Logic Circuit Synthesis

Synthesizing Switching Functions

A "logic circuit" is composed of switching functions

Think of this as an application of functional programming.

outputs

outputs

"not" bubbles

"and" shapes

"or" shape

inputs

f(u, v, w, x) = u v' w' x' + u' v w' x + u' v' w x'

"and" gates

x y

input

output

x y

v w x y

"or" gates

x y

input

output

v w x y

"not" Gates

"inverter"

x

¬x

implied "nots" (bubbles)

y

¬(x ⋁ y)

x y

¬x ¬y

"Bill" Gates
Gates are Symbolic

- Gates are not just electronic; they can be
  - Mechanical
  - Hydraulic ("fluid logic")
  - Biological
  - Sub-atomic
  - Quantum-mechanical

Mechanical Gates

- "or" gate

Mechanical Gates

- "and" gate

Mechanical Gates

- "inverter"

Fluid Gates

- "or" gate

Biological Gates

- "Nanotechnology"
DNA-based Gates

Synthesizing Switching Functions

A "logic circuit" is composed of switching functions.

A logic circuit is composed of switching functions.

Think of this as an application of functional programming.

Ways to Specify Switching Functions

- Logic circuit
- Functional expression
  - SOP form
    - Minterm expansion
- Truth table

Note the Connection

f(u, v, w, x) = u v' w' x' + u' v w' x + u' v' w x'

Definition of "minterm"

- In the context of an n-variable switching function, a minterm is a function that is a conjunction ("and") of each variable or its complement (but not both).

  Minterm Examples (4 variables: u, v, w, x):
  - uvw'x
  - u'vw'x

  Non-Minterm Examples (4 variables):
  - uv
  - x
  - uw
  - u'uw

The "1" rows of the truth table correspond exactly to the minterms.

Note the Connection

Read off each primed variable as 0, each unprimed as 1.
Shorthands

\[ (u, v, w, x) = u'v'w'x' + u'vwx' + uv'w'x' \]

Show only the "1" rows (be careful)

Represent whole table by a set of "minterm numbers":

\[ \{2, 5, 8\} \]

Number of Switching Functions

- How many switching functions of \( n \) variables are there?

- This will be on the final, so might as well memorize it now.

These are Equal

- The number of switching functions of \( n \) variables.

- The number of ways to assign 0 or 1 to the \( 2^n \) combinations of \( n \) variables.

- The number of subsets of \( \{0, 1, 2, ..., 2^n - 1\} \)

- \( 2^{2^n} \)

The 16 switching functions of 2 variables

<table>
<thead>
<tr>
<th>args</th>
<th>Function number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1</td>
</tr>
<tr>
<td>1 1 0</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1</td>
</tr>
</tbody>
</table>

Implication Lattice of 2-variable functions

Number of Switching Functions

- \( 2^{2^n} \)
- \( n = 1: 2^1 = 4 \)
- \( n = 2: 2^2 = 16 \)
- \( n = 3: 2^3 = 256 \)
- \( n = 4: 2^4 = 65,536 \)
- \( n = 5: 2^5 = 4,294,967,296 \)
- \( n = 6: 2^6 = 18,446,744,073,709,551,616 \)

- Also remember that \( 2^{10} \approx 1024 \) approx. 1000
- \( 2^{20} \approx 1,000,000 \) etc.

Each level squares the previous, since

\[ 2^{2^{n+1}} = 2^{2^n} \cdot 2^{2^n} \cdot (2^n)^2 \]
Logic Synthesis: Abstraction to Implementation

- **From:** Verbal problem description
- **To:** Implementation as a network of basic switching functions

Logic Synthesis: Stages

1. **Provide** verbal problem description.
2. **Tabulate** description as function on finite sets.
3. **Encode** finite sets into bits.
4. **Transcribe** the encoded tables.
5. **Split** into individual switching functions.
6. **Realize** as a network of basic gates.

Example

- **Provide verbal description:** Implement a "mod 3 adder using logic gates"
- **A definition of mod3 addition:**
  \[ f(a, b) = (a+b) \mod 3; \]
  \[ \text{where } a, b \in \{0, 1, 2\} \]

Tabulate definition of function

<table>
<thead>
<tr>
<th>(f(x, y)\mod 3)</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Encode sets into bits

- **Set to be encoded:** \{0, 1, 2\}
- **Chosen encoding (among many):**
  - 0 → 00
  - 1 → 01
  - 2 → 10

Transcribe the Function to the Encoded Values

<table>
<thead>
<tr>
<th>Function</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2</td>
<td>0 1 2</td>
</tr>
<tr>
<td>0 0 1</td>
<td>1 0 1</td>
</tr>
<tr>
<td>1 1 0</td>
<td>2 1 0</td>
</tr>
<tr>
<td>2 2 1</td>
<td>0 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Encoded Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>UV</td>
</tr>
<tr>
<td>00 01 10</td>
</tr>
<tr>
<td>01 01 00</td>
</tr>
<tr>
<td>10 00 01</td>
</tr>
</tbody>
</table>
Split the Encoded Function into individual switching functions

Encoded Function

<table>
<thead>
<tr>
<th>uv</th>
<th>00</th>
<th>01</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>xy</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Call this \( f_1 \).

Call this \( f_2 \).

As the unspecified values will never occur, we can give the function either 0 or 1.

For the time being, let's just make them all 0.

Now we can "read off" an expression for each function.

\[
\begin{align*}
&f_1(u, v, w, x) = u'v'w + u'vw' + uv'w' \\
&f_2(u, v, w, x) = u'v'w' + u'v + uvw'
\end{align*}
\]

The resulting switching functions generally will only be partially specified; some combinations don't occur.

<table>
<thead>
<tr>
<th>uv</th>
<th>00</th>
<th>01</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>xy</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Exercise: "read off" the expression for \( f_2 \).

Realize each function by gates

\[
\begin{align*}
&f_1(u, v, w, x) = u'v'w + u'vw' + uv'w' \\
&f_2(u, v, w, x) = u'v'w' + u'v + uvw'
\end{align*}
\]
Check by “Reverse Engineering”
(Try all combinations; one combination is shown)

Rex Program for Checking all Combinations

The Testing Code

```
test(addByCircuit(0, 0), add3(0, 0));
test(addByCircuit(0, 1), add3(0, 1));
test(addByCircuit(0, 2), add3(0, 2));
test(addByCircuit(1, 0), add3(1, 0));
test(addByCircuit(1, 1), add3(1, 1));
test(addByCircuit(1, 2), add3(1, 2));
test(addByCircuit(2, 0), add3(2, 0));
test(addByCircuit(2, 1), add3(2, 1));
test(addByCircuit(2, 2), add3(2, 2));
```