

Alliance 5.0 Tools HOWTO

but Read The Fine Manual pages as well

N.J.Gunton 01/01/01, 10/10/02, 03/04/03

School of Computer Systems Engineering

Grouping	Individual / Team
Prerequisites	Knowledge of Unix and Emacs
Courses	CRTS, any.
Requirements	Linux or Unix system with Alliance tools and programming ability.
Summary	Provides an overview of the VHDL tools used for UQC149S2 and/or UQC143H3 and/or UFS004C2

Note that this worksheet provides a summary of the tools that you will be using and their inter-relationships. It is not intended to provide the fine detail that you may need. For this you will have to refer to the `man` pages or the other worksheets.

1. The Alliance toolkit

To quote from the Alliance home page, (there's a link from my web page).

Alliance is a complete set of free CAD tools and portable libraries for VLSI design. It includes a VHDL compiler and simulator, logic synthesis tools, and automatic place and route tools. Advanced verification tools for functional abstraction and static timing analysis are part of the system. A complete set of portable CMOS libraries is provided, including a RAM generator, a ROM generator and a data-path compiler. Alliance is the result of a ten year effort spent at ASIM departement of LIP6 laboratory of the Pierre et Marie Curie University (Paris VI, France). Alliance has been used for research projects such as the 875 000 transistors StaCS superscalar microprocessor and 400 000 transistors IEEE Gigabit HSL Router.

However you only need to get to grips with some of the tools in the toolset. As you might gather from the quote, this is a complex and sophisticated set of tools that covers the entire process of ASIC design from VHDL simulation and testing, technology mapping onto standard cells, routing and placing of the cells, timing analysis, gate level extraction and generation of the files for a silicon foundry. Our interest lies in the tools for simulation and testing of VHDL code and the mapping into the technology that we will be using, the Lattice Semiconductor Corp. ispLSI® CPLDs. The first stages of mapping into these devices is done using locally developed tools while final fitting is through the Lattice synthesis tools.

The Alliance toolkit works with subsets of VHDL, namely those aspects of the language that can be synthesised, or implemented in hardware¹. In order to simplify this it breaks the language down into three subsets:

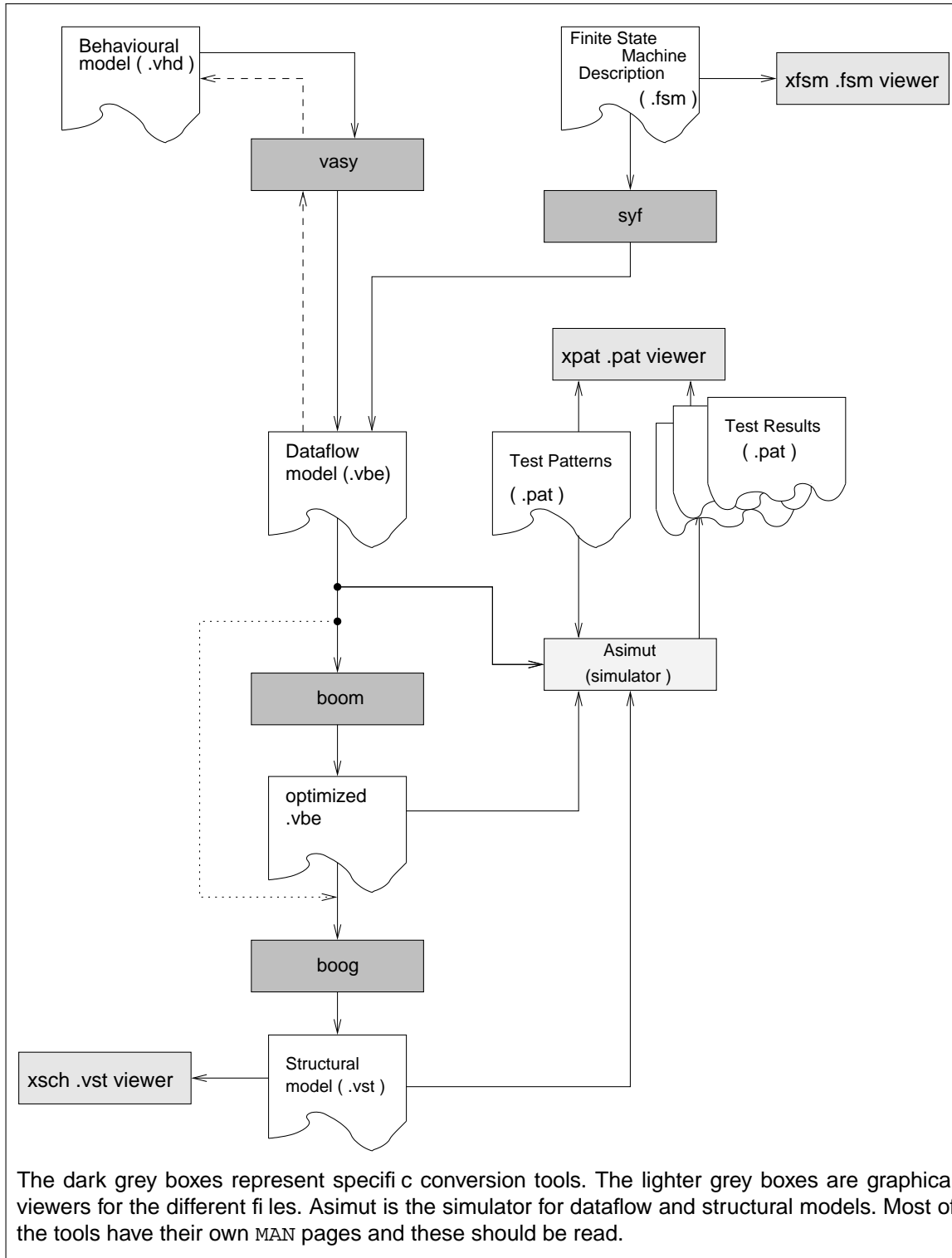
- **Dataflow** descriptions, identified by a `.vbe` suffix.
- **Structural** descriptions (hardware/schematic), identified by a `.vst` suffix
- **State-Machine** descriptions, identified by an `.fsm` suffix.
- **Behavioural** descriptions and industry standard descriptions, identified by a `.vhd` suffix.

Most of the tools do not require you to give the file suffix as they infer it from the context. When using the tool to convert a state-machine description into a dataflow description it will look for `filename.fsm` when given `filename`. If there is no file with the appropriate suffix then an error will be generated. This helpfully (confusingly?) allows you to use the same filename at all stages of the process, but with different suffixes.

¹ For example you can specify an input as receiving text but how does that translate into hardware?. It could be implemented as a serial interface or as a parallel interface, 7 bits or 8 bits or ???

2. Design Process Overview

There are specific tools for each language subset and for converting between them. For example a typical development flow, starting from either a dataflow or state machine description, would be :-



3. Design Flow

3.1. From a State Machine

Leaving aside, for the moment, the use of purely behavioural models as the starting point, the following is a typical design flow beginning with a finite state machine (fsm) description.

- Design an FSM and code it in the `.fsm` subset, see lab2 worksheet for an example. This would then be converted to a behavioral description (`.vbe`) for testing under simulation. The conversion is done using `syf`.

```
syf2 -aEV my_state_machine
```

This will convert an `fsm` model into a dataflow model using `asp` encoding and create two files, one will contain the dataflow (`.vbe`) description and the other the mapping, or encoding, of the state-names in your state machine to a binary representation³.

3.2. Model Simulation

- After developing your test sequences, or pattern files (`man 5 pat`, separate worksheet and below), you can test your dataflow design in simulation ...

```
asimut -b my_state_machinea4 test_input test_output
```

This requires the dataflow (`.vbe`) file, the name of the test pattern file and the name of the file to write the results to. The results can be viewed using `xpat` which will display them as waveforms.

There are now two possible routes at this stage. The dataflow model can be optimized before being synthesized, or 'fitted', to a particular implementation technology, or we can skip the optimization. If the eventual target is a CPLD or an FPGA then it is best to do minimal optimisation at this point, i.e. skip it. The 'fitter' for the target technology will endeavor to do some optimization as well in order to obtain the best 'fit' possible. If we have already optimized our model then there is little room for maneuver at the final fitting.

3.3. Optimization

- The optimization can be for speed or for area as there is a trade off between these two criteria. There are numerous other constraints that can be applied to the optimization process, see `man boom` for details.

```
boom my_state_machinea
```

This will use default options, optimising equally on area and on speed. It will create an output file with `_o` appended and a `.vbe` suffix. The effect of the optimisation is shown in the two extracts below. They are from the output messages of the fitter `boog` for an unoptimized version of the read-write buffer⁵ and an optimized version using the default settings for both `boom` and `boog`. Note that there is a considerable difference in the actual logic gates used for each example yet both implement the same overall function. This topic will be revisited later.

² details of command flags for all the commands are given at the end of this document

³ There are many different ways of encoding states in a state machine. See the reference on the <http://www.cems.uwe.ac.uk/~ngunton/UFS004C3> web page.

⁴ the `a` added to the filename is from the encoding algorithm used in the example. Alternatively provide an output filename as the final parameter to `syf`

```
Quick estimated critical path (no warranty)...932 ps
  from 'read_write' to 'controller_current_state 0'
Quick estimated area (with over-cell routing)...26500 lambda2
Details...
  inv_x2: 3
  no2_x1: 2
  sff1_x4: 2
  noa22_x1: 2
  ao2o22_x2: 2
  na2_x1: 2
  o2_x2: 2
  on12_x1: 1
  Total: 16
```

unoptimized results

```
Quick estimated critical path (no warranty)...1483 ps
  from 'controller_current_state 1'
  to 'controller_current_state 0'
Quick estimated area (with over-cell routing)...24750 lambda2
Details...
  an12_x1: 2
  no2_x1: 2
  sff1_x4: 2
  inv_x2: 2
  a2_x2: 2
  oa22_x2: 1
  no3_x1: 1
  oa2a22_x2: 1
  xr2_x1: 1
  Total: 14
```

optimized, with default settings

Note that the area (gate count) has been reduced but at the expense of performance. Optimization will tend to have more impact on large designs than on small designs such as the example used here.

3.4. Synthesis 1.

- If required, the dataflow description can be mapped into a particular target technology. This is the first part of the process of synthesising the VHDL into hardware. A structural, or netlist, description is produced. The description is in terms of the gates or cells available to the target hardware. A library of cell descriptions is selected and the dataflow description mapped into this library. The default library used with the Alliance 'fitter' is a portable standard CMOS cell library called **sxlib**. The Alliance synthesis tool is called `boog` and its simplest usage is ...

```
boog my_state_machine
```

The minimum set of parameters required is the name of the dataflow description. The name of the input file will be used to write the structural description into. The file name can be the same as it is the file suffix that differentiates them (`.vbe`, `.vst`). `boog` will also do some optimization as well as providing an estimate for the critical path through the model. See above for examples.

The technology is selected by the value of the environment variable `MBK_TARGET_LIB` and for our purposes is normally set to `sxlib` or `latlib`. The latter is a local library and is not part of the Alliance toolkit.

⁵ from lab worksheet 2

3.5. Simulation 2

• Simulation testing is normally repeated at this stage. This uses the same test patterns that were used for the dataflow testing and the same simulator. However we are now testing our hardware description. The test results should be identical.

```
asimut my_state_machine test_input test_output2
```

Note the absence of the `-b` flag as we are NOT simulating a dataflow description.

3.6. Synthesis 2

• Assuming all has gone well so far we can now create an **EDIF**⁶ format file for transfer to the Lattice synthesis tool. This can be done from either a dataflow or a structural description⁷.

```
sxconv -b my_state_machine or sxconv -s my_state_machine
```

Note that structural descriptions can be written from scratch and that they may often be hierarchical, ie. descriptions of subcomponents can be called from a higher level description. An hierarchical description must first be flattened into a single logical level before being converted to .edf with `xmap`. this is done with the tool `flatlo`, flatten logical model as in ...

```
flatlo -r my_complex_structure my_flattened_structure
```

There are a number of issues regarding the conversion of files to the Lattice flavour of EDIF and these are covered in detail in a separate section/worksheet

This completes the overview of the process from state machine to synthesis-able code. The actual synthesis and download onto the Lattice devices is done using the Lattice tools on the NT platform and there are separate worksheets covering this process.

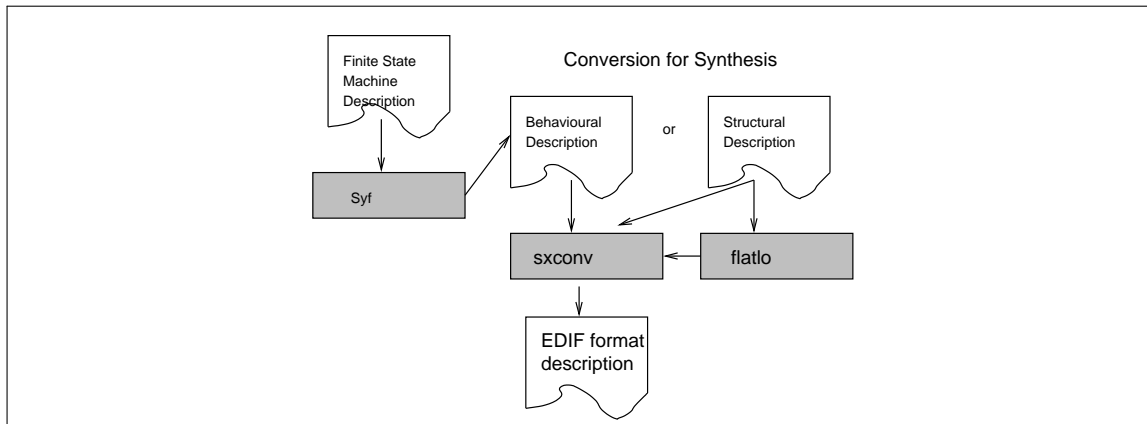
There are MAN pages describing the valid syntax for each subset of the language as well as design & code examples in

3P11/3P27/3P28 Linux machines:

```
/usr/local/alliance/examples  
/usr/local/alliance/share/tutorials
```

Solaris machines (milly etc.)

```
/pub_domain/alliance/share/tutorials
```



typical conversion process for CPLD synthesis

⁶ EDIF (Electronic Design Interchange Format) is an industry standard format for the exchange of electronic (as in hardware) designs and is intended to simplify the process of moving files between various design packages. The reality isn't quite as good as that!

⁷ Note that you may get some funny errors when using `sxconv` to convert hand written structural vhd. This is because `sxconv` was written to handle machine generated files and relies on the layout of the file. This will be resolved in the next release of `sxconv`.

4. Tool Descriptions

syf : Finite State Machine synthesizer

For full details rtfmp. The main issues in converting a state machine description to a dataflow description are:

•encoding:

The way in which each state is represented in the hardware. Each state will be assigned a unique binary value which will be held in a set of registers.

The number of registers, n , that will be needed, at least, can be calculated where the number of states $\leq 2^n$. ie. 1 register = 2 states, 2 registers = 4 states, 3 registers = 8 states etc.

If 'one-hot' encoding is used then as many registers as there are states will be needed. ie. 2 states = 2 registers, 3 states = 3 registers etc.

•latches:

How the outputs are handled. The outputs can be purely the result of combinational logic and change to reflect input changes regardless of state or they can be latched (set on input and state or set on state) and held until actively changed.

•commands:

Below is a subset of the command line syntax showing the more important flags. Note that an output filename is not required, the default will be the input filename with the initial of the encoding method appended to it.

```
syf -a|o [-EOV] input_name [output_name]
```

SYF		
Purpose	Flag	Description
Encoding	-a -o	asp encoding one-hot encoding Check out chapter 9 of Contemporary Logic Design on this site www.redbrick.dcu.ie/academic/CLD for an extensive discussion of FSM encoding schemes.
Options	-E	generate an encoding file showing the mapping between the register values and your state names. Useful for debugging!
	-O	ensure that all outputs are registered outputs. ie. Outputs will remain latched until explicitly changed by the state machine.
	-V	Verbose mode, explain in great detail each step of the synthesis process and print out some statistics.

vasy Vhdl Analyzer for SYnthesis

This very useful tool maps most VHDL RTL descriptions into the Alliance subset and Alliance subsets into a form accepted by most synthesis tools. It allows the use of variables, expands generics, parameters and unrolls static loops. The most common usage will be in creating alliance dataflow models with the command line

```
vasy -vaop -I vhd filename
```

VASY			
Purpose	Flag	Description	Example
output format	-a	specify the output file format as Alliance	<code>vasy -a -I vhd myrtlmodel</code>
input format	-l <i>fmt</i>	specify the input format suffix	<code>vasy -a -I vhd myrtlmodel</code>
control synthesis	-L	generate parameter file listing signals to be kept during synthesis stage	<code>vasy -aoL -I vhd myrtlmodel</code>
insert power connectors	-p	add vss & vdd to the entity description	<code>vasy -aop -I vhd myrtl</code>
tell me lots	-v	be verbose during the translation process	<code>vasy -vaop -I vhd myrtl</code>
Use local packages	-P <i>name</i>	reference a tool specific file containing package names	<code>vasy -a -I vhd myrtl -P pkg_file</code>

asimut : A SIMULaTor

Again, if in doubt, RTFMP. Otherwise the minimum needed is given in the table. It requires three filenames at minimum, the code to be simulated, the input pattern and the name of the output file to write to. It assumes a `.vst` file as default. Further options permit you to split the pattern file into several parts and save the simulation state at intermediate points and restart/continue⁸.

ASIMUT			
Purpose	Flag	Description	Example
select input	-b	datafbw input	<code>asimut -b myrtlmodel inpattern outpattern</code>
	-c	compile only, ie check for errors but don't simulate	<code>asimut -c mynetmodel</code>
	none	assumes structural input	<code>asimut mynetmodel inpattern outpattern</code>
initialize state	-i filename	read a save file (<code>.sav</code>), and initialize the simulator to it before reading the pattern file. Note that save files are created by entries in the pattern file.	<code>asimut -b -i mysave myrtl inpat outpat</code>

*** Note that it seems that a `.sav` file can only be created at the end of simulating an rtl description and that attempting to save state from a structural simulation results in a segmentation fault. Awaiting feedback from the developers.

boog : Binding and Optimizing On Gates

```
boog [-hmxold] infile [outfile] [laxfile]
```

This combines the optimization of the datafbw description and the mapping into a specific technology. This tool takes our description and creates a netlist based on the selected cell library. Much of the time we will use the default library.

⁸ See also `genpat`, a set of C macros for writing large or complex or repetitive pattern files. See also man page for `pat` (pattern file structure).

You can get away with no flags to this command, however there are occasions when we need to modify the defaults. You might find the `-x` flag useful, this provides a colour map for `xsch` either highlighting the critical path in your design or providing a delay gradient. The other flags are shortcuts for values that can be placed in a `.lax` (logic synthesis parameter) file. This file is referred to by many of the tools so is worth a section in its own right (*qv*)

BOOG			
Purpose	Flag	Description	Example
critical path	<code>-x 0</code>	select xsch mode	<code>boog -x 0 myrtlmodel</code>
timing gradient	<code>-x 1</code>	select xsch mode	<code>boog -x 1 myrtlmodel</code>
control synthesis	<code>-l file</code>	use parameter file controlling many aspects of the synthesis process. Different parts of the file are used by different tools	<code>boog myrtlmodel -l laxfile</code>

Side-bar: The default technology library

The default library that is used for the mapping is `sxlib`. This is a library of symbolic standard cells with timing values based on a 0.35 micron process. A cell is the equivalent of a simple logic gate or logic block such as a flip-flop or multiplexor. These symbolic cells would be mapped to a specific process at a later stage.

xpat or how to view your pattern files sanely

Unfortunately there is little documentation for this tool at all, or for the similar `xsch` the schematic viewer :-| However the following should help with both the waveform viewer `xpat` and the schematic viewer `xsch`. Start the program on the command line with the command `xpat&` to keep the xterm free (you have to be running X to use this program). Wait⁹ for the box 'Touch Me' to appear and click on it, it's a nuisance but Adjust the size of the application to suit yourself, it starts out rather large so resize it, select **Setup** → **Save Config** and in future it will be your preferred size as default. Use **File** → **Open** and select a suitable pattern file¹⁰.

- The drop down menus are fairly self explanatory, Edit is a bit of a misnomer, it contains:
 - Identify:

Selecting this and then clicking on a waveform will pop up a box giving the waveforms I/O name, I/O direction, current value, previous value and next value. In `xsch`, when applied to a component, this will give the component name and the instance name. When applied to a signal, it will list the ports to which the signal is connected and which instances they apply to.
 - Connected:

This is mainly of use in `xsch`. select this and then click on a signal (wire). It will highlight the signal, allowing you to trace it on the schematic.
 - Add/Delete_Cursor:

This only applies to `xpat`. It allows you to add (delete) a marker on your waveform.

xsch : X SChematic viewer

```
xsch [-l file_name] [-xor] [-install] [-force] [-I input_format]
[-slide file_name ...]
```

⁹ **Handy Hint #49** Grabbing the `xpat/xsch` window with the mouse and moving it causes the pop-up to appear immediately :).

¹⁰ There's one or two in the examples directory `/usr/local/alliance/examples` on kenny. There are also some example `vst` files that `xsch` will display as schematics in the examples directory.

After many years this tool finally has a `man` page (`qv`). This provides a schematic view of your structural models. It can also provide schematic views of the transistor level (`.al`) and `.vbe` levels as well.

If it produces garbage on a hand-crafted schematic then you've got your wiring wrong. Alas, if this happens and you then want to view your modified schematic, you will have to quit `xsch` and restart it as it won't let go of the original, even though the file will have changed on disk. It will also output a copy of your schematic in `xfig` format although there are a couple of bugs in the exported file. The worst is the 124 metre long line drawn from a multiplexor! The multiplexor is the only device thus afflicted. The other feature is that the output file will contain all the labels including those switched off in the on-screen display.

The default input format is the standard `.vst` file. In fact it is rare to use the command line options as apart from the slideshow and input formats everything else is available from the pull-down menus.

Note that both `xsch` and `xpat` are designed to run with the `motif` x libraries and will perform extremely sluggishly with the free `lesstif` libraries that are distributed with many GNU/Linux distributions. You will need to replace the `lesstif` libraries with the `openmotif` libraries in order to resolve this.

GLOSSARY:

ASIC

Application Specific Integrated Circuit, A custom design created by engineers specifically for one IC. Very few full custom ICs are created today. See CBIC.

Behavioural Model

The term given to an abstract model of hardware that describes the behaviour of the hardware in terms of its functionality. It often contains HDL constructs that cannot be synthesized. eg. it may contain floating point or text input/output. **Warning** : The Alliance toolkit often refers to behavioural models when it means dataflow or RTL level descriptions as in the use of the `-b` flag to `asimut`.

CBIC

Cell Based ASIC, pronounced 'see-bick', uses predesigned logic blocks or cells with custom, user defined interconnects between the cells.

Cell (standard)

Refers to standard predesigned logic units such as AND gates, OR gates and other more complex units. They are used for the development of Cell-based ASICs, sometimes called CBICs. They allow for fast development by designers as the cells are pre-tested. They are a little like bricks in a wall in that they are normally all of the same height which simplifies automatic assembly of the ASIC. To view a cell try `graal` and browse to `$ALLIANCE_TOP/cells/sxlib/inv_x1.ap`, this is a simple inverter. Also have a look at MJS Smiths book on ASICs, ISBN 0-201-50022-1.

CPLD

Complex Programmable Logic Device. These are the largest of the programmable logic device family. The interconnects between the logic elements are programmable allowing rapid customization of the device to the users design. Note that the distinction between these and FPGAs is becoming increasingly blurred.

Dataflow Model

Also referred to as an RTL description. An HDL description that describes how the data moves from the inputs to the outputs. The movement is described in terms of clocks, register transfers, busses etc and is a concurrent description. These designs can usually be synthesised.

EDIF

Electronic Design Interchange Format. A nominally portable format for describing electronic designs as either netlists or schematic icons. Many of the problems arise because different synthesis tools use different names for standard logic units, eg 'a2, and2, and_2' all referring to a 2 input and gate.

'fitter' 'fitting'

The software used (the fitter) for the process of converting, or fitting, the design into a particular FPGA or CPLD. The design would be described in an HDL.

FPGA

Field Programmable Gate Array, similar in concept to the CPLD in that it uses logic gates and programmable interconnects, however the logic may be SRAM based which means that the programming is lost on power-down, thus requiring extra start-up logic to read in the programming data from ROM.

HDL Hardware Description Language, a high level language for describing the behaviour or the structure of a hardware design. Some HDLs are higher level than others. Examples include VHDL, Verilog, ABL, EDIF, although the latter is normally machine generated rather than written by humans.

Netlist

The term given to a description of a design which specifies how the various components are connected together rather than a functional description. A schematic showing logic gates is akin to a netlist. In VHDL it is represented by a structural model eg an Alliance `.vst` model.

RTL or Register Transfer Level description shows the movement of data through the registers and busses of the design, often called a dataflow model(qv) in VHDL.

Structural Model

The lowest level HDL description. Often referred to as a 'netlist'. It specifies the hardware in terms of active components and the interconnections between the components. It is the

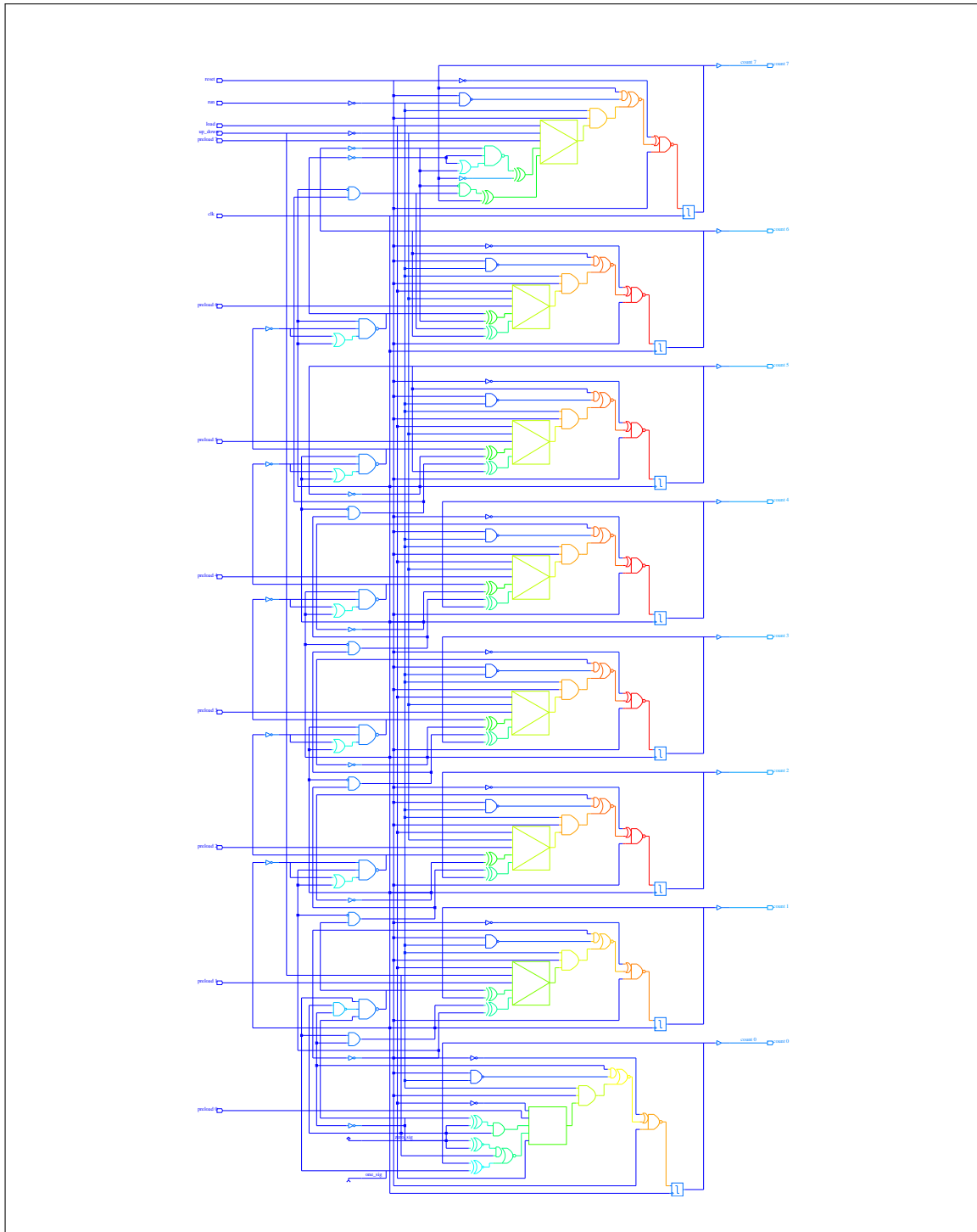
easiest to synthesize. The simulation of a structural model can include detailed timing models for the components used.

Synthesis

The process of converting our VHDL description into a technology specific format suitable for 'fitting' (qv) to a device, or for a silicon foundry (chip manufacturer) to manufacture from.

Technology

In this context it refers to the synthesis target architecture. For example the library of components that we choose may be dictated by our choice of target. Examples of targets might include cell based ASIC, custom ASIC, FPGA or CPLD, with the latter two having vendor specific variants (Xilinx, Lattice, Altera etc.). The technology library file allows us to map our RTL or dataflow design into a specific technology when creating our netlist.



Example netlist with timing delays shown through the use of a colour gradient