Getting started with ABC: A System for Sequential Synthesis and Verification

Ana Petkovska
ana.petkovska@epfl.ch
EPFL – IC – LAP

Lausanne, 28.02.2016

Outline

- **Introduction**
- Download, Install, Use
- Inside ABC
- Create Your First Command

What is ABC?

- ABC is a growing software system for synthesis and verification of binary sequential logic circuits appearing in synchronous hardware designs.
- ABC combines scalable logic optimization based on And-Inverter Graphs (AIGs), optimal-delay DAG-based technology mapping for look-up tables and standard cells, and innovative algorithms for sequential synthesis and verification.
- You can find more information at http://www.eecs.berkeley.edu/~alanmi/abc/

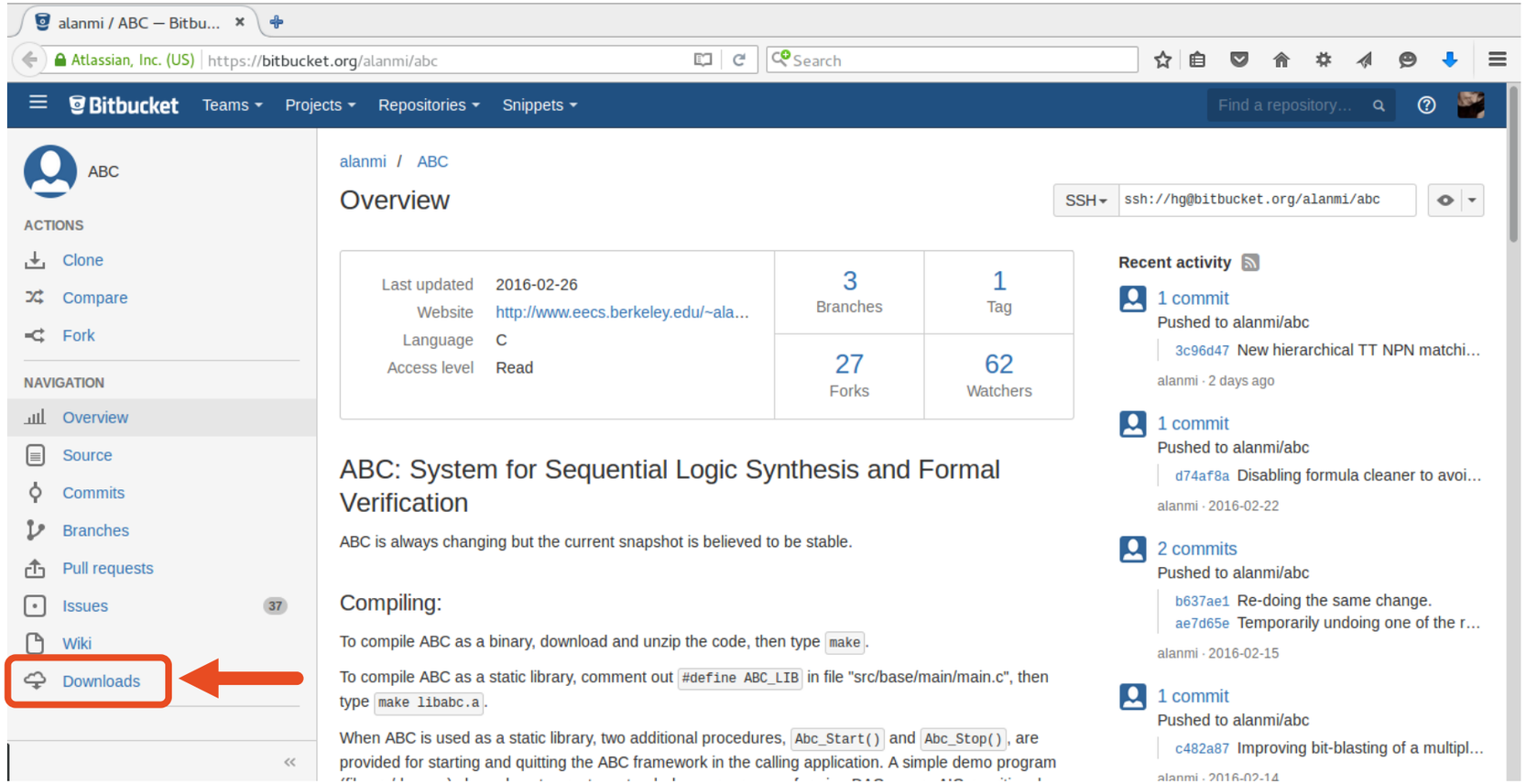
Source http://www.eecs.berkeley.edu/~alanmi/abc/

Outline

- Introduction
- **Download, Install, Use**
- Inside ABC
- Create Your First Command

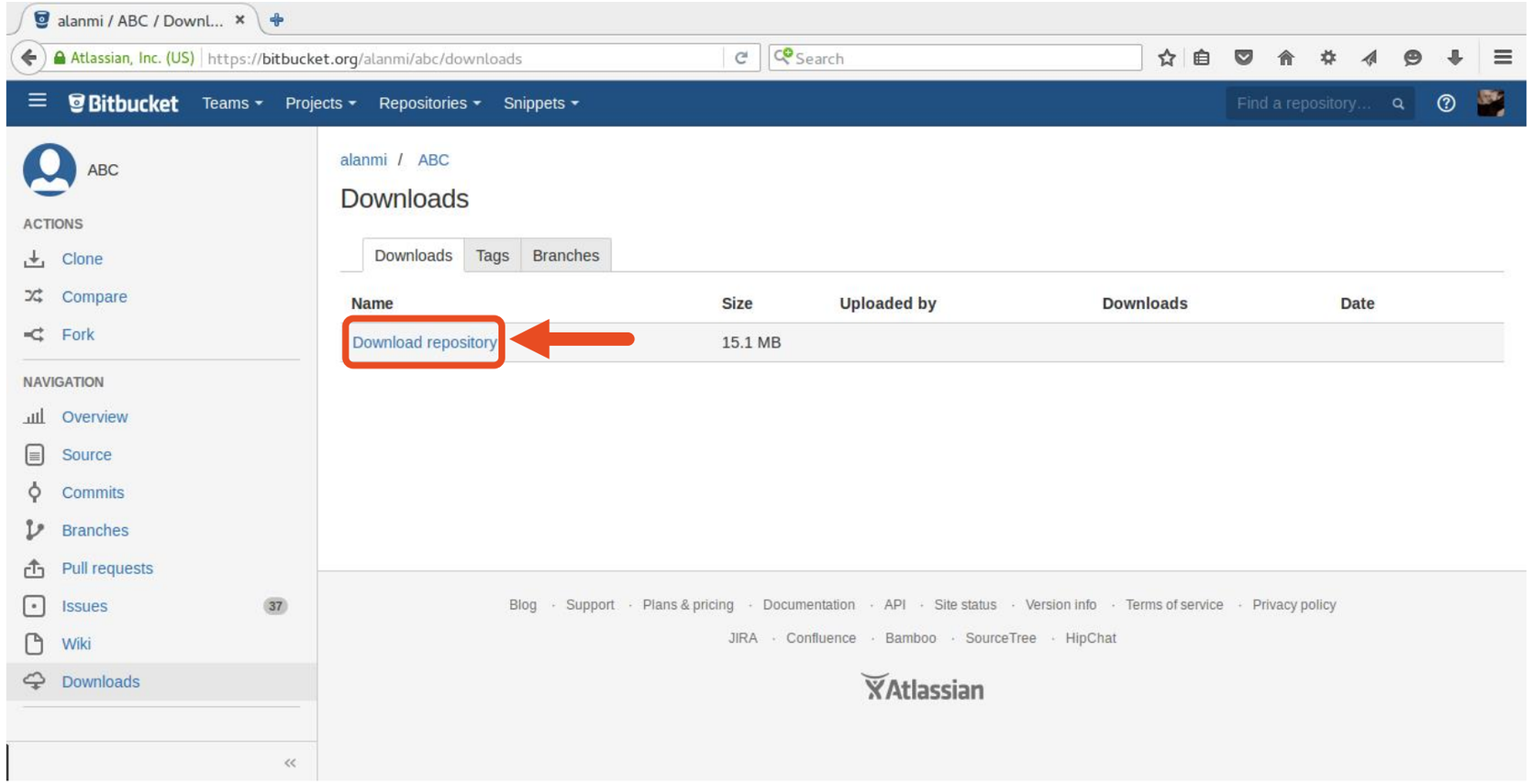
Get ABC [1]

- The latest version of ABC can be downloaded from https://bitbucket.org/alanmi/abc



Get ABC [2]

- The latest version of ABC can be downloaded from https://bitbucket.org/alanmi/abc



Install ABC

- Compile ABC as a binary
  - download and unzip the code
  - go in the directory and type **make**

```
ana@E6330:~$ cd abc/
ana@E6330:~/abc$ make
```

- This executes the Makefile [**abc/Makefile**]
- If you want to delete all intermediate and output files created in the compilation process type **make clean** (Do not do it now!)

Install ABC

- If the compilation process ends successfully you will get

```
`` Compiling: /src/aig/hop/hopTable.c
`` Compiling: /src/aig/hop/hopTruth.c
`` Compiling: /src/aig/hop/hopUtil.c
`` Building binary: abc
ana@E6330:~/abc$
```

- If there are errors in the compilation process then read the “Troubleshooting” section in the **readme** file [**abc/readme.md**]

Use ABC

- After successful compilation, you can call the command line interpreter of ABC

```
ana@E6330:~/abc$ ./abc
UC Berkeley, ABC 1.01 (compiled Apr 29 2013 10:32:22)
abc 01>
```

where you can execute commands implemented into ABC

Short Example

- Create a new folder called **examples**
- Copy the following Verilog code in a file named **rca2.v**

```
module rca2 (a0, b0, a1, b1, s0, s1, s2);
//-------------Input Ports Declarations----------------------------
input a0, b0, a1, b1;
//-------------Output Ports Declarations----------------------------
output s0, s1, s2;
//-------------Wires-----------------------------------------------
wire c0;
//-------------Logic-----------------------------------------------
assign s0 = a0 ^ b0 ;
assign c0 = a0 & b0 ;
assign s1 = a1 ^ b1 ^ c0;
assign s2 = (a1 & b1) | (c0 & (a1 ^ b1));
endmodule
```

Short Example

- Go in the directory **abc/examples**
- Start ABC by typing **../abc**
- Read the circuit defined in **rca2.v** using the command **read** (type **read rca2.v**)
- Do structural hashing using the command **strash**
- Print the information about the circuit using the command **print_stats**

```
ana@E6330:~/abc$ cd examples/
ana@E6330:~/abc/examples$ ../abc
UC Berkeley, ABC 1.01 (compiled Apr 29 2013 10:32:22)
abc 01> read rca2.v
abc 02> strash
abc 03> print_stats
rca2          : i/o =    4/    3  lat =    0  and =     13  lev =  4
abc 03> quit
ana@E6330:~/abc/examples$
```

Commands

- You can find short description of part of the existing commands at

  http://www.eecs.berkeley.edu/~alanmi/abc/

  in the section "Command summary"
- You can use aliases defined in the **abc.rc** file [**abc/abc.rc**]
  - For example, instead of **read rca2.v** you can write **r rca2.v**
  - You can also define your own aliases there

Outline

- Introduction
- Download, Install, Use
- **Inside ABC**
- Create Your First Command

Inside ABC

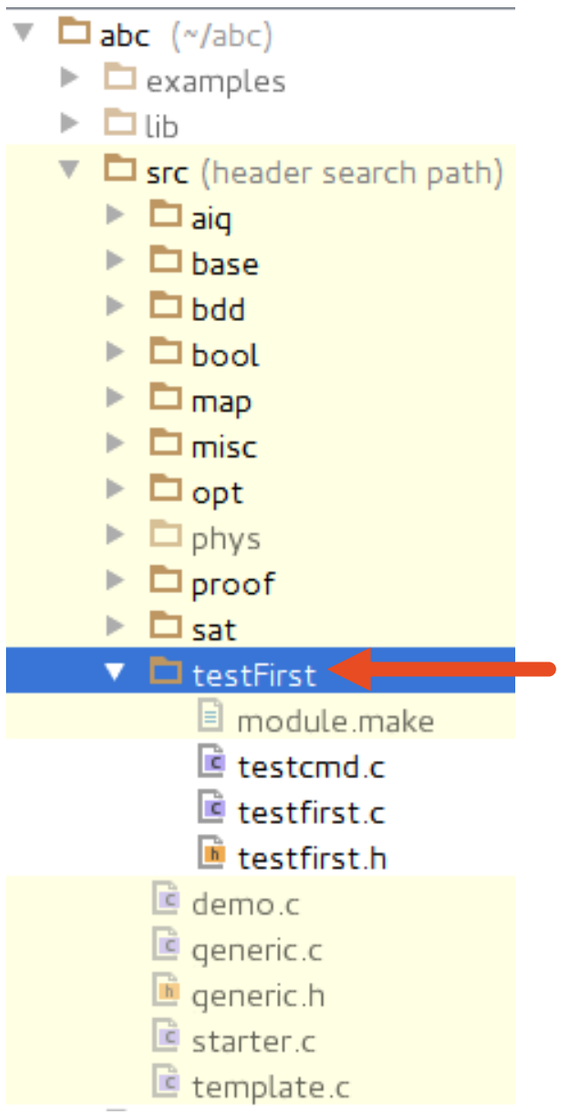
- Most of the implemented commands are defined in the following files
  - **src/base/abci/abc.c**
  - **src/base/io/io.c**
- The declarations of the basic commands for working with ABC networks can be found in
  - **src/base/abc/abc.h**

Outline

- Introduction
- Download, Install, Use
- Inside ABC
- **Create Your First Command**

Create a new command

- In the folder **src** create new folder **testFirst** with the following files
  - **module.make** where you will list your .c files for compilation
  - **testcmd.c** where you will declare and define your commands
  - **testfirst.c** where you will define your main functions
  - **testfirst.h** where you will declare your main functions



File module.make

- In this file you should list your .c files for compilation; thus it has the following content

testFirst/module.make

```
SRC +=  src/testFirst/testcmd.c \
        src/testFirst/testfirst.c
```

- Also, you should list your folder as new module in the **Makefile** [**abc/Makefile**]

Makefile

```

CC   := gcc
CXX  := g++
LD   := $(CXX)

MSG_PREFIX ?=

$(info $(MSG_PREFIX)Using CC=$(CC))
$(info $(MSG_PREFIX)Using CXX=$(CXX))
$(info $(MSG_PREFIX)Using LD=$(LD))

PROG := abc

MODULES := \
    $(wildcard src/ext*) \
    src/testFirst \
    src/base/abc src/base/abci src/base/cmd src/base/io src/base/main \
    src/base/ver src/base/wlc src/base/bac src/base/cba src/base/pla src/base/test \
```

File testfirst.c [1]

- It is a good practice to start all files with information about the file itself.
- Then, we list the needed libraries and the declarations of the functions that are defined in the file and are used just in this file.

testfirst.c ×

```c
/**CFile***********************************************************************

  FileName    [testfirst.c]

  SystemName  [ABC: Logic synthesis and verification system.]

  PackageName [Getting Started with ABC.]

  Synopsis    [Main functions for our new commands.]

  Author      [Ana Petkovska]

  Affiliation [EPFL IC LAP]

  Date        [Ver. 1.0. Started - February 28, 2016.]

  Revision    []

***********************************************************************/

#include "base/main/main.h"

ABC_NAMESPACE_IMPL_START

////////////////////////////////////////////////////////////////////////
///                        DECLARATIONS                            ///
////////////////////////////////////////////////////////////////////////

int TestFirst_FirstFunction(Abc_Ntk_t * pNtk);

```

File testfirst.c [2]

- Next, we define the function that will be called from our command.
- It extracts the network that is read into ABC and calls another function.

testfirst.c

```c

////////////////////////////////////////////////////////////////////////
///                      FUNCTION DEFINITIONS                        ///
////////////////////////////////////////////////////////////////////////

/**Function*************************************************************

  Synopsis    [The function for our first command.]

  Description [Extracts the ABC network and executes the main function for the command.]

  SideEffects []

  SeeAlso     []

***********************************************************************/
int TestFirst_FirstFunctionAbc(Abc_Frame_t * pAbc) {
    Abc_Ntk_t * pNtk;
    int result;

    // Get the network that is read into ABC
    pNtk = Abc_FrameReadNtk(pAbc);

    if(pNtk == NULL) {
        Abc_Print(-1, "TestFirst_FirstFunctionAbc: Getting the target network has failed.\n");
        return 0;
    }

    // Call the main function
    result = TestFirst_FirstFunction(pNtk);

    return result;
}

```

File testfirst.c [3]

- Finally, we define a function that use the network and prints information about it.

testfirst.c

```c
/**Function*************************************************************

  Synopsis    [Main function for our first command.]

  Description [Prints information for a structurally hashed network.]

  SideEffects []

  SeeAlso     []

***********************************************************************/
int TestFirst_FirstFunction(Abc_Ntk_t * pNtk) {
    // check if the network is strashed
    if(!Abc_NtkIsStrash(pNtk)){
        Abc_Print(-1, "TestFirst_FirstFunction: This command is only applicable to strashed networks.\n");
        return 0;
    }

    // print information about the network
    Abc_Print(1, "The network with name %s has:\n", Abc_NtkName(pNtk));
    Abc_Print(1, "\t- %d primary inputs;\n", Abc_NtkPiNum(pNtk));
    Abc_Print(1, "\t- %d primary outputs;\n", Abc_NtkPoNum(pNtk));
    Abc_Print(1, "\t- %d AND gates.\n", Abc_NtkNodeNum(pNtk));

    return 1;
}

////////////////////////////////////////////////////////////////////////
///                       END OF FILE                              ///
////////////////////////////////////////////////////////////////////////

ABC_NAMESPACE_IMPL_END
```

File testfirst.h

- In this file we declare the functions from our module that can be globally used.

testfirst.h

```
21  #ifndef TESTFIRST_h
22  #define TESTFIRST_h
23
24  ////////////////////////////////////////////////////////////////////////
25  ///                          INCLUDES                              ///
26  ////////////////////////////////////////////////////////////////////////
27
28  #include "base/main/main.h"
29
30  ////////////////////////////////////////////////////////////////////////
31  ///                        PARAMETERS                              ///
32  ////////////////////////////////////////////////////////////////////////
33
34  ABC_NAMESPACE_HEADER_START
35
36  ////////////////////////////////////////////////////////////////////////
37  ///                        BASIC TYPES                             ///
38  ////////////////////////////////////////////////////////////////////////
39
40  ////////////////////////////////////////////////////////////////////////
41  ///                    FUNCTION DECLARATIONS                       ///
42  ////////////////////////////////////////////////////////////////////////
43
44  /*==================================================================*/
45  /*=== testfirst.c ==================================================*/
46  /*==================================================================*/
47  extern int TestFirst_FirstFunctionAbc(Abc_Frame_t * pAbc);
48  /*==================================================================*/
49
50  #endif
51
52  ABC_NAMESPACE_HEADER_END
53
```

File testcmd.c [1]

- First, we list the needed libraries and the declarations of the functions that define the commands.
- Next, we include one initialization function for module initialization and for inserting the command in the system.

testcmd.c

```c
#include "base/main/main.h"
#include "testfirst.h"

ABC_NAMESPACE_IMPL_START

////////////////////////////////////////////////////////////////////////
///                       DECLARATIONS                             ///
////////////////////////////////////////////////////////////////////////

static int TestFirst_CommandTestFirst(Abc_Frame_t * pAbc, int argc, int ** argv);

////////////////////////////////////////////////////////////////////////
///                     FUNCTION DEFINITIONS                       ///
////////////////////////////////////////////////////////////////////////

/**Function*************************************************************

  Synopsis    [Package initialisation procedure.]

  Description []

  SideEffects []

  SeeAlso     []

***********************************************************************/
void TestFirst_Init(Abc_Frame_t * pAbc) {
    Cmd_CommandAdd(pAbc, "Various", "firstcmd", TestFirst_CommandTestFirst, 0);
}
```

File testcmd.c [2]

- Next, we give the definitions of the functions that implement our commands.

testcmd.c

```c
int TestFirst_CommandTestFirst(Abc_Frame_t * pAbc, int argc, int ** argv) {
    int fVerbose;
    int c, result;

    // set defaults
    fVerbose = 0;

    // get arguments
    Extra_UtilGetoptReset();
    while ((c = Extra_UtilGetopt(argc, argv, "vh")) != EOF) {
        switch (c) {
            case 'v':
                fVerbose ^= 1;
                break;
            case 'h':
                goto usage;
            default:
                goto usage;
        }
    }

    // call the main function
    result = TestFirst_FirstFunctionAbc(pAbc);

    // print verbose information if the verbose mode is on
    if (fVerbose) {
        Abc_Print(1, "\nVerbose mode is on.\n");
        if (result)
            Abc_Print(1, "The command finished successfully.\n");
        else Abc_Print(1, "The command execution has failed.\n");
    }

    return 0;
usage:
    Abc_Print(-2, "usage: firstcmd [-vh] \n");
    Abc_Print(-2, "\t        Our first command in ABC. It prints information about the function read into ABC\n");
    Abc_Print(-2, "\t-v     : toggle printing verbose information [default = %s]\n", fVerbose ? "yes" : "no");
    Abc_Print(-2, "\t-h     : print the command usage\n");
    return 1;
}
```

File src/base/main/mainInit.c

- The file **src/base/main/mainInit.c** is the only file from the source code of ABC that we need to change.
- To include our command in the system and to initialize it, we use the previously mentioned initialization function in the function **Abc_FrameInit()** for what we add the two lines shown below.

mainInit.c

```
extern void Abc85_Init( Abc_Frame_t * pAbc );
extern void Abc85_End( Abc_Frame_t * pAbc );
extern void TestFirst_Init(Abc_Frame_t * pAbc);
```

mainInit.c

```
**********************************************************
void Abc_FrameInit( Abc_Frame_t * pAbc )
{
    Abc_FrameInitializer_t* p;
    Cmd_Init( pAbc );
    Cmd_CommandExecute( pAbc, "set checkread" );
    Io_Init( pAbc );
    Abc_Init( pAbc );
    If_Init( pAbc );
    Map_Init( pAbc );
    Mio_Init( pAbc );
    Super_Init( pAbc );
    Libs_Init( pAbc );
    Load_Init( pAbc );
    Scl_Init( pAbc );
    Wlc_Init( pAbc );
    Bac_Init( pAbc );
    Cba_Init( pAbc );
    Pla_Init( pAbc );
    Test_Init( pAbc );
    TestFirst_Init(pAbc);
    for( p = s_InitializerStart ; p ; p = p->next )
        if(p->init)
            p->init(pAbc);
}
```

Test the new command

- To test the new command recompile ABC by typing **make**
- Then, start ABC, read a network and test the command

```
ana@E6330:~/abc$ make
`` Dependency: /src/base/main/mainInit.c
`` Dependency: /src/testFirst/testfirst.c
`` Dependency: /src/testFirst/testcmd.c
`` Compiling: /src/testFirst/testcmd.c
`` Compiling: /src/testFirst/testfirst.c
`` Compiling: /src/base/main/mainInit.c
`` Building binary: abc
ana@E6330:~/abc$ cd examples/
ana@E6330:~/abc/examples$ ../abc
UC Berkeley, ABC 1.01 (compiled Apr 29 2013 16:30:16)
abc 01> read rca2.v
abc 02> firstcmd
Error: TestFirst_FirstFunction: This command is only applicable to strashed networks.
abc 02> strash
abc 03> firstcmd -v
The network with name rca2 has:
        - 4 primary inputs;
        - 3 primary outputs;
        - 13 AND gates.

Verbose mode is on.
The command finished successfully.
abc 03>
```

Recommended reading

- **“Going Places with ABC”** - a presentation that introduces basics of programming in ABC:
- **“Quick Look under the Hood of ABC: A Programmer’s Manual”** - a paper that gives an overview of the ABC programming environment
- **“Constructing AIGs in ABC: A Tutorial”** - a write-up that shows how to programmably construct AIG in the user’s application
- All documents are available at
  http://www.eecs.berkeley.edu/~alanmi/abc/
  under the section “Programming notes”