ABC : A System for Sequential Synthesis and Verification Lesson 101

Presenter: Nian-Ze Lee

Instructor: Jie-Hong Roland Jiang

ALCom Lab

EE Dept./ Grad. Inst. of Electronics Eng.

National Taiwan University

Before we start…

- Recommended LINUX commands / tools:
  - ctag / cscope
  - grep
  - man
- Version “abc70930” used for the following slides
  - abc70930

Outline

- Programming with ABC
- Basic data structure in ABC
- Case study: the cec command

PROGRAMMING WITH ABC

Programming with ABC (1/8)

- Use ABC as a static library
  - see readme for compiling ABC as a library
  - see demo.c for an example
- Plug in your own packages in ABC
  - recommended style
  - need to understand how commands are registered in ABC

Programming with ABC (2/8)

- step 1: create a directory “eda" in the src directory

```
nianze@ubuntu-ibmx3500m4:~/EDA/abc70930$ cd src/
nianze@ubuntu-ibmx3500m4:~/EDA/abc70930/src$ ls
aig  base  bdd  eda  generic.c  generic.h  map  misc  opt  sat
nianze@ubuntu-ibmx3500m4:~/EDA/abc70930/src$
```

Programming with ABC (3/8)

- step 2: create a eda.c file as the command file for this package
  - define your own command
  - copy src/base/abci/abc.c as a template

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

  FileName    [eda.c]

  SystemName  [ABC: Logic synthesis and verification system.]

  PackageName [Package for demo how to plug in a package.]

  Synopsis    [Command file.]

  Author      [Nian-Ze Lee]

  Affiliation [NTU]

  Date        [June 3 , 2016.]

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

#include "eda.h"
#include "mainInt.h"

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

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

```

Programming with ABC (4/8)

- step 3: create two functions
  - Eda_Init() : register commands in this package
  - Eda_End() : shut down this package

```
32 /**Function*************************************************************
33
34   Synopsis    [Start / Stop the eda package]
35
36   Description []
37
38   SideEffects []
39
40   SeeAlso     []
41
42 ******************************************************************************/
43
44 void
45 Eda_Init( Abc_Frame_t * pAbc )
46 {
47    Cmd_CommandAdd( pAbc , "z EDA" , "hello" , EdaCommandHello , 0 );
48 }
49
50 void
51 Eda_End()
52 {
53 }
```

Programming with ABC (5/8)

- A typical command in ABC contains:
  - option flags
  - parse options
  - execution
  - usage

```
67 int
68 EdaCommandHello( Abc_Frame_t * pAbc , int argc , char ** argv )
69 {
70     int fVerbose , c;
71
72     fVerbose = 0;
73     Extra_UtilGetoptReset();
74     while ( ( c = Extra_UtilGetopt( argc, argv, "vh" ) ) != EOF )
75     {
76        switch ( c )
77        {
78           case 'v':
79              fVerbose ^= 1;
80              break;
81           case 'h':
82              goto usage;
83           default:
84              goto usage;
85        }
86     }
87
88     Eda_SayHello( fVerbose );
89     return 0;
90
91 usage:
92     fprintf( pAbc->Err, "usage: hello [-vh]\n" );
93     fprintf( pAbc->Err, "\t         Let ABC say hello to everyone\n" );
94     fprintf( pAbc->Err, "\t-v      : toggle verbose hello [default = %s]\n", fVerbose ? "yes":"no" );
95     fprintf( pAbc->Err, "\t-h      : prints the command summary\n" );
96     return 1;
97 }
```

Programming with ABC (6/8)

- In the function Abc_FrameInit() (src/base/main/mainInit.c), add
  - Eda_Init() to initialize the package
- In the function Abc_FrameEnd() (src/base/main/mainInit.c), add
  - Eda_End() to stop the package

Programming with ABC (7/8)

- In Makefile
  - add src/eda to MODULES
- Create a file "module.make" in src/eda
  - SRC += src/eda/eda.c \
    src/eda/edaHello.c \
    … (more files to compile)

Programming with ABC (8/8)

- Good practices
  - use eda.h to record all functions in this package

```
44 ////////////////////////////////////////////////////////////////////////
45 ///                      FUNCTION DECLARATIONS                       ///
46 ////////////////////////////////////////////////////////////////////////
47
48 /*=== edaHello.c ========================================================*/
49 extern void Eda_SayHello( int );
```

BASIC DATA STRUCTURE IN ABC

Overview of ABC data structure

- Abc_Frame_t (src/base/main/mainInt.h)
  - top manager :
    - command tables
    - current network
- Abc_Ntk_t (src/base/abc/abc.h)
  - network manager :
    - objects (PI , PO , gates , etc)
    - functionality managers (pManFunc)
- Abc_Obj_t (src/base/abc/abc.h)
  - object type :
    - id , fanin , fanout , etc

Abc_Frame_t

```
44 struct Abc_Frame_t_
45 {
46     // general info
47     char *          sVersion;    // the name of the current version
48     // commands, aliases, etc
49     st_table *      tCommands;   // the command table
50     st_table *      tAliases;    // the alias table
51     st_table *      tFlags;      // the flag table
52     Vec_Ptr_t *     aHistory;    // the command history
53     // the functionality
54     Abc_Ntk_t *     pNtkCur;     // the current network
55     int             nSteps;      // the counter of different network processed
56     int             fAutoexac;   // marks the autoexec mode
57    int            fBatchMode;  // are we invoked in batch mode?
58     // output streams
59     FILE *          Out;
60     FILE *          Err;
61     FILE *          Hst;
62     // used for runtime measurement
63     int             TimeCommand; // the runtime of the last command
64     int             TimeTotal;   // the total runtime of all commands
65     // temporary storage for structural choices
66     Vec_Ptr_t *     vStore;      // networks to be used by choice
67     // decomposition package
68     void *          pManDec;     // decomposition manager
69     DdManager *     dd;          // temporary BDD package
70     // libraries for mapping
71     void *          pLibLut;     // the current LUT library
72     void *          pLibGen;     // the current genlib
73     void *          pLibSuper;   // the current supergate library
74     void *          pLibVer;     // the current Verilog library
75 };
```

Abc_Ntk_t

```
172 struct Abc_Ntk_t_
173 {
174     // general information
175     Abc_NtkType_t     ntkType;       // type of the network
176     Abc_NtkFunc_t     ntkFunc;       // functionality of the network
177     char *            pName;         // the network name
178     char *            pSpec;         // the name of the spec file if present
179     Nm_Man_t *        pManName;      // name manager (stores names of objects)
180     // components of the network
181     Vec_Ptr_t *       vObjs;         // the array of all objects (net, nodes, latches, etc)
182     Vec_Ptr_t *       vPis;          // the array of primary inputs
183     Vec_Ptr_t *       vPos;          // the array of primary outputs
184     Vec_Ptr_t *       vCis;          // the array of combinational inputs  (PIs, latches)
185     Vec_Ptr_t *       vCos;          // the array of combinational outputs (POs, asserts, latches)
186     Vec_Ptr_t *       vPios;         // the array of PIOs
187     Vec_Ptr_t *       vAsserts;      // the array of assertions
188     Vec_Ptr_t *       vBoxes;        // the array of boxes
189     // the number of living objects
190     int               nObjs;         // the number of live objs
191     int nObjCounts[ABC_OBJ_NUMBER];  // the number of objects by type
192     // the backup network and the step number
193     Abc_Ntk_t *       pNetBackup;    // the pointer to the previous backup network
194     int               iStep;         // the generation number for the given network
195     // hierarchy
196     Abc_Lib_t *       pDesign;
197     short             fHieVisited;   // flag to mark the visited network
198     short             fHiePath;      // flag to mark the network on the path
199     // miscellaneous data members
200     int               nTravIds;      // the unique traversal IDs of nodes
201     Extra_MmFixed_t * pMmObj;        // memory manager for objects
202     Extra_MmStep_t *  pMmStep;       // memory manager for arrays
203     void *            pManFunc;      // functionality manager (AIG manager, BDD manager, or memory manager for SOPs)
204 //    Abc_Lib_t *       pVerLib;       // for structural verilog designs
205     Abc_ManTime_t *   pManTime;      // the timing manager (for mapped networks) stores arrival/required times for all nodes
206     void *            pManCut;       // the cut manager (for AIGs) stores information about the cuts computed for the nodes
207     int               LevelMax;      // maximum number of levels
208     Vec_Int_t *       vLevelsR;      // level in the reverse topological order (for AIGs)
209     Vec_Ptr_t *       vSupps;        // CO support information
210     int *             pModel;        // counter-example (for miters)
211     Abc_Ntk_t *       pExdc;         // the EXDC network (if given)
212     void *            pData;         // misc
213     Abc_Ntk_t *       pCopy;
214     Hop_Man_t *       pHaig;         // history AIG
215     // node attributes
216     Vec_Ptr_t *       vAttrs;        // managers of various node attributes (node functionality, global BDDs, etc)
217 };
218
```

Abc_Obj_t

```
145 struct Abc_Obj_t_ // 12 words
146 {
147     // high-level information
148     Abc_Ntk_t *       pNtk;          // the host network
149     int               Id;            // the object ID
150     int               TravId;        // the traversal ID (if changed, update Abc_NtkIncrementTravId)
151     // internal information
152     unsigned          Type    :  4;  // the object type
153     unsigned          fMarkA  :  1;  // the multipurpose mark
154     unsigned          fMarkB  :  1;  // the multipurpose mark
155     unsigned          fMarkC  :  1;  // the multipurpose mark
156     unsigned          fPhase  :  1;  // the flag to mark the phase of equivalent node
157     unsigned          fExor   :  1;  // marks AIG node that is a root of EXOR
158     unsigned          fPersist:  1;  // marks the persistant AIG node
159     unsigned          fCompl0 :  1;  // complemented attribute of the first fanin in the AIG
160     unsigned          fCompl1 :  1;  // complemented attribute of the second fanin in the AIG
161     unsigned          Level   : 20;  // the level of the node
162     // connectivity
163     Vec_Int_t         vFanins;       // the array of fanins
164     Vec_Int_t         vFanouts;      // the array of fanouts
165     // miscellaneous
166     void *            pData;         // the network specific data (SOP, BDD, gate, equiv class, etc)
167     Abc_Obj_t *       pNext;         // the next pointer in the hash table
168     Abc_Obj_t *       pCopy;         // the copy of this object
169     Hop_Obj_t *       pEquiv;        // pointer to the HAIG node
170 };
171
```

CASE STUDY: THE CEC COMMAND

Combinational Equivalence Checking

- We will learn how ABC performs combinational equivalence checking (CEC)
  - typical command structure
  - AIG operations
  - SAT engine usage
- Both API and operations on internal structures will be covered
  - suggestion: always search for appropriate API before digging into the internal structure!

CEC command usage

- Demo
  - read in c6288 (16-bit multiplier)
  - run “resyn” script (defined in abc.rc)
  - perform equivalence checking on the circuits before and after synthesis by command cec
    - SAT only
    - FRAIG + SAT (much more powerful!)
      - FRAIG: uniqueness of each gate
- ./abc –f c6288.script
- command alias “eda” and “eda2” (abc.rc)

CEC command API (1/3)

- Typical command structure
  - src/base/abci/abc.c
  - parse options
  - read in circuits
    - Io_Read()
    - Abc_NtkPrepareTwoNtks()
  - execution
    - Abc_NtkCecSat()
    - Abc_NtkCecFraig()

CEC command API (2/3)

- SAT only: Abc_NtkCecSat()
  - Abc_NtkMiter() : miter two networks
  - Abc_NtkMulti() : convert the miter into a CNF
  - Abc_NtkMiterSat() : solve the CNF
- FRAIG + SAT: Abc_NtkCecFraig()
- FRAIG:
  - Abc_NtkFraig() : construct a FRAIG network
  - Fraig_NodeAndCanon() : the core procedure of FRAIG

CEC command API (3/3)

- Fraig_NodeAndCanon() : when performing the "AND" operation on two nodes
  - check for trivial cases
  - structural hashing
  - simulation (FEC : functional equivalence candidate)
  - apply SAT solving to FEC
- From PI to PO : such order is important!
  - Why?

AIG operations

- Abc_FrameReadNtk() :
  - read the current network
- Abc_NtkStrash() :
  - convert the network into structural hashed AIG
- In src/base/abc/abc.h :
  - Abc_NtkObjNum() , Abc_NtkPiNum()
  - Abc_NtkCreateObj() , Abc_NtkCreatePi()
  - Abc_ObjId() , Abc_ObjFanin0()
  - Abc_AigAnd() , Abc_AigXor()
  - etc

SAT engine usage

- We will learn how to :
  - convert a network into a CNF formula
  - write the CNF into a SAT solver
  - solve the CNF formula
  - get corresponding variables of a certain gate
  - make assumptions for the SAT solver
    - we can require the output of some gates to be 0/1
- Refer to the command “dsat” for details

SAT engine API

- Abc_NtkToDar() :
  - initialize the AIG manager from Abc_Ntk_t
- Cnf_Derive() :
  - derive the CNF formula
- Cnf_DataWriteIntoSolver() :
  - initialize the SAT solver with the given CNF
- sat_solver_solve()

SAT engine operations

- Cnf_Dat_t :
  - int * pVarNums : map object id to variable id
  - ex : var = pCnf->pVarNums[Abc_ObjId(pObj)]
- Variable to literal conversion :
  - lit = toLitCond(var , 0) → negative literal
- Make assumptions :
  - sat_solver_solve(pSat , int * , int * , …)
  - ex :

```
int  pLit[2];
pLit[0] = toLitCond( var0 , 0 );
pLit[1] = toLitCond( var1 , 1 );
sat_solver_solve ( pSat , pLit , pLit+2 , … )
```

Conclusions

- Get familiar with ctag / cscope , grep first
- Try to plug in your own package
- Important files :
  - src/bace/abc/abc.h : most of definitions of the data structure and basic operations
  - src/base/abci/abc.c : most of the commands
- Good practices :
  - find API first
  - use defined functions to access data members
- Have fun!