Micro Benchmark Documentation

Release 0.0.268

Xifan Tang

May 14, 2024

DATASHEET

CHAPTER

ONE

SIMPLE REGISTERS

1.1 Blinking

1.1.1 Introduction

This benchmark is designed to test the flip-flop/registers in FPGAs. It is a 1-bit clock divider, which outputs half frequency of the input clock.

1.1.2 Source codes

See details in `simple_registers/blinking`

1.1.3 Block Diagram

Fig. 1.1: Blinking schematic

1.1.4 Performance

Expect to consume only 1 LUT and 1 flip-flop of an FPGA. It can reflect the maximum speed of an FPGA between a LUT and a flip-flop.

Warning: The following resource utilization is just an estimation! Different tools in different versions may result differently.

Table 1.1: Estimated resource Utilization

Tool/Resource	Inputs	Outputs	LUT4	FF	Carry	DSP	BRAM
General	0	1	1	1	0	0	0

1.2 Clock Divider

1.2.1 Introduction

This benchmark is designed to test the flip-flop/registers in FPGAs. The benchmark takes an input clock signal clk_i and generates an output clock signal clk_o with one fourth the frequency of the input clock. The module also has a reset input rst to reset output signals.

1.2.2 Source codes

See details in `simple_registers/clk_divider`

1.2.3 Block Diagram

Fig. 1.2: Clock divider schematic

1.2.4 Performance

Expect to consume only 2 LUT and 2 flip-flop of an FPGA. It can reflect the maximum speed of an FPGA between a LUT and a flip-flop.

Warning: The following resource utilization is just an estimation! Different tools in different versions may result differently.

Table 1.2: Estimated resource Utilization

Tool/Resource	Inputs	Outputs	LUT5	FF	Carry	DSP	BRAM
General	0	1	2	2	0	0	0

1.3 PWM Generaotr

1.3.1 Introduction

This benchmark is designed to test the flip-flop/registers in FPGAs. This code generates a PWM signal using a clock divider to control the frequency of the PWM. The clock divider is implemented using a counter that counts up to the value of the DIVISORparameter and resets to 0. The PWM signal is generated using another counter that counts up tothe PERIOD and DUTY_CYCLE parameters and sets the pwm output signal accordingly.

1.3.2 Source codes

See details in `simple_registers/pwm_generator`

1.3.3 Block Diagram

Fig. 1.3: PWM Generator schematic

1.3.4 Performance

Expect to consume only 50 LUT and 33 flip-flop of an FPGA. It can reflect the maximum speed of an FPGA between a LUT and a flip-flop.

Warning: The following resource utilization is just an estimation! Different tools in different versions may result differently.

Table 1.3: Estimated resource Utilization

Tool/Resource	Inputs	Outputs	LUT5	FF	Carry	DSP	BRAM
General	0	1	50	33	0	0	0

CHAPTER

TWO

FINITE STATE MACHINES

2.1 Scalable Sequence Detector

Warning: This benchmark may have some modification/addition of features in future.

2.1.1 Introduction

This benchmark is to detect any sequence of length equal to 2^STATE_BITS. This is a finite state machine based on moore model. All output signals/msgs depends only on current state of machine. The design is scalable, STATE_BITS defines the number of states, the hardware shall be generated based on STATE_BITS. Also, the sequence needs to be passed to DUT as a bus signal, while the x inputs takes sequence bit by bit for detection and msgs are shown on output signals.

2.1.2 Source codes

See details in `fsm/scalable_seq_detector`

2.1.3 Block Diagram / Schematic

Fig. 2.1: Scalable Sequence Detector schematic

2.1.4 Performance

Warning: The following resource utilization is just an estimation! Different tools in different versions may result differently.

Table 2.1: Estimated resource Utilization

Tool/Resource	Inputs	Outputs	LUT4	FF	Carry	DSP	BRAM
General	10	37	28	3	0	0	0

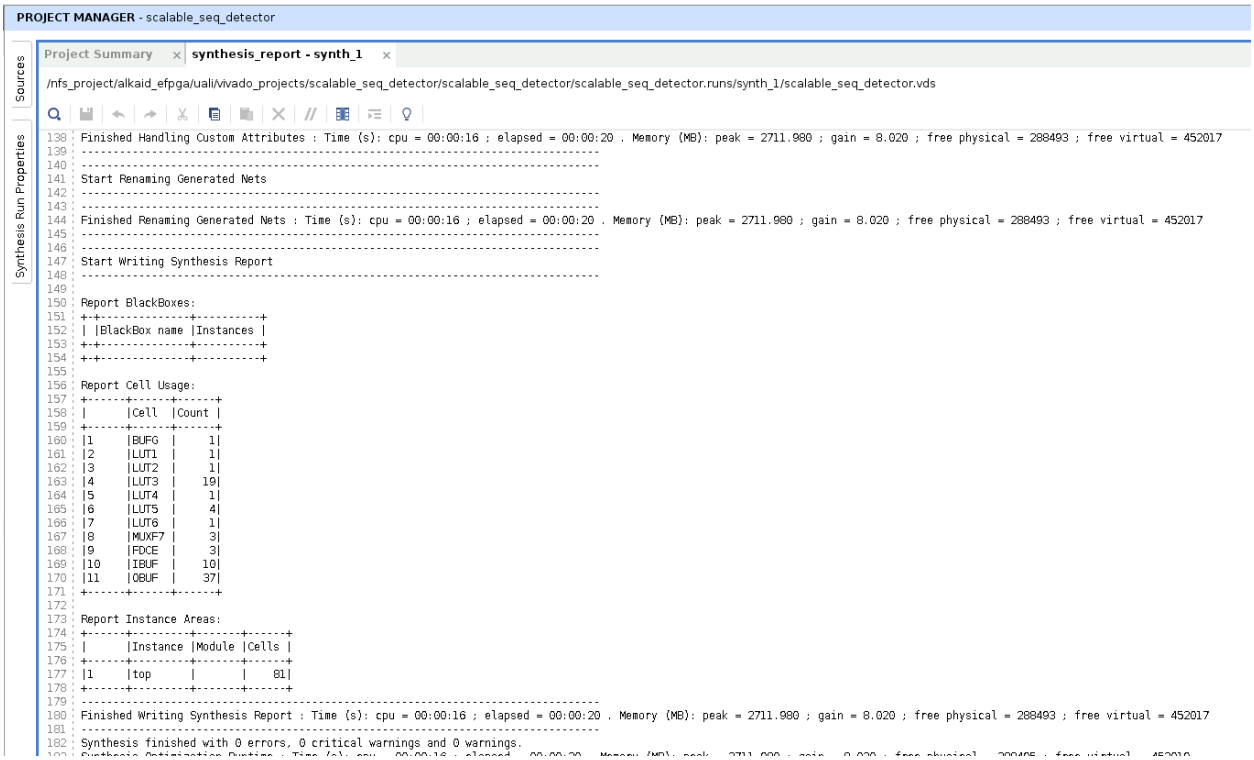

Fig. 2.2: Scalable Sequence Detector performance report using Xilinx Vivado

CHAPTER

THREE

NAMING CONVENTION

3.1 Counter Design Names

We recommend developers to follow the naming convention when adding any counter designs

```
counter[down]<size>_[async|sync]_[set|reset|setb|resetb]
```

down

represent a counting down counter

size

size is an integer, indicating the number of bits for a counter

[async|sync]

represent the feature of reset and set signal

[setp|resetp|setn|resetn]

indicates the existence of reset/set signal as well as polarity. In particular, suffix `p` denotes active-high signals while suffix `n` denotes active-low signals

For instance,

```
counterdown8_async_resetn
```

shows a counter with the following features:

- counting down
- 8-bit in width
- Asynchronous active-low reset

3.2 Pin Names

Note: Please use lowercase as much as you can

For code readability, the pin name should follow the convention

```
<Pin_Name>_<Polarity><Direction>
```

Pin_Name

Represents the pin name

Polarity

Represents polarity of the pin, it can be

- n denotes a negative-enable (active_low) signal

> **Note:** When not specified, by default we assume this is a postive-enable (active-high) signal

Direction

Represents the direction of a pin, it can be

- i denotes an input signal
- o denotes an output signal

A quick example

```
clk_ni
```

represents an input clock signal which is negative-enable

Another example

```
q_no
```

represents an output Q signal which is negative to the input

CHAPTER

FOUR

CONTRIBUTOR GUIDELINES

4.1 Motivation

Github projects involve many parties with different interests. It is necessary to establish rules to

- guarantee the quality of each pull request by establishing a standard
- code review for each pull request is straightforward
- contributors have confidence when submitting changes

4.2 Create Pull requests

- Contributors should state clearly their motivation and the principles of code changes in each pull request
- Contributors should be active in resolving conflicts with other contributors as well as maintainers. In principle, all the maintainers want every pull request in and are looking for reasons to approve it.
- Each pull request should pass all the existing tests in CI (See *Check-in System* for details). Otherwise, it should not be merged unless you get a waiver from all the maintainers.
- Contributors should not modify any codes/tests which are unrelated to the scope of their pull requests.
- The size of each pull request should be small. Large pull request takes weeks to be merged. The recommend size of pull request is up to 500 lines of codes changes. If you have one large file, this can be waived. However, the number of files to be changed should be as small as possible.

Note: For large pull requests, it is strongly recommended that contributors should talk to maintainers first or create an issue on the Github. Contributors should clearly define the motivation, detailed technical plan as well as deliverables. Through discussions, the technical plan may be requested to change. Please do not start code changes blindly before the technical plan is approved.

- For any new feature/functionality to be added, there should be
  - Dedicated test cases in CI which validates its correctness
  - An update on the documentation, if it changes user interface
  - Provide sufficient code comments to ease the maintenance

4.3 Check-in System

See also:

The check-in system is based on continous integration (CI). See details in *Continous Integration*

The check-in system aims to offer a standardized way to

- ensure quailty of each contribution
- resolve conflicts between teams

It is designed for efficient communication between teams.

4.4 Add Benchmarks

FPGA requires a set of benchmark suites to validate its correctness before tape-out. When add a new benchmark to the project, the following steps have to followed.

4.4.1 Choose a benchmark suite

Benchmarks are catorized into different suites, each of which are designed to validate a specific architecture enhancement of the FPGA. For example, `dsp` are designed to validate DSP blocks in the FPGA architecture. When adding a new benchmark, developer should propose to maintainers which category it should belong to. Once agreed, the benchmark can be added to the dediciated directory, e.g., `dsp/<benchmark_suite_name>`

Note: If your benchmark is out of any existing category, you may create a new category. But you should discuss with maintainer first.

4.4.2 Required files

A benchmark should include the following files, so that it can be integrated to the design verification system

- HDL files (`.v`)
  - You may add multiple HDL files for a complex benchmark
  - Please name the top-level module to be the same as the name of benchmark.
- Cocotb testbenches (`.py`).
  - See details in cocotb documentation
  - Please name the testbench file as `test_<benchmark_name>.py`
- Cocotb Makefile (`Makefile`).
  - This is used to run cocotb simulation using iVerilog or other simulators

---

Note: All the files should be placed or linked in a dedicated directory, e.g., `benchmarks/<benchmark_suite_name>`

4.4.3 Update workflow

- If you are adding a benchmark to an existing category, you need to update the list.

See an example under `utils/tasks/simple_gates_rtl_list.yaml`

Note: For file format of the list file, please see *RTL List File*

- If you are creating a new category for benchmark, you need to update workflows by adding the benchmark suite to configuration matrix.

See example

4.4.4 Update documentation

Each benchmark should be properly documented using the existing template. Documentation should covers the following factors

- An brief introduction about the benchmark, presenting the motivation and principles.
- Point to where the source code is located in the repository. Show guidelines about how to compile source codes if special rules are applied.
- A block diagram or a gate-level illustration, depending on the complexity of the design
- Performance prediction. Prefer to show resource utilization for at least one existing FPGA.

See example

CHAPTER

FIVE

FILE FORMATS

5.1 RTL List File

The RTL list file is in yaml format, where users can define one or multiple RTL projects.

A quick example for including a RTL project:

```
<rtl_project_name>:
  source:
    - <rtl_source_0>
    - <rtl_source_1>
    - <rtl_source_2>
  top_module: <top-level_module name>
  cocotb_dir: <directory to the cocotb testbench for this project>
```

Detailed syntax are as follows:

rtl_project_name

Specify the name of this RTL project. This is the unique identifier for the project.

source

You can define a number of rtl source files under this node. Please include the relative path to each source file, based on the project home. For example, `simple_gates/and2/and2.v`

top_module

Specify the name of top-level module among all the source files

cocotb_dir

Specify the directory to the cocotb testbench for this project. For example, `simple_gates/and2`. If not specified, cocotb tests will not be run on this project

> **Note:** Do not include the cocotb python script in the path.

CHAPTER

SIX

CONTINOUS INTEGRATION

6.1 Motivation

Continous Integration (CI) systems are built to ensure that input and output files of each teams are

- Correct
- Reproducable
- Consistent with other teams

CI system is automatically triggered on

- Main branch: the master branch of the codebase
- A pull request on main branch

6.2 Workflows

TBD

CHAPTER

SEVEN

VERSION NUMBER

7.1 Convention

This project follows the semantic versioning, where the version number is in the form of

```
[Major].[Minor].[Patch]
```

For example, version `1.2.300` denotes

- One major milestone is achieved.
- Two minor milestone is achieved after the major revision `1.0.0`
- `300` patches has been applied after the minor revision `1.2.0`

7.2 Version Update Rules

> **Warning:** Please discuss with maintainers before modifying major and minor numbers.

> **Warning:** Please do not modify patch number manually.

To update the version number, please follow the rules:

- Major and minor version number are defined by maintainers
- Patch number is automatically updated through github actions. See detailed in the workflow file

Version updates are made in the following scenario

- When a minor milestone is achieved, the minor revision number can be increased by 1. The following condition is considered as a minor milestone: - a new feature has been developed. - a critical patch has been applied. - a sufficient number of small patches has been applied in the past quarter. In other words, the minor revision will be updated by the end of each quarter as long as there are patches.
- When several minor milestones are achieved, the major revision number can be increased by 1. The following condition is considered as a major milestone: - significant improvements on Quality-of-Results (QoR). - significant changes on user interface. - a techical feature is developed and validated by the community, which can impact the complete design flow.

CHAPTER

EIGHT

API

8.1 Makefile System

To keep EDA flows simple, all the design flows are called through Makefiles and python scripts.

8.1.1 Principles

Makefiles exist in either top-level or lower-level directories, each of which may contain multiple build targets.

- The build targets in top-level makefile are most frequently used design flows across multiple domains, e.g., generate bitstreams
- The build targets in low-level makefile are frequently used design flows within a specific domain, e.g., run HDL simulations.

When call a makefile, please follow the convention

` make <build_target_name> <variables> `

8.1.2 Variables

BENCHMARK_SUITE_NAME=<string>

Define the name of benchmark suite to be run. This is required when running RTL compatibility and RTL verification tests.

8.2 Makefile Targets

8.2.1 Top-level Makefile

Top Makefile

This is the top-level makefile of the project

help

compile

This command compiles the RTL designss under a given specific benchmark suite name This command uses the RTL list generated by the `rtl_list` target

cocotb_test

This command run HDL simulations for the RTL designss with cocotb testbenches under a given specific benchmark suite name This command uses the RTL list generated by the `rtl_list` target

clean

This command removes all the intermediate files during rtl compilation and cocotb verification

vexriscv

This command will checkout the latest VexRiscV, then update RTL and testbenches

verilog-spi

This command will checkout the latest SPI, then update RTL and testbenches

dspfilters

This command will checkout the latest DSP filters, then update RTL and testbenches

cordic

This command will checkout the latest cordic designs, then update RTL and testbenches

update_version

Update the patch count in the version number

release_version

Update the patch count in the version number

generate_initial_tagged_commit

Create the first version of tagged commit file, used for version update

CHAPTER

NINE

CONTACT

Xifan Tang

xifan@osfpga.org

CHAPTER

TEN

REFERENCES

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[TGA+19] X. Tang, E. Giacomin, A. Alacchi, B. Chauviere, and P. Gaillardon. Openfpga: an opensource framework enabling rapid prototyping of customizable fpgas. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, volume, 367–374. Sep. 2019. doi:10.1109/FPL.2019.00065.

INDEX