VHDL: Very High speed integrated circuit Description Language:

All VHDL files have two sections: architecture and entity

```
-- Massachusetts (Obsolete) Stoplight Example

library ieee;
use ieee.std_logic_1164.all;
entity check is
  port(r, y, g: in std_logic;
       ok: out std_logic);
end check;

architecture logical of check is
  signal t1, t2, t3: std_logic;
begin
  t1 <= r and (not g);
  t2 <= y and (not g);
  t3 <= (not r) and (not y) and g;
  ok <= t1 or t2 or t3;
end logical;
```
library clause describes the basic library to make reference to

use clause establishes definitions of many important items
for most situations, use these 'as is'

library ieee;
use ieee.std_logic_1164.all;

Other libraries will be used and you will have the opportunity to make libraries of your own.
The entity declaration can be quite complex and has a lot of information
I/O signals are referred to as **PORTs**. These signals have **Mode** and **Type**

The Mode of a signal can be **in**, **out**, **buffer** or **inout**
- **in** and **out** are straightforward
- **buffer** is like **out**, but is available within the architecture
- **inout** is a tri-state (bidirectional)

Note how vectors (multi-bit) signals are handled.

```vhd
ENTITY black_box IS PORT
  (clk, rst : IN  std_logic;
   d     : IN  std_logic_vector(7 DOWNTO 0);
   q     : OUT std_logic_vector(7 DOWNTO 0);
   co    : OUT std_logic);
END black_box;
```

![Diagram](image)
We can avoid using Mode BUFFER

library ieee;
use ieee.std_logic_1164.all;
entity foo is
       port (in1, in2: in std_logic;
             out1, out2: out std_logic);
end foo;

architecture no_buffer_mode of foo is
       signal inside: std_logic;
begin
       inside <= in1 AND in2;
       out1 <= inside;
       out2 <= inside OR (not in1);
       -- really wanted out2 <= out1 OR (not in1);
end no_buffer_mode;

Note the additional declaration of signal inside
the architecture section.
Note the names in the architecture section need
not be unique and are there for readability
Type of signals are defined in

```
LIBRARY ieee;
use ieee.std_logic_1164.all;
```

(VHDL is defined by IEEE Standard 1164)

std_logic types can take values:

- U  Uninitialized
- X  Unknown
- 0  Zero
- 1  One
- Z  Tristate (Must be upper case!)
- W  Weak unknown
- L  Weak Zero
- H  Weak One
-  - Don't care

Note that in most cases we don't really need to use all of these values
Extract of the report file (*.rpt)

**DESIGN EQUATIONS** (12:32:59)

\[
\begin{align*}
t_1 &= r * /g \\
t_2 &= y * /g \\
t_3 &= /r * /y * g \\
\text{ok} &= /r * /y * g \\
&+ r * /g \\
&+ y * /g
\end{align*}
\]
More from the report file: If YOU don't set pin numbers, the compiler will.

**C22V10**

<table>
<thead>
<tr>
<th>g</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>2</td>
</tr>
<tr>
<td>r</td>
<td>3</td>
</tr>
</tbody>
</table>

not used * | 4 |
not used * | 5 |
not used * | 6 |
not used * | 7 |
not used * | 8 |
not used * | 9 |
not used * | 10 |
not used * | 11 |
not used * | 12 |

| 24 | * not used |
| 23 | = t3       |
| 22 | = t1       |
| 21 | * not used |
| 20 | * not used |
| 19 | * not used |
| 18 | * not used |
| 17 | * not used |
| 16 | * not used |
| 15 | = t2       |
| 14 | = ok       |
| 13 | * not used |
Easy Way to Assign Pins:

Don't assign pins first.

Let galaxy pick them and wire to those pins.

Find out the pins from the report file

To put them in to avoid rewiring.

click on Files->Annotate

After a pop up, this produces and xxx.ctl file which then is used along with xxx.vhd.

OR you can use the pin_numbers attribute (next slide)

Be careful not to put a pin number in here which conflicts with a pin_avoid attribute in your xxx.vhd file.
Attributes provide information about VHDL constructs such as

Entities

Architectures

Types

Signals

Pin_numbers maps external signals to specific pins

Pin_avoid means to not use specific pins.

See the xxx.vhd files in /mit/6.111/cpld/sources/ for guidance in choosing pins and/or avoiding pins.
Example Using Pin_avoid Attribute:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity fulladd is
  port (ina, inb, inc : in std_logic;
       sumout, outc : out std_logic);
  ATTRIBUTE pin_avoid of fulladd :ENTITY is
    " 19 " &
    " 12 " ;
end fulladd;
```

Here is the contents of a control (.ctl) file:

```vhdl
Attribute PIN_NUMBERS of Reserved2 is "19" ;
Attribute PIN_NUMBERS of outc is "14" ;
Attribute PIN_NUMBERS of sumout is "13" ;
Attribute PIN_NUMBERS of Reserved1 is "12" ;
Attribute PIN_NUMBERS of ina is "3" ;
Attribute PIN_NUMBERS of inb is "2" ;
Attribute PIN_NUMBERS of inc is "1" ;
```
So here is one example of a VHDL implementation

The issue is an adder: we can make a 'full adder' from two 'half adders' and a little bit of logic. Here, to start is the half adder:

**Arithmetic Operation: one bit addition**

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>b</th>
<th>c</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>1</td>
<td>0</td>
</tr>
<tr>
<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>
</tr>
</tbody>
</table>

\[
b = x \oplus y
\]

\[
c = x \cdot y
\]
So here is the 'half adder' implemented in VHDL:

```vhdl
library ieee;
use ieee.std_logic_1164.all;

-- here is the entity
entity halfadd is
port (a, b : in std_logic;
     sum, c : out std_logic);
end halfadd;

architecture comp of halfadd is
begin
  -- a concurrent statement implementing the and gate
  c <= a and b;
  -- a concurrent statement implementing the xor gate
  sum <= a xor b;
end comp;

These statements are 'concurrent', which means they are executed at the same time and with no precedence.
```
Now how would you make a 'full' adder?

<table>
<thead>
<tr>
<th>cin</th>
<th>x</th>
<th>y</th>
<th>bout</th>
<th>cout</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>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<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>1</td>
</tr>
</tbody>
</table>

\[ \text{bout} = x \oplus y \oplus \text{cin} \]

\[ \text{cout} = x'y + x'y'\text{cin} + x'x\text{cin}' = x'y + x \oplus y'\text{cin} \]

**Arithmetic Operation: Full Adder: Include Carry In**

So a cascade of 2 half adders and an or gate does it.
Here is an implementation of the full adder using component instantiation through a port map:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity fulladd is
  port (ina, inb, inc : in std_logic;
        sumout, outc : out std_logic);
end fulladd;
architecture top of fulladd is
  component halfadd
    port (a, b : in std_logic;
          sum, c : out std_logic);
  end component;
signal s1, s2, s3 : std_logic;
begins
  -- a structural instantiation of two half adders
  h1: halfadd port map( a => ina, b => inb,
                        sum => s1, c => s3);
  h2: halfadd port map( a => s1, b => inc,
                        sum => sumout, c => s2);
  outc <= s2 or s3;
end top;
```
So here is how a compilation and simulation of this simple problem might go.

setup 6.111

Galaxy &
Now use the pulldown **files -> add**
At this point you add files: click on file in left window and then the arrow that shows up in the middle. Add all the files to be compiled. Then OK
Here is what the project screen looks like now, with the files added.

Next Step is to select a Device and select top file and Set Top
We pick the device: here a 22v10 will hold the required logic. We also select a package (not really important if we are only simulating) and we also pick a simulation scheme.
Here it is ready to compile: device and top design file are selected.
We will use’ Smart’ Compile: the program figures out what to do
This is what the compile screen looks like. If there are errors they will show up here. Note we have an error here at the very last step, which is setting up for NOVA. Not to worry: this was just a disagreement over displays. Note a lot of stuff scrolls by: see the scroll bar on the right.
Here is the opening screen for Nova, a simple simulator. Invoke from ‘tools’ on the project screen or from the command line. Use file->open
We must select a .jed (JEDEC) file for the simulation. In this case the correct file gets its name from the top design file and is `fullad.jed`
Here is the simulation. Use Edit to set up the inputs: here we just set each input to be driven as a ‘clock’ with different (X2) periods to cycle through all possible inputs. Then Simulate generates the output.