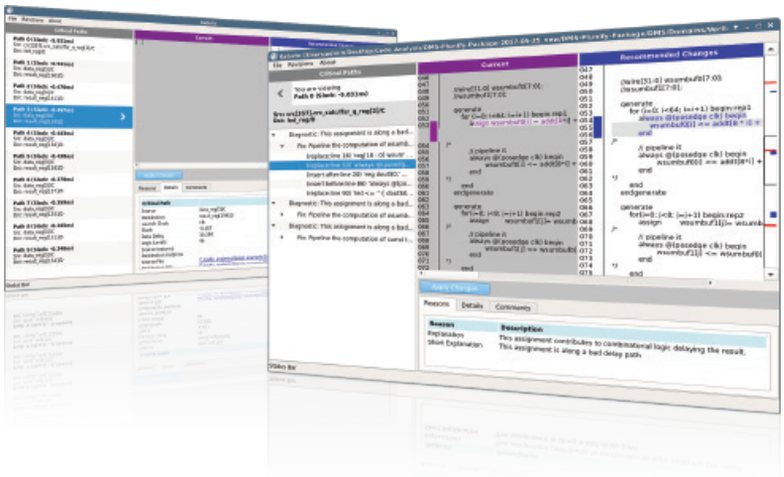
PLUNIFY

Kabuto[Beta] Datasheet



Kabuto is an expert software, driven by machine-learning, that helps RTL designers write better code to solve timing failures and improve design performance. At its heart is a pattern recognition engine that identifies “bad” RTL code, meaning Verilog and VHDL that cause timing or performance problems. Kabuto prioritizes critical paths in a design and provides intelligent recommendations on how to fix the corresponding RTL segments. The designer makes the final decision to accept or edit the recommended changes. Kabuto works together with InTime as well.

RTL synthesis and place-and-route tools from different vendors process code differently, so what is “bad” code for one vendor’s tools may not cause performance issues in another. On top of this, different types of device families can require RTL coding styles that are not uniform. Kabuto recognizes and automatically accounts for such disparities.

How it works

Kabuto analyzes the timing reports of a compiled design and focuses on the critical paths. From the critical path data, Kabuto zeroes in on the RTL code segments that are relevant to each failing path. Kabuto figures out what the problems with each particular code segment are and provides the necessary code fix recommendations to the designer. The designer assesses the validity of the recommendations and chooses to accept them or manually edit the RTL before saving the design.

Key features & benefits

- Smart pattern recognition engine detects complex issues, such as pipelining
- Analysis and recommendation engine provides education and guidance
- Code style correction to ease and assist transition to new designs and tool chains
- Customizable pattern recognition engines incorporate custom RTL guidelines with different design goals

What it can detect

Kabuto detects various RTL issues. Examples of what Kabuto can detect includes:

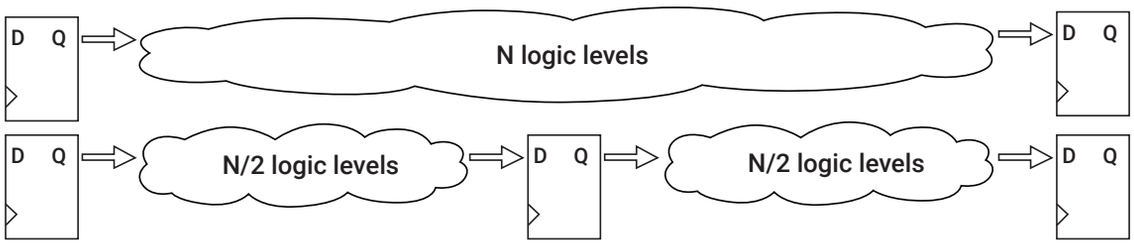
A: Multiply-by-a-constant

Kabuto uses a conversion algorithm that produces a new expression using shift and addition/subtraction operations to replace a multiply with a constant operation. This proves to be more synthesis-friendly in term of timing.

Detected	Kabuto Recommended Change
Out1 = in1 * 60;	Out1 = (in1<<6) – (in<<2);

B: Add-pipeline-stage

Pipelining breaks up the long delay path into smaller pipeline stages separated by registers. Kabuto also detects and corrects pipeline dependencies to prevent hazards.



Detected	Kabuto Recommended Change
assign wsumbuf0[i] = addt[8*i] + addt[8*i+1] + addt[8*i+2] + addt[8*i+3] + addt[8*i+4] + addt[8*i+5] + addt[8*i+6] + addt[8*i+7];	always @(posedge clk) begin wsumbuf0[i] <= addt[8*i] + addt[8*i+1] + addt[8*i+2] + addt[8*i+3] + addt[8*i+4] + addt[8*i+5] + addt[8*i+6] + addt[8*i+7]; end

Detected (pipeline dependent signals)	Kabuto Recommended Change

always @(posedge clk) begin
if (!rst)
led <= 'b0;
else
led <= ^{dout,sumd};
end
	reg dout$0;
	
	always @(posedge clk) begin
	dout$0 <= dout;
	end
	
	always @(posedge clk) begin
	if (!rst)
	led <= 'b0;
	else
	led <= ^{dout$0,sumd};
	end

C: If-no-else lead to unintentional latches

Latches can cause timing problems and even race conditions, so compilers will generally warn about them. One common cause of unintentional latches is the lack of an "else" clause in an "if" statement.

Detected	Kabuto Recommended Change

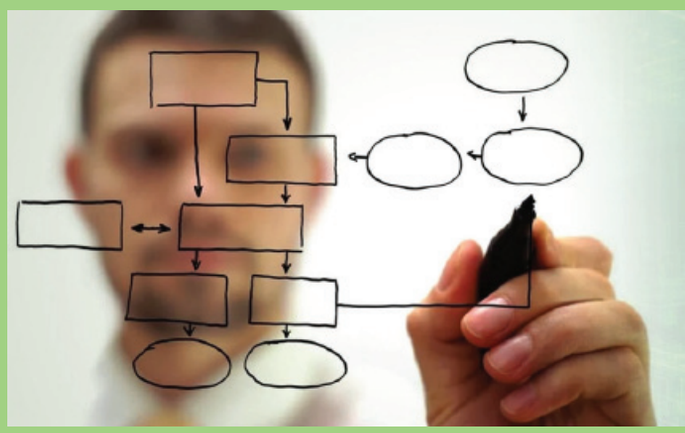
process(vi, en)
begin
if en = '1' then
for i in vi'range loop
vo(i) <= vi(nbits-1 - i);
end loop;
end if;
end process;
	process(vi, en)
	begin
	if en = '1' then
	for i in vi'range loop
	vo(i) <= vi(nbits-1 - i);
	end loop;
	else
	for i in vi'range loop
	vo(i) <= 'X';
	end loop;
	end if;
	end process;

Highly customizable recognition engine

Kabuto comes with a highly sophisticated RTL recognition engine that can be customized to identify various RTL issues, such as area, power.

Many organizations have internal coding guidelines or training to adhere to specific standards. These standards can be different for different platforms such as FPGA versus ASICs, or coding styles for newer FPGA device families.

Kabuto can be used as a tool to assist and help designers write better RTL to target the different platform and devices.



System requirements

- Minimum 1GB RAM, with 4 GB+ virtual memory
- At least 100MB free disk space for Kabuto software
- Processor: Intel i3 CPU or similar
- Java: Java Runtime Environment (JRE 1.6 and above)

Licensing

- Annual license priced according to the number of users/seats

Specifications

- Supported OS: Windows 7 and above. Ubuntu and Centos
- Supported FPGA tools: Quartus 15 and above. Vivado 2015.4 and above

About Plunify

Plunify helps chip design companies optimize FPGA designs with big data and machine learning. Plunify is based in Singapore and in the United States.

Plunify Pte Ltd
Email: tellus@plunify.com

Singapore
82, Lorong 23 Geylang,
Atrix Building, #05-14,
Singapore (388409)

United States
4962 El Camino Real #225
Los Altos, CA 94022
USA

China
Room 1737, Level 17, Raffles
City Tower 2, No. 3 Section 4,
South Renmin Road, Wuhou
District, Chengdu, 610041,
China

PLUNIFY