Symbiflow Installation Guide and Tutorial

About This Document

This document provides the details of the Symbiflow package installation and the various commands supported by the tool. It covers the usage of the tool by going over a simple example, and how to install Symbiflow on the Linux operating systems.

System Requirements

Requirements	Linux	CentOS	Ubuntu
Processor	Intel Xeon® or similar processors	Intel Xeon or similar processors	Intel Xeon or similar processors
RAM Size	2 GB or more		
Free Hard-Disc space	5GB or more		

Software Requirements

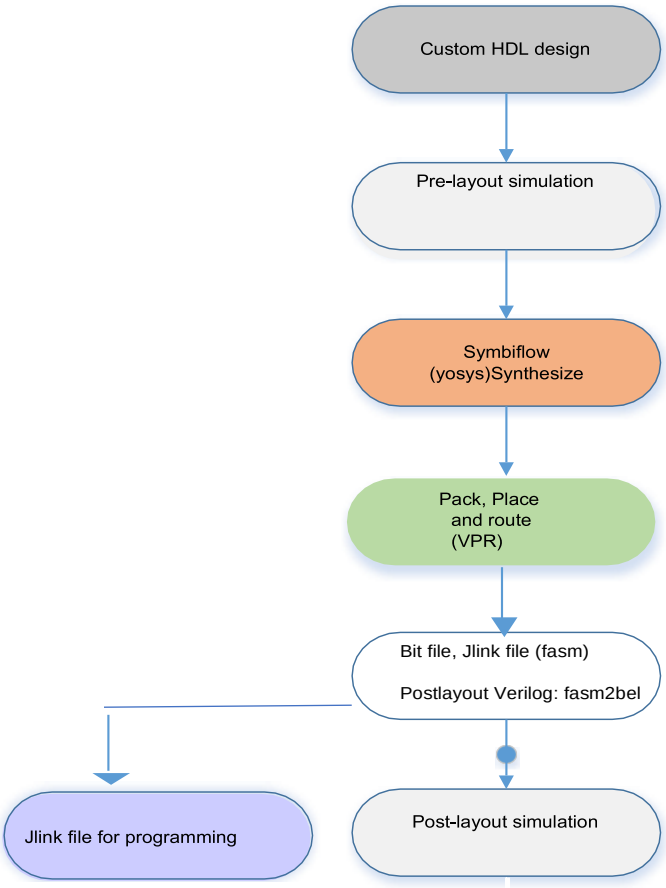
Refer : https://github.com/QuickLogic-Corp/quicklogic-fpga-toolchain

Installing Symbiflow on Linux

To install Symbiflow on Linux:

1. Execute the following commands in the terminal to set the execute permission for the .run file:
   `> chmod 755 Symbiflow<version>.gz.run`
2. Set the "INSTALL_DIR" variable by executing the following command:
   `> export INSTALL_DIR=<path>"`
3. Execute the .run file from the terminal:
   `>./Symbiflow<version>.gz.run`

Symbiflow: Design flow



Below are the supported commands:

Command option	Represented for	**Options**
-synth	Synthesis using yosys	-
-compile	Run pack, place, route and generate fasm file	-
-src <source path>	Source file folder	-
-d <device>	Device supported	ql-eos-s3
-P <package>	Packages	PD64 (BGA), WR42(WLCSP), PU64 (QFN)
-p <pcf file>	Fix Placement constraints of IO's	-
-s <sdc file>	Timing Constraint File (SDC)	Refer online documents section for SDC constraints supported

-r <router flag>	Timing: means no attention is paid to delay. Congestion: means nets on the critical path pay no attention to congestion.	timing, congestion
-t <top module>	Top module of the Verilog design	-
-v <Verilog list files>	Verilog source files	Only Verilog supported
-dump <output to be dumped>	Dump the output format file	Jlink, post_verilog, header

Run design flow on a simple counter design:

Setup environment:

To run any example, perform these steps once.

```
>export INSTALL_DIR="/opt/symbiflow/eos-s3"
# adding symbiflow toolchain binaries to PATH
>export
PATH="$INSTALL_DIR/install/bin:$INSTALL_DIR/install/bin/pyt
hon:$PATH"
>source "$INSTALL_DIR/conda/etc/profile.d/conda.sh"
>conda activate
```

Entering an HDL Design

To enter an HDL design:

1.Write a Verilog code for the design using any text editor.

2.Verify the syntax.

3.Create the simulation stimuli using any text editor.

Below are the design (counter_16bit.v) and testbench(counter_16bit_tb.v)


counter_16bit_tb.v


counter_16bit.v

The code and testbench is present at:
<Install_Path>/install/tests/counter_16bit/

Performing the Pre-Layout Simulation

To perform a pre-layout simulation:

Using Icarus Verilog:

To create the VCD output file that will be used to perform graphical analysis of the

Design, the following lines are added in the TB:

```
     initial begin
          $dumpfile("counter_16bit_tb.vcd");
          $dumpvars(0,counter_16bit_tb);
          $display("\t\ttime,\tclk,\treset,\tenable,\tcount");
          $monitor("%d,\t%b,\t%b,\t%b,\t%d",$time,
clk,reset,enable,count);
     end
```

The "iverilog" and "vvp" commands are the most important commands available to users of Icarus Verilog. The "iverilog" command is the compiler, and the "vvp" command is the simulation runtime engine.

```
> cd <INSTALL_PATH>/install/tests/counter_16bit
```

The "iverilog" command supports multi-file designs by two methods. The simplest is to list the files on the command line:

```
> iverilog -o my_design counter_16bit.v counter_16bit_tb.v
> vvp my_design
```

This command compiles the design, which is spread across two input files, and generates the compiled result into the "my_design" file.

Another technique is to use a commandfile, which lists the input files in a text file. For example, create a text file called "file_list.txt" with the files listed one per line:

```
counter_16bit.v
counter_16bit_tb.v
```

Then compile and execute the design with command:

```
> iverilog -o my_design -c file_list.txt
> vvp my_design
```

VCD file is created, it can be viewed using GTKWave:

```
> gtkwave testbench.vcd &
```

Performing Design Synthesis

To perform a design synthesis:

In SymbiFlow, the synthesis of Verilog files is performed with Yosys. Yosys parses Verilog files, applies basic optimizations, performs technological mapping to FPGA blocks, and generates JSON and EBLIF files for the place and route tool.

Syntax:

```
> ql_symbiflow -synth -src <source complete path> -d
<device> -t <top module name> -v <verilog files> -p <pcf
file> -P <Package file> -s <SDC file>
```

Output files for synthesis are:
<TOP>.eblif : netlist file for the design
<TOP>_synth.log : synthesis log information

```
cd <INSTALL_PATH>/install/tests/counter_16bit
```

and run the below command:

```
> ql_symbiflow -synth -d ql-eos-s3 -t top -v
counter_16bit.v -p counter_16bit.pcf -P PD64
```

Note: All the output log files will be dumped in {source path}/build folder
For pcf related information, please refer pcf sample.
-src command is optional if run from the same directory where source files are present.

Running pack, Place and Route tools

The eblif file generated during the synthesis is used for pack, place and route along with device information, pcf and the sdc file.

Syntax:

```
> ql_symbiflow -compile -src <source complete path> -d
<device> -t <top module name> -v <verilog files> -p <pcf
file> -P <Package file> -s <SDC file>
```

The output files dumped will be:
<TOP>.net : Once packing is complete.
<TOP>.place : Placer file from VPR

<TOP>.route : Router file from VPR

One can refer to the pack.log, placer.log, router.log for more information related to each tool.

```
> cd <INSTALL_PATH>/install/tests/counter_16bit
> ql_symbiflow -compile -src $PWD -d ql-eos-s3 -t top -v
counter_16bit.v -p counter_16bit.pcf -P PD64 -s
counter_16bit.sdc
```

The above command will also run synthesis if it was not run before.

To Generate Various files during compile, use the below options

Common command with just output file change:

```
> ql_symbiflow -compile -src $PWD -d ql-eos-s3 -t top -v
counter_16bit.v -p counter_16bit.pcf -P PD64 -s
counter_16bit.sdc -dump jlink/post_verilog/header
```

To Generate the Post-Layout Verilog file

This is the Verilog file used for the functional simulation to verify the Place and Route output.

Syntax:

```
> ql_symbiflow -compile -src <source complete path> -d
<device> -t <top module name> -v <verilog files> -p <pcf
file> -P <Package file> -s <SDC file> -dump post_verilog
```

The output files dumped will be :
<TOP>_bit.v : Post layout Verilog file.

```
> ql_symbiflow -compile -src $PWD -d ql-eos-s3 -t top -v
counter_16bit.v -p counter_16bit.pcf -P PD64 -s
counter_16bit.sdc -dump post_verilog
```

Performing the Post-Layout Simulation

The testbench is present at:
<Install_Path>/install/tests/counter_16bit/
The post-layout design netlist is present at:
<Install_Path>/install/counter_16bit/top_bit.v
The primitive file library file is present at:
<Install_Path>/install/share/techmaps/quicklogic/techmap/cells_sim.v

To perform a post-layout simulation:

1.Perform a post-layout simulation (without SDF, SDF file not supported) of the Verilog code using your HDL simulator of choice.

2.View the simulation results in the Waveform/ Data Analyzer and verify.

To Generate the Jlink file

JLINK file contains a script that can flash the board with the generated FPGA configuration via the JLink Connector

Syntax:

```
> ql_symbiflow -compile -src <source complete path> -d
<device> -t <top module name> -v <verilog files> -p <pcf
file> -P <Package file> -s <SDC file> -dump jlink
```

The output files dumped will be:
<TOP>.jlink ->jlink file.

```
> ql_symbiflow -compile -src $PWD -d ql-eos-s3 -t top -v
counter_16bit.v -p counter_16bit.pcf -P PD64 -s
counter_16bit.sdc -dump jlink
```

For details on how to configure the FPGA using the top.jlink file, refer to Download Binaries using Jlink SWD section in the QuickFeather_UserGuide.pdf. You can find the document at:

https://github.com/QuickLogic-Corp/quick-feather-dev-board/blob/3b8566c83ed9df56282701710165a9afbb5c5a49/doc/QuickFeather_UserGuide.pdf

To Generate the ASCII header file format

Ascii header file can be generated from the jlink or the .bit file.

Syntax:

```
> ql_symbiflow -compile -src <source complete path> -d
<device> -t <top module name> -v <verilog files> -p <pcf
file> -P <Package file> -s <SDC file> -dump header
```

The output files dumped will be :
<TOP>_jlink.h : file generated from the jlink input
<TOP>_bit.h : file generated from the bit file input

```
> ql_symbiflow -compile -src $PWD -d ql-eos-s3 -t top -v
counter_16bit.v -p counter_16bit.pcf -P PD64 -s
counter_16bit.sdc  -dump header
```

The generated header file can be used in M4 application program to load FPGA

The output files can be dumped for all as:

```
> ql_symbiflow -compile -src $PWD -d ql-eos-s3 -t top -v
counter_16bit.v -p counter_16bit.pcf -P PD64 -s
counter_16bit.sdc  -dump header jlink post_verilog
```

PCF: Sample

For package PD64, the counter_16bit has the below IO placements:
Syntax: set_io <port_name> <Package IO>
set_io clk A3
set_io enable C1
set_io reset A1
set_io count(0) A2
set_io count(1) B2
set_io count(2) C3
set_io count(3) B3
set_io count(4) B1
set_io count(5) C4
set_io count(6) B4
set_io count(7) A4
set_io count(8) C5
set_io count(9) B5
set_io count(10) D6
set_io count(11) A5
set_io count(12) C6
set_io count(13) E7
set_io count(14) D7
set_io count(15) E8

PCF reference file for various the below packages:

The highlighted pins are the clock ports and can also be used as BIDIR IO.

PD64	
IO Location	**IO Type**
B1	BIDIR
C1	BIDIR

PU64	
IO Location	**IO type**
4	BIDIR
5	BIDIR

WR42	
IO Location	**IO Type**
A7	BIDIR
B7	BIDIR

A1	BIDIR
A2	BIDIR
B2	BIDIR
C3	BIDIR
B3	BIDIR
A3	BIDIR/CLOCK
C4	BIDIR/CLOCK
B4	BIDIR
A4	BIDIR
C5	BIDIR
B5	BIDIR
D6	BIDIR
A5	BIDIR
C6	BIDIR
E7	BIDIR
D7	BIDIR
E8	BIDIR
H8	BIDIR
G8	BIDIR
H7	BIDIR
G7	BIDIR/CLOCK
H6	BIDIR/CLOCK
G6	BIDIR/CLOCK
F7	BIDIR
F6	BIDIR
H5	BIDIR
G5	BIDIR
F5	BIDIR
F4	BIDIR
G4	BIDIR
H4	SDIOMUX
E3	SDIOMUX
F3	SDIOMUX
F2	SDIOMUX
H3	SDIOMUX
G2	SDIOMUX
E2	SDIOMUX
H2	SDIOMUX
D2	SDIOMUX
F1	SDIOMUX
H1	SDIOMUX

6	BIDIR
2	BIDIR
3	BIDIR
64	BIDIR
62	BIDIR
63	BIDIR/CLOCK
61	BIDIR/CLOCK
60	BIDIR
59	BIDIR
57	BIDIR
56	BIDIR
55	BIDIR
54	BIDIR
53	BIDIR
40	BIDIR
42	BIDIR
38	BIDIR
36	BIDIR
37	BIDIR
39	BIDIR
34	BIDIR/CLOCK
33	BIDIR/CLOCK
32	BIDIR/CLOCK
31	BIDIR
30	BIDIR
28	BIDIR
27	BIDIR
26	BIDIR
25	BIDIR
23	BIDIR
22	SDIOMUX
21	SDIOMUX
20	SDIOMUX
18	SDIOMUX
17	SDIOMUX
15	SDIOMUX
16	SDIOMUX
11	SDIOMUX
13	SDIOMUX
14	SDIOMUX
10	SDIOMUX

C7	BIDIR
A6	BIDIR
B6	BIDIR/CLOCK
A5	BIDIR
B5	BIDIR
A4	BIDIR
B4	BIDIR
E1	BIDIR
D1	BIDIR
C1	BIDIR
F2	BIDIR
E2	BIDIR/CLOCK
D2	BIDIR/CLOCK
D3	BIDIR
F3	BIDIR
E3	BIDIR
F4	BIDIR
E4	BIDIR
D5	SDIOMUX
F5	SDIOMUX
E6	SDIOMUX
F6	SDIOMUX
D7	SDIOMUX
E7	SDIOMUX
F7	SDIOMUX

D1	SDIOMUX
E1	SDIOMUX
G1	SDIOMUX

7	SDIOMUX
8	SDIOMUX
9	SDIOMUX

Hardware features that are not supported in this release:

1. IO registers:
   - Usage of IO registers available in the IO block (Hardware)

2. RAM Initialization:
   - RAM initialization as part of the FPGA configuration
   - We can initialize the FPGA RAM through wishbone interface using M4 (after FPGA configuration). We need to have the wishbone slave interface in the FPGA IP that we design.
     `M4 -> Wishbone master (in ASSP)-> Wishbone slave (in FPGA IP)-> FPGA RAMs`

3. gclkbuff support:
   - Usage of gclkbuff (clock buffer) in designs

4. Yosys
   - Yosys does not target wider muxes (4-8 input) on Logic cell

Online Documents

Below are the links which you can refer:

SDC constraints information:

- https://docs.verilogtorouting.org/en/latest/vpr/sdc_commands/

VPR flow and the files dumped:

- https://docs.verilogtorouting.org/en/latest/vpr/basic_flow/