FBDD Logic Synthesis System User Manual (1.1)

Toronto Synthesis Group

Introduction   FBDD is a Binary Decision Diagram (BDD) [1] based combinational logic optimization system targeting area and runtime. It takes a gate level circuit described in Berkeley Logic Interchange Format (BLIF) and produces an area optimized circuit in BLIF and structural Verilog formats.

FBDD is inspired by the BDD-based Logic Synthesis system (BDS) developed at University of Massachusetts [2, 3]. In contrast to traditional logic synthesis systems exemplified by SIS [4], which use cube set to represent logic functions of the gates, and literal count as the area metric, BDS uses BDD to represent logic functions, and the corresponding BDD node count as the area metric. A comprehensive set of decomposition methods exploiting the structural properties of BDDs have been developed in BDS, leading to good results in area for some benchmarks, and better runtime result in general.

FBDD builds a similar flow as BDS and implements its core set of decomposition algorithms. In addition, FBDD attempts to produce much better runtime and more competitive area based on two ideas. First, FBDD attempts to exploit the inherent regularity contained in logic networks to dramatically improve runtime by amortizing the cost of logic transformations over equivalent classes of subnetworks. Described in [5], this strategy is referred to as logic folding and is applied in almost all logic transformations in FBDD. Second, FBDD attempts to perform sharing extraction in the same spirit of SIS by explicitly enumerating divisors in each gate and selectively extracting those that profit area. Our algorithm, described in [6], performs multiple variable disjunctive extraction and operates directly on BDDs. A comprehensive description of the logic synthesis system can be found in [7, 8].

Installation and Compilation

Software   Before buidling FBDD, you need to install the CUDD package by Fabio Somenzi [9].
Requirements   FBDD performs only technology independent logic optimizations and needs an external technology mapper to complete the logic synthesis flow. For standard cell technology, one can use the Berkeley SIS mapper. For FPGA, one can use the UCLA rasp_syn mapper. Note that although the latter is not required to build and run FBDD, the path to the `rasp_syn` executable needs to be pointed to by the PATH environment variable in order to run the provided scripts and automatically produce benchmark results.

Building	FBDD software uses the standard GNU build system based on the Autotool set (autoconf/automake) as well as the GCC compiler suite.

To build FBDD in release mode, type the following under a Unix (e.g. solaris) or Unix-like environment (e.g., Linux or Cygwin for Windows):

```
>aclocal
>autoheader
>autoconf
>automake
>./configure --with-cudd=/your/cudd/path --with-prefix=/your/install/path
>gmake
```

To build FBDD in debug mode:

```
>aclocal
>autoheader
>autoconf
>automake
>./configure --enable-debug --with-cudd=/your/cudd/path \
--with-prefix=/your/install/path
>gmake
```

Installation   To install FBDD:

```
>gmake install
```

This should install the executable file `fbdd` on the path specified by the `with-prefix` argument during configuration.

Usage   FBDD performs a series of logic transformations on a logic network in an effort to improve its quality. It first performs the "simplify" operation to reduce the logic complexity of each gate. Since the gate logic functions are represented by BDDs, the simplification can be achieved by BDD variable reordering to find a BDD representation with less node count. It then performs the "sweep" operation to clean up the network by propagating constants, merging repeating support variables and removing dangling gates. This is followed by the "eliminate" operation to selectively collapsing some gates into their fanouts in order to remove redundancy between gates, after which another "sweep" operation is performed. Next, the "sharing extraction" operation and the "decomposition" operation are applied. The former identifies common logic expressions among and selectively extract them as new gates. The latter decomposes a large gate into smaller gates. The two operations are interleaved: extractions are performed first to find as much logic sharing as possible; when no more extractors can be found, decomposition is applied, which may lead to more extractors to be found. This process is repeated until the circuit is fully decomposed, after which another "sweep" pass is applied.

FBDD can be invoked as follows, where `ckt.blif` corresponds to the input logic network.

```
>fbdd [-options] ckt.blif
```

Command Line Options   The default synthesis flow can be altered using the following options.

- -ne skip elimination pass;

- -ntwe skip sharing extraction pass;

- -nd skip decomposition pass;

These options can be used to understand the contributions on synthesis quality for different logic transformations.

The following options configure the sharing extraction algorithm.

- `-maxextsize size`: find extractors up to size 'size' (default 2);

- `-tweexact`: perform exact sharing extraction;

- `-twefast`: perform fast sharing extraction (default).

FBDD performs sharing extraction by matching disjunctive extractors from among gates in the network. Disjunctive extractors are functions with decompositions of the form $F(X) = R(E(X_r), X_e))$ such that X_r and X_e do not share supports. FBDD puts a limit on the size of extractors considered. The maximum extractor size is specified using the `-maxextsize` option and can be set from size 2 to 5. Be warned however, that the runtime grows exponentially with the extractor size. There is also a choice between two sharing extraction algorithms. The exact algorithm `-tweexact` finds all extractors up to the size specified. The fast algorithm -twefast only applies when -maxextsize is 2. It sacrifices finding ALL two variable extractors in exchange for, on average, a 3X runtime improvement. For best area-runtime trade-off, We recommend running FBDD with default options of `-maxextsize=2` and `-twefast`. Our experimental results show that the area penalty for using default options is negligible. The extra options were left in to demonstrate this finding.

The following options are added in FBDD 1.1 so that FBDD can operate as a translator, which converts an input BLIF file into its equivalent Verilog file acceptable by commercial tools, to enable comparative studies. Since FBDD only performs optimization on combinational circuit, the translator can optionally convert a sequential circuit into a combinational circuit, by adding a primary input and output for each latch in the original circuit.

- -wv write pre-synthesis verilog;

- -rl replace latches and flip flops with primary inputs and outputs.

Input File   FBDD takes a logic network description in Berkeley Logic Interchange Format (BLIF). BLIF is used widely as an academic standard.

FBDD supports the subset of BLIF that can specify a technology independent combinational or sequential logic network. All standard benchmarks are specified within this subset.

The supported commands include:

- .model
- .inputs
- .outputs
- .clock
- .names
- .latch
- .end

The unsupported commands include:

- .exdc
- .subckt
- .gate
- .mlatch
- .search
- kiss FSM descriptions
- clock constraints
- delay constraints

FBDD typically supports BLIF commands written out by SIS. If your circuit contains unsupported BLIF commands, you may be able to rewrite the circuit to work with FBDD by simply reading and then writing the circuit using SIS. Do this by typing:

```
>sis -sxc "read_blif ckt.blif; write_blif ckt.blif;"
```

Output Files   FBDD outputs an optimized logic network either in BLIF format or a restricted form of Verilog file. The latter can be used to interface with commercial CAD tools.

Benchmark Results   The MCNC [10] and ITC [11] benchmarks are included in this distribution. Also included are scripts to automate optimization with FBDD, apply FPGA and standard cell technology mapping, and tabulate the results in a report file. To run the benchmark scripts, type

```
>cd benchmark
>gmake result
```

This will apply FBDD to the MCNC and ITC benchmarks and summarize their results in fbdd.mcnc.csv and fbdd.itc.csv respectively. The .csv files are space delimited tables which can be imported into Excel or StarOffice spreadsheet for viewing and further processing.

Comparative Study

The FBDD distribution includes scripts that automatically run other academic logic synthesis tools, as well as the commercial Xilinx ISE and Altera Quartus tools for comparison purposes. To use these scripts, the selected tools are required to be pointed to by the PATH environment variable.

Academic Tools

To compare with academic tools, one can modify the COMPARE_MCNC or COMPARE_ITC variables in benchmark/Makefile.am to include different scripts for running academic tools, and for MCNC and ITC benchmarks respectively. The seperate list is necessary since not all tools can handle the sequential ITC benchmark circuits. The supplied scripts include

- sis: Berkeley SIS using script.rugged;

- sisa: Berkeley SIS using script.albegraic;

- bds: Massachusetts BDS using the default options;

- bdspga: Massachusetts BDS-PGA using the default options.

The default setting is:

```
COMPARE_MCNC = sis bds bdspga
COMPARE_ITC  = sisa
```

Type

```
>cd benchmark
>gmake academic-mcnc
>gmake academic-itc
```

to create a .csv file for each tool with respect to a benchmark. For example, sis.mcnc.csv and sis.itc.csv are results for running SIS for MCNC and ITC benchmarks respectively. *It is highly recommended tp run the comparison under a Linux environment in order to collect the most comprehensive data.*

Commercial Tools

To compare with commercial tools, scripts are provided to automatically run Xilinx ISE and Altera Quartus. These scripts first convert BLIF files to presynthesis Verilog by running FBDD as the converter. They will then generate files necessary to drive the commercial tools. The LUT count and the runtime information are then automatically extracted from the native synthesis report.

To ensure comparison is done as fair as feasible, the translated Verilog files are pure combinational circuits to disable sequential optimization, which FBDD does not yet perform. The default options for these tools are used, except that the optimization goal is set to area. The default targeted devices are CycloneII (EP2C70F896I8) for Quartus and Spartan3 (xc3s1500-4-fg676) for ISE. For these devices, it has been confirmed that no FPGA achitectural features, such as dedicated multipliers, LUTs configured in arithmetic mode are used for the MCNC and ITC benchmarks. The exception is with Xilinx FXMUXes, which when included, inflates the LUT count by 5%.

Type

```
>cd benchmark
>gmake xilinx
>gmake altera
```

to create a .csv file for each tool with respect to a benchmark.

Release Notes (Version 1.1)

New options

Command line options -wv and -rl are added for pre-synthesis Verilog translation.

New scripts   Tcl scritps are added to carry out comparative studies.

Bug fixes   Some BLIF parser bugs in 1.0 release are fixed.

Credits   FBDD is conceived, developed and released by the Synthesis Research Group, the Department of Electrical and Computer Engineering, University of Toronto. The primary contributor of FBDD release 1.0 is Dennis Wu, who implemented the core algorithms and completed a master thesis [12] on the subject.

Web   http://www.eecg.toronto.edu/~jzhu/fbdd.html

References

[1]   R. E. Bryant, "Graph-based algorithms for boolean function manipulation," in *IEEE Transactions on Computer*, 1986, pp. 677–691.

[2]   C. Yang, M. Ciesielski, and V. Singhal, "BDS: A BDD-based logic optimization system," in *Proceedings of the Design Automation Conference*, 2000, pp. 92–97.

[3]   N. Vemuri, P. Kalla, and R. Tessier, "BDD-based logic synthesis for LUT-based FPGAs," *ACM Transactions on Design Automation of Electronics Systems*, vol. 7, no. 4, pp. 501–525, October 2002.

[4]   E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanaha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuits synthesis," Tech. Rep. UCB/ERL M92/41, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720, 1992.

[5]   D. Wu and J. Zhu, "Folded logic decomposition," in *International Workshop on Logic and Synthesis*, Laguna Beach, California, June 2003.

[6]   D. Wu and J. Zhu, "BDD-based two-variable sharing extraction," in *Proceeding of Asia and South Pacific Design Automation Conference*, Shanghai, China, January 2005.

[7]   Dennis Wu and Jianwen Zhu, "FBDD: A folded logic synthesis system," in *International Conference on ASIC (ASICON)*, Shanghai, China, Oct. 2005.

[8]   D. Wu and J. Zhu, "FBDD: A folded logic synthesis system," Tech. Rep. TR-08-01-04, Department of Electrical and Computer Engineering, University of Toronto, August 2004.

[9]   F. Somenzi, "CUDD: CU decision diagram package release 2.4.1," Tech. Rep., Department of Electrical and Computer Engineering, University of Colorado at Boulder, 2005.

[10] Saeyang Yang, "Logic synthesis and optimization benchmarks user guide version 3.0," Tech. Rep., Microelectronics Center of North Carolina, P. O. Box 12889, Research Triangle Park, NC 27709, 1991.

[11] F. Corno, M. Sonza Reorda, and G. Squillero, "RT-level ITC 99 benchmarks and first atpg results," in *IEEE Design & Test of Computers*, 2000, pp. 44–53.

[12] Dennis Wu, "Towards scalable BDD-based logic synthesis," M.S. thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Jan. 2005.