NAME

iverilog-fpga − FPGA code generator for Icarus Verilog

SYNOPSIS

iverilog -tfpga [iverilog-options] sourcefile

DESCRIPTION

The FPGA code generator supports a variety of FPGA devices, writing EDIF output depending on the target. You can select the architecture of the device, and the detailed part name. The architecture is used to select library primitives, and the detailed part name is written into the generated file for the use of downstream tools.

The code generator is invoked with the -tfpga flag to iverilog. It understands the part= and the arch= parameters, which can be set with the -p flag of iverilog:

```
iverilog -parch=virtex -ppart=v50-pq240-6 -tfpga foo.vl
```

This example selects the Virtex architecture, and give the detailed part number as v50-pq240-6. The output is written into a.out unless a different output file is specified with the -o flag.

OPTIONS

iverilog -tfpga accepts the following options:

-parch=*family*

The *family* setting further specifies the target device family. See FPGA FAMILIES below.

-ppart=*device*

This specifies a specific device in the form of a detailed part number. The format of this number is defined by the part vendor. In most cases, the device string is taken literally and written as is to the EDIF output.

FPGA FAMILIES

The following is a list of architecture types that this code generator supports.

lpm This is a device independent format, where the gates are device types as defined by the LPM 2 1 0 specification. Some backend tools may take this format, or users may write interface libraries to connect these netlists to the device in question.

The **lpm** family is the default if no other is specified.

virtex If this is selected, then the output is formatted as an EDIF 2 0 0 file, suitable for Virtex class devices. This is supposed to know that you are targeting a Virtex part, so can generate primitives instead of using external macros. It includes the VIRTEX internal library, and should work properly for any Virtex part.

virtex2 If this is selected, then the output is EDIF 2 0 0 suitable for Virtex-II and Virtex-II Pro devices. It uses the VIRTEX2 library, but is very similar to the Virtex target.

EDIF ROOT PORTS

The EDIF format is explicit about the interface into an EDIF file. The code generator uses that control to generate an explicit interface definition into the design. (This is *not* the same as the PADS of a part.) The generated EDIF interface section contains port definitions, including the proper direction marks.

With the (rename ...) s-exp in EDIF, it is possible to assign arbitrary text to port names. The EDIF code generator therefore does not resort to the mangling that is needed for internal symbols. The base name of the signal that is an input or output is used as the name of the port, complete with the proper case.

However, since the ports are single bit ports, the name of vectors includes the string "[0]" where the number is the bit number. For example, the module:

```
module main(out, in);
   output out;
   input [2:0] in;
   [...]
endmodule
```

creates these ports:

```
out   OUTPUT
in[0] INPUT
in[1] INPUT
in[2] INPUT
```

Target tools, including Xilinx Foundation tools, understand the [] characters in the name and recollect the signals into a proper bus when presenting the vector to the user.

PADS AND PIN ASSIGNMENT

The ports of a root module may be assigned to specific pins, or to a generic pad. If a signal (that is a port) has a PAD attribute, then the value of that attribute is a list of locations, one for each bit of the signal, that specifies the pin for each bit of the signal. For example:

```
module main( (* PAD = "P10" *) output out,
          (* PAD = "P20,P21,P22" *) input [2:0] in);

   [...]

endmodule
```

In this example, port ‘‘out’’ is assigned to pin 10, and port ‘‘in’’ is assigned to pins 20-22. If the architecture supports it, a pin number of 0 means let the back end tools choose a pin. The format of the pin number depends on the architecture family being targeted, so for example Xilinx family devices take the name that is associated with the "LOC" attribute.

NOTE: If a module port is assigned to a pin (and therefore attached to a PAD) then it is *not* connected to a port of the EDIF file. This is because the PAD (and possibly IBUF or OBUF) would become an extra driver to the port. An error.

SPECIAL DEVICES

The code generator supports the "cellref" attribute attached to logic devices to cause specific device types be generated, instead of the usual device that the code generator might generate. For example, to get a clock buffer out of a Verilog buf:

```
buf my_gbuf(out, in);
$attribute(my_buf, "cellref", "GBUF:O,I");
```

The "cellref" attribute tells the code generator to use the given cell. The syntax of the value is:

```
<cell type>:<pin name>,...
```

The cell type is the name of the library part to use. The pin names are the names of the type in the library, in the order that the logic device pins are connected.

EXAMPLES

COMPILING WITH XILINX FOUNDATION/iSE Compile a single-file design with command line tools like so:

```
% iverilog -parch=virtex -o foo.edf foo.vl
% edif2ngd foo.edf foo.ngo
% ngdbuild -p v50-pq240 foo.ngo foo.ngd
% map -o map.ncd foo.ngd
% par -w map.ncd foo.ncd
```

AUTHOR

Steve Williams (steve@icarus.com)

SEE ALSO

iverilog(1), **<http://www.icarus.com/eda/verilog/>**

COPYRIGHT