

# Transaction Equivalence Formal Check(DPV) in Video Algorithm/FPU/AI Area

Minqi Bao, Enflame, Shanghai, China(bob.bao@enflame-tech.com)



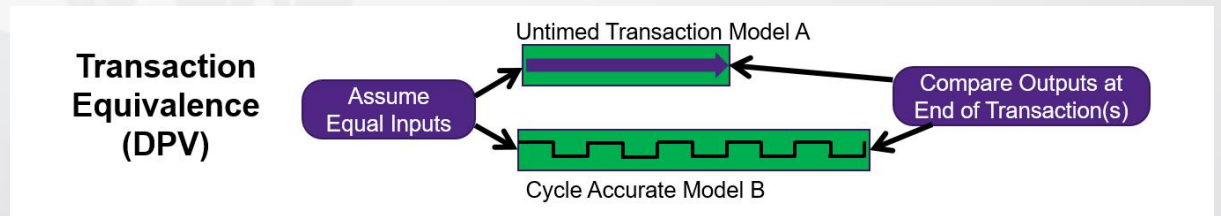
# OUTLINE

- Introduction
- Methodology
  - General flow
  - Video algorithm use model and results
  - Floating point computation use model and results
  - AI computation use model and results
  - Debugging method

# Introduction

All these pain points drive us to find a more efficient verification methodology. In this paper, we will use “DPV”, a transaction equivalence based formal tool, to gain more than 10X efficiency verification improvement in datapath verification. By transaction, we mean the following:

- A transaction consists of Inputs, Input State (optional), State change Outputs, Output State (optional).
- A transaction can be combinational, sequential overlapping / pipelined, or sequential non-overlapping.

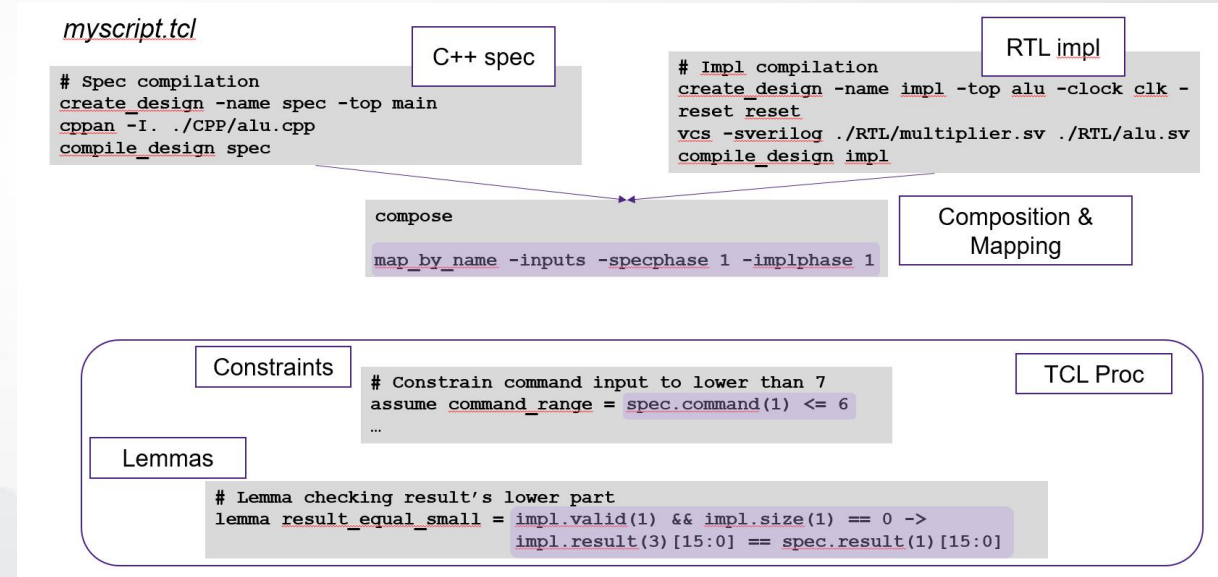


# Methodology

## General flow

DPV general flow is very simple and can be divided into four parts

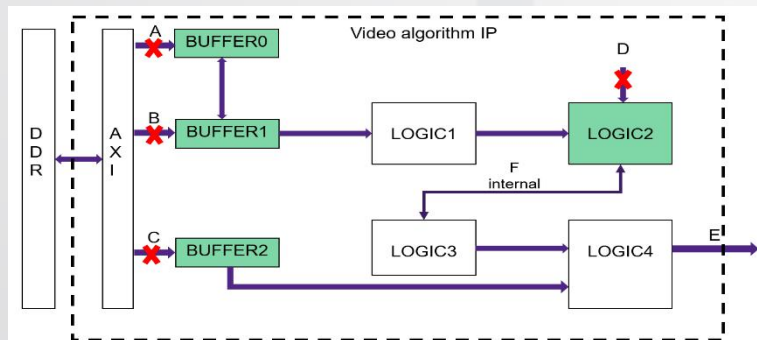
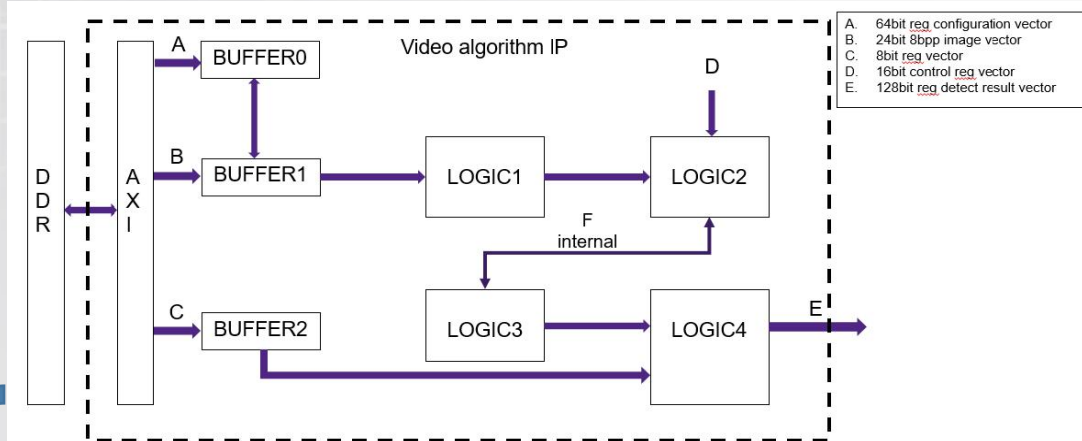
- Compile C model – gnu compatible
- Compile RTL – vcs compatible
- Compose and mapping – compose automatically done by tool and mapping needs manual work usually
- Set constrain(constraints) and checkers(lemmas) and prove them by formal engines



# Methodology

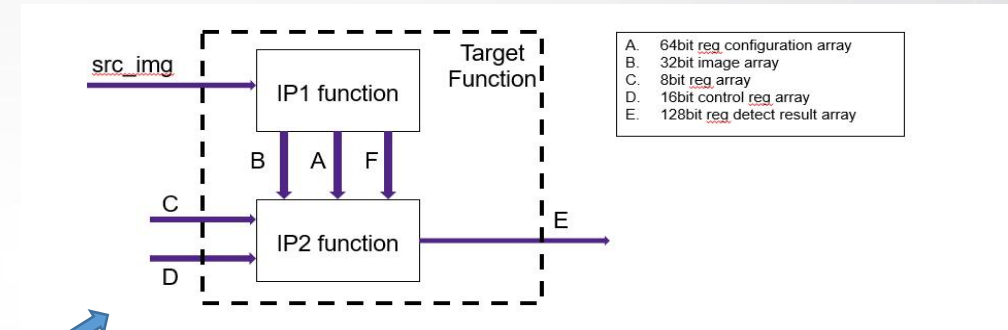
## Video algorithm use model and results

Original RTL



Optimized RTL structure for DPV

C



Corner bug found with transactional equivalence:

The maximum of the gradient is represented by a 11bits signal. After updated, the gradient operator is changed and the maximum of the gradient should be a 13bits signal while it keeps 11bits in design which caused the inconsistent with C++ golden

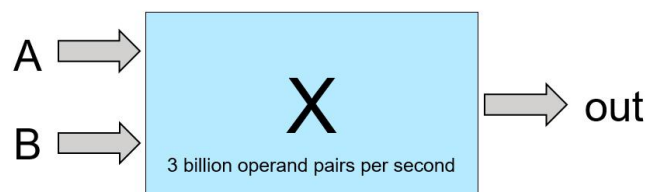
Method	Time to uncover this bug
Simulation	Days or weeks depends on specific pattern
DPV (transactional equivalence)	3 hours without any pattern effort



# Methodology

## Floating point computation use model and results

In FPU side, we choose a FP64 FMA with in-house radix booth encoding. It needs at least 2 weeks for all UVM testbench and coverage points setup while it just requires one-hour setup for transaction equivalence exhaustive verification.

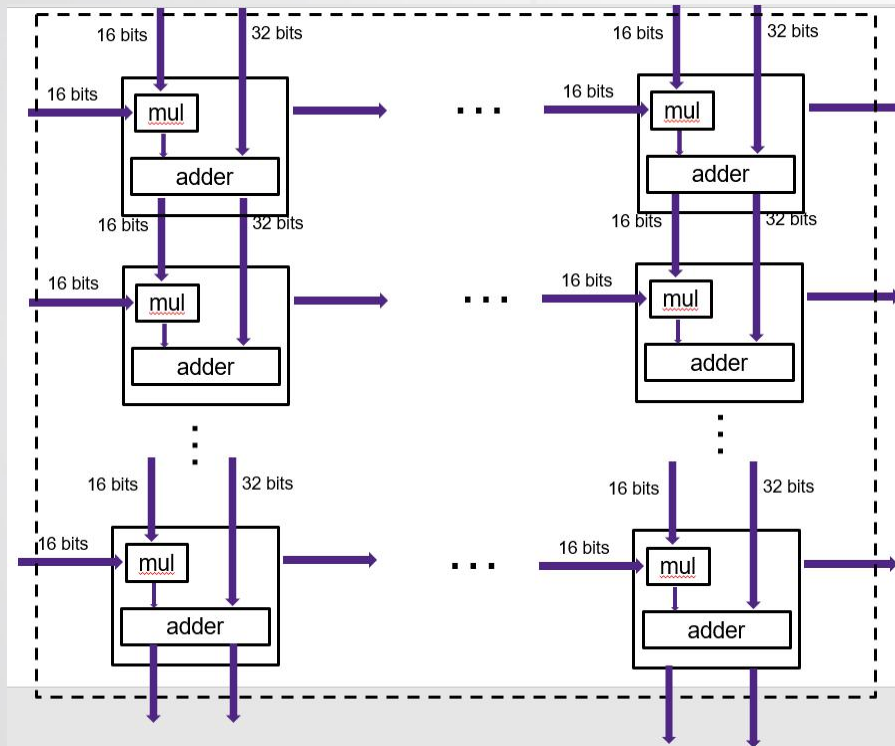


16-bit operands	32-bit operands	64-bit operands
1.5 seconds	195 years	$3.5 \times 10^{21}$ years > earth age

Key Concerns	Traditional flow	Transaction Equivalence flow
Algorithm design block verification	<ul style="list-style-type: none"><li>Develop UVM testbench</li><li>Random test to improve coverage</li></ul>	<ul style="list-style-type: none"><li>Easy <u>tcl</u> setup and no <u>testbench</u> needed</li><li>Exhaustive proof</li></ul>
FP64 FMA	<ul style="list-style-type: none"><li>Completed basic verification via UVM test with 14 days typically</li><li>Impossible to be exhaustive</li></ul>	<ul style="list-style-type: none"><li>Setup formal check in one day</li><li>Possible to do exhaustive verification</li></ul>
Sign-off	<ul style="list-style-type: none"><li>Need huge manual coverage review to meet <u>datapath</u> function sign off quality</li></ul>	<ul style="list-style-type: none"><li>Comprehensive <u>datapath</u> function sign off flow</li></ul>

# Methodology

## AI computation use model and results



Scenario	Results
Matrix 8 x 8	proved in seconds
Matrix 32 x 32	proved in minutes
Matrix 64 x 64	proved in a few hours
Matrix 128 x 128	proved in 30 hours

# Methodology

## Debugging method

Total #lemmas    Proven    Falsified    All    Other Status    AEP/user lemma    Successful engine    Elapsed time

VCF TaskList		VCF GoalList				
Name	Progress	Result	status	name	engine	elapsed_time
p	16:10:4:2		1	p	orch_satonly	1
q	16:10:4:2		2	q	orch_satonly	1
			3	spec_SCV_REG_AEP_2(2)	aep	1
			4	spec_SCV_REG_AEP_1(2)	aep	1
			5	spec_SCV_REG_AEP_0(2)	aep	1
			6	spec_SCV_COMB_AEP_1(3)	aep	1
			7	spec_SCV_COMB_AEP_1(1)	aep	1
			8	spec_SCV_COMB_AEP_0(3)	aep	1
			9	spec_SCV_COMB_AEP_0(1)	aep	1
			10	impl_SCV_REG_AEP_2(2)	aep	1
			11	impl_SCV_REG_AEP_1(2)	aep	1
			12	impl_SCV_REG_AEP_0(2)	aep	1
			13	impl_SCV_COMB_AEP_1(3)	aep	1

Double click to open cex trace

spec/impl indicator

The screenshot shows the VCF interface with a list of verification targets. The 'Verification Targets' table includes columns for status, name, type, engine, elapsed\_time, and expression. The 'status\_3' target is highlighted. Below the table, the 'Constraints: ALL' section lists several constraints. The bottom part of the interface shows a 'spec/impl indicator' window with a trace of the verification process.

status	name	type	engine	elapsed_time	expression
1	resultCheck	user	caseplit		spec.r11 == impl.r11
2	status_0	user	caseplit		spec.status(0) == impl.status(0)
3	status_1	user	caseplit		spec.status(1) == impl.status(1)
4	status_2	user	caseplit		spec.status(2) == impl.status(2)
5	status_3	user	caseplit		spec.status(3) == impl.status(3)
6	status_4	user	caseplit		spec.status(4) == impl.status(4)

name	Prop ID	expression
1	10	spec.r11 == impl.r11
2	11	(spec.r11 <= 0)
3	12	spec.b11 == impl.b11
4	13	spec.r11 == impl.r11