Chapter 0
GV Preliminary

GV tool (General Verification) mainly integrates two open source engines, "berkeley-abc" and "yosys". The former focuses on synthesis and verification of gate-level circuits, while the latter is powerful at its framework for Verilog RTL synthesis. Hence, GV provides an end-to-end platform to do further application for input DUV (Design Under Verification), which means GV is switching between the two engines.

berkeley-abc

- official API document
- source code
- In GV, "berkeley-abc" engine supports reading ".v", ".aig", ".blif".

yosys

- official API document
- source code
- In GV, "yosys" engine supports reading ".v", ".sv", ".blif".
- In GV, "yosys" engine supports writing files to ".aig", ".blif", ".btor".

Besides, GV tool has two modes, "setup" and "vrf" (verification), to differentiate APIs according to their properties. Please note that you need to read in a DUV first before executing other commands.

For example, we need to read in a DUV or even rename its wire, get the DUV's information, and convert the DUV to another format (btor, blif, aig, etc.). These preprocessing are setting a DUV, so the APIs for above operations will be categorized into "setup" mode.

After finishing to preprocess a DUV, it should not be modified, so we will switch to "vrf" mode to do verification, such as formal verification, simulation, plot designs, etc.

APIs in "setup" and "vrf" mode

- refer to “Appendix B: APIs and GV Mode”

In the following, we will introduce how to switch engines or switch modes.

1. After successful compilation of GV, the prompt will be default in “setup” mode with “yosys” engine.

   ```
   yosysSETUP>
   ```

2. You can call any APIs in setup mode that yosys supports, e.g. reading a RTL file.

   ```
   yosysSETUP> read design -v <filename>.v
   ```

3. But if you want to read in “aig” file, which is supported by “berkeley-abc” engine instead of “yosys”, then you can switch the engine from “yosys” to “berkeley-abc”, and vice versa.

   ```
   yosysSETUP> set engine abc
   abcSETUP> set engine yosys
   yosysSETUP>
   ```

4. Please note that for yosys, it is not allowed to read a new design if there exists one, so we need to reset the yosys before reading another design. While DUV in berkeley-abc can be overwritten.

   ```
   yosysSETUP> read design -v <filename>.v
   yosysSETUP> reset system yosys
   yosysSETUP> read design -v <new_filename>.v
   ```

5. After preprocessing, if we want to do verification, such as formal verification, random simulation, plot a design, we should switch to “vrf” mode to ensure the DUV will not be modified, and vice versa.

   ```
   yosysSETUP> read design -v <filename>.v
   yosysSETUP> set system vrf
   vrf> set system setup
   yosysSETUP>
   ```

Chapter 1
Read and Write Designs

1.1 Introduction

In this tutorial we demonstrate how users can read designs into GV and write designs out from GV by commands. In addition, we illustrate how to print network information or plot networks by GV.

1.2 Prerequisites

Please download the latest GV, and let *gv0/* be the root directory. Make sure GV has been successfully installed such that an executable named *gv* is located under *gv0/*. In addition, check if the following files exists under gv0/design/V3/alu:

- ☐ alu.v : Verilog design
- ☐ alu.blif : BLIF design
- ☐ alu.aig : AIGER design

1.3 Read Designs

Command: REad Design <-Verilog | -BLif | -Aig> <filename>
Engine:

- **berkekey-abc**: supports reading "-Verilog (.v)", "-BLif (.blif)", and "-Aig (.aig)". But for ".v", it basically supports combinational circuits.
- **yosys**: supports reading "-Verilog (.v / .sv)", and "-BLif (.blif)".

When reading a design, please use "set engine <abc | yosys>" to choose the compatible frontend engine. Also, please use "reset system <abc | yosys>" when you want to read in another design. After reading in a design, then it will be parsed and stored in GV's data structure to do further application.

Example:

1. Read a Verilog RTL design by yosys: design/V3/alu/alu.v

   yosysSETUP> read design -v design/V3/alu/alu.v

2. Read a Blif design by yosys: design/V3/alu/alu.blif

   yosysSETUP> reset system yosys

   yosysSETUP> read design -blif design/V3/alu/alu.blif

3. Read an Aig design by berkeley-abc: design/V3/alu/alu.aig

   yosysSETUP> reset system yosys

   yosysSETUP> set engine abc

   abcSETUP> read design -aig design/V3/alu/alu.aig

1.4 Print Information of a Design

Command: PRint Info [-Verbose]
Engine:

- **berkekey-abc**: supports reading "-Verilog (.v)", "-BLif (.blif)", and "-Aig (.aig)". But for ".v", it basically supports combinational circuits.
- **yosys**: supports reading "-Verilog (.v / .sv)", and "-BLif (.blif)".

If call "berkeley-abc" as the frontend engine, then the information of the design will use "berkeley-abc" engine's information format. Likewise for "yosys".

Example:

1. Read a Verilog RTL design by yosys and print information

   yosysSETUP> read design -v design/V3/alu/alu.v

   yosysSETUP> print info -verbose

```
yosysSETUP> print info -v
I am GVPrintInfoCmd
Modules in current design: \alu(11 wires, 5 cells)
================================================
   MUX                                         0
   AND                                         0
   ADD                                         1
   SUB                                         1
   MUL                                         1
   EQ                                          0
   NOT                                         1
   LT                                          0
   GE                                          0
------------------------------------------------
   PI                                          4
   PO                                          1
================================================
```

2. Read an Aig design by berkeley-abc and print information

   yosysSETUP> set engine abc

   abcSETUP> read design -aig design/V3/alu/alu.aig

   abcSETUP> print info -verbose

```
abcSETUP> print info -v
I am GVPrintInfoCmd

Circuit Statistics
==================
  PI           27
  PO           32
  Node       1292
------------------
  Total      1351
```

1.5 Plot a Design

Command: SHow <-Vcd <filename>.vcd | -SCHematic>
Engine:

- **GTKWave**: input vcd file, and plot waveform (-Vcd)
- **yosys**: input DUV, and plot RTL design (-SCHematic)

Please note that the command “show” is under “vrf” mode, so switch to “vrf” mode before executing “show” (refer to “Appendix B: APIs and GV Mode”)

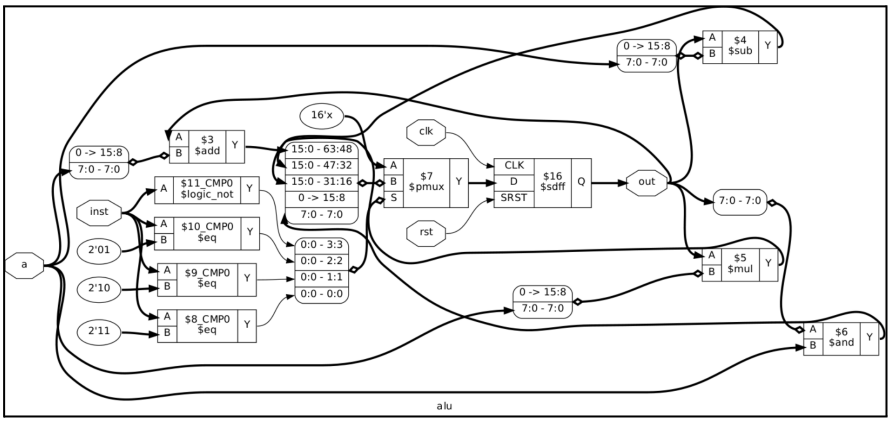
Example:

1. Read a Verilog RTL design by yosys and plot its architecture

   yosysSETUP> read design -v design/V3/alu/alu.v

   yosysSETUP> set system vrf

   vrf> show -schematic



1.6 Write Designs

Command: FILE2 Aig <-Verilog | -BLif> -TOP <top_module> -Output <output_filename>
Command: FILE2 BTOR <top_module> <output_filename>
Command: FILE2 BLIF <top_module> <output_filename>
Engine:

- **yosys**: yosys backend supports converting DUV to “.aig”, “.btor”, “.blif”, etc.

“File2 Aig” needs to specify whether the current DUV is “.v” or “.blif”.

Example:

1. Read a Verilog RTL design by yosys and convert it to “.aig”, “.btor”, and “.blif” file

   yosysSETUP> read design -v design/V3/alu/alu.v

   yosysSETUP> file2 aig -v -top alu -output alu.aig

   yosysSETUP> file2 btor alu alu.btor

   yosysSETUP> file2 blif alu alu.blif

Chapter 2
Design Simulation

1.1 Introduction

In this tutorial we demonstrate how users can simulate its design. GV provides random simulation or using user-provided patterns to get output results.

1.2 Random Simulation

Command: RAndom Sim [options]
[options]

- [-clk]
  - clock signal name in DUV, default to be “clk”
- [-rst]
  - reset signal name in DUV if it is **actively high**, default to be “reset”
- [-rst_n]
  - reset signal name in DUV if it is **actively low**, default to be “reset_n”
- [-sim_cycle]
  - specify the simulation cycle, cycle default to be “20”
- [-output]
  - output filename, dump the simulation result in it
- [-v]
  - verbose, print the simulation result on the terminal
- [-file]
  - input pattern file name, please refer to “Appendix A: Format of Input Pattern for Simulation” to generate your own pattern file
- [-vcd]
  - output filename, dump the simulation result in a VCD file

Engine:

- **yosys:** use simulator in yosys

Please note that the command “random sim” is under “vrf” mode, so switch to “vrf” mode before executing “random sim” (refer to “Appendix B: APIs and GV Mode”). Also, please refer to “Appendix A: Format of Input Pattern for Simulation” if want to use self-provided input patterns.

Example:

1. Read a Verilog RTL design by yosys and switch to “vrf” mode

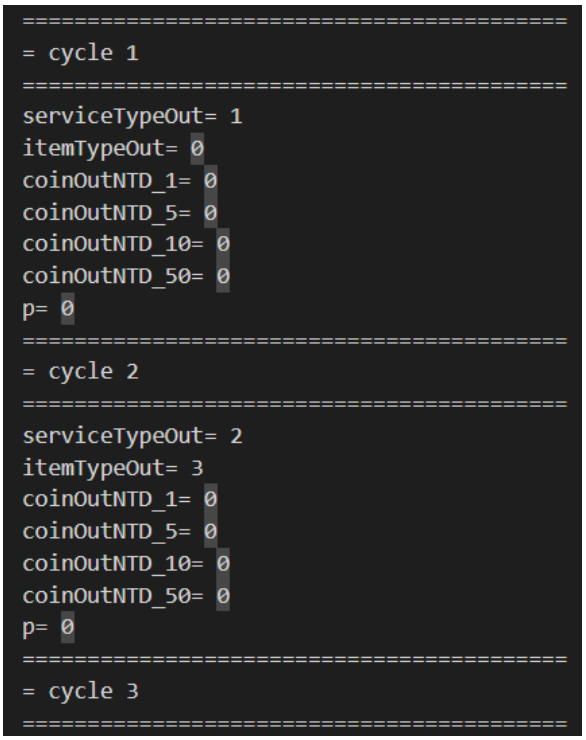
   yosysSETUP> read design -v ../vending-simple/vending.v

   yosysSETUP> set system vrf

   vrf>

2. input a user-provided “input.pattern” to do simulation, and output the result to file “output.txt”, then the output wire and its result will be dumped

   vrf> random sim -v -file ../vending-simple/input.pattern -sim_cycle 40 -clk clk -rst_n reset -output ../vending-simple/output.txt



Appendix A
Format of Input Pattern for Simulation

1.1 Introduction

In “Chapter 2: Design Simulation”, GV introduces a command “RAndom Sim [options]”, and users can input their own input pattern file through the option “-file <filename>”. In this chapter, we will define the format of input pattern file for simulation.

1.2 Input DUV

Here we use “vending.v” and “input.pattern” under *vending-simple/* directory as our DUV example.

In the DUV, except for “clk” and “reset” signal, we can find that the module “vendingMachine” contains: (*fig.1*)

- **5 input ports**
  - itemTypeIn, coinInNTD_1, coinInNTD_5, coinInNTD_10, coinInNTD_50
- **6 output ports**
  - serviceTypeOut, itemTypeOut, coinOutNTD_1, coinOutNTD_5, coinOutNTD_10, coinOutNTD_50, p

(note that here we use inverse order for the ports compared with the module “vendingMachine”, because the simulator will also input/output in this order)

For the input pattern file, each row represents one pattern, and each pattern contains N **decimal** numbers for an N -input module. Between each decimal number, it needs a “space” to separate.
(note that the first row (first pattern) can be arbitrary for the initialization)

- e.g. input.pattern for vending.v under *vending-simple/* directory

1 0 0 0 0
3 2 2 2 2
2 2 2 2 2
…
…

- **row 1**: arbitrary for initialization
- **row 2**: pattern 1, (itemTypeIn, coinInNTD_1, coinInNTD_5, coinInNTD_10, coinInNTD_50) = (3, 2, 2, 2, 2) = (2'b11, 2'b10, 2'b10, 2'b10, 2'b10)

- e.g. output result for input.pattern above (*fig.2*)
  - dump each output ports results at every cycle

```
module vendingMachine(
    // Property Output Ports
    p,
    // General I/O Ports
    clk,
    reset,
    // Input Ports
    coinInNTD_50,
    coinInNTD_10,
    coinInNTD_5,
    coinInNTD_1,
    itemTypeIn,
    // Output Ports
    coinOutNTD_50,
    coinOutNTD_10,
    coinOutNTD_5,
    coinOutNTD_1,
    itemTypeOut,
    serviceTypeOut
);
```

(*fig.1*)

```
========================================
= cycle 1
========================================
serviceTypeOut= 1
itemTypeOut= 0
coinOutNTD_1= 0
coinOutNTD_5= 0
coinOutNTD_10= 0
coinOutNTD_50= 0
p= 0

========================================
= cycle 2
========================================
serviceTypeOut= 2
itemTypeOut= 3
coinOutNTD_1= 0
coinOutNTD_5= 0
coinOutNTD_10= 0
coinOutNTD_50= 0
p= 0
```

(*fig.2*)

Appendix B
APIs and GV Mode

1.1 Introduction

In this chapter, we will categorize GV released commands into “setup” or “vrf” mode, if you want to delve into the usage of a command, please type “help <command>” to see.

In the following, the commands will be listed in green font color, and the capital part of a command means “at least you need to type these words”.

1.2 SETUP mode commands

- **common command :** (compatible in both “setup” and “vrf” mode)
  - DOfile : Execute the commands in the dofile
  - HELp : Print the help message
  - HIStory : Print command history
  - USAGE : Report resource usage
  - Quit : Quit the execution
- **mode command :** (compatible in both “setup” and “vrf” mode)
  - SEt SYStem : Switch to setup/vrf mode
  - RESET SYStem : Delete all networks in GV & reset to setup mode
  - WIZard : Start the GV tutorial wizard
- **network command :**
  - BLAst NTK : Convert network to AIG
  - CIRGate : Report a gate
  - CIRPrint : Print circuit
  - CIRRead : Read in a circuit & construct the netlist
  - FILE2 Aig : Convert Verilog file into AIG
  - FILE2 BLIF : Convert Verilog file into BLIF
  - PRInt Aig : Print the AIG network information

    - PRint Info : Print circuit information extracted by GV
    - REad Design : Read in design
    - SEt Engine : Set the specific engine to parse the design
    - WRite Aig : Write out the processing design into AIGER file
    - YOSYSCMD : Directly call Yosys's command
- **partial berkeley-abc command :**
    - ABCRead : Read netlist by ABC
    - ABCPrint : Print netlist information
    - ABCCMD : Directly call ABC's command
- **BDD command :**
    - BAND : BDD AND
    - BINV : BDD Inverter
    - BNAND : BDD NAND
    - BNOR : BDD NOR
    - BOR : BDD OR
    - BXNOR : BDD XNOR
    - BXOR : BDD XOR
    - BCOFactor : Retrieve BDD cofactor
    - BCOMpare : BDD comparison
    - BConstruct : Build BDD from current design
    - BDRAW : BDD graphic draw
    - BEXist : Perform BDD existential quantification
    - BREPort : BDD report node
    - BRESET : BDD reset
    - BSETOrder : Set BDD variable order from circuit
    - BSETVar : BDD set a variable name for a support
    - BSIMulate : BDD simulation

1.3 VRF mode commands

- **common command :** (compatible in both “setup” and “vrf” mode)
  - DOfile : Execute the commands in the dofile
  - HELp : Print the help message
  - HIStory : Print command history
  - USAGE : Report resource usage
  - Quit : Quit the execution
- **mode command :** (compatible in both “setup” and “vrf” mode)
  - SEt SYStem : Switch to setup/vrf mode
  - RESET SYStem : Delete all networks in GV & reset to setup mode
  - WIZard : Start the GV tutorial wizard
- **formal verification command :**
  - Formal Verify : Use options to execute formal engine (e.g. abc)
- **simulation command :**
  - RAndom Sim : Conduct random simulation & print the results
  - SET SAfe : Set safe property for random simulation
  - SHow : Show the waveform or schematic
- **prove command (BDD) :**
  - PCHECKProperty : Check the monitor by BDDs
  - PIMAGe : Build the next sate images in BDDs
  - PINITialstate : Set initial state BDD
  - PTRansrelation : Build the transition relationship in BDDs
- **itp command (SAT) :**
  - SATVerify BMC : Check the monitor by bounded model checking
  - SATVerify ITP : Check the monitor by interpolation-based method