6.111 Lecture # 5

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;
```

Entity section describes input and output

```
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;
```

Architecture section describes what to do with those signals

`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 referrred to as PORTs. These signals have Mode and Type

The Mode of a signal can be in, out, buffer or inout

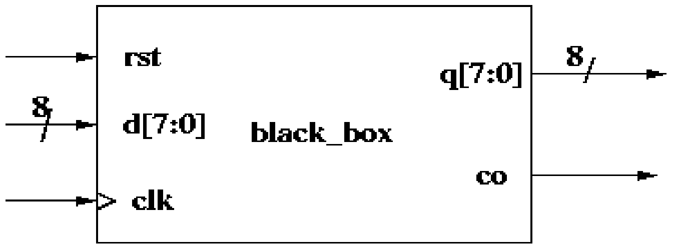
in and out are straightforward

buffer is like out, but is available within the architecture

inout is a tri-state (bidirectional)

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

```
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;
```



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)


    t1 =
          r * /g

    t2 =
          y * /g

    t3 =
          /r * /y * g

    ok =
          /r * /y * g
        + r * /g
        + y * /g
```

More from the report file: If YOU don't set pin numbers, the compiler will.

```
                        C22V10
             __________________________________________
       g =| 1|                                    |24|* not used
       y =| 2|                                    |23|= t3
       r =| 3|                                    |22|= t1
not used *| 4|                                    |21|* not used
not used *| 5|                                    |20|* not used
not used *| 6|                                    |19|* not used
not used *| 7|                                    |18|* not used
not used *| 8|                                    |17|* not used
not used *| 9|                                    |16|* not used
not used *|10|                                    |15|= t2
not used *|11|                                    |14|= ok
not used *|12|                                    |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 extrernal 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:

```
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:

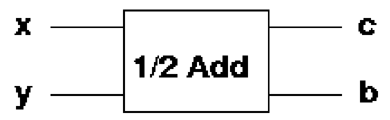
```
Attribute PIN_NUMBERS of Reserved2 is "19" ;
Attribute PIN_NUMBERS of outc is "14" ;
Attribute PIN_NUMBERS of sumout is "13" ;
Attribute PIN_NUMBERS of Reserved1 is "12" ;
Attribute PIN_NUMBERS of ina is "3" ;
Attribute PIN_NUMBERS of inb is "2" ;
Attribute PIN_NUMBERS of inc is "1" ;
```

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:

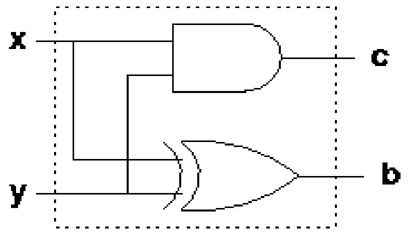
Arithmetic Operation: one bit addition

x	y	b	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



$b = x \oplus y$

$c = x \ * \ y$



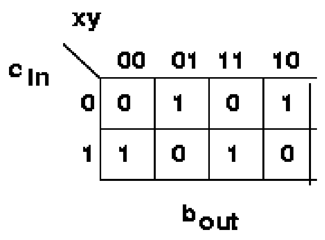
So here is the 'half adder' implemented in VHDL:

```
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.

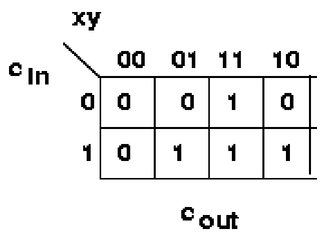
Now how would you make a 'full' adder?

cin	x	y	bout	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



$$\begin{array}{c|cccc} c_{in} \backslash xy & 00 & 01 & 11 & 10 \\ \hline 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{array}$$

b_{out}



$$\begin{array}{c|cccc} c_{in} \backslash xy & 00 & 01 & 11 & 10 \\ \hline 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \end{array}$$

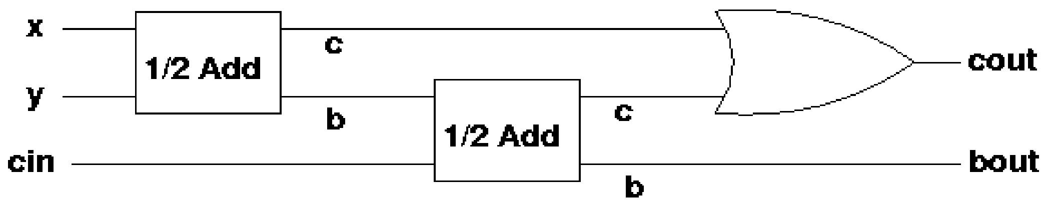
c_{out}

$$bout = x \oplus y \oplus cIn$$

$$cout = x^*y + x^*/y^*cIn + /x^*y^*cIn = x^*y + x \oplus y^*cIn$$

So a cascade of 2 half adders and an or gate does it

Arithmetic Operation: Full Adder: Include Carry In

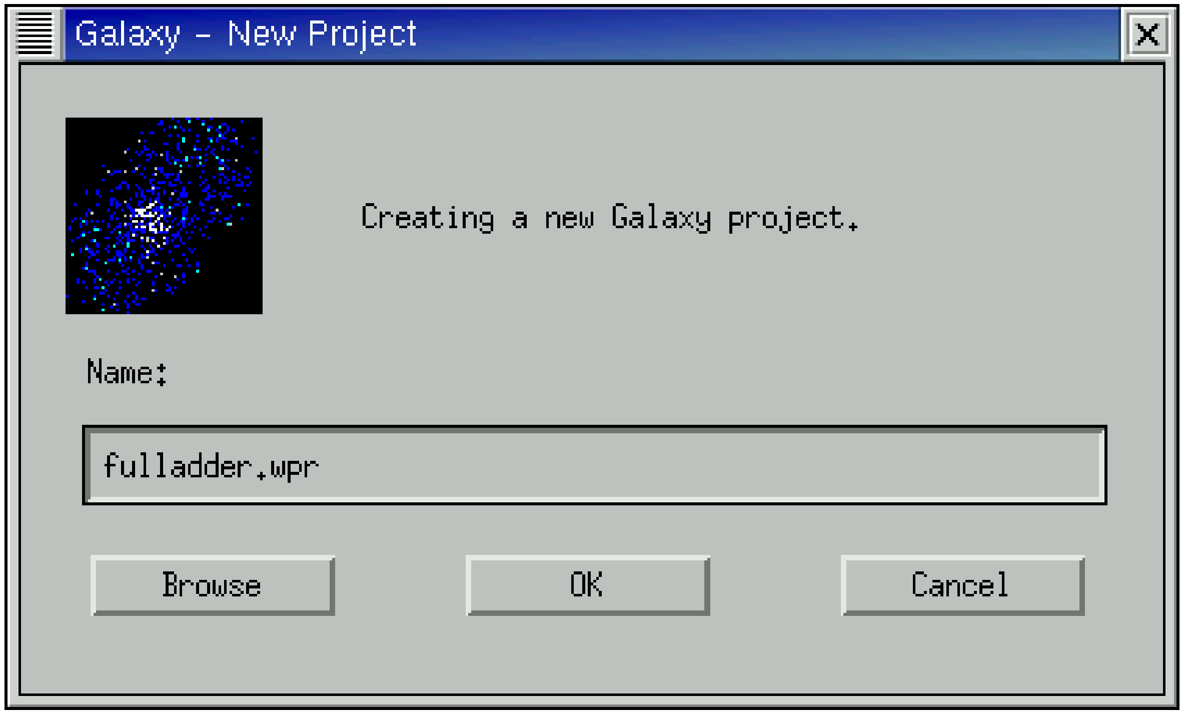


Here is an implementation of the full adder using component instantiation through a port map:

```
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;
begin
  -- 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 &
```

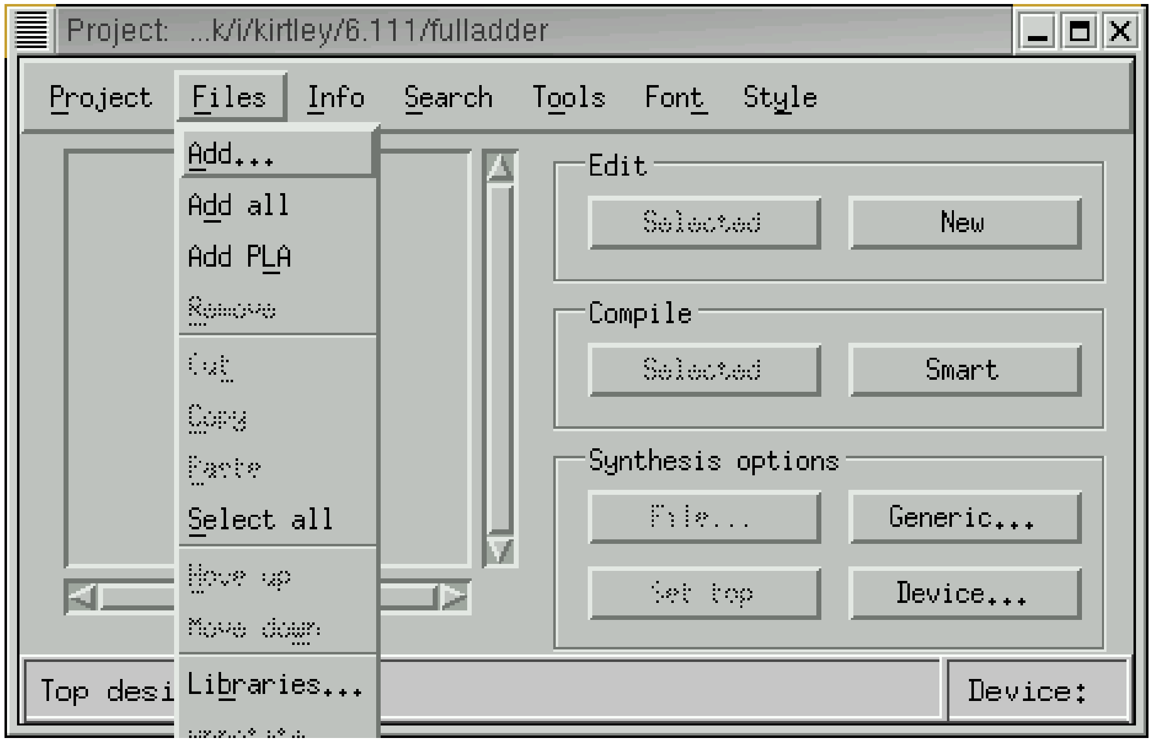

Galaxy - New Project

Creating a new Galaxy project.

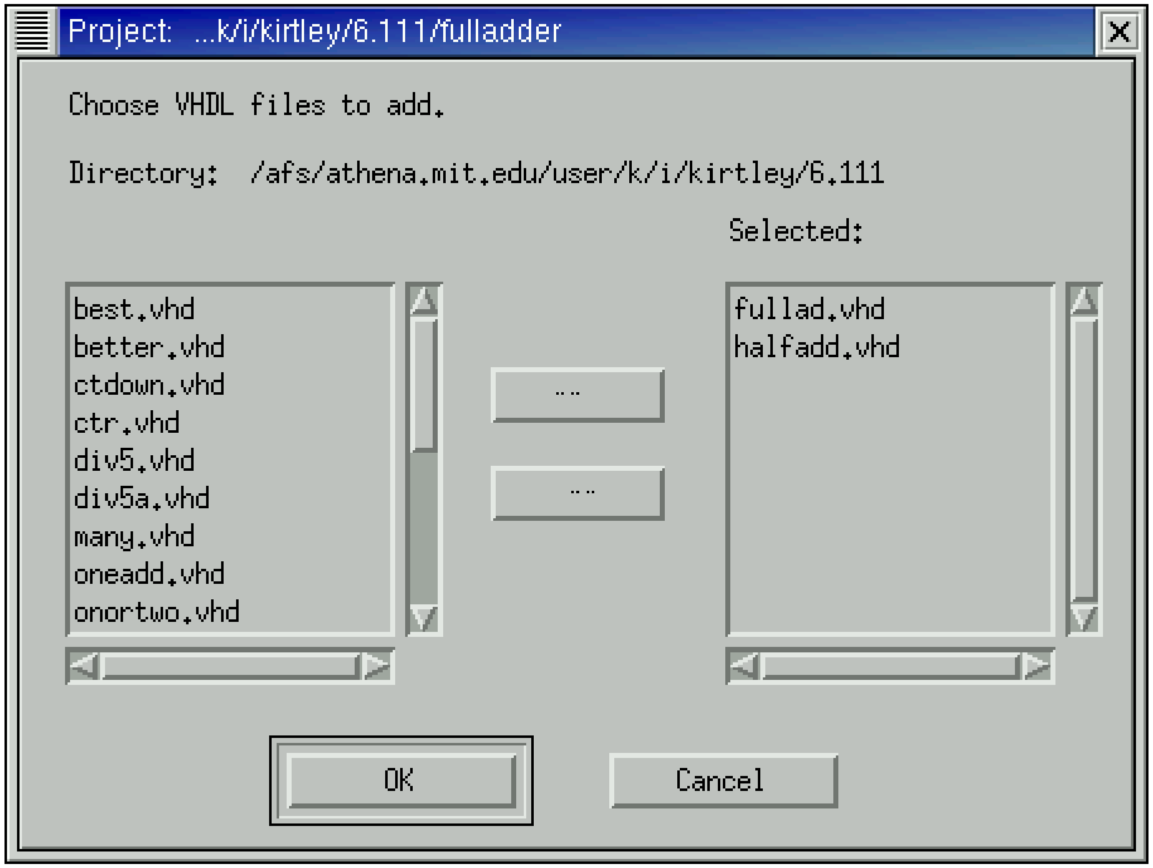
Name:

fulladder.wpr

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



Project: ...k/i/kirtley/6.111/fulladder

Choose VHDL files to add.

Directory: /afs/athena.mit.edu/user/k/i/kirtley/6.111

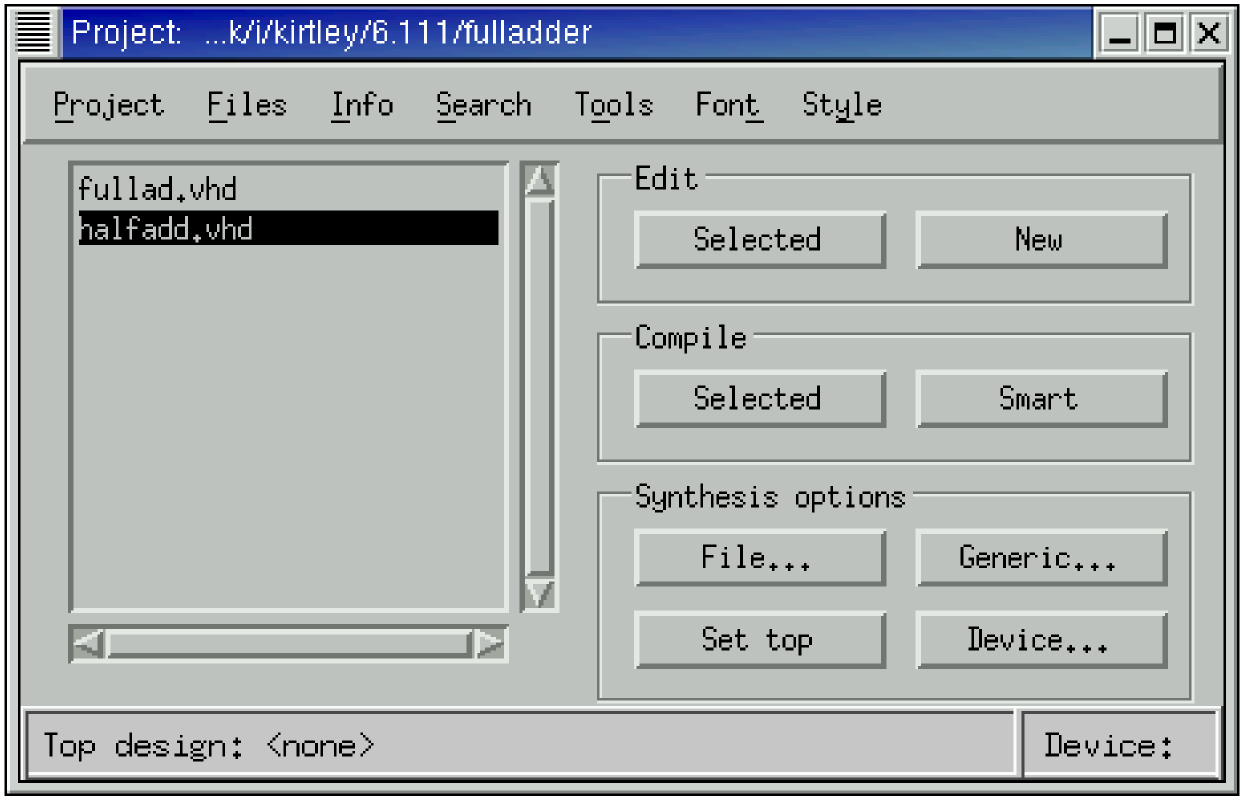
Selected:

best.vhd
better.vhd
ctdown.vhd
ctr.vhd
div5.vhd
div5a.vhd
many.vhd
oneadd.vhd
onortwo.vhd

fullad.vhd
halfadd.vhd

Here is what the project screen looks like now, with the files added.

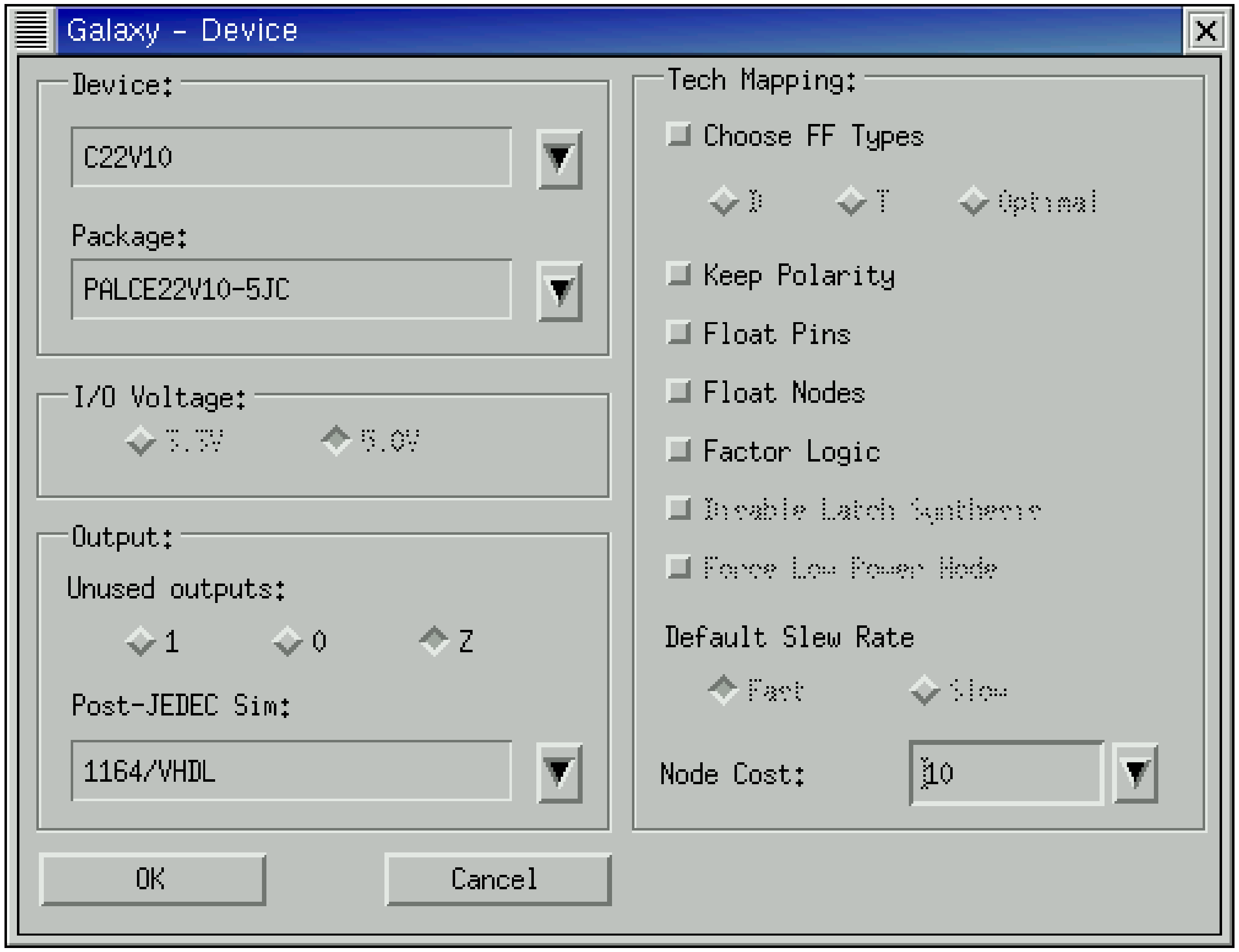
Next Step is to select a Device and select top file and Set Top



Top design: <none>

Device:

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.



Galaxy - Device

Device:
C22V10

Package:
PALCE22V10-5JC

I/O Voltage:
3.3V [✓ 5.0V]

Output:
Unused outputs:
1 0 [✓ Z]

Post-JEDEC Sim:
1164/VHDL

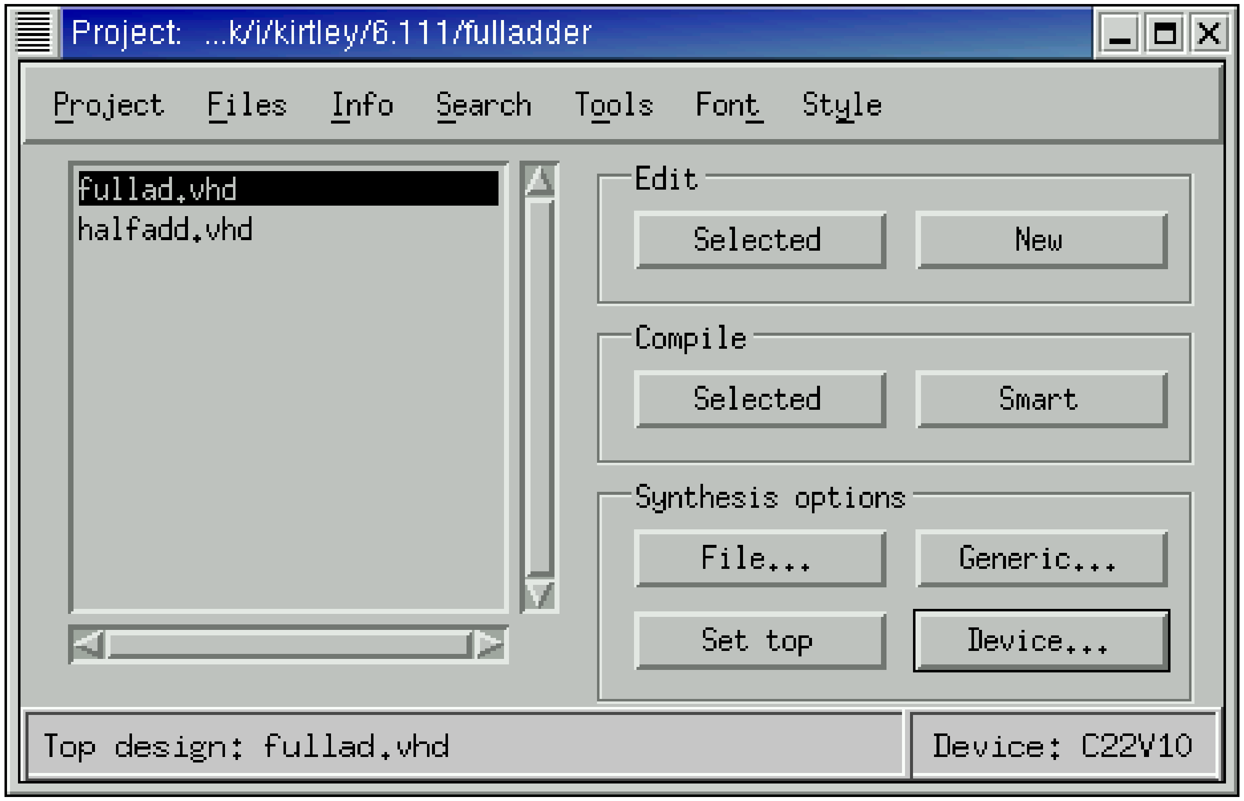
Tech Mapping:
☐ Choose FF Types
D T Optimal
☐ Keep Polarity
☐ Float Pins
☐ Float Nodes
☐ Factor Logic
☐ Disable Latch Synthesis
☐ Force Low Power Mode

Default Slew Rate
[✓ Fast] Slow

Node Cost: 10

Here it is ready to compile: device and top design file are selected.

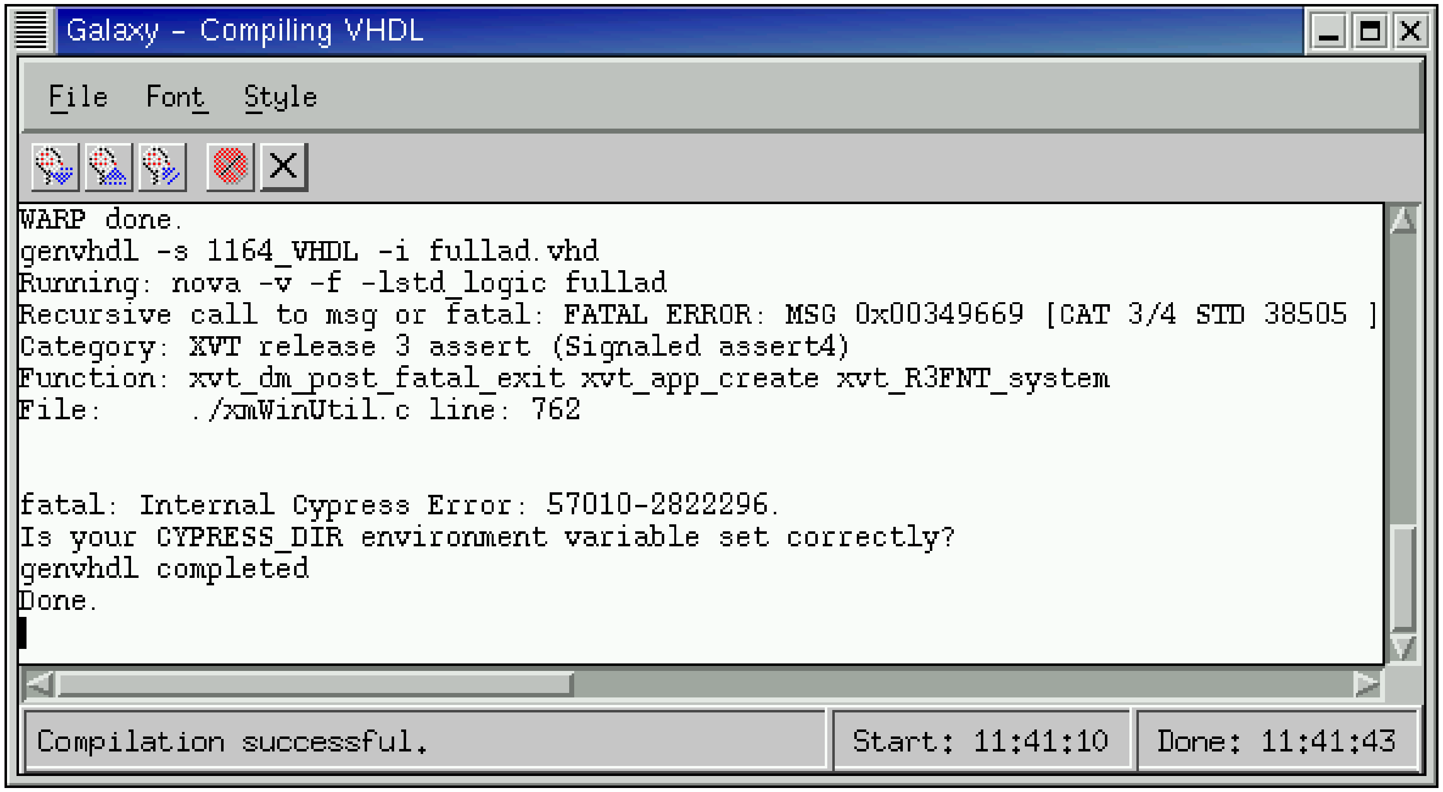
We will use' Smart' Compile: the program figures out what to do



Top design: fullad.vhd

Device: C22V10

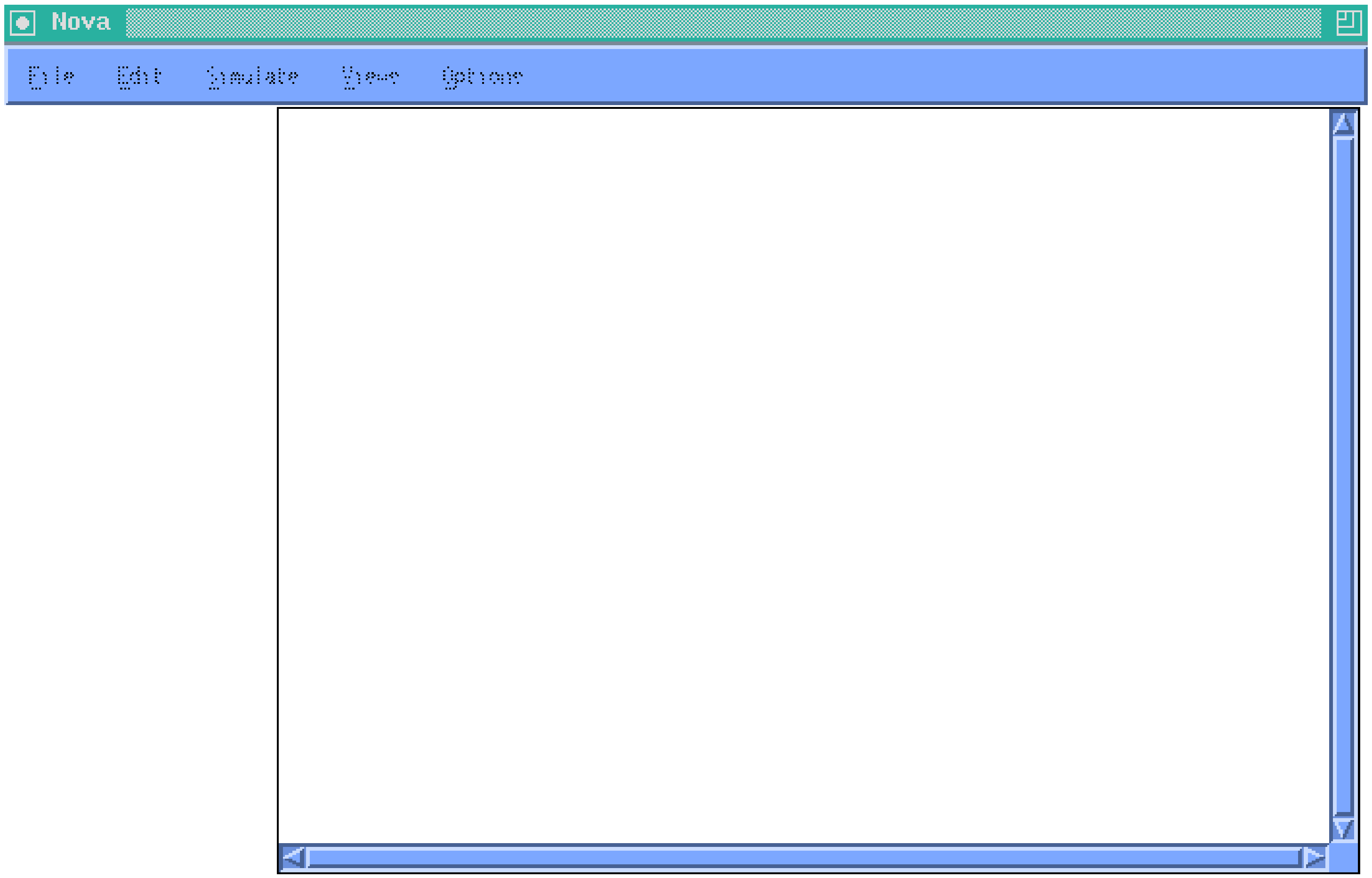
This is what the compile screen looks like. If therre are errors they will show up here. Note we have an error here at the very last step, which is setting up for NOVA. Not to worry: this was just a disagreemnt over displays. Note a lot of stuff scrolls by: see the scroll bar on the right.



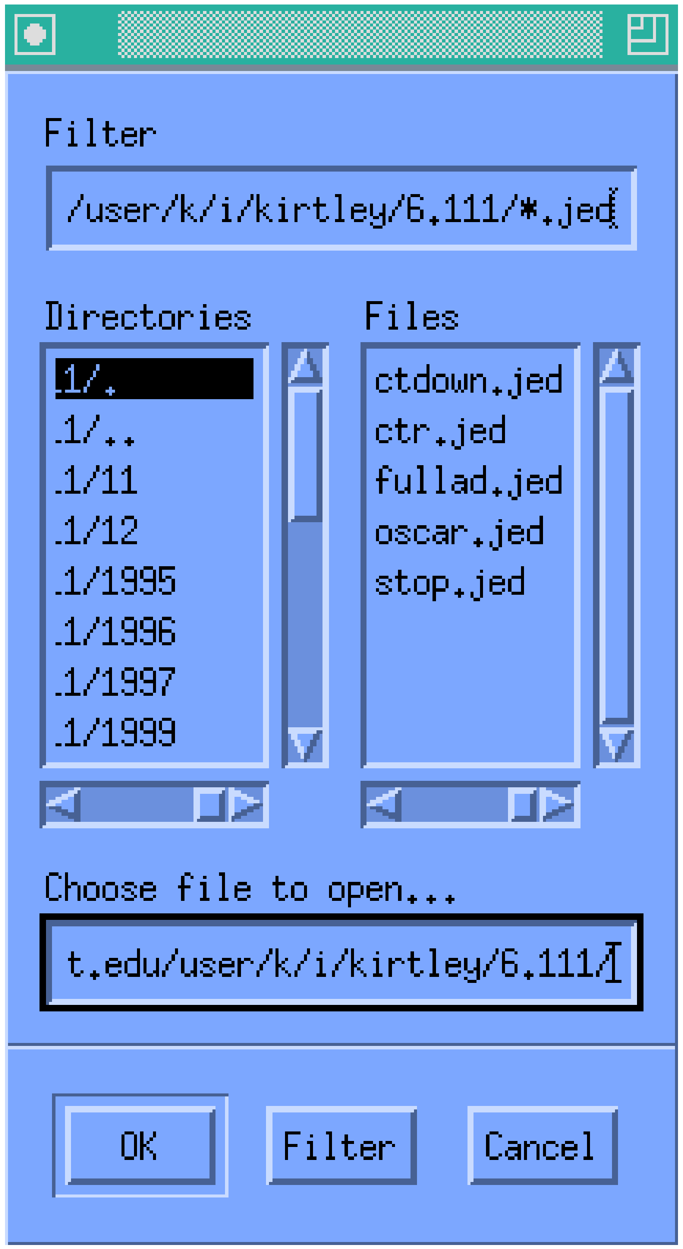
```
WARP done.
genvhdl -s 1164_VHDL -i fullad.vhd
Running: nova -v -f -lstd_logic fullad
Recursive call to msg or fatal: FATAL ERROR: MSG 0x00349669 [CAT 3/4 STD 38505 ]
Category: XVT release 3 assert (Signaled assert4)
Function: xvt_dm_post_fatal_exit xvt_app_create xvt_R3FNT_system
File:      ./xmWinUtil.c line: 762


fatal: Internal Cypress Error: 57010-2822296.
Is your CYPRESS_DIR environment variable set correctly?
genvhdl completed
Done.
```

```
Compilation successful.    Start: 11:41:10    Done: 11:41:43
```

Here is he 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`**



Filter

```
/user/k/i/kirtley/6.111/*.jed
```

Directories

```
.1/.
.1/..
.1/11
.1/12
.1/1995
.1/1996
.1/1997
.1/1999
```

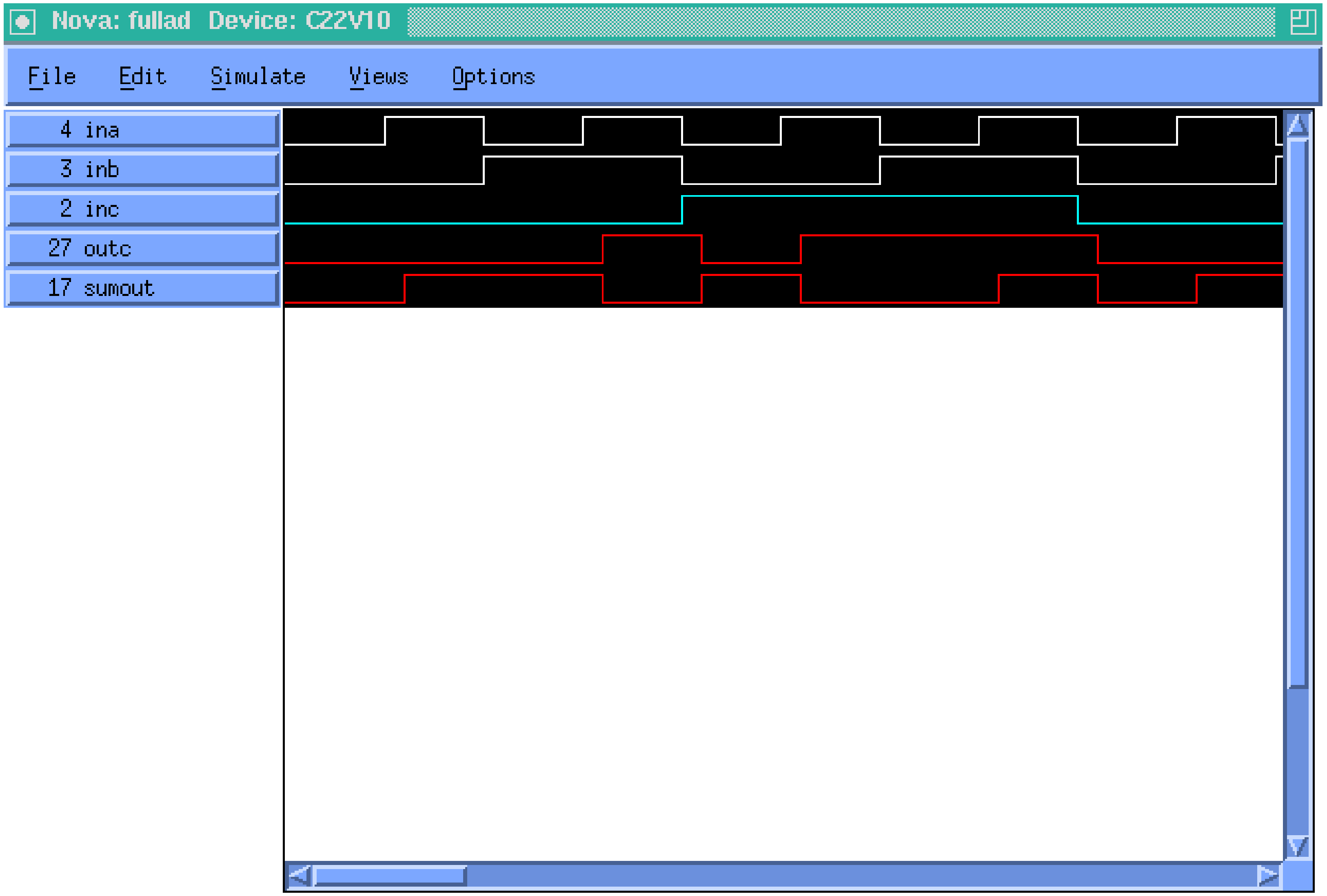
Files

```
ctdown.jed
ctr.jed
fullad.jed
oscar.jed
stop.jed
```

Choose file to open...

```
t.edu/user/k/i/kirtley/6.111/
```

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.


Nova: fullad Device: C22V10
File
Edit
Simulate
Views
Options
4 ina
3 inb
2 inc
27 outc
17 sumout