Tutorial – ABC and Other Tools for CAD Contest

ALCom Lab

Outline

- ABC Introduction
- Logic Circuit Descriptions
- Useful Commands in ABC
- Programming with ABC
- Other Tools

ABC INTRODUCTION

Introduction

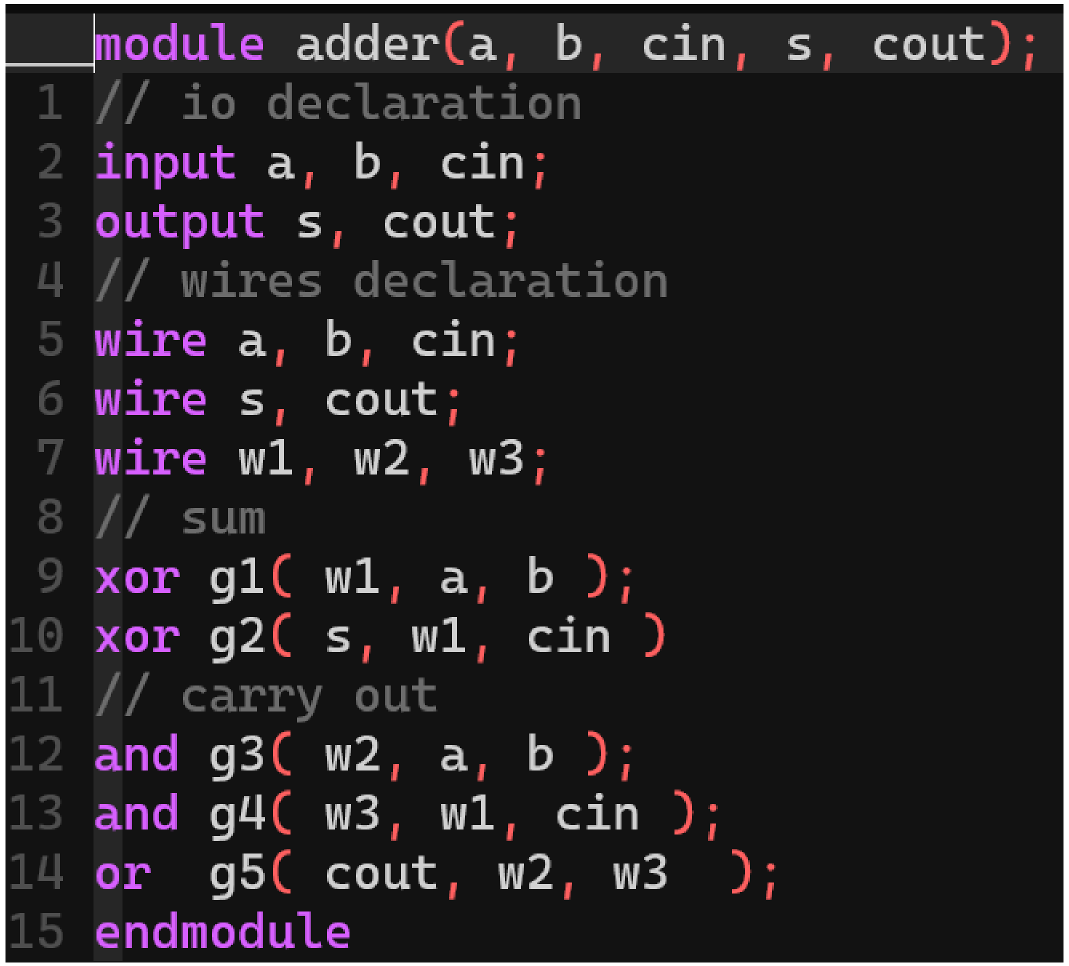
- ABC [2](ABC: System for Sequential Logic Synthesis and Formal Verification) is an academic open source front-end EDA tool
  - Logic synthesis
  - Optimization
  - Verification
  - Technology mapping

Installation

- Prerequisites: linux or mac OS environment
  - for windows, try wsl (recommended) or virtual machine
  - or just work on your work station
- Download (clone/fork) from https://github.com/berkeley-abc/abc
- Simply type "*make*" at the root directory
  - the compilation could take a few minutes
  - "make –j8"

LOGIC CIRCUIT DESCRIPTIONS

Verilog

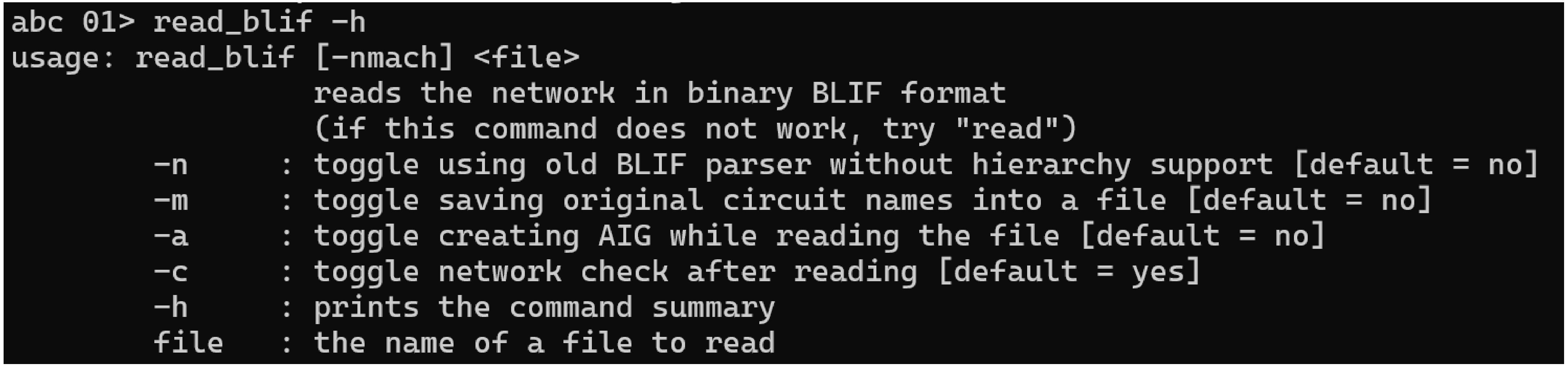


```
module adder(a, b, cin, s, cout);
// io declaration
input a, b, cin;
output s, cout;
// wires declaration
wire a, b, cin;
wire s, cout;
wire w1, w2, w3;
// sum
xor g1( w1, a, b );
xor g2( s, w1, cin )
// carry out
and g3( w2, a, b );
and g4( w3, w1, cin );
or  g5( cout, w2, w3  );
endmodule
```

USEFUL COMMANDS IN ABC

Commands Usage

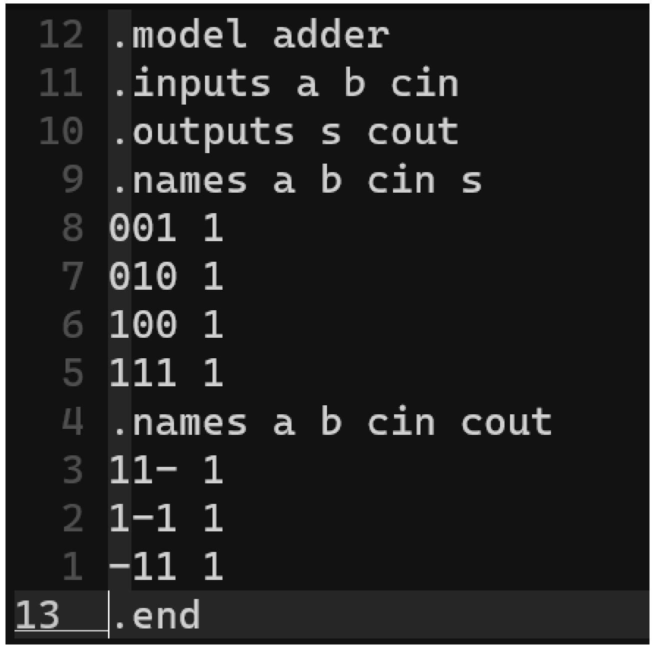
- *help* lists all the commands
- *help –d* list all the commands with details
- Adding option *-h* shows the usage and description of a command



```
abc 01> read_blif -h
usage: read_blif [-nmach] <file>
                 reads the network in binary BLIF format
                 (if this command does not work, try "read")
        -n     : toggle using old BLIF parser without hierarchy support [default = no]
        -m     : toggle saving original circuit names into a file [default = no]
        -a     : toggle creating AIG while reading the file [default = no]
        -c     : toggle network check after reading [default = yes]
        -h     : prints the command summary
        file   : the name of a file to read
```

Read / Write

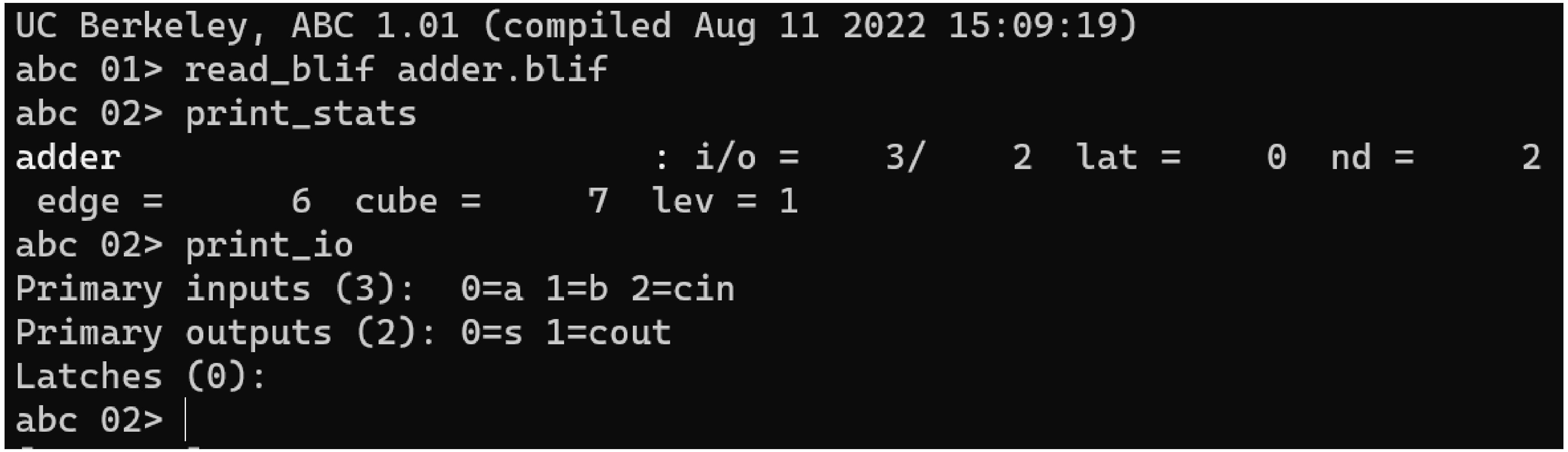
- *read_blif*, *read_aiger*, *read_verilog*, *read_truth*
  - Note that *read_verilog* cannot parse primitive gates in Problem A
- *write_blif*, *write_aiger*, *write_verilog*, *read_truth*



```
.model adder
.inputs a b cin
.outputs s cout
.names a b cin s
001 1
010 1
100 1
111 1
.names a b cin cout
11- 1
1-1 1
-11 1
.end
```

Printing

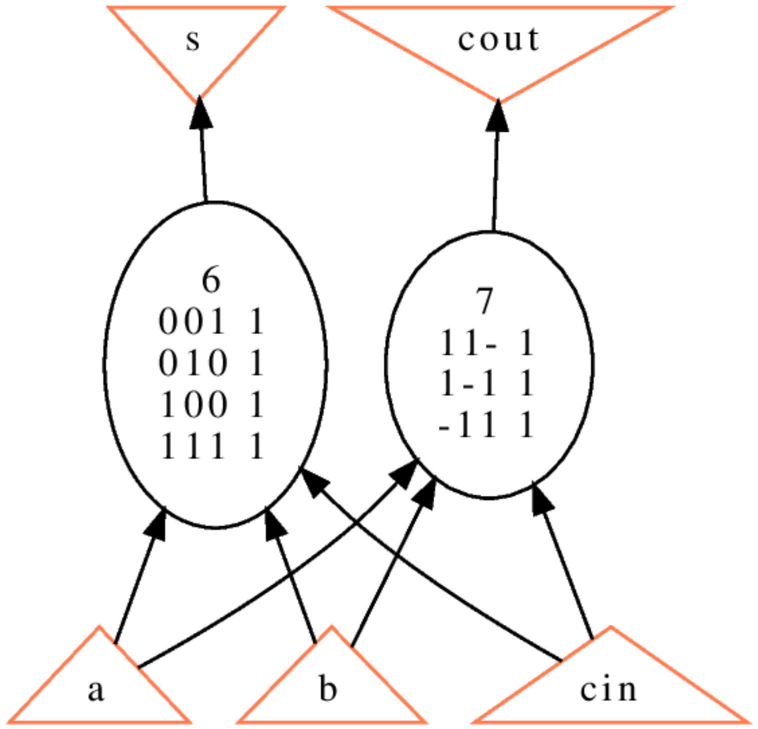
- *print_stats, print_io*

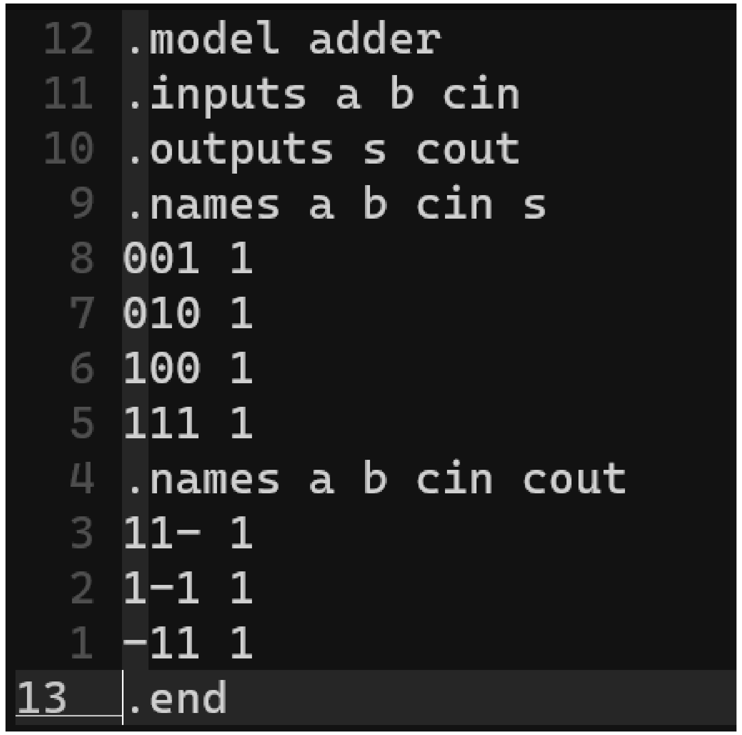


```
UC Berkeley, ABC 1.01 (compiled Aug 11 2022 15:09:19)
abc 01> read_blif adder.blif
abc 02> print_stats
adder                         : i/o =    3/    2  lat =     0  nd =      2
 edge =       6  cube =      7  lev = 1
abc 02> print_io
Primary inputs (3):  0=a 1=b 2=cin
Primary outputs (2): 0=s 1=cout
Latches (0):
abc 02>
```

Printing

- *show*

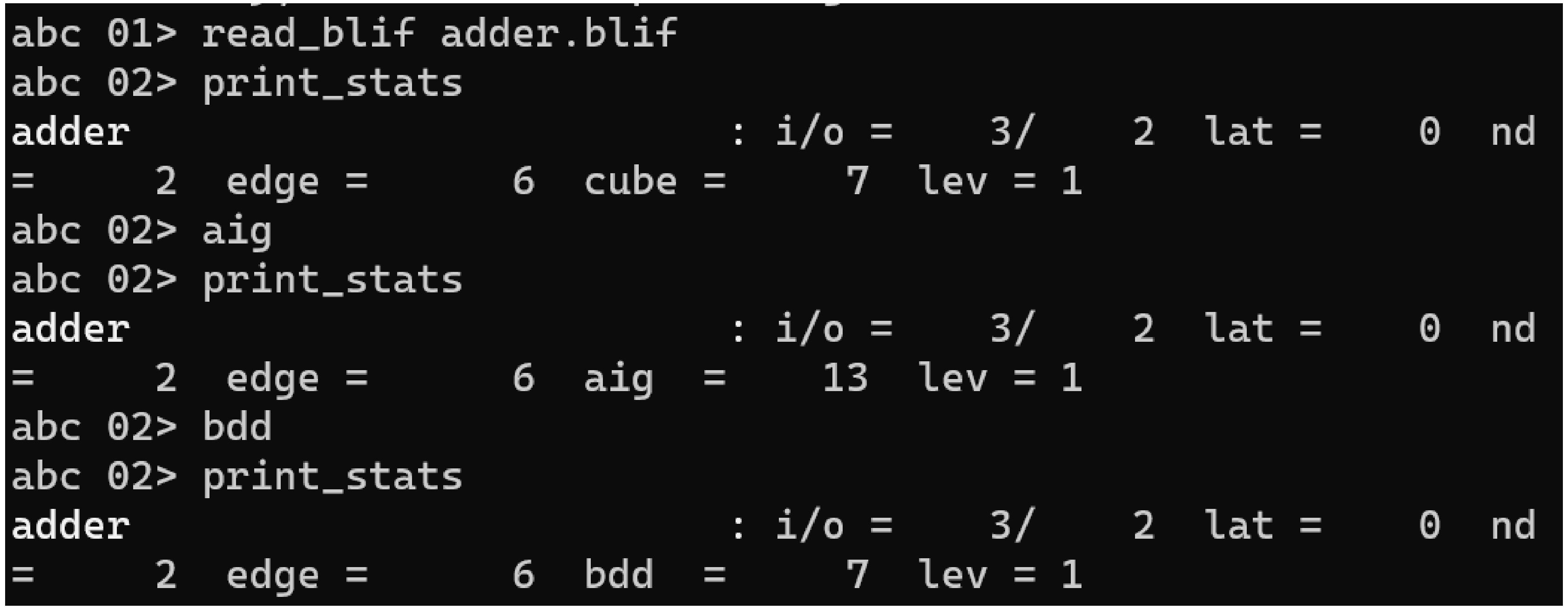




```
.model adder
.inputs a b cin
.outputs s cout
.names a b cin s
001 1
010 1
100 1
111 1
.names a b cin cout
11- 1
1-1 1
-11 1
.end
```

Local Function Representation

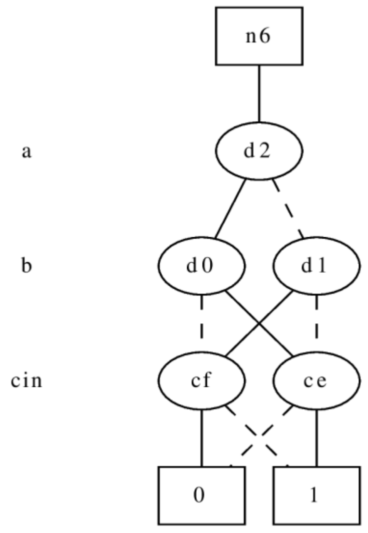
- *aig, bdd, sop*

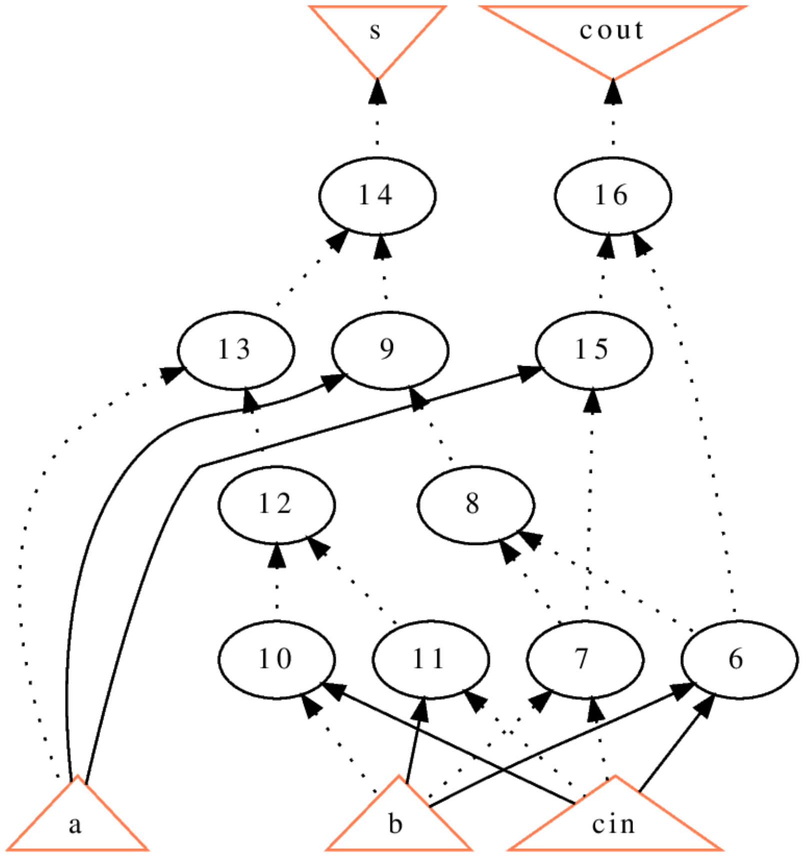


```
abc 01> read_blif adder.blif
abc 02> print_stats
adder                             : i/o =    3/    2  lat =    0  nd
=      2  edge =      6  cube =      7  lev = 1
abc 02> aig
abc 02> print_stats
adder                             : i/o =    3/    2  lat =    0  nd
=      2  edge =      6  aig  =     13  lev = 1
abc 02> bdd
abc 02> print_stats
adder                             : i/o =    3/    2  lat =    0  nd
=      2  edge =      6  bdd  =      7  lev = 1
```

Network Structure

- *collapse* (one BDD for each PO)
  - to print BDD, use *show_bdd*



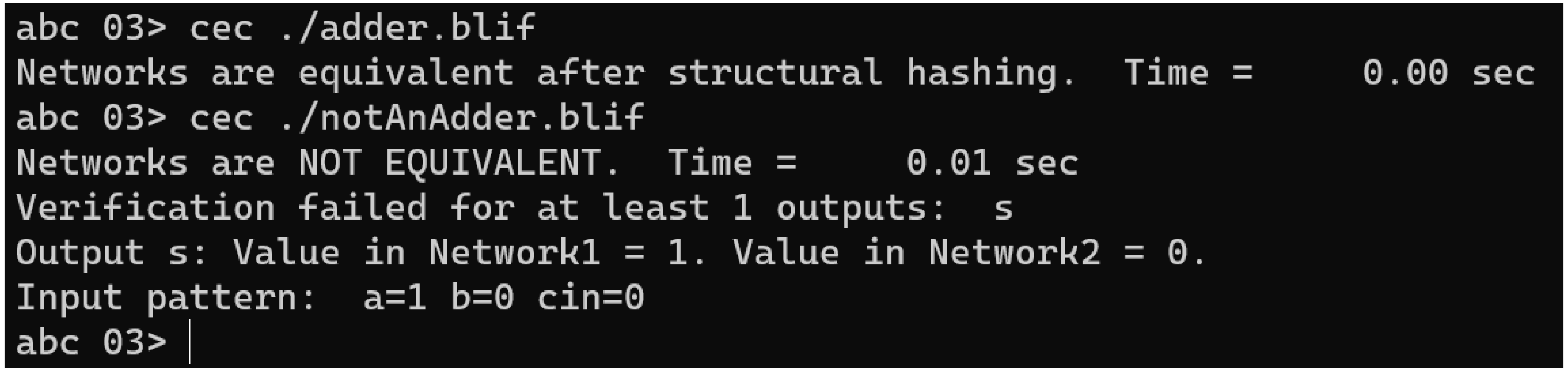


Optimization

- Commands
  - *strash* (structural hashed aig)
  - *fraig* (functionally reduced aig)
  - *dc2, rewrite, balance, resub, refactor, …*
- Scripts (sequence of commands, defined in “abc.rc”)
  - *resyn, resyn2, compress, share, …*

Verification

- *cec, miter, sim, sat*



```
abc 03> cec ./adder.blif
Networks are equivalent after structural hashing.  Time =     0.00 sec
abc 03> cec ./notAnAdder.blif
Networks are NOT EQUIVALENT.  Time =     0.01 sec
Verification failed for at least 1 outputs:  s
Output s: Value in Network1 = 1. Value in Network2 = 0.
Input pattern:  a=1 b=0 cin=0
abc 03>
```

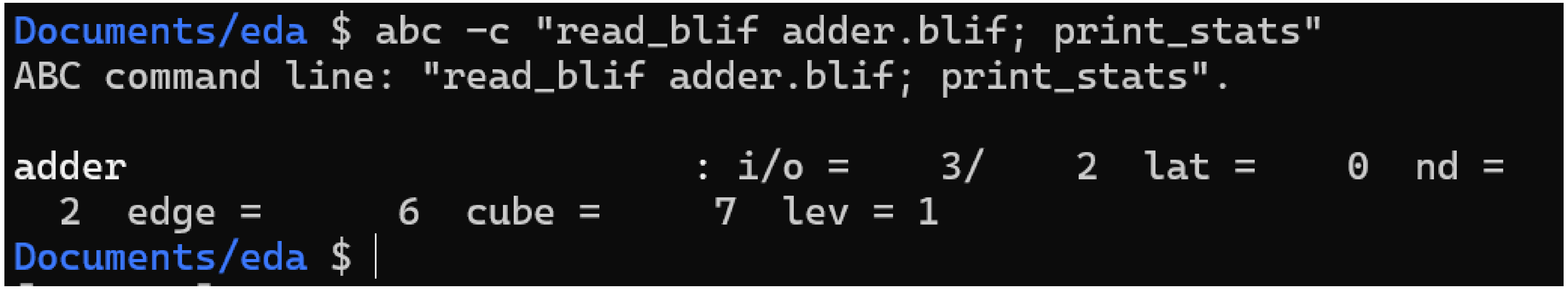
PROGRAMMING WITH ABC

Use ABC as an External Package

- For top-level tasks, you can call abc from command line or other program if…
  - you are using other programming language
  - you want to develop your program from scratch
- For C/C++, abc can also be compiled as a static library
  - type "make –libabc.a" in the source code directory

Use ABC as an External Package

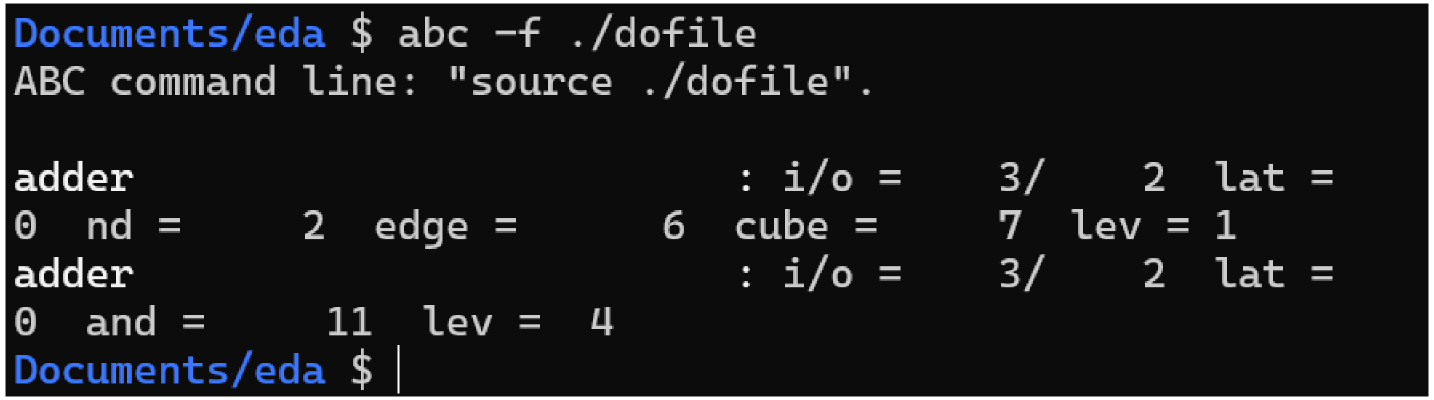
- Execute commands
  - *"abc –c <cmd>"* can execute *<cmd>* directly
  - *<cmd>* can be a string with multiple commands separated by ;



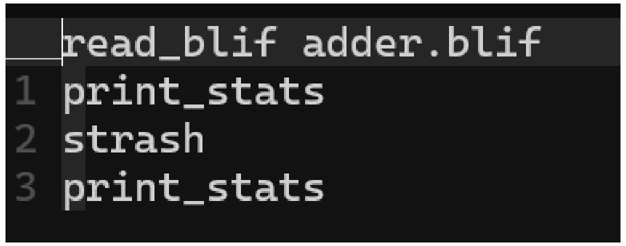
```
Documents/eda $ abc -c "read_blif adder.blif; print_stats"
ABC command line: "read_blif adder.blif; print_stats".

adder                          : i/o =    3/    2  lat =    0  nd =
  2  edge =       6  cube =      7  lev = 1
Documents/eda $
```

Use ABC as an External Package

- Execute scripts (dofiles)
  - *"abc –f <dofile>"* executes the commands in *<dofile>*



```
Documents/eda $ abc -f ./dofile
ABC command line: "source ./dofile".

adder                         : i/o =    3/    2  lat =
0  nd =      2  edge =      6  cube =     7  lev = 1
adder                         : i/o =    3/    2  lat =
0  and =      11  lev =  4
Documents/eda $ 
```



```
read_blif adder.blif
print_stats
strash
print_stats
```

Adding Your Own Commands

- Creating an external package
  - It works without the need to change other parts of ABC
  - You can then use the data structures and functions defined in ABC and other packages

Create External Package in ABC

- First, create a directory named “ext…” under “./src” for your package
  - The makefile of abc looks for any directory under “./src” whose name starts with “ext”

Create External Package in ABC (cont.)

- Under the created directory, say “ext_eda”, create a file named “module.make” and the .c/.cpp files you need.
- In “module.make”, type:

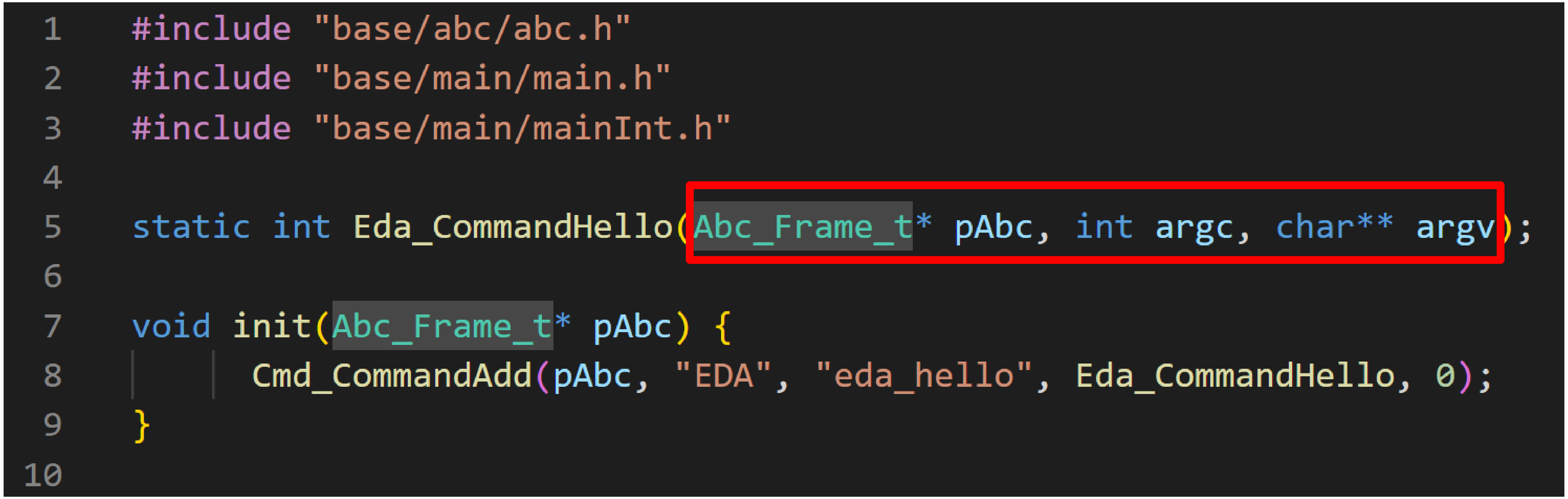
```
SRC += src/ext_eda/file1.cpp \
       src/ext_eda/file2.cpp \
       …
       src/ext_eda/file3.cpp
```

Create External Package in ABC (cont.)

- In one of your .cpp files, you have to register your package and commands

Create External Package in ABC (cont.)

- Each command function should take exactly these three arguments
- Commands are registered in this init() function

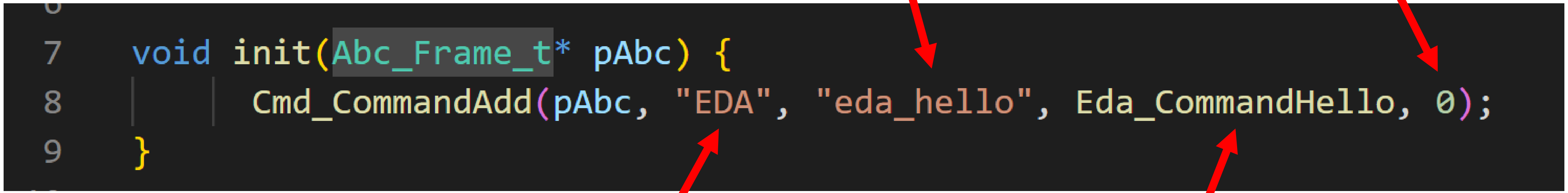


```c
#include "base/abc/abc.h"
#include "base/main/main.h"
#include "base/main/mainInt.h"

static int Eda_CommandHello(Abc_Frame_t* pAbc, int argc, char** argv);

void init(Abc_Frame_t* pAbc) {
        Cmd_CommandAdd(pAbc, "EDA", "eda_hello", Eda_CommandHello, 0);
}

```

Create External Package in ABC (cont.)

Will this function change the current network?

The string to call this command

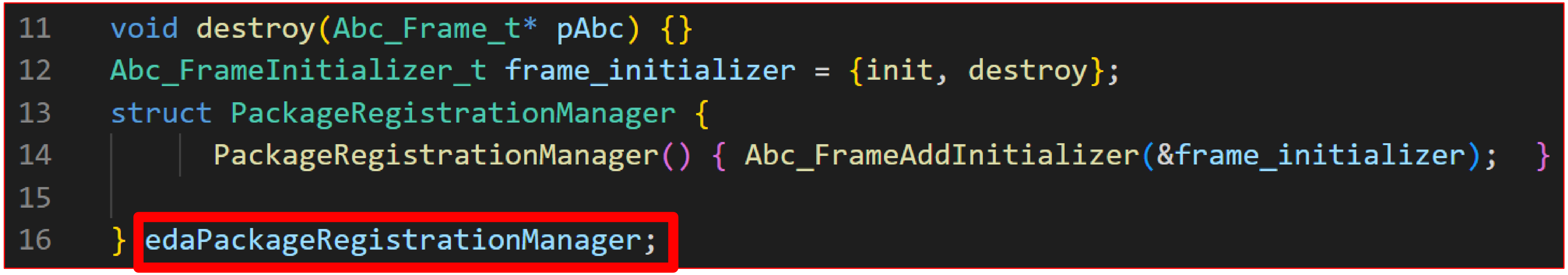


```
void init(Abc_Frame_t* pAbc) {
        Cmd_CommandAdd(pAbc, "EDA", "eda_hello", Eda_CommandHello, 0);
}
```

Group of your command (shown in *help*)

The function that implements the command

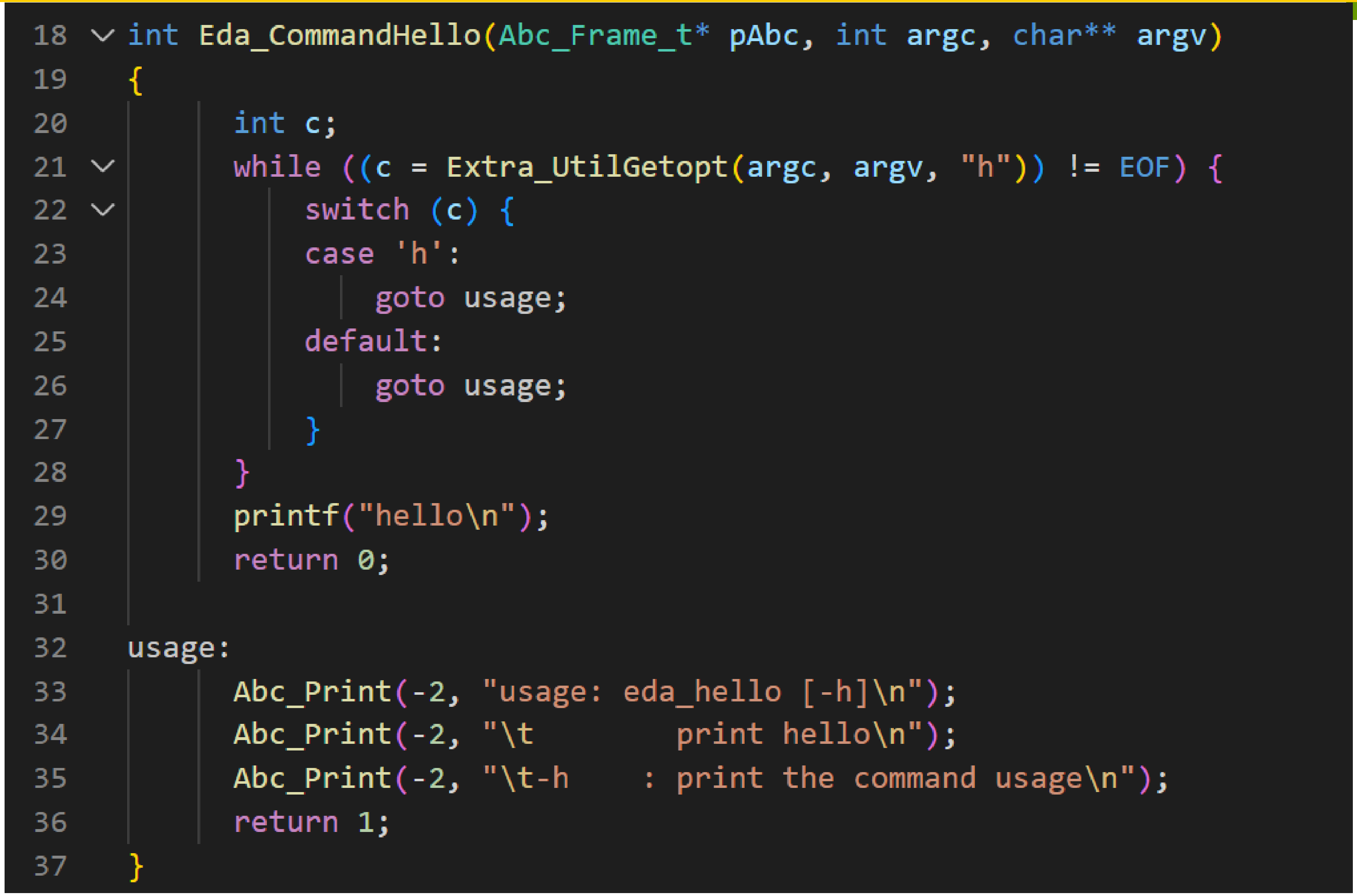
Create External Package in ABC (cont.)



```
void destroy(Abc_Frame_t* pAbc) {}
Abc_FrameInitializer_t frame_initializer = {init, destroy};
struct PackageRegistrationManager {
    PackageRegistrationManager() { Abc_FrameAddInitializer(&frame_initializer);  }

} edaPackageRegistrationManager;
```

Just a variable name, can be anything

Create External Package in ABC(cont.)



```
int Eda_CommandHello(Abc_Frame_t* pAbc, int argc, char** argv)
{
        int c;
        while ((c = Extra_UtilGetopt(argc, argv, "h")) != EOF) {
            switch (c) {
            case 'h':
                goto usage;
            default:
                goto usage;
            }
        }
        printf("hello\n");
        return 0;

usage:
        Abc_Print(-2, "usage: eda_hello [-h]\n");
        Abc_Print(-2, "\t         print hello\n");
        Abc_Print(-2, "\t-h    : print the command usage\n");
        return 1;
}
```

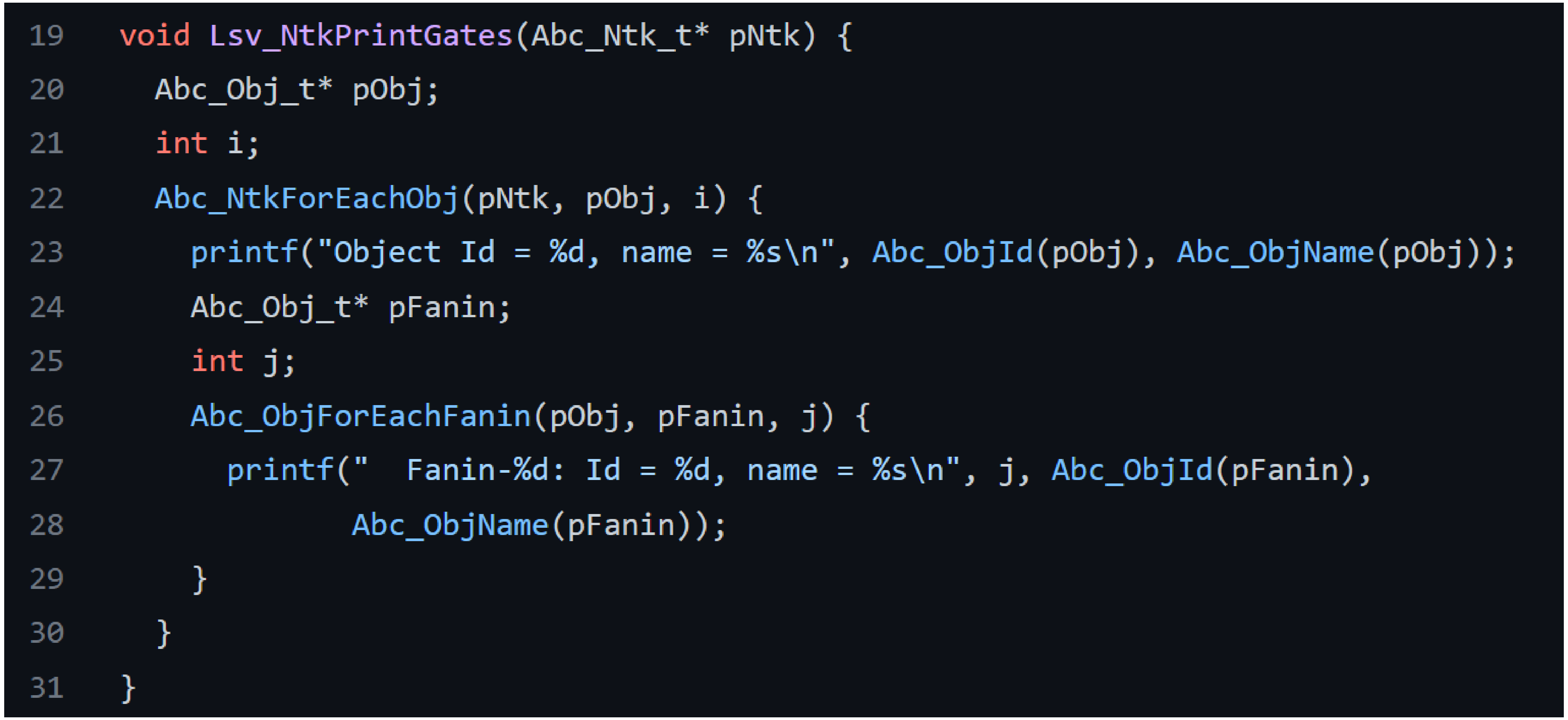
Parse options

Do anything you want

Print usage

Example Code on GitHub

- This example can be found here



```c
void Lsv_NtkPrintGates(Abc_Ntk_t* pNtk) {
  Abc_Obj_t* pObj;
  int i;
  Abc_NtkForEachObj(pNtk, pObj, i) {
    printf("Object Id = %d, name = %s\n", Abc_ObjId(pObj), Abc_ObjName(pObj));
    Abc_Obj_t* pFanin;
    int j;
    Abc_ObjForEachFanin(pObj, pFanin, j) {
      printf("  Fanin-%d: Id = %d, name = %s\n", j, Abc_ObjId(pFanin),
             Abc_ObjName(pFanin));
    }
  }
}
```

Example Code

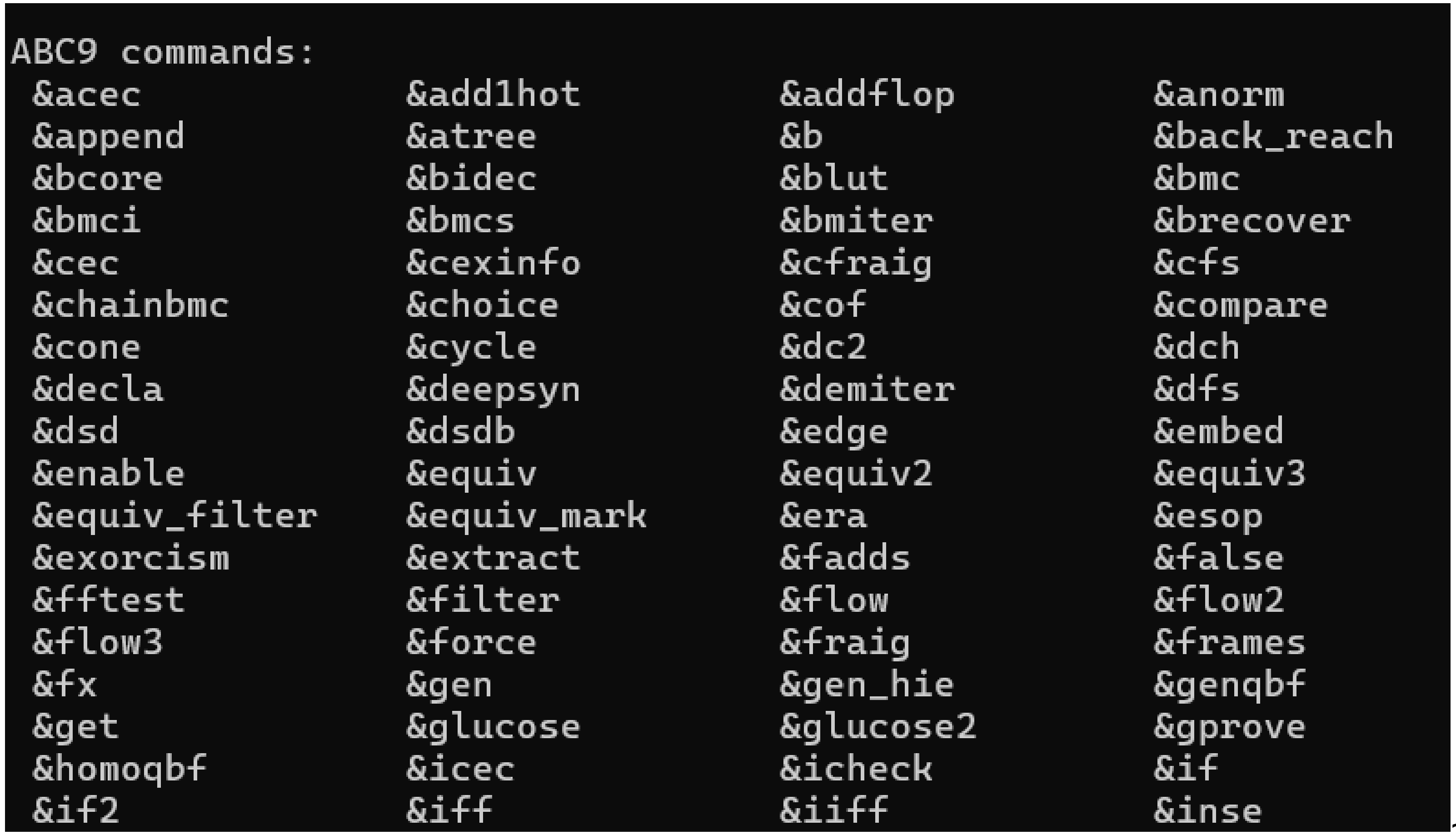
```
#include "base/abc/abc.h"
#include "base/main/main.h"
#include "base/main/mainInt.h"

static int Eda_CommandHello(Abc_Frame_t* pAbc, int argc, char** argv);

void init(Abc_Frame_t* pAbc) {
        Cmd_CommandAdd(pAbc, "EDA", "eda_hello", Eda_CommandHello, 0);
}

void destroy(Abc_Frame_t* pAbc) {}
Abc_FrameInitializer_t frame_initializer = {init, destroy};
struct PackageRegistrationManager {
        PackageRegistrationManager() { Abc_FrameAddInitializer(&frame_initializer);  }

} edaPackageRegistrationManager;

int Eda_CommandHello(Abc_Frame_t* pAbc, int argc, char** argv)
{
        int c;
        while ((c = Extra_UtilGetopt(argc, argv, "h")) != EOF) {
                switch (c) {
                case 'h':
                        goto usage;
                default:
                        goto usage;
                }
        }
        printf("hello\n");
        return 0;

usage:
        Abc_Print(-2, "usage: eda_hello [-h]\n");
        Abc_Print(-2, "\t        print hello\n");
        Abc_Print(-2, "\t-h    : print the command usage\n");
        return 1;
}
```

Data Structure in ABC

- *Abc_Frame_t*
  - command registration, storing the current network, etc
- *Abc_Ntk_t*
  - Model a network
- *Abc_Obj_t*
  - Model a gate (PI, PO, node, constant, latch)
- See *"src/base/abc/abc.h"* for more details
- A simple AIG package "src/aig/gia"

Data Structure in ABC

- A simple AIG package "src/aig/gia"
  - &r, &w
  - &ps
  - &fraig
  - &cec
  - &dc2



```
ABC9 commands:
 &acec              &add1hot           &addflop           &anorm
 &append            &atree             &b                 &back_reach
 &bcore             &bidec             &blut              &bmc
 &bmci              &bmcs              &bmiter            &brecover
 &cec               &cexinfo           &cfraig            &cfs
 &chainbmc          &choice            &cof               &compare
 &cone              &cycle             &dc2               &dch
 &decla             &deepsyn           &demiter           &dfs
 &dsd               &dsdb              &edge              &embed
 &enable            &equiv             &equiv2            &equiv3
 &equiv_filter      &equiv_mark        &era               &esop
 &exorcism          &extract           &fadds             &false
 &fftest            &filter            &flow              &flow2
 &flow3             &force             &fraig             &frames
 &fx                &gen               &gen_hie           &genqbf
 &get               &glucose           &glucose2          &gprove
 &homoqbf           &icec              &icheck            &if
 &if2               &iff               &iiff              &inse
```

Programming in ABC

- For network and nodes, many inline functions are provided for the basic iteration, access, modification, etc.
  - Can be found in *"src/base/abc/abc.h"*
- You can also refer to the source code of other built-in commands
  - most commands can be traced from *"./src/base/abci/abc.c"*

OTHER TOOLS

Logic Synthesis & Verification

- Yosys [4](Yosys Open SYnthesis Suite)
  - Stronger Verilog parser (for problem A)
  - well-documented
- PyVerilog [3](Python package)

Physical Design (Floorplanning & Placement)

- The OpenROAD Project [1]: an integrated chip physical design tool
  - Floorplanning
  - Placement
  - Timing optimization, verification, and analysis
  - Routing

Physical Design (Floorplanning & Placement)

- Stand-alone 2D analytical placers
  - NTUPlace3 [5]
  - Coloquinte_placement [6]
  - RePlAce [7]
  - DREAMPlace [8]
  - …

Solvers

- SAT solver
  - Kisssat [9]
  - Minisat [10]
  - …
- ILP/MIP solver
  - lp_solve (C/C++)
  - Glpk
  - …

Reference

[1] "The OpenROAD Project." https://theopenroadproject.org/

[2] Brayton, R.; and Mishchenko, A. 2010. "ABC: An Academic Industrial-Strength Verification Tool." In *Proceedings of International Conference on Computer Aided Verification*

[3] Takamaeda-Yamazaki Shinya. 2015. "PyVerilog: A Python-based hardware design processing toolkit for verilog HDL." In *Applied Reconfigurable Computing.*

[4] Clifford Wolf. "Yosys open synthesis suite." https://github.com/YosysHQ/yosys

[5] T. -C. Chen, Z. -W. Jiang, T. -C. Hsu, H. -C. Chen and Y. -W. Chang, "NTUplace3: An Analytical Placer for Large-Scale Mixed-Size Designs With Preplaced Blocks and Density Constraints." In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*

Reference (Cont.)

[6] Gabriel Gouvine, 2014. “Coriolis Etesian: an analytical VLSI placer.” https://github.com/Coloquinte/Coloquinte_placement

[7] C.-K. Cheng, A. B. Kahng, I. Kang and L. Wang, 2018. "RePlAce: Advancing Solution Quality and Routability Validation in Global Placement.“ In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.*

[8] Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany and David Z. Pan, 2019. “DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement.” In *Proceedings of Design Automation Conference*

[9] Armin Biere, Katalin Fazekas, Mathias Fleury, Maximillian Heisinger. 2020. “Kissat SAT Solver” http://fmv.jku.at/kissat/

[10] Niklas S¨orensson, Niklas Een, 2013. “Minisat: A minimalistic and high-performance SAT solver” https://github.com/niklasso/minisat