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).

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 modes.

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

   ```
   setup>
   ```

2. You can call any APIs in setup mode, e.g. reading a RTL file by “cirread -v”.

setup> cirread -v <filename>.v

3. Please note that it is not allowed to read a new design if there exists one, so we need to replace the gv original design before reading another design.

   setup> cirread -v <filename>.v

   setup> cirread -replace -v <new_filename>.v

4. 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.

   setup> cirread -v <filename>.v

   setup> set system vrf

   vrf> set system setup

   setup>

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 *socv-1122/* be the root directory. Make sure GV has been successfully installed such that an executable named *gv* is located under *socv-1122/*. In addition, check if the following files exists under *socv-1122/design/V3/alu*:

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

1.3 Read Designs

Command: CIRRead <-Verilog | -BLif | -Aig> <filename>

Example:

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

   setup> cirread -v design/V3/alu/alu.v

2. Read a Blif design: design/V3/alu/alu.blif
   setup> cirread -blif design/V3/alu/alu.blif

3. Read an Aig design: design/V3/alu/alu.aig

   setup> cirread -aig design/V3/alu/alu.aig

1.4 Print Information of a Design

When reading designs in Verilog or BLIF format,

Command: PRint Info [-Verbose]

When reading designs in aig format,

Command: CIRPrint -summary

Example:

1. Read a Verilog RTL design and print information

   setup> cirread -v design/V3/alu/alu.v

   setup> print info -verbose

```
setup> cirread -v design/V3/alu/alu.v
Converted 0 1-valued FFs and 16 DC-valued FFs.

setup> print info -verbose
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 and print information
   setup> cirread -aig design/V3/alu/alu.aig
   setup> cirprint

```
setup> CIRPrint

Circuit Statistics
==================
  PI          11
  PO          16
  LATCH       16
  AIG       1310
------------------
  Total     1353
```

1.5 Plot a Design

Command: SHow

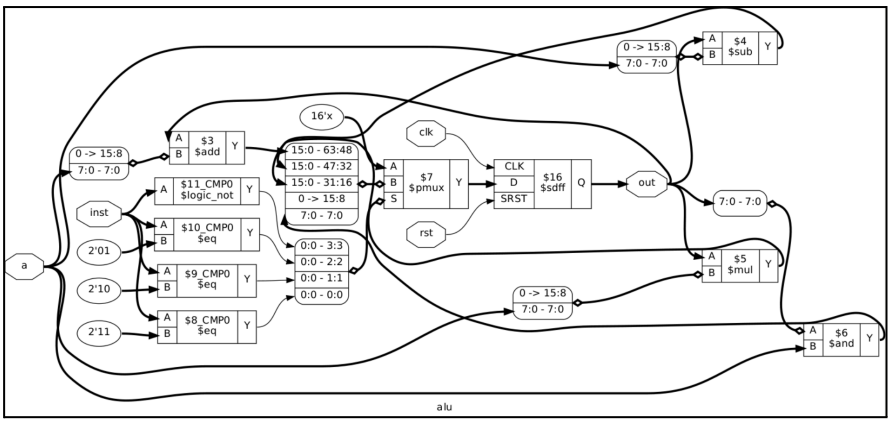
Please note that the command “show” is under “setup” mode, so don’t switch to “vrf” mode before executing “show” (refer to “Appendix B: APIs and GV Mode”)

Example:

1. Read a Verilog RTL design and plot its architecture

   setup> cirread -v design/V3/alu/alu.v

   setup> show



1.6 Write Designs

Command: CIRWrite <-Verilog | -BLif | -aig> -Output <output_filename>

Example:

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

```
setup> cirread -v design/V3/alu/alu.v
setup> cirwrite -v -output alu.v
setup> cirwrite -blif -output alu.blif
setup> cirwrite -aig -output alu.aig
```

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”
- [-reset]
  - reset signal name in DUV if it is **actively high**, default to be “reset”
- [-n_reset]
  - 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

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 you want to use self-provided input patterns.

Example:

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

   setup> cirread -v *design/SoCV/vending/*vending-simple.v

   setup> 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 ./design/SoCV/vending/input.pattern /

   -sim_cycle 40 -clk clk -n_reset reset /

-output ./design/SoCV/vending/output.txt

```
==========================================
= 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
==========================================
= cycle 3
==========================================
```

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
- **network command :**
  - CIRGate : Report a gate
  - CIRPrint : Print circuit
  - CIRRead : Read in a circuit & construct the netlist
  - CIRWrite: Write the netlist to an AIG/AAG/BLIF file
- **partial berkeley-abc command :**
  - ABCCMD : Directly call ABC’s command
- **partial Yosys command :**
  - PRint Info : Print circuit information extracted by GV

    - SHow : Show the waveform or schematic
    - YSYSet: Set the option of Yosys
- **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
- **formal verification command :**
    - Formal Verify : Use options to execute formal engine (e.g. abc)
    - PDR: Model checking using property directed reachability in abc.
- **simulation command :**
    - RAndom Sim : Conduct random simulation & print the results

    - SET SAfe : Set safe property for random simulation
- **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