## NAME
iverilog-vpi - Compile front end for VPI modules

## SYNOPSIS
**iverilog-vpi** [options] *sourcefile...*

## DESCRIPTION
*iverilog−vpi* is a tool to simplify the compilation of VPI modules for use with Icarus Verilog. It takes on the command line a list of C or C++ source files, and generates as output a linked VPI module. See the **vvp**(1) man page for a description of how the linked module is loaded by a simulation.

By default the output is named after the first source file. For example, if the first source file is named *foo.c*, the output becomes *foo.vpi*.

## OPTIONS
*iverilog−vpi* accepts the following options:

**-l***library*  Include the named library in the link of the VPI module. This allows VPI modules to further reference external libraries.

**-I***directory*
> Add *directory* to the list of directories that will be searched for header files.

**-D***define*  Define a macro named *define*.

**--name=***name*
> Normally, the output VPI module will be named after the first source file passed to the command. This flag sets the name (without the .vpi suffix) of the output vpi module.

## PC-ONLY OPTIONS
When built as a native Windows program (using the MinGW toolchain), by default *iverilog−vpi* will attempt to locate the MinGW tools needed to compile a VPI module on the system path (as set by the PATH environment variable). As an alternative, the user may specify the location of the MinGW tools via the following option.

**-mingw=***path*
> Tell the program the root of the MinGW compiler tool suite. The **vvp** runtime is compiled with this compiler, and this is the compiler that *iverilog−vpi* expects to use to compile your source code. If this option accompanies a list of files, it will apply to the current build only. If this option is provided on its own, *iverilog−vpi* will save the *path* in the registry and use that path in preference to the system path for subsequent operations, avoiding the need to specify it on the command line every time.

## INFORMATIONAL OPTIONS
*iverilog−vpi* includes additional flags to let Makefile gurus peek at the configuration of the *iverilog* installation. This way, Makefiles can be written that handle complex VPI builds natively, and without hard-coding values that depend on the system and installation. If used at all, these options must be used one at a time, and without any other options or directives.

**--install-dir**
> Print the install directory for VPI modules.

**--cflags**   Print the compiler flags (CFLAGS or CXXFLAGS) needed to compile source code destined for a VPI module.

**--ldflags**   Print the linker flags (LDFLAGS) needed to link a VPI module.

**--ldlibs**   Print the libraries (LDLIBS) needed to link a VPI module.

Example GNU makefile that takes advantage of these flags:

```
CFLAGS = −Wall −O $(CFLAGS_$@)
VPI_CFLAGS := $(shell iverilog-vpi −−cflags)
CFLAGS_messagev.o = $(VPI_CFLAGS)
CFLAGS_fifo.o = $(VPI_CFLAGS)
messagev.o fifo.o: transport.h
messagev.vpi: messagev.o fifo.o
        iverilog-vpi $ˆ
```

## AUTHOR
Steve Williams (steve@icarus.com)

## SEE ALSO
iverilog(1), vvp(1), **<http://iverilog.icarus.com/>**, **<http://mingw-w64.yaxm.org/>**,

## COPYRIGHT
Copyright ©  2002−2015 Stephen Williams

This document can be freely redistributed according to the terms of the
GNU General Public License version 2.0