How to program in ABC

Presenter: Nian-Ze Lee

Instructor: Jie-Hong Roland Jiang

ALCom Lab

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

National Taiwan University

Before we start …

- Useful LINUX commands:
  - ctags / cscope
  - grep
  - man

Outline

- Add a command in ABC
- Basic data structure in ABC
- PA0

Add a command in ABC

- Step.1: create a directory “lsv" in the src directory
- Step.2: create a lsvCmd.cpp file as the command file for this package
- Step.3: Lsv_Init() / Lsv_End()
- Step.4: Abc_FrameInit() / Abc_FrameEnd()
- Step.5: Modify Makefile

Overview of ABC data structure

- Abc_Frame_t (src/base/main/mainInt.h)
  - Command table
  - Current network
- Abc_Ntk_t (src/base/abc/abc.h)
  - Objects (PI, PO, gates, etc)
  - Functionality managers (pManFunc)
- Abc_Obj_t (src/base/abc/abc.h)
  - 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
```

PA0

- Get the current network
- Convert to a structurally hashed AIG
- Iterate every node in an AIG
  - Id
  - Ids of its fanins