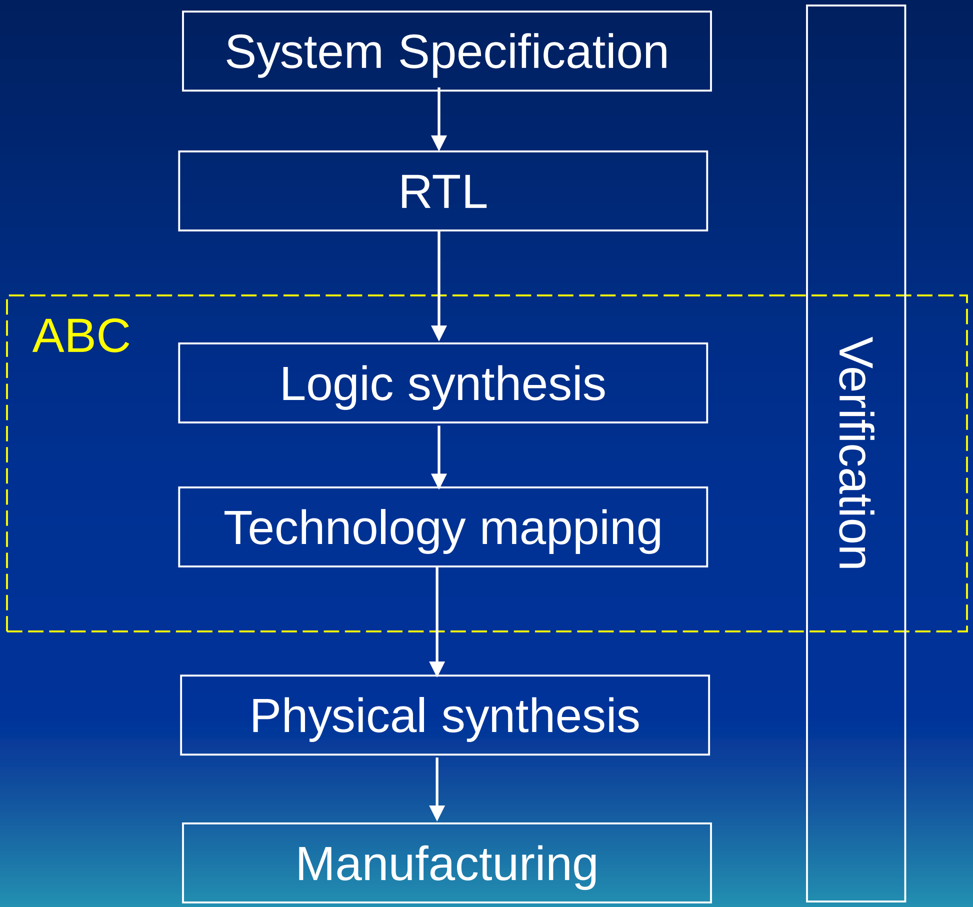
Going Places with ABC

Berkeley Logic Synthesis and Verification Group

Overview

- Introduction
- Logic networks
- And-Inverter Graphs (AIGs)
- Other aspect

Territory


System Specification
RTL
ABC
Logic synthesis
Technology mapping
Physical synthesis
Manufacturing
Verification

Destinations

- Combinational synthesis
  - traditional (SIS)
  - AIG-based
  - tech mapping
  - equivalence checking

- Sequential synthesis
  - traditional (retiming)
  - AIG-based
  - tech mapping
  - verification
    - bounded
    - unbounded
    - based on synthesis

Means of Transportation

- Netlist
  - contains nets along with nodes, latches, and PI/PO terminals
  - currently only used for I/O of networks to/from files
- Logic network
  - traditional logic network as in SIS; does not have nets
  - nodes have SOP/BDD representation of local functions
- AIG
  - innovative network representation
  - unifies synthesis, mapping, and equivalence checking
- Sequential AIG
  - a generalization of AIGs for sequential networks

Types of Fuel

- SOPs
  - Sum-Of-Products (two-level AND-OR representation) traditionally used to store node functions
  - convenient for factoring but has a tendency to grow large
- BDDs
  - graph-based representation, canonical for a fixed variable ordering
  - convenient for some applications but has a tendency to grow large
- AND2s
  - networks of two-input ANDs and inverters
  - scalable, non-canonical representation
- Gates
  - primitives from the gate library assigned to the nodes by tech mapping

Allowed Combinations

- Networks are composed of objects
  - several network types are supported
- Nodes are objects having logic function
  - Several node functionality types are possible

Network type	Node functionality type			
Netlist	SOP			Gates
Logic network	SOP	BDD		Gates
AIG			AND2	
Seq AIG			AND2	

Object Types

- Net
  - an object used in the netlist to denote connection among nodes
- Node
  - an object having a local logic function (e.g. ab+cd or AND2)
- Latch
  - a technology-independent D-flip-flops with an initial state
  - all latches in a network should belong to the same clock domain
- PIs/POs
  - named network terminals w/o local logic function

Object Data Structure

```
struct Abc_Obj_t_ // 12 words
{
    Abc_Ntk_t *      pNtk;          // host network
    int              Id;            // object ID

    unsigned         Type    :  3;  // object type
    unsigned         fMarkA  :  1;  // multipurpose mark
    unsigned         fMarkB  :  1;  // multipurpose mark
    unsigned         fMarkC  :  1;  // multipurpose mark
    unsigned         fPhase  :  1;  // flag to mark the phase of a node (AIG)
    unsigned         fExor   :  1;  // marks a node that is a root of EXOR (AIG)
    unsigned         fCompl0 :  1;  // complemented attribute of the first fanin (AIG)
    unsigned         fCompl1 :  1;  // complemented attribute of the second fanin (AIG)
    unsigned         TravId  : 10;  // traversal ID
    unsigned         Level   : 12;  // level of the node

    Vec_Int_t        vFanins;       // array of fanins
    Vec_Int_t        vFanouts;      // array of fanouts

    void *           pData;         // network specific data (SOP/BDD/gate,etc)
    Abc_Obj_t *      pNext;         // next pointer in the hash table (AIG)
    Abc_Obj_t *      pCopy;         // copy of this object
};
```

Overview

- Introduction
- Logic networks
- And-Inverter Graphs (AIGs)
- Other aspect

Logic Network

- Similar to SIS network
  - contains nodes, latches, PI/PO terminals
  - adding, duplicating, removing nodes are similar
  - nodes can have a logic function or a gate assigned
- Differences
  - after construction, finalizing steps are required
  - internal node names are currently not stored
  - node functionality can be a BDD

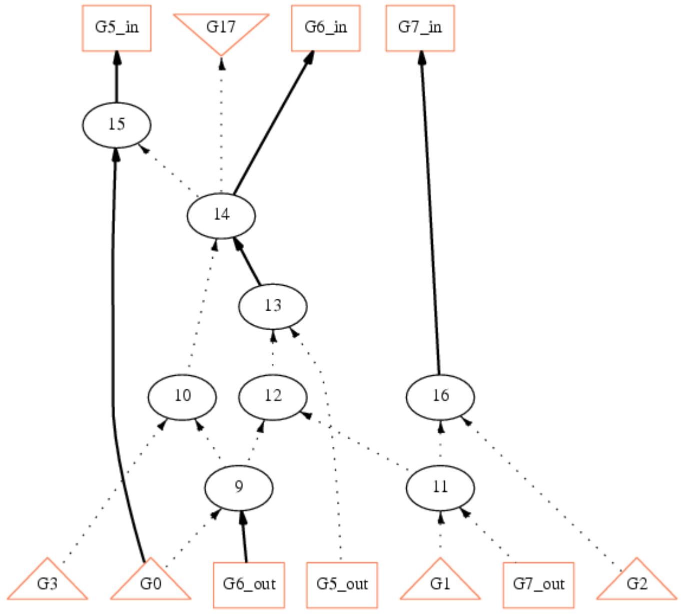
Constructing Logic Network

- Typically a new network is created from an old network
  - Abc_NtkStartFrom // copies PIs, POs, latches, etc
- Objects can be added in any order
  - Abc_NtkDupObj // duplicates an object
  - Abc_NtkCreateNode // creates a new node
- Object should be connected
  - Abc_ObjAddFanin // adds fanin to an object
- In the end, one or more finalizing functions may be called
  - Abc_NtkFinalize // connects POs to nodes (unless done manually)
  - Abc_NtkDupCioNamesTable // should be called always
  - Abc_ManTimeDup // duplicates timing info if present
  - Abc_NtkLogicMakeSimpleCos // often needed (e.g. converting from AIG)
  - Abc_NtkMinimumBase // removes duplicated or vacuous fanins if present
  - Abc_NtkReassignIds // restores topological ordering of nodes (AIGs only)

Overview

- Introduction
- Logic networks
- And-Inverter Graphs (AIGs)
- Other aspect

And-Inverter Graph



Requirements for AIGs in ABC

- AIG is always stored in the structurally hashed form:
  - two AND2s with the same fanins are merged
  - the constants are propagated
  - there is no single-input nodes (inverters/buffers)

- Additionally, the following requirements are satisfied:
  - there are no dangling nodes (nodes without fanouts are deleted)
  - the level of each AND2 reflects the levels of its fanins
  - the EXOR-status of each AND2 is up-to-date
  - the nodes are stored in the topological order
  - the constant 1 node has always number 0 in the object list

Manipulation of AIGs

- AIGs are uniform, compact, and versatile
- Computation based on AIG is fast and scalable
- Deriving AIGs from logic networks is easy (strash)
  - simple AIG conversion rules are used for gates
  - algebraic factoring is used to convert large logic nodes
- AIGs are the primary network representation in ABC
  - required by tech-mappers and most of synthesis commands (balance, collapse, renode, rewrite, refactor, retime)
- Extensively used in sequential synthesis
- Manipulation is different compared to logic networks
  - e.g. cannot duplicate or collapse nodes in an AIG

Operations Performed on AIG

- Building new AND nodes
  - Abc_AigAnd
- Computing elementary Boolean functions
  - Abc_AigOr, Abc_AigXor, etc
- Replacing one node with another
  - Abc_AigReplace
- Propagating constants (computing cofactors)
  - Abc_AigReplace

- Structural hashing and other requirements are automatically enforced by the AIG manager

Overview

- Introduction
- Logic networks
- And-Inverter Graphs (AIGs)
- Other aspect
  - Mapping information
  - Timing information
  - BDD representation
  - SAT solvers
  - FRAIG package
  - Snapshots
  - Sequential AIGs
  - Visualizations

Using Mapping Information

- Pointer to the library
  - attached to pNtk->pManFunc of the network
- Pointers to gates
  - attached to pNode->pData of all internal nodes
- What to do with the mapped network?
  - print mapping statistics (print_stats)
  - print gates used in the mapping (print_gates –l)
  - print delay profile (print_level)
  - sweep equivalent nodes without unmapping (fraig_sweep)
  - write into a BLIF file (write_blif)
  - replace gates by SOPs at all nodes (unmap)
  - continue synthesis (network is unmapped automatically)

Using Timing Information

- Timing info is stored in timing manager
- Timing info of the network
  - accepts arrival times of the PIs
  - Abc_NtkDelayTrace (similar to SIS)
- Timing info of the nodes
  - Abc_NodeReadArrival, etc

```
struct Abc_Time_t_
{
    float        Rise;
    float        Fall;
    float        Worst;
};
```

- Future work
  - support of required times
  - procedures to update timing incrementally (useful for resynthesis)

Using BDD Representation

- CUDD by Fabio Somenzi is used for all BDD manipulation
- Pointer to the BDD manager is in pNtk->pManFunc
- Pointers to the local functions are in pNode->pData
- Global functions can be computed by Abc_NtkGlobalBdds
  - uses dynamic variable reordering
  - handles the case when BDDs blow up
- When constructing a new logic network, it is often convenient to use BDDs to represent local functions
  - in the end, it is possible to convert to a logic network with SOPs by calling Abc_NtkBddToSop (or vice versa, by calling Abc_NtkSopToBdd)

- There is no support for don't-cares in the current version

Using SAT Solvers

- Two networks can be transformed into a miter
  - if the networks are sequential, the miter is a product machine, which can be unrolled for bounded equivalence checking (miter)
- What to do with a combinational miter?
  - print statistics (print_stats)
  - convert into CNF for calling an external SAT solver (cnf)
    - uses smart AIG-to-CNF conversion (due to Miroslav Velev)
  - solve using brute-force SAT (sat)
    - internally calls MiniSat-1.14 (by Niklas Eén, Niklas Sörensson)
  - solve by merging equivalent nodes and applying SAT to the resulting miter (fraig –p)
    - internally calls our version of MiniSat-1.12

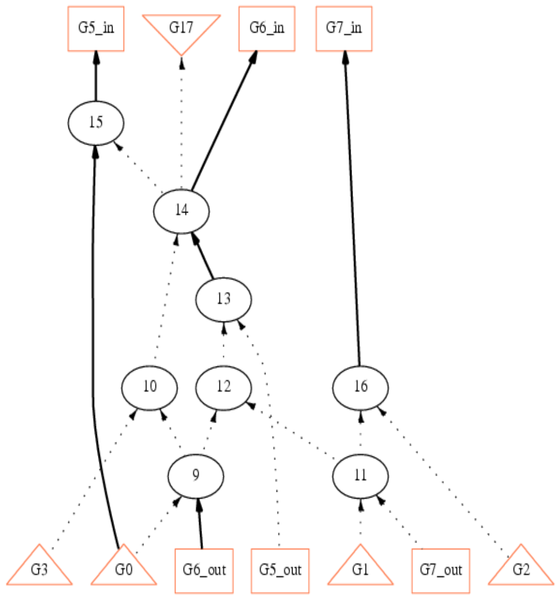
Using FRAIG Package

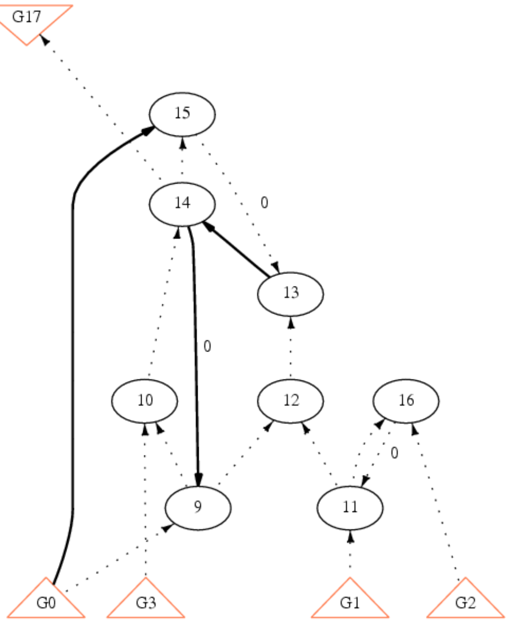
- Transforms AIG into a functionally reduced AIG (FRAIG)
- Lossless synthesis
  - several snapshots are converted into a network with choices (fraig_store, fraig_restore)
- Sweeping logic networks
  - detects and merges functionally equivalent nodes (fraig_sweep)
- Equivalence checking
  - simplifies and proves the miter to be 0, or finds a bug (fraig –p)

Using Snapshots

- Motivation
  - multiple snapshot reduce structural bias in tech mapping
- Creation
  - the current network (both logic network and AIG) can be recorded as a snapshot (fraig_store)
- Storage
  - all snapshots are stored in an internal AIG (different from the current network) while ABC is running
- Use
  - can be restored into the current network with choices (fraig_restore)
- File I/O
  - choice networks can be written into BLIF files (in which OR gate denote equivalence classes of nodes)
  - reading choice networks efficiently can be done using fraig_trust

Using Sequential AIGs





Using Visualizations

- Numerous printout commands are available
  - print_exdc, print_factor, print_fanio, print_gates, print_io, print_latch, print_level, print_sharing, print_stats, etc

- The following visualizations can be used
  - local functions can shown as K-maps (print_kmap) (works well for up to 6 variables)
  - local functions can be shown as BDDs (show_bdd)
  - structure of logic networks (with mapping) (show_ntk)
  - visualization of (sequential) AIGs (show_aig)

ABC Compared with Other Tools

- Industrial
  - well documented, fewer bugs
  - too specialized, no source code, often costly
- SIS
  - traditionally very popular
  - some data structures / algorithms are outdated, no sequential synthesis
- VIS
  - features Verilog input, implementation of BDD-based verification
  - not meant for binary logic synthesis
- MVSIS
  - allows for multi-valued logic synthesis and finite automata manipulation
  - complicated for experimental programming, not for binary synthesis

Conclusion

- Reviewed basic concepts of ABC
- Outlined basic programming principles

Updated programmer’s manual is online:

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