁ = 1, S₀ = 1
More Than Two Bits
More Than Two Bits

A diagram showing a sequence of logic gates with inputs and outputs labeled with bits. The diagram includes labels for $A_3$, $A_2$, $A_1$, $A_0$, $C_{out}$, $S_3$, $S_2$, $S_1$, $S_0$, with the expected outputs $0$, $0$, $0$, $0$ and $0$, $1$.
More Than Two Bits
More Than Two Bits

A diagram illustrating a circuit with inputs $A_3, A_2, A_1, A_0$ and outputs $C_{out}, S_3, S_2, S_1, S_0$. The values for $A_3, A_2, A_1, A_0$ are 0, 0, 0, 0, and the values for $C_{out}, S_3, S_2, S_1, S_0$ are 0, 0, 0, 1.
More Than Two Bits

A diagram showing a circuit with inputs $A_3, A_2, A_1, A_0$ and outputs $C_{out}, S_3, S_2, S_1, S_0$. The circuit processes the inputs through multiple stages, with the final output being $1$. The table below the diagram shows the input states and the corresponding output states.

<table>
<thead>
<tr>
<th>$A_3$</th>
<th>$A_2$</th>
<th>$A_1$</th>
<th>$A_0$</th>
<th>$C_{out}$</th>
<th>$S_3$</th>
<th>$S_2$</th>
<th>$S_1$</th>
<th>$S_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
More Than Two Bits
More Than Two Bits

A diagram showing a circuit with inputs A_3, A_2, A_1, A_0 and outputs C_out, S_3, S_2, S_1, S_0. The inputs are connected to the circuit as follows:

- A_3 connects to C
- A_2 connects to B
- A_1 connects to C
- A_0 connects to B

The outputs are:

- C_out
- S_3
- S_2
- S_1
- S_0

The input values are:

<table>
<thead>
<tr>
<th>A_3</th>
<th>A_2</th>
<th>A_1</th>
<th>A_0</th>
<th>C_out</th>
<th>S_3</th>
<th>S_2</th>
<th>S_1</th>
<th>S_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

1
More Than Two Bits

A 3
    |      |
C --+      +--> B
    |      |
S     --+--

A 2
    |      |
C --+      +--> B
    |      |
S     --+--

A 1
    |      |
C --+      +--> B
    |      |
S     --+--

A 0
    |      |
C --+      +--> B
    |      |
S     --+--

C out

<table>
<thead>
<tr>
<th>A 3</th>
<th>A 2</th>
<th>A 1</th>
<th>A 0</th>
<th>C out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S 3</th>
<th>S 2</th>
<th>S 1</th>
<th>S 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

| 1   | 0   | 0   | 0   |

## More Than Two Bits

<table>
<thead>
<tr>
<th>(A_3)</th>
<th>(A_2)</th>
<th>(A_1)</th>
<th>(A_0)</th>
<th>(C_{\text{out}})</th>
<th>(S_3)</th>
<th>(S_2)</th>
<th>(S_1)</th>
<th>(S_0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</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>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

...
All That Just To Do A+1?

- Any N-bit number can be incremented with N half-adder logic blocks.
- We can build a similar combination of logic blocks to decrement any N-bit number.
- We can build an N-bit zero detector with AND and NOT blocks.

- Where does that leave us?
**Addition**

```plaintext
define add(x, y) =
    if (isZero? x)
        then answer = y
    else answer = add(dec(x),
                      inc(y))
```
define mult(x, y) =
    if (isZero? y)
        then answer = 0
    else answer =
        add(x, mult(x, dec(y)))
Multiplication

define mult(x, y) =
  if (isZero? y)
    then answer = 0
  else answer =
    add(x, mult(x, dec(y)))

• Yikes!

• Is this really what my computer / calculator is doing?
### Addition – the Less Scenic Route

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>C_{in}</th>
<th>C_{out}</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
The “Full Adder” carries in and out

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>C_{in}</th>
<th>C_{out}</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

![Diagram of a Full Adder circuit]
The Ripple Adder

\[ \begin{array}{c c c c}
X_3 & Y_3 & X_2 & Y_2 \\
C_{\text{out}} & C_{\text{in}} & C_{\text{out}} & C_{\text{in}} \\
S_3 & S_2 & S_1 & S_0 \\
\end{array} \]
The Ripple Adder

\[ C_{\text{out}} S_{3} S_{2} S_{1} S_{0} = X_{3} X_{2} X_{1} X_{0} + Y_{3} Y_{2} Y_{1} Y_{0} + C_{\text{in}} \]
Before we dive into logic block multiplication, let's review “normal”, base-10 multiplication.

Think algorithmically
How do we teach long multiplication to grade school children?
How is base-10 long multiplication the same as base-2 long multiplication?
How is it different?
\[ \begin{array}{c}
A_{N} A_{N-1} \ldots A_{2} A_{1} A_{0} \\
\times B_{m} B_{m-1} \ldots B_{2} B_{1} B_{0}
\end{array} \]
\[ A_N A_{N-1} \ldots A_2 A_1 A_0 \times B_m B_{m-1} \ldots B_2 B_1 B_0 \]

\[ B_0 x A_1 x 10 + B_0 x A_0 \]
\[ B_0 \times A_2 \times 100 + B_0 \times A_1 \times 10 + B_0 \times A_0 \]
\[
\cdots B_0 \times A_2 \times 100 + B_0 \times A_1 \times 10 + B_0 \times A_0
\]
... B_0 \times A_2 \times 100 + B_0 \times A_1 \times 10 + B_0 \times A_0 \\
+ ... B_1 \times A_1 \times 100 + B_1 \times A_0 \times 10
\[ A_0 A_1 \ldots A_{N-1} A_N \times B_0 B_1 \ldots B_{m-1} B_m \]

\[
\ldots B_0 A_2 x 100 + B_0 A_1 x 10 + B_0 A_0 + \ldots B_1 A_2 x 100 + B_1 A_1 x 10 + B_1 A_0 + \ldots B_2 A_1 x 1000 + B_2 A_0 x 1000
\]
for i = 0..M
  for j = 0..N
    term = A_j x B_i x 10^j
    partProd = partProd + term
    product = product + partProd x 10^i
Peasant's Multiplication

37  x  5
Peasant's Multiplication

\[
\begin{array}{cc}
37 & 5 \\
18 & 10 \\
\end{array}
\]
Peasant's Multiplication

37
18
9

5
10
20
# Peasant's Multiplication

<table>
<thead>
<tr>
<th>37</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
</tr>
</tbody>
</table>
### Peasant's Multiplication

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>80</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Peasant's Multiplication

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>5</td>
</tr>
<tr>
<td>18</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>1</td>
<td>160</td>
</tr>
</tbody>
</table>
Peasant's Multiplication

37       5
18       10
9        20
4        40
2        80
1       160
Peasant's Multiplication

<table>
<thead>
<tr>
<th>37</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>1</td>
<td>+ 160</td>
</tr>
</tbody>
</table>
**Peasant's Multiplication**

<table>
<thead>
<tr>
<th></th>
<th>37</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>18</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>---</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>1</td>
<td>+</td>
<td>160</td>
</tr>
<tr>
<td>---</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td></td>
<td>185</td>
<td></td>
</tr>
</tbody>
</table>
Peasant's Multiplication

37       5
18       10
 9       20
 4       40
 2       80
 1   + 160

185

Why does this work?
Peasant's Multiplication

\[
\begin{align*}
37 & \quad 5 \\
18 & \quad 10 \\
9 & \quad 20 \\
4 & \quad 40 \\
2 & \quad 80 \\
1 & \quad 160 \\
\hline
1 & \quad + \quad 160 \\
\hline
185
\end{align*}
\]
Peasant's Multiplication

\[
\begin{align*}
37 & \quad 5 \quad 5 \times 2^0 \times 1 \\
18 & \quad 10 \quad + \ 5 \times 2^1 \times 0 \\
9 & \quad 20 \quad + \ 5 \times 2^2 \times 1 \\
4 & \quad 40 \quad + \ 5 \times 2^3 \times 0 \\
2 & \quad 80 \quad + \ 5 \times 2^4 \times 0 \\
1 & \quad 160 \quad + \ 5 \times 2^5 \times 1 \\
\hline
& \quad 185
\end{align*}
\]
Peasant's Multiplication

\[
\begin{array}{ccc}
37 & 5 & 5 \times 2^0 \times 1 \\
18 & 10 & + 5 \times 2^1 \times 0 \\
9 & 20 & + 5 \times 2^2 \times 1 \\
4 & 40 & + 5 \times 2^3 \times 0 \\
2 & 80 & + 5 \times 2^4 \times 0 \\
1 & 160 & + 5 \times 2^5 \times 1 \\
\hline
185
\end{array}
\]

\[37_{10} = 100101_2\]
Binary Multiplication

\[ 37 \times 5 \]
\[ 100101 \times 000101 \]
Binary Multiplication

37
x 5

100101
x 000101

100101
Binary Multiplication

37
x 5
---
100101
x 000101
---
100101
100101
100101
000000
Binary Multiplication

37\[\times\]5

100101

x 000101

100101

000000

100101
Binary Multiplication

37  
\times 5

100101
\times 000101
-------------------
100101
000000
100101
000000
000000
000000
Binary Multiplication

\[
\begin{array}{c}
37 \\
\times 5 \\
\end{array}
\quad \begin{array}{c}
100101 \\
\times 000101 \\
\quad 100101 \\
\quad 000000 \\
\quad 100101 \\
\quad 000000 \\
\quad 000000 \\
\quad + 000000 \\
\hline
00010111001
\end{array}
\]
Binary Multiplication

\[
\begin{array}{c}
37 \\
\times 5 \\
\hline
185
\end{array}
\]

\[
\begin{array}{c}
100101 \\
\times 000101 \\
\hline
100101 \\
000000 \\
100101 \\
000000 \\
000000 \\
\hline
00010111001 = 185_{10}
\end{array}
\]
How do we multiply two bits with Logic Blocks?
### Back to Logic Blocks

How do we multiply two bits with Logic Blocks?

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X*Y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How do we multiply two bits with Logic Blocks?

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X*Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
How do we multiply two bits with Logic Blocks?

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X*Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
How do we multiply two bits with Logic Blocks?

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X * Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
How do we multiply two bits with Logic Blocks?

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X*Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
How do we multiply two bits with Logic Blocks?

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X*Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
How do we multiply two bits with Logic Blocks?

\[
\begin{array}{cccc}
X & Y & X \cdot Y \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

Multiplying two bits never carries into next column!
A group of 'N' AND Blocks produces the partial product from a one-bit multiplier.
Logic Block Multiplication

- With a Bit Multiplier Logic Block and a Full Adder Logic Block for each bit of multiplier, we can construct a Logic Block for performing long multiplication of binary numbers.
Logic Block Multiplication

- With a Bit Multiplier Logic Block and a Full Adder Logic Block for each bit of multiplier, we can construct a Logic Block for performing long multiplication of binary numbers.

- The Logic Block Game has very simple rules.
- Only a small number of basic blocks types required.
- Surprisingly complex logic can be constructed from very little.
Wisconsin State Standards

Standard A – Mathematical Processes

• A.12.1: Use Reason and Logic to
  - Evaluate Information
  - Perceive patterns
  - Identify relationships
  - Formulate questions, pose problems, and make and test conjectures
  - Pursue ideas that lead to further understandings and deeper insights
Wisconsin State Standards

• A.12.3: Analyze non-routine problems and arrive at solutions by various means, including models and simulations...

• A.12.5: Organize work and present mathematical procedures and results clearly, systematically, succinctly, and correctly.
Wisconsin State Standards

Standard B – Number Operations & Relationships

• B.12.1 Use complex counting procedures such as union and intersection of sets and arrangements (permutations, combinations) to solve problems.

• B.12.4 In problem-solving situations involving the application of different number systems select and use appropriate:
  • computational procedures
  • properties (e.g., commutativity, associativity...)
  • modes of representation
Wisconsin State Standards

Standard F – Algebraic Relationships

• F.12.2 Use mathematical functions in a variety of ways, including
  • recognizing that a variety of mathematical and real-world phenomena can be modeled by the same type of function
  • translating different forms of representing them
Standard F – Algebraic Relationships

- F.12.4 Model and solve a variety of mathematical and real-world problems by using algebraic expressions, equations, and inequalities

Overlap with PLTW – Digital Electronics