QuickLogic-FPGA-Toolchain

Release 1.0.0

Dec 14, 2020

Contents

CHAPTER 1

Symbiflow Installation Guide and Tutorial

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

System Requirements

Require-ments	Linux	CentOS	Ubuntu
Processor	Intel Xeon® or similar proces-sors	Intel Xeon or similar proces-sors	Intel Xeon or similar proces-sors

Ram Size: 2 GB or more Free Hard-Disc space: 5GB or more

1.1 Installing Symbiflow on Linux

Download the required Symbiflow Installer

To install Symbiflow on Linux:

1. Set the execute permission for the .run file

```
chmod 755 Symbiflow*<version>*.gz.run
```

2. Set the *<INSTALL_DIR>:* variable:

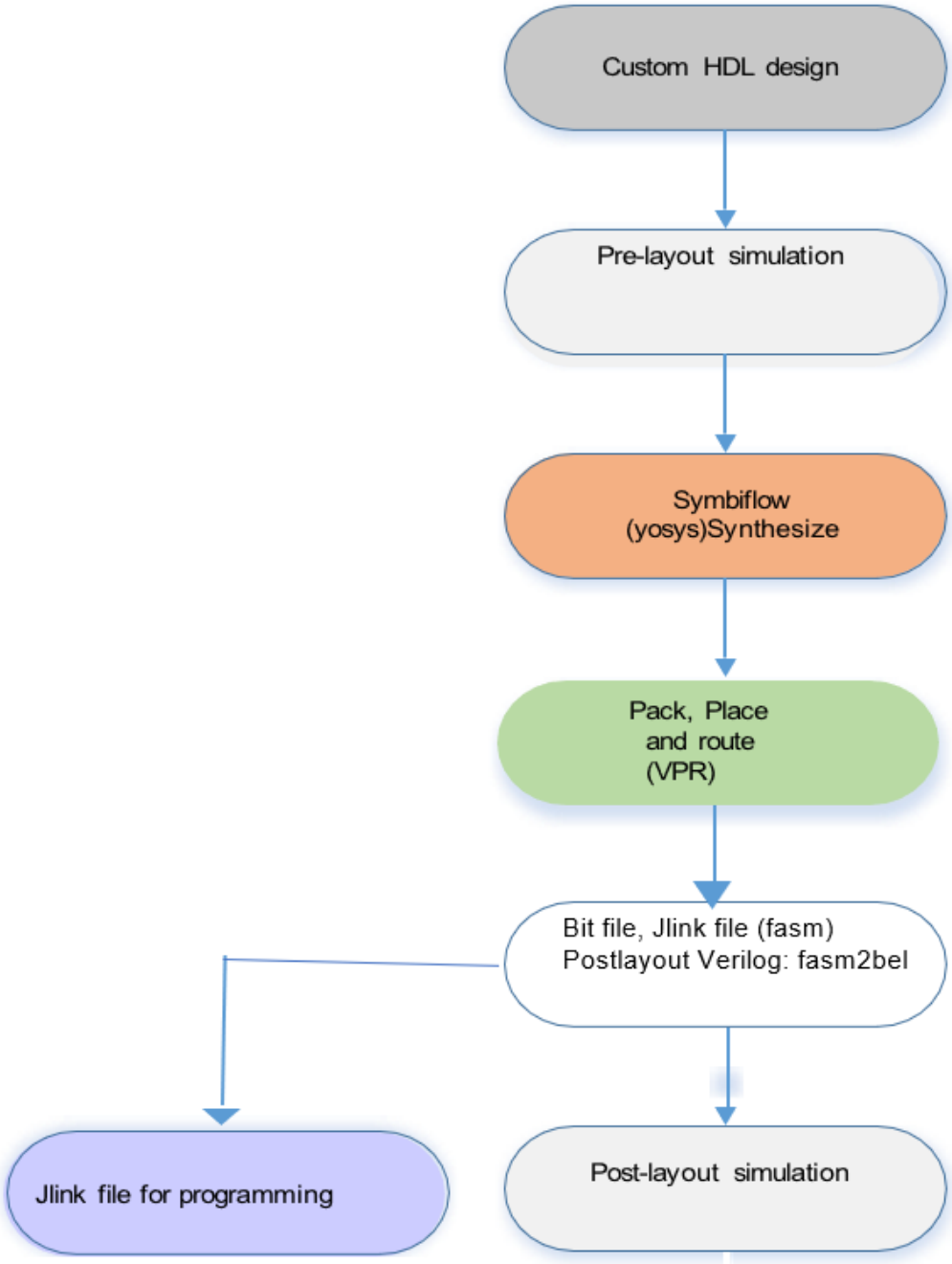
```
export INSTALL_DIR=<Install path>
```

3. Execute the .run file from the terminal:

```
bash Symbiflow<*version*>.gz.run
```

1.2 Symbiflow: Design flow

The below figure shows the complete design flow from the HDL to the programming file:



1.2.1 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

1.3 Run design flow on a simple counter design

Setup environment

To run any example, perform these steps once.

```
export INSTALL_DIR="specify the installpath"
#adding symbiflow toolchain binaries to PATH
export PATH="$INSTALL_DIR/quicklogic-arch-defs/bin:$INSTALL_
↪DIR/quicklogic-arch-defs/bin/python:$PATH"
source "$INSTALL_DIR/conda/etc/profile.d/conda.sh"
conda activate
```

Entering 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.

The code and testbench for the example design are present at: <*Install_Path*>/quicklogic-arch-defs/tests/counter_16bit/

1.3.1 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>/quicklogic-arch-defs/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
```

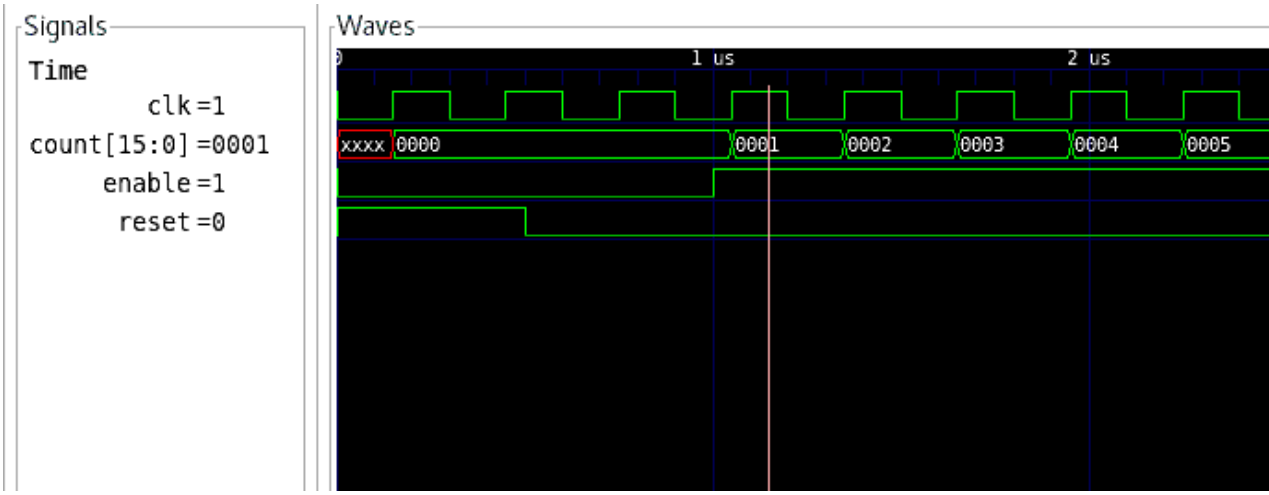
This command compiles the design, which is spread across two input files, and generates the compiled result into the “my_design” file. Another technique is to use a commandfile, which lists the input files in a text file. For example, create a text file called “file_list.txt” with the files listed one per line:

```
#Inside file_list.txt
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 &
```



1.3.2 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>
```

```
cd <INSTALL_PATH>/quicklogic-arch-defs/tests/counter_16bit
```

and run the below command:

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

Output files for synthesis are: <TOP>.eblif : netlist file for the design <TOP>_synth.log : synthesis log information, refer this file for any issues during synthesis

Resource utilization in the top_synth.log of the counter:

Number of wires: 384 Number of wire bits: 384 Number of public wires: 382 Number of public wire bits: 382 Number of memories: 0 Number of memory bits: 0 Number of processes: 0 Number of cells: 67 BIDIR_CELL 18 CLOCK_CELL 1 C_FRAG 4 GMUX_IP 1 Q_FRAG 16 T_FRAG 27

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.

1.3.3 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 s

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>/quicklogic-arch-defs/tests/counter_16bit
ql_symbiflow -compile -src $PWD -d ql-eos-s3 -t top -v counter_16bit.v -
↪p chandalar.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 chandalar.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 (for verifying the configuration bits) top_post_synthesis.v : Post layout Verilog file (for timing simulation) top_post_synthesis.sdf : SDF file (for timing simulation)

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

The Timing analysis refer the files *report_timing.hold.rpt, report_timing.setup.rpt and top.log* inside the build folder

For the counter design below is the timing report from the top.log file:

Hold Worst Negative Slack (hWNS): 0 ns Hold Total Negative Slack (hTNS): 0 ns Setup Worst Negative Slack (sWNS): -35.2295 ns Setup Total Negative Slack (sTNS): -1062.29 ns Final critical path: 35.2295 ns, Fmax: 28.3853 MHz

1.3.4 Performing the Post-Layout Simulation (verifying the configuration bits)

Post layout Simulation using the configuration bits, uses the device configuration bit file *top_bit.v*

The testbench for the counter design is present at: <*Install_Path*>/quicklogic-arch-defs/tests/counter_16bit/

The post-layout design netlist is present at: <*Install_Path*>/quicklogic-arch-defs/counter_16bit/top_bit.v

The primitive file library file is present at: <*Install_Path*>/conda/share/yosys/quicklogic/cells_sim.v

---

Note: cells_sim.v : This file has the definition for predefined macros

---

1.3.5 Performing the Post-Layout Timing Simulation

Post layout Timing simulation uses the SDF(Standard Delay Format) file.

The testbench for the counter design is present at: <*Install_Path*>/quicklogic-arch-defs/tests/counter_16bit/

The post-layout design netlist is present at: <*Install_Path*>/quicklogic-arch-defs/counter_16bit/build/top_post_synthesis.v

The SDF file is present at: <*Install_Path*>/quicklogic-arch-defs/counter_16bit/build/top_post_synthesis.sdf

The primitive file library file is present at: <*Install_Path*>/quicklogic-arch-defs/share/techmaps/quicklogic/techmap/cells_sim.v

The ram primitive file is present at: <*Install_Path*>/quicklogic-arch-defs/share/arch/ql-eos-s3_wlcsp/cells/ram_sim.v

To perform a post-layout simulation: - Perform a post-layout simulation of the Verilog code use iverilog. - View the simulation results in the Waveform/ Data Analyzer and verify.

Note: cells_sim.v : This file has the definition for technology mapped macros ram_sim.v : Has the ram definition

---

1.3.6 Generate the Jlink and openOCD 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 chandalar.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.

openOCD is an on-chip debugger 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 openocd
```

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

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

1.3.7 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 chandalar.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 chandalar.pcf -P PD64 -s counter_16bit.
↪sdc  -dump header jlink post_verilog
```

1.3.8 PCF Sample

The PCF file is for fix placing the IO to a particular IO location on the device. For S3B device we have 3 packages PD64, PU64 and WR42.

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. Either IO Location or Alias name can be used.

PD64		
IO Locatio	Alias	IO Type
B1	IO_0	BIDIR
C1	IO_1	BIDIR
A1	IO_2	BIDIR
A2	IO_3	BIDIR
B2	IO_4	BIDIR
C3	IO_5	BIDIR
B3	IO_6	BIDIR
A3	IO_7	BIDIR/CLOCK
C4	IO_8	BIDIR/CLOCK
B4	IO_9	BIDIR
A4	IO_10	BIDIR
C5	IO_11	BIDIR
B5	IO_12	BIDIR
D6	IO_13	BIDIR
A5	IO_14	BIDIR
C6	IO_15	BIDIR
E7	IO_16	BIDIR
D7	IO_17	BIDIR
E8	IO_18	BIDIR
H8	IO_19	BIDIR
G8	IO_20	BIDIR
H7	IO_21	BIDIR
G7	IO_22	BIDIR/CLOCK
H6	IO_23	BIDIR/CLOCK
G6	IO_24	BIDIR/CLOCK
F7	IO_25	BIDIR
F6	IO_26	BIDIR
H5	IO_27	BIDIR
G5	IO_28	BIDIR
F5	IO_29	BIDIR
F4	IO_30	BIDIR
G4	IO_31	BIDIR
H4	IO_32	SDIOMUX
E3	IO_33	SDIOMUX
F3	IO_34	SDIOMUX
F2	IO_35	SDIOMUX
H3	IO_36	SDIOMUX
G2	IO_37	SDIOMUX
E2	IO_38	SDIOMUX
H2	IO_39	SDIOMUX
D2	IO_40	SDIOMUX
F1	IO_41	SDIOMUX
H1	IO_42	SDIOMUX
D1	IO_43	SDIOMUX
E1	IO_44	SDIOMUX
G1	IO_45	SDIOMUX

PU64		
IO Locatio	Alias	IO type
4	IO_0	BIDIR
5	IO_1	BIDIR
6	IO_2	BIDIR
2	IO_3	BIDIR
3	IO_4	BIDIR
64	IO_5	BIDIR
62	IO_6	BIDIR
63	IO_7	BIDIR/CLOCK
61	IO_8	BIDIR/CLOCK
60	IO_9	BIDIR
59	IO_10	BIDIR
57	IO_11	BIDIR
56	IO_12	BIDIR
55	IO_13	BIDIR
54	IO_14	BIDIR
53	IO_15	BIDIR
40	IO_16	BIDIR
42	IO_17	BIDIR
38	IO_18	BIDIR
36	IO_19	BIDIR
37	IO_20	BIDIR
39	IO_21	BIDIR
34	IO_22	BIDIR/CLOCK
33	IO_23	BIDIR/CLOCK
32	IO_24	BIDIR/CLOCK
31	IO_25	BIDIR
30	IO_26	BIDIR
28	IO_27	BIDIR
27	IO_28	BIDIR
26	IO_29	BIDIR
25	IO_30	BIDIR
23	IO_31	BIDIR
22	IO_32	SDIOMUX
21	IO_33	SDIOMUX
20	IO_34	SDIOMUX
18	IO_35	SDIOMUX
17	IO_36	SDIOMUX
15	IO_37	SDIOMUX
16	IO_38	SDIOMUX
11	IO_39	SDIOMUX
13	IO_40	SDIOMUX
14	IO_41	SDIOMUX
10	IO_42	SDIOMUX
7	IO_43	SDIOMUX
8	IO_44	SDIOMUX
9	IO_45	SDIOMUX

WR42		
IO Locatio	Alias	IO Type
A7	IO_0	BIDIR
B7	IO_1	BIDIR
C7	IO_3	BIDIR
A6	IO_6	BIDIR
B6	IO_8	BIDIR/CLOCK
A5	IO_9	BIDIR
B5	IO_10	BIDIR
A4	IO_14	BIDIR
B4	IO_15	BIDIR
E1	IO_16	BIDIR
D1	IO_17	BIDIR
C1	IO_19	BIDIR
F2	IO_20	BIDIR
E2	IO_23	BIDIR/CLOCK
D2	IO_24	BIDIR/CLOCK
D3	IO_25	BIDIR
F3	IO_28	BIDIR
E3	IO_29	BIDIR
F4	IO_30	BIDIR
E4	IO_31	BIDIR
D5	IO_34	SDIOMUX
F5	IO_36	SDIOMUX
E6	IO_38	SDIOMUX
F6	IO_39	SDIOMUX
D7	IO_43	SDIOMUX
E7	IO_44	SDIOMUX
F7	IO_45	SDIOMUX

1.4 Hardware Limitations and Online References

Limitations:

- RAM/FIFO: Only symmetrical combinations of RAM are supported

References: SDC constraints information VPR flow and the files info

CHAPTER 2

S3B Device

This provides the details of in-built RAMs/FIFOs and Multipliers in S3B and their usage in the user design.

2.1 S3B Device

Hardmacro Resources

- RAMs - 8 blocks of 8K bits
- FIFO - 8 in-build FIFO controllers - Can be configured into FIFOs using above RAM blocks
- Multipliers - 2 (32X32) in-built multipliers

2.1.1 RAM Features

- 8Kbits/RAM
- X8, x16 and x32 data bus width
- Independent programmability of read and write data bus widths
- Asynchronous clocks
- Supports 2 RAM block vertical or horizontal concatenation((i.e. 16Kbits RAM combining 2 8Kbits RAM Blocks)
- Supports clock disabling during idle operation

2.1.2 FIFO Features

- X8, x16 and x32 data bus width
- Configurable Synchronous/Asynchronous operation

- Supports 2 RAM block vertical or horizontal concatenation((i.e. 16Kbits RAM combining 2 - 8Kbits RAM Blocks)
- Independent programmability of data bus width on PUSH and POP side
- Switchable clock domain between PUSH and POP side during asynchronous operation
- Supports clock disabling during idle operation
- Asynchronous Reset (aside from the synchronous FLUSH) going to the pointers

2.1.3 Multiplier Features

- 2 (32-bit) Multiplier
- Can be configured as 4 16-bit multipliers
- Signed multiplier
- Supports Latched input

2.1.4 Macro Usage and examples

RAM Usage

The RAM macros (RAM_8K_BLK.v & RAM_16K_BLK.v) are part of pp3_cells_sim.v at: <*Install_Path*>/conda/share/yosys/quicklogic

The examples of RAMs are present at: <*Install_Path*>/quicklogic-arch-defs/tests/RAM_Examples

The Design example using S3 SRAM block are present at: <*Install_Path*>/quicklogic-arch-defs/tests/ram_test

1. RAM_8K_BLK RAM macro is 8K bits block which can be configured as:
   - 1 independent 8Kbits RAM block
   - RAM can be initialized through INIT or INIT_FILE
2. RAM_16K_BLK RAM macro is a 16K bits block which can be configured as:
   - A horizontal concatenated block (16Kbits)
   - A vertical concatenated block (16Kbits)
   - RAM can be initialized through INIT or INIT_FILE

Example 1: 1 8Kbits RAM

```
module r512x16_512x16 (WA,RA,WD,WClk,RClk,WClk_En,RClk_En,WEN,RD);
    input [8:0] WA;
    input [8:0] RA;
    input WClk,RClk;
    input WClk_En,RClk_En;
    input [1:0] WEN;
    input [15:0] WD;
    output [15:0] RD;

    parameter [16383:0] INIT = 16384'b0;
    parameter INIT_FILE="init_512x16.hex";

    parameter addr_int = 9 ;
```

(continues on next page)

(continued from previous page)

```
    parameter data_depth_int = 512;
    parameter data_width_int = 16;
    parameter wr_enable_int = 2;
    parameter reg_rd_int = 0;

    RAM_8K_BLK #(.addr_int(addr_int),.data_depth_int(data_depth_int),.data_width_
↪int(data_width_int),.wr_enable_int(wr_enable_int),.reg_rd_int(reg_rd_int),
                                .INIT(INIT),.INIT_FILE(INIT_FILE)
                                )
    RAM_INST (      .WA(WA), .RA(RA), .WD(WD), .WClk(WClk), .RClk(RClk), .WClk_
↪En(WClk_En), .RClk_En(RClk_En), .WEN(WEN), .RD(RD));
endmodule
```

Example 2: Horizontal Concatenation

- 512x32 RAM (16 Kbits)
- RAM can be initialized through INIT or INIT_FILE

```
module r512x32_512x32 (WA,RA,WD,WClk,RClk,WClk_En,RClk_En,WEN,RD);
    input [8:0] WA;
    input [8:0] RA;
    input WClk,RClk;
    input WClk_En,RClk_En;
    input [3:0] WEN;
    input [31:0] WD;
    output [31:0] RD;

    parameter [16383:0] INIT = 16384'b0;
    parameter INIT_FILE="init_512x32.hex";

    parameter addr_int = 9 ;
    parameter data_depth_int = 512;
    parameter data_width_int = 32;
    parameter wr_enable_int = 4;
    parameter reg_rd_int = 0;

    RAM_16K_BLK #(.addr_int(addr_int),.data_depth_int(data_depth_int),.data_width_
↪int(data_width_int),.wr_enable_int(wr_enable_int),.reg_rd_int(reg_rd_int),
                                .INIT(INIT),.INIT_FILE(INIT_FILE)
                                )
    RAM_INST (      .WA(WA), .RA(RA), .WD(WD), .WClk(WClk), .RClk(RClk), .WClk_
↪En(WClk_En), .RClk_En(RClk_En), .WEN(WEN), .RD(RD));
endmodule
```

Example 3: Vertical Concatenation

- 2048 x 8 RAM (16 Kbits)
- RAM can be initialized through INIT or INIT_FILE

```
module r2048x8_2048x8 (WA,RA,WD,WClk,RClk,WClk_En,RClk_En,WEN,RD);
    input [10:0] WA;
    input [10:0] RA;
    input WClk,RClk;
    input WClk_En,RClk_En;
    input WEN;
    input [7:0] WD;
```

(continues on next page)

(continued from previous page)

```
    output [7:0] RD;

    parameter [16383:0] INIT = 16384'b0;
    parameter INIT_FILE="init_2048x8.hex";

    parameter addr_int = 11 ;
    parameter data_depth_int = 2048;
    parameter data_width_int = 8;
    parameter wr_enable_int = 1;
    parameter reg_rd_int = 0;


    RAM_16K_BLK #(.addr_int(addr_int),.data_depth_int(data_depth_int),.data_width_
↪int(data_width_int),.wr_enable_int(wr_enable_int),.reg_rd_int(reg_rd_int),
                          .INIT(INIT),.INIT_FILE(INIT_FILE)
                                )
    RAM_INST (      .WA(WA), .RA(RA), .WD(WD), .WClk(WClk), .RClk(RClk), .WClk_
↪En(WClk_En), .RClk_En(RClk_En), .WEN(WEN), .RD(RD));
endmodule
```

FIFO Usage

The FIFO macros (FIFO_8K_BLK.v & FIFO_16K_BLK.v) are part of pp3_cells_sim.v at: <*Install_Path*>/conda/share/yosys/quicklogic

The examples of FIFOs are present at: <*Install_Path*>/quicklogic-arch-defs/tests/FIFO_Examples

The Design example using S3 FIFO block are present at: <*Install_Path*>/quicklogic-arch-defs/tests/fifo_test

Example 1: 1 8Kbits FIFO

- FIFO 512 x 16
- Asynchronous FIFO

```
module af512x16_512x16 (DIN,Fifo_Push_Flush,Fifo_Pop_Flush,PUSH,POP,Push_Clk,Pop_Clk,
↪Push_Clk_En,Pop_Clk_En,Fifo_Dir,Async_Flush,Almost_Full,Almost_Empty,PUSH_FLAG,POP_
↪FLAG,DOUT);
    input Fifo_Push_Flush,Fifo_Pop_Flush;
    input Push_Clk,Pop_Clk;
    input PUSH,POP;
    input [15:0] DIN;
    input Push_Clk_En,Pop_Clk_En,Fifo_Dir,Async_Flush;
    output [15:0] DOUT;
    output [3:0] PUSH_FLAG,POP_FLAG;
    output Almost_Full,Almost_Empty;

    parameter data_depth_int = 512;
    parameter data_width_int = 16;
    parameter reg_rd_int = 0;
    parameter sync_fifo_int = 0;

    FIFO_8K_BLK  # (.data_depth_int(data_depth_int),.data_width_int(data_width_int),.
↪reg_rd_int(reg_rd_int),.sync_fifo_int(sync_fifo_int)
            )
      FIFO_INST  ( .DIN(DIN), .PUSH(PUSH), .POP(POP), .Fifo_Push_Flush(Fifo_Push_
↪Flush), .Fifo_Pop_Flush(Fifo_Pop_Flush), .Push_Clk(Push_Clk),
```

(continues on next page)

(continued from previous page)

```
            .Pop_Clk(Pop_Clk),.PUSH_FLAG(PUSH_FLAG), .POP_FLAG(POP_FLAG), .Push_Clk_
↪En(Push_Clk_En), .Pop_Clk_En(Pop_Clk_En),
            .Fifo_Dir(Fifo_Dir),  .Async_Flush(Async_Flush), .Almost_Full(Almost_
↪Full), .Almost_Empty(Almost_Empty), .DOUT(DOUT));
endmodule
```

Example 2: Horizontal Concatenation

- FIFO 512 x 32
- Asynchronous FIFO

```
module af512x32_512x32 (DIN,Fifo_Push_Flush,Fifo_Pop_Flush,PUSH,POP,Push_Clk,Pop_Clk,
↪Push_Clk_En,Pop_Clk_En,Fifo_Dir,Async_Flush,Almost_Full,Almost_Empty,PUSH_FLAG,POP_
↪FLAG,DOUT);
    input Fifo_Push_Flush,Fifo_Pop_Flush;
    input Push_Clk,Pop_Clk;
    input PUSH,POP;
    input [31:0] DIN;
    input Push_Clk_En,Pop_Clk_En,Fifo_Dir,Async_Flush;
    output [31:0] DOUT;
    output [3:0] PUSH_FLAG,POP_FLAG;
    output Almost_Full,Almost_Empty;

    parameter data_depth_int = 512;
    parameter data_width_int = 32;
    parameter reg_rd_int = 0;
    parameter sync_fifo_int = 0;

    FIFO_16K_BLK  # (.data_depth_int(data_depth_int),.data_width_int(data_width_int),.
↪reg_rd_int(reg_rd_int),.sync_fifo_int(sync_fifo_int)
                                    )
      FIFO_INST  ( .DIN(DIN), .PUSH(PUSH), .POP(POP), .Fifo_Push_Flush(Fifo_Push_
↪Flush), .Fifo_Pop_Flush(Fifo_Pop_Flush), .Push_Clk(Push_Clk),
            .Pop_Clk(Pop_Clk), .PUSH_FLAG(PUSH_FLAG), .POP_FLAG(POP_FLAG), .Push_Clk_
↪En(Push_Clk_En), .Pop_Clk_En(Pop_Clk_En), .Fifo_Dir(Fifo_Dir),
            .Async_Flush(Async_Flush), .Almost_Full(Almost_Full), .Almost_
↪Empty(Almost_Empty), .DOUT(DOUT));

endmodule
```

Example 3: Vertical Concatenation

- FIFO 1024 x 16
- Synchronous FIFO

```
module f1024x16_1024x16 (DIN,Fifo_Push_Flush,Fifo_Pop_Flush,PUSH,POP,Clk,Clk_En,Fifo_
↪Dir,Async_Flush,Almost_Full,Almost_Empty,PUSH_FLAG,POP_FLAG,DOUT);
    input Fifo_Push_Flush,Fifo_Pop_Flush;
    input Clk;
    input PUSH,POP;
    input [15:0] DIN;
    input Clk_En,Fifo_Dir,Async_Flush;
    output [15:0] DOUT;
    output [3:0] PUSH_FLAG,POP_FLAG;
    output Almost_Full,Almost_Empty;
```

(continues on next page)

(continued from previous page)

```
    parameter data_depth_int = 1024;
    parameter data_width_int = 16;
    parameter reg_rd_int = 0;
    parameter sync_fifo_int = 1;

    FIFO_16K_BLK  # (.data_depth_int(data_depth_int),.data_width_int(data_width_int),.
↪reg_rd_int(reg_rd_int),.sync_fifo_int(sync_fifo_int)
                                    )
      FIFO_INST (.DIN(DIN), .PUSH(PUSH), .POP(POP), .Fifo_Push_Flush(Fifo_Push_Flush),
↪ .Fifo_Pop_Flush(Fifo_Pop_Flush), .Push_Clk(Clk),
            .Pop_Clk(Clk), .PUSH_FLAG(PUSH_FLAG), .POP_FLAG(POP_FLAG), .Push_Clk_
↪En(Clk_En), .Pop_Clk_En(Clk_En), .Fifo_Dir(Fifo_Dir),
            .Async_Flush(Async_Flush), .Almost_Full(Almost_Full), .Almost_
↪Empty(Almost_Empty), .DOUT(DOUT));
endmodule
```

Multiplier Usage

The Multiplier macros are present at: <*Install_Path*>/conda/share/yosys/quicklogic

1. 32 x 32 Multiplier

- Configured as 1 32x32 multiplier

```
module MULT_32BIT (Amult, Bmult, Valid_mult, Cmult);
    input [31:0] Amult;
    input [31:0] Bmult;
    input  Valid_mult;
    output [63:0] Cmult;

    wire [1:0] valit_int;

    assign valit_int = {Valid_mult,Valid_mult};

    //qlal4s3_mult_cell_macro
    qlal4s3_mult_cell_macro u_qlal4s3_mult_cell_macro (.Amult(Amult), .Bmult(Bmult), .
↪Valid_mult(valit_int), .sel_mul_32x32(1'b1), .Cmult(Cmult));
endmodule
```

2. 16 x 16 Multiplier

- Configured as 1 16x16 multiplier

```
module MULT_16BIT_X2 ( Amult1, Bmult1, Valid_mult1, Cmult1,
                       Amult2, Bmult2, Valid_mult2, Cmult2 );
    input [15:0] Amult1;
    input [15:0] Bmult1;
    input Valid_mult1;
    output [31:0] Cmult1;

    input [15:0] Amult2;
    input [15:0] Bmult2;
    input Valid_mult2;
    output [31:0] Cmult2;

    wire [31:0] amult_int;
```

(continues on next page)

(continued from previous page)

```
    wire [31:0] bmult_int;
    wire [63:0] cmult_int;
    wire [1:0] valit_int;

    assign valit_int = {Valid_mult2,Valid_mult1};
    assign amult_int = {Amult2,Amult1};
    assign bmult_int = {Bmult2,Bmult1};
    assign Cmult1 = cmult_int[31:0];
    assign Cmult2 = cmult_int[63:32];

    //qlal4s3_mult_cell_macro
    qlal4s3_mult_cell_macro u_qlal4s3_mult_cell_macro (.Amult(amult_int), .
↪Bmult(bmult_int), .Valid_mult(valit_int), .sel_mul_32x32(1'b0), .Cmult(cmult_int));
endmodule
```

Design example Using SRAMs

The Design example using S3 SRAM block are present at: <*Install_Path*>/quicklogic-arch-defs/tests/ram_test

Address Map:

Register	Description	Reset Value	Remarks
0x40020000	FPGA IP ID	0x56A37E57	Read only
0x40020004	Revision Number	0x100	Read only (Rev. 1.00)
0x40020008	GPIO Input	0x0	GPIO_IN [7:0], Input to FPGA IP from External PAD
0x4002000C	GPIO Output	0x0	GPIO_OUT [7:0], Output from FPGA IP to External PAD
0x40020010	GPIO Direction Control	0x0	GPIO_OE [7:0] 0 – Tri-State Output 1 – Drive Output
0x40022000 – 0x400227FC	RAM 0 Access port (512x16_512x16)		RAM Input and Output Port (Write and Read from the Wishbone Interface)
0x40024000 – 0x40024FFC	RAM 1 Access port (1024x16_1024x16)		RAM Input and Output Port (Write and Read from the Wishbone Interface)
0x40026000 – 0x40026FFC	RAM 2 Access port (1024x8_1024x8)		RAM Input and Output Port (Write and Read from the Wishbone Interface)
0x40028000 – 0x400287FC	RAM 3 Access port (512x32_512x32)		RAM Input and Output Port (Write and Read from the Wishbone Interface)
0x4002a000 – 0x4002bFFC	RAM 4 Access port (2048x8_2048x8)		RAM Input and Output Port (Write and Read from the Wishbone Interface)
0x40028000 – 0x4003FFFC	Not Used	0x0	

RAM Initialization files are at: <*Install_Path*>/quicklogic-arch-defs/tests/ram_test Ex. init_2048x8.hex, init_512x32.hex, init_1024x8.hex

Design example Using FIFOs

The Design example using S3 FIFO block are present at: <*Install_Path*>/quicklogic-arch-defs/tests/fifo_test

Address Map:

Register	Description	Reset Value	Remarks
0x40020000	FPGA IP ID	0xF1F07E57	Read only
0x40020004	Revision Number	0x100	Read only (Rev. 1.00)
0x40020008	GPIO Input	0x0	GPIO_IN [7:0], Input to FPGA IP from External PAD
0x4002000C	GPIO Output	0x0	GPIO_OUT [7:0], Output from FPGA IP to External PAD
0x40020010	GPIO Direction Control	0x0	GPIO_OE [7:0] 0 – Tri-State Output 1 – Drive Output
0x40020100	FIFO 1 Access port (af512x16_512x16)		Asynchronous FIFO FIFO Push & Pop port
0x40020104	FIFO 1 Flags		bit [31:16] – Reserved **bit [15] - Almost_Empty** bit [14:12] – Reserved **bit [11:8] – POP Flag** **bit [7] - Almost_Full** bit [6:4] – Reserved **bit [3:0] – PUSH Flag**
0x40020200	FIFO 2 Access port (f1024x16_1024x16)		Synchronous FIFO FIFO Push & Pop port
0x40020204	FIFO 2 Flags		bit [31:16] – Reserved **bit [15] - Almost_Empty** bit [14:12] – Reserved **bit [11:8] – POP Flag** **bit [7] - Almost_Full** bit [6:4] – Reserved **bit [3:0] – PUSH Flag**
0x40020400	FIFO 3 Access port (af512x32_512x32)		Asynchronous FIFO FIFO Push & Pop port
0x40020404	FIFO 3 Flags		bit [31:16] – Reserved **bit [15] - Almost_Empty** bit [14:12] – Reserved **bit [11:8] – POP Flag** **bit [7] - Almost_Full** bit [6:4] – Reserved **bit [3:0] – PUSH Flag**
0x40020800 – 0x4003FFFC	Not Used	0x0	

Push Flag Description:

Value	Status
0000	Full
0001	Empty
0010	Room for more than one-half
0011	Room for more than one-fourth
0100	Room for less than one-fourth full to 64
1010	Room for 32 to 63
1011	Room for 16 to 31
1100	Room for 8 to 15
1101	Room for 4 to 7
1110	Room for at least 2
1111	Room for at least 1
Others	Reserved

Pop Flag Description:

Value	Status
0000	Empty
0001	1 entry in FIFO
0010	At least 2 entries in FIFO
0011	At least 4 entries in FIFO
0100	At least 8 entries in FIFO
0101	At least 16 entries in FIFO
0110	At least 32 entries in FIFO
1000	Less than one-fourth to 64 full
1101	One-fourth or more full
1110	One-half or more full
1111	Full
Others	Reserved

- genindex
- search

Index