CAST is both a language and a toolkit originally designed for asynchronous design, but recently adapted for synchronous design.

Lecture Contents:

• What is the Production Rule?

• How to get the most out of the CAST language.

Basic idea: describe circuits digitally in a programming notation, build them with magic, check that you built what you intended.

CAST is used for conveniently describing the digital circuits.
The Production Rule

Disclaimer: This is the ‘synchronized’ definition of PRs.

\[ S \rightarrow E \]

\( S \): This is a boolean expression that, when true for sufficient time, causes event \( E \) to happen.

\( E \): The event is limited to pulling a single boolean value, representing a node in a circuit, to a power supply (boolean constant \texttt{true} or \texttt{false}).
The Two Senses

For CMOS, production rules come in two flavors, pull-ups:

\[ \neg s \rightarrow e↑ \]

And pull-downs:

\[ s \rightarrow e↓ \]

The pull-up rule is that all values in the guard must be tested in the negative sense. That does not mean the signals are inverted. The inverse holds for the pull-downs.

Examples of valid CMOS production rules:

\[ a \land b_\neg \rightarrow out↓ \]
\[ \neg a \land \neg b_\neg \rightarrow out↑ \]
\[ \neg (a \lor b_\neg) \rightarrow out↑ \]
Example

Consider the production-rule set:

\[
\begin{align*}
y & \rightarrow y_\downarrow \\
\neg y & \rightarrow y_\uparrow \\
\end{align*}
\]

\[
\begin{align*}
x \land y_\downarrow & \rightarrow z_\downarrow \\
\neg x \land \neg y_\downarrow & \rightarrow z_\uparrow \\
\end{align*}
\]

This would be implemented by the circuit:
The Pen is Mightier than the Keyboard

In order to express production rules easily on a computer, the following conventions are used:

\[ \land = \& \]
\[ \lor = | \]
\[ \neg = \sim \]
\[ \rightarrow = \rightarrow \]
\[ \uparrow = + \]
\[ \downarrow = - \]

Example:
\[ \neg a \land \neg b_\neg \rightarrow out\uparrow \]

Is typed as follows:
\[ \sim a \& \sim b_\neg \rightarrow out+ \]
The CAST Production Rule

CAST Production Rules operate on a basic circuit element called the node. The following illustrates a CAST specification of a NAND gate:

```plaintext
node a, b;
node nand;
prs{
    a & b -> nand-
    ~a | ~b -> nand+
}
```
CAST is not executed in the usual sense of the word, rather it is a specification for a circuit that can be executed. The = operator in cast does not assign a value, but specifies that the two operands shall be aliases of a single object. The NAND gate could have been written like this:

```plaintext
node a, a2, b, b2;
node nand, nand2;
a = a2;
b = b2;
nand = nand2;
prs{
    a & b -> nand-
    ~a2 | ~b2 -> nand2+
}
```
CAST allows you to define cells to organize and simplify a specification. The NAND example again:

```plaintext
define Nand2()(node a, b; node o)
{
    prs{
        a & b -> o-
        ~a | ~b -> o+
    }
}
```

- This does not instantiate a NAND gate, but rather specifies it.
Instantiation

We have already seen instantiation in the following line:

```c
node a;
```

This instantiates a basic type in CAST. Alternatively we can instantiate a user-defined type such as the Nand2, and ‘alias’ to its interface:

```c
node a, b;
node out;
Nand2 myGate( a, b, out );
```
Arrays

CAST allows both sparse and dense arrays. Sparse-array instantiation:

```plaintext
node x[5], y[5];
```

Dense-array instantiation:

```plaintext
node[5] x, y;
```

This allows you to create the array all at once, or piece by piece.

- Aliasing can be done either on an element of an array to another node, or on pieces of an array, or on entire arrays:

```plaintext
node z;
node[5] x, y, a, b;
x[0] = z;
x[2..3] = y[1..2];
a = b;
```
Parameterizing Cells

Parameters are values used by CAST when instantiating a circuit. They are not present in the final circuit. The repetition construct iterates over a range $R$ and substitutes for the variable $I$ in the statements $S$.

\[ < I:R:S > \]

Parameterization, along with the repetition construct can be used for such things as specifying an N-bit processor. The following example illustrates parameters and repetition.

```plaintext
define Connect(int N)
    (node[N] in, out)
    {
        < i:N: in[i] = out[i]; >
    }
node[10] a,b;
Connect(10) myConnect(a,b);
```

- In the above repetition, $N$ is shorthand for the range $0..N-1$
Misc Topics

• Multiple Files: You will want to use multiple files for a specification. To reference instantiations or definitions in another file, use the import command:

        import "mydir/myfile.cast";

• What qualifies as an identifier? Practically anything, and if you are unsure, just put quotes around it.

• This is just a brief introduction to the CAST language. There are complete manuals in lab and on the website.

• CAST is a set of homemade tools, and you are beta testers.